On the Space Complexity of Storing Small Sets in the Bitprobe Model

Thesis submitted in partial fulfilment of the requirements

for the award of the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Mirza Galib Anwarul Husain Baig

Under the supervision of

Prof. Deepanjan Kesh

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Guwahati - 781039 Assam India

January, 2020

To my family

Acknowledgements

Above all, I would like to thank my supervisor, Prof. Deepanjan Kesh, for being a great supervisor. This thesis would not have been possible without him helping me to find out the right problem and approach to work on. Further, I would also like to thank him for the excellent working environment, which he provided me throughout my stay in IIT Guwahati.

I would like to thank Chirag Sodani for several helpful discussion during my collaboration with him.

I am indebted to Prof. Jaikumar Radhakrishnan, Prof. Venkatesh Raman, Prof. Srinivasa Rao Satti, and Prof. Saswata Sannigrahi who responded promptly to my email(s) and shared with me materials related to my work. I thank Prof. Shaik Rafi Ahamed for some nice bits of advice.

Abhishek deserves a special thanks for being a great friend. Discussions with him, V.A. Amarnath and Sukanta Dey, whom I would like to thank as well, was really helpful during my stay in Guwahati.

Above all, I thank my parents, brothers, and sister Naima for the love and support they have given throughout my life.

January 5, 2019 Mirza Galib Anwarul Husain Baig

Declaration

I certify that

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisor(s).
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor(s) are not in a position to check for any possible instance of plagiarism within this submitted work.

January 5, 2019 Mirza Galib Anwarul Husain Baig

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

Dr. Deepanjan Kesh
Assistant Professor
Email : deepkesh@iitg.ac.in
Phone : +91-361-258-2377

Certificate

This is to certify that this thesis entitled "**On the Space Complexity of Storing Small Sets in the Bitprobe Model**" submitted by **Mirza Galib Anwarul Husain Baig**, in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy, to the Indian Institute of Technology Guwahati, Assam, India, is a record of the bonafide research work carried out by him under my guidance and supervision at the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Assam, India. To the best of my knowledge, no part of the work reported in this thesis has been presented for the award of any degree at any other institution.

Date: January 5, 2019
Place: Guwahati

Prof. Deepanjan Kesh
(Thesis Supervisors)

Abstract

In this thesis, we study static membership problem which involves the study and construction of such data structures that can store an arbitrary subset $\mathcal{S}$ of size at most n from the universe $\mathcal{U}$ of size m such that membership queries of the form "Is x in $\mathcal{S}$?" can be answered correctly and efficiently. A special category of the static membership problem is the bitprobe model in which we evaluate our solutions w.r.t. the following resources – the size of the data structure, s, required to store the subset $\mathcal{S}$, and the number of bits, t, of the data structure read to answer membership queries. It is the second of these resources that lends the name to this model.

In this model, the design of data structures and query algorithms are known as schemes. For a given universe $\mathcal{U}$ and a subset $\mathcal{S}$, the algorithm to set the bits of our data structure to store the subset is called the storage scheme, whereas the algorithm to answer membership queries is called the query scheme. Schemes are divided into two categories depending on the nature of our query scheme. Upon a membership query for an element, if the decision to probe a particular bit depends upon the answers received in the previous bitprobes of this query, then such schemes are known as adaptive schemes. If the locations of the bitprobes are fixed for a given element of $\mathcal{U}$, then such schemes are called *non-adaptive schemes*.

Our work in this thesis gives improved bounds for various values of n, m, and t. Bhurman *et. al.*[1] gave non explicit adaptive $(2, m, \mathcal{O}(m^{3/4}), 2)$-scheme. This was further improved by Radhakrishnan *et. al.* [2] to explicit adaptive $(2, m, \mathcal{O}(m^{2/3}), 2)$-scheme. We [3, 4] improve the $(2, m, \mathcal{O}(m^{2/3}), 2)$-scheme to $(3, m, \mathcal{O}(m^{2/3}), 2)$-scheme. Later, Nicholson[5] came up with a $(3, m, \mathcal{O}(m^{2/3}), 2)$-scheme, and posed a problem that if similar scheme exists for $n = 4$. We [6] answer their problem with a $(4, m, \mathcal{O}(m^{5/6}), 2)$-scheme, and further improve the space to $\mathcal{O}(m^{4/5})$. Furthermore, we [7] come up with a $(5, m, \mathcal{O}(m^{10/11}), 2)$-scheme. And, further we [8] improve the above scheme to $\mathcal{O}(m^{5/6})$. For subsets of size five ($n = 5$), the best known lower bound was due to Alon and Feige [9] which is $\Omega(m^{1/2})$. The $\Omega(m^{2/3})$ lower bound for $n = 3$ also puts an improved bound

for the $n = 5$ case. We [8] improve it to $\Omega(m^{3/4})$. Apart from this, we have given some adaptive and non-adaptive schemes, which improves slightly the existing results in the literature.

Contents

List of Figures

List of Tables

"It is difficult and often impossible to judge the value of a problem correctly in advance; for the final award depends upon the gain which science obtains from the problem. Nevertheless we can ask whether there are general criteria which mark a good mathematical problem. An old French mathematician said: "A mathematical theory is not to be considered complete until you have made it so clear that you can explain it to the first man whom you meet on the street". This clearness and ease of comprehension, here insisted on for a mathematical theory, I should still more demand for a mathematical problem if it is to be perfect; for what is clear and easily comprehended attracts, the complicated repels us."

–David Hilbert, Address to the International Congress of Mathematicians, 1900

1

Introduction

In computer science, data structure refers to a particular way of organizing and storing data, usually in computer memory, so that it can be used efficiently. In the last two decades, we have seen a massive increase in online data. For example, there are billions of web pages on the internet. Further, we know that companies like *Google* or *DuckDuckGo* handles around tens of thousands of queries on those web pages. In this case, the obvious implementation of data structure where each query passes through all the data element does not work well. So, we need data structures where every query to the data set does not pass through all the data elements stored, and still uses minimal possible space. To sum it up, we need time and space-efficient data structures to handle the massively growing online data.

In this thesis, we study the space and time efficiency of few data structure problems. So, we would aim to organize and store data in computer memory in such a way that query operations on those data sets are optimized. Further, we would like to do this using a minimum possible space. In the next section, we will first talk about the model of computation, and then we will move to the problem statement and few definitions. In the last section, we will talk about the outline of the thesis and our contribution.

1.1 Model of Computation

The model of computation used in this thesis is named as bitprobe model. Minsky and Papert first introduced this model in their 1969 book "Perceptron" [10]. This is a simple model where memory is organized as cells, each capable of storing a single bit. Further, memory is random access, and each query to the memory returns a single bit stored in the query location. The complexity of data structure problems in this model is measured in terms of the size of the data structure denoted by s, and the number of bits of the data structure accessed denoted by t . It is the later of the two properties which lend its name bitprobe model.

1.2 The Problem Statement

We study the static membership problem. In this problem– given a universe $\mathcal{U}$ containing m elements, we want to store an arbitrary subset $\mathcal{S}$ of $\mathcal{U}$ whose size is at most n, such that we can answer membership queries of the form "Is x in $\mathcal{S}$?" efficiently. This problem is static in a sense that we do not support insertion or deletion operation.

Solutions to the above-mentioned problems in this model are termed as schemes. Each scheme consists of two parts, one is the storage scheme, and the other is query scheme. Storage scheme maps an arbitrary subset $\mathcal{S}$ of cardinality n from a universe $\mathcal{U}$ of size m given to be stored to the s bits of the data structure. Query scheme maps every element belonging to the universe m to the t locations of the data structure, and it decides the membership of the query element by reading those t locations. Formally, storage scheme is a function of the form $\phi : \binom{[m]}{\leq n} \rightarrow \{0,1\}^s$, that takes a set of size at most n and returns its s-bit representation. On the other hand, query scheme associates with each element x the t probe locations $(i_1(x), ..., i_t(x)) \in [s]^t$ and a function $f(x) : \{0,1\}^t \rightarrow 0, 1$. The query schemes evaluate to one if and only if the query element is part of the set given to be stored. The storage and query scheme together gives a (n, m, s, t)-scheme which stores a subset $\mathcal{S}$ of size at most n from a universe $\mathcal{U}$ of size m and uses s bits in such a way that membership query can be answered in t probes.

Let us discuss a few trivial solutions. Given a n elements subset $\mathcal{S}$ of universe $\mathcal{U}$ of size m, we store all the elements in sorted order in memory. Storing all the elements will take at most $n \lg m$ bits. Further, given a query element $x \in \mathcal{U}$, we can do a binary search

on the stored elements and find out whether a query element is part of the set given to be stored or not. Number of queries taken in this solution will be at most $(\lg n \lg m)$ bits. If the size of universe $\mathcal{U}$ is very large then this solution takes a very small amount of space, but at the cost of a large number of queries.

Let us now consider another trivial solution to the problem mentioned above. In this solution, we store a table of size m in a memory, where each cell in the table is capable of storing a single bit. Our table stores a single bit for each element of the universe. If the element is part of the subset $\mathcal{S}$ given to be stored, then we store one for that element. On the other hand, if the element is not part of the subset $\mathcal{S}$ given to be stored, then we store zero for that element in the table. Now, given a query element $x \in \mathcal{U}$, we look into the table and say that query element is part of the subset given to be stored if and only if the stored bit returned is one. Though the number of queries required in this solution is one, the amount of our space, i.e. m is huge. In this thesis, we will study the trade-offs between the amount of space taken, and the number of bit queries made to answer the membership problem.

1.3 Definitions

In this section, we will give a few definitions associated with the schemes in the bitprobe model.

1.3.1 Adaptive Scheme

If the location where the current bitprobe is going to be depends on the answer obtained from the previous bitprobes, then such schemes are called *adaptive schemes*.

1.3.2 Non-Adaptive Scheme

If the location of the current bitprobe is independent of the answers obtained in the previous bitprobes, then such schemes are called *non-adaptive schemes*.

1.3.3 Deterministic Scheme

A scheme is said to be deterministic if the answer returned by it is always correct.

1.3.4 Randomized Scheme

In this scheme query scheme has access to random bits. Further, this scheme can answer incorrectly with a small probability.

1.3.5 Explicit or Non-Explicit Scheme

Non-explicit schemes use a probabilistic technique to show the existence of storage and query schemes, but it does not tell how to construct it efficiently. Whereas in an explicit scheme, storage and query scheme can be constructed efficiently. A scheme is said to be fully explicit if storage scheme can be constructed in time polynomial in s and location of the probe to be queried can be computed in time polynomial in $\lg m$ and t.

Radhakrishnan *et al.* [2] introduced the notation $(n, m, s, t)_A$ and $(n, m, s, t)_N$ to denote the adaptive and non-adaptive schemes, respectively. Sometimes the space requirement of the two classes of schemes will also be denoted as $s_A(n, m, t)$ and $s_N(n, m, t)$, respectively.

Nicholson *et al.* [11] has surveyed the bitprobe model with discussions about the current state of the art and a selection of open problems.

1.4 Historical Notes

Minsky and Papert in their 1969 book "Perceptron" [10] studied average case bitprobe complexity of membership problem, this problem was further studied in context of retrieval problem by Elias and Flower [12]. Elias and Flower in his paper gave formal model of retrieval problem. This problem was further studied by Yao [13] in the cell probe model. In this model unlike bitprobe model each cell is capable of storing one element, i.e $\lg m$ bits, and a single query to to memory returns $\lg m$ bits. Buhrman, Miltersen and Radhakrishnan revived the study of bitprobe model computation. The static membership problem is a well studied problem over several decades in this model and it has been discussed in [1, 2, 5, 9, 11, 14, 15] and [16].

1.5 Thesis Outline and Contributions

The thesis is organized as follows:

1.5.1 Chapter 1

This chapter introduces the static membership problem in bitprobe model. Further, we give a few definitions and notations in the context of this problem. We end this chapter with the outline and contribution of the thesis.

1.5.2 Chapter 2

In this chapter, we first give an explicit adaptive $(2, m, 2.5m^{2/3}, 2)$ scheme. This gives an alternate scheme for the two probes adaptive scheme storing two elements. This scheme also improves the constant factor of Theorem 2 of Radhakrishnan *et al.* [2]. Further, we look into explicit adaptive $(9, m, 7m^{2/3}, 3)$ scheme. Earlier maximum number of elements that could be handled using three adaptive probe and $\mathcal{O}(m^{2/3})$ space was five due to Radhakrishnan et al. [2]. At the end of this chapter, we generalize this scheme. The generalized result slightly improves Theorem 3 of Radhakrishnan *et al.* [2].

1.5.3 Chapter 3

This chapter starts with few simple explicit non-adaptive upper bounds, and finally uses it to obtain an explicit non-adaptive scheme which uses slightly fewer bitprobes, and slightly more space than the existing state of the art results.

1.5.4 Chapter 4

This chapter first presents an explicit adaptive $(3, m, 3m^{2/3}, 2)$ scheme. Earlier maximum number of elements that could be handled using $\mathcal{O}(m^{2/3})$ space-bound and two adaptive bitrobes was two due to Radhakrishnan et al. [2]. Further, we look into an explicit non-adaptive $(4, m, \mathcal{O}(m^{2/3}), 4)$ scheme, and improve it further to store the subset of size at most five using the same amount of space. The earlier scheme for four non-adaptive bit probe was generalized and non-explicit due to Noga and Alon [9], which uses $\mathcal{O}(m^{3/4})$ amount of space.

1.5.5 Chapter 5

In this chapter, we give an explicit adaptive $(4, m, \mathcal{O}(m^{5/6}), 2)$ scheme. Further, we improve the space to $\mathcal{O}(m^{4/5})$. This answers an open problem posed by Patrick K.

Nicholson [5] which asked if a scheme using the idea of blocks due to Radhakrishnan *et al.* [2] exists that stores four elements and answers membership queries using two bitprobes.

1.5.6 Chapter 6

This chapter gives an explicit adaptive $(5, m, \mathcal{O}(m^{10/11}, 2)$ scheme. Our scheme improves upon the non-explicit by Garg and Radhakrishnan [15] which takes $\mathcal{O}(m^{20/21})$ space. Further, our scheme also improves the explicit scheme by Garg [16] which takes $\mathcal{O}(m^{18/19})$ space.

1.5.7 Chapter 7

In this chapter, we improve the lower bound for two probe scheme storing five elements using the notion of the universe of sets in Kesh [17]. We also believe that the idea can be generalized to give a lower bound for an arbitrary subset of size n. Furthermore, we present an improved scheme for this problem which uses $\mathcal{O}(m^{5/6})$ space.

"...Ring the bells that still can ring

Forget your perfect offering

There is a crack, a crack in everything

Thats how the light gets in..."

–Leonard Cohen, Selected Poems, 1956-1968

2

Explicit Adaptive Bitprobe Scheme

2.1 Introduction

This chapter focuses on the study of set membership problem. In this problem, we are given a subset $\mathcal{S}$ of size at most n from a universe $\mathcal{U}$ of size m, and our goal is to come up with a succinct storage and query scheme which answers the membership queries of the form "Is x in $\mathcal{S}$?" We study this problem in the bitprobe model of computation. In this model, complexity is measured in terms of the amount of space taken by the data structure and the number of bit queries made to answer the membership query.

Solutions to the abovementioned problem in the bitprobe model are termed as schemes, and it is represented by (n, m, s, t), where s and t represents the amount of space taken, and the number of bit queries made to the data structure respectively. Space required by such a scheme is also denoted as $s(n, m, t)$. Schemes are classified as adaptive and non-adaptive depending upon the nature of queries made. If, after the first probe, subsequent probes depend upon the values of the previous probe, then such schemes are called adaptive, otherwise, they are called non-adaptive. Furthermore, schemes are also classified into non-explicit and explicit schemes. A non-explicit scheme guarantees the existence of a scheme for some setting of n, m, s, and t, without showing its construction. Whereas, explicit schemes give a construction of a scheme.

2.1.1 Our Contribution

Most of our schemes in this chapter uses the idea of Radhakrishnan *et.al* [2] to divide universe $\mathcal{U}$ of size m into blocks and superblocks. Our adaptive schemes are based on Theorem 2 of Radhakrishnan *et al.* [2]. Our first adaptive scheme (Theorem 2.1) in this chapter improves the constant factor of Theorem 2 of Radhakrishnan *et al.* [2]. Our second theorem (Theorem 2.2) presents an explicit adaptive scheme that can store at most nine elements , and using $\mathcal{O}(m^{2/3})$ amount of space, i.e $s_A(9, m, 3) \leq \mathcal{O}(m^{2/3})$. Previously, the maximum number of elements that could be handled using three adaptive bitprobe and $\mathcal{O}(m^{2/3})$ amount of space was five due to Radhakrishnan et al. [2]. Furthermore, we generalize this scheme to get $(n, m, (2^t-1)m^{2/3}, 1+\lceil \lg(\lceil \frac{1}{2}\lfloor \frac{n}{2} \rfloor \rceil + 2)\rceil)$-scheme(Theorem 2.3), which slightly improves the theorem 3 of Radhakrishnan et al. [2].

2.2 Revisiting Two Adaptive Bitprobe Scheme

In this section, we look into adaptive schemes with two bitprobes ($t = 2$) and subset size of at most two ($n \leq 2$). For subsets of size two ($n = 2$), Radhakrishnan *et al.* [2] proposed a scheme that takes $3m^{2/3}$ amount of space, and further conjectured that asymptotically it is the minimum amount of space required for any scheme. Though progress has been made to prove the conjecture [2, 14], it as yet remains unproven.

We start this section with a design of data structure which stores an arbitrary subset $\mathcal{S}$ of size at most two from a universe $\mathcal{U}$ of size m, and answers membership queries in two adaptive bitprobes using $2.5m^{2/3}$ bits of space.

2.2.1 The Bitprobe Model

The scheme presented in this section is an adaptive scheme that uses two bitprobes to answer membership queries. We now discuss in detail the bitprobe model in the context of two adaptive bitprobe.

The data structure in this model consists of three tables – T, T_0, and T_1 – arranged as shown in Figure 2.1. Any element e in the universe m has a location in each of these three tables, which are denoted by $T(e), T_0(e)$, and $T_1(e)$.

Any bitprobe scheme has two components – the *storage* scheme, and the *query*

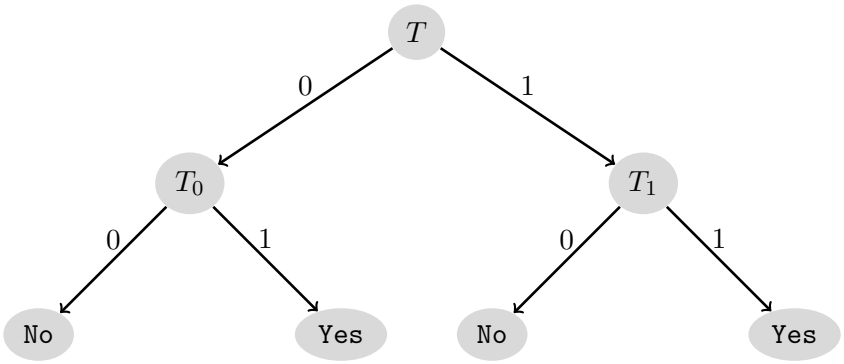


Figure 2.1: The decision tree of an element.

scheme. Given a subset $\mathcal{S}$, the storage scheme sets the bits in the three tables such that the membership queries can be answered correctly. The flow of the query scheme is traditionally captured in a tree structure, called the *decision tree* of the scheme (Figure 2.1). It works as follows. Given a query "Is x in $\mathcal{S}$?", the first bitprobe is made in table T at location $T(x)$. If the bit stored is 0, the second query is made in table T_0, else it is made in table T_1. If the answer received in the second query is 1, then we declare that the element x is a member of $\mathcal{S}$, otherwise we declare that it is not.

2.2.2 Our Data Structure

Similar to the scheme by Radhakrishnan *et al.*[2], our scheme divides the universe $\mathcal{U}$ into blocks of size $m^{1/3}$, and then combines $m^{1/3}$ consecutive blocks into superblocks. So, our scheme will have $m^{2/3}$ blocks, and $m^{1/3}$ superblocks. We number the superblock from 0 to $m^{1/3}-1$. Further, we name the first superblock as S_0, second superblock as S_1, and so on. Superblocks are further divided into two groups, i.e, superblocks numbered even and the superblocks numbered odd. We number the $m^{1/3}$ blocks inside a superblock from 0 to $m^{1/3}-1$. We name the first block inside a superblock as B_0, second block as B_1 and so on. Our scheme consist of three tables T, T_0 and T_1.

Table T contains one bit of space for each of the blocks as shown in Table 2.1 . Thus, it uses $m^{2/3}$ bits of space.

Structure of table T_0 is shown in Table 2.2. Table T_0 consist of $m^{2/3}$ cells, each capable of storing single bit. From storage point of view table T_0 is divided into two parts, first $m^{2/3}/2$ bits are used to map blocks belonging to even superblocks, whereas the next $m^{2/3}/2$ bits are used to map blocks from odd superblocks. Further, combining

$m^{1/3}$ consecutive bits of each part of the table T_0, we get $m^{1/3}/2$ blocks for the even and the odd superblocks. Now, since each superblock has $m^{1/3}$ blocks, and we have only space for $m^{1/3}/2$ blocks of a superblock, we superimpose the blocks such that block 0 (B_0) and block 1 (B_1) maps to the first block, block 1 (B_1) and block 2 (B_2) maps to the second block and so on.

Structure of table T_1 is shown in Table 2.3. Table T_1 consist of $m^{2/3}/2$ bits of space. Combining $m^{1/3}$ consecutive bits, we divide this table into $m^{1/3}/2$ blocks of size $m^{1/3}$. Each block in this table is used to map blocks from two superblocks. Now, since we have $m^{1/3}$ superblocks, and the number of blocks in table T_1 is $m^{1/3}/2$, if each block maps blocks from two superblocks, we can map all the blocks in table T_1. So, the blocks from superblock zero (S_0) and the superblock one (S_1) are mapped at first block, the blocks from superblock two (S_2) and the superblock three (S_3) are mapped at the second block, and so on.

Table 2.1: Mapping of blocks in table T

SE				SO			
$\leftarrow m^{1/3} \rightarrow$				$\leftarrow m^{1/3} \rightarrow$			
		.....				.....	
0	$m^{1/3}-1$		$m^{2/3}/2-1$	0	$m^{1/3}-1$		$m^{2/3}/2-1$
B_0	B_2		$B_{m^{1/3}-2}$	B_0	B_2		$B_{m^{1/3}-2}$
B_1	B_3		$B_{m^{1/3}-1}$	B_1	B_3		$B_{m^{1/3}-1}$

Table 2.2: Mapping of blocks in table T_0

$\leftarrow m^{1/3} \rightarrow$					
S_0	S_2	S_4	S_6	.....	$S_{m^{1/3}-2}$
S_1	S_3	S_5	S_7		$S_{m^{1/3}-1}$
0	$m^1/3-1$	$2m^{1/3}-1$	$3m^{1/3}-1$	$4m^{1/3}-1$	$m^{2/3}/2-1$

Table 2.3: Mapping of blocks in table T_1

2.2.2.1 Query Scheme

Given a query element $e \in [m]$, we first find out the block and the superblock to which this element belongs. First query is made to the table T. If the bit returned by table T is zero, next query is made to the table T_0 otherwise to the table T_1. We say the query element is part of the set given to be stored if and only if the last bit returned is one.

2.2.2.2 Storage Scheme

Our storage scheme is divided into various cases, which takes into account all possible way an arbitrary subset of size at most two can be chosen from $[m]$. Given an arbitrary subset of size at most two, we set the bit of our three tables according to the following:

Case 1. Both elements belong to same superblock.

Case 1.1 Both elements do not belong to consecutive blocks of superblock. In this case, we send both the blocks which contain elements given to be stored into table T_0, and store its characteristic vector there. Further, we send rest of the blocks to table T_1, and store its characteristic vector there.

Case 1.2 If both the elements belong to consecutive blocks of a superblock, then send one of the block to T_0, and the other block to T_1. Further, all the other blocks of that superblock are sent to T_0, and all the blocks of the superblock conflicting with the element sent to T_1 is sent to T_0, and rest of the block is sent to T_1. Since the conflicting superblock would be of opposite type, i.e, one of them will be even and other will be odd, blocks having conflicting bit will not map at the same place.

Case 2 Both elements belong to different superblocks.

Case 2.1 If both elements do not belong to consecutive superblocks, then send both the blocks which contains elements given to be stored to the table T_1, and store its characteristic vectors there . Further, rest of the blocks are sent to T_0, where all the bits are set to zero.

Case 2.2 If both elements belongs to consecutive superblocks, then send both the blocks which contains elements to be stored to table T_0, and store its characteristic vector there. Further, rest all of the blocks are sent to the table T_1, where all the bits are set to zero.

2.2.3 Correctness

Blocks having elements are given separate space either in table T_0 or in table T_1, and we have stored its characteristic vector there. So, the query belonging to these blocks will always be answered correctly. Further, empty blocks, i.e, blocks which do not have any elements given to be stored do not conflict with block having elements given to be stored either in table T_0 or in table T_1, and we have stored zero for all of them. So the query belonging to those blocks will also be answered correctly.

We summaries finding of this section in the following theorem:

Theorem 2.1. *There is an explicit adaptive two bitprobe scheme, which stores an arbitrary subset of size at most two and uses* $2.5m^{2/3}$ *bits of space.*

Let us now choose size of blocks and superblocks more carefully. Let x be the block size, and we combine k consecutive blocks to form superblocks. So the size of table T would be m/x. Size of table T_0 would be kx. Furthermore, size of table T_1 would be $1/2(m/kx) \times x$. Summing up the space taken by all the tables we get the following equation:

$$f(k, x) = \frac{m}{x} + kx + \frac{m}{2k}. \tag{2.1}$$

Substituting $x = 2k$, and further choosing $x = 2^{1/3}m^{1/3}$, we get space taken by the data structure to be $\frac{3m^{2/3}}{2^{1/3}}$. Which is approximately equal to $2.3811m^{2/3}$.

2.3 On the Three Adaptive Bitprobe Scheme

In Section 2.3.3, we present an explicit adaptive scheme that can store at most nine elements using $\mathcal{O}(m^{2/3})$ amount of space, i.e $s_A(9, m, 3) \leq \mathcal{O}(m^{2/3})$. Previously, the maximum number of elements that could be handled using three adaptive bitprobe and $\mathcal{O}(m^{2/3})$ amount of space was five due to Radhakrishnan *et al.* [2]. Furthermore, in

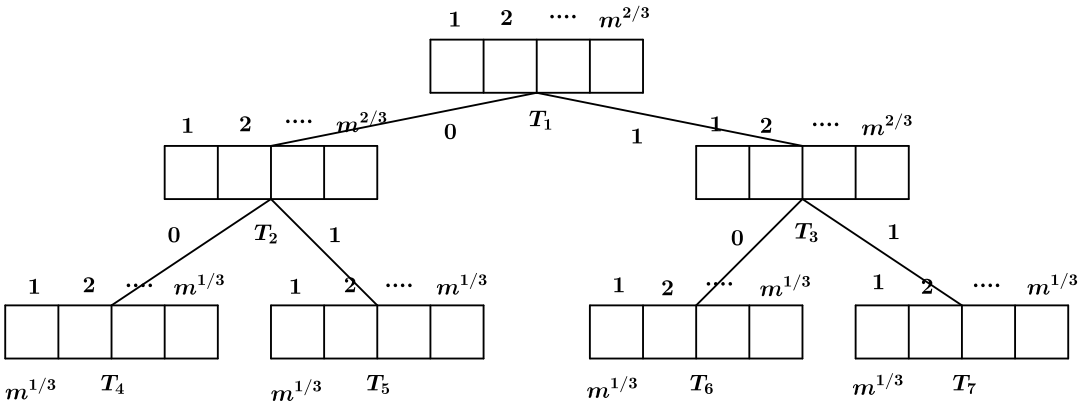


Figure 2.2: Decision tree for the three adaptive bitprobe

Section 2.3.4, we generalize this scheme to get $(n, m, (2^t-1)m^{2/3}, 1+\lceil \lg(\lceil \frac{1}{2}\lfloor \frac{n}{2} \rfloor \rceil + 2)\rceil)$-scheme, which slightly improves the $(n, m, \mathcal{O}(n \cdot m^{2/3}), 1+\lceil \lg(\lceil \lfloor \frac{n}{2} \rfloor \rceil + 2)\rceil)$ of Radhakrishnan *et al.* [2].

2.3.1 Previous Results

Most of the results in the bitprobe model are discussed in the survey paper by Nicholson *et al.* [11]. In this section, we discuss the results in the context of our problem. Radhakrishnan *et al.* [2] came up with an explicit adaptive $(2, m, \mathcal{O}(m^{2/3}), 2)$-scheme. Using this scheme they came up with explicit adaptive (n, m, s, t)-scheme with $s = \mathcal{O}(n \cdot m^{2/3})$ and $t = 1 + \lceil \lg(\lceil \lfloor \frac{n}{2} \rfloor \rceil + 2)\rceil$. They further generalized this scheme to come up with explicit adaptive $(n, m, s, 2 + \log\log n)$-scheme where s is $o(m)$ and n is $\mathcal{O}(m^{1/\lg\lg m})$. For three probes ($t = 3$), Alon and Feige [9] gave a non-explicit $(n, m, s, 3)$-scheme with $s = \mathcal{O}(n^{1/3}m^{2/3})$ and $n = o(m)$. This result has been further improved by Garg and Radharishnan [15]. They came up with a non-explicit adaptive $s(n, m, s, 3)$-scheme with $s = \mathcal{O}(\sqrt{mn \lg \frac{2m}{n}})$.

2.3.2 Our Contribution

In this chapter, we have focused on the design of explicit schemes using blocks and superblocks idea given by Radhakrishnan *et al.* [2]. For $t = 3$ and $s = \mathcal{O}(m^{2/3})$ the maximum number of elements that can be stored using an explicit scheme is five ($n = 5$), which is derived from $(n, m, \mathcal{O}(n \cdot m^{2/3}), 1+\lceil \lg(\lceil \lfloor \frac{n}{2} \rfloor \rceil + 2)\rceil)$-scheme by Radhakrishnan *et al.* [2]. Our first result improves this to the following.

Result 2.1 (Theorem 2.2.)**.** *There is a three probe explicit adaptive scheme which stores an arbitrary subset $\mathcal{S}$ of size at most nine ($n = 9$) from a universe $\mathcal{U}$ of size m, and uses $7 \cdot m^{2/3}$ bits of space.*

Our second result generalizes the aforementioned scheme, and also slightly improves the $(n, m, \mathcal{O}(n \cdot m^{2/3}), 1 + \lceil \lg (\lceil \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil)$-scheme by Radhakrishnan *et al.* [2] to the following.

Result 2.2 (Theorem 2.3.)**.** *There is an explicit adaptive (n, m, s, t)-scheme with $t = 1 + \lceil \lg (\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil$ and $s = (2^t - 1)m^{2/3}$.*

2.3.3 Our Data Structure for $n = 9$ and $t = 3$

In this section, we give a three probe explicit adaptive scheme storing at most nine elements. Our scheme uses ideas which are similar to Radhakrishnan *et al.* [2] in the Theorem 2 and 3, i.e given a universe $\mathcal{U}$ of size m, we divide it into blocks of size $m^{1/3}$. So, we have $m/m^{1/3} = m^{2/3}$ blocks of size $m^{1/3}$ each. Further, we collect $m^{1/3}$ consecutive blocks to form superblocks of size $m^{2/3}$. So we have $m/m^{2/3} = m^{1/3}$ superblocks of size $m^{2/3}$.

2.3.3.1 Decision Tree

In this scheme, we have seven tables of size $m^{2/3}$ bits, as shown in Figure 2.2. Table T_1 at the root of the tree contains one bit for each of the blocks. Similar to that table at left(T_2) and right(T_3) child of table T_1 contains one bit for each of the blocks. At the leaf, we have four tables named as T_4, T_5, T_6 and T_7 from left to right. Table T_4 has one block of space for each superblock, so all the blocks of superblock one can be mapped at first $m^{1/3}$ bits of a Table T_4. Similarly, all the blocks of superblock two can be mapped to next $m^{1/3}$ bits of Table T_4 and so on. Now, we will talk about the mapping of blocks in tables T_5, T_6 and T_7. The first block from every superblock is given first $m^{1/3}$ bits, the second block from every superblock is given next $m^{1/3}$ bits and so on. So we can see that $m^{2/3}$ bits of space in these tables are divided into blocks of size $m^{1/3}$ bits and index one block from every superblock can be mapped at block one in any of these tables. Similarly, index two blocks from any superblock can be mapped to block two in any of this table and so on.

Given a query element $x \in \mathcal{U}$, we find out the block which contains this element. Further, we make a query to the bit contained for this block in table T_1. If the bit returned is zero, we make the next query to the table kept at the left child else to the table at the right child. We continue in this way, and say the query element is part of the set given to be stored if and only if the last query returns one.

2.3.3.2 Swap Technique

In Figure 2.3, we have shown that there are two superblocks, *superblock* $\mathcal{A}$ and *superblock* $\mathcal{B}$. *superblock* $\mathcal{A}$ is having two elements in it, and *superblock* $\mathcal{B}$ is having one element in it. Furthermore, there are two tables T_0 and T_1. Table T_0 has one block of space for each superblock, and its structure is similar to table T_4 explained earlier. Furthermore, table T_1 can store a complete superblock, and its structure is similar to table T_5 as explained earlier. In the arrangement shown in Figure 2.3, the second block of the *superblock* $\mathcal{A}$ and *superblock* $\mathcal{B}$ cannot be stored together in table T_1, since their position conflicts in table T_1. In this case, we send the block two of *superblock* $\mathcal{A}$ to table T_0 and rest all the blocks of this superblock to table T_1. Furthermore, we send the block two of *superblock* $\mathcal{B}$ to table T_1 and rest all the empty blocks of this superblock are sent to table T_0. We can handle more nonempty blocks using this technique if we have more tables of type T_1. We call the technique described above as the *swap technique* and it has been used throughout the chapter.

2.3.3.3 Storage Scheme

Since table T_4 contains one block of space for each superblock, we can map all the blocks of empty superblock there. So now our only concern is superblock having some element from a set given to be stored. If a superblock contains two or more elements from a set given to be stored, then we call such superblock as a heavy superblock. We divide our storage scheme into two parts depending upon the number of heavy superblocks. Now, since we are given a set of size at most nine, we can have either three or more heavy superblocks or we can have less than or equal to two heavy superblocks.

Case 1. Let us first consider the case where we have less than or equal to two heavy superblocks. In this case, we store the characteristic vector of heavy superblocks in the

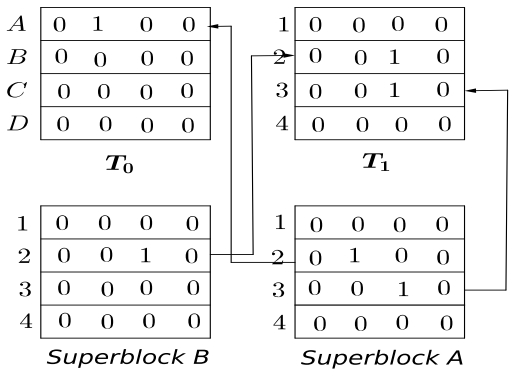


Figure 2.3: The swapping technique

tables T_6 and T_7. Now we could be left with superblocks having only one element. We send the blocks having elements to table T_4 and all the empty blocks to table T_5. Now, since table T_4 contains one block of space for each superblock, blocks having an element will not overlap in the table T_4.

Case 2. Now, we are left with the case where we are having three or more heavy superblocks. In this case, we store the three heavy superblocks in table T_5, T_6 and T_7. After this, we can be left with at most three blocks having elements in it, and we have three tables T_5, T_6 and T_7 to store it. So, we can use the *swap technique* described earlier to store it.

2.3.3.4 Correctness

The correctness of the scheme relies on the fact that blocks having elements are given separate space in one of the table. Also, empty and nonempty blocks are not mapped to the same locations.

We summarize the finding of this section with the following theorem.

Theorem 2.2. *There is a three probe explicit adaptive scheme which stores an arbitrary subset $\mathcal{S}$ of size at most nine from a universe $\mathcal{U}$, and uses $7 \cdot m^{2/3}$ bits of space.*

In the next section, we will use similar technique to generalize this scheme.

2.3.4 Generalized Scheme

We follow the same procedure as that of Theorem 2.2. So, we divide the universe into blocks of size $m^{1/3}$. We then merge $m^{1/3}$ consecutive blocks to form superblocks of size $m^{2/3}$. So we have $m^{1/3}$ superblocks of size $m^{2/3}$. Moreover, we use *swap technique* used in previous section to give an explicit adaptive $(n, m, (2^t - 1)m^{2/3}, t)$-scheme where $t = 1 + \lceil \lg (\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil$.

2.3.4.1 Our Approach

Given a n elements set from a universe of size m, we can have at most $\lfloor \frac{n}{2} \rfloor$ superblock which contains at least two elements. We store $\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil$ tables each having $m^{2/3}$ bits of space in it. Let us name these tables as a table of type $\mathcal{T}$. Mapping of blocks in these tables is similar to the mapping of blocks in tables T_5, T_6 and T_7 explained in Theorem 2.2. Apart from these $\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil$ tables we have two more tables of size $m^{2/3}$ bits. Let us call these tables as Table A and Table B. Mapping of blocks in these tables is similar to that of table T_4 in Theorem 2.2. So we have $\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2$ tables of size $m^{2/3}$ bits, which we are using to map a universe of size m. We store these tables at the leaf of a complete binary tree. Each internal node of this tree contains a table of size $m^{2/3}$ bits, where we have stored single bit for every block.

2.3.4.2 Storage Scheme

Given a n elements set from a universe $\mathcal{U}$ of size m, we construct a decision tree with $\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2$ leaves each capable of holding a table of size $m^{2/3}$ bits. Such a decision tree will have a height $\lceil \lg (\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil$. At each internal node of the decision tree, we store a table of size $m^{2/3}$ bits. In these tables, we have one bit for each block. These internal node guides blocks to the leaves. We follow the following procedure to map elements into the table. We first map all the blocks which are not having any element from the set given to be stored to Table B. Now, our only concern is blocks of superblocks having elements from the set given to be stored. We sort these superblocks according to the number of elements it contains. The block(s) of superblock which contains a maximum number of elements is given a place in the first table of type $\mathcal{T}$. Then we chose next superblock having a maximum number of elements and map the block(s)

of that superblock having an element(s) in it to the second table of type $\mathcal{T}$. In case of tie, superblock can be chosen arbitrarily. We repeat this procedure until we have finished mapping all the blocks having elements, or we have mapped blocks in all the table of type $\mathcal{T}$. At the end of this procedure, we can be left with the following two cases.

Case 1. We are left with superblocks having at most two elements in it. If the superblocks contain only one element, we store the blocks belonging to those superblocks containing an element in Table A. For the superblocks having two elements, we follow the following procedure. Since we were giving preference to superblocks having more elements while mapping, now we can be left with at most $\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil$ superblock each having two elements in it. Now, since each superblock has one block of space in table A and table B, we send one block from remaining superblocks having an element(s) in it to the table A. Now we can have at most $\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil$ blocks belonging to different superblocks having an element in it. We will accommodate these blocks using the *swap technique* described earlier, in the table of type $\mathcal{T}$. We pick a block from remaining blocks having an element in it and map it to the first table of type $\mathcal{T}$, and we map the block which was initially mapped to that place to the table A. Now we take the second block and repeat the procedure with the next table of type $\mathcal{T}$. Since the number of blocks left were at most $\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil$, we would be able to accommodate all of them in the different table of type $\mathcal{T}$.

Case 2. We are left with superblocks having three elements in it. If we are left with a superblock having three elements, then we must have accommodated at least $3\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil$ elements in the table of type $\mathcal{T}$. Since we were always giving preference to a superblock having more elements while storing in the table of type $\mathcal{T}$. So we are now left with at most $\lceil \frac{n}{4} \rceil$ elements which can be part of at most $\lceil \frac{n}{4} \rceil$ blocks. We use *swap technique* described in case one to store the blocks in different tables of type $\mathcal{T}$.

2.3.4.3 Query Scheme

Our query scheme has a binary tree of height $\lceil \lg (\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil$ as described above. It stores the elements in the tables at the leaf nodes. Our query scheme starts from the root node, and for a given element, it queries the bit corresponding to the block which

contains it. If the bit returns one, it makes the next query to the table at right child else; it makes a query to the table at the left child. This process continues until the query is made at the table at a leaf node. If the last query returns one we say the query element is part of the set otherwise, we say that the element is not part of the set given to be stored. Since the height of the tree is $\lceil \lg (\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil$ our query scheme makes $t = 1 + \lceil \lg (\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil$ queries. Space required by all the nodes of the tree is $s = (2^t - 1)m^{2/3}$ bits.

2.3.4.4 Correctness

The correctness of the scheme relies on the fact that blocks which contain elements from the set given to be stored are always given a separate place in one of the tables at the leaf. Moreover, blocks which do not contain any elements from the set given to be stored are always given a place in table B where every bit is set to zero. Hence the query scheme always returns a correct answer.

We summarize the finding of this section with the following theorem.

Theorem 2.3. *There is an explicit adaptive* (n, m, s, t)*-scheme with* $t = 1 + \lceil \lg (\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil$ *and* $s = (2^t - 1)m^{2/3}$.

2.4 Conclusion

In this chapter, we have used the idea of Radhakrishnan *et al.* [2] to divide a universe $\mathcal{U}$ of size m into blocks and superblocks. Using this idea, we have given an explicit adaptive scheme for storing subsets of size at most nine, which answers membership queries with three bitprobes that improves upon the existing explicit schemes in the literature. Additionally, we generalize the abovementioned scheme to a $(n, m, (2^t - 1)m^{2/3}, 1 + \lceil \lg (\lceil \frac{1}{2} \lfloor \frac{n}{2} \rfloor \rceil + 2) \rceil)$-scheme, which slightly improves the Theorem 3 of Radhakrishnan *et al.* [2].

"The intellect has little to do on the road to discovery. There comes a leap in consciousness, call it Intuition or what you will, the solution comes to you and you don't know how or why".

–Albert Einstein

3

Explicit Non-Adaptive Bitprobe Scheme

3.1 Introduction

In this chapter, we will present a few explicit non-adaptive schemes. Non-adaptive schemes are different from adaptive schemes, in a way that in non-adaptive scheme queries are made in parallel, rather than sequential. Our first non-adaptive scheme uses a geometric technique proposed by Kesh [18] to map elements from a universe $\mathcal{U}$ of size m on a rectangular grid. Using this technique, we have obtained a $n+1$ bitprobes scheme storing n elements. Furthermore, we have used this scheme to come up with a generalized scheme, which stores an arbitrary subset $\mathcal{S}$ of size n^2+2n, and uses $1+2n+\lceil \lg(n+1) \rceil$ bitprobes. Our scheme uses $\mathcal{O}(n^{1.5}m^{2/3})$ amount of space. There is an explicit non-adaptive scheme derived from the explicit adaptive scheme by Radhakrishnan *et al.* [2]. It stores an arbitrary subset $\mathcal{S}$ of size n using $2\sqrt{n}+\lg(n)$ non-adaptive bitprobes, and uses $\mathcal{O}(n^{1/2}m^{2/3})$ amount of space. If we compare this result with our non-adaptive scheme in Theorem 3.3, we see that our scheme stores slightly more elements for the same number of non-adaptive bitprobes. However, our scheme uses slightly more space in terms of size of a subset, i.e. n given to be stored.

3.2 Non-adaptive Upper Bounds

In this section, we present a few simple upper bound on the non-adaptive scheme and then use it to come up with better bounds for the same.

3.2.1 A $n+1$ Probes Scheme Storing n Elements

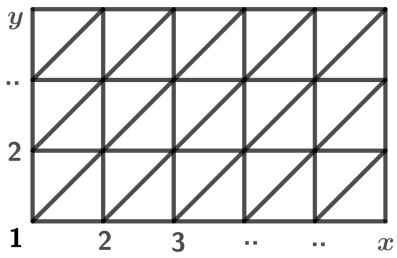


Figure 3.1: Family of lines drawn with slope 1

We place elements from a universe $\mathcal{U}$ of size m on the rectangular grid of size $x \times y$. Further, given a subset $\mathcal{S}$ of size at most n from a universe $\mathcal{U}$, we draw family of lines on the rectangular grid with slope $1, 1/2, 1/3, ..., 1/n+1$. Family of lines with slope 1 is shown in Figure 3.1. Our scheme has $n+1$ tables, named as $T_1, T_2, T_3, ..., T_{n+1}$. Table T_1 contains one bit of space for each line drawn of slope one. Similarly, Table T_2 contains one bit of space for each line drawn of slope $1/2$, and so on.

3.2.1.1 Query Scheme

Given a query element, we see the bit corresponding to all the $n+1$ lines passing through it, in all the tables. We say the given query element is part of the set given to be stored if and only if all the tables return one.

3.2.1.2 Storage Scheme

Given a subset $\mathcal{S}$ of size at most n from a universe $\mathcal{U}$, we set the bits of all the lines passing through these elements on the rectangular grid to be one. Further, bits corresponding to the rest of the lines are set to zero.

3.2.1.3 Space of the data structure

Looking at the way lines are drawn in Figure 3.1, we can say that number of lines drawn for the slope $1/i$ is less than or equal to $x + c \cdot iy$, where c is a constant. Summing it up for all the slopes, we get total number of lines to be less than or equal to

$$nx + c_1 n^2 y \tag{3.1}$$

, where c_1 is a constant. Since all the elements from a universe $\mathcal{U}$ of size m are mapped on the rectangular grid of size m, we have $xy = m$. Substituting $x = m/y$ in equation 3.1, we get following equation

$$S(y) = \frac{mn}{y} + c_1 n^2 y. \tag{3.2}$$

Further, choosing $y = c \cdot \frac{m}{n}$, we get space taken by our data structure to be $\mathcal{O}(n^{1.5}m^{1/2})$.

3.2.1.4 Correctness

If the point is part of the set given to be stored, we have set the bit corresponding to all the lines passing through the point to be one. So the query belonging to these elements will always be answered correctly. Further, if the query element is not part of the set given to be stored, then it can have at most n lines from elements $\mathcal{S}$ to be one. So, the $n+1$th line passing through that element must have bit zero set for it. So query for this element will also be answered correctly.

We summarize the finding of this section in the following theorem:

Theorem 3.1. *There is a $n+1$ probes explicit non adaptive scheme which stores an arbitrary subset $\mathcal{S}$ of size at most n, from a universe $\mathcal{U}$ of size m and requires $\mathcal{O}(n^{1.5}m^{1/2})$ amount of space.*

3.2.2 A $n+3$ Probes Scheme Storing $2n+1$ Elements

In this scheme, we divide the universe $\mathcal{U}$ of size m into blocks of size $m^{2/3}$. So, we have $m^{1/3}$ blocks of size $m^{2/3}$. Further, our scheme has $n+3$ tables, named as $T_1, T_2, ..., T_{n+3}$. Given a subset $\mathcal{S}$ of size at most $2n+1$ from a universe $\mathcal{U}$ of size m, we use the following procedure to answer the membership queries correctly.

3.2.2.1 Storage Scheme

In table T_1, we store one bit for each block. Since there are $m^{1/3}$ blocks, size of table T_1 will be $m^{1/3}$. Further, since the subset given to be stored is of size at most $2n+1$, there could be at most one block having $n+1$ elements or more in it. We store zero for the block having $n+1$ or more element in it, in table T_1. For rest of the blocks, we store one in table T_1. Table T_2 stores the characteristic vector of the block having at least $n+1$ elements. If there is not any block having at least $n+1$ elements, we store

all zeros in table T_2. Now, we are left with blocks having at most n elements in it, and we are left with $n+1$ tables. There could be at most $m^{1/3}$ blocks of such kind. We use Theorem 3.1 to map the elements from such blocks in table $T_3, T_4, ..., T_{n+3}$. Space taken by Theorem 3.1 would be $cm^{1/3} \cdot n^{1.5}(m^{2/3})^{1/2}$. So the size of our data structure is $\mathcal{O}(n^{1.5}m^{2/3})$.

3.2.2.2 Query Scheme

Since our scheme is non-adaptive query for each element will be done in all the tables. But, our query function is designed in such a way that if query is made from a block having at least $n+1$ elements, it ignores the last $n+1$ queries. Also, our query function ignore the first two probes if the query is from the block having at most n elements in it. Our query function is

$$f(t_1, t_2, ..., t_{n+3}) = (\neg t_1 \wedge t_2) \vee (t_1 \wedge t_3 \wedge t_4 \wedge t_5 \wedge ..... \wedge t_{n+3}). \tag{3.3}$$

Where $t_1, t_2, ..., t_{n+3}$ are the bits returned for a query element in table $T_1, T_2, ..., T_{n+3}$. It is easy to see that if the query is made from the block with at least $n+1$ elements in it, then query function ignores last $n+1$ query, and since characteristic vector of the block having at least $n+1$ elements is stored in table T_2, query will be answered correctly. Further, if the query is made from other blocks, as these blocks contain at most n elements from the set given to be stored, and we have $n+1$ query left we can answer correctly using Theorem 3.1.

We summarize the finding of this section in the following theorem.

Theorem 3.2. *There is a $n+3$ probes explicit non adaptive scheme which stores an arbitrary subset $\mathcal{S}$ of size at most $2n+1$, from a universe $\mathcal{U}$ of size m and requires $\mathcal{O}(n^{1.5}m^{2/3})$ amount of space.*

3.2.3 A $1 + 2n + \lceil \lg(n+1) \rceil$ Probes Scheme Storing $n^2 + 2n$ Elemnts

We follow the similar technique, mentioned in the description of Theorem 3.2. We divide the universe $\mathcal{U}$ of size m, into blocks of size $m^{2/3}$. So, we have $m^{1/3}$ blocks. Given a $n^2 + 2n$ elements subset $\mathcal{S}$ from the universe $\mathcal{U}$, we can have at most n blocks with at least $n+1$ elements. We store the characteristic vector of these blocks separately, in

tables numbered from 2 to $n+1$. In table T_1, we have $\lceil \lg(n+1) \rceil$ bits of space for each block . For the block having at least $n+1$ elements, rank could be any number from 1 to n. For the blocks having at most n elements, we store $\lceil \lg(n+1) \rceil$ zeros. So, the size of table T_1 is $m^{1/3} \lg(n+1)$. Furthermore, we use Theorem 3.1 to map the blocks having at most n elements from the set given to be stored. Since there could be at most $m^{1/3}$ blocks of such kind, total space taken by our data structure is

$$S \leq m^{1/3} \lg(n+1) + nm^{2/3} + cn^{1.5}m^{2/3} = \mathcal{O}(n^{1.5}m^{2/3}). \tag{3.4}$$

Our query function behaves similar to Theorem 3.2. If a query is made from the block having at least $n+1$ elements it ignores all the tables except the table belonging to its rank, where it's characteristic vector is stored. On the contrary, if the query is made from the block having at most n elements, it ignores the bit returned by tables which stores characteristic vectors of blocks having at least $n+1$ elements, and answer the query depending upon the last $n+1$ queries. Correctness follows from Theorem 3.2.

We summarize the finding of this subsection in the following theorem.

Theorem 3.3. *There is a* $1+2n+\lceil \lg(n+1) \rceil$ *probes explicit non adaptive scheme which stores an arbitrary subset* $\mathcal{S}$ *of size at most* n^2+2n*, from a universe* $\mathcal{U}$ *of size* m *and requires* $\mathcal{O}(n^{1.5}m^{2/3})$ *amount of space.*

3.3 Conclusion

In this chapter, we have come up with a non-adaptive scheme, which stores an arbitrary subset $\mathcal{S}$ of size n. Our scheme uses slightly fewer bitprobes, and slightly more space than the existing state of the arts results in terms of n.

"Archimedes will be remembered when Aeschylus is forgotten, because languages die and mathematical ideas do not."

–G.H. Hardy, A Mathematicians Apology

4

Two Improved Bitprobe Scheme for Storing Small Sets

4.1 Introduction

As discussed in previous chapters, in this chapter as well, we consider a universe $\mathcal{U}$ of m elements. Further, we consider a subset $\mathcal{S}$ of $\mathcal{U}$ containing n elements. In the *bitprobe model*, we study the problem of storing the subset $\mathcal{S}$ in a data structure of size s such that membership queries can be answered by probing at most t bits of the data structure. For a given m and n, the schemes in this model try to optimize the space used by the data structure, s, and the number of bitprobes used for membership queries, t. Such schemes are often denoted by (n, m, s, t). To decide membership of an element of $\mathcal{U}$ in the subset $\mathcal{S}$, the location of a bitprobe might depend on the answers of the previous bitprobes. Such schemes are called *adaptive*. If the location of every bitprobe is independent of the answers we receive in other bitprobes, the schemes are called *non-adaptive*. Nicholson *et al.* [11] has surveyed the bitprobe model with discussions about current state of the art and a selection of open problems.

In this chapter, we use the idea of Kesh [18] to map the elements from universe $\mathcal{U}$ of size m on integral point of suitable size cube. Further, we draw lines passing through the integral points of the cube to partition the universe into sets. Specifically, we partition the cube with respect to $\mathcal{X}, \mathcal{Y}$ and $\mathcal{Z}$-axis. Further, we see the interaction of various sets

to come up with a storage and query scheme to store a subset $\mathcal{S}$ given to be stored.

4.1.1 The Problem Statements

In particular, we address the following two problems. The first one is to design an explicit adaptive scheme for storing three elements and deciding membership queries using two bitprobes, i.e., an adaptive scheme for the case when $n = 3$ and $t = 2$. A corollary of this scheme would be a non-adaptive scheme for $n = 3$ and $t = 3$. The second problem we tackle is to design a non-adaptive scheme which decides membership using four non-adaptive bitprobes, i.e., a non-adaptive scheme with $t = 4$. For four non-adaptive bitprobes, we give a scheme which stores an arbitrary subset of size at most four and uses $\mathcal{O}(m^{2/3})$ amount of space. Further, we improve it to store a subset of size at most five, using asymptotically same space.

4.1.2 Previous Results

Alon and Fiege [9] in their seminal paper presented an adaptive $(n, m, s, 2)$ scheme where $s = o(m)$. This scheme has been further improved by Garg and Jaikumar Radhakrishnan [15], they have proposed a generalised scheme that can store arbitrary subsets of size n, where $n < \log m$, and uses $\mathcal{O}(m^{1-\frac{1}{4n+1}})$ amount of space. For the particular case of $n = 3$, the space requirement turns out to be $\mathcal{O}(m^{12/13})$. Garg [16] further improved the bounds to $\mathcal{O}(m^{1-\frac{1}{4n-1}})$(for $n < (1/4)(\log m)^{1/3}$), which improved the scheme for $n = 3$ to $\mathcal{O}(m^{10/11})$. The explicit scheme that accomodates the most number of elements for $t = 2$ and uses $\mathcal{O}(m^{2/3})$ amount of space is due to Radhakrishnan *et al.* [2], where they present a $(2, m, \mathcal{O}(m^{2/3}), 2)$-scheme.

For the case $n = 3$ and $t = 3$ in the non-adaptive bitprobe model, there does not exist any explicit scheme with $s = \mathcal{O}(m^{2/3})$.

In the non-adaptive bitprobe model, the best known explicit scheme with $t = 4$ and $s = o(m)$ is due to Blue [19], and their scheme can handle a subset of size at most three($n = 3$); the best known non-explicit scheme is due to Alon and Fiege [9], and it can handle $n = o(m)$.

4.1.3 Our Contribution

We present an explicit adaptive scheme for $n = 3$ and $t = 2$ that uses $s = \mathcal{O}(m^{2/3})$ bits of storage. This scheme improves upon the $(2, m, \mathcal{O}(m^{2/3}), 2)$ scheme of Radhakrishnan *et al.* [2]. It uses a technique by Kesh [18] of mapping the elements of our universe into the integral points of a three dimensional cube, and then looking into the projections of the various points onto the two dimensional faces of the cube. A by-product of this scheme is a non-adaptive scheme for $n = 3$ and $t = 3$, and using $s = \mathcal{O}(m^{2/3})$ bits. Our next scheme is a non-adaptive scheme for $n = 4$ and $t = 4$, and uses $s = \mathcal{O}(m^{2/3})$ bits. This scheme, too, uses the approach described above. Furthermore, we have used idea of Radhakrishnan *et al.* [2] to divide a universe into blocks and superblocks. Using this idea we have improved our non-adaptive scheme to store subset of size five ($n = 5$) using four non-adaptive bitprobes ($t = 4$), and still using $s = \mathcal{O}(m^{2/3})$ bits of space.

4.2 Adaptive Scheme for $n = 3$ and $t = 2$

In this section, we present two adaptive bitprobe data structure storing three elements.

4.2.1 Arrangement of Elements

In the three dimensional space with coordinate axes x, y, and z, consider a cube in the first orthant. The cube has sides of magnitude $m^{1/3}$, and it is so placed that one of its vertices lies on the origin, and its sides are parallel to the coordinate axes. The number of points, within and on the cube, with all integer coordinates is m. We place all of the m elements of our universe $\mathcal{U}$ on those points. Going forward, we would refer to an element of $\mathcal{U}$ by the coordinates of the point on which it lies. As an example, an element of $\mathcal{U}$ lying on the point (a, b, c) will be called as the element (a, b, c). As a consequence, we will use the words 'element' and 'point' interchangeably.

Based on simple geometric constructions, we now define four distinct partitions of our universe $\mathcal{U}$. The partitions are named $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$, and $\mathcal{D}$. We start by defining the partition $\mathcal{X}$.

For an element (a, b, c), the set $X(a, b, c)$ is defined as follows. Draw a line through the point (a, b, c) which is normal to the yz-plane and parallel to the x-axis. The elements

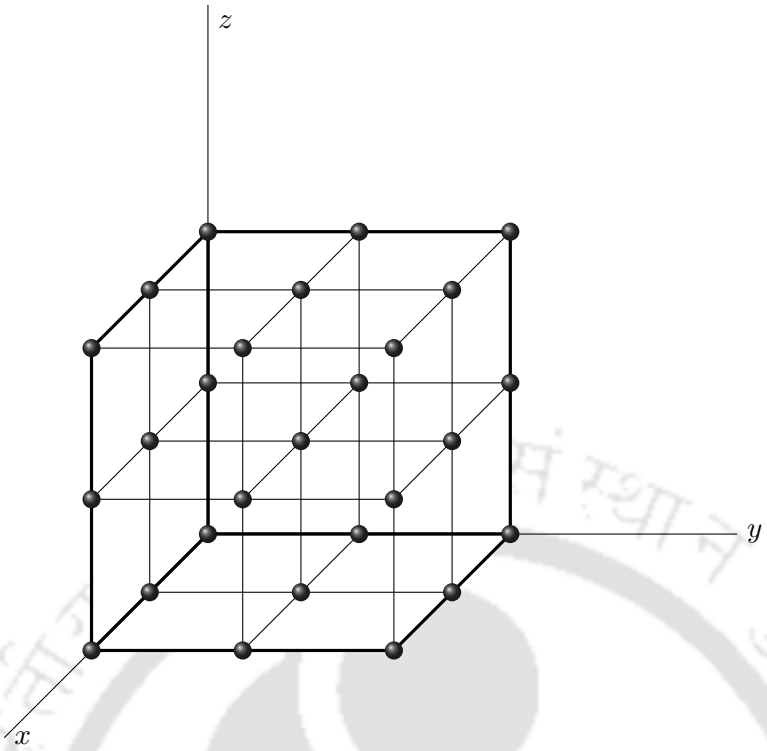


Figure 4.1: Elements placed on three dimensional cube

of $\mathcal{U}$ that falls on this line belong to the set $X(a, b, c)$. More formally,

$$X(a, b, c) = \{\ (d, e, f) \in \mathcal{U}\ \mid\ e = b \text{ and } f = c\ \}.$$

The next two observations would prove that for two elements (a, b, c) and (d, e, f) of our universe, the corresponding sets $X(a, b, c)$ and $X(d, e, f)$ are either equal or disjoint.

Observation 4.1. *If an element (d, e, f) belongs to the set $X(a, b, c)$, then the sets $X(a, b, c)$ and $X(d, e, f)$ are equal.*

Proof. From the geometric intuition of the sets $X(a, b, c)$ and $X(d, e, f)$, it is clear that the two sets must be equal. There is only one line that is normal to the yz-plane, is parallel to the x-axis, and passes through both the points.

We now argue the same formally. As (d, e, f) belongs to $X(a, b, c)$, we have $e = b$ and $f = c$. So, the point (d, e, f) is actually the point (d, b, c). Let (g, h, i) be a member of $X(d, b, c)$. Then, we must have $h = b$ and $i = c$. So, the point (g, h, i) is the same as the point (g, b, c), and hence, it also belongs to $X(a, b, c)$. Therefore, we have $X(d, e, f) \subseteq X(a, b, c)$.

What we have argued so far is the following – if a point (d, e, f) belongs to the set

$X(a, b, c)$, then $X(d, e, f) \subseteq X(a, b, c)$. If we can now prove that the point (a, b, c) also belongs to the set $X(d, e, f)$, then we would have established that $X(a, b, c) \subseteq X(d, e, f)$, which in turn would prove the observation.

As it is given that $(d, e, f) \subseteq X(a, b, c)$, so, we have $e = b$ and $f = c$. So, the point (a, b, c) is actually the point (a, e, f), and from the definition of the set $X(d, e, f)$, if follows that $(a, b, c) = (a, e, f) \in X(d, e, f)$. □

Observation 4.2. *If an element (d, e, f) does not belong to the set $X(a, b, c)$, then the sets $X(a, b, c)$ and $X(d, e, f)$ are disjoint.*

Proof. The geometric interpretation of the sets $X(a, b, c)$ and $X(d, e, f)$ tells us that if the point (d, e, f) does not lie on the line defining the set $X(a, b, c)$, then the line defining the set $X(d, e, f)$ is parallel to the line $X(a, b, c)$. We proceed to argue formally.

As the point (d, e, f) does not lie on the line defining the set $X(a, b, c)$, then either $e \neq b$, or $f \neq c$, or both. Without loss of generality, let us assume that the y-coordinates of the two points are unequal, i.e. $e \neq b$. For an arbitrary point (g, h, i) belonging to the set $X(d, e, f)$, we have $h = e \neq b$. So, the point (g, h, i) cannot belong to the set $X(a, b, c)$. This proves that the sets $X(a, b, c)$ and $X(d, e, f)$ are disjoint. □

We now define the partition $\mathcal{X}$ as follows –

$$\mathcal{X} = \left\{ X(0, a, b) \mid 0 \leq a, b < m^{1/3} \right\}.$$

For $\mathcal{X}$ to be a partition, the sets forming $\mathcal{X}$ must be disjoint and they must cover the whole universe $\mathcal{U}$. Observation 4.2 guarantees that the first property is satisfied. We prove next that every element of $\mathcal{U}$ belongs to some member of $\mathcal{X}$. This is easy to see as an element (a, b, c) lies on the line defining the set $X(0, b, c)$.

The following lemma states the size of the partition.

Lemma 4.1. *The size of the partition $\mathcal{X}$ is $m^{2/3}$.*

Proof. This is an easy consequence of the definition of $\mathcal{X}$. □

The partitions $\mathcal{Y}$ and $\mathcal{Z}$ are similarly defined. We start by defining the sets $Y(a, b, c)$

and $Z(a, b, c)$.

$$\begin{aligned} Y(a,b,c) &= \{\ (d,e,f) \in \mathcal{U} \ \mid\ d = a \text{ and } f = c\ \}; \\ Z(a,b,c) &= \{\ (d,e,f) \in \mathcal{U} \ \mid\ d = a \text{ and } e = b\ \}. \end{aligned}$$

So, the set $Y(a, b, c)$ is defined by the line through the point (a, b, c) which is normal to the xz-plane and parallel to the y-axis. Similarly, the set $Z(a, b, c)$ is defined by the line through the point (a, b, c) which is normal to the xy-plane and parallel to the z-axis. We can now define the partitions $\mathcal{Y}$ and $\mathcal{Z}$.

$$\begin{aligned} \mathcal{Y} &= \{\ Y(a,0,b) \ \mid\ 0 \le a, b < m^{1/3}\ \} \\ \mathcal{Z} &= \{\ Z(a,b,0) \ \mid\ 0 \le a, b < m^{1/3}\ \} \end{aligned}$$

We can also make the following comment about the size of these partitions.

Lemma 4.2. *The size of the partitions $\mathcal{Y}$ and $\mathcal{Z}$ are both equal to $m^{2/3}$.*

We will not formally argue any of the facts about the partitions $\mathcal{Y}$ and $\mathcal{Z}$ as the arguments follows closely along the lines of the proof of the properties of partition $\mathcal{X}$.

The last partition we define is the partition $\mathcal{D}$. It will be used in the next chapter. As usual, we start by defining the set $D(a, b, c)$. This set consists of all those points that lie on that line through the point (a, b, c) which lies completely on the xy-plane through the point and has a slope of 45° on that plane. Formally, the set is defined thus.

$$D(a,b,c) = \{\ (d,e,f) \in \mathcal{U} \ \mid\ f = c \text{ and } d - a = e - b\ \}$$

Finally, the partition $\mathcal{D}$ is defined as follows.

$$\mathcal{D} = \left\{\ D(a,0,b) \ \mid\ 0 \le a, b < m^{1/3}\ \right\} \bigcup \left\{\ D(0,a,b) \ \mid\ 0 \le a, b < m^{1/3}\ \right\}$$

The size of this partition is given in the following lemma.

Lemma 4.3. *The size of partition $\mathcal{D}$ is $2 \times m^{2/3}$.*

Again, the proof of the properties of this partition follows directly from the geometric interpretation of the sets that form the partition, and we leave it for the reader to argue.

4.2.2 Our Data Structure

Our data structure consists of three tables, one for each of the partitions $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ of Section 4.2.1. To abstain from introducing too many notations and risk losing clarity, we abuse the notation for the partitions and use them to denote the tables in our data structure as well. It will be clear from the context whether the notation $\mathcal{X}$ denotes the partition of $\mathcal{U}$ or the table corresponding to that partition.

For every set in the various partitions, we reserve one bit in the corresponding table in our data structure. Again, we abuse the notation and use the name for a set to also refer to its corresponding bit in our data structure. To take an example, $X(a, b, c)$ would refer to a set in partition $\mathcal{X}$ and also the bit reserved for the set in the table $\mathcal{X}$.

The following lemma follows directly from the Lemmas 4.1 and 4.2.

Lemma 4.4. *The size of our data structure is* $3 \times m^{2/3}$.

4.2.3 The Query Scheme

In the adaptive bitprobe model, the query scheme is described by a binary tree, called in the literature as the *decision tree*. This tree tells us the location of a bitprobe, given that the answers of the previous bitprobes are known. The decision tree for our scheme is shown in Figure 4.2.

Consider an element (a, b, c) of $\mathcal{U}$, and we want to determine whether the element belongs to the subset $\mathcal{S}$. From the definitions of sets and partitions, we know that the element belongs to the set $X(0, b, c)$ in table $\mathcal{X}$, the set $Y(a, 0, c)$ in $\mathcal{Y}$, and the set $Z(a, b, 0)$ in table $\mathcal{Z}$. The decision tree tells us that the first query will be made in table $\mathcal{Z}$. The location of the bitprobe will be at $Z(a, b, 0)$. If the bit stored in that location is 0, we need to follow the left child and query the location $Y(a, 0, c)$ in table $\mathcal{Y}$. On the other hand, we need to query the location $X(0, b, c)$ in table $\mathcal{X}$ if the bit stored is 1. We deduce that the element (a, b, c) belongs to the subset $\mathcal{S}$ if and only if the second query returns 1.

4.2.4 The Storage Scheme

Consider an arbitrary subset $\mathcal{S} = \{ (a_1, b_1, c_1), (a_2, b_2, c_2), (a_3, b_3, c_3) \}$ of our universe $\mathcal{U}$. The storage scheme describes how to set the bits of our datastructure so that the query

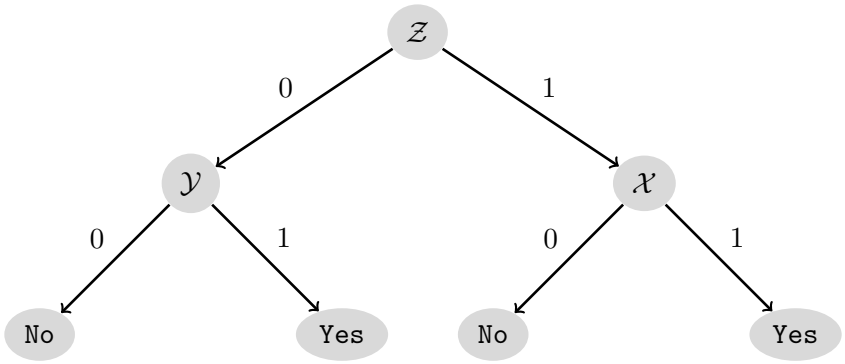


Figure 4.2: The decision tree for the adaptive bitprobe scheme.

scheme can answer correctly. The assignment of bits depend on the how the members of $\mathcal{S}$ are chosen from $\mathcal{U}$. We describe each such case separately, and provide the proof of correctness alongside it.

Case I Let us consider the scenario when the x-coordinate of the three points are equal, i.e. $a_1 = a_2 = a_3 = a$ (say). The members of $\mathcal{S}$ in this case looks like $\{ (a, b_1, c_1), (a, b_2, c_2), (a, b_3, c_3) \}$. We set the bits in table $\mathcal{Z}$ as follows – $Z(a, b_1, 0) = Z(a, b_2, 0) = Z(a, b_3, 0) = 1$. The rest of the bits in the table is set to 0. In table $\mathcal{Y}$, all bits are set to 0. In table $\mathcal{X}$, we have the following arrangement – $X(0, b_1, c_1) = X(0, b_2, c_2) = X(0, b_3, c_3) = 1$, and the rest of the bits are set to 0.

We now argue that in this case the query algorithm gives correct answers. It is easy to see that the query scheme for any element of subset $\mathcal{S}$ will always return `Yes`. If any member of $\mathcal{S}$ is queried, the assignment of the bits in table $\mathcal{Z}$ tells us that it will get a 1, and hence query table $\mathcal{X}$. In table $\mathcal{X}$, precisely those bits that correspond to the three members of $\mathcal{S}$ have been set to 1.

Let us now consider an element (a', b', c') of $\mathcal{U}$ which upon query in our datastructure got a `Yes`. Then it must got the `Yes` from table $\mathcal{X}$ as all the bits of table $\mathcal{Y}$ are set to 0. To go to table $\mathcal{X}$, it must get a 1 from table $\mathcal{Z}$. Without loss of generality, let it get the 1 from the bit $Z(a, b_1, 0)$. So, it must be the case that $a' = a$ and $b' = b_1$, and the element in question is $(a', b', c') = (a, b_1, c')$. In table $\mathcal{X}$, it will query such a bit whose y-coordinate is b_1, and which has been set to 1. One such bit is $X(0, b_1, c_1)$. If $b_1 = b_2$, then $X(0, b_2, c_2)$ could also be a possible candidate. If our element queries the bit $X(0, b_1, c_1)$, then $c' = c_1$. So, the element in question is $(a', b', c') = (a, b_1, c_1)$, which

indeed is a member of $\mathcal{S}$. We can argue similarly in other cases as well.

Case II Let us now consider the scenario where two elements of $\mathcal{S}$ have the same x-coordinate, and the remaining element's x-coordinate is different from the other two. Without loss of generality, let $a_1 = a_2 = a$ (say), and $a \neq a_3$. So, the elements of the set $\mathcal{S}$ are $\{ (a, b_1, c_1), (a, b_2, c_2),$
$(a_3, b_3, c_3) \}$. We would consider two subcases, one in which the y-coordinate of the third element is equal to the y-coordinate of one of the first two elements, the other in which the y-coordinate of the third element is distinct from the y-coordinates of the first two elements.

Case II(A) We consider the first subcase here, in which the y-coordinate of the third element is equal to the y-coordinate of one of the first two elements. Without loss of generality, let $b_1 = b_3 = b$ (say). So, the elements of $\mathcal{S}$ are $\{ (a, b, c_1), (a, b_2, c_2), (a_3, b, c_3) \}$. A further complication might arise if $b = b_2$. We consider the possibility in the subcases below.

Case II(A)(a) Let us consider the scenario where $b \neq b_2$. So, the elements of $\mathcal{S}$ remain as $\{ (a, b, c_1), (a, b_2, c_2), (a_3, b, c_3) \}$. If such is the case, in table $\mathcal{Z}$, we set $Z(a, b', 0) = 1$, for all $b' \neq b$. We also set $Z(a_3, b, 0) = 1$. The rest of the bits of table $\mathcal{Z}$ are set to 0. In table $\mathcal{Y}$, we set $Y(a, 0, c_1) = 1$, and all the other bits to 0. Finally, in table $\mathcal{X}$, we set $X(0, b_2, c_2) = X(0, b, c_3) = 1$, and the other bits to 0.

We now argue that this arrangement of bits in our datastructure is correct. The first member of $\mathcal{S}$, namely (a, b, c_1), will get a 0 from table $\mathcal{Z}$ ($Z(a, b, 0)$ has been set to 0), go to table $\mathcal{Y}$ at $Y(a, 0, c_1)$ and get a 1. As $Z(a, b_2, 0)$ and $Z(a_3, b, 0)$ have both been set to 1, the second and the third elements of $\mathcal{S}$ will go to table $\mathcal{X}$ at the locations $X(0, b_2, c_2)$ and $X(0, b, c_3)$, respectively, and get 1. So, the query scheme for all the members of $\mathcal{S}$ will return `Yes`.

Consider an element (a', b', c') that got a `Yes` upon query in our datastructure. Since both tables $\mathcal{Y}$ and $\mathcal{X}$ have one or more bits set to 1, (a', b', c') could have got its `Yes` answer from either of them.

Let us first consider the case where the element (a', b', c') went to table $\mathcal{Y}$ and got its 1 from there. $Y(a, 0, c_1)$ is the only bit in that table that is set to 1. So, we have $a' = a$ and $c' = c_1$, which makes the element (a', b', c') to be (a, b', c_1). The only bit

in table $\mathcal{Z}$ which is set to 0 and has its x-coordinate as a is $Z(a, b, 0)$, and our element must query this bit. So, we further have $b' = b$. Thus the element $(a', b', c') = (a, b, c_1)$, a member of $\mathcal{S}$.

The other case to consider is when the element (a', b', c') went to table $\mathcal{X}$ and got a `Yes`. The way bits are set in $\mathcal{X}$, we have either $b' = b_2, c' = c_2$ or $b' = b, c' = c_3$. So, the element could be one of (a', b_2, c_2) and (a', b, c_3). The only way (a', b_2, c_2) can get a 1 in table $\mathcal{Z}$ is by querying the bit $Z(a, b_2, 0)$, which implies that $a' = a$, and $(a', b', c') = (a, b_2, c_2)$. (a', b, c_3) can get a 1 from table $\mathcal{Z}$ by querying the bit $Z(a_3, b, 0)$. So, we have $a' = a_3$, and hence $(a', b', c') = (a_3, b, c_3)$. So, in all of the cases, (a', b', c') turns out to be a member of $\mathcal{S}$.

Case II(A)(a)(b) We next consider the case where $b = b_2$. The elements of $\mathcal{S}$, now, would be $\{ (a, b, c_1), (a, b, c_2), (a_3, b, c_3) \}$. This case is not too dissimilar from the previous case in the arrangement of its elements. The assignment of table Z remains unchanged. In table $\mathcal{Y}$, we set $Y(a, 0, c_1) = Y(a, 0, c_2) = 1$, and all the other bits to 0. Finally, in table $\mathcal{X}$, we set $X(0, b, c_3) = 1$, and the other bits to 0.

The proof of correctness is similar to the previous case, and we omit it for the sake of brevity.

Case II(B) We now consider the second subcase where the y-coordinate of the the third element is distinct from the y-coordinate of the other two elements. So, the set $\mathcal{S}$ is $\{ (a, b_1, c_1),$
$(a, b_2, c_2), (a_3, b_3, c_3) \}$. If such is the case, we set $Z(a, b_1, 0) = Z(a, b_2, 0) = Z(a_3, b_3, 0) = 1$, and the rest of the bits of table $\mathcal{Z}$ to 0. All bits of table $\mathcal{Y}$ are set to 0. The table $\mathcal{X}$, only the following bits are set to 1 – $X(0, b_1, c_1), X(0, b_2, c_2), X(0, b_3, c_3)$.

We again argue that only elements of $\mathcal{S}$ upon query in our datastructure will get a `Yes` answer. Only those bits of tables $\mathcal{Z}$ and $\mathcal{X}$ that correspond to the members of $\mathcal{S}$ are set to 1. So, the members of $\mathcal{S}$ will query table $\mathcal{Z}$, get 1 and consequently query table $\mathcal{X}$ and get `Yes`.

Let (a', b', c') be an arbitrary element which got a `Yes` from our datastructure. Then, it must go to table $\mathcal{X}$ to get that answer, as all the elements of table $\mathcal{Y}$ are set to 0. Let it be the case that it got a 1 by querying the bit $X(0, b_1, c_1)$. So, it must be the case that $b' = b_1$ and $c' = c_1$, and hence $(a', b', c') = (a', b_1, c_1)$. For such an element to get a 1 from table $\mathcal{Z}$, it must query $Z(a, b_1, 0)$, and thus $a' = a$. So, we have $(a', b', c') = (a, b_1, c_1)$,

a member of $\mathcal{S}$. We can similarly argue that if (a', b', c') queries some other bit in $\mathcal{X}$ to get a 1, it will still be a member of $\mathcal{S}$.

Case III The last case to consider is when the x-coordinates of all the members of $\mathcal{S}$ are distinct. We set the bits of the three tables as follows. In table $\mathcal{Z}$, $Z(a_1, b_1, 0) = Z(a_2, b_2, 0) = Z(a_3, b_3, 0) = 0$, and the rest of the bits are set to 1. In table $\mathcal{Y}$, $Y(a_1, 0, c_1) = Y(a_2, 0, c_2) = Y(a_3, 0, c_3) = 1$, and the other bits are set to 0. All bits in table $\mathcal{X}$ are set to 0.

We first prove that the members of $\mathcal{S}$ upon query will get `Yes`. Bits in table $\mathcal{Z}$ corresponding to the three members are set to 0, so the second query for all of the members will be made in table $\mathcal{Y}$. In table $\mathcal{Y}$, all the bits corresponding to the members of $\mathcal{S}$ have been set to 1, hence they will get a `Yes`.

Now we argue that if an element, say (a', b', c'), upon query in our datastructure got a `Yes`, then it must be a member of $\mathcal{S}$. The element can get a `Yes` only from table $\mathcal{Y}$. Without loss of generality, let us assume that it queried $Y(a_1, 0, c_1)$. In this case, we have $a' = a_1, c' = c_1$, and hence $(a', b', c') = (a_1, b', c_1)$. In table $\mathcal{Z}$, it must get a 0 so that it can go to table $\mathcal{Y}$, and the only bit which is set to 0 and whose x-component is a_1 is the bit $Z(a_1, b_1, 0)$. This gives us $b' = b_1$, and thus $(a', b', c') = (a_1, b_1, c_1)$, which is a member of $\mathcal{S}$.

This concludes the description of the storage scheme and our proof of correctness. We can now summarise the conclusions of this section as follows.

Theorem 4.5. *There is an explicit adaptive* $(3, m, 3 \times m^{2/3}, 2)$ *scheme.*

This scheme also gives rise to a non-adaptive scheme. If we decide to probe all the tables of our decision tree irrespective of the findings on our first query in table $\mathcal{Z}$, then we would have made three bitprobes in our datastructure, instead of two. More importantly, the scheme now becomes non-adaptive as the location of every query is fixed. The query scheme, on getting the results of the three queries, can now decide membership by consulting the decision tree of Figure 5.1. Thus, we can claim the following.

Corollary 4.1. *There is an explicit non-adaptive* $(3, m, 3 \times m^{2/3}, 3)$ *scheme.*

4.3 A Non-adaptive Scheme for $n = 4$ and $t = 4$

We present our final scheme, an explicit non-adaptive scheme for four elements ($n = 4$) and four queries ($t = 4$).

4.3.1 Our Data structure

Our data structure consists of four tables, one for each partition of Section 4.2.1, namely $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$, and $\mathcal{D}$. As in the previous section, we refrain from introducing too many notations and use the notations for the partitions to denote the tables in our data structure as well. Furthermore, we reserve one bit for every set in a partition in its corresponding table. As before, we use the same name for the sets in the partitions and the corresponding bits in the tables. As an example, $D(a, b, c)$ would refer to a set in partition $\mathcal{D}$ and also the bit reserved for the set in the table $\mathcal{D}$.

The following lemma follows directly from the Lemmas 4.1, 4.2, and 4.3.

Lemma 4.6. *The size of our data structure is* $5 \times m^{2/3}$.

4.3.2 The Query Scheme

If we want to ascertain the membership in set $\mathcal{S}$ of an element (a, b, c) of $\mathcal{U}$, we query its corresponding bits in each of the tables of our data structure, namely the bit $X(0, b, c,)$ in table $\mathcal{X}$, the bit $Y(a, 0, c,)$ in table $\mathcal{Y}$, and so on. Upon receiving the query answers, we apply the majority function to determine the membership in $\mathcal{S}$. If the majority of the bits returned is 1, then and only then we declare that the element in question is a member of $\mathcal{S}$.

4.3.3 The Storage Scheme

In this section, we describe how to set the bits of our data structure such that the query scheme can correctly answer membership queries. The way bits are set depends upon how the members of $\mathcal{S}$ are chosen. We discuss below each such case, and provide proof of correctness of the scheme alongside it.

In the following discussion, we assume that $\mathcal{S} = \{(a_1, b_1, c_1), (a_2, b_2, c_2), (a_3, b_3, c_3), (a_4, b_4, c_4)\}$.

Case I – Let us assume that all the four members of $\mathcal{S}$ lie in the same xy-plane, i.e. $c_1 = c_2 = c_3 = c_4 = c$(say). In this case, we set the bits corresponding to each member of $\mathcal{S}$ in tables $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ to 1. The rest of the bits in all of the tables, including $\mathcal{D}$, are set to 0. So, for the element (a_1, b_1, c), $X(0, b_1, c) = Y(a_1, 0, c) = Z(a_1, b_1, 0) = 1$, and similarly for the other members of $\mathcal{S}$.

We now provide the correctness proof of our scheme in this scenario. Let us assume that an element (a', b', c'), upon query in our data structure, has got the majority of its answers as 1. As all of the bits of table $\mathcal{D}$ is set to 0, it must get 1 from each of the tables $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$. As only four bits in table $\mathcal{Z}$ are set to 1, let us assume that it got its 1 from the bit $Z(a_1, b_1, 0)$. This implies that $a' = a_1$ and $b' = b_1$, and hence the point in question is (a_1, b_1, c'). This point also got a 1 when it queried table $\mathcal{X}$. This table also has four bits set to 1 corresponding to the four members of $\mathcal{S}$. The element (a_1, b_1, c') will query such of bit of $\mathcal{X}$ which is set to 1 and whose y-coordinate is b_1. $X(a_1, b_1, c)$ is one such bit. If $b_1 = b_2$, then $X(a_2, b_2, c)$ is also a possible candidate. Let us assume that (a_1, b_1, c') queried the set $X(a_1, b_1, c)$. It immediately gives us $c' = c$, and thus the point (a', b', c') is actually the point (a_1, b_1, c), which indeed is a member of $\mathcal{S}$. We can similarly argue if other sets are queried.

Case II – We now assume that three members of $\mathcal{S}$ are in one xy-plane, and the other member is in a different plane. Let the three members in the same plane be the first three members of $\mathcal{S}$. So, we have $c_1 = c_2 = c_3 = c$(say), and $c_4 \neq c$. This scenario gives rise to two different arrangement of the elements that have to be handled differently.

Case II(A) – Let us assume that in partition $\mathcal{Z}$, the element (a_4, b_4, c_4) lies in one of the sets of the other three elements of $\mathcal{S}$. Without loss of generality, let that element be (a_1, b_1, c). From Observation 4.1, if $(a_4, b_4, c_4) \in Z(a_1, b_1, c)$ then $Z(a_4, b_4, c_4) = Z(a_1, b_1, c) = Z(a_1, b_1, 0)$. Thus, we have $a_1 = a_4$ and $b_1 = b_4$. So, the four elements of $\mathcal{S}$ are $\{(a_1, b_1, c), (a_2, b_2, c), (a_3, b_3, c), (a_1, b_1, c_4)\}$. If that is the case, we set the bits corresponding to each member of $\mathcal{S}$ in tables $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ to 1. The rest of the bits in all of the tables, including $\mathcal{D}$, are set to 0.

If an element (a', b', c') got a majority of its query answers as 1, it must get those from tables $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$. In table $\mathcal{Z}$, it can either query the bit $Z(a_1, b_1, 0)$ or any of the

other two sets that are set to 1. If it queries $Z(a_1, b_1, 0)$, we have $a' = a_1$ and $b' = b_1$. In table $\mathcal{X}$, it will query such a set whose y-coordinate is b_1 and which is set to 1. Two such sets are $X(0, b_1, c)$ and $X(0, b_1, c_4)$. If $b_1 = b_2$, then $X(0, b_2, c_2)$ is another candidate. If our element queries $X(0, b_1, c_1)$, then we have $c' = c$, and thus $(a', b', c') = (a_1, b_1, c)$ which is a member of $\mathcal{S}$. We can similarly argue the other cases.

If instead of querying the set $Z(a_1, b_1, 0)$ in table $\mathcal{Z}$, if (a', b', c') queries any of the other sets that are set to 1, say $Z(a_2, b_2, 0)$, we can similarly argue that the element (a', b', c') will again be a member of $\mathcal{S}$. We leave the details in such cases to the reader.

Case II(B) – We now consider the scenario when the element (a_4, b_4, c_4) does not lie in any of the sets of the other three elements in partition $\mathcal{Z}$. Then for the element (a_4, b_4, c_4), we set the bits $X(0, b_4, c_4)$ and $Y(a_4, 0, c_4)$ in tables $\mathcal{X}$ and $\mathcal{Y}$, respectively, to 1. Also in table $\mathcal{D}$, we set one of the bits $D(a_4 - b_4, 0, c_4)$ or $D(0, b_4 - a_4, c_4)$ to 1, according as a_4 is greater than or less than b_4. Without loss of generality, let us assume that $a_4 \geq b_4$. For the other three elements, we set their bits in tables $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ to 1. All the other bits in all of the tables are set to 0.

Consider an element (a', b', c') which got majority of its queries as 1. The z-coordinate of the point could either be equal to c, or be equal to c_4, or it could be distinct from both c and c_4. We consider each of these cases separately.

Case II(B)(i) – Let c' be different from c and c_4. Then, its queries into the tables $\mathcal{X}$ and $\mathcal{Y}$ must return 0. This is due to the fact that the z-coordinates of all the bits that are set to 1 in tables $\mathcal{X}$ and $\mathcal{Y}$ are either c or c_4. To take an example, if it queried the bit $X(0, b_1, c)$ and hence got a 1, then $c' = c$, which contradicts our assumption. So, in this scenario, the element (a', b', c') cannot get more than two 1s, and hence no such element can get a `Yes` answer.

Case II(B)(ii) – Let c' be equal to c_4. To get a majority of its answer as 1, it must get a 1 from one of the tables $\mathcal{X}$ and $\mathcal{Y}$. Let it be the table $\mathcal{X}$. We want such a set of $\mathcal{X}$ which has its z-coordinate equal to c_4 and is set to 1. The only set satisfying the constraints is $X(0, b_4, c_4)$, which gives us $b' = b_4$.

We now look at query it made in table $\mathcal{D}$. If it returned a 0, then for the sake of

majority, its query into table $\mathcal{Y}$ must return 1. The set whose z-coordinate is c_4 and is set to 1 in this table is $Y(a_4, 0, c_4)$, and hence $a' = a_4$. So, we have $(a', b', c') = (a_4, b_4, c_4)$, a member of $\mathcal{S}$. If the query made in table $\mathcal{D}$ returned a 1, then it must be the case that $a' - (a_4 - b_4) = b' - 0 = b_4 - 0$, which implies that $a' = a_4$. So, we have $(a', b', c') = (a_4, b_4, c_4)$, a member of $\mathcal{S}$. Other cases similarly follows.

Case II(B)(iii) – The final case we look into is when $c' = c$. As all bits that are set to 1 in table $\mathcal{D}$ has the z-coordinate equal to c_4, the element (a', b', c) must get a 0 upon query in this table. So, the query answers from all the other tables must be 1. In table $\mathcal{Z}$, there are three sets whose corresponding bits are set to 1. If the element queries the set $Z(a_1, b_1, 0)$, then we have $a' = a_1$ and $b' = b_1$. So, the element (a', b', c') must be (a_1, b_1, c), a member of $\mathcal{S}$. We can argue the other cases similarly.

Case III – We now consider the scenario when at most two members of $\mathcal{S}$ belong to the same xy-plane. This case is much simpler that the previous ones – we set the bits corresponding to the members of $\mathcal{S}$ in tables $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{D}$ to 1, and the rest of the bits, including those of table $\mathcal{Z}$ to 0.

The proof is this case is also similar to the proofs done in the previous cases. The only thing that we have to consider is when an xy-plane contains two elements, and when all the elements are on a separate xy-plane. As a demonstration we prove one such case next, and leave the rest of the cases to the reader.

Let the first two elements of $\mathcal{S}$, namely (a_1, b_1, c_1) and (a_2, b_2, c_2), belong to the same xy-plane. We also assume that $a_1 \geq b_1$ and $a_2 \geq b_2$. This implies in the bits $D(a_1 - b_1, 0, c_1)$ and $D(a_2 - b_2, 0, c_2)$ are set to 1. Consider the element (a', b', c') which also belongs to this plane. We prove that in this scenario, if the element (a', b', c') upon query in our data structure got a `Yes`, then it must be a member of $\mathcal{S}$. From our assumptions, we have $c_1 = c_2 = c' = c$(say). As all the bits of table $\mathcal{Z}$ has its bits set to 0, (a', b', c) must get 1 from rest of the tables.

In table $\mathcal{X}$, the only bits set to 1 and with z-coordinate equal to c is $X(0, b_1, c)$ and $X(0, b_2, c)$. Let the element query the set $X(0, b_1, c)$. This gives us $b' = b_1$, and hence the element in question is (a', b_1, c). The bits of table $\mathcal{Y}$ that are set to 1 and have the z-coordinate c are $Y(a_1, 0, c)$ and $Y(a_2, 0, c)$, and our element must have queried one

of these sets. So, the element (a', b', c') could be one of (a_1, b_1, c) and (a_2, b_1, c). If the element is (a_1, b_1, c), it is already a member of $\mathcal{S}$ and we have nothing to prove.

We now consider the case of $(a', b', c') = (a_2, b_1, c)$. The bits of table $\mathcal{D}$ that are set to 1 with z-coordinate c are $D(a_1 - b_1, 0, c)$ and $D(a_2 - b_2, 0, c)$. This tells us that if the element (a', b', c') is actually the element (a_1, b_2, c), then either $a_1 = a_2$ or $b_1 = b_2$. In both of these cases, the element is a member of $\mathcal{S}$.

We now conclude the description of our storage scheme and the proof of correctness. The following theorem summarises the result of this section.

Theorem 4.7. *There is an explicit non-adaptive* $(4, m, 5 \times m^{2/3}, 4)$ *scheme.*

4.4 A $(n = 5, t = 4)$-Non-Adaptive Scheme

In this section, we present a non-adaptive scheme that improves upon the scheme of the previous section. The scheme presented can store five elements ($n = 5$), uses four non-adaptive bitprobes ($t = 4$), and takes up $\mathcal{O}(m^{2/3})$ bits of storage. This scheme is a departure from the approach taken in the previous scheme in that it uses and extends the idea presented by Radhakrishnan *et al.* [2] as part of theorem 2, that of dividing the universe into blocks and superblocks.

4.4.1 Our Data structure

As alluded to earlier, our scheme is based on the scheme described in theorem 2 of Radhakrishnan *et al.* [2], which is an adaptive scheme for storing two elements using two adaptive bitprobes. Their data structure consists of three tables T, T_0, and T_1, each of size $m^{2/3}$ bits.

In our data structure, we have four tables instead – the three tables of Radhakrishnan *et al.* [2], and one more table to differentiate those blocks that contain one or more elements of $\mathcal{S}$ from those which doesn't. For the sake of completeness, we describe the arrangement of elements in the three tables T, T_0, and T_1 along with our additional table.

Every element in our universe $\mathcal{U}$ is given a unique tuple (u, v) as its label, where u, v are non-negative integers. If the size of our universe is m, then u ranges over 0 and

$m^{2/3} - 1$, and v ranges over 0 and $m^{1/3} - 1$.

Our data structure, as mentioned earlier, consists of four tables – $\mathcal{A}, \mathcal{B}, \mathcal{C}$, and $\mathcal{D}$. Consider the elements $(u, v), (u_1, v_1)$ and (u_2, v_2) of our universe $\mathcal{U}$. Let $u = qm^{1/3} + r, u_1 = q_1 m^{1/3} + r_1$, and $u_2 = q_2 m^{1/3} + r_2$, where $0 \leq r, r_1, r_2 < m^{1/3}$. We now describe the criteria for any two elements of $\mathcal{U}$ to query the same location in a table of our data structure.

In table $\mathcal{A}$, two elements (u_1, v_1) and (u_2, v_2) query the same bit if and only if $u_1 = u_2$. So, the size of table $\mathcal{A}$ is $m^{2/3}$, one bit for every unique value of the first component of the tuple. For an element (u, v), we will use the notation $\mathcal{A}(u)$ to denote the location the element queries.

In table $\mathcal{B}$, elements (u_1, v_1) and (u_2, v_2) will map to the same location if and only if $q_1 = q_2$ and $v_1 = v_2$. So, this table will have a unique bit for every pair of values of the quotient and the second component of the tuple. Hence, the size of table $\mathcal{B}$ is $m^{1/3} \times m^{1/3} = m^{2/3}$. The location where the element (u, v) queries in this table will be denoted by $\mathcal{B}(q, v)$.

As for table $\mathcal{C}$, the two elements will map to the same location if and only if $r_1 = r_2$ and $v_1 = v_2$. It is easy to see that in this case too the size of the table is $m^{1/3} \times m^{1/3} = m^{2/3}$. The location queried by (u, v) here will be denoted by $\mathcal{C}(r, v)$.

The structure of table $\mathcal{D}$ is exactly same as that of table $\mathcal{A}$, i.e. any two elements of the universe will query the same bit in this table if and only if their first components are identical. It follows that the size of the table is $m^{2/3}$. As was done for table $\mathcal{A}$, the location queried by an element (u, v) will be denoted by $\mathcal{D}(u)$.

We conclude our discussion about the data structure with the following lemma.

Lemma 4.8. *The size of our data structure is* $4 \times m^{2/3}$.

4.4.2 The Query Scheme

Let (u, v) be an element of our universe $\mathcal{U}$, where u is expressed as $qm^{1/3}+r, 0 \leq r < m^{1/3}$. For this element, our query scheme will query the following locations in our data structure – $\mathcal{A}(u), \mathcal{B}(q, v), \mathcal{C}(r, v)$, and $\mathcal{D}(u)$. The hierarchical structure of the four tables is depicted in Figure 4.3

For an element (u, v) the query scheme decides its membership in the following way.

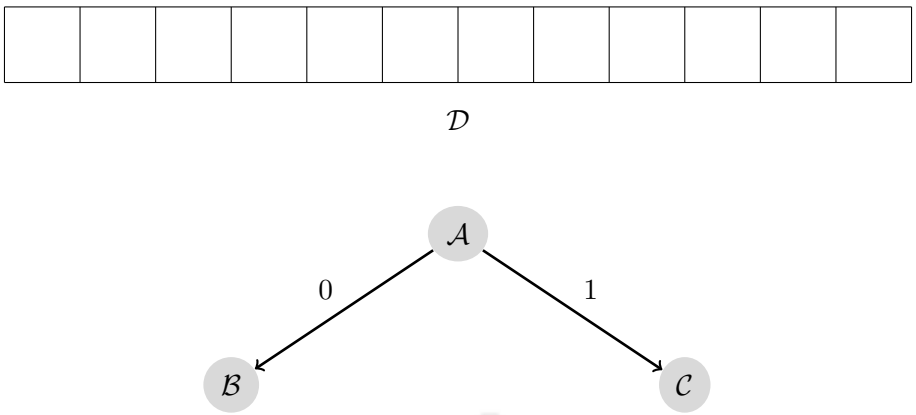


Figure 4.3: The hierarchy of tables in the non-adaptive bitprobe scheme.

If $\mathcal{D}(u)$ is 0, then no matter what is stored in the other tables, the scheme will return `No`. Let us now consider what happens when $\mathcal{D}(u)$ is 1. If $\mathcal{A}(u)$ is 0 and $\mathcal{B}(q, v)$ is 1, then no matter what is stored at $\mathcal{C}(r, v)$, the scheme returns `Yes`. On the other hand, if $\mathcal{A}(u)$ is 1 and $\mathcal{C}(r, v)$ is 1, then no matter what is stored at $\mathcal{B}(q, v)$, the scheme also returns `Yes`. In all other cases, the scheme returns 0.

Hence, our query scheme will declare the element to be a member of the set $\mathcal{S}$ if the bits stored in the corresponding locations of our data structure conform to one of the following two patterns.

Pattern 1 $\mathcal{A}(u)$ is 0, $\mathcal{B}(q, v)$ is 1, and $\mathcal{D}(u)$ is 1; we do not care what bit is stored at $\mathcal{C}(r, v)$.

Pattern 2 $\mathcal{A}(u)$ is 1, $\mathcal{C}(r, v)$ is 1, and $\mathcal{D}(u)$ is 1; in this case, the bit stored at $\mathcal{B}(q, v)$ has no bearing on the final answer.

4.4.3 The Storage Scheme

As was done for the previous schemes, in this section we will describe our storage scheme and provide the proof of correctness alongside it.

For this scheme, the size of our subset $\mathcal{S}$ is at most five. Let its members be denoted by $(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4)$, and (u_5, v_5). We further express each u_i as $q_i m^{1/3} + r_i$, where r_i lies between 0 and $m^{1/3} - 1$, and i ranges between 1 and 5. Also, let (u, v) be an arbitrary element of $\mathcal{U}$, where $u = qm^{1/3} + r, 0 \leq r < m^{1/3}$.

As before, our discussion will be broken up into several cases, each case being characterised by the nature of the members of $\mathcal{S}$.

Case I – Let us first consider the scenario when all of the q_is, $1 \leq i \leq 5$, are equal. We set in table $\mathcal{A}$, $\mathcal{A}(u_i) = 1$ for all i. The rest of the bits of $\mathcal{A}$ could be set in any way we choose. All of the bits of table $\mathcal{B}$ are set to 0. In table $\mathcal{C}$, $\mathcal{C}(r_i, v_i)$ is set to 1, $1 \leq i \leq 5$, and the rest of the bits are set to 0. Finally, in table $\mathcal{D}$, $\mathcal{D}(u_i)$ is set to 1 for $1 \leq i \leq 5$, and the rest of its bits are set to 0.

We now proceed to argue that this storage scheme is indeed correct. The assignment to the four tables tell us that the bits corresponding to the members of $\mathcal{S}$ will conform to pattern 2, and hence the query scheme will return `Yes`.

Consider an element $(u, v) \in \mathcal{U}$ which upon query in our data structure was decided to be a member of $\mathcal{S}$. As mentioned above, $u = qm^{1/3} + r$, where $0 \leq r < m^{1/3}$.

In this scenario, it must be the case that $\mathcal{D}(u) = 1$. As only the bits corresponding to members of $\mathcal{S}$ are set to 1 in table $\mathcal{D}$, we have $u = u_i$ for some i. Without loss of generality, let $u = u_1$, and hence $q = q_1$ and $r = r_1$.

As $\mathcal{A}(u) = \mathcal{A}(u_1) = 1$, the only way it could be deduced that (u, v) is a member of $\mathcal{S}$ is if its bits in the four tables are set according to pattern 2, i.e. if its location in $\mathcal{C}$ is set to 1. One way $\mathcal{C}(r, v) = \mathcal{C}(r_1, v)$ could be 1 is $v = v_1$. So, (u, v) is the same as the element (u_1, v_1), a member of $\mathcal{S}$. It could also happen that $r_1 = r_2$, and $\mathcal{C}(r_2, v_2)$ is 1, and (u, v) queries this location and got a 1. In this case, we will have $r = r_1 = r_2$ and $v = v_2$, and hence the element (u, v) is actually the element (u_2, v_2). One can similarly verify all the other possibilities.

Case II – Let us now consider the case when four of the five q_is are equal. Without loss of generality, let those four elements be the first four elements of $\mathcal{S}$, and q_5 is different from the rest.

In table $\mathcal{A}$, $\mathcal{A}(u_i) = 1$ for $1 \leq i \leq 4$, and $\mathcal{A}(u_5) = 0$. The rest of its bits could be set in any way we like. In table $\mathcal{B}$, $\mathcal{B}(q_5, v_5) = 1$, and the rest of the its bits are set to 0. In table $\mathcal{C}$, $\mathcal{C}(r_i, v_i) = 1$ for $1 \leq i \leq 4$, and the remaining bits are equal to 0. In table $\mathcal{D}$, $\mathcal{D}(u_i) = 1$ for $1 \leq i \leq 5$. The rest of the bits in this table are set to 0.

The above assignment tells us that the bits stored for the first four members of $\mathcal{S}$ will conform to pattern 2, and the bits for the fifth member will conform to pattern 1. So, the query scheme will correctly answer for the members of $\mathcal{S}$.

If the arbitrary element $(u, v) \in \mathcal{U}$ is to be declared a member of $\mathcal{S}$, its bit in table $\mathcal{D}$ must be 1. So, its first component, u, could be either one of the first four elements, or it could be equal to u_5. We consider the two cases separately.

Case II(A) – Let it be the case that $u = u_5$, and hence $q = q_5$. Then, it follows that $\mathcal{A}(u) = \mathcal{A}(u_5) = 0$, which can only mean that the bits corresponding to (u, v) should be set according to pattern 1. More specifically, $\mathcal{B}(q, v)$ should be equal to 1. The only bit in table $\mathcal{B}$ set to 1 is $\mathcal{B}(q_5, v_5)$, and thus we have $v = v_5$. So, we have $(u, v) \in \mathcal{S}$.

Case II(B) – We now consider the scenario where u is equal to one of $u_1, \ldots, u_4$. Without loss of generality, let $u = u_1$, and thus $q = q_1$ and $r = r_1$. As both $\mathcal{A}(u) = \mathcal{A}(u_1)$ and $\mathcal{D}(u) = \mathcal{D}(u_1)$ are set to 1, the bits of (u, v) must follow pattern 2. So, it must be the case that $\mathcal{C}(r, v) = \mathcal{C}(r_1, v) = 1$. One way this is possible is if $v = v_1$, in which case $(u, v) = (u_1, v_1) \in \mathcal{S}$. Another way it could be possible is if $r_1 = r_2$, and $v = v_2$. In that case, $(u, v) = (u_2, v_2) \in \mathcal{S}$. All the remaining cases can be handled similarly.

Case III – The next scenario to consider is the one in which three of the q_is are equal, and the remaining two are different from these. Without loss of generality, let us assume that $q_1 = q_2 = q_3$. In this case, two situations might arise depending on the values of q_4 and q_5 – one in which $q_4 = q_5$, and the other in which $q_4 \neq q_5$. We handle the easier of these two cases first.

Case III(A) – We first discuss the case where $q_1 = q_2 = q_3$, and q_4 and q_5 are distinct from everybody else. In table $\mathcal{A}$, $\mathcal{A}(u_1) = \mathcal{A}(u_2) = \mathcal{A}(u_3) = 1$, and $\mathcal{A}(u_4) = \mathcal{A}(u_5) = 0$. The rest of its bits could be set in any way whatsoever. In table $\mathcal{B}$, $\mathcal{B}(q_4, v_4) = \mathcal{B}(q_5, v_5) = 1$, and the rest of the bits are 0. In table $\mathcal{C}$, $\mathcal{C}(r_1, v_1) = \mathcal{C}(r_2, v_2) = \mathcal{C}(r_3, v_3) = 1$, and the remaining bits are set to 0. The only bits of table $\mathcal{D}$ that are set to 1 are those corresponding the the members of $\mathcal{S}$.

From this assignment, the bits of the first three elements of $\mathcal{S}$ will conform to pattern 2, and the bits of the last two elements will conform to pattern 1. So, the query scheme correctly works for the members of $\mathcal{S}$.

An element (u, v) will be declared to be a member of $\mathcal{S}$ if its bits are set in any one

of the two allowed bit patterns. Let us consider the two scenarios, one at a time.

Let the bits corresponding to the element (u, v) be set according to that in pattern 1. It follows that its bit in table $\mathcal{D}$ is 1 and in table $\mathcal{A}$ is 0. Thus, u could be one of u_4 and u_5. Without loss of generality, let $u = u_4$, implying $q = q_4$. The only bit set to 1 in table $\mathcal{B}$ that matches this scenario is $\mathcal{B}(q_4, v_4)$. So, we have $v = v_4$, and hence $(u, v) = (u_4, v_4) \in \mathcal{S}$.

Let us now consider the scenario where the bits of (u, v) are set according to pattern 2. Similar to the case above, we can deduce by looking at tables $\mathcal{A}$ and $\mathcal{D}$ that u must be one of u_1, u_2, and u_3. Without loss of generality, let $u = u_1$, and hence $q = q_1$ and $r = r_1$. One way it is possible that $\mathcal{C}(r, v) = \mathcal{C}(r_1, v) = 1$ is when $v = v_1$. In that case, $(u, v) = (u_1, v_1) \in \mathcal{S}$. We could also have $r = r_1 = r_2$ and (u, v) queries $\mathcal{C}(r_2, v_2)$. If that is the case, then we have $u = u_2$ and $v = v_2$, and (u, v) is actually the element (u_2, v_2), a member of $\mathcal{S}$. All other cases can be handled similarly.

Case III(B) – Let us now discuss what happens when we have $q_1 = q_2 = q_3$, and $q_4 = q_5$, but $q_1 \neq q_4$. For the fourth and the fifth element, we set $\mathcal{A}(u_4) = 1$ and $\mathcal{A}(u_5) = 0$.

Here too, two subcases arise depending on whether or not r_4 is equal to one of r_1, r_2 and r_3. We handle them separately below.

Case III(B)(a) – Let it be the case that r_4 is distinct from all of r_1, r_2, and r_3. Then the bits of the first three elements in table $\mathcal{A}$ are set thus – $\mathcal{A}(u_1) = \mathcal{A}(u_2) = (u_3) = 1$. The rest of the bits of table $\mathcal{A}$ can be set in any way whatsoever. In table $\mathcal{B}$, only the bit $\mathcal{B}(q_5, v_5)$ is set to 1, and all of the other bits to 0. In table $\mathcal{C}$, the bits corresponding to the first four elements, namely $\mathcal{C}(r_i, v_i)$ where $1 \leq i \leq 4$, are set to 1 and the rest to 0. In table $\mathcal{D}$, only the bits corresponding to the five members of $\mathcal{S}$ are set to 1, and all the rest are set to 0.

The way the bits are set in the four tables tell us that the first four elements will conform to bit pattern 2, and the fifth element will conform to bit pattern 1. So, the query scheme will answer `Yes` for all the members of $\mathcal{S}$.

If an element (u, v) matches the bit pattern 1, then it must be the case that $\mathcal{A}(u) = 0$ and $\mathcal{D}(u) = 1$. It is only possible if $u = u_5$. The only bit that is set to 1 in table $\mathcal{B}$ is

$\mathcal{B}(q_5, v_5)$, and hence we have $v = v_5$. So, the element (u, v) is actually the same as the element (u_5, v_5), a member of $\mathcal{S}$.

On the other hand, if the element (u, v)'s bit pattern is that of pattern 2, then it must be the case that u is equal to one of u_1, u_2, u_3, and u_4. If $u = u_4$, then, combined with the fact that q_4 is distinct from q_1, q_2, and q_3, and r_4 is distinct from r_1, r_2, and r_3, the only way $\mathcal{C}(r, v) = \mathcal{C}(r_4, v)$ can be 1 is if $v = v_4$, which makes (u, v) equal to (u_4, v_4).

On the other hand, if $u = u_1$, one way for $\mathcal{C}(r, v) = \mathcal{C}(r_1, v)$ to be 1 is if $v = v_1$. It can also happen that $r = r_1 = r_2$, and $\mathcal{C}(r, v)$ queries the bit $\mathcal{C}(r_2, v_2)$. In that case, we have $q = q_1 = q_2$, $r = r_1 = r_2$, and $v = v_2$. So, we have $u = u_2$ and thus the element (u, v) is the same as (u_2, v_2). In all of the other cases that might arise, we can similarly show that (u, v) is always a member of $\mathcal{S}$.

Case III(B)(b) – We now handle the case where r_4 is equal to one of r_1, r_2, and r_3. Without loss of generality, let $r_4 = r_1$.

The discussion that follows implicitly assumes that $r_1 \neq r_2, r_3$, and thus r_4 is distinct from r_2 and r_3. If that is not the case, and, say, $r_1 = r_2$, then, combined with the fact that $q_1 = q_2$, we have $u_1 = u_2$. In this case, the bits of (u_2, v_2) will be set in the same way as that of (u_1, v_1) (described below), and the argument will follow along the lines laid out below.

In table $\mathcal{A}$, $\mathcal{A}(u_1)$ is set to 0, and $\mathcal{A}(u_2) = \mathcal{A}(u_3) = 1$. We have already set the $\mathcal{A}(u_4)$ to 1, and $\mathcal{A}(u_5)$ to 0. The other bits can be set in any way we choose. In table $\mathcal{B}$, $\mathcal{B}(q_1, v_1)$ and $\mathcal{B}(q_5, v_5)$ are set to 1, and all of the other bits are set to 0. In table $\mathcal{C}$, we have $\mathcal{C}(r_2, v_2) = \mathcal{C}(r_3, v_3) = \mathcal{C}(r_4, v_4) = 1$, and the remaining bits are 0. In table $\mathcal{D}$, only the bits corresponding to the members of $\mathcal{S}$ are set to 1.

In this case, the bits are set in such a way that those for the first and fifth element conform to pattern 1, and those for the other three elements conform to pattern 2. So the query scheme works correctly for the elements of $\mathcal{S}$.

If the bit pattern of an element (u, v) matches that of pattern 1, then it must be that case that u is either u_1 or u_5. As $q_1 \neq q_5$, it cannot be the case that u is equal to both u_1 and u_5. If $u = u_1$, implying $q = q_1 \neq q_5$, $\mathcal{B}(q, v)$ can only be 1 if $v = v_1$. So, we have $(u, v) = (u_1, v_1) \in \mathcal{S}$. The case where $u = u_5$ can be similarly argued.

We now consider the possibility that the bits of (u, v) matches that of pattern 2,

and thus u is equal to one of u_2, u_3, and u_4. If $u = u_4$, we have $q = q_4 \neq q_2, q_3$, and $r = r_4 \neq r_2, r_3$. In this scenario, the only way $\mathcal{C}(r, v)$ can be 1 is if $v = v_4$, which implies that $(u, v) = (u_4, v_4) \in \mathcal{S}$. If $u = u_2$, we have $q = q_2 = q_3$ and $r = r_2 \neq r_4$. One way for $\mathcal{C}(r, v) = \mathcal{C}(r_2, v_2)$ to be 1 is if $v = v_2$. It could also be the case that $r = r_2 = r_3$. In that case, we have $u = u_2 = u_3$. So, (u, v) could also query $\mathcal{C}(r_3, v_3)$ which means that $v = v_3$ and thus $(u, v) = (u_3, v_3) \in \mathcal{S}$. Any other case that might arise can be similarly handled.

Case IV – Next, we handle the case where two of the q_is are equal. We need to discuss the scenarios when one pair of q_is are equal, and when two pairs of q_is are equal, separately.

Case IV(A) – We will discuss the easier of the two scenarios first, that of one pair of q_is being equal. Without loss of generality, let it be the case that $q_1 = q_2$, and that the other q_is are distinct from them, and distinct from each other.

In such a case, we have in table $\mathcal{A}$, $\mathcal{A}(u_1) = \mathcal{A}(u_2) = 1$, and $\mathcal{A}(u_3) = \mathcal{A}(u_4) = \mathcal{A}(u_5) = 0$. In table $\mathcal{B}$, we set $\mathcal{B}(q_3, v_3), \mathcal{B}(q_4, v_4)$, and $\mathcal{B}(q_5, v_5)$ to 1, and the rest of the bits to 0. In table $\mathcal{C}$, we set $\mathcal{C}(r_1, v_1)$ and $\mathcal{C}(r_2, v_2)$ to 1 and the remaining bits to 0. In table $\mathcal{D}$, we set only the bits corresponding to the members of $\mathcal{S}$ to 1.

In the assignment above, we see that the bits are so set that the first and the second elements of $\mathcal{S}$ conform to pattern 2 and the remaining elements conform to pattern 1. Hence, the query scheme works correctly for the elements of $\mathcal{S}$.

An element (u, v) of $\mathcal{U}$ can match either of the two patterns and be declared to be a member of $\mathcal{S}$. If it matches pattern 1, then u has to be one of u_3, u_4, and u_5. Without loss of generality, let it be u_3. This implies that $q = q_3$. Combined with the fact that q_3 is distinct from all other q_is, the only way for $\mathcal{B}(q, v) = \mathcal{B}(q_3, v)$ to be 1 is if $v = v_3$, which results in $(u, v) = (u_3, v_3) \in \mathcal{S}$.

On the other hand, if the bits of (u, v) match pattern 2, then u could be one of u_1 and u_2. Without loss of generality, let it be u_1, which implies that $r = r_1$. In table $\mathcal{C}$, one way for $\mathcal{C}(r, v) = \mathcal{C}(r_1, v)$ to be 1 is if $v = v_1$, which implies that $(u, v) = (u_1, v_1) \in \mathcal{S}$. It could also be the case that $r = r_1 = r_2$, and (u, v) queries the bit $\mathcal{C}(r_2, v_2)$. This combined with the fact that $q = q_1 = q_2$ gives us $u = u_2$ and $v = v_2$. So, $(u, v) = (u_2, v_2) \in \mathcal{S}$.

Case IV(B) – Let us now discuss the scenario when two pairs of q_is are equal, but the pairs are not equal to each other. Without loss of generality, let is be the case that $q_1 = q_2$, $q_3 = q_4$, and $q_1 \neq q_3$. Moreover, q_5 is distinct from everybody else. In this case, we need to consider two scenarios depending on whether or not r_1 is equal to one of r_3, r_4.

Case IV(B)(a) – Let it be the case that r_1 is distinct from both r_3 and r_4. Then the bits of our data structure are set in the following way.

In table $\mathcal{A}$, $\mathcal{A}(u_1, v_1) = \mathcal{A}(u_3, v_3) = \mathcal{A}(u_4, v_4) = 1$, $\mathcal{A}(u_2, v_2) = \mathcal{A}(u_5, v_5) = 0$, and the rest of the bits are set in any way we choose. In table $\mathcal{B}$, $\mathcal{B}(q_2, v_2) = \mathcal{B}(q_5, v_5) = 1$, and the rest of the bits are 0. In table $\mathcal{C}$, $\mathcal{C}(r_1, v_1) = \mathcal{C}(r_3, v_3) = \mathcal{C}(r_4, v_4) = 1$, and the rest of the bits are set to 0. As usual, in table $\mathcal{D}$, only the bits corresponding to the members of $\mathcal{S}$ are set to 1.

In this case, the bits corresponding to the second and fifth elements of $\mathcal{S}$ conform to pattern 1, and those corresponding to the first, third, and fourth elements conform to pattern 2. Hence, the query scheme returns `Yes` for the members of $\mathcal{S}$.

If an element (u, v)'s bits match that of pattern 1, then it must be the case that u is one of u_2, u_5. If $u = u_2$, implying that $q = q_2$, and combined with the fact that $q_2 \neq q_5$, the only way $\mathcal{B}(q, v) = \mathcal{B}(q_2, v)$ can be 1 is if $v = v_2$, which implies that $(u, v) = (u_2, v_2) \in \mathcal{S}$. The case where $u = u_5$ can be argued similarly.

If the bits of (u, v) match that of pattern 2, then u can only be one of u_1, u_3, u_4. If $u = u_1$, implying that $r = r_1$, and combined with the fact that r_1 is distinct from r_3, r_4, the only way $\mathcal{C}(r, v) = \mathcal{C}(r_1, v)$ can be 1 is if $v = v_1$, which implies that $(u, v) = (u_1, v_1) \in \mathcal{S}$. If, on the other hand, we have $u = u_3$, and hence $r = r_3$, one way for $\mathcal{C}(r, v) = \mathcal{C}(r_3, v)$ to be 1 is if $v = v_3$, which gives us $(u, v) = (u_3, v_3) \in \mathcal{S}$. It could also be the case that $r = r_3 = r_4$, and (u, v) queried $\mathcal{C}(r_4, v_4)$. In that case, as $q_3 = q_4$, we have $u = u_3 = u_4$, and $v = v_4$, which gives us $(u, v) = (u_4, v_4) \in \mathcal{S}$.

Case IV(B)(b) – We now describe the case when r_1 is equal to one of r_3 and r_4. Without loss of generality, let it be the case that $r_1 = r_3$.

Similar to one of the cases before, in the following discussion we implicitly assume that $r_3 \neq r_4$. If that is not the case, we will have $u_3 = u_4$, and the bits of (u_4, v_4) will

be set in the same way as that of (u_3, v_3), and the correctness can be argued along the lines laid out below.

In table $\mathcal{A}$, $\mathcal{A}(u_1) = \mathcal{A}(u_4) = 1$, $\mathcal{A}(u_2) = \mathcal{A}(u_3) = \mathcal{A}(u_5) = 0$, and the rest of the bits can be set in any way we like. In table $\mathcal{B}$, $\mathcal{B}(q_2, v_2) = \mathcal{B}(q_3, v_3) = \mathcal{B}(q_5, v_5) = 1$, and the rest of the bits are set to 0. In table $\mathcal{C}$, $\mathcal{C}(r_1, v_1) = \mathcal{C}(r_4, v_4) = 1$, and the rest of the bits are set to 0. As before, only the bits of the members of $\mathcal{S}$ are set to 1 in table $\mathcal{D}$.

As can be seen in the assignment above, the bits corresponding to the first and the fourth element conform to pattern 2, and those corresponding to the remaining elements conform to pattern 1. So, the query for the elements of $\mathcal{S}$ are handled correctly.

If the bits of an element $(u, v) \in \mathcal{U}$ is set according to pattern 1, then it must be the case that u is one of u_2, u_3, u_5. If $u = u_2$, implying $q = q_2$, and combined with the fact that q_2 is distinct from q_3, q_5, the only way $\mathcal{B}(q, v) = \mathcal{B}(q_2, v)$ can be 1 is if $v = v_2$, which implies that $(u, v) = (u_2, v_2) \in \mathcal{S}$. The other cases can be handled similarly.

On the other hand, if the bits of (u, v) are according to pattern 2, then u can only be one of u_1, u_4. If $u = u_1$, implying $r = r_1$, and combined with the fact that $r_1 \neq r_4$, the only way $\mathcal{C}(r, v) = \mathcal{C}(r_1, v)$ can be 1 is if $v = v_1$, which implies that $(u, v) = (u_1, v_1) \in \mathcal{S}$. The case of $u = u_4$ can be argued similarly.

Case V – We come to the final case of our discussion, one in which all of the q_is are distinct. In this case, in table $\mathcal{A}$, all of the bits of the members of $\mathcal{S}$ are set to 0, the rest of the bits to anything we see fit. In table $\mathcal{B}$, the bits of the members of $\mathcal{S}$ are set to 1, and the rest of its bits to 0. In table $\mathcal{C}$, all of the bits are set to 0. Finally, in table $\mathcal{D}$, the bits of the members of $\mathcal{S}$ are set to 1, and the rest to 0.

In this case, the bits corresponding to the members of $\mathcal{S}$ are set to 0 in table $\mathcal{A}$, and those in table $\mathcal{B}$ are set to 1. Their bits in table $\mathcal{D}$ are also set to 1. This means that their bits are so set that they conform to pattern 1, and hence the query scheme works correctly for them.

It also follows that for an element to be declared a member of $\mathcal{S}$, its bits must conform to pattern 1. So, it must be the case that $\mathcal{A}(u) = 0$ and $\mathcal{D}(u) = 1$. This is true for only the members of $\mathcal{S}$. Without loss generality, let $u = u_1$, and hence $q = q_1$. As all the q_is of the members of $\mathcal{S}$ are distinct, the only way $\mathcal{B}(q, v) = \mathcal{B}(q_1, v)$ be equal to 1 is if $v = v_1$. So, the element (u, v) is actually the element (u_1, v_1), an element of $\mathcal{S}$.

This concludes the description of our storage scheme and its proof of correctness. The following theorem summarises the result of this section.

Theorem 4.9. *There is an explicit non-adaptive* $(5, m, 4 \times m^{2/3}, 4)$ *scheme.*

4.5 Conclusion

In this chapter, we have used the geometrical technique by Kesh [18] to visualizing the arrangement of elements by placing them on the integral points of a suitably sized three-dimensional cube. This gives us, what essentially is, three new results, one in the domain of adaptive bitprobe model, and two in the non-adaptive model. This technique can be extended to higher-dimensional cubes and has already given interesting results in the two queries adaptive bitprobe model (Kesh [18]). Further, we have used the idea of Radhakrishnan *et al.* [2] to divide the universe $\mathcal{U}$ of size m into blocks and superblocks. Using this idea, we have improved our non-adaptive scheme to store a subset $\mathcal{S}$ of larger size and still using the same number of bitprobes and space. We believe that the combination of the above-mentioned technique will give improved results in several other scenarios in both the adaptive and non-adaptive model.

I only get frightened — and it happens very rarely — when I think I have an idea.

–J. Robert Oppenheimer, *Interview with R. Murrow*, 1955

5

Explicit Adaptive Two Bitprobe Scheme Storing Four Elements

5.1 Introduction

We studied an explicit adaptive two bitprobe ($t = 2$) scheme which stores an arbitrary subset $\mathcal{S}$ of size three ($n = 3$) from a universe $\mathcal{U}$ of size m, and uses $\mathcal{O}(m^{2/3})$ amount of space. In this chapter, we study a scheme which extends the subset size to four ($n = 4$), i.e., we give an explicit adaptive two bitprobe scheme, which stores an arbitrary subset of size at most four. The solution to the problem requires ideas form Radhakrishnan [2] to divide the universe $\mathcal{U}$ into blocks and superblocks, and from Kesh [18] to map the elements on a two dimensional grid.

5.1.1 Previous Results

In this chapter, we look into adaptive schemes with two bitprobes ($t = 2$). When the subset size is one ($n = 1$), the problem is well understood – the space required by the data structure is $\Omega(m^{1/2})$, and we have a scheme that matches this bound [9, 20].

For subsets of size two ($n = 2$), Radhakrishnan *et al.* [2] proposed a scheme that takes $\mathcal{O}(m^{2/3})$ amount of space, and further conjectured that it is the minimum amount of space required for any scheme. Though progress has been made to prove the conjecture [2, 14], it as yet remains unproven.

For subsets of size three ($n = 3$), Baig and Kesh [3] have recently proposed a scheme that takes $\mathcal{O}(m^{2/3})$ amount of space. It has been subsequently proven by Kesh [17] that $\Omega(m^{2/3})$ is the lower bound for this problem. So, the space complexity question for $n = 3$ stands settled.

5.1.2 Our Contribution

We look into problem where the subset size is four ($n = 4$), i.e, an adaptive bitprobe scheme that can store subsets of size atmost four, and answers membership queries using two bitprobes. Garg and Radhakrishnan [15] have proposed a generalised scheme that can store arbitrary subsets of size $n(< \log m)$, and uses $\mathcal{O}(m^{1-\frac{1}{4n+1}})$ amount of space. For the particular case of $n = 4$, the space requirement turns out to be $\mathcal{O}(m^{16/17})$. Garg [16] further improved the bounds to $\mathcal{O}(m^{1-\frac{1}{4n-1}})$(for $n < (1/4)(\log m)^{1/3}$), which improved the scheme for $n = 4$ to $\mathcal{O}(m^{14/15})$.

We propose a scheme for the problem whose space requirement is $\mathcal{O}(m^{5/6})$ (Theorem 5.5), thus improving upon the existing schemes in the literature. Our claim is the following:

$$s_A(4, m, 2) = \mathcal{O}(m^{5/6}). \text{ (Theorem 5.5)}$$

Furthermore, we will replace the square grid used in Theorem 5.5 with a rectangular grid, this improves the space requirement to $\mathcal{O}(m^{4/5})$. So our claim is the following:

$$s_A(4, m, 2) = \mathcal{O}(m^{4/5}). \text{ (Theorem 5.6)}$$

The existence of such a scheme also answers in the affirmative an open problem posed by Patrick K. Nicholson [5] which asked if a scheme using the idea of blocks due to Radhakrishnan *et al.* [2] exists that stores four elements and answers membership queries using two bitprobes. As the description of our data structure in the following section would show that our scheme extends the ideas of blocks and superblocks using a geometric approach to solve the problem.

Finally, in Section 5.2.11 we provide an instance of a five-element subset of the universe $\mathcal{U}$ which cannot be stored correctly in our data structure, illustrating that a different construction is required to accommodate subsets of larger size.

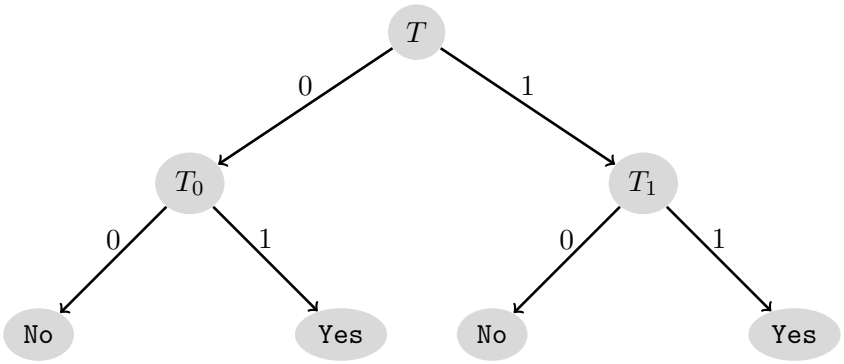


Figure 5.1: The decision tree of an element.

5.1.3 The Bitprobe Model

The scheme presented in this paper is an adaptive scheme that uses two bitprobes to answer membership queries. We now discuss in detail the bitprobe model in the context of two adaptive bitprobes.

The data structure in this model consists of three tables – T, T_0, and T_1 – arranged as shown in Figure 5.1. Any element e in the universe $\mathcal{U}$ has a location in each of these three tables, which are denoted by $T(e), T_0(e)$, and $T_1(e)$. By a little abuse of notation, we will use the same symbols to denote the bits stored in those locations.

Any bitprobe scheme has two components – the *storage* scheme, and the *query* scheme. Given a subset $\mathcal{S}$, the storage scheme sets the bits in the three tables such that the membership queries can be answered correctly. The flow of the query scheme is traditionally captured in a tree structure, called the *decision tree* of the scheme (Figure 5.1). It works as follows. Given a query "Is x in $\mathcal{S}$?", the first bitprobe is made in table T at location $T(x)$. If the bit stored is 0, the second query is made in table T_0, else it is made in table T_1. If the answer received in the second query is 1, then we declare that the element x is a member of $\mathcal{S}$, otherwise we declare that it is not.

5.2 Our Data structure

In this section, we provide a detailed description of our data structure. To achieve a space bound of $o(m)$, more than one element must necessarily share the same location in each of the three tables. We discuss how we arrange the elements of the universe $\mathcal{U}$, and which all elements share the same location in any given table.

Along with the arrangement of elements, we will also talk about the size of our data

structure. The next few sections prove the following theorem.

Theorem 5.1. *The size of our data structure is* $\mathcal{O}(m^{5/6})$.

5.2.1 Table T

Given the universe $\mathcal{U}$ containing m elements, we partition the universe into sets of size $m^{1/6}$. Borrowing the terminology from Radhakrishnan *et al.* [2], we will refer to these sets as *blocks*. It follows that the total number of blocks in our universe is $m^{5/6}$.

The elements within a block are numbered as $1, 2, 3, \ldots, m^{1/6}$. We refer to these numbers as the *index* of an element within a block. So, an element of $\mathcal{U}$ can be addressed by the number of the block to which it belongs, and its index within that block.

In table T of our data structure, we will have one bit for every block in our universe. As there are $m^{5/6}$ blocks, the size of table T is $m^{5/6}$.

5.2.2 Superblocks

The blocks in our universe are partitioned into sets of size $m^{4/6}$. Radhakrishnan *et al.* [2] used the term *superblocks* to refer to these sets of blocks, and we will do the same in our discussion. As there are $m^{5/6}$ blocks, the number of superblocks thus formed is $m^{1/6}$. These superblocks are numbered as $1, 2, 3, \ldots, m^{1/6}$.

For a given superblock, we arrange the $m^{4/6}$ blocks that it contains into a square grid, whose sides are of size $m^{2/6}$. The blocks of the superblock are placed on the integral points of the grid. The grid is placed at the origin of a two-dimensional coordinate space with its sides parallel to the coordinate axes. This gives a unique coordinate to each of the integral points of the grid, and thus to the blocks placed on those points. It follows that if (x, y) is the coordinate of a point on the grid, then $0 \leq x, y < m^{2/6}$.

We can now have a natural way of addressing the blocks of a given superblock – we will use the x-coordinate and the y-coordinate of the point on which the block lies. So, a given block can be uniquely identified by the number of the superblock to which it belongs, and the x and y coordinates of the point on which it lies. Henceforth, we will address any block by a three-tuple of the form (s, x, y), where the s is its superblock number, and (x, y) are the coordinates of the point on which it lies.

To address a particular element of the universe, apart from specifying the block to

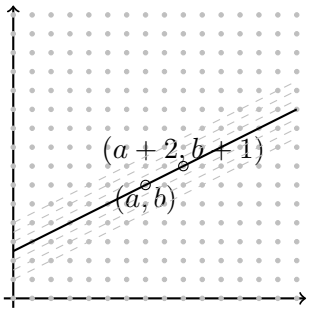


Figure 5.2: The figure shows the grid for superblock 2, and some of the lines with slope 1/2. Note that the line passing through (a, b) intersects the y-axis at a non-integral point.

which it belongs, we need to further state its index within that block. So, an element will be addressed by a four-tuple such as (s, x, y, i), where the first three components specify the block to which it belongs, and the fourth component specifies its index.

5.2.3 Table T_1

Table T_1 of our data structure has the space to store one block for every possible point of the grid (described in the previous section). So, for the coordinate (x, y) of the grid, table T_1 has space to store one block; similarly for all other coordinates. As every superblock has one block with coordinate (x, y), all of these blocks share the same location in table T_1. So, we can imagine table T_1 as a square grid containing $m^{4/6}$ points, where each point can store one block.

There are a total of $m^{4/6}$ points in the grid, and the size of a block is $m^{1/6}$, so the space required by table T_1 is $m^{5/6}$.

5.2.4 Lines for Superblocks

Given a superblock whose number is i, we associate a certain number of lines with this superblock each of whose slopes is $1/i$. In the grid arrangement of the superblock (Section 5.2.2), we draw enough of these lines of slope $1/i$ so that every grid point falls on one of these lines. Figure 5.2 shows the grid and the lines.

So, all lines of a given superblock have the same slope, and lines from different superblocks have different slopes. As there are $m^{1/6}$ superblocks, and they are numbered

$1, 2, \ldots, m^{1/6}$, so, we have the slopes of the lines vary as

$$0 < i \leq m^{1/6}. \tag{5.1}$$

There are two issues to consider – the number of lines needed to cover every point of the grid, and the purpose of these lines. We address the issue of the count of the lines in this section, and that of the purpose of the lines in the next.

We introduce the notation $l_i(a, b)$ to denote the line that has slope $1/i$, and passes through the point (a, b). We now define the collection of all lines of slope $1/i$ that we are going to draw for the superblock i.

$$L_i = \left\{ \; l_i(a, 0) \;\; | \;\; a \in \mathbb{Z}, \; -i(m^{2/6} - 1) \leq a < m^{2/6} \; \right\}. \tag{5.2}$$

In the following three lemmas, we show the properties of this set of lines.

Lemma 5.2. *Every line of L_i contains at least one point of the grid.*

Proof. Consider an arbitrary line $l_i(a, 0)$ of L_i. If $0 \leq a < m^{2/6}$, then $(a, 0)$ itself is a member of the grid, and $l_i(a, 0)$ is non-empty.

Let us now consider the scenario where $-i(m^{2/6} - 1) \leq a < 0$. Let $-a = qi + r$, where $0 \leq r < i$.

If $r = 0$, we show that $(0, q)$ is a point that falls on the line through $(a, 0)$, and it also belongs to the grid. First,

$$\frac{q - 0}{0 - a} = \frac{q}{qi + 0} = \frac{1}{i},$$

which shows that the point falls on the required line. Also,

$$\begin{array}{rccccc} & -i(m^{1/2} - 1) & \leq & a & < & 0 \\ \implies & -i(m^{1/2} - 1) & \leq & -qi - 0 & < & 0 \\ \implies & m^{1/2} - 1 & \geq & q & > & 0, \end{array}$$

which shows that $(0, q)$ belongs to the grid. Together they show that $(0, q) \in l_i(a, 0)$.

On the other hand, if $0 < r < i$, the point to consider is $(i - r, q + 1)$. The following

equality shows that the point lies on the line through $(a, 0)$ –

$$\frac{q+1-0}{i-r-a} = \frac{q+1-0}{i-r+qi+r} = \frac{1}{i}.$$

To show that the point belongs to the grid, the x-coordinate satisfies the following $0 < i - r < m^{1/6}$ (Equation 5.1). As for the y-coordinate, we have

$$\begin{aligned} & -i(m^{2/6}-1) \leq a < 0 \\ \implies & -i(m^{2/6}-1) \leq -qi-r < 0 \\ \implies & m^{2/6}-1 \geq q+r/i > 0 \\ \implies & m^{2/6}-1 \geq \lceil q+r/i \rceil > 0 \\ \implies & m^{2/6}-1 \geq q+1 > 0 \end{aligned}$$

This shows that even when r in non-zero, $l_i(a, 0)$ is non-empty. □

Lemma 5.3. *Every point of the grid belongs to some line of L_i.*

Proof. Let (a, b) be an arbitrary element of the grid. By construction, a and b are both integers, and $0 \leq a, b < m^{2/6}$. If $b = 0$, then $(a, 0) \in l_i(a, 0)$.

If $b \neq 0$, consider the point $(a - bi, 0)$. As

$$\frac{a-bi-a}{0-b} = \frac{1}{i},$$

(a, b) falls on the line through $(a - bi, 0)$. And using arguments similar to the one employed in the previous lemma, one can show that $i(m^{2/6} - 1) \leq a - bi < m^{2/6}$. So, (a, b) falls on the line $l_i(a - bi, 0)$. □

Lemma 5.4. $| L_i | = (i + 1)(m^{2/6} - 1) + 1$.

Proof. The equality is a direct consequence of the definition of L_i (Equation 5.2). □

5.2.5 Table T_0

In table T_0, we have space to store one block for every line of every superblock. That means that for a superblock, say i, all of its blocks that fall on the line $l_i(a, b)$ share the same block in table T_0; and the same is true for all lines of every superblock.

The i^{th} superblock contains $\mid L_i \mid = (i+1)(m^{2/6}-1)+1$ lines (Lemma 5.4), so the total number of lines from all of the superblocks is

$$\begin{aligned} & \mid L_1 \mid + \mid L_2 \mid + \cdots + \mid L_{m^{1/6}} \mid \\ = & \sum_{i=1}^{m^{1/6}} \Big((i+1)(m^{2/6}-1)+1 \Big) \\ = & \left(\tfrac{(m^{1/6})(m^{1/6}+1)}{2} + m^{1/6} \right) (m^{2/6}-1) + m^{1/6} \\ = & \mathcal{O}(m^{4/6}). \end{aligned}$$

As mentioned earlier, we reserve space for one block for each of these lines. Combined with the fact that the size of a block is $m^{1/6}$, we have space required by table T_0 is $\mathcal{O}(m^{5/6})$.

5.2.6 Notations

As described in Section 5.2.2, any element of the universe $\mathcal{U}$ can be addressed by a four-tuple, such as (s, x, y, i), where s is the superblock to which it belongs, (x, y) are the coordinates of its block within that superblock, and i is its index within the block.

Table T has one bit for each block, so all elements of a block will query the same location. As the block number of the element (s, x, y, i) is (s, x, y), so the bit corresponding to the element is $T(s, x, y)$; or in other words, the element (s, x, y, i) will query the location $T(s, x, y)$ in table T.

In table T_1, there is space for one block for every possible coordinates of the grid. The coordinates of the element (s, x, y, i) is (x, y), and T_1 has space to store an entire block for this coordinate. So, there is one bit for every element of a block, or, in other words, every index of a block. So, the bit corresponding to the element (s, x, y, i) is $T_1(x, y, i)$.

Table T_0 has a block reserved for every line of every superblock. The element (s, x, y, i) belongs to the line $l_s(x, y)$, and thus table T_0 has space to store one block corresponding to this line. As the index of the element is i, so the bit corresponding to the element in table T_0 is $T_0(l_s(x, y), i)$.

5.2.7 Query Scheme

The query scheme is easy enough to describe once the data structure has been finalised; it follows the decision tree as discussed earlier (Figure 5.1). Suppose we want to answer the following membership query – "Is (s, x, y, i) in $\mathcal{S}$?" We would make the first query in table T at location $T(s, x, y)$. If the bit stored at that location is 0, we query in table T_0 at $T_0(l_s(x, y), i)$, otherwise we query table T_1 at $T_1(x, y, i)$. If the answer from the second query is 1, then we declare the element to be a member of $\mathcal{S}$, else we declare that it is not a member of $\mathcal{S}$.

5.2.8 The Storage Scheme

The essence of any bitprobe scheme is the storage scheme, i.e. given a subset $\mathcal{S}$ of the universe $\mathcal{U}$, how the bits of the data structure are set such that the query scheme answers membership questions correctly. We start the description of the storage scheme by giving an intuition for its construction.

5.2.8.1 Intuition

The basic unit of storage in the tables T_0 and T_1 of our data structure, in some sense, is a block – table T_0 can store one block of any line of any superblock, and table T_1 can store one block of a given coordinate from any superblock. We show next that our storage scheme must ensure that an empty and non-empty block cannot be stored together in a table.

Suppose, the block (s, x, y) of table T is non-empty, and it contains the member (s, x, y, i) of subset $\mathcal{S}$. If we decide to store this member in table T_0, then we have to store the block (s, x, y) in table T_0. So, we have to set in table T the following – $T(s, x, y) = 0$. Thus, (s, x, y, i) upon first query will get a 0 and go to table T_0. In table T_0, we store the block (s, x, y) at the storage reserved for the line $l_s(x, y)$. Particularly, we have to set $T_0(l_s(x, y), i) = 1$.

If (s, x', y') is a block that is empty, i.e. it does not contain any member of $\mathcal{S}$, and it falls on the aforementioned line, i.e. $l_s(x', y') = l_s(x, y)$, then we cannot store this block in table T_0, and hence $T(s, x', y')$ must be set to 1. If this is not the case, and $T(s, x', y') = 0$, then the first query for the element (s, x', y', i) will get a 0, go to table

T_0 and query the location $T_0(l_s(x', y'), i)$ which is same as $T_0(l_s(x, y), i)$. We have set this bit to 1, and we would incorrectly deduce that (s, x', y', i) is a member of $\mathcal{S}$.

The same discussion holds true for table T_1. If we decide to store the block (s, x, y) in table T_1, we have to set $T(s, x, y)$ to 1. In table T_1, we have space reserved for every possible coordinate for a block, and we would store the block at the coordinate (x, y); particularly, we would set $T_1(x, y, i)$ to 1. This implies that all empty blocks from other superblocks having the same coordinate cannot be stored in table T_1, and hence must necessarily be stored in table T_0. To take an example, if (s', x, y) is empty, then it must stored it in table T_0, and hence $T(s', x, y) = 0$.

To summarize, for any configuration of the members of subset $\mathcal{S}$, as long as we are able to keep the empty and the non-empty blocks separate, our scheme will work correctly. For the reasons discussed above, we note the following.

1. We have to keep the non-empty blocks and empty blocks separate.

2. We have to keep the non-empty blocks separate from each other; and

3. The empty blocks can be stored together.

Our entire description of the storage scheme would emphasize on how to achieve the aforementioned objective.

5.2.8.2 Description

Let the four members of subset $\mathcal{S}$ be

$$\mathcal{S} = \Big\{ (s_1, x_1, y_1, i_1),\ (s_2, x_2, y_2, i_2),\ (s_3, x_3, y_3, i_3),\ (s_4, x_4, y_4, i_4) \Big\}.$$

So, the relevant blocks are

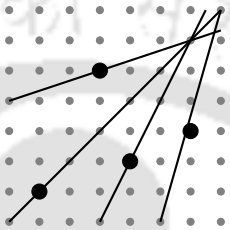
$$\Big\{ (s_1, x_1, y_1),\ (s_2, x_2, y_2),\ (s_3, x_3, y_3),\ (s_4, x_4, y_4) \Big\},$$

and the relevant lines are

$$\Big\{ l_{s_1}(x_1, y_1),\ l_{s_2}(x_2, y_2),\ l_{s_3}(x_3, y_3),\ l_{s_4}(x_4, y_4) \Big\}.$$

In the discussion below, we assume that no two members of $\mathcal{S}$ belong to the same block. This implies that there are exactly four non-empty blocks. The scenario where a block contains multiple members of $\mathcal{S}$ is handled in Section 5.2.9.

The lines for the members of $\mathcal{S}$ need not be distinct, say when two elements belong to the same superblock and fall on the same line. We divide the description of our storage scheme into several cases based on the number of distinct lines we have due to the members of $\mathcal{S}$, and for each of those cases, we provide the proof of correctness alongside it.



Case I: Suppose we have four distinct lines for the four members of $\mathcal{S}$. The slopes of some of these lines could be same, or they could all be different. We know that all lines of a given superblock have the same slope, and lines from different superblocks have different slopes (Section 5.2.4). We also know that if two of these lines, say $l_{s_1}(x_1, y_1)$ and $l_{s_2}(x_2, y_2)$, have the same slope, then the corresponding members of $\mathcal{S}$ belong to the same superblock, i.e. $s_1 = s_2$. On the other hand, if their slopes are distinct, then they belong to different superblocks, and consequently, $s_1 \neq s_2$.

Table T_0 has space to store one block for every line in every superblock. As the lines for the four members of $\mathcal{S}$ are distinct, the space reserved for the lines are also distinct. So we can store the four non-empty blocks in table T_0, and all of the empty blocks in table T_1.

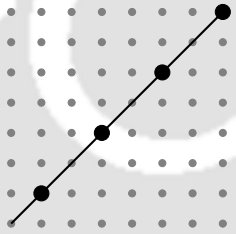
To achieve the objective, we set $T(s_j, x_j, y_j) = 0$ for $1 \leq j \leq 4$, and set the bits in table T for every other block to 1. In table T_0, we set the bits $T_0(l_{s_j}(x_j, y_j), i_j) = 1$, for $1 \leq j \leq 4$, and all the rest of the bits to 0. In table T_1, all the bits are set to 0.

So, if e is an element that belongs to an empty block, it would, according to the assignment above, get a 1 upon its first query in table T. Its second query will be in table T_1, and as all the bits of table T_1 are set to 0, we would conclude that the element e is not a member of $\mathcal{S}$.

Suppose, (s, x, y, i) be an element that belongs to one of the non-empty blocks.

Then, its coordinates must correspond to one of the four members of $\mathcal{S}$. Without loss of generality let us assume that $s = s_1, x = x_1$, and $y = y_1$.

It follows that $T(s, x, y)$, which is same as $T(s_1, x_1, y_1)$, is 0, and hence the second query for this element will be in table T_0. The line corresponding to the element is $l_s(x, y)$, which is same as $l_{s_1}(x_1, y_1)$, and hence the second query will be at the location $T_0(l_s(x, y), i) = T_0(l_{s_1}(x_1, y_1), i)$. As the four lines for the four members of $\mathcal{S}$ are distinct, so $T_0(l_{s_1}(x_1, y_1), i)$ will be 1 if and only if $i = i_1$. So, we will get a `Yes` answer for the query if and only if the element (s, x, y, i) is actually the element (s_1, x_1, y_1, i_1), a member of $\mathcal{S}$.



Case II: Let us consider the case when there is just one line for the four members of $\mathcal{S}$. As all of their lines are identical, and consequently, the slopes of the lines are the same, all the elements must belong to the same superblock. So, we have $s_1 = s_2 = s_3 = s_4$.

As all the non-empty blocks belong to the same superblock, all of their coordinates must be distinct. Table T_1 can store one block for each distinct coordinate of the grid, and hence we can store the four non-empty blocks there. All the empty blocks will be stored in table T_0.

To this end, we set $T(s_j, x_j, y_j) = 1$ for $1 \leq j \leq 4$, and the rest of the bits of table T, which correspond to the empty blocks, to 0. In table T_0, all bits are set to 0. In table T_1, the bits corresponding to the four elements are set to 1, i.e. $T_1(x_j, y_j, i_j) = 1$ for $1 \leq j \leq 4$. The rest of the bits of table T_1 are set to 0.

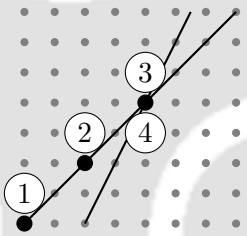
The proof of correctness follows directly from the assignment, and the reasoning follows along the lines of the previous case. If the element e belongs to an empty block, it will get a 0 from table T upon its first query, consequently go to table T_0 for its second query, and get a 0, implying e is not a member of $\mathcal{S}$.

If the element (s, x, y, i) belongs to a non-empty block, then its coordinates must

correspond to one of the members of $\mathcal{S}$. Without loss of generality, let $s = s_1, x = x_1$, and $y = y_1$.

The first query of the element will be at the location $T(s, x, y) = T(s_1, x_1, y_1)$, and hence it will get a 1 from table T, and go to table T_1 for its second query. In this table, it will query the location $T_1(x, y, i)$, which is same as $T_1(x_1, y_1, i)$. As the coordinates of the four members of $\mathcal{S}$ are distinct, $T_1(x_1, y_1, i)$ will be 1 if and only if $i = i_1$. So, we get a 1 in the second query if and only if we have $(s, x, y, i) = (s_1, x_1, y_1, i_1)$, a member of $\mathcal{S}$.

Case III: The next case that we consider is when there are two distinct lines corresponding to the four members of subset $\mathcal{S}$. The members can be distributed in one of two ways – one line contains three elements and the other line one, or the elements might be divided equally among the two lines. We consider the cases separately below.



Case III(A): Consider the case when one line contains three elements, and the other line contains one. Without loss of generality, let the first three members of $\mathcal{S}$ belong to one line, and the fourth one to another one. So, we have $l_{s_1}(x_1, y_1) = l_{s_2}(x_2, y_2) = l_{s_3}(x_3, y_3)$, and the line $l_{s_4}(x_4, y_4)$ is different from the others. As lines with same slopes belong to the same superblock, we have $s_1 = s_2 = s_3$. Whether the fourth member belongs to the aforementioned superblock, or to a different superblock depends on whether the slope of $l_{s_4}(x_4, y_4)$ is same as the other line or it is distinct.

As the first three elements belong to the same superblock, all will have coordinates distinct from one another. The coordinates of the fourth element could be distinct, or it could overlap with one of the first three.

The case of the coordinates of the four members of $\mathcal{S}$ being distinct is one we have seen in Case II, where the elements too had distinct coordinates. The assignment for this scenario will be identical to that case, and consequently, the correctness proof follows.

Let us say that the coordinates of the fourth element coincides with one of the other three members. Without loss of generality, let us assume that the third and the fourth elements have identical coordinates, that is to say $x_3 = x_4$ and $y_3 = y_4$. As two blocks of a superblock cannot have the same coordinates, we must have $s_3 \neq s_4$. Moreover, different superblocks have different slopes for its lines, implying $l_{s_1}(x_1, y_1) = l_{s_2}(x_2, y_2) = l_{s_3}(x_3, y_3) \neq l_{s_4}(x_4, y_4)$.

The assignment in this case will be as follows – we will store the blocks corresponding to the first two elements in table T_1, and the blocks corresponding to the last two elements in table T_0. The empty blocks accordingly will have to be distributed among the two tables.

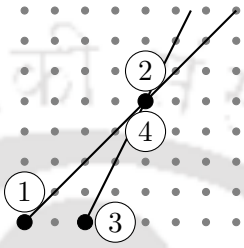
Accordingly, we set $T(s_1, x_1, y_1)$ and $T(s_2, x_2, y_2)$ to 1, and set $T(s_3, x_3, y_3)$ and $T(s_4, x_4, y_4)$ to 0. The bits corresponding to the remaining blocks in the two lines, which are $l_{s_1}(x_1, y_1)$ and $l_{s_4}(x_4, y_4)$, are set to 1. The bits of the blocks of all the other lines in all of the superblocks are set to 0.

In table T_0, the bits corresponding to the third and the fourth element is set to 1, i.e. $T_0(l_{s_3}(x_3, y_3), i_3) = T_0(l_{s_4}(x_4, y_4), i_4) = 1$, and all the remaining bits are set to 0. In table T_1, only the bits corresponding to the first two elements are set to 1, i.e. $T_1(x_1, y_1, i_1) = T_1(x_2, y_2, i_2) = 1$; the rest of the bits of this table are set to 0.

We now prove that the assignment above is correct. If an element e belongs to a line other than the lines $l_{s_1}(x_1, y_1)$ and $l_{s_4}(x_4, y_4)$, then the bit for its block has been set to 0. Consequently, it will query table T_0. Table T_0 has separate space for each line, and only certain bits of the non-empty lines have been set to 1. As e falls on a line different from $l_{s_1}(x_1, y_1)$ and $l_{s_4}(x_4, y_4)$, so the second query for e will also return a 0.

Suppose e belongs to an empty block falling on one the lines $l_{s_1}(x_1, y_1)$ and $l_{s_4}(x_4, y_4)$. According to our assignment, the bits of the empty blocks from the lines are set to 1, and hence the second query for e will go to table T_1. All blocks falling on a line have distinct coordinates, so the coordinates of the block of e will be distinct from the coordinates of the non-empty blocks of the two lines. As table T_1 has space to store one block for each distinct coordinate, the space for the empty blocks of the two lines will be different from the non-empty ones. As we have set certain bits of the only the non-empty blocks of table T_1 to 1, all the bits of the block of e must be 0, and hence the answer to second query for e will be 0.

It remains to verify whether the queries corresponding to the elements of the four non-empty blocks give correct answers. We have argued above that the empty blocks are stored in locations distinct from the non-empty blocks. The assignment tells us that we have stored the non-empty blocks in its entirety. These two facts together imply that queries for elements in the non-empty blocks will also give correct answers.



Case III(B): We now consider the case when the four members of $\mathcal{S}$ are divided equally among the two lines. Without loss of generality, let us assume that the first two members belong to one line, and the other two members belong to the other line. So, we have $l_{s_1}(x_1, y_1) = l_{s_2}(x_2, y_2)$ and $l_{s_3}(x_3, y_3) = l_{s_4}(x_4, y_4)$. Consequently, we have $s_1 = s_2$ and $s_3 = s_4$.

In this scenario, we may have the four non-empty blocks occupying four distinct coordinates of the grid. This situation is familiar to us, and we will handle it as we have done in Case II.

The other scenario is when coordinates of non-empty blocks overlap. As the lines are distinct, they can have an intersection point if and only if they have different slopes. It implies that the lines belong to different superblocks, and hence $s_1 = s_2 \neq s_3 = s_4$. Further, as there is only one common point between the two lines, only one pair of non-empty blocks from the two lines can overlap, i.e. have the same coordinates. Without loss of generality, let it be the second and fourth member of $\mathcal{S}$. So, we have $x_2 = x_4$ and $y_2 = y_4$.

For all blocks which do not fall on any of the two aforementioned lines, and hence implying that they are empty, we set their bits in table T to 0. So, the second query for the elements of these blocks will be in table T_0. As we already know, table T_0 has seperate space reserved for all lines, and we set all the bits of all of those empty lines to 0.

An important thing to note so far is we have not stored anything in table T_1 yet. We now look into the assignment of the blocks that fall on the two non-empty lines. The blocks that fall on a line have distinct coordinates, so the blocks on the line $l_{s_1}(x_1, y_1)$ have distinct spaces in table T_1, and we store all these blocks in table T_1. We accordingly set the corresponding bits in table T and T_1.

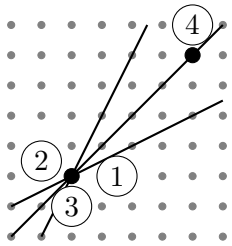
We now look into the assignment of the blocks on the other line, namely $l_{s_3}(x_3, y_3)$. There is only one block on this line whose coordinate is same as a point on the other line – the block corresponding to the fourth member of $\mathcal{S}$ has the same coordinate as the second member of $\mathcal{S}$. Then, we cannot store the block (s_4, x_4, y_4) in table T_1 as it is already occupied by the block (s_2, x_2, y_2) from the other line. We store this block in table T_0 at the space reserved for the line $l_{s_3}(x_3, y_3)$. All other blocks of this line can then be stored in table T_1 without any conflict.

The assignment tells us how the empty and the non-empty blocks have been kept separate. An explicit proof of correctness follows along the lines of the previous cases.

Case IV: The final case to consider is when the number of distinct lines due to the non-empty blocks is three. Without loss of generality, let us assume that the blocks corresponding to the third and fourth elements fall on the same line, i.e. $l_{s_3}(x_3, y_3) = l_{s_4}(x_4, y_4)$. This also means that these two blocks belong to the same superblock, and hence, $s_3 = s_4$. It further implies that the coordinates of the two blocks are distinct.

As seen in the previous cases, those lines of the superblocks which do not contain any non-empty block is easy to handle – we simply store them in table T_0 at the space reserved for the respective lines. A point to note is that it also leaves table T_1 untouched. In the discussion below, we will then concentrate on how we handle the blocks from the three lines which are non-empty.

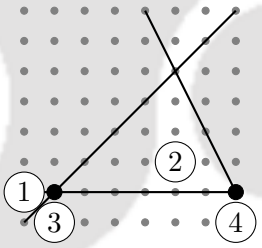
The discussion will be divided into three parts based on how many of those points coincide. As the blocks corresponding to the third and the fourth members have distinct coordinates, it follows that at most three of the non-empty blocks can coincide.



Case IV(A): Let us consider the scenario when three of the non-empty blocks coincide. Without loss of generality, let it be the first three blocks, i.e. $x_1 = x_2 = x_3$ and $y_1 = y_2 = y_3$.

We store all the blocks on the line $l_{s_3}(x_3, y_3)$ in table T_1. There is only one point in each of the other two lines, namely $l_{s_1}(x_1, y_1)$ and $l_{s_2}(x_2, y_2)$, that is common with this line – we store the blocks corresponding to those points in table T_0, and the rest of the blocks of the other lines in table T_1. So, the blocks (s_1, x_1, y_1) and (s_2, x_2, y_2) are stored in the location reserved in table T_0 for the lines $l_{s_1}(x_1, y_1)$ and $l_{s_2}(x_2, y_2)$, and the rest of the blocks of these lines are stored in table T_1.

This assignment keeps the empty blocks and the non-empty blocks separate from each other, and the correctness follows.



Case IV(B): Let us consider the case where two pairs of non-empty blocks coincide. Without loss of generality, let the first block coincide with the third and the second block coincide with the fourth.

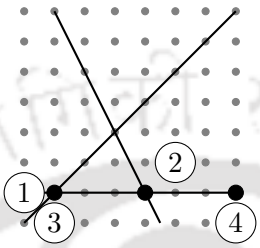
The assignment that we devised for the previous case works in this scenario as well – we store the blocks of the line $l_{s_3}(x_3, y_3)$ in table T_1, and the blocks (s_1, x_1, y_1) of line $l_{s_1}(x_1, y_1)$ and (s_2, x_2, y_2) of line $l_{s_2}(x_2, y_2)$ in table T_0. The other blocks of the lines $l_{s_1}(x_1, y_1)$ and $l_{s_2}(x_2, y_2)$ are stored in table T_1.

The correctness proof of the previous case holds in this scenario as well.

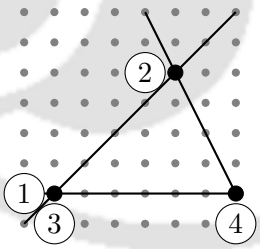
Case IV(C): Let us next consider the scenario where only one pair of non-empty blocks coincide. Without loss of generality, let the first block coincide with the third.

So, we have $x_1 = x_3$ and $y_1 = y_3$. As only one pair of non-empty blocks coincide, the block of the second element do not lie on any of the other non-empty blocks, and hence has coordinates distinct from the rest.

The assignment in this arrangement will depend on the coordinates of the block of the second block – it lies on the line $l_{s_3}(x_3, y_3)$, or it doesn't. We address each of these cases below.



Case IV(C)(i): We store all of the blocks on the line $l_{s_3}(x_3, y_3)$ in table T_1. From the line $l_{s_1}(x_1, y_1)$, only one block lies in the previous line, the block containing the first element. This block will be stored in table T_0 at the location reserved for the line $l_{s_1}(x_1, y_1)$, and the rest of the blocks can be stored in table T_1. From the last line, i.e. $l_{s_2}(x_2, y_2)$, only one block lies on this line, the block that contains the second element. This blocks will be stored in table T_0 at the location for the line $l_{s_2}(x_2, y_2)$, and the rest of the blocks can be stored in table T_1 without conflict.



Case IV(C)(ii): We next consider the case when the block for the second element does not lie on the line $l_{s_3}(x_3, y_3)$. We, in this case, store the second block, i.e. (s_2, x_2, y_2) in table T_1, and the rest of the blocks on its line, i.e. $l_{s_2}(x_2, y_2)$, in table T_0 at its alloted location. We do the same for the block of the first element – store the non-empty block (s_1, x_1, y_1) in table T_1, the rest of the blocks on its line $l_{s_1}(x_1, y_1)$ in table T_0.

The only locations used up in table T_1 are locations for the first and second block, and the blocks left to be allocated space are those falling on the line $l_{s_3}(x_3, y_3)$. The second block do not lie on this line, and hence would not affect the allocations of the

line. The first block coincide with third block falling on this line, so the third block, namely (s_3, x_3, y_3) must necessarily be stored in table T_0 in the space alloted for the line $l_{s_3}(x_3, y_3)$. The rest of the blocks of the line $l_{s_3}(x_3, y_3)$ can now be stored in table T_1 without conflict.

Case IV(D): This is the final configuration to consider when there are three distinct lines due to the non-empty blocks - no block coincide with any other block. This implies that the four non-empty blocks have distinct coordinates, and hence all of them can be stored in table T_1. All the empty blocks can then be stored in table T_0, and we would have avoided all conflict.

5.2.9 Blocks with Multiple Members

In the discussion above, we had assumed that each block can contain at most one member of the subset $\mathcal{S}$, and we have shown for every configuration of the members of $\mathcal{S}$, the bits of the data structure can be so arranged that the membership queries are answered correctly.

In general, a single block can contain upto four members of $\mathcal{S}$, and we need to propose an assignment for such a scenario. As has been noted in the previous section, our basic unit of storage is a block and we differentiate between empty and non-empty blocks. At a given location in table T_0 or T_1, a block is stored in its entirety, or it isn't stored at all. This implies that the number of members of $\mathcal{S}$ a non-empty block contains is of no consequence, as we always store an entire block. The scheme from the previous section would thus hold true for blocks containing multiple members.

We now summarize the result in the theorem below.

Theorem 5.5. *There is an explicit adaptive scheme that stores subsets of size at most four and answers membership queries using two bitprobes such that*

$$s_A(4, m, 2) = \mathcal{O}(m^{5/6}).$$

5.2.10 Revisiting Space Requirement

In Section 5.2.2, we mapped the blocks of superblocks on a square grid. One can see that if block size is taken as y and size of grid to be x, then size of table T becomes m/y, size of table T_1 becomes x^2y, and the size of table T_0 becomes $c \cdot (\frac{m}{x^2y})^2 xy = C \cdot (\frac{m^2}{x^3y})$. This gives us following equation for the space:

$$S(x, y) = x^2y + c \cdot (\frac{m^2}{x^3y}) + \frac{m}{y}. \tag{5.3}$$

We get local minima of above equation at $x = c_1 m^{1/3}$ and $y = c_2 m^{1/6}$. Therefore, we get space requirement for our data structure to be $\mathcal{O}(m^{5/6})$.

Let us now map blocks of size z from a superblock of size xyz to a rectangular grid of size $x \times y$. Further, following the same steps to get the space requirements we get following equation to optimize:

$$S(x, y, z) = xyz + \frac{m}{z} + \frac{m}{y} + c \cdot (\frac{m^2}{x^2yz}). \tag{5.4}$$

We get local minima of above function at $x = m^{2/5}$, and $y = z = m^{1/5}$. So the space requirement becomes $\mathcal{O}(m^{4/5})$. Since going from square grid to rectangular grid does not change the storage and query scheme of the Theorem 5.5, so we can still use the same storage and query scheme to get the following theorem:

Theorem 5.6. *There is an explicit adaptive scheme that stores subsets of size at most four and answers membership queries using two bitprobes such that*

$$s_A(4, m, 2) = \mathcal{O}(m^{4/5}).$$

5.2.11 Counterexample

We now provide an instance of a five-member subset of the universe $\mathcal{U}$ which cannot be stored correctly using our scheme; that is to say, if the storage scheme does indeed store the five elements in our data structure, queries for certain elements will be answered incorrectly.

The Arrangment

Consider four lines from four different superblocks which are arranged as shown in Figure 5.3. Let us suppose that the four superblocks are s_1, s_2, s_3, and s_4, and the labels of the lines are L_1, L_2, L_3, and L_4, respectively. We will put in $\mathcal{S}$ one element each from the first three superblocks, and two elements from the fourth superblock.

Our subset $\mathcal{S}$ will contain the elements e_1 and e_2 from the superblocks s_1 and s_2, respectively. These elements have the property that the blocks they belong to share the same coordinates, and hence lie on the intersection of the lines L_1 and L_2. The fact that they have the same coordinates also implies thay they share the same location in table T_1. Let the elements be $e_1 = (s_1, x, y, i_1)$ and $e_2 = (s_2, x, y, i_2)$. We would also have $i_1 \neq i_2$. This would imply that the two non-empty blocks (s_1, x, y) and (s_2, x, y) cannot both be stored in table T_1.

Consider that block of superblock s_3 that lies on the intersection of the lines L_3 and L_4. We will put one element from that block in our subset $\mathcal{S}$. Let that element be $e_3 = (s_3, x_3, y_3, i_3)$.

Finally we will put two elements of the superblock s_4 in $\mathcal{S}$ – one element from that block of s_4 which lies on the intersection of the lines L_4 and L_1, namely $e_{4,1}$, and another from the block of s_4 which lies on the intersection of the lines L_4 and L_2, namely $e_{4,2}$. These two elements are described as $e_{4,1} = (s_4, x_{4,1}, y_{4,1}, i_{4,1})$ and $e_{4,2} = (s_4, x_{4,2}, y_{4,2}, i_{4,2})$.

The Contradiction

We can store the element e_1 of superblock s_1 in one of two tables T_0 and T_1. Let us assume that we store e_1 in table T_0. As the block containing e_1 lies on the line L_1, we cannot store any of the other empty blocks on the line L_1 in table T_0, and hence they must be stored in table T_1.

The non-empty block of s_4 containing element $e_{4,1}$ which falls on the line L_1, then, cannot be stored in table T_1, and hence must be stored in table T_0. So, the other blocks of L_4 must be stored in table T_1, including the block containing the element $e_{4,2}$.

The non-empty block of s_3 containing the element e_3 falls on the line L_4, and hence must be stored in table T_0. So, all blocks on the line L_3 must now be store in table T_1.

The element e_2 of the superblock s_2 falls on the line L_3 and hence must be stored

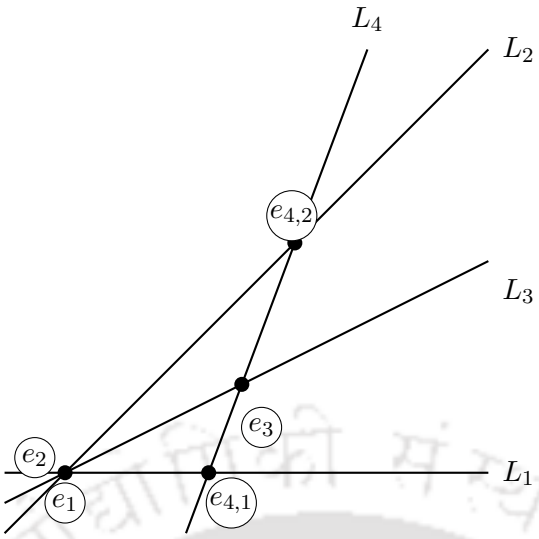


Figure 5.3: Counterexample

in table T_0. So, all blocks of line L_2 must be stored in table T_1.

The block of s_4 containing the element $e_{4,2}$ must be stored in table T_0 by the same argument as above. But we have already argued that $e_{4,2}$ has to be stored in table T_1, and hence we arrive at a contradiction.

The preceeding argument tells us that we cannot store the element e_1 in table T_0. So, we must store it in table T_1. If such is the case, and arguing as above, we can show that this results e_2 being stored in table T_0, which results in $e_{4,2}$ being stored in table T_0. This, in turn, results in e_3 being stored in table T_0, which would force e_1 to be stored in table T_0.

But we have started with the premise that e_1 is being stored in table T_1, and again we reach a contradiction. So, we conclude that this arrangement of elements cannot be stored correctly in our data structure, and hence our data structure is not suitable for storing sets of size five or higher.

In the next chapter, we will avoid this counterexample by mapping blocks of elements on the integral points of three-dimensional cube.

5.3 Conclusion

In this chapter, we have proposed an adaptive scheme for storing subsets of size four and answering membership queries with two bitprobes that improves upon the existing schemes in the literature. This scheme also resolves an open problem due to Patrick

K. Nicholson [5] about the existence of such a scheme that uses the ideas of blocks and superblocks due to Radhakrishnan *et al.* [2]. The technique used is that of arranging the blocks of a superblock in a two-dimensional grid, and grouping them along lines. We believe that this technique can be extended to store larger subsets by extending the idea of an arrangement in a two-dimensional grid to arrangements in three and higher dimensional grids.

"The geometrical mind is not so closely bound to geometry that it cannot be drawn aside and transferred to other departments of knowledge. A work of morality, politics, criticism, perhaps even eloquence will be more elegant, other things being equal, if it is shaped by the hand of Geometry."

–*Bernard le Bovier de Fontenelle*, 1729, 'Preface sur l'Utilite des Mathematiques et la Physique', translated by F. Cajori

6

Explicit Adaptive Two Bitprobe Scheme Storing Five Elements

6.1 Introduction

In this chapter, we are dealing with the design of the explicit adaptive scheme in the bitprobe model to store an arbitrary subset $\mathcal{S}$ of size at most five from a universe $\mathcal{U}$ of size m, and answer the membership query in two adaptive bit probes. Our scheme is built on the counterexample shown in the Section 5.2.11. In the counterexample, we see that line L_4 cuts three lines, i.e L_1, L_2 and L_3 passing through a point at three different points. Now, if in our scheme, we draw lines in such a way that no three lines passing through a point lies in a same plane, then the counterexample will not exist. Our scheme uses the geometric idea of Kesh [18], to overcome the counterexample by mapping the universe $\mathcal{U}$ of size m on the integral point of a three-dimensional cube. Further, we use the idea of Radhakrishnan [2] to divide the universe $\mathcal{U}$ into blocks and superblocks. Superblocks are nothing but the collection of a suitable number of blocks. Using the combination of the above ideas, we have a geometric data structure for the aforementioned set and query sizes.

6.1.1 Our Contribution

As mentioned earlier, in this chapter we are dealing with the design of explicit adaptive two bitprobe scheme for the subset $\mathcal{S}$ of size five ($n = 5$). Now, we will talk about results in the context of this problem. There is a non-explicit $(n, m, c \cdot m^{1-\frac{1}{4n+1}}, 2)$-scheme by Garg and Radhakrishnan [15], for the set of size five ($n = 5$), their scheme takes $\mathcal{O}(m^{16/17})$ space. Furthermore, there is an explicit $(n, m, c \cdot m^{1-\frac{1}{4n-1}}, 2)$-scheme given by Garg [16], for the set of size four ($n = 5$) his scheme takes $\mathcal{O}(m^{14/15})$ space. Our result improves upon the aforementioned schemes for the subset $\mathcal{S}$ of size five ($n = 5$). Our claim is the following.

Result 6.1 (Theorem 6.3)**.** $s_A(5, m, 2) = \mathcal{O}(m^{10/11})$.

The bitprobe model in the context of two adaptive bitprobes is discussed in detail in Section 5.1.3.

6.2 Our Data Structure

In this section, we present a two adaptive bitprobe data structure which stores an arbitrary subset $\mathcal{S}$ of size 5 from a universe $\mathcal{U}$ of size m.

6.2.1 Our Approach to the Problem

Our scheme has borrowed the idea of the geometric arrangement of elements on a three-dimensional cube from Kesh [18]. Kesh in his paper used the idea of geometric arrangements of elements on high dimensional cubes to come up with $(2, m, c \cdot m^{1/(t-2^{-1})}, t)$-scheme for $t \geq 2$. We have also used the idea of dividing the universe into blocks and superblocks from Radhakrishnan *et al.* [2]. Baig and Kesh [3] used the combination of the ideas mentioned above to come up with a tight explicit adaptive scheme for $n = 3$. Baig *et al.* [6] used a similar idea to map elements on a square grid to come up with an improved scheme for $n = 4$. In this section, we use this geometrical technique to map the blocks of elements from a superblock to the integral points of a three-dimensional cube as shown in Figure 6.1. Moreover, we partition the cube by drawing slices, and further partitioning the slices by drawing lines. We do this for all the superblocks.

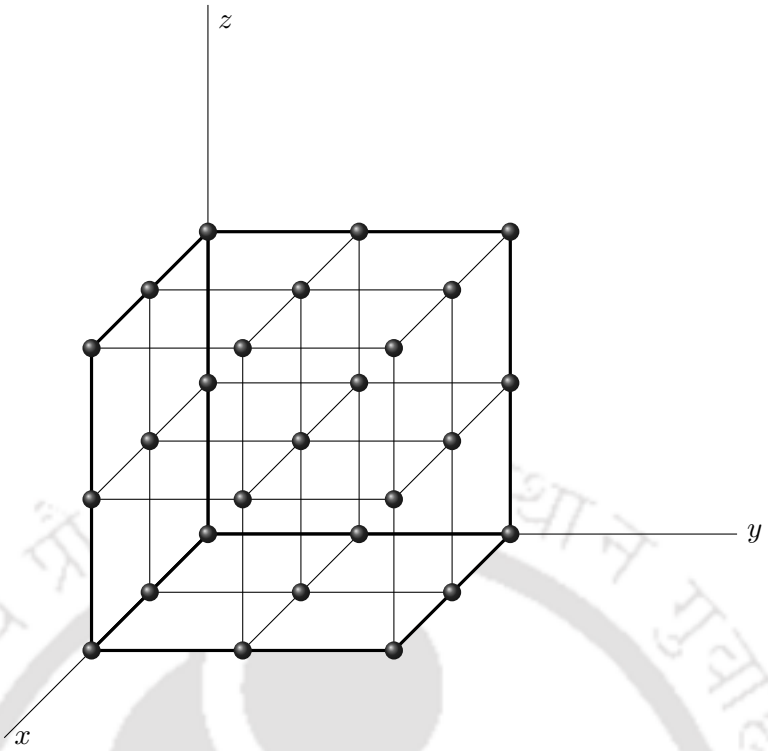


Figure 6.1: Blocks of Superblocks placed on the integral points of a cube

We divide the universe $\mathcal{U}$ of size m into blocks and superblocks similar to the Radhakrishnan et al. [2]. We divide the universe into blocks of size y, so the number of blocks will be m/y. We then collect x^3 consecutive blocks to form a superblock of size x^3y. So we will have m/x^3y superblocks of size x^3y.

Table T

This table consists of one bit of space for each block. So the size of Table T is m/y bits.

Table T_1

Table T_1 is arranged in a three-dimensional cube of side x. So in this cube, we have x^3 integral points. Each integral point on or inside the cube contains a block of size y. So the size of table T_1 is x^3y. Since the size of each superblock is x^3y, all the blocks belonging to a superblock can be mapped on or inside the integral point of the cube. All other superblocks can be thought of as superimposed over each other in the cube. So each point in the cube or table T_1 is shared by blocks of m/x^3y superblocks.

Table T_0

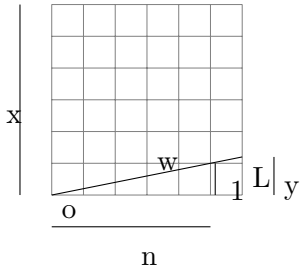


Figure 6.2: A line with slope $1/n$ in the bottom most layer of the cube

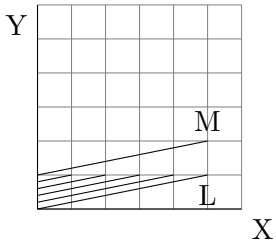


Figure 6.3: Figure showing number of lines drawn between two same slope lines

While discussing the structure of table T_1, we saw that each superblock is mapped on a three-dimensional cube in such a way that all of them are superimposed. Now, for the n^{th} superblock, we first draw a family of lines in the bottom-most layer of the cube in the XY-plane with slope $1/n$ in such a way that all the integral points are covered by the lines.

Lemma 6.1. *The number of lines passing through all the integral points of a square grid with slope* $1/n$ *is* $2x + (n-1)(x-1) - 1$*, where* x *is the length of the square grid.*

Proof. As shown in Figure 6.3, if the slope of the line M and L is $1/n$, then between them, there can be only $n-1$ lines of slope $1/n$ passing through integral points of the grid. So the total number of lines that we can draw with slope $1/n$ is x lines from integral points on X-axis, $x-1$ lines from the integral points on Y axis and $(n-1)(x-1)$ lines between lines from the integral points on Y-axis. So we have $2x + (n-1)(x-1) - 1$ lines with slope $1/n$. □

Using Lemma 6.1, we can say that the total number of lines with slope $1/n$ in the bottom-most layer can be $c \cdot nx$, where c is a constant, and x is the side of the cube. Now, we cut slices of the cube along these lines and perpendicular to XY-plane. So the total number of slices for the n^{th} superblock will be equal to the number of lines drawn in the bottom-most layer of the cube for the n^{th} superblock, i.e $c \cdot nx$. All the slices have a height equal to the length of the cube, i.e x. Let us now calculate the maximum width of a slice of slope $1/n$. Width of the slice formed by line segment OL as shown in Figure 6.2 can be calculated to be $x/n\sqrt{1+n^2}$. We can see from Figure 6.2 that all other slices with this slope will have width less than or equal to $x/n\sqrt{1+n^2}$. So a slice belonging to n^{th} superblock will have length x and width less than equal to $x/n\sqrt{1+n^2}$.

We now draw a family of lines in all the slices of all the superblocks. For the slices belonging to the n^{th} superblock, we draw lines with slope $n^2/\sqrt{1+n^2}$ as shown in Figure 6.4. The lines are drawn in such a way that all the integral points are covered. Let us now calculate the maximum number of lines drawn on a slice belonging to the n^{th} superblock. We can see from Figure 6.4 that the total number of lines drawn from Z-axis is equal to x. Also, number of integral points on the width of the slice is equal to $x/n\sqrt{1+n^2}/\sqrt{1+n^2} = x/n$. So we can draw $\frac{x}{n}$ lines through those integral points on the width of the slice. Now from Figure 6.4, we can see that the number of lines that can pass between two consecutive integral points on the width of the slice is n^2-1. So the total number of lines drawn on this slice with slope $n^2/\sqrt{1+n^2}$ is equal to $x + x/n + n^2 \cdot x/n$ i.e $c \cdot nx$. We say that a slice is having slope $1/n$ if it's projection on the XY-plane has slope $1/n$. Now let us bound the total number of lines drawn on slices whose projections on the XY-plane makes slope $1/n$. The total number of lines should be less than the product of the number of lines drawn on a slice of a maximum width of slope $1/n$ and the total number of slices of slope $1/n$. So the total number of lines drawn for the n^{th} superblock is less than $c_1 \cdot n^2x^2$. We need to sum this for all the superblocks to get the total number of lines drawn. So the total number of lines drawn is

$$\sum_{i=1}^{m/x^3y} c_1 \cdot i^2(x)^2 \leq c \cdot \frac{m^3}{x^7y^3}. \tag{6.1}$$

For each line drawn in a slice, we have a block of space in table T_0. So the total size of table T_0 is $c \cdot m^3/x^7y^2$ bits. Summing up the space taken by tables T, T_0 and T_1, we get the following equation:

$$S(x,y) = x^3y + C \cdot (\frac{m^3}{x^7y^2}) + \frac{m}{y}. \tag{6.2}$$

Choosing $x = m^{3/11}$ and $y = m^{1/11}$, we get space taken by our data structure to be $\mathcal{O}(m^{10/11})$. Now we will prove the following lemma:

Lemma 6.2. *No three lines passing through an integral point of the cube lies in the same plane.*

Proof. From the construction of the table, we can see that all the lines drawn in the cube for a given superblock are parallel to each other. So the lines which pass through the

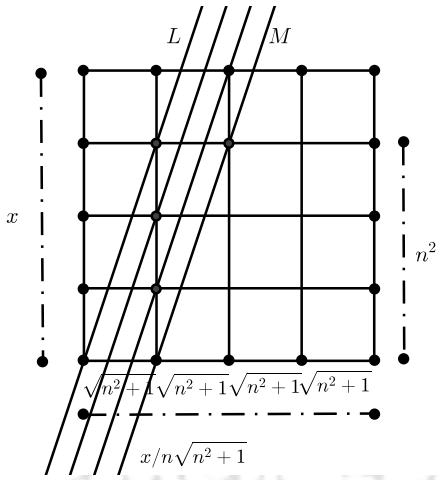


Figure 6.4: A Slice Belonging to n^{th} Superblock

same integral point of the cube belong to the different superblocks. Let us consider the three arbitrary superblocks to which our lines belong. Without loss of generality let us say that the projection of these slices on the XY-plane makes angle $1/n_1, 1/n_2$ and $1/n_3$ with the X-axis. Our lines lie completely in the slices belonging to their superblock. While drawing lines in the slices for the first, second, and third superblock, we are going up in Z direction by n_1^2, n_2^2, and n_3^2. Hence our lines cannot lie in the same plane.

□

From the structure of tables, we may draw the following conclusion. In general, two blocks having elements should not map at the same location in table T_1 or T_0. Otherwise, we may make a mistake on the query belonging to these blocks. Further, if the block having element is mapped in table T_1 or T_0, then no other block should be sent to that table whose position is matched with the block having an element. So if a block having an element on a line is mapped to table T_0, then all other blocks lying on that line should be sent to table T_1. As for each line, we have only one block of space in table T_0. On the contrary, if the block having an element from a line is sent to table T_1, then other blocks lying on the line which contains this block can be sent to table T_0 or T_1. The blocks which are not having any elements given to be stored can be mapped at the same location in table T_1 or T_0. Further, in the rest of the chapter, whenever we say the line passing through a block or the line having a block, we always mean the line drawn in the superblock to which the block belongs.

6.2.2 The Query Scheme

Given a query element, we find the block and superblock to which it belongs. Further, we query the first table, if the first table returns zero, we query to table T_0 else we query to table T_1. We say that element is part of the set to be stored if and only if the last query returns one.

6.2.3 The Storage Scheme

In this section, we talk about the way bits of tables are set to store the subset of size at most five from a universe of size m. We divide the storage scheme into various cases depending upon the way blocks having elements are distributed on the line belonging to their superblock. To generate all the cases, first of all, we partition the number five; then we put those many elements into different superblocks. Further, the positions of the blocks having elements on the line belonging to their superblocks are considered. While handling cases, we see the intersections of the lines, which contains blocks having elements given to be stored. We then decide which block to send to table T_0 and which to T_1. As in our data structure, we always send a block to either table T_0 or T_1, and we store its bit vector there, so we will always assume that elements which are given to be stored lies in the different block. Proving the results for elements belonging to different blocks proves the result when many elements belong to the same block. In this section, we will discuss few cases. The rest of the cases can be generated and handled similarly, and are mentioned in Appendix 8.3.

Case 1. If all the elements of S lie in one superblock, then we send the blocks having elements to table T_1 and all the empty blocks to table T_0.

Case 2. If four elements $S_1 = \{n_1, n_2, n_3, n_4\}$ lie in one superblock and one element $S_2 = \{n_5\}$ in other superblock then we can have two cases, either the block containing the element n_5 coincides with one of the block containing element from S_1 in table T_1 or it does not coincides. So if the block containing the element n_5 coincides with one of the blocks containing an element from S_1, then we send the block having the element n_5 to table T_1 and send the block from which it was coinciding to table T_0. All other blocks

of superblock which contain elements from S_1 are sent to table T_1. All other blocks of superblock which contain the element n_5 are sent to table T_0. Rest all the empty blocks are sent to table T_0. On the other hand, if the block containing element n_5 do not coincide with any of the block having element from S_1 in table T_1, then we send all the blocks having elements from S_1 and S_2 to table T_1, and rest all the empty blocks to table T_0.

Case 3. If three elements $S_1 = \{n_1, n_2, n_3\}$ lie in one superblock and two elements $S_2 = \{n_4, n_5\}$ in other superblock then we store according to following scheme.

Case 3.1 All the blocks to which elements from S_1 belong lies on the same line of their superblock. From here onwards, whenever we say line passing through a block or blocks lying on a line, we mean the line drawn in the superblock to which these blocks belong.

Case 3.1.1 Two blocks to which elements from S_2 belong coincides with the blocks corresponding to the elements from S_1 in table T_1. In this case, we send the blocks having elements from S_2 to table T_0. Further, we send empty blocks lying on the lines to which elements from S_2 belongs to table T_1. We send all the blocks which contain elements from S_1 in table T_1. Finally, we send the rest of the empty blocks to table T_0.

Case 3.1.2

Only one block which contains an element from S_2 coincides with the block corresponding to the elements from S_1 in table T_1. In this case, we send all the blocks which contain elements from S_1 to table T_1. We send the coinciding block of the element from S_2 to table T_0 and the rest of the blocks, which lies on the line containing this block to table T_1. If after this other nonempty block having an element from S_2 is still unassigned, then we send it to table T_1, and all the empty blocks lying on the line containing this block to table T_0 . Rest all the empty blocks are sent to table T_0 .

Case 3.1.3 None of the blocks which contain an element from S_2 coincides with the block, which contains an element from S_1 in table T_1 . In this case, we send all the

nonempty blocks to table T_1 and all the empty blocks to table T_0.

Case 3.2 Two blocks that contain elements from S_1 lies on the same line, and another lie on a different line.

Case 3.2.1 All the blocks which contain an element from S_1 lies in the same slice. From here onward, whenever we say blocks belonging to a slice, we mean the slice drawn in a superblock to which these blocks belong.

Case 3.2.1.1 All the blocks which contain elements from S_2 coincides with blocks which contain elements from S_1 in table T_1. In this case, we can use the assignment made in Case 3.1.1.

Case 3.2.1.2 Only one block which contains an element from S_2 coincides with the block, which contains an element from S_1 in table T_1. In this case, we can use the assignment made in Case 3.1.2

Case 3.2.1.3 None of the blocks which contain an element from S_2 coincides with the block, which contains an element from S_1 in table T_1. This case is the same as Case 3.1.3.

Case 3.2.2 Two blocks that contain elements from S_1 lies in a slice and another block that contains an element from S_1 in another slice.

Case 3.2.2.1 Both the blocks which contain elements from S_2 coincides with blocks which contain elements from S_1 in table T_1. If the coinciding blocks lie in the same slices as that of two blocks that contain elements from S_1, then we send both the coinciding blocks which contain the elements from S_2 to table T_0. Further, we send all the empty blocks lying on the lines which contain these blocks to table T_1. Two blocks that contain the elements from S_1 and lying in the same slice are sent to table T_1. The remaining block which contains the element is sent to table T_0 and all the empty blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

Now consider the case in which the coinciding blocks which contain the elements

from S_2 lies in different slices of the superblock, which contains the elements from S_1. In this case, we send the two blocks which contain the elements from S_1 lying in a slice to table T_1. The rest of the block, which contains the element from S_1 lying in the other slice, is sent to table T_0, and the empty blocks which lie on the line containing this block are sent to table T_1. One of the blocks which contains an element from S_2 and is lying in the slice containing two elements from S_1 is sent to table T_0 and the rest of the blocks on this line is sent to table T_1. If after this assignment, other block having the element from S_2 is still unassigned, then we send it to table T_1, and the empty blocks on the line containing this block to table T_0. Rest all the empty blocks are sent to table T_0.

Case 3.2.2.2 Only one block which contains an element from S_2 coincides with a block that contains an element from S_1 in table T_1. Let us first consider the case where coinciding block having element form S_2 lies in the slice, which contains two blocks having elements from S_1. Without loss of generality, let us say that the blocks having elements n_1 and n_2 lies in the same slice and the block having the element n_4 coincide with the block having the element n_1. In this case, we send the block having the element n_4 to table T_0, and all the blocks on the line containing this block is sent to table T_1.

If the block which contains an element n_3 lies on the line which contains the block having the element n_4, then we send the block having n_3 to table T_0, and empty blocks of the line which contains block having n_3 to table T_1. Now, if the block having the element n_5 is still unassigned, then we send the block having the element n_5 to table T_0, and empty blocks on the line containing this block to table T_1. We send the block having the element n_2 to table T_0, and all the blocks which lie on the line containing this block table T_1. Rest all the empty blocks are sent to table T_0.

Further, let us consider the case where the block that contains the element n_3 does not lie on the line, which contains the block having the element n_4. In this case, we send the block having n_3 to table T_1, and the empty blocks on the line containing this block are sent to table T_0. Now, if the block having the element n_5 is still unassigned, then we send it to table T_0, and the empty blocks lying on the line containing this block to table T_1. We send the block having the element n_2 to table T_0, and all the blocks which lie on the line containing this block table T_1. Rest all the empty blocks are sent to table T_0.

Now we are left with the case where coinciding block of S_2 having an element n_4

coincides with the block having the element n_3. In this case, we send the block having the element n_3 to table T_0 and empty block, which lies on the line containing this block to table T_1. We send the block having the element n_4 to table T_1 and empty blocks which lies on the line containing this block to table T_0.

Now we see the position of the block having the element n_5. If the block having the element n_5 lies on the line, which contains block having the element n_3, then we send the block having the element n_5 to table T_0. Further, we send the empty blocks of the line, which contains block having the element n_5 to table T_1. Now, consider the case where the line containing the block having the element n_5 passes through one of the blocks having the element n_1 or n_2. Without loss of generality, let us say that the line containing the block having the element n_5 passes through the block having the element n_1. In this case, we send the block having the element n_1 to table T_0 and rest all the blocks lying on this line to table T_1. Rest all the empty blocks are sent to table T_0. If the line which contains the block having element n_5 does not pass through the block block having element n_1 or n_2, in this case we can send both the blocks having elements n_1 and n_2 to table T_1. Rest all the empty blocks are sent to table T_0.

Now consider the case where block having the element n_5 does not lie on the line, which contains block having the element n_3. In this case, we send the block having the element n_5 to table T_1 and all the empty blocks lying on the line containing this block to table T_0. Blocks having elements n_1 and n_2 are sent to table T_1, and rest all the empty blocks are sent to table T_0.

Case 3.2.2.3 None of the blocks which contain elements from S_2 coincide with blocks having elements from S_1 in table T_1. This case is the same as Case 3.1.3.

Case 3.3 All the blocks which contain elements from S_1 lies on the different lines.

Case 3.3.1 Both the blocks having elements n_4 and n_5 coincides with the blocks having elements from S_1 in table T_1. Without loss of generality, let us say that block having element n_1, coincides with the block having the element n_4, and the block having the element n_2 coincide with the block having the element n_5. In this case, we send the blocks having elements n_1, n_2 and n_3 to table T_0 and all the empty blocks lying on the

lines which contain these blocks to table T_1. Further, we send the blocks having the element n_4 and n_5 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 3.3.2 Only one of the block having element say n_4 from S_2 coincides with blocks having element from S_1 in table T_1. Without loss of generality, let us say that block having the element n_1 coincides with the block having the element n_4. Similar to the last case, in this case also, we send the blocks having elements n_1, n_2 and n_3 to table T_0, and all the empty blocks lying on the lines which contain these blocks to table T_1. Further, We send the block having element n_4 to table T_1. We send the block having element n_5 to table T_0, and all the empty blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

Case 3.3.3 None of the blocks which contain an element from S_2 coincides with the block, which contains an element from S_1. This case is the same as Case 3.1.3.

Correctness: The correctness of the scheme relies on the fact that blocks having the elements do not coincide in table T_1 or in table T_0. Also, the blocks which are not having the elements are not sent to the place where block having elements are mapped.

We summaries the conclusion of this section as follows.

Theorem 6.3. *There is a two probe explicit adaptive scheme which stores an arbitrary subset $\mathcal{S}$ of size at most five from a universe $\mathcal{U}$ of size* m *and uses $\mathcal{O}(m^{10/11})$ bits of space.*

6.2.4 Counterexample

In this subsection, we will show that the above-mentioned scheme can not store a subset $\mathcal{S}$ of size six. Let us consider a subset $\mathcal{S} = \{n_1, n_2, n_3, n_4, n_5, n_6\}$ of six elements from a universe $\mathcal{U}$ of size m. Further, let us consider that all these elements belong to the different blocks. As shown in the Figure 6.5, blocks having the elements n_1, n_2, n_3 and n_4 lies on the line L_1, L_2, L_3 and L_4 respectively. Whereas, blocks having the elements n_5 and n_6 lies on the line L_5. Now, we claim that if we store the configuration shown in the

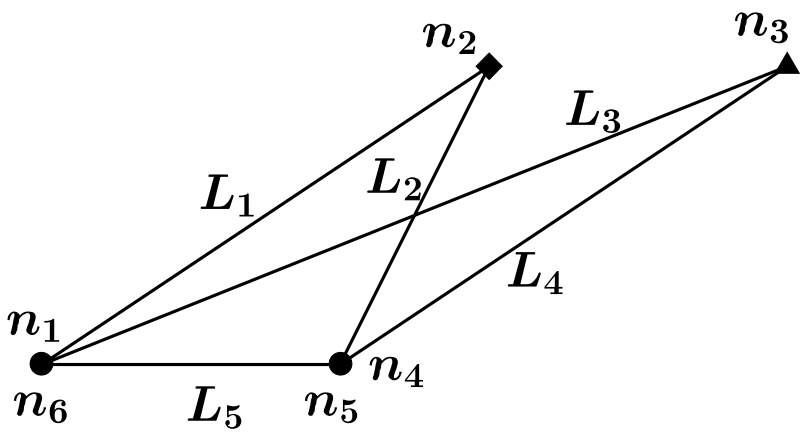


Figure 6.5: A counterexample for a six elements subset

Figure 6.5 in our $(5, m, m^{10/11}, 2)$-scheme then our query scheme will answer incorrectly.

To show that our claim is true, let us consider the block having the element n_1. Now, this block can either go to table B or to table C. Let us first consider the case where the block having the element n_1 goes to table B.

If the block having the element n_1 goes to table B, then rest all the blocks lying on the line L_1 must go to table C. Therefore, the block having the element n_2 must go to table B. Now, rest all the blocks lying on the line L_2 must go to table C. Therefore, blocks having the elements n_4 and n_5 must go to table B. If the block having the element n_4 goes to table B, then rest all the blocks lying on the line L_4 must go to table C. Therefore, the block having the element n_3 must go to table B. Now, rest all the blocks lying on the line L_3 must go to table C. Therefore, block having the element n_6 must go to table B. Since we have already sent the block having the element n_5 to table B, so our query scheme will answer incorrectly for the queries belonging to these blocks.

Now, we are left with the case where the block having the element n_1 goes to table C. In this case, the block having the element n_6 must go to table B. Therefore, rest all the blocks lying on the line L_5 must go to table C. Now, the block having the element n_4 must go to table B. Therefore, rest all the blocks lying on the line L_4 must go to table C. It forces the block having the element n_3 to table B. Therefore, rest all the blocks lying on the line L_3 must go to table C. However, we have already sent the block having the element n_1 to table C. So our query scheme will answer incorrectly for the queries belonging to the block having the element n_1 and the empty block lying on the

line L_3 and coinciding with the block having the element n_1.

The block containing the element n_1 can either go to table B or to table C, and our query scheme answers incorrectly in both the cases. Therefore, the above configuration of the elements cannot be stored in the $(5, m, m^{10/11}, 2)$-scheme.

6.3 Conclusion

In this chapter we have come up with an explicit adaptive $(5, m, \mathcal{O}(m^{10/11}), 2)$-scheme, which improves upon the non-explicit scheme by Garg and Radhakrishnan [15] and the explicit scheme by Garg [16] for the given set and query sizes. We have borrowed the idea of the geometrical arrangement of elements on the three-dimensional cube from Kesh [18] and the idea of dividing the universe into blocks and superblocks from Radhakrishnan et al. [2]. Using these ideas, there are improved schemes for the set of size three, four, and five. We believe that this technique can be extended to store larger subsets by extending the idea of an arrangement in a three-dimensional cube to arrangements in the higher dimensional cube.

"The main hurdle in proving a lower bound is the existence of an algorithm."

–Steven Rudich

7

Improved Bounds for Two Bitprobe Scheme Storing Five Elements

7.1 Introduction

A basic goal of this chapter is to reduce the gap between upper and lower bound for the two adaptive bitprobe schemes storing five elements ($n = 5$). To improve the lower bound, we extend the idea of first-order universe of an element e of a *set* by Kesh [17] to a *second order* universe. Whereas to improve the upper bound, we use the geometrical technique described in earlier chapters in more clever ways.

Data structures with two bitprobes ($t = 2$) consist of three tables, namely $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$. The first bitprobe is always made in table $\mathcal{A}$; the location of the bit being probed, of course, depends on the element which is being queried. The second bitprobe is made in table $\mathcal{B}$ or in table $\mathcal{C}$, depending on whether 0 was returned in the first bitprobe or 1 was returned. The final answer of the query scheme is `Yes` if 1 is returned by the second bitprobe, otherwise it is `No`. The data structure and the query scheme can be succintly denoted diagrammatically by what is known as the *decision tree* of the scheme (Figure 7.1).

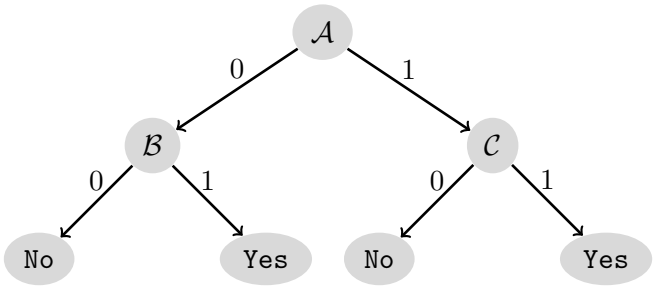


Figure 7.1: The decision tree of an element.

7.1.1 Our Contribution

We study schemes when the number of allowed bitprobes is two ($t = 2$) and the subset size is at most five ($n = 5$). Some progress has been made for subsets of smaller sizes.

When the subset size is odd, the problem is well understood. More particularly, when $n = 1$, there exists a scheme that takes $\mathcal{O}(m^{1/2})$ amount of space, and it has been shown that it matches with the lower bound $\Omega(m^{1/2})$ [1, 9, 20]. When $n = 3$, Baig and Kesh [3] have shown that there exists a $\mathcal{O}(m^{2/3})$-scheme, and Kesh [17] has proven that it matches with the lower bound $\Omega(m^{2/3})$.

For even sized subsets, tight bounds are yet to be proven. For $n = 2$, Radhakrishnan *et al.* [2] have proposed a scheme that takes $\mathcal{O}(m^{2/3})$ space, and for $n = 4$, Baig *et al.* [6] have presented a $\mathcal{O}(m^{5/6})$-scheme, but it is as of yet unknown whether these bounds are tight.

For subsets of size five ($n = 5$), the best known lower bound was due to Alon and Feige [9] which is $\Omega(m^{1/2})$. The $\Omega(m^{2/3})$ lower bound for $n = 3$ also puts an improved bound for the $n = 5$ case. Our first result improves the bound to $\Omega(m^{3/4})$.

Result 7.1 (Theorem 7.3)**.** $s_A(5, m, 2) = \Omega(m^{3/4})$.

We also propose an improved scheme for the problem. The best known upper bound was due to Garg [16] which was $\mathcal{O}(m^{18/19})$, which was improved by Baig *et al.* [7] to $\mathcal{O}(m^{10/11})$. In this paper, we improve the bound to $\mathcal{O}(m^{5/6})$.

Result 7.2 (Theorem 7.4)**.** $s_A(5, m, 2) = \mathcal{O}(m^{5/6})$.

One thing to note is that the space for the scheme storing five elements now matches the space for the scheme storing four elements. Moreover, the two results stated above combined together significantly reduces the gap between the upper and lower bounds for the problem under consideration.

7.2 Lower Bound

In this section, we present our proof for the lower bound of $s_A(5, m, 2)$. So, the size of our subset $\mathcal{S}$ that we want to store in our data structure is at most five.

In table $\mathcal{A}$ of our data structure, multiple elements must necessarily map to the same bit to keep the table size to $o(m)$. The set of elements that map to the same bit in this table is referred to in the literature as a *block* (Radhakrishnan *et al.* [2]). We refer by $\mathcal{A}(e)$ to the block to which the element e belongs. Elements mapping to the same bit in tables $\mathcal{B}$ and $\mathcal{C}$ will be referred to as just *sets*. That set of table $\mathcal{B}$ to which the element e belongs will be denoted by $\mathcal{B}(e)$. $\mathcal{C}(e)$ is similarly defined.

Storing a member e of our subset $\mathcal{S}$ in table $\mathcal{B}$ is an informal way to state the following – the bit corresponding to $\mathcal{A}(e)$ is set to 0, and $\mathcal{B}(e)$ is set to 1. So, upon query for the element e, we will get a 0 in our first bitprobe, query table $\mathcal{B}$ at location $\mathcal{B}(e)$ to get 1, and finally answer `Yes`. Similarly, storing an elememt f which is not in $\mathcal{S}$ in table $\mathcal{C}$ would entail assigning 1 to $\mathcal{A}(f)$ and 0 to $\mathcal{C}(f)$.

To start with, we make the following simplifying assumptions about any scheme for the aforementioned problem.

1. All the tables of our datastructure have the same size, namely s, and hence the size our data structure is $3 \times s$.
2. If two elements belong to the same block in table $\mathcal{A}$, they do not belong to the same sets in either of tables $\mathcal{B}$ or $\mathcal{C}$.

In the conclusion of this section, we will show that these assumptions do not affect the space asymptotically, but rather by constant factors.

7.2.1 Universe of an Element

We now define the notion of the *universe* of an element. This is similar to the definition of the universe of a set in Kesh [17].

Definition 7.1. *The universe of an element e w.r.t. to table $\mathcal{B}$, denoted by $\mathcal{U}_{\mathcal{B}}(e)$, is defined as follows.*

$$\mathcal{U}_{\mathcal{B}}(e) = \bigcup_{f \in \mathcal{B}(e) \setminus \{e\}} \mathcal{A}(f) \setminus \{f\}.$$

Similarly, the universe of an element e w.r.t. to table $\mathcal{C}$, denoted by $\mathcal{U}_{\mathcal{C}}(e)$, is defined as follows.

$$\mathcal{U}_{\mathcal{C}}(e) = \bigcup_{f \in \mathcal{C}(e) \setminus \{e\}} \mathcal{A}(f) \setminus \{f\}.$$

A simple property of the universe of an element, which will be useful later, is the following.

Observation 7.1. *1. $\mathcal{A}(e) \cap \mathcal{U}_{\mathcal{B}}(e) = \phi$ and $\mathcal{B}(e) \cap \mathcal{U}_{\mathcal{B}}(e) = \phi$.*

2. $\mathcal{A}(e) \cap \mathcal{U}_{\mathcal{C}}(e) = \phi$ and $\mathcal{C}(e) \cap \mathcal{U}_{\mathcal{C}}(e) = \phi$.

Proof. Due to Assumption 2, e is the only element of the block $\mathcal{A}(e)$ in set $\mathcal{B}(e)$. So, it follows from the definition that no element of $\mathcal{A}(e)$ is part of $\mathcal{U}_{\mathcal{B}}(e)$. No element of $\mathcal{B}(e)$ is part of $\mathcal{U}_{\mathcal{B}}(e)$ as we specifically preclude those elements from $\mathcal{U}_{\mathcal{B}}(e)$. The other scenarios can be similarly argued. □

We make the following simple observations about the universe of an element to help illustrate the constraints any storage scheme must satisfy to correctly store a subset $\mathcal{S}$.

Observation 7.2. *If $\mathcal{B}(e) \cap \mathcal{S} = \{e\}$, and we want to store e in table $\mathcal{B}$, then all the elements of $\mathcal{U}_{\mathcal{B}}(e)$ must be stored in table $\mathcal{C}$.*

Proof. As e has been stored in table $\mathcal{B}$, the bit corresponding to the set $\mathcal{B}(e)$ must be set to 1. Consider an element f, different from e, in $\mathcal{B}(e)$. According to Assumption 2, e and f cannot belong to the same block. As $f \notin \mathcal{S}$, so f cannot be stored in table $\mathcal{B}$; if we do so, the query for f will look at the bit $\mathcal{B}(e)$ and incorrectly return 1. So, the element f and its block $\mathcal{A}(f)$ must be stored in table $\mathcal{C}$.

The above argument applies to any arbitrary element of $\mathcal{B}(e) \setminus \{e\}$. So, according to Definition 7.1, all of the elements of $\mathcal{U}_{\mathcal{B}}(e)$ must be stored in table $\mathcal{C}$. □

We make the same observation, without proof, in the context of table $\mathcal{C}$.

Observation 7.3. *If $\mathcal{C}(e) \cap \mathcal{S} = \{e\}$, and we want to store e in table $\mathcal{C}$, then all the elements of $\mathcal{U}_{\mathcal{C}}(e)$ must be stored in table $\mathcal{B}$.*

Next, we define, what could be referred to as, a higher-order universe of an element, built on top of the universe of the element.

Definition 7.2. *The 2-universe of an element e w.r.t. table $\mathcal{B}$, denoted by $\mathcal{U}^2_{\mathcal{B}}(e)$, is defined as follows.*

$$\mathcal{U}^2_{\mathcal{B}}(e) = \bigcup_{f \in \mathcal{U}_{\mathcal{B}}(e)} \mathcal{C}(f) \setminus \{f\}.$$

Similarly, the 2-universe of an element e w.r.t. table $\mathcal{C}$, denoted by $\mathcal{U}^2_{\mathcal{C}}(e)$, is defined as follows.

$$\mathcal{U}^2_{\mathcal{C}}(e) = \bigcup_{f \in \mathcal{U}_{\mathcal{C}}(e)} \mathcal{B}(f) \setminus \{f\}.$$

The following observations provide more constraints for our storage schemes.

Observation 7.4. *Consider an element e such that $\mathcal{B}(e) \cap \mathcal{S} = \{e\}$, and suppose we want to store e in table $\mathcal{B}$. If f is a member of $\mathcal{U}_{\mathcal{B}}(e)$ such that $\mathcal{C}(f) \cap \mathcal{S} = \{f\}$, then all the other members of $\mathcal{C}(f)$ must be stored in table $\mathcal{B}$.*

Proof. If e, a member of $\mathcal{S}$, is stored in table $\mathcal{B}$, then Observation 7.2 tells us that all members of $\mathcal{U}_{\mathcal{B}}(e)$ must be stored in table $\mathcal{C}$. As $f \in \mathcal{U}_{\mathcal{B}}(e) \cap \mathcal{S}$, so f must be stored in table $\mathcal{C}$, and consequently, the bit corresponding to $\mathcal{C}(f)$ must be set to 1. As the other members of $\mathcal{C}(f)$ do not belong to $\mathcal{S}$, they cannot be stored in table $\mathcal{C}$, and hence must be stored in table $\mathcal{B}$. □

Observation 7.5. *Consider an element e such that $\mathcal{B}(e) \cap \mathcal{S} = \{e\}$, and suppose we want to store e in table $\mathcal{B}$. If f is a member $\mathcal{U}_{\mathcal{B}}(e)$ such that $\mathcal{C}(f) \cap \mathcal{S} = \{x\}$, where $x \neq f$, then x must be stored in table $\mathcal{B}$.*

Proof. As e, a member of $\mathcal{S}$, is stored in $\mathcal{B}$, Observation 7.2 tells us that all the members of $\mathcal{U}_{\mathcal{B}}(e)$, and f in particular, must be stored in table $\mathcal{C}$. As $f \notin \mathcal{S}$, so the bit corresponding to $\mathcal{C}(f)$ has to be 0. As $x \in \mathcal{C}(f)$ belongs to $\mathcal{S}$, then storing x in table $\mathcal{C}$ would imply that $\mathcal{C}(f)$ must be set to 1, which is absurd. So, x must be stored in table $\mathcal{B}$. □

We next state the same observations in the context of table $\mathcal{C}$.

Observation 7.6. *Consider an element e such that $\mathcal{C}(e) \cap \mathcal{S} = \{e\}$, and suppose we want to store e in table $\mathcal{C}$. If f is a member of $\mathcal{U}_{\mathcal{C}}(e)$ such that $\mathcal{B}(f) \cap \mathcal{S} = \{f\}$, then all the other members of $\mathcal{B}(f)$ must be stored in table $\mathcal{C}$.*

Observation 7.7. *Consider an element e such that $\mathcal{C}(e) \cap \mathcal{S} = \{e\}$, and suppose we want to store e in table $\mathcal{C}$. If f is a member $\mathcal{U}_{\mathcal{C}}(e)$ such that $\mathcal{B}(f) \cap \mathcal{S} = \{x\}$, where $x \neq f$, then x must be stored in table $\mathcal{C}$.*

7.2.2 Bad Elements

We now define the notion of *good* and *bad* elements. These notions are motivated by the notions of *large* and *bounded sets* from Kesh [17].

Definition 7.3. *e is a bad element w.r.t. table $\mathcal{B}$ if one of the following holds.*

1. *Two elements of $\mathcal{U}_{\mathcal{B}}(e)$ share a set in table $\mathcal{C}$.*
2. *The size of $\mathcal{U}^2_{\mathcal{B}}(e)$ is greater than $2s$, i.e. $|\mathcal{U}^2_{\mathcal{B}}(e)| > 2 \cdot s$.*

Otherwise, it is said to be good. Bad and good elements w.r.t. to table $\mathcal{C}$ are similarly defined.

The next claims state the consequences of an element being bad due to any of the above properties getting satisfied.

Claim: If two elements of $\mathcal{U}_{\mathcal{B}}(e)$ share a set in table $\mathcal{C}$, then $\exists$ a subset $\mathcal{S}$ that contains e and has size two such that to store $\mathcal{S}$, e cannot be stored in $\mathcal{B}$.

Proof. Suppose the elements $x, y \in \mathcal{U}_{\mathcal{B}}(e)$ share the set Y in table $\mathcal{C}$. We would prove that to store the subset $\mathcal{S} = \{e, x\}$, e cannot be stored in table $\mathcal{B}$.

Let us say that e indeed can be stored in table $\mathcal{B}$. According to Observation 7.2, all elements of $\mathcal{U}_{\mathcal{B}}(e)$, including x and y, must be stored in table $\mathcal{C}$. As $x \in \mathcal{S}$, the bit corresponding to the set Y must be set to 1. As $y \notin \mathcal{S}$, the bit corresponding to Y must be set to 0. We thus arrive at a contradiction. So, e cannot be stored in table $\mathcal{B}$. □

Claim:

If the size of $\mathcal{U}^2_{\mathcal{B}}(e)$ is greater than $2s$, then $\exists$ a subset $\mathcal{S}$ that contains e and has size at most three such that to store $\mathcal{S}$, e cannot be stored in table $\mathcal{B}$.

Proof. Consider the set of those elements f of $\mathcal{U}^2_{\mathcal{B}}(e)$ such that it is the only member of its block to belong to $\mathcal{U}^2_{\mathcal{B}}(e)$. As there are a total of s blocks, there could be at most s such elements. Removing those elements from $\mathcal{U}^2_{\mathcal{B}}(e)$ still leaves us with more than s elements in $\mathcal{U}^2_{\mathcal{B}}(e)$. These remaining elements have the property that there is at least one other element from its block that is present in $\mathcal{U}^2_{\mathcal{B}}(e)$. Let this set be denoted by Z.

As the size of Z is larger than the size of table $\mathcal{B}$, there must exist at least two elements $x, y \in Z$ that share a set X in table $\mathcal{B}$. According to Definition 7.2, this implies

that there exists elements $z, z' \in \mathcal{U}_{\mathcal{B}}(e)$ such that $x \in \mathcal{C}(z) \setminus \{z\}$ and $y \in \mathcal{C}(z') \setminus \{z'\}$. It might very well be that $z = z'$.

If $x \in \mathcal{A}(e)$, as e has been stored in table $\mathcal{B}$, so all the elements of $\mathcal{A}(e)$, including x, must have been stored in table $\mathcal{B}$. Consider the subset $\mathcal{S} = \{e, x, z'\}$. As $x \in \mathcal{S}$, the bit corresponding to set X must be set to 1. As e is stored in table $\mathcal{B}$, Observation 7.2 tells us that $z' \in \mathcal{U}_{\mathcal{B}}(e)$ must be stored in table $\mathcal{C}$. As $\mathcal{C}(z') \cap \mathcal{S} = \{z'\}$, Observation 7.4 tells us that y must be stored in table $\mathcal{B}$. So, the bit corresponding to set X must be set to 0, which is absurd. So, to store $\mathcal{S}$, e cannot be stored in table $\mathcal{B}$. This argument holds even if $x = e$. Similar is the case if $y \in \mathcal{A}(e)$.

If $x \in \mathcal{B}(e) \setminus \{e\}$, and as e has been stored in table $\mathcal{B}$, Observation 7.2 tells us that x must be stored in table $\mathcal{C}$. Consider the subset $\mathcal{S} = \{e, z\}$. Observation 7.4 tells us that as $z \in \mathcal{U}_{\mathcal{B}}(e)$ is in $\mathcal{S}$, $x \in \mathcal{C}(z)$ cannot be stored in table $\mathcal{C}$, which is absurd. So, to store $\mathcal{S}$, e cannot be stored in table $\mathcal{B}$. We can similarly argue the case $y \in \mathcal{B}(e)$.

We now consider the case when $x, y \notin \mathcal{A}(e)$ and $\notin \mathcal{B}(e)$. If $\mathcal{S}$ contains e and x, and we store e in table $\mathcal{B}$, Observation 7.2 tells us that $z \in \mathcal{U}_{\mathcal{B}}(e)$ must be stored in table $\mathcal{C}$, and as $x \in \mathcal{C}(z)$, Observation 7.5 tells us that x must be stored in table $\mathcal{B}$. As $x \in \mathcal{S}$, hence the bit corresponding to set of x in table $\mathcal{B}$, which is X, must be set to 1.

If $z \neq z'$, we include z' in $\mathcal{S}$, and according to Observation 7.4, $y \in \mathcal{C}(z')$ must be stored in table $\mathcal{B}$. As $y \in X$ is not in $\mathcal{S}$, X must be set to 0, and we arrive at a contradiction for the subset $\mathcal{S} = \{e, x, z'\}$.

It could also be the case that $z = z'$. As $y \in Z$, there exists an element $y' \in \mathcal{A}(y) \cap \mathcal{U}^2_{\mathcal{B}}(e)$. Let $y' \in C(z'')$, where $z'' \in \mathcal{U}_{\mathcal{B}}(e)$. In this scenario, we consider storing the subset $\mathcal{S} = \{e, x, z''\}$. As, $z'' \in \mathcal{S} \cap \mathcal{U}_{\mathcal{B}}(e)$, and $y' \notin \mathcal{S}$, Observation 7.4 implies that y', and hence the whole of block $\mathcal{A}(y')$, including y, must be stored in table $\mathcal{B}$. As $y \notin \mathcal{S}$, the set of y in table $\mathcal{B}$, which is X, must be set to 0, and we again arrive at a contradiction.

So, we conclude that e in either of the cases cannot be stored in table $\mathcal{B}$. □

The two claims above imply the following – if an element e is bad w.r.t. table $\mathcal{B}$ (Definition 7.3) due to Property 1, or if this property does not hold but Property 2 does, then there exists a subset, say $\mathcal{S}_1$, of size at most three containing e such that to store $\mathcal{S}_1$, e cannot be stored in table $\mathcal{B}$. The claims above also hold w.r.t. table $\mathcal{C}$. So, we can claim that if e is bad w.r.t. to table $\mathcal{C}$, then there exists a subset $\mathcal{S}_2$ containing e of size

at most three such that to store $\mathcal{S}_2$, e cannot be stored in table $\mathcal{C}$.

Consider the set $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$. As e is common in both the subsets, size of $\mathcal{S}$ is at most five. If e is bad w.r.t. to table $\mathcal{B}$ and table $\mathcal{C}$, then to store subset $\mathcal{S}$, we cannot store e in either of the tables, which is absurd. We summarise the discussion in the following lemma.

Lemma 7.1. *If an element e is bad w.r.t. $\mathcal{B}$, then it must be good w.r.t $\mathcal{C}$.*

7.2.3 Good Schemes

Based on the above lemma, we can partition our universe $\mathcal{U}$ into two sets $\mathcal{U}_1$ and $\mathcal{U}_2$ – one that contains all the good elements w.r.t. to table $\mathcal{B}$, and one that contains the bad elements. We now partition each block and each set of the three tables of our datastructure into two parts, one containing elements from $\mathcal{U}_1$, and one containing the elements from $\mathcal{U}_2$. For elements of $\mathcal{U}_1$, only those blocks and sets that contain elements of $\mathcal{U}_1$ will be affected; similarly for the elements of $\mathcal{U}_2$.

In effect, we have two independent schemes, one for $\mathcal{U}_1$ and one for $\mathcal{U}_2$. In the scheme for $\mathcal{U}_1$, all the elements in table $\mathcal{B}$ are good. In the scheme for $\mathcal{U}_2$, all the elements in table $\mathcal{B}$ are bad, and consequently, Lemma 7.1 tells us that all the elements of table $\mathcal{C}$ are good. In the scheme for $\mathcal{U}_2$, we now relabel the table $\mathcal{B}$ to $\mathcal{C}$ and relabel the table $\mathcal{C}$ to $\mathcal{B}$. To make the new scheme for $\mathcal{U}_2$ work, we now have to store 0 in the blocks of table $\mathcal{A}$ for $\mathcal{U}_2$ when earlier we were storing 1, and have to store 1 when earlier we were storing 0.

This change gives us a new scheme with two important properties – the size of the datastructure has doubled from the earlier scheme, and all the elements in table $\mathcal{B}$ are now good.

Lemma 7.2. *Given a $(5, m, s, 2)$-scheme, we can come up with a $(5, m, 2 \times s, 2)$-scheme such that all the elements of $\mathcal{U}$ are good w.r.t. to table $\mathcal{B}$ in the new scheme.*

7.2.4 Space Complexity

Consider a $(5, m, 3 \times s, 2)$-scheme all of whose elements are good w.r.t. table $\mathcal{B}$. The table sizes then are each equal to s. According to Lemma 7.2, the 2-universe of each

element w.r.t. to $\mathcal{B}$ will be at most $2s$. So, the sum total of all the 2-universe sizes of all the elements is upper bounded by $m \times 2s$.

We now consider how much each set of table $\mathcal{C}$ contribute to the total. From Definition 7.2, we have the following –

$$\sum_{e \in \mathcal{U}} | \mathcal{U}^2_{\mathcal{B}}(e) | = \sum_{e \in \mathcal{U}} | \bigcup_{f \in \mathcal{U}_{\mathcal{B}}(e)} \mathcal{C}(f) \setminus \{f\} | = \sum_{e \in \mathcal{U}} \left(\sum_{f \in \mathcal{U}_{\mathcal{B}}(e)} | \mathcal{C}(f) \setminus \{f\} | \right).$$

As all the elements are good, and hence for every element e, no two elements of $\mathcal{U}_{\mathcal{B}}(e)$ share a set in table $\mathcal{C}$, we can thus convert the union in Definition 7.2 to summation.

We have shown in Appendix 8.1 that

$$\sum_{e \in \mathcal{U}} | \mathcal{U}^2_{\mathcal{B}}(e) | \geq \ c \cdot \frac{m^4}{s^3},$$

for some constant c. The proof show that the minimum value is achieved when all the blocks and the sets in the three tables are of the same size, i.e. m/s. This combined with the upperbound for total sum of the sizes of all 2-universes gives us

$$c \cdot \frac{m^4}{s^3} \leq m \times 2s.$$

Resolving the equation gives us

$$s = \Omega(m^{3/4}).$$

This bound applies to good schemes that respect the two assumptions declared at the beginning of this section.

Suppose we have an arbitrary adaptive $(5, m, s, 2)$-scheme. If we want to make all the tables in this scheme of the same size, we can add extra bits which will make the size of the data structure at most $3 \cdot s$. So, we get a $(5, m, 3 \times s, 2)$-scheme that respect Assumption 1.

In Kesh [17], sets which have multiple elements from the same block were referred to as *dirty sets*. *Clean sets* were those which contain elements from distinct blocks. It was shown in Section 3 that any scheme with dirty sets can be converted into a scheme with only clean sets by using twice amount of space. Though the final claim was made in context of $n = 3$, but the proof applies to any n. So, we can now have a $(5, m, 6 \times s, 2)$-

scheme that respects both of our assumptions.

Such a scheme can be converted into a scheme with only good elements in table $\mathcal{B}$ by using twice the amount of space as before. We now have a $(5, m, 12 \times s, 2)$-scheme, where the table sizes are all $4s$, and we have shown that $4s = \Omega(m^{3/4})$.

We summarise our discussion in the following theorem on the lower bound for two-adaptive bitprobes schemes storing five elements.

Theorem 7.3. $s_A(5, m, 2) = \Omega(m^{3/4})$.

7.3 Our Data Structure

In this section, we will present a scheme which stores an arbitrary subset of size at most five from a universe of size m, and answers the membership queries in two adaptive bitprobes. This scheme improves upon the $\mathcal{O}(m^{10/11})$-scheme by the authors [7], and is fundamentally different from that scheme in the way that here block sizes are nonuniform, and any two blocks in table $\mathcal{C}$ share at most one bit. As per the convention of that scheme, we will use the label T to refer to the table $\mathcal{A}$, T_0 to refer to the table $\mathcal{B}$, and T_1 to refer to the table $\mathcal{C}$.

Superblock: In this scheme, we have used the idea of Kesh [18] to map the elements on a square grid. Furthermore, we have used the idea of Radhakrishnan *et al.* [2] to divide the universe into blocks and superblocks. Our scheme divides the universe of size m into superblock of size x^2zt. Each superblock is made up of rectangular grids of size $t \times z$, and there are x^2 of them as shown in Figure 7.2. Further, each integral point on a grid represents a unique element.

Block: For the 1st superblock we draw lines with slope 1 as shown in Figure 7.3. Each line drawn represents a block. From Figure 7.3, we can see that some blocks are of equal size and some are of different size. We do this for all the superblocks, and hence partitioning the universe into blocks. For the ith superblock we draw lines with slope $1/i$.

Table T_1: This table has space equal to that of a single superblock, i.e., x^2zt. All the superblocks can be thought of as superimposed over each other in this table. Structure of this table is shown in Figure 7.2.

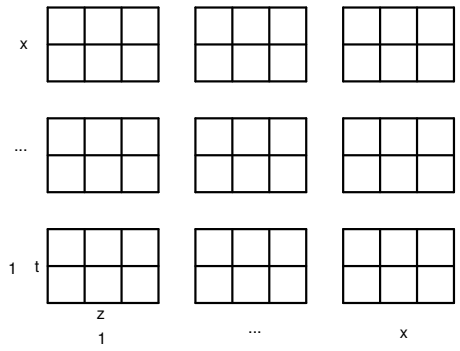


Figure 7.2: Figure showing structure of a superblock

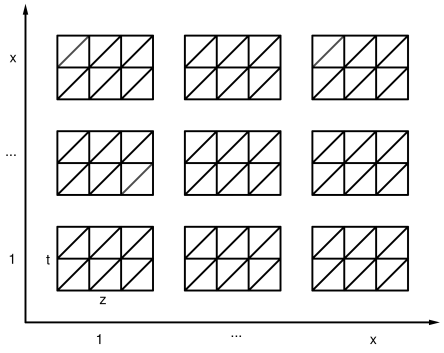


Figure 7.3: Lines drawn in the first superblock

Table T: In this table, we store a single bit for each block. Let there be n superblocks in total. Now let us concentrate on a single grid of Figure 7.3. The number of lines drawn for the ith superblock is equal to $z+c\cdot it$, where c is a constant. If we sum this for all the superblocks total number of lines drawn for the single grid will be equal to $nz + c \cdot n^2t$. Now, since there are $x \times x$ grids, the total number of lines drawn for all the superblocks will be $(nz + c \cdot n^2t)x^2$. As mentioned earlier, each line represents a block, and for each block, we have one bit of space in table T. So the size of this table T is $(nz+c\cdot n^2t)x^2$ bits.

Table T_0: In addition to lines drawn in superblocks to divide them into blocks, we also draw dotted lines in all the superblocks, as shown in Figure 7.4. For the ith superblock we draw dotted lines with slope $1/i$. Further, we store a block of size t in table T_0 for each dotted lines drawn. Now, we can see that for a specific superblock there could be many blocks belonging to that superblock which lies on the same dotted line. All the blocks which lie completely on the same dotted line query the same block in table T_0 kept for the dotted line. Now let us talk about the space taken by table T_0. Using the idea shown in Figure 7.4 to draw the dotted lines, if we sum the total number of dotted lines drawn for all the superblocks which pass through x-axis, we will get nzx. Further, if we sum the total number of dotted lines drawn for all the superblocks from the y-axis, we get it to be less than or equal to $c_1 \cdot n^2t \times x$, where c_1 is a positive constant. If we sum the total number of dotted lines drawn for all the superblocks from x and y-axis, we get $nzx + c_1 \cdot n^2t \times x$. Since we store a block of size t for each dotted line drawn, total space for table T_0 is $(nzx + c_1 \cdot n^2t \times x) \times t$.

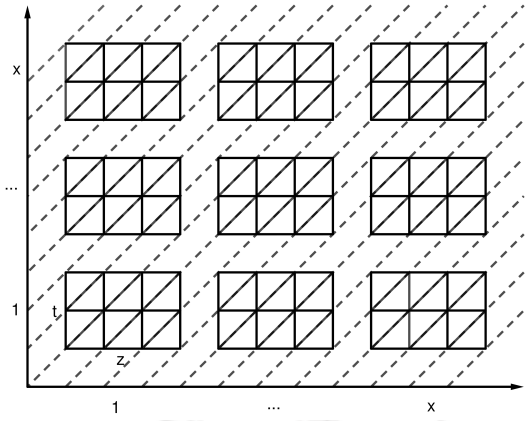


Figure 7.4: Dotted lines drawn for the first superblock

Size of data structure: Summing up the space taken by all the tables we get the following equation:

$$s(x, z, t) = x^2zt + (nz + c \cdot n^2t)x^2 + (nzx + c_1 \cdot n^2t \times x) \times t \tag{7.1}$$

As mentioned earlier size of each superblock is x^2zt, so the total number of superblocks are $n = m/(x^2zt)$. Substituting this in the above equation, we get the following:

$$s(x, z, t) = c_1 \cdot \frac{m^2}{x^3z^2} + c \cdot \frac{m^2}{x^2z^2t} + \frac{m}{x} + \frac{m}{t} + x^2zt \tag{7.2}$$

Choosing $x = t = m^{1/6}$ and $z = m^{2/6}$, we get the space taken by our data structure to be $\mathcal{O}(m^{5/6})$.

7.3.1 Query Scheme

Our query scheme has three tables T, T_0 and T_1. Given a query element, we first find out the blocks to which it belongs. Further, we query the bit stored for this block in table T. If the bit returned is zero, we make the next query to table T_0 otherwise to table T_1. We say that query element is part of the set given to be stored if and only if last bit returned is one.

7.3.2 Storage Scheme

Our storage scheme sets the bits of tables T, T_0 and T_1 to store an arbitrary subset of size at most five in such a way that membership queries can be answered correctly. Storage scheme sets the bit of data structure depending upon the distribution of elements in various superblocks. Distribution of elements into various superblocks leads to various cases of the storage scheme. While generating various cases we consider an arbitrary subset $S = \{n_1, n_2, n_3, n_4, n_5\}$ of size five given to be stored. Each block is either sent to Table T_0 or T_1, and we store its bit-vector there. While sending blocks to either T_0 or T_1, we make sure that no two blocks sharing a bit have conflicting bit common in either of the tables, the correctness of the scheme relies on this fact. Keeping in mind the space constraints, we have discussed a few cases in this section, and for the sake of completeness the rest of the cases which can be handled in a similar fashion are mentioned in the Appendix 8.2. Most of the cases generated and assignment made are similar to those generated in the previous paper on the problem by Baig *et al.* [7] to store an arbitrary subset of size at most five.

Case 1 All the elements belonging to S belongs to the same superblock. In this case, we send all the blocks having elements from the set given to be stored to table T_1. All the empty blocks, i.e., blocks which do not have any elements from S are sent to table T_0.

Case 2 Four elements $S_1 = \{n_1, n_2, n_3, n_4\}$ lies in one superblock and one $S_2 = \{n_5\}$ in other. In this case, we send the block having element n_5 to table T_1 and rest all the blocks belonging to superblock which contains this element to table T_0. All the blocks which are having conflicting bit common with the block having element n_5 are sent to table T_0. Remaining all the blocks of superblocks which contains elements from S_1 are sent to table T_1. Furthermore, rest all the empty blocks of all the superblocks are sent to table T_0.

Case 3 All the elements n_1, n_2, n_3, n_4 and n_5 lies in the different superblocks. In this case, we send all the blocks having elements to table T_0 and all the empty blocks to table T_1.

Case 4 Three elements $S_1 = \{n_1, n_2, n_3\}$ belong to one superblock and two elements $S_2 = \{n_4, n_5\}$ to the other.

Case 4.1 All the blocks to which elements from S_1 belong lies on the same dotted line of their superblock.

Case 4.1.1 Two blocks to which elements from S_2 belong have a conflicting bit common with the blocks corresponding to the elements from S_1 in table T_1. In this case, we send the blocks having elements from S_2 to table T_0. Further, we send empty blocks lying on the dotted lines to which blocks having elements from S_2 belongs to table T_1. We send all the blocks which contain elements from S_1 in table T_1. We send the rest of the empty blocks to table T_0.

Case 4.1.2 Only one block which contains an element from S_2 has a conflicting bit common with the block corresponding to the elements from S_1 in table T_1. In this case, we send all the blocks which contain elements from S_1 to table T_1. We send the block having an element from S_2, and having conflicting bit common with block having an element from S_1, to table T_0, and the rest of the blocks which lies on the dotted line containing this block to table T_1. If after this other nonempty block having an element from S_2 is still unassigned then we send it to table T_1, and all the empty blocks lying on the dotted line containing this block to table T_0 . Rest all the empty blocks are sent to table T_0.

Case 4.1.3 None of the blocks which contain an element from S_2 have a conflicting bit common with the block which includes an element from S_1 in table T_1 . In this case, we send all the nonempty blocks to table T_1 and all the empty blocks to table T_0.

Case 4.2 Two blocks which contain elements say n_1 and n_2 from S_1 lies on the same dotted line and other say n_3 lies on a different dotted line.

Case 4.2.1 All the blocks which contain elements from S_2 have a conflicting bit com-

mon with blocks which include elements from S_1 in table T_1.

4.2.1.1 Let us first consider the case where blocks having elements from S_2 have a conflicting bit common with the blocks having elements n_1 and n_2. In this case, we send the blocks having element n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these block to table T_1. Further, we send the blocks having elements n_1 and n_2 to table T_1. Block having element n_3 is sent to table T_0, and all the empty blocks lying on the dotted line containing this block is sent to table T_1. Rest all the empty blocks are sent to table T_0.

4.2.1.2 Without loss of generality let us now consider the case where blocks having an element from S_2 have a conflicting bit common with blocks having element n_1 and n_3. In this case, we send the blocks having elements n_4 and n_5 to table T_1, rest all the blocks lying on the dotted line(lines) containing these blocks to table T_0. Further, we send the blocks having element n_1 and n_3 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. Rest all the blocks are sent to table T_0.

4.2.1.3 Now we are left with a case where block having an element from S_2 have a conflicting bit common with blocks having elements n_1, n_2 and n_3. In this case, we send the blocks having element n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted lines containing this block to table T_1. Further, we send all the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

Case 4.2.2 Only one block having an element from S_2 have a conflicting bit common with the block(blocks) having an element(elements) from S_1.

Case 4.2.2.1 All the blocks having elements from S_2 lies on the same dotted line. Without loss of generality, let us say block having element n_4 from S_2 have a conflicting bit common with a block having an element from S_1. In this case, we send the blocks having element n_4 and n_5 to table T_1, and all the empty blocks lying on the dotted line containing these blocks to table T_0. Further, we send the block(blocks) having an element(elements) from S_1, and having a conflicting bit common with a block having

element n_4 to table T_0, and all the blocks lying on the dotted line containing this block to table T_1. We now send the rest of the block(blocks) having an element from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 4.2.2.2 Now let us consider a case where blocks having an element from S_2 lies on the different dotted line. Without loss of generality lets say block having element n_4 have a conflicting bit common with a block(blocks) having an element(elements) from S_1.

Let us first consider the case where block having element n_4 have a conflicting bit common with either block having element n_1 or n_2. Without loss of generality, let us say block having element n_4 have a conflicting bit common with a block having element n_1. In this case, we send the block having element n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted line containing these blocks to table T_1. Now, we see the positions of the blocks having elements n_1 and n_2. Let us first consider the case where blocks having element n_1 or n_2 have conflicting bit common with one of the empty blocks lying on the dotted line which contains block having element n_5. Without loss of generality let us say that block having element n_1 have conflicting bit common with one of the empty blocks lying on the dotted line which contains block having element 5. In this case, we send the block having element n_1 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if the block having element n_1 or n_2 do not have conflicting bit common with empty blocks lying on the dotted line which contains block having element n_5, then we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks lying on the dotted line containing these blocks to the table T_0. Rest all the empty blocks are sent to the table T_0.

Now let us consider the case where block having element n_4 have a conflicting bit common with a block having element n_3. Now, we see if the block having element n_4 have conflicting bit common with block having element n_1 or n_2. Without loss of generality, let us say that block having element n_4 have conflicting bit common with block having element n_1. In this case, we can use the assignment made in previous paragraph. On the other hand, if the block having element n_4 do not have conflicting bit common with block having element n_1 or n_2, then we send the blocks having elements n_3 and n_5

to table T_0, and all the empty blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the block having element n_4 to table T_1, and all the empty blocks lying on the dotted line containing this block to table T_0. Now we see the position of the blocks having elements n_1 and n_2. Let us first consider the case where on of the blocks having elements n_1 or n_2 has conflicting bit common with one of the empty blocks lying on the dotted line which contains block having element n_5. Without loss of generality, let us say that block having element n_1 has conflicting bit common with one of the empty blocks lying on the dotted line which contains block having element n_5. In this case, we send the block having element n_1 to table T_0, and all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if none of the block having element n_1 or n_2 has conflicting bit common with empty blocks lying on the dotted line which contains block having element n_5, then we send the block having element n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0.

Case 4.2.3 None of the blocks which contain an element from S_2 have a conflicting bit common with the block which includes an element from S_1 in table T_1 . In this case, we send all the nonempty blocks to table T_1 and all the empty blocks to table T_0.

Case 4.3 All the blocks having an element from S_1 lies on the different dotted lines.

Case 4.3.1 Both the blocks having elements n_4 and n_5 have a conflicting bit common with the blocks having elements from S_1 in table T_1. Now we see the positions of the blocks having elements n_4 and n_5. Blocks having elements n_4 and n_5 can either lie on the same dotted line or on the different dotted lines. If the blocks having elements n_4 and n_5 lie on the different dotted lines, then we send all the blocks having elements to table T_0, and all the empty blocks to table T_1. Now we consider the case where blocks having elements n_4 and n_5 lies on the same dotted line. Furthermore, without loss of generality let us consider that blocks having elements n_4 and n_5 conflicts with blocks having elements n_1 and n_2 respectively. In this case, we send the blocks having elements n_1 and n_2 to table T_0, and all the empty blocks lying on the dotted lines containing these blocks to the table T_1. Further, we send the blocks having elements n_4 and n_5 to

table T_1, and all the empty blocks lying on the dotted line containing these blocks to table T_1. Now, we see the position of the block having element n_3. If the block having element n_3 has conflicting bit common with blocks having elements n_4 or n_5, then we send the block having element n_3 to table T_0, and all the empty blocks lying on the dotted line which contains this block to table T_1. Rest all the empty blocks to table T_0. On the other hand, if the block having element n_3 do not have conflicting bit common with block having element n_4 or n_5, then we send the block having element n_3 to table T_1, and rest all the empty blocks to table T_0.

Case 4.3.2 Only one of the block having element say n_4 from S_2 have a conflicting bit common with blocks having an element from S_1 in table T_1. Similar to the last case, in this case also we see whether blocks having elements n_4 and n_5 lie on the same dotted line or on the different dotted lines. If the blocks having elements n_4 and n_5 lies on the different dotted lines, then we send the blocks having elements to table T_0, and all the empty blocks to table T_1. Now, we consider the case where blocks having elements n_4 and n_5 lies on the same dotted lines. Furthermore, without loss of generality let us say that block having the element n_1 have a conflicting bit common with the block having the element n_4. In this case, we send the blocks having elements n_1 and n_4 to table T_0, and all the empty blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the blocks having elements n_2, n_3 and n_5 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 4.3.3 None of the blocks which contain an element from S_2 have a conflicting bit common with the block which includes an element from S_1. This case is the same as Case 4.1.3.

We conclude this section with the following theorem:

Theorem 7.4. *There is a fully explicit two adaptive bitprobe scheme, which stores an arbitrary subset of size at most five, and uses $\mathcal{O}(m^{5/6})$ space.*

7.4 Conclusion

In this paper, we have studied those schemes that store subsets of size at most five and answer membership queries using two adaptive bitprobes. Our first result improves upon the known lower bounds for the problem by generalising the notion of universe of sets in Kesh [17] to what may be referred to as *second order* universe. We hope that suitably defining still higher order universes will help address the lower bounds for subsets whose sizes are larger than five. Though the lower bound of $\Omega(m^{3/4})$ is an improvement, we believe that it is not tight.

We have also presented an improved scheme for the problem. It refines the approach taken by Baig *et al.* [7] and alleviates the need for blocks that overlap completely to save space. This approach helps us achieve an upper bound of $\mathcal{O}(m^{5/6})$ which is a marked improvement over existing schemes.

We note that there is still a gap between the upper and lower bounds for the problem, and we believe the following to be true.

1. There should exist a $\mathcal{O}(m^{4/5})$-scheme for subsets of size four ($n = 4$).
2. $s_A(5, m, 2) = \Theta(m^{4/5})$.

8

Appendix

8.1 Appendix A

In this section, we prove our expression for the space lower bound. We start by proving a simple fact about sum of products.

Claim: Given that $\sum_{i=1}^{n} a_i \geq C_1$ and $\sum_{i=1}^{n} b_i \geq C_2$, then

$$\sum_{i=1}^{n} a_i b_i \geq \frac{C_1 C_2}{n}.$$

Proof. Consider the following sum –

$$\sum_{i=1}^{n} (a_i + b_i)^2.$$

This is minimised when the all of the summands are equal. Thus,

$$\sum_{i=1}^{n} (a_i + b_i)^2 \geq \sum_{i=1}^{n} \left(\frac{C_1 + C_2}{n} \right)^2 = \frac{(C_1 + C_2)^2}{n}.$$

We can now expand the sum to prove the desired inequality.

$$\begin{aligned}\sum_{i=1}^{n}(a_i+b_i)^2 &= \sum_{i=1}^{n}a_i^2+\sum_{i=1}^{n}b_i^2+\sum_{i=1}^{n}2a_ib_i\\ &\geq n\left(\frac{C_1}{n}\right)^2+n\left(\frac{C_2}{n}\right)^2+\sum_{i=1}^{n}2a_ib_i\\ &\geq \frac{(C_1+C_2)^2}{n}\\ \implies \sum_{i=1}^{n}2a_ib_i &\geq \frac{(C_1+C_2)^2}{n}-n\left(\frac{C_1}{n}\right)^2-n\left(\frac{C_2}{n}\right)^2\\ &= 2\frac{C_1C_2}{n}.\end{aligned}$$

□

We apply the claim above repeatedly to prove the following lemma. It is important to note that the sum is computed w.r.t. table $\mathcal{B}$, and in table $\mathcal{B}$ all the elements are good.

Lemma 8.1. $\sum_{e\in\mathcal{U}} \mid \mathcal{U}_{\mathcal{B}}^2(e) \mid \geq\ c\cdot\frac{m^4}{s^3}$.

Proof. We have the following expression for the sum of the sizes of all 2-universes of all elements.

$$\begin{aligned}\sum_{e\in\mathcal{U}} \mid \mathcal{U}_{\mathcal{B}}^2(e) \mid &= \sum_{e\in\mathcal{U}} \mid \bigcup_{f\in\mathcal{U}_{\mathcal{B}}(e)} \mathcal{C}(f)\setminus\{f\} \mid\\ &= \sum_{e\in\mathcal{U}}\left(\sum_{f\in\mathcal{U}_{\mathcal{B}}(e)} \mid \mathcal{C}(f)\setminus\{f\} \mid\right)\end{aligned}$$

We could convert the union in the expression above into the summation as no two elements of $\mathcal{U}_{\mathcal{B}}(e)$ share a set. We can similarly expand $\mathcal{U}_{\mathcal{B}}(e)$ from Definition 7.1.

$$\begin{aligned}\sum_{e\in\mathcal{U}} \mid \mathcal{U}_{\mathcal{B}}^2(e) \mid &= \sum_{e\in\mathcal{U}}\left(\sum_{f\in\mathcal{U}_{\mathcal{B}}(e)} \mid \mathcal{C}(f)\setminus\{f\} \mid\right)\\ &= \sum_{e\in\mathcal{U}}\left(\sum_{g\in\mathcal{B}(e)\setminus\{e\}}\left(\sum_{f\in\mathcal{A}(g)\setminus\{g\}}(\mid \mathcal{C}(f)\setminus\{f\} \mid)\right)\right)\end{aligned}$$

We will first compute the value of the following expression.

$$\sum_{e\in\mathcal{U}} |\mathcal{B}(e)\setminus\{e\}| = \sum_{e\in\mathcal{U}} (|\mathcal{B}(e)|-1) = \sum_{e\in\mathcal{U}} |\mathcal{B}(e)| - m$$
$$= \sum_{X\in\mathcal{B}} c_X|X| - m. \qquad \text{(collecting over the sets of } \mathcal{B})$$

Here, the sum of the coefficients c_X is m, and the number of terms, which is same as the number of sets of $\mathcal{B}$, is s. Further, $\sum_{X\in\mathcal{B}} |X| = m$. So, applying the above claim,

$$\sum_{e\in\mathcal{U}} |\mathcal{B}(e)\setminus\{e\}| = \sum_{X\in\mathcal{B}} c_X|X| - m$$
$$\geq \frac{m\cdot m}{s} - m \qquad \text{(sum of the sizes of the sets of } \mathcal{B} \text{ is } m)$$
$$\geq c\frac{m^2}{s} \qquad \text{(for some suitable coefficient c)}$$

Next, we compute an expression the sum of whose coefficients is the above sum.

$$\sum_{e\in\mathcal{U}} \left(\sum_{g\in\mathcal{B}(e)\setminus\{e\}} |\mathcal{A}(g)\setminus\{g\}| \right) \geq c\sum_{Z\in\mathcal{A}} c_Z|Z|, \qquad \text{(collecting over the blocks of } \mathcal{A})$$
$$\geq c'\cdot\frac{\frac{m^2}{s}m}{s} = c'\frac{m^3}{s^2}. \qquad \text{(for some suitable coefficient c')}$$

We finally compute the desired expression of which the sum of coefficients is the above expression.

$$\sum_{e\in\mathcal{U}} |\,\mathcal{U}^2_{\mathcal{B}}(e)\,| = \sum_{e\in\mathcal{U}} \left(\sum_{g\in\mathcal{B}(e)\setminus\{e\}} \left(\sum_{f\in\mathcal{A}(g)\setminus\{g\}} (|\,\mathcal{C}(f)\setminus\{f\}\,|) \right) \right)$$
$$\geq c\cdot\sum_{Y\in\mathcal{C}} c_Y|Y|, \qquad \text{(collecting over the sets of } \mathcal{C})$$
$$= c'\cdot\frac{\frac{m^3}{s^2}m}{s} = c'\frac{m^4}{s^3}. \qquad \text{(for some suitable coefficient c')}$$

□

8.2 Appendix B

Rest of the case of subsection 7.3.2 is discussed here.

Case 5 The elements in $S_1 = \{n_1, n_2, n_3\}$ lies in a superblock and the elements n_4 and n_5 in the different superblocks.

Case 5.1 Blocks having element n_1, n_2 and n_3 lies on a same dotted line.

Case 5.1.1 Blocks having element n_4 and n_5 have a conflicting bit common with the blocks having elements from S_1 in table T_1. In this case, we send the blocks having elements n_4 and n_5 to table T_0 and the rest of the empty blocks which lie on the dotted lines containing these blocks to table T_1. We send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 5.1.2 Only one of the block having element say n_4 have a conflicting bit common with the block having an element from S_1 in table T_1. Without loss of generality let us say block having the element n_1 have a conflicting bit common with the block having the element n_4. In this case, we send the blocks having elements n_4 and n_5 to table T_0 and all the empty blocks lying on the dotted line containing these blocks to table T_1. Now we see whether the dotted line which contains block having the element n_5 passes through the block having element elements from S_1 or not. Let us first consider a case where the dotted line which contains block having the element n_5 passes through one of the blocks having elements from S_1, without loss of generality let us say it passes through block having the element n_2. In this case, we send the block having the element n_2 to table T_0 and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On another hand, if the dotted line which contains block having the element n_5 does not pass through any of the block having element from S_1 then we send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 5.1.3 None of the blocks which contains element n_4 or n_5 have a conflicting bit

common with the blocks which include an element from S_1 in table T_1. Now, the blocks having elements n_4 and n_5 can conflict among themselves or it does not. If the block having elements n_4 and n_5 do not conflict among themselves, then we send all the blocks having elements to table T_1, and rest all the empty blocks to table T_0. On the other hand, if the blocks having elements n_4 and n_5 conflict among themselves, then we see whether they conflict on the dotted line which contains block having elements from S_1. If they conflict on the dotted line which contains block having elements from S_1, then we send the blocks having elements n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Further, we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0. If the blocks having elements n_4 and n_5 do not conflict on the dotted line which contains blocks having elements from S_1, then we send the block having element n_4 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Further, we send the block having element n_5 to table T_1, and all the empty blocks lying on the dotted line containing this block to table T_0. Now, we see whether the dotted line which contains block having element n_4 passes through block having element n_1 or n_2. Without loss of generality, let us say that the dotted line which contains block having element n_4 passes through block having element n_1. In this case, we send the block having element n_1 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If the dotted lines which contains block having elements n_4 or n_5 do not pass through block having elements from S_1, then we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

Case 5.2 Two elements say n_1 and n_2 lies on the same dotted line and the element n_3 lies on a different dotted line.

Case 5.2.1 Blocks having the elements n_4 and n_5 have a conflicting bit common with the blocks having elements from S_1 in table T_1.

Case 5.2.1.1 Blocks having the elements n_4 and n_5 have a conflicting bit common with the block having elements n_1 and n_2 in table T_1. In this case, we send the blocks

having elements n_4 and n_5 to table T_0 and all the empty blocks lying on the dotted lines containing these blocks to table T_1. Also, we send the blocks having elements n_1 and n_2 to table T_1. We send the block which contains the element n_3 to table T_0 and the rest of the empty block lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

Case 5.2.1.2 Blocks having elements n_4 and n_5 have a conflicting bit common with the blocks lying on the different dotted lines, say block having elements n_1 and n_3 in table T_1. In this case, we send the blocks having elements n_4 and n_5 to table T_1 and the rest of the empty blocks lying on the dotted line containing these blocks to table T_0 . We send the blocks having elements n_1 and n_3 to table T_0 and the rest of the blocks lying on these dotted lines to table T_1. Rest all the empty blocks are sent to table T_0.

Case 5.2.1.3 Blocks having element n_4 and n_5 have a conflicting bit common in table T_1. If these blocks have a conflicting bit common with the block having the element n_1 or n_2 then we send both the blocks having the element n_4 and n_5 to table T_0 and the rest of the empty block lying on the dotted line containing these blocks to table T_1. Also, we send the blocks having the element n_1, n_2 and the blocks lying on the dotted line containing these blocks to table T_1. We send the block which contains element n_3 to table T_0 and the rest of the empty block which lies on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

If the blocks containing elements n_4 and n_5 have a conflicting bit common with the block which contains element n_3 in table T_1 then we send the block containing element n_3 to table T_0 and the rest of the empty block which lie on the dotted line containing this block to table T_1. Also, we send the block containing n_4 to table T_1 and the rest of the empty block lying on the dotted line containing this block to table T_0. We send the block having the element n_5 to table T_0 and the rest of the empty block lying on the dotted line containing this block to table T_1. Now we see whether the dotted line which contains block having the element n_5 passes through the block having the element n_1 or n_2. Without loss of generality let us first consider the case where the dotted line which contains block having the element n_5 passes through the block having the element n_1. In this case, we send the block having the element n_1 to table T_0 and all the blocks lying

on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If the dotted line which contains block having element n_5 does not pass through the blocks having elements n_1 or n_2 then we send the blocks having elements n_1 and n_2 to table T_1 and rest all the empty blocks to table T_0.

Case 5.2.2 Only one block having an element n_4 or n_5, have a conflicting bit common with a block having the element from S_1 in table T_1. Let us first consider the case where block having element n_4 have a conflicting bit common with the block having element n_1 or n_2. Without loss of generality let us say block having the element n_4 have a conflicting bit common with the block having the element n_1. Now we can have two cases, either the block having element n_4 conflicts with block having the element n_3 or it does not. Let us first consider the case where block having the element n_4 do not conflict with block having the element n_3. In this case, we send the blocks having the elements n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted line containing these blocks to table T_1. Now we see whether the empty blocks lying on the dotted line which contains block having element n_5 passes through blocks having elements n_1 or n_2. Without loss of generality let us say that empty block lying on the dotted line which contains block having element n_5 passes through block having element n_1. In this case, we send the block having the element n_1 to table T_0 and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if the dotted line which contains block having element n_5 do not pass through blocks having element n_1 or n_2, then we send the block having element n_1 and n_2 to T_1, and rest all the empty blocks to table T_0. Now we consider the case where block having element n_4 have conflicting bit common with block having element n_3. In this case, we send the block having element n_4 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Now we see the position of the block having element n_5. If the block having element n_5 have conflicting bit common with empty block lying on the dotted line containing blocks having elements n_1 and n_2, then we send the block having element n_1, n_2, n_3 and n_5 to table T_1, and rest all the empty blocks to table T_0. If the block having element n_5 do not have conflicting bit common with empty block lying on the dotted line containing blocks n_1 and n_2, then we send the block having element n_5 to table T_0, and all the empty blocks lying on the dotted line

containing this block to table T_1. Now we see whether the blocks having element n_1 or n_2 have conflicting bit common with empty block lying on the dotted line containing block having element n_5. Without loss of generality let us say that block having element n_1 have conflicting bit common with empty block lying on the dotted line containing block having element n_5. In this case, we send the block having element n_1 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Further, we send the block having n_3 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If none of the blocks having element n_1 or n_2 have conflicting bit common with empty block lying on the dotted line containing element n_5, then we send the blocks having elements n_3 and n_5 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Furthermore, we send the blocks having element n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0.

Now let us consider the case where block having the element n_4 have a conflicting bit common only with the block having the element n_3. Now we see whether the blocks having elements n_4 and n_5 conflicts or not. Let us first consider the case where blocks having elements n_4 and n_5 do not conflicts. Now we see the position of the block having element n_5. Let us first consider the case where block having element n_5 conflicts with empty block lying on the dotted line which contains block having element n_1. In this case, we send the block having the element n_1, n_2, n_4 and n_5 to table T_1, and rest all the empty blocks lying on the dotted lines containing these blocks to table T_0. We send the block having the element n_3 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If the block having the element n_5 have a conflicting bit common with a empty block lying on the dotted line which contains block having the element n_3, then we send the blocks having the elements n_3 and n_5 to table T_1, and all the empty blocks lying on the dotted lines containing these blocks to table T_0. We send the block having element n_4 to table T_0, and rest all the empty blocks lying on the dotted line containing this block to table T_1. If the block having the element n_1 have a conflicting bit common with block lying on the dotted line which contains block having the element n_4, then we send the block having the element n_1 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. Similar

is the case if the block having the element n_2 have a conflicting bit common with block lying on the dotted line which contains block having the element n_4. On the other hand, if the dotted line which contains block having element n_4 do not pass through blocks having elements n_1 or n_2, then we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0. If none of the above case occurs, and the block having the element n_5 does not have a conflicting bit common with a block lying on the dotted line which contains block having the element n_3, then we send the block having the element n_1, n_2, n_4 and n_5 to table T_1, and rest all the empty blocks lying on the dotted lines containing these blocks to table T_0. Further, we send the block having element n_3 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. Now, we see the case where block having element n_4 and n_5 conflicts. Now we can have several cases depending upon whether blocks having element n_4 and n_5 conflicts with empty block lying on the dotted line which contains block having element n_1. Let us first consider the case where block having element n_5 conflicts with empty block lying on the dotted line which contains block having element n_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and rest all the empty blocks lying on the dotted lines containing these blocks to table T_1. Now we see whether the dotted line which contains block having element n_4 passes through block having element n_1 or n_2. Without loss generality let us say that dotted line which contains block having element n_4 passes through block having element n_1. In this case, we send the block having the element n_1 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if the dotted line which contains block having element n_4 do not pass through blocks having elements n_1 or n_2, then we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0. Similar is the case when block having element n_4 conflicts with empty block lying on the dotted line which contains block having element n_1. If none of the above occurs, then we send the blocks having elements n_3 and n_5 to table T_0, and rest all the empty blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the block having element n_4 to table T_1, rest all the empty blocks lying on the dotted line containing this block to table T_0. Now we see whether the dotted line which contains block having element n_5 passes through block having element n_1 or n_2.

Without loss generality let us say that dotted line which contains block having element n_5 passes through block having element n_1. In this case, we send the block having the element n_1 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if the dotted line which contains block having element n_4 do not pass through blocks having elements n_1 or n_2, then we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0.

Case 5.2.3 None of the blocks having the elements n_4 or n_5 have a conflicting bit common with blocks having elements from S_1 in table T_1. We can have several cases depending upon whether the blocks having element n_4 and n_5 conflicts or not. Let us first consider the case where blocks having elements n_4 and n_5 do not conflict. In this case we send all the blocks having elements to table T_1, and all the empty blocks to table T_0. Now let us consider the case where blocks having elements n_4 and n_5 conflicts. Now, we see the position of the block having elements n_4 and n_5. Let us first consider the case where blocks having elements n_4 and n_5 conflicts on the dotted line which contains block having element n_1. In this case, we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks lying on the dotted line containing these blocks to table T_0. Further, we send the blocks having elements n_3, n_4 and n_5 to table T_0, rest all the empty blocks lying on the dotted lines containing these blocks to the table T_1. Rest all the empty blocks are sent to table T_0. Now let us consider the case where block having elements n_4 and n_5 conflicts outside the dotted line which contains block having element n_1. Now we can have a case where either block having elements n_4 or n_5 conflicts with empty block lying on the dotted line which contains block having element n_1. Without loss of generality let us say that block having element n_4 conflicts with empty block lying on the dotted line which contains block having element n_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and rest all the empty blocks lying on the dotted line containing these blocks to table T_1. Now we see whether the dotted line which contains block having element n_5 passes through block having element n_1 or n_2. Without loss of generality, let us say that the dotted line which contains block having element n_5 passes through block having element n_1. In this case, we send the block having element n_1 to table T_0, and rest all the blocks lying on the

dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if the dotted line which contains block having element n_5 do not pass through block having element n_1 or n_2, then we send the block having element n_1 and $n2$ to table T_1, and rest all the empty blocks to table T_0. Now we consider the case where block having element n_4 or n_5 do not conflict with empty block lying on the dotted line which contains block having element n_1. In this case, we send the blocks having elements n_3 and n_5 to table T_1, and rest all the empty blocks lying on the dotted lines containing these blocks to table T_0. Further, we send the block having element n_4 to table T_0, and rest all the empty blocks lying on the dotted line containing this block to table T_1. Now we see if the dotted line which contains block having element n_4 passes through block having element n_1 or n_2. Without loss of generality let us say that dotted line which contains block having element n_4 passes through block having element n_1. In this case, we send the block having element n_1 to T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If the dotted line which contains block having element n_4 do not pass through block having element n_1 or n_2, then we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0.

Case 5.3 All the elements belonging to S_1 lies on the different dotted lines. In this case, we send all the blocks having elements to table T_0 and all the empty blocks to table T_1.

Case 6 Two elements $S_1 = \{n_1, n_2\}$ lies in a superblock other two elements $S_2 = \{n_3, n_4\}$ lies in other superblock and an element $S_3 = \{n_5\}$ in a different superblock.

Case 6.1 Blocks having elements belonging to S_1 lies on the same dotted line, blocks having elements belonging to S_2 lies on the same dotted line.

Case 6.1.1 Two blocks having elements from S_2 and S_3 have a conflicting bit common with the blocks having elements from S_1 in table T_1. Without loss of generality, let us say that block having element n_3 have a conflicting bit common with the block having the element n_1 and the block having the element n_5 have a conflicting bit common with

the block having the element n_2. In this case, we send the block having the element n_4 to table T_0 and the rest of the block lying on the dotted line containing this block to table T_1. We send the block having the element n_1 to table T_0, and the rest of the block lying on this dotted line to table T_1. Also, we send the block having the element n_5 to table T_0, and the rest of the empty block lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

If the blocks having element from S_2 and S_3 have a conflicting bit common with the same block having the element from S_1, then we send the blocks having a conflicting bit common from S_2 and S_3 to table T_0, and the rest of the blocks lying on these dotted lines to table T_1. Also, we send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 6.1.2 Only one block having the element from S_2 or S_3 have a conflicting bit common with the block having the element from S_1.

Case 6.1.2.1 Block having an element from S_2 have a conflicting bit common with the block having an element from S_1. Without loss of generality let us say that the block having the element n_3 have a conflicting bit common with the block having the element n_1. Now, we can have two cases, either the block containing element n_5 have a conflicting bit common with the block containing element n_4 or it does not.

If the block containing n_5 have a conflicting bit common with the block containing n_4, then we send the block containing n_3 to table T_0, and the rest of the block lying on the dotted line containing this block to table T_1. We send the block containing n_5 to table T_0, and the rest of the block lying on the dotted line containing this block to table T_1. Now we see whether the dotted line which contains block having element n_5 passes through block having element n_2 or not. Let us first consider the case where dotted line which contains block having element n_5 passes through block having element n_2. In this case, we send the block having the element n_2 to table T_0, and the rest of the block lying on this dotted line to table T_1. Rest all the empty block are sent to table T_0. On the other hand, if the dotted line which contains block having element n_5 do not pass through block having element n_2, then we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0.

If the block containing n_5 do not have a conflicting bit common with the block having the element n_4, then we send the block having the element n_3 to table T_0 and the rest of the block lying on this dotted line to table T_1. Now we see the position of the block which contains the element n_5 to make the assignment. If the block which contains element n_5 have conflicting bit common with empty block lying on the dotted line which contains block having elements from S_1, then we send the block having the element n_5 to table T_1, and rest all the empty blocks lying on the dotted line which contains this block to table T_0. Also, we send blocks having elements from S_1 to table T_1, and the rest of the empty blocks lying on the dotted line which contains this block to table T_0. Rest of the empty blocks we send to table T_0.

Rest for all other positions of n_5, we send blocks having elements n_3 to table T_0, and all the blocks lying on the dotted line containing this block to table T_1. Further, we send the block having the element n_5 to table T_0, and all other blocks lying on the dotted line containing this block to table T_1. Now we see whether the dotted line which contains block having element n_5 passes through block having element n_1 or n_2. Without loss of generality, let us say that dotted line which contains block having element n_5 passes through block having element n_2, in this case we send the block having element n_2 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand if the dotted line which contains block having element n_5 do not pass through block having element n_1 or n_2, then we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0.

Case 6.1.2.2 Block having an element from S_3 have a conflicting bit common with the block having an element from S_1. Without loss of generality let us say that block having element n_5 have a conflicting bit common with the block having the element n_1. In this case, we send the block having the element n_5 to table T_0 and the rest of the block lying on the dotted line which contains this block to table T_1. Now we see the position of the block having the element from S_2.

One of the block having an element from S_2 have a conflicting bit common with the empty block on the dotted line which contains block having the element n_5. Without loss of generality let us say that block having element n_3 have a conflicting bit common with

an empty block on the dotted line which contains block having the element n_5. In this case, we send the block having an element n_3 to table T_0, and the rest of the block lying on the dotted line containing this block to table T_1. Now we see the position of the block having element n_4, if it has a conflicting bit common with a empty block lying on the dotted line which contains block having element from S_1, then we send the block having element n_1 and n_2 to table T_1, and the rest of the empty block lying on the dotted line containing these blocks to table T_0. Rest all the empty blocks are sent to table T_0. On another hand, if the block having the element n_4 does not have a conflicting bit common with a block lying on the dotted line which contains block having the element from S_1 then we send the block having the element n_2 to table T_0, and the rest of the block lying on the dotted line containing this block to table T_1. We send the rest of the empty block to table T_0.

If the block having the element from S_2 do not have a conflicting bit common with a block lying on the dotted line which contains block having the element n_5 then we send the blocks having an element from S_2 to table T_1, and the rest of the empty block lying on the dotted line which contains these blocks to table T_0. Also, we send the blocks having the element n_1 and n_2 to table T_1. We send rest of the empty block to table T_0.

Case 6.1.3 None of the block having elements from S_2 or S_3 has a conflicting bit common with the blocks having an element from S_1 in table T_1. Now, here, we can have two cases. Either the block having the element from S_3 have a conflicting bit common with the block having the element from S_2 or it does not.

Let us first consider the case where one of the blocks having the element from S_3 have a conflicting bit common with one of the blocks having an element from S_2. Without loss of generality, we can say that block having the element n_5 have a conflicting bit common with the block having the element n_4. Now, we see whether the dotted lines containing blocks having elements from S_2 or S_3 passes through block having elements from S_1 or not. Without loss of generality, let us say that dotted line which contains block having element n_5 passes through block having element n_1. In this case, we send the block having the element n_5 to table T_1, and the rest of the empty blocks lying on the dotted line which contains this block to table T_0. We send the block having the element n_4 to table T_0, and rest of the block lying on the dotted line which contains this block to

table T_1. Now we see the positions of the blocks having an element from S_1. Let us first consider the case where a block having an element from S_1 have a conflicting bit common with a block lying on the dotted line which contains blocks having elements from S_2. Without loss of generality let us say block having the element n_2 have a conflicting bit common with block lying on the dotted line which contains blocks having elements from S_2. In this case, we send the block having the element n_2 to table T_0, and all other block lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if the blocks having elements from S_1 do not have a conflicting bit common with block lying on the dotted line which contains blocks having elements from S_2, then we send the block having elements from S_1 to table T_1, and rest all the empty blocks to table T_0. If none of the dotted lines which contains blocks having elements from S_2 or S_3 passes through block having elements from S_1, then we send the blocks having elements n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

If the block having an element from S_3 do not have a conflicting bit common with block having an element from S_2, then we send all the block having elements to table T_1, and all the empty block to table T_0.

Case 6.2 Now we consider the case where one of the sets having elements lie on a dotted line and other set having element lie on the different dotted lines. Without loss of generality consider the case where blocks having elements belonging to S_1 lies on a dotted line and blocks having elements belonging to S_2 lies on the different dotted lines.

Case 6.2.1 All the blocks having elements from S_2 and S_3 have a conflicting bit common with the blocks having elements from S_1. In this case, we send the block having an element from S_2 and S_3 to table T_0, and rest of the empty blocks lying on the dotted lines containing these blocks to table T_1. Also, we send the blocks having elements from S_1 to table T_1 and the rest of the empty blocks lying on the dotted line which contains these blocks to table T_0. We send the rest of the empty blocks to table T_0.

Case 6.2.2 Two blocks having elements from S_2 and S_3 have a conflicting bit com-

mon with the block having an element from S_1. Now, here we can have two cases, either those two blocks have elements belonging to S_2, or we can have one element belonging to S_2, and other to S_3.

Let us first consider the case where two blocks having elements from S_2 have a conflicting bit common with blocks having an element from S_1. Without loss of generality, let us say that block having element n_3 have a conflicting bit common with the block having the element n_1 and the block having the element n_4 have a conflicting bit common with the block having n_2. In this case, we send the block having elements n_3 and n_4 to table T_0 and rest of the empty block lying on the dotted lines containing these blocks to table T_1. Now we see the position of the block having the element n_5. Here we can have two cases either the dotted line which contains the block having the element n_5 passes through one of the block having an element from S_1 or it does not. Let us first consider the case where it passes through one of the blocks having elements from S_1. Without loss of generality let us say that the dotted line which contains block having the element n_5 passes through the block having the element n_1. In this case, we send the block having the element n_5 to table T_0, and rest of the block lying on the dotted line which contains this block to table T_1. Also, we send the block having the element n_1 to table T_0, and rest of the block lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On another hand, if the dotted line which contains the block having the element n_5 do not pass through blocks having elements from S_1, then we send the block having the element n_5 to table T_0, and rest of the empty blocks on the dotted line containing this block to table T_1. Also, we send the blocks having elements from S_1 to table T_1, and rest of the empty blocks to table T_0. If the block having elements n_3 and n_4 conflicts with the same block having elements from S_1, say n_1, then assignment made above will work in this case.

Now we consider the case where one of the blocks having an element from S_2 and the block having an element from S_3 have a conflicting bit common with the block(blocks) having an element from S_1. Without loss of generality say blocks having the element n_3 and n_5 have a conflicting bit common with the block having elements from S_1. Now here we can have two cases either blocks having elements n_3 and n_5 have a conflicting bit common with the same block having an element from S_1 or it have a conflicting bit common with different blocks having elements from S_1. Let us first consider the case

where blocks having element n_3 and n_5 have a conflicting bit common with the same block having an element from S_1 say n_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and rest of the blocks lying on the dotted lines containing these blocks to table T_1. Now we see whether the dotted line which contains block having element n_4 passes through block having element from S_1 or not. Without loss of generality let us say that dotted line which contains block having element n_4 passes through block having element n_2. In this case, we send the block having the element n_2 to table T_0, and rest of the block which lies on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If the dotted line which contain block having element n_4 do not pass through block having element from S_1, then we send the blocks having elements from S_1 to table T_1, rest all the empty blocks to table T_0. Now we consider the case where blocks having elements n_3 and n_5 have a conflicting bit common with different blocks having elements from S_1. Without loss of generality let us say that block having element n_3 have a conflicting bit common with the block having the element n_1 and the block having the element n_5 have a conflicting bit common with the block having the element n_2. In this case also assignment made above will work.

Case 6.2.3 Only one block having an element from S_2 or S_3 have a conflicting bit common with the block having an element from S_1 in table T_1. Now here we can have two cases either the block having an element from S_2 have a conflicting bit common or the block having an element from S_3 have a conflicting bit common with the block having an element from S_1.

Case 6.2.3.1 The block having an element from S_2 have a conflicting bit common with a block having an element from S_1 in table T_1. Without loss of generality let us say that the block having the element n_3 conflicts with the block having the element n_1. Now we see the position of the blocks having the element n_4 and n_5. Let us first consider the case where blocks having elements n_4 and n_5 have a conflicting bit common.

If the blocks having elements n_4 and n_5 have a conflicting bit common on the dotted line which contains blocks having elements from S_1, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. We send the block having element n_1 and n_2 to

table T_1. Rest all the empty blocks are sent to table T_0.

If blocks having elements n_4 and n_5 have a conflicting bit common outside the dotted line which contains blocks having elements from S_1, then we see the positions of the dotted lines which contains block having elements n_4 and n_5. Without loss of generality let us say that the dotted line which contains block having element n_5 passes through block having element n_2, and the dotted line which contains block having element n_4 passes through block having element n_1. In this case, we send the blocks having elements n_2, n_3 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the block having element n_4 to table T_1. Rest all the empty blocks are sent to table T_0. Now we consider the case where only one dotted line which contains block having element n_4 or n_5 passes through block having element from S_1. Without loss of generality let us say that block having element n_5 passes through block having element n_2. In this case, we send the blocks having elements n_2, n_3, n_4 and n_5 to table T_0, and rest all the blocks lying on the dotted lines containing these block to table T_1. Further, we send rest all the empty blocks to table T_0. If none of the dotted lines, which contains blocks having element n_4 or n_5 passes through block having elements from S_1, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks are sent to table T_0.

Now we are left with the case where blocks having elements n_4 and n_5 do not have a conflicting bit common. This can have several sub-cases. Let us first consider the case where both the blocks having elements n_4 and n_5 passes through the dotted line which contains blocks having elements from S_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. We send the blocks having elements n_1 and n_2 to table T_1. Rest all the empty blocks are sent to table T_1.

Now let us consider the case where n_4 and n_5 do not have a conflicting bit common with block lying on the dotted line which contains block having elements from S_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_1, and all the blocks lying on the dotted lines containing these blocks to table T_0. Block having the element n_1 is sent to table T_0, and rest all the blocks lying on the dotted line containing these

blocks to table T_1. Rest all the empty blocks are sent to table T_0.

Now, we can also have a case where only one block having the element n_4 or n_5 have a conflicting bit common with block lying on the dotted line which contains blocks having elements from S_1. Let us first consider the case where block having the element n_4 have a conflicting bit common with the block lying on the dotted line which contains block having elements from S_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all other blocks lying on the dotted line containing these blocks to table T_1. Now, we see whether the block having elements from S_1 have a conflicting bit common with block lying on the dotted line which contains block having the element n_5. Without loss of generality let us say block having the element n_1 have a conflicting bit common with block lying on the dotted line which contains block having the element n_5. In this case, we send the block having the element n_1 to table T_0, and all other blocks lying on the dotted line which contains this block to table T_1. Rest all the empty blocks are sent to table T_1. On another hand, if the dotted line which contains block having the element n_5 do not pass through block having element from S_1, then we send the block having element from S_1 to table T_1, and all the blocks lying on the dotted line containing this block to table T_0. Rest all the empty blocks are sent to table T_0. Now let us consider the case where block having the element n_5 have a conflicting bit common with block lying on the dotted line which contains block having elements from S_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0 and all other blocks lying on the dotted line containing these blocks to table T_1. Now we see whether the dotted line which contains block having element n_4 passes through block having element n_1 or n_2. Without loss of generality let us say that dotted line which contains block having element n_4 passes through block having element n_2. In this case, we send the block having element n_2 to table T_0, rest all the blocks lying on the dotted line which contains this block to table T_1. Rest all the empty blocks are sent to table T_0. If the dotted line which contains block having element n_4 do not pass through block having element from S_1, then we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

Now let us consider the case where none of the blocks having elements n_4 or n_5 conflict with the dotted line which contains block having elements from S_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying

on the dotted lines containing these blocks to table T_1. Further, blocks having elements from S_1 are sent to table T_1, and rest all the empty blocks are sent to table T_0.

Case 6.2.3.2 The block having an element from S_3 have a conflicting bit common with a block having an element from S_1 in table T_1. Without loss of generality let us say that block having element n_5 have a conflicting bit common with the block having the element n_1. Now we see the position of the blocks having elements from S_2.

Both the blocks having an element from S_2 have a conflicting bit common with block lying on the dotted line which contains block having an element from S_1 in table T_1. In this case, we send the blocks having the elements n_3, n_4 and n_5 to table T_0 and all the blocks lying on the dotted lines containing these blocks to table T_1. Blocks having elements from S_1 are sent to table T_1 and rest all the empty blocks are sent to table T_0.

Both the blocks having an element from S_2 do not have a conflicting bit common with block lying on the dotted line which contains blocks having an element from S_1. Now we see whether the blocks having elements from S_2 conflicts with block having element n_5. Let us first consider the case where none of the blocks having elements from S_2 conflicts with block having element n_5. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_1, and all the empty blocks lying on the dotted lines containing these blocks to table T_0. Further, we send the block having the element n_1 to table T_0, and all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. Now let us consider the case where only one block having element from S_2 conflicts with block having element n_5. Without loss of generality let us say that block having element n_4 conflicts with block having element n_5. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1.Now we see whether the dotted line which contains block having element n_3 passes through block having element n_1 or n_2. Without loss of generality let us say that the dotted line which contains block having element n_3 passes through block having element n_2. In this case, we send the block having element n_2 to table T_0, all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If the dotted line which contains block having element n_3 do not pass through block having element from S_1, then we send the block having element from S_1 to table T_1,

rest all the empty blocks to table T_0. Now we consider the case where both the blocks having elements from S_2 conflicts with block having element n_5. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, all the blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the block having element n_2 to table T_0, all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. Only one block having the element from S_2 have a conflicting bit common with block lying on the dotted line which contains blocks having elements from S_1 in table T_1. Without loss of generality let us say that block having element n_3 have a conflicting bit common with block lying on the dotted line which contains block having an element from S_1. Now we see whether the block having element n_4 conflicts with block having element n_5. Let us first consider the case where blocks having element n_4 and n_5 conflicts. In this case, we send the blocks having elements n_1, n_2, n_3 and n_4 to table T_1, and all the empty blocks lying on the dotted lines which contain these blocks to table T_0. We send the blocks having the element n_5 to table T_0, and rest all the blocks lying on the dotted lines which contain these blocks to table T_1. Rest all the empty blocks are sent to table T_0. If the blocks having elements n_4 and n_5 do not conflict, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Now we see whether a block having element from S_1 conflicts with empty block lying on the dotted line which contains block having element n_4. Without loss of generality, let us say that block having element n_2 conflicts with empty block lying on the dotted line which contains block having element n_4. In this case, we send the block having element n_2 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If none of the block having element from S_1 conflicts with empty block lying on the dotted line which contains block having element n_4, then we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

Case 6.2.4 None of the blocks having an element from S_2 or S_3 have a conflicting bit common with the block having elements from S_1 in table T_1. In this case, we can have either the block having the element n_5 have a conflicting bit common with the block having elements from S_2 or it does not.

Let us first consider the case where the block having the element n_5 have a conflicting bit common with one of the blocks having the element from S_2. Without loss of generality let us say that the block having the element n_5 have a conflicting bit common with the block having the element n_3. Now we see whether the block having the element n_4 have conflicting bit common with a block lying on the dotted line which contains block having elements from S_1 or not. Let us first consider the case where block having the element n_4 have conflicting bit common with block lying on the dotted line which contains block having an element from S_1. In this case, we send the block having the element n_3 to table T_1, and all the blocks lying on the dotted line containing this block to table T_0. We send the block having the element n_5 to table T_0, and all the blocks lying on the dotted line which contains this block to table T_1. Now we see whether any of the block having elements from S_1 have a conflicting bit common with block lying on the dotted line which contains element n_5 or not. Without loss of generality let us say that the block having the element n_1 have conflicting bit common with block lying on the dotted line which contains block having the element n_5. In this case, we send the blocks having the elements n_1 and n_4 to table T_0, and all other blocks lying on the dotted lines which contain these blocks to table T_1. Rest all the empty blocks are sent to table T_0. If none of the blocks having elements from S_1 have conflicting bit common with block lying on the dotted line which contains block having the element n_5, then we send the blocks having elements n_1, n_2 and n_4 to table T_1, and rest all the empty blocks to table T_0. Now let us consider the case where block having the element n_4 do not have conflicting bit common with block lying on the dotted line which contains block having elements from S_1. In this case, we see whether a block having element n_5 have conflicting bit common with a block having element n_4. Let us first consider the case where a block having element n_5 do not have conflicting bit common with a block having element n_4. In this case, we send the blocks having elements n_4 and n_5 to table T_1, and all the blocks lying on the dotted lines containing these blocks to table T_0. Further, we send the block having element n_3 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Now, we see if either of the blocks having element n_1 or n_2 have conflicting bit common with block lying on the dotted line which contains block having element n_3. Without loss of generality, let us say block having element n_1 have conflicting bit common with block lying on the dotted line which contains block

having element n_3. In this case, we send the block having element n_1 to table T_0, and all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the contrary, if neither of the block having element n_1 or n_2 has conflicting bit common with block lying on the dotted line which contains block having element n_3, then we send the block having an element from S_1 to table T_1, and rest all the empty to table T_0. We are now left with the case where block having element n_4 have conflicting bit common with a block having element n_5. In this case, we send the block having n_3 and n_4 to table T_1, and all the blocks lying on the dotted line containing this block to table T_0. Further, we send the block having element n_5 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Now we see whether any of the block having elements from S_1 have a conflicting bit common with block lying on the dotted line which contains element n_5 or not. Without loss of generality let us say that the block having the element n_1 have conflicting bit common with block lying on the dotted line which contains block having the element n_5. In this case, we send the block having element n_1 to table T_0, and all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If none of the blocks having elements from S_1 have conflicting bit common with block lying on the dotted line which contains block having the element n_5, then we send the blocks having elements from S_1 to table T_1 and rest all the empty blocks to table T_0.

Now we are left with the case where block having the element n_5 do not have a conflicting bit common with the block having an element from S_2. In this case, we send the blocks having elements to table T_1 and all the empty blocks to table T_0.

Case 6.3 All the elements belonging S_1, S_2 and S_3 lies on the different dotted line. In this case, we send the blocks having elements to table T_0, and all the empty blocks to table T_1.

Case 7 Two blocks having elements belonging to $S_1 = \{n_1, n_2\}$ lies in a same superblock and the element n_3, n_4 and n_5 to the different superblocks. If the blocks having elements from S_1 lies on the different dotted line, then assignment made in Case 5.3 can be used. So let us consider the case where blocks having the element from S_1 lies on the

same dotted line.

Case 7.1 Only one block having element from n_3, n_4 and n_5 have conflicting bit common with the block having element from S_1. Without loss of generality let us say block having the element n_3 have conflicting bit common with the block having the element n_1.

Case 7.1.1 Blocks having elements n_4 and n_5 have conflicting bit common.

Case 7.1.1.1 Blocks having elements n_4 and n_5 have conflicting bit common with block lying on the dotted line which contains block having an element from S_1. In this case, we send the block having elements n_3, n_4 and n_5 to table T_0, and rest of the empty blocks lying on the dotted lines containing these blocks to table T_1. Blocks having the elements n_1 and n_2 are sent to table T_1 and rest all the empty blocks are sent to table T_0.

Case 7.1.1.2 Blocks having elements n_4 and n_5 does not have conflicting bit common with block lying on the dotted line, which contains block having an element from S_1. Now here we can have several cases depending upon whether the dotted lines which contain blocks having elements n_4 and n_5 passes through blocks having elements from S_1 or not.

Without loss of generality let us say that dotted line which contains block having the element n_4 passes through the block having the element n_1, and the dotted line which contain block having the element n_5 passes through the block having the element n_2. Now in this case we send the blocks having elements n_4 and n_1 to table T_1. Also, we send the blocks having elements n_3, n_5 and n_2 to table T_0, and rest of the blocks lying on the dotted lines containing these blocks to table T_1. We send rest of the empty blocks to table T_0.

Now consider a case where only one dotted line which contains a block having element n_4 or n_5 passes through the block having an element from S_1. Without loss of generality let us say that block having the element n_4 passes through the block having an element from S_1. Let us first consider the case where the dotted line which contains block having the element n_4 passes through the block having the element n_2. In this case, we send the blocks having elements n_2, n_3, n_4 and n_5 to table T_0, and rest of the

blocks lying on the dotted lines containing these blocks to table T_1. Rest all the empty blocks are sent to table T_0. Now we can also have a case where block having the element n_4 passes through the block having the element n_1. In this case, we send the block having the elements n_1, n_3, n_4 and n_5 to table T_0, and rest of the blocks lying on the dotted lines containing these blocks to table T_1. Rest all the empty blocks are sent to table T_0. If none of the dotted lines which contains blocks having elements n_4 and n_5 passes through blocks having elements from S_1 then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. Also, we send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 7.1.2 Blocks having the elements n_4 and n_5 do not have conflicting bit common in table T_1 .

Case 7.1.2.1 Both the blocks having element n_4 and n_5 have conflicting bit common with block lying on the dotted line which contains block having an element from S_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and rest of the empty blocks lying on the dotted line containing these blocks to table T_1. Also, we send the blocks having elements n_1 and n_2 to table T_1, and rest of the empty blocks to table T_0.

Case 7.1.2.2 One of the blocks having an element n_4 or n_5 have conflicting bit common with block lying on the dotted line which contains blocks having elements from S_1 and other do not have conflicting bit common. Without loss of generality let us say block having the element n_4 have conflicting bit common with block lying on the dotted line which contains blocks having elements from S_1, and block having the element n_5 lies outside it. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. If any block having an element from S_1 have conflicting bit common with block lying on the dotted line which contains block having elements n_5, then we send that block to table T_0 all other blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On another hand, if none of the blocks having elements from S_1 have conflicting bit common with block lying on the dotted line which

contains block having elements n_5 then we send the block having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

Case 7.1.2.3 None of the blocks having elements n_4 or n_5 have conflicting bit common with block lying on the dotted line which contains blocks having elements from S_1. Now we see position of the block having element n_4 and n_5. Let us first consider the case where blocks having elements n_4 and n_5 do not conflicts with block having element n_3. In this case, we send the blocks having elements n_2, n_3, n_4 and n_5 to table T_1. Further, we send the block having the element n_1 to table T_0 and the rest of the empty blocks lying on the dotted line which contains this block to table T_1. Rest all the empty blocks are sent to table T_0. If both the blocks having elements n_4 and n_5 conflicts with block having element n_3. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines which contain these blocks to table T_1. Further, we send the block having element n_2 to table T_0, and all the blocks lying on the dotted line which contain this block to table T_1. Rest all the empty blocks are sent to table T_0. Now we are left with the case, where only one block having element n_4 or n_5 conflicts with block having element n_3. Without loss of generality let us say that block having element n_4 conflicts with block having element n_3. In this case, we send the block having element n_3 and n_5 to table T_0, and rest all the empty blocks lying on the dotted lines containing theses blocks to table T_1. We send the block having element n_4 to table T_1, and rest all the empty blocks lying on the dotted line containing this block to table T_0. Now we see whether the dotted line which contains block having element n_5 passes through block having element n_1 or n_2. Without loss of generality let us say that dotted line which contains block having element n_5 passes through block having element n_1. In this case, we send the block having element n_1 to table T_0, and rest all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if the dotted line which contains block having element n_5 do not pass through block having element n_1 or n_2, then we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

Case 7.2 Two blocks having elements from n_3, n_4 or n_5 have conflicting bit common with the block having elements from S_1. Without loss of generality let us say that blocks

having elements n_3 and n_4 have conflicting bit common with the block having elements from S_1. Now, here we can have two cases, either the blocks having elements n_3 and n_4 have conflicting bit common with the same block having an element from S_1 or it has conflicting bit common with the different blocks having elements from S_1.

Let us first consider the case where blocks having the element n_3 and n_4 have conflicting bit common with the different blocks having elements from S_1. Without loss of generality, let us say that block having element n_3 have conflicting bit common with the block having the element n_1 and the block having the element n_4 have conflicting bit common with the block having the element n_2. Now let us consider the intersection of the dotted line which contains block having the element n_5 from the blocks having elements from S_1 . Without loss of generality, let us say that the dotted line which contains block having element n_5 passes through the block having element n_1, in this case, we send the blocks having elements n_1, n_3, n_4 and n_5 to table T_0 and all other blocks lying on the dotted lines which contains these blocks to table T_1. Rest all the empty blocks are sent to table T_1. On the another hand, if the block having element n_5 do not pass through the blocks having elements from S_1, then we send the blocks having n_3, n_4 and n_5 to table T_0, and all other blocks lying on the dotted lines which contains these blocks to table T_1. Further, we send the blocks having elements from S_1 to table T_1 and rest all the empty blocks to table T_0.

Now we are left with the case where blocks having the element n_3 and n_4 have conflicting bit common with only one block having an element from S_1. Without loss of generality let us say that blocks having elements n_3 and n_4 have conflicting bit common with the block having the element n_1. In this case, we see the position of the block having the element n_5. If the block having the element n_5 have conflicting bit common with empty block lying on the dotted line having elements from S_1, then we send the block having the element n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted lines containing these blocks to table T_1. We send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0. Further, if the block having the element n_5 do not have conflicting bit common with empty block lying on the dotted line which contains blocks having element from S_1 then we send the block having element n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted lines containing these blocks to table T_1. Now we see whether the dotted line which

contains block having the element n_5 passes through a block having an element from S_1 or not. Let us first consider the case where the dotted line which contains block having the element n_5 passes through a block having an element from S_1. Without loss of generality, let us say that the dotted line which contains block having the element n_5 passes through the block having the element n_1. In this case, we send the block having the element n_1 to table T_0, and all other blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On another hand, if the dotted line which contains block having the element n_5 do not pass through the block having an element from S_1, then we send the blocks having the elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 7.3 All the three blocks having elements n_3, n_4 and n_5 have conflicting bit common with the blocks having elements from S_1. In this case, we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. Blocks having elements from S_1 are sent to table T_1. Rest all the empty blocks are sent to table T_0.

Case 7.4 None of the blocks having elements from n_3, n_4 and n_5 have conflicting bit common with the blocks having elements from S_1.

Let us first consider the case where all the blocks having elements n_3, n_4 and n_5 have conflicting bit common. If all of them have conflicting bit common on the dotted line which contains blocks having elements from S_1 then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and rest of the empty blocks lying on the dotted lines containing these blocks to table T_1. Also we send the blocks having elements n_1 and n_2 to table T_1. Rest all the empty blocks are sent to table T_0.

If the blocks having elements n_3, n_4 and n_5 have conflicting bit common outside the dotted line which contains blocks having elements from S_1, then we see the intersection of the dotted lines containing these blocks with the blocks having elements from S_1. Now since all the blocks having elements from n_3, n_4 and n_5 have conflicting bit common at most two dotted lines having blocks containing these elements can have conflicting bit common with the blocks having elements from S_1. Without loss of generality let us say that the dotted line which contains block having the element n_3 passes through the block

having the element n_1 and the dotted line which contains block having the element n_4 passes through the block having the element n_2. In this case, we send the blocks having element n_1 and n_3 to table T_1. Also, we send the blocks having elements n_2, n_4 and n_5 to table T_0, and rest of the empty blocks lying on the dotted line containing these blocks to table T_1. Rest all the empty blocks are sent to table T_0.

Without loss of generality let us now consider a case where only one dotted line having element say n_3 passes through the block having element say n_1, then we send the blocks having elements n_1, n_2 and n_3 to table T_1. Also, we send the blocks having elements n_4 and n_5 to table T_0, and rest of the empty blocks lying on the dotted lines containing these blocks to table T_1.Rest all the empty blocks are sent to table T_0.

If none of the dotted lines containing blocks having elements n_3, n_4 and n_5 passes through the blocks having element from S_1 then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and rest of the empty blocks lying on the dotted line containing these blocks to table T_1. Further, we send the blocks having element n_1 and n_2 to table T_1, and rest of the empty blocks to table T_0.

Now let us consider the case where only two blocks having elements from n_3, n_4 and n_5 have conflicting bit common. Without loss of generality let us say the blocks having elements n_3 and n_4 have conflicting bit common. Now we see the intersection of dotted lines containing the blocks n_3 and n_4. Let us first consider the case where both the dotted lines containing blocks n_3 and n_4 passes through the blocks having elements from S_1. Without loss of generality let us say the dotted line which contains block having the element n_3 passes through the block having the element n_1 and the dotted line which contains block having the element n_4 passes through the block having the element n_2. Now, we see the position of the block having element n_5. Let us first consider the case where block having n_5 have conflicting bit common with the block lying on the dotted line which contains block having an element from S_1. Further, let us consider that the dotted line which contains block having element n_5 intersects the block having element n_3 and n_4. Now, it is interesting to note that if two nonempty and an empty blocks intersect, then we can either send both nonempty blocks to table T_1, or we can send a nonempty block and an empty to table T_1. Let us first consider the case where we can send both the blocks having element n_3 and n_4 to table T_1. In this case, we send all the blocks having elements to table T_1, and all the empty blocks to table T_0. Now, without

loss of generality let us consider that we can send the blocks having element n_3 and the empty block lying on the dotted line which contains block having element n_5 to table T_1. In this case, we send the block having element n_2, n_4 and n_5 to table T_0, and all the blocks lying on the dotted line containing this block to table T_1. Further, we send the block having element n_3 to table T_1. Rest all the empty blocks are sent to table T_0. Let us now consider the case where dotted line which contains block having element n_5 do not pass through both the block having element n_3 and n_4. Now since dotted line which contains block having element n_5 do not intersect with both the blocks having elements n_4 and n_5 it can conflict with at most one block having element. Without loss of generality, let us say that empty block lying on the dotted line which contains block having element n_5 either conflict with a block having element n_3 or it does not. In this case, we send the blocks having elements n_1, n_3 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the blocks having element n_2 and n_4 to table T_1. Rest all the empty blocks are sent to table T_0. Let us now consider the case where block having element n_5 do not intersect with the dotted line which contains block having elements from S_1. Now let us consider the case where block having element n_5 do not conflict with block lying on the dotted line having elements from S_1. In this case, we send the block having element n_5 to table T_1 and all the empty blocks lying on the dotted line containing this block to table T_0. Now we see if the block having element n_5 have conflicting bit common either with block lying on the dotted line which contains block having element n_3 or n_4. Without loss of generality, let us say block having element n_5 have conflicting bit common with empty block lying on the dotted line which contains block having element n_3. In this case, we send the block having element n_1 and n_3 to table T_1. Further, we send the block having element n_2 and n_4 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. Rest all the empty blocks are sent to table T_0. If the block having element n_5, do not have conflicting bit common with block lying on the dotted line which contains block having element n_3 or n_4 then previous assignment works.

Let us consider the case where only one dotted line which has block having the element n_3 or n_4 passes through the block having elements from S_1. Without loss of generality let us say that dotted line which contains block having the element n_3 passes through the block having the element n_1. In this case, we see the position of the block

having the element n_5. Let us first consider the case where block having the element n_5 have conflicting bit common with block lying on the dotted line which contains blocks having elements from S_1. In this case, we send the block having element n_1, n_3, n_4 and n_5 to table T_0, and all the blocks lying on the dotted lines containing these blocks to table T_1. Further, we send rest all the empty blocks to table T_0.

Now let us consider the case where the block having the element n_5 do not have conflicting bit common with block lying on the dotted line which contains blocks having elements from S_1. Now we see whether the block having element n_5 conflicts with the dotted line which contain block having element n_3 or n_4. If the block having element n_5 conflicts with block lying on the dotted line which contains block having element n_3, then we send the blocks having elements n_2, n_4 and n_5 to table T_0, rest all the blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the block having element n_3 to table T_1, and rest all the empty blocks to table T_0. If the block having element n_5 conflicts with block lying on the dotted line which contains block having element n_4, then we send the blocks having elements n_4 and n_5 to table T_1, and rest all the blocks lying on the dotted lines containing these blocks to table T_1. Further, we send the blocks having elements n_1 and n_3 to table T_0, and all the blocks lying on the dotted line containing these blocks to table T_1. Rest all the empty blocks are sent to table T_0. If the block having element n_5 do not conflict with block lying on the dotted line which contains block having element n_3 or n_4, then we send the blocks having elements n_1, n_2, n_3 and n_5 to table T_1, and all the blocks lying on the dotted lines containing these blocks to table T_0. Further, we send the block having element n_4 to table T_0, and all the empty blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

If none of the dotted lines which contains block having elements n_3 or n_4 passes through block having element from S_1, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the dotted lines containing these blocks to table T_1. Now either the block having an element from S_1 have conflicting bit common with block lying on the dotted line which contains block having the element n_5 or it does not. Let us first consider the case where block having elements from S_1 have conflicting bit common with block lying on the dotted line which contains block having the element n_5. Without loss of generality let us say that block having element n_1 have

conflicting bit common with block lying on the dotted line which contains block having the element n_5. In this case, we send the block having the element n_1 to table T_0, and all the blocks lying on the dotted line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If none of the block having an element from S_1 has conflicting bit common with block lying on the dotted line which contains block having the element n_5, then we send the blocks having an element from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Now we are left with the case where none of the blocks having elements n_3, n_4 and n_5 have conflicting bit common. In this case, we send the blocks having elements to table T_1, and all the empty blocks to table T_0.

8.3 Appendix C

In this section, we discuss rest of the cases of Section 6.2.3.

Case 4 The elements in $S_1 = \{n_1, n_2, n_3\}$ lies in a superblock and the elements n_4 and n_5 in the different superblocks.

Case 4.1 Elements n_1, n_2 and n_3 lies on a same line.

Case 4.1.1 Both the elements n_4 and n_5 coincides with the blocks having elements from S_1 in table T_1. In this case, we send the blocks having elements n_4 and n_5 to table T_0, and the rest of the empty blocks which lie on the lines containing these blocks to table T_1. We send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to the table T_0.

Case 4.1.2 Only one of the blocks having an element say n_4 coincides with the block having an element from S_1 in table T_1. Without loss of generality, let us say that block having the element n_1 coincides with the block having the element n_4. In this case, we send the blocks having elements n_4 and n_5 to table T_0 and all the empty blocks lying on the line containing these blocks to table T_1. Now we see whether the line, which contains block having the element n_5 passes through the block having element elements from S_1 or not. Let us first consider a case where the line which contains block having the element n_5 passes through one of the blocks having elements from S_1, without loss of generality, let us say it passes through block having the element n_2. In this case, we send the block having the element n_2 to table T_0, and rest all the blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On another hand, if the line which contains block having the element n_5 does not pass through any of the block having element from S_1, then we send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 4.1.3 None of the blocks which contains element n_4 or n_5 coincides with the blocks which contains an element from S_1 in table T_1. This case is the same as Case

3.1.3.

Case 4.2 Two elements say n_1 and n_2 lies on a same line, and the element n_3 lies on a different line.

Case 4.2.1 Blocks having the elements n_4 and n_5 coincides with the blocks having elements from S_1 in table T_1.

Case 4.2.1.1 Blocks having the elements n_4 and n_5 coincides with the block having elements n_1 and n_2 in table T_1. In this case, we send the blocks having elements n_4 and n_5 to table T_0, and all the empty blocks lying on the lines containing these blocks to table T_1. Further, we send the blocks having elements n_1 and n_2 to table T_1. We send the block that contains the element n_3 to table T_0, and the rest of the empty block lying on the line containing this block to table T_1. Rest all the empty blocks are sent to the table T_0.

Case 4.2.1.2 Blocks having elements n_4 and n_5 coincides with the block lying on the different line, say n_1 and n_3 in table T_1. In this case, we send the blocks having elements n_4 and n_5 to table T_1 and the rest of the empty blocks lying on the line containing these blocks to table T_0 . We send the blocks having elements n_1 and n_3 to the table T_0 and the rest of the blocks lying on these lines to the table T_1. Rest all the empty blocks are sent to table T_0.

Case 4.2.1.3 Blocks having element n_4 and n_5 coincides in table T_1. If it coincides with the block having the element n_1 or n_2, then we send both the block having the element n_4 and n_5 to table T_0, and the rest of the empty block lying on the line containing these blocks to table T_1. Further, we send the blocks having the element n_1 and n_2 to table T_1. We send the block that contains element n_3 to the table T_0, and the rest of the empty block, which lies on the line containing this block to the table T_1. Rest all the empty blocks are sent to the table T_0.

If the blocks containing elements n_4 and n_5 coincides with the block which contains element n_3 in table T_1, then we send the block containing element n_3 to the table T_0,

and the rest of the empty block, which lie on the line containing this block to the table T_1. Further, we send the block containing n_4 to table T_1, and the rest of the empty block lying on the line containing this block to the table T_0. We send the block having the element n_5 to the table T_0, and the rest of the empty block lying on the line containing this block to table T_1. Now we see whether the line which contains block having the element n_5 passes through the block having the element n_1 or n_2. Without loss of generality, let us first consider the case where the line which contains block having the element n_5 passes through the block having the element n_1. In this case, we send the block having the element n_1 to table T_0, and all the blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If the line which contains block having element n_5 does not pass through the blocks having elements n_1 or n_2, then we send the blocks having elements n_1 and n_2 to table T_1, and rest all the empty blocks to table T_0.

Case 4.2.2 Only one block having an element say n_4 coincides with the block having the element from S_1 in table T_1. Let us first consider the case where block having element n_4 coincide with the block having element n_1 or n_2. Without loss of generality, let us say block having the element n_4 coincide with the block having the element n_1. In this case, we send the block having the element n_4 to table T_1, and all the empty blocks lying on the line containing this block to table T_0. We send the block having the element n_1 to table T_0, and rest all the blocks lying on the line containing this block to the table T_1. Now we see the position of the block having the element n_5. If it lies on the line which contains block having the element n_1, then we send the block having the element n_5 to table T_0, and all the empty blocks lying on the line containing this block to the table T_1. We send the block having the element n_3 to table T_0, and all the empty blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On another hand, if the block having the element n_5 does not lie on the line, which contains block having the element n_1, then we send the block having the element n_3 and n_5 to table T_1. Rest all the empty blocks are then sent to table T_1.

Now let us consider the case where block having the element n_4 coincide with the block having the element n_3. In this case, we send the block having the element n_4 to table T_1, and rest all the empty blocks lying on the line containing this block to table T_0.

We send the block having the element n_3 to table T_0, and all the empty blocks lying on the line containing this block to table T_1. Now we see the position of the block having the element n_5. If the block having the element n_5 lies on the line which contains block having the element n_3 , then we send the block having the element n_5 to the table T_0, and all the empty blocks lying on the line containing this block to the table T_1. If the block having the element n_1 lies on the line which contains block having the element n_5, then we send the block having the element n_1 to table T_0, and rest all the blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to the table T_1. Similar is the case if the block having the element n_2 lies on the line, which contains block having the element n_5. If the block having the element n_5 does not lie on the line which contains block having the element n_3, then we send the blocks having the elements n_1, n_2 and n_5 to table T_1, and rest all the empty blocks to table T_0.

Case 4.2.3 None of the blocks having the elements n_4 or n_5 coincides with blocks having elements from S_1 in table T_1. This case is the same as Case 3.1.3.

Case 4.3 All the elements belonging to S_1 lies on the different lines. In this case, we send all the blocks having elements to table T_0, and all the empty blocks to table T_1.

Case 5 Two elements $S_1 = \{n_1, n_2\}$ lies in a superblock other two elements $S_2 = \{n_3, n_4\}$ lies in other superblock, and an element $S_3 = \{n_5\}$ in a different superblock.

Case 5.1 Elements belonging to S_1 lie on the same line, elements belonging to S_2 lies on a line.

Case 5.1.1 Two blocks having elements from S_2 and S_3 coincides with the blocks having elements from S_1 in table T_1. Without loss of generality, let us say that block having element n_3 coincides with the block having the element n_1, and the block having the element n_5 coincides with the block having the element n_2. In this case, we send the block having the element n_4 to table T_0, and the rest of the block lying on the line containing this block to table T_1. We send the block having the element n_1 to table T_0, and the rest of the block lying on this line to table T_1. Further, we send the block having

the element n_5 to table T_0, and the rest of the empty block lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

If the blocks from S_2 and S_3 coincides with the same block having the element from S_1, then we send the coinciding block having elements from S_2 and S_3 to table T_0, and the rest of the blocks lying on these lines to table T_1. Furthermore, we send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 5.1.2 Only one block having the element from S_2 or S_3 coincides with the block having the element from S_1.

Case 5.1.2.1 Block having an element from S_2 coincides with the block having an element from S_1. Without loss of generality, let us say that the block having the element n_3 coincides with the block having the element n_1. Now, we can have two cases, either the block containing element n_5 coincides with the block containing element n_4 or it does not.

If the block containing n_5 coincides with the block containing n_4, then we send the block containing n_4 to table T_0, and the rest of the block lying on the line containing this block to the table T_1. We send the block containing n_5 to table T_1, and the rest of the block lying on the line containing this block to table T_0. Further, we send the block having the element n_1 to table T_0, and the rest of the block lying on this line to table T_1. Rest all the empty block are sent to table T_0.

If the block containing n_5 does not coincide with the block having the element n_4, then we send the block having the element n_3 to table T_0, and the rest of the block lying on this line to table T_1. Now we see the position of the block that contains the element n_5 to make the assignment. If the block which contains element n_5 lies on the line which contains block having elements from S_1, then we send the block having the element n_5 to table T_1, and rest all the empty blocks lying on the line which contains this block to table T_0. Further, we send blocks having elements from S_1 to table T_1, and the rest of the empty blocks lying on the line which contains this block to table T_0. For the rest of the empty blocks, we send it to table T_0.

Rest for all other positions of block having element n_5, we send block having elements n_3, n_4 and n_5 to table T_1, and all the empty blocks lying on the lines containing

these blocks to table T_0. Further, we send the block having the element n_1 to table T_0, and all other blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

Case 5.1.2.2 Block having an element from S_3 coincides with the block having an element from S_1. Without loss of generality, let us say that block having element n_5 coincides with the block having the element n_1. In this case, we send the block having the element n_5 to table T_0, and the rest of the block lying on the line, which contains this block to table T_1. Now we see the position of the block having the element from S_2.

One of the blocks having an element from S_2 coincides with the empty block on the line, which contains block having the element n_5. Without loss of generality, let us say that block having element n_3 coincides with an empty block on the line, which contains block having the element n_5. In this case, we send the block having an element n_3 to table T_0, and the rest of the block lying on the line containing this block to table T_1. Now we see the position of the block having element n_4, if it lies on the line which contains block having element from S_1, then we send the block having element n_1 and n_2 to table T_1, and the rest of the empty block lying on the line containing these blocks to table T_0. Rest all the empty blocks are sent to table T_0. On another hand, if the block having the element n_4 does not lie on the line which contains block having the element from S_1, then we send the block having the element n_2 to table T_0, and the rest of the block lying on the line containing this block to table T_1. We send the rest of the empty block to table T_0.

If the block having the element from S_2 do not lie on the line which contains block having the element n_5, then we send the blocks having an element from S_2 to table T_1, and the rest of the empty blocks on the line which contains these blocks to table T_0. Also, we send the blocks having the element n_1 and n_2 to table T_1. We send rest of the empty blocks to table T_0.

Case 5.1.3 None of the block having elements from S_2 or S_3 coincides with the blocks having element from S_1 in the table T_1. In this case, we can have two cases. Either the block having the element from S_3 coincides with the block having the element from S_2, or it does not.

Let us first consider the case where a block having the element from S_3 coincides with one of the blocks having an element from S_2. Without loss of generality, we can say that block having the element n_5 coincides with the block having the element n_4. In this case, we send the block having the element n_5 to table T_1, and the rest of the empty block lying on the line, which contains this block to table T_0. We send the block having the element n_4 to table T_0, and the rest of the block lying on the line, which contains this block to table T_1. Now we see the positions of the blocks having an element from S_1. Let us first consider the case where a block having the element from S_1 lies on the line, which contains blocks having elements from S_2. Without loss of generality, let us say block having the element n_1 lies on the line, which contains blocks having elements from S_2. In this case, we send the block having the element n_1 to the table T_0, and all other block lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if the blocks having elements from S_1 do not lie on the line which contains blocks having elements from S_2, then we send the block having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

If the block having an element from S_3 do not coincide with a block having an element from S_2 then we send all the block having elements to table T_1, and all the empty blocks to table T_0.

Case 5.2 Now we consider the case where one of the sets having elements lie on a line and other set having elements lie on the different lines. Without loss of generality, let us consider the case where elements belonging to S_1 lies on a line and the elements belonging to S_2 lies on the different lines.

Case 5.2.1 All the blocks having elements from S_2 and S_3 coincides with the block having an element from S_1. In this case, we send the block having an element from S_2 and S_3 to table T_0, and rest of the empty block lying on the lines containing these blocks to table T_1. Further, we send the blocks having elements from S_1 to table T_1, and the rest of the empty blocks lying on the line, which contains these blocks to table T_0. We send the rest of the empty blocks to table T_0.

Case 5.2.2 Two blocks having elements from S_2 and S_3 coincides with the block hav-

ing an element from S_1. Now here we can have two cases either those two blocks have elements belonging to S_2 or we can have one element belonging to S_2 and other to S_3.

Let us first consider the case where two blocks having elements from S_2 coincides with a block having an element from S_1. without loss of generality, let us say that block having element n_3 coincides with the block having the element n_1, and the block having the element n_4 coincides with the block having n_2. In this case, we send the block having elements n_3 and n_4 to table T_0, and the rest of the empty block lying on the lines containing these blocks to table T_1. Now we see the position of the block having the element n_5. At this point, we can have two cases either the line which contains the block having the element n_5 passes through one of the block having an element from S_1 or it does not. Let us first consider the case where it passes through one of the blocks having elements from S_1. Without loss of generality, let us say that the line which contains block having the element n_5 passes through the block having the element n_1. In this case, we send the block having the element n_5 to table T_0, and the rest of the block lying on the line, which contains this block to table T_1. Further, we send the block having the element n_1 to table T_0, and the rest of the block lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On another hand, if the line which contains the block having the element n_5 do not pass through blocks having elements from S_1, then we send the block having the element n_5 to table T_0, and rest of the empty blocks on the line containing this block to table T_1. Furthermore, we send the blocks having elements from S_1 to table T_1, and the rest of the empty blocks to table T_0.

Now we consider the case where one of the blocks having an element from S_2, and another block having an element from S_3 coincides with the block having an element from S_1. Without loss of generality, say blocks having the element n_3 and n_5 coincides with a block(blocks) having an element(elements) from S_1. Now here we can have two cases, either n_3 and n_5 coincides with the same block having an element from S_1, or it coincides with different blocks having elements from S_1. Let us first consider the case where blocks having element n_3 and n_5 coincides with same block having an element from S_1, say n_1. In this case, we send the blocks having elements from S_2 and S_3 to table T_0, and the rest of the empty blocks lying on the lines containing these blocks to table T_1. Further, we send the block having the element n_2 to table T_0, and the rest of the block, which lies on the line containing this block to table T_1. Rest all the empty blocks are sent to table

T_0. Now we consider the case where blocks having elements n_3 and n_5 coincides with different blocks having elements from S_1. Without loss of generality, let us say that block having element n_3 coincide with the block having the element n_1, and the block having the element n_5 coincide with the block having the element n_2. In this case, we send the blocks having elements from S_2 and S_3 to table T_0, and all the empty blocks lying on the line containing these blocks to table T_1. Further, we send the block having the element n_2 to table T_0, and rest of the block lying on the line which contains this block to table T_1.

Case 5.2.3 Only one block having element from S_2 or S_3 coincides with the block having an element from S_1 in table T_1. Now here we can have two cases, either the block having an element from S_2 coincides, or the block having an element from S_3 coincides with the block having an element from S_1.

Case 5.2.3.1 A block having an element from S_2 coincides with a block having an element from S_1 in table T_1. Without loss of generality, let us say that the block having the element n_3 coincides with the block having the element n_1. Now we see the position of the blocks having the element n_4 and n_5. Let us first consider the case where blocks having elements n_4 and n_5 coincide.

If the blocks having elements n_4 and n_5 coincides on the line which contains blocks having elements from S_1, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the lines containing these blocks to table T_1. We send the block having element n_1 and n_2 to table T_1. Rest all the empty blocks are sent to table T_0.

If the blocks having elements n_4 and n_5 coincides outside the line which contains blocks having elements from S_1, then we send the blocks having elements n_2, n_3 and n_4 to table T_0, and all the blocks lying on the lines containing these blocks to table T_1. We send the blocks having elements n_1 and n_5 to table T_1. Rest all the empty blocks are sent to table T_0.

Now we are left with the case where blocks having elements n_4 and n_5 do not coincide. This can have several cases. Let us first consider the case where both the blocks having elements n_4 and n_5 lies on the line, which contains blocks having elements from S_1. In this case, we send the blocks having elements n_3, n_4, and n_5 to table T_0, and

all the blocks lying on the lines containing these blocks to table T_1. We send the blocks having elements n_1 and n_2 to table T_1. Rest all the empty blocks are sent to table T_1.

Now let us consider the case where n_4 and n_5 do not lie on the line, which contains block having elements from S_1. In this case, we send the blocks having elements n_3, n_4, and n_5 to table T_1, and all the blocks lying on the lines containing these blocks to table T_0. Block having the element n_1 is sent to table T_0, and rest all the blocks lying on the line containing this block is sent to table T_1. Rest all the empty blocks are sent to table T_0.

Now, we can also have a case where only one block having the element n_4 or n_5 lies on the line, which contains blocks having elements from S_1. Let us first consider the case where block having the element n_4 lies on the line, which contains block having elements from S_1. In this case, we send the blocks having elements n_3, n_4, and n_5 to table T_0, and all other blocks lying on the line containing these blocks to table T_1. Now, we see whether the block having elements from S_1 lies on the line, which contains block having the element n_5. Without loss of generality, let us say block having the element n_1 lies on the line, which contains block having the element n_5. In this case, we send the block having the element n_1 to table T_0, and all other blocks lying on the line, which contains this block to table T_0. Rest all the empty blocks are sent to table T_1. On another hand, if the line which contains block having the element n_5 do not pass through block having element from S_1, then we send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0. Now let us consider the case where block having the element n_5 lies on the line, which contains block having elements from S_1. In this case, we send the blocks having elements n_2, n_3, n_4, and n_5 to table T_0, and all other blocks lying on the line containing these blocks to table T_1. Rest all the empty blocks are sent to table T_0.

Case 5.2.3.2 The block having element from S_3 coincides with block having element from S_1 in table T_1. Without loss of generality, let us say that block having element n_5 coincides with the block having the element n_1. Now we see the position of the blocks having elements from S_2.

Both the blocks having an element from S_2 lies on the line, which contains block having an element from S_1 in table T_1. In this case, we send the blocks having the

elements n_3, n_4, and n_5 to table T_0, and all the blocks lying on the lines containing these blocks to table T_1. Blocks having elements from S_1 are sent to table T_1 and rest all the empty blocks are sent to table T_0.

Both the blocks having an element from S_2 do not lie on the line, which contains blocks having an element from S_1. In this case, we send the blocks having elements n_3, n_4, and n_5 to table T_1, and all the empty blocks lying on the lines containing these blocks to table T_0. Further, we send the block having the element n_1 to table T_0, and all the blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

Now let us consider the case where only one block having the element from S_2 lies on the line, which contains blocks having elements from S_1 in table T_1. Without loss of generality, let us say that block having element n_3 lies on the line, which contains block having an element from S_1. In this case, we send the blocks having elements n_4 and n_5 to table T_1, and all the empty blocks lying on the lines which contain these blocks to table T_0. We send the blocks having the element n_1 and n_3 to table T_0, and rest all the blocks lying on the lines, which contain these blocks to table T_1. Rest all the empty blocks are sent to table T_0.

Case 5.2.4 None of the blocks having element from S_2 or S_3 coincides with the block having elements from S_1 in table T_1. In this case, we can have either the block having the element n_5 coincides with the block having elements from S_2 or it does not.

Let us first consider the case where the block having the element n_5 coincides with one of the blocks having the element from S_2. Without loss of generality, let us say that the block having the element n_5 coincides with the block having the element n_3. Now we see whether the block having the element n_4 lies on the line, which contains block having elements from S_1 or not. Let us first consider the case where block having the element n_4 lies on the line, which contains block having an element from S_1. In this case, we send the block having the element n_3 to table T_0, and all the blocks lying on the line containing this block to table T_1. We send the block having the element n_5 to table T_1, and all the blocks lying on the line, which contains this block to table T_0. Now we see whether any of the block having elements from S_1 lies on the line having the element n_3 or not. Without loss of generality, let us say the block having the element

n_1 lies on the line, which contains block having the element n_3. In this case, we send the blocks having the elements n_1 and n_4 to table T_0, and all other blocks lying on the lines which contain these blocks to table T_1. Rest all the empty blocks are sent to table T_0. If none of the blocks having elements from S_1 lies on the line, which contains block having the element n_3, then we send the blocks having elements n_1, n_2, and n_4 to table T_1, and rest all the empty blocks to table T_0. Now let us consider the case where block having the element n_4 do not lie on the line, which contains block having elements from S_1. In this case, the assignment made in the previous paragraph will work if we send the block having the element n_4 to table T_1, and all the blocks lying on the line containing this block to table T_0. Now, we are left with the case where block having the element n_5 do not coincide with the block having an element from S_2. In this case, we send the blocks having elements to table T_1, and all the empty blocks to table T_0.

Case 5.3 All the elements belonging S_1, S_2 and S_3 lies on the different line. In this case, we send the blocks having elements to table T_0 and all the empty blocks to table T_1.

Case 6 Two elements belonging to $S_1 = \{n_1, n_2\}$ lies in a same superblock, and the element n_3, n_4 and n_5 to the different superblocks. If the blocks having elements from S_1 lies on the different lines, then the assignment made in Case 5.3 can be used. So let us consider the case where blocks having the element from S_1 lies on the same line.

Case 6.1 Only one block having element from n_3, n_4 and n_5 coincide with the block having element from S_1. Without loss of generality let us say block having the element n_3 coincide with the block having the element n_1.

Case 6.1.1 Blocks having elements n_4 and n_5 coincide.

Case 6.1.1.1 Blocks having elements n_4 and n_5 coincides on the line which contains blocks having elements from S_1. In this case, we send the blocks having elements n_3, n_4, and n_5 to table T_0, and the rest of the empty blocks lying on the lines containing these blocks to table T_1. Blocks having the elements n_1 and n_2 are sent to table T_1, and rest

all the empty blocks are sent to table T_0.

Case 6.1.1.2 Blocks having elements n_4 and n_5 coincides outside the line which contains element from S_1. Now here we can have several cases depending upon whether the lines which contain blocks having elements n_4 and n_5 passes through blocks having elements from S_1 or not.

Without loss of generality, let us say that line which contains block having the element n_4 passes through the block having the element n_1, and the line which contain block having the element n_5 passes through the block having the element n_2. Now in this case we send the blocks having elements n_4 and n_1 to table T_1. Further, we send the blocks having elements n_3, n_5 and n_2 to table T_0, and rest of the blocks lying on the lines containing these blocks to table T_1. We send rest of the empty blocks to table T_0.

Now consider a case where only one line which contains a block from n_4 or n_5 passes through the block having an element from S_1. Without loss of generality, let us say block having the element n_4 passes through the block having an element from S_1. Lets first consider the case where the line which contains block having the element n_4 passes through the block having the element n_2. In this case, we send the blocks having elements n_2, n_3, n_4, and n_5 to table T_0, and rest of the blocks lying on the lines containing these blocks to table T_1. Further, we send the block having the element n_1 to table T_1, and the rest of the empty blocks to table T_0. Now without loss of generality, we can also have a case where block having the element n_4 passes through the block having the element n_1. In this case, we send the block having the elements n_1, n_3, n_4, and n_5 to table T_0, and the rest of the blocks lying on the lines containing these blocks to table T_1. We send the block having the element n_2 to table T_1, and the rest of the empty blocks to table T_0. If none of the lines which contains blocks having elements n_4 and n_5 passes through blocks having elements from S_1, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the blocks lying on the lines containing these blocks to table T_1. Further, we send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 6.1.2 Blocks having the elements n_4 and n_5 do not coincide.

Case 6.1.2.1 Both the blocks having element n_4 and n_5 lies on the line which contains element from S_1. In this case, we send the blocks having elements n_3, n_4, and n_5 to table T_0, and rest of the empty blocks lying on the line containing these blocks to table T_1. Further, we send the blocks having elements n_1 and n_2 to table T_1, and rest of the empty blocks to table T_0.

Case 6.1.2.2 One of the blocks having an element n_4 or n_5 lies on the line, which contains blocks having elements from S_1 and other lies outside of it. Without loss of generality, let us say block having the element n_4 lies on the line, which contains blocks having elements from S_1, and block having the element n_5 lies outside it. In this case, we send the blocks having elements n_3, n_4, and n_5 to table T_0, and all the blocks lying on the lines containing these blocks to table T_1. If any block having elements from S_1 lies on the line, which contains block having elements n_5, then we send that block to table T_0, and all other blocks lying on the line containing that block to table T_1. Rest all the empty blocks are sent to table T_1. On another hand, if none of the blocks having elements from S_1 lies on the line which contains block having elements n_5, then we send the block having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

Case 6.1.2.3 None of the blocks having elements n_4 or n_5 lies on the line, which contains blocks having elements from S_1. In this case we send the blocks having elements n_2, n_3, n_4 and n_5 to table T_1. Further, we send the block having the element n_1 to table T_0, and the rest of the empty blocks lying on the line, which contains this block to table T_1. Rest all the empty blocks are sent to table T_0.

Case 6.2 Two blocks having elements from n_3, n_4 or n_5 coincides with the block having elements from S_1. Without loss of generality, let us say that blocks having elements n_3 and n_4 coincides with the block having elements from S_1. Now, here we can have two cases, either the blocks having elements n_3 and n_4 coincides with the same block having an element from S_1 or it coincides with the different blocks having elements from S_1.

Let us first consider the case where blocks having the element n_3 and n_4 coincides with different blocks having elements from S_1. Without loss of generality, let us say that block having element n_3 coincides with the block having the element n_1 and the

block having the element n_4 coincides with the block having the element n_2. Now we see position of the block having element n_5. The block containing element n_5 can either lie on the line, which contains blocks having element from S_1 or not. Let us first consider the case where block having element n_5 do not lie on the line , which contains block having element from S_1. Further, let us consider the intersection of the line, which contains block having the element n_5 from the blocks having elements from S_1. Without loss of generality let us say that the line which contains block having element n_5 passes through the block having element n_1, in this case, we send the blocks having elements n_1, n_3, n_4 and n_5 to table T_0, and all other blocks lying on the lines which contains these blocks to table T_1. Rest all the empty blocks are sent to table T_0. On the another hand, if the block having element n_5 do not pass through the blocks having elements from S_1, then we send the blocks having n_3, n_4 and n_5 to table T_0, and all other blocks lying on the lines which contains these blocks to table T_1. Further, we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0. If the block having element n_5 lies on the line which contains blocks having elements from S_1, then we send the blocks having elements n_3, n_4 and n_5 to the table T_0, and all the empty blocks lying on these lines to table T_1. Further, we send the blocks having elements from S_1 to table T_1, and rest all the empty blocks to table T_0.

Now we are left with the case where blocks having the element n_3 and n_4 coincides with only one block having an element from S_1. Without loss of generality, let us say that blocks having elements n_3 and n_4 coincides with the block having the element n_1. In this case, we see the position of the block having the element n_5. If the block having the element n_5 lies on the line which contains blocks having elements from S_1, then we send the block having the element n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the lines containing these blocks to table T_1. We send the blocks having elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0. Further, if the block having the element n_5 do not lie on the line which contains blocks having element from S_1 then we send the block having element n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the lines containing these blocks to table T_1. Now we see whether the line which contains block having the element n_5 passes through a block having an element from S_1 or not. Let us first consider the case where the line which contains block having the element n_5 passes through a block having an element from S_1. Without loss

of generality, let us say that the line which contains block having the element n_5 passes through the block having the element n_1. In this case, we send the block having the element n_1 to table T_0, and all other blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On another hand, if the line which contains block having the element n_5 do not pass through the block having an element from S_1, then we send the blocks having the elements from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

Case 6.3 All the three blocks having elements n_3, n_4 and n_5 coincides with the blocks having elements from S_1. In this case, we send the blocks having elements n_3, n_4, and n_5 to table T_0, and all the blocks lying on the lines containing these blocks to table T_1. Blocks having elements from S_1 are sent to table T_1. Rest all the empty blocks are sent to table T_1.

Case 6.4 None of the blocks having elements from n_3, n_4 and n_5 coincides with the blocks having elements from S_1.

Let us first consider the case where all the elements n_3, n_4 and n_5 coincides. If all of them coincides on the line which contains blocks having elements from S_1, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and rest of the empty blocks lying on the lines containing these blocks to table T_1. Further, we send the blocks having elements n_1 and n_2 to table T_1. Rest all the empty blocks are sent to table T_0.

If the blocks having elements n_3, n_4, and n_5 coincide outside the line, which contains blocks having elements from S_1, then we see the intersection of the lines containing these blocks with the blocks having elements from S_1. Now since all the blocks having elements from n_3, n_4, and n_5 coincides at most two lines having blocks containing these elements can intersect with the blocks having elements from S_1. Without loss of generality, let us say that the line which contains block having the element n_3 passes through the block having the element n_1, and the line which contains block having the element n_4 passes through the block having the element n_2. In this case we send the blocks having element n_1 and n_3 to table T_1. Further, we send the blocks having elements n_2, n_4, and n_5 to table T_0, and the rest of the empty blocks lying on the line containing these blocks to table T_1. Rest all the empty blocks are sent to table T_0.

Without loss of generality, let us now consider a case where only one line having element say n_3 passes through the block having element, say n_1, then we send the blocks having elements n_1, n_2 and n_3 to table T_1. Further, we send the blocks having elements n_4 and n_5 to table T_0, and the rest of the empty blocks lying on the lines containing these blocks to table T_1. Rest all the empty blocks are sent to table T_0.

If none of the lines containing blocks having elements n_3, n_4 and n_5 passes through the blocks having element from S_1, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and rest of the empty blocks lying on the line containing these blocks to table T_1. Further, we send the blocks having element n_1 and n_2 to table T_1, and rest of the empty blocks to table T_0.

Now let us consider the case where only two blocks having elements from n_3, n_4, and n_5 coincides. Without loss of generality, let us say the blocks having elements n_3 and n_4 coincide. Now we see the intersection of lines containing the blocks n_3 and n_4. Let us first consider the case where both the lines containing blocks n_3 and n_4 passes through the blocks having elements from S_1. Without loss of generality, let us say the line which contains block having the element n_3 passes through the block having the element n_1, and the line which contains block having the element n_4 passes through the block having the element n_2. In this case, we send the blocks having elements n_1 and n_3 to table T_0, and the rest of the blocks lying on the line containing these blocks to table T_1. We send the block having the element n_2 and n_4 to table T_1, and all other blocks lying on the lines containing these blocks to table T_0. Now, we see the position of the block having the element n_5. Kindly note that it is important to remember that no three lines passing through a point lie in the same plane in this case. If the block having the element n_5 lies on the line, which contains block having the element, then we send the block having the element n_5 to table T_0, and all other blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0.

Now let us consider the case where only one line which has block having the element n_3 or n_4 passes through the block having elements from S_1. Without loss of generality, let us say that block having the element n_3 passes through the block having the element n_1. In this case, we send the block having the element n_3 to table T_1, and all the blocks lying on the line containing these blocks to table T_0. The block having the element n_4 is sent to table T_0, and all the blocks lying on the line containing this block to table T_1.

Now we see the position of the block having the element n_5. Let us first consider the case where block having the element n_5 lies on the line, which contains blocks having elements from S_1. In this case, if the line which contains block having the element n_5 passes through the block having the element n_3 then we send the block having the element n_5 to table T_1, and all the block lying on the line containing this block to table T_0. The blocks having elements from S_1 are sent to table T_1, and all the blocks lying on the line containing these blocks to table T_0. Rest all the empty blocks are sent to table T_0. On another hand, if the line which contains block having the element n_5 do not pass through the block having the element n_3 then we send the block having n_5 to table T_0, and all the blocks lying on the line containing this block to table T_1. Further, the blocks having elements from S_1 are sent to table T_1, and all the blocks lying on the line containing these blocks to table T_0. Rest all the empty blocks are sent to table T_0. If the block having the element n_5 do not lie on the line which contains blocks having elements from S_1, then we send the block having the element n_5 to table T_0, and all the blocks lying on the line containing this block to table T_1. Now we see whether the block having elements from S_1 lies on the line, which contains block having the element n_5 or not. Without loss of generality, let us say that block having element n_1 lies on the line, which contains block having the element n_5. In this case, we send the block having the element n_1 to table T_0, and all the blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. On the other hand, if none of the blocks having element from S_1 lies on the line which contains block having element n_5, then we send the block having elements from S_1 to table T_1, and all the empty blocks lying on the line containing these blocks to table T_0. Rest all the empty blocks are sent to table T_0.

If none of the lines which contains block having elements n_3 or n_4 passes through block having element from S_1, then we send the blocks having elements n_3, n_4 and n_5 to table T_0, and all the empty blocks lying on the lines containing these blocks to table T_1. Now, either the block having an element from S_1 lies on the line which contains block having the element n_5 or it does not. Let us first consider the case where block having elements from S_1 lies on the line, which contains block having the element n_5. Without loss of generality, let us say that block having element n_1 lies on the line, which contains block having the element n_5. In this case, we send the block having the element n_1 to

table T_0, and all the blocks lying on the line containing this block to table T_1. Rest all the empty blocks are sent to table T_0. If none of the block having an element from S_1 lies on the line, which contains block having the element n_5, then we send the blocks having an element from S_1 to table T_1. Rest all the empty blocks are sent to table T_0.

We are left with the case where none of the blocks having elements from n_3, n_4, and n_5 coincides. In this case, we send the blocks having elements to table T_1, and the rest of the empty blocks to table T_0.

Case 7 All the blocks having elements n_1, n_2, n_3, n_4 and n_5 lies in the different superblocks. In this case, we can use the assignment made in Case 5.3.

Publications

Manuscripts Published

[1] Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. "Improved bounds for two query adaptive bitprobe schemes storing five elements". In: *Theoretical Computer Science* 838 (2020), pp. 208–230. ISSN: 0304-3975.

[2] Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. "Two improved schemes in the bitprobe model". In: *Theoretical Computer Science* 806 (2020), pp. 543–552. ISSN: 0304-3975.

[3] Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. "Improved Bounds for Two Query Adaptive Bitprobe Schemes Storing Five Elements". In: *The 13th Annual International Conference on Combinatorial Optimization and Applications, COCOA 2019.* (2019).

[4] Mirza Galib Anwarul Husain Baig, Deepanjan Kesh, and Chirag Sodani. "An Improved Scheme in the Two Query Adaptive Bitprobe Model". In: *Combinatorial Algorithms - 30th International Workshop, IWOCA 2019, Pisa, Italy, July 23-25, 2019, Proceedings*. 2019, pp. 22–34.

[5] Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. "Two New Schemes in the Bitprobe Model". In: *WALCOM: Algorithms and Computation - 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, March 3-5, 2018, Proceedings*. 2018, pp. 68–79.

[6] Mirza Galib Anwarul Husain Baig, Deepanjan Kesh, and Chirag Sodani. "A Two Query Adaptive Bitprobe Scheme Storing Five Elements". In: *WALCOM: Algorithms and Computation - 13th International Conference, WALCOM 2019, Guwahati, India, February 27 - March 2, 2019, Proceedings*. 2019, pp. 317–328.

Publication Under Submission

[1] Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. "Storing Four Elements in the Two Query Bitprobe Model." In: *"Discrete Applied Mathematics"*, (2019,**Under Minor Revision**).

Bibliography

[1] Harry Buhrman, Peter Bro Miltersen, Jaikumar Radhakrishnan, and Srinivasan Venkatesh. "Are bitvectors optimal?" In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*. 2000, pp. 449–458.

[2] Jaikumar Radhakrishnan, Venkatesh Raman, and S. Srinivasa Rao. "Explicit Deterministic Constructions for Membership in the Bitprobe Model". In: *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*. 2001, pp. 290–299.

[3] Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. "Two New Schemes in the Bitprobe Model". In: *WALCOM: Algorithms and Computation - 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, March 3-5, 2018, Proceedings*. 2018, pp. 68–79.

[4] "Two improved schemes in the bitprobe model". In: *Theoretical Computer Science* (2019).

[5] Patrick K. Nicholson. "Revisiting explicit adaptive two-probe schemes". In: *Inf. Process. Lett.* 143 (2019), pp. 1–3.

[6] Mirza Galib Anwarul Husain Baig, Deepanjan Kesh, and Chirag Sodani. "An Improved Scheme in the Two Query Adaptive Bitprobe Model". In: *Combinatorial Algorithms - 30th International Workshop, IWOCA 2019, Pisa, Italy, July 23-25, 2019, Proceedings*. 2019, pp. 22–34.

[7] Mirza Galib Anwarul Husain Baig, Deepanjan Kesh, and Chirag Sodani. "A Two Query Adaptive Bitprobe Scheme Storing Five Elements". In: *WALCOM: Algorithms and Computation - 13th International Conference, WALCOM 2019, Guwahati, India, February 27 - March 2, 2019, Proceedings*. 2019, pp. 317–328.

[8] Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. "Improved Bounds for Two Query Adaptive Bitprobe Schemes Storing Five Elements". In: *Combinatorial Optimization and Applications - 13th International Conference, COCOA 2019, Xiamen, China, December 13-15, 2019, Proceedings*. 2019, pp. 13–25.

[9] Noga Alon and Uriel Feige. "On the power of two, three and four probes". In: *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*. 2009, pp. 346–354.

[10] Marvin Minsky and Seymour Papert. *Perceptrons - an introduction to computational geometry*. MIT Press, 1987. ISBN: 978-0-262-63111-2.

[11] Patrick K. Nicholson, Venkatesh Raman, and S. Srinivasa Rao. "A Survey of Data Structures in the Bitprobe Model". In: *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*. 2013, pp. 303–318.

[12] Peter Elias and Richard A. Flower. "The Complexity of Some Simple Retrieval Problems". In: *J. ACM* 22.3 (1975), pp. 367–379.

[13] Andrew Chi-Chih Yao. "Should Tables Be Sorted?" In: *J. ACM* 28.3 (1981), pp. 615–628.

[14] Jaikumar Radhakrishnan, Smit Shah, and Saswata Shannigrahi. "Data Structures for Storing Small Sets in the Bitprobe Model". In: *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part II*. 2010, pp. 159–170.

[15] Mohit Garg and Jaikumar Radhakrishnan. "Set membership with a few bit probes". In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*. 2015, pp. 776–784.

[16] Mohit Garg. "The Bit-probe Complexity of Set Membership". PhD thesis. Tata Institute of Fundamental Research, Homi Bhabha Road, Navy Nagar, Colaba, Mumbai 400005, India: School of Technology and Computer Science, 2016.

[17] Deepanjan Kesh. "Space Complexity of Two Adaptive Bitprobe Schemes Storing Three Elements". In: *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*. 2018, 12:1–12:12.

[18] Deepanjan Kesh. "On Adaptive Bitprobe Schemes for Storing Two Elements". In: *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part I*. 2017, pp. 471–479.

[19] Ryan Blue. "The Bit Probe Model for Membership Queries: Non-Adaptive Bit Queries". MA thesis. College Park: Graduate School of the University of Maryland, 2009.

[20] Moshe Lewenstein, J. Ian Munro, Patrick K. Nicholson, and Venkatesh Raman. "Improved Explicit Data Structures in the Bitprobe Model". In: *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*. 2014, pp. 630–641.