An Improved Scheme in the Two Query Adaptive Bitprobe Model

Mirza Galib Anwarul Husain Baig, Deepanjan Kesh(✉), and Chirag Sodani

Indian Institute of Technology Guwahati, Guwahati 781039, Assam, India
{mirza.baig,deepkesh,chirag.sodani}@iitg.ac.in

Abstract. In this paper, we look into the adaptive bitprobe model that stores subsets of size at most four from a universe of size m, and answers membership queries using two bitprobes. We propose a scheme that stores arbitrary subsets of size four using $\mathcal{O}(m^{5/6})$ amount of space. This improves upon the non-explicit scheme proposed by Garg and Radhakrishnan [5] which uses $\mathcal{O}(m^{16/17})$ amount of space, and the explicit scheme proposed by Garg [4] which uses $\mathcal{O}(m^{14/15})$ amount of space. The proposed scheme also answers an open problem posed by Nicholson [8] in the affirmative. Furthermore, we look into a counterexample that shows that our proposed scheme cannot be used to store five or more elements.

Keywords: Data structure · Set membership problem · Bitprobe model · Adaptive scheme

1 Introduction

Consider the following static membership 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 $\mathcal{S}$?" Solutions to problems of this nature are called *schemes* in the literature. The resources that are considered to evaluate the schemes are the size of the data structure devised to store the subset $\mathcal{S}$, and the number of bits read of the data structure to answer the membership queries, called *bitprobes*. The notations for the space used and the number of bitprobes required are s and t, respectively. This model of the static membership problem is called the *bitprobe model*.

Schemes in the bitprobe model are classified as *adaptive* and *non-adaptive*. If the location where the current bitprobe is going to be depends on the answers obtained from the previous bitprobes, then such schemes are called *adaptive schemes*. On the other hand, 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*. Radhakrishnan *et al.* [9] 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.

C. J. Colbourn et al. (Eds.): IWOCA 2019, LNCS 11638, pp. 22–34, 2019.
https://doi.org/10.1007/978-3-030-25005-8_3

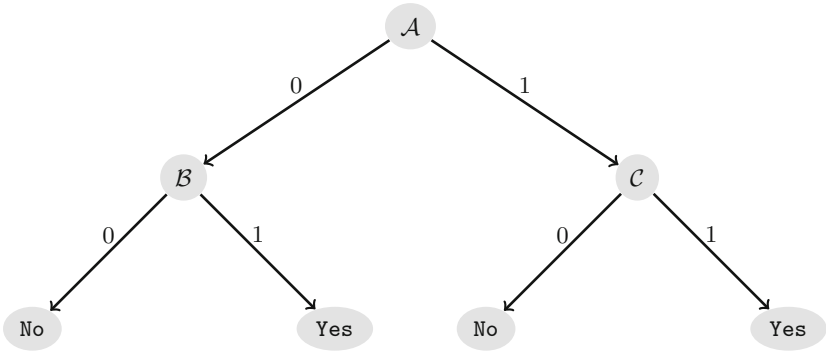


Fig. 1. The decision tree of an element.

1.1 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 – $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$ – arranged as shown in Fig. 1. Any element e in the universe $\mathcal{U}$ has a location in each of these three tables, which are denoted by $\mathcal{A}(e), \mathcal{B}(e)$, and $\mathcal{C}(e)$. 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 (Fig. 1). It works as follows. Given a query "Is x in $\mathcal{S}$?", the first bitprobe is made in table $\mathcal{A}$ at location $\mathcal{A}(x)$. If the bit stored is 0, the second query is made in table $\mathcal{B}$, else it is made in table $\mathcal{C}$. 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.

1.2 The Problem Statement

As alluded to earlier, 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 [1,7].

For subsets of size two ($n = 2$), Radhakrishnan *et al.* [9] 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 [9,10], it as yet remains unproven.

For subsets of size three ($n = 3$), Baig and Kesh [2] have recently proposed a scheme that takes $\mathcal{O}(m^{2/3})$ amount of space. It has been subsequently proven

by Kesh [6] that $\Omega(m^{2/3})$ is the lower bound for this problem. So, the space complexity question for $n = 3$ stands settled.

In this paper, 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 [5] 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 [4] 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 2), thus improving upon the existing schemes in the literature. Our claim is the following:

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

The existence of such a scheme also answers in the affirmative an open problem posed by Nicholson [8] which asked if a scheme using the idea of blocks due to Radhakrishnan *et al.* [9] 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 Sect. 5 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.

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 of the elements of the universe 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 1. *The size of our data structure is* $\mathcal{O}(m^{5/6})$.

2.1 Table $\mathcal{A}$

Suppose we are given the following universe of elements –

$$\mathcal{U} = \{\ 1, 2, 3, \ldots, m\ \}.$$

We partition the m elements of the universe into sets of size $m^{1/6}$. Borrowing the terminology from Radhakrishnan *et al.* [9], 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 $\mathcal{A}$ 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 $\mathcal{A}$ is $m^{5/6}$.

2.2 Superblocks

The blocks in our universe are partitioned into sets of size $m^{4/6}$. Radhakrishnan *et al.* [9] 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 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.

2.3 Table $\mathcal{C}$

Table $\mathcal{C}$ 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 $\mathcal{C}$ 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 $\mathcal{C}$. So, we can imagine table $\mathcal{C}$ 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 $\mathcal{C}$ is $m^{5/6}$.

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 are $1/i$. In the grid arrangement of the superblock (Sect. 2.2), we draw enough of these lines of slope $1/i$ so that every grid point falls on one of these lines. Figure 2 shows the grid and the lines.

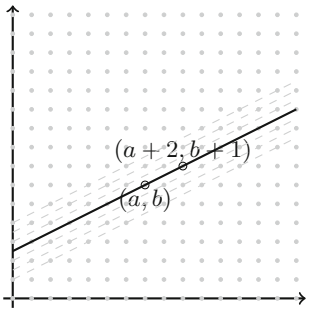


Fig. 2. The figure shows the grid for superblock 2, and some of the lines with slope 1/2. Note that the line passing through (a, b) intersects the y-axis at a non-integral point.

So, all lines of a given superblock has 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

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

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.

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

In the following three lemmas, we show the properties of this set of lines – they follow from elementary coordinate geometry.

Lemma 1. *Every line of L_i contains at least one point of the grid.*

Lemma 2. *Every point of the grid belongs to some line of L_i.*

Lemma 3. $| L_i | = (i + 1)(m^{2/6} - 1) + 1$.

Proof. The equality is a direct consequence of the definition of L_i (Eq. 2).

2.5 Table $\mathcal{B}$

In table $\mathcal{B}$, 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 $\mathcal{B}$; 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 3), so the total number of lines from all of the superblocks is

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

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

$$|\mathcal{C}| = \mathcal{O}(m^{5/6}).$$

2.6 Notations

As described in Sect. 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 $\mathcal{A}$ 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 $\mathcal{A}(s, x, y)$; or in other words, the element (s, x, y, i) will query the location $\mathcal{A}(s, x, y)$ in table $\mathcal{A}$.

In table $\mathcal{C}$, 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 $\mathcal{C}$ 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 $\mathcal{C}(x, y, i)$.

Table $\mathcal{B}$ 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 $\mathcal{B}$ 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 $\mathcal{B}$ is $\mathcal{B}(l_s(x, y), i)$.

3 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 (Fig. 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 $\mathcal{A}$ at location $\mathcal{A}(s, x, y)$. If the bit stored at that location is 0, we query in table $\mathcal{B}$ at $\mathcal{B}(l_s(x, y), i)$, otherwise we query table $\mathcal{C}$ at $\mathcal{C}(x, y, i)$. 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 $\mathcal{S}$.

4 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.

4.1 Intuition

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

Suppose, the block (s, x, y) of table $\mathcal{A}$ 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 $\mathcal{B}$, then we have to store the block (s, x, y) in table $\mathcal{B}$. So, we have to set in table $\mathcal{A}$ the following – $\mathcal{A}(s, x, y) = 0$. Thus, (s, x, y, i) upon first query will get a 0 and go to table $\mathcal{B}$. In table $\mathcal{B}$, we store the block (s, x, y) at the storage reserved for the line $l_s(x, y)$. Particularly, we have to set $\mathcal{B}(l_s(x, y), i) = 1$.

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 $\mathcal{B}$, and hence $\mathcal{A}(s, x', y')$ must be set to 1. If this is not the case, and $\mathcal{A}(s, x', y') = 0$, then the first query for the element (s, x', y', i) will get a 0, go to table $\mathcal{B}$ and query the location $\mathcal{B}(l_s(x', y'), i)$ which is same as $\mathcal{B}(l_s(x, y), i)$. 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 $\mathcal{C}$. If we decide to store the block (s, x, y) in table $\mathcal{C}$, we have to set $\mathcal{A}(s, x, y)$ to 1. In table $\mathcal{C}$, 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 $\mathcal{C}(x, y, i)$ to 1. This implies that all empty blocks from other superblocks having the same coordinate cannot be stored in table $\mathcal{C}$, and hence must necessarily be stored in table $\mathcal{B}$. To take an example, if (s', x, y) is empty, then it must stored it table $\mathcal{B}$, and hence $\mathcal{A}(s', x, y) = 0$.

To summarise, 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.

4.2 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 Sect. 4.3.

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.

We provide the detailed description of the cases when there are four distinct lines or when there is one line. The extended version of this paper (Baig *et. al* [3]) contains the cases of three lines and two lines. The cases described here would illustrate how to arrange the elements and how to argue its correctness.

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 (Sect. 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, $s_1 \neq s_2$.

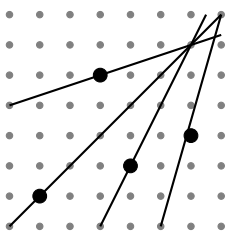


Table $\mathcal{B}$ has space to store one block for every line in every superblock. As the lines for the four members of $\mathcal{S}$ are distinct, the space reserved for the lines are also distinct. So we can store the four non-empty blocks in table $\mathcal{B}$, and all of the empty blocks in table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

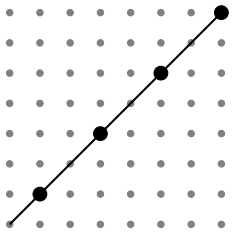
To achieve the objective, we set $\mathcal{A}(s_j, x_j, y_j) = 0$ for $1 \leq j \leq 4$, and set the bits in table $\mathcal{A}$ for every other block to 1. In table $\mathcal{B}$, we set the bits $\mathcal{B}(l_{s_j}(x_j, y_j), i_j) = 1$, for $1 \leq j \leq 4$, and all the rest of the bits to 0. In table $\mathcal{C}$, 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 $\mathcal{A}$. Its second query will be in table $\mathcal{C}$, and as all the bits of table $\mathcal{C}$ 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 $\mathcal{A}(s, x, y)$, which is same as $\mathcal{A}(s_1, x_1, y_1)$, is 0, and hence the second query for this element will be in table $\mathcal{B}$. 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 $\mathcal{B}(l_s(x, y), i) = \mathcal{B}(l_{s_1}(x_1, y_1), i)$. As the four lines for the four members of $\mathcal{S}$ are distinct, so $\mathcal{B}(l_{s_1}(x_1, y_1), i)$ will be 1 if and only if $i = i_1$. So, we will get a `Yes` answer for your 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 $\mathcal{C}$ 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 $\mathcal{B}$.

To this end, we set $\mathcal{A}(s_j, x_j, y_j) = 1$ for $1 \leq j \leq 4$, and the rest of the bits of table $\mathcal{A}$, which correspond to the empty blocks, to 0. In table $\mathcal{B}$, all bits are set to 0. In table $\mathcal{C}$, the bits corresponding to the four elements are set to 1, i.e. $\mathcal{C}(x_j, y_j, i_j) = 1$ for $1 \leq j \leq 4$. The rest of the bits of table $\mathcal{C}$ 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 $\mathcal{A}$ upon its first query, consequently go to table $\mathcal{B}$ 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 $\mathcal{A}(s, x, y) = \mathcal{A}(s_1, x_1, y_1)$, and hence it will get a 1 from table $\mathcal{A}$, and go to table $\mathcal{C}$ for its second query. In this table, it will query the location $\mathcal{C}(x, y, i)$, which is same as $\mathcal{C}(x_1, y_1, i)$. As the coordinates of the four members of $\mathcal{S}$ are distinct, $\mathcal{C}(x_1, y_1, i)$

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 $\mathcal{S}$.

4.3 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 a 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 $\mathcal{B}$ or $\mathcal{C}$, 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 summarise the result in the theorem below.

Theorem 2. *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 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.

5.1 The Arrangment

Consider four lines from four different superblocks which are arranged as shown in Fig. 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 $\mathcal{C}$. 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 $\mathcal{C}$.

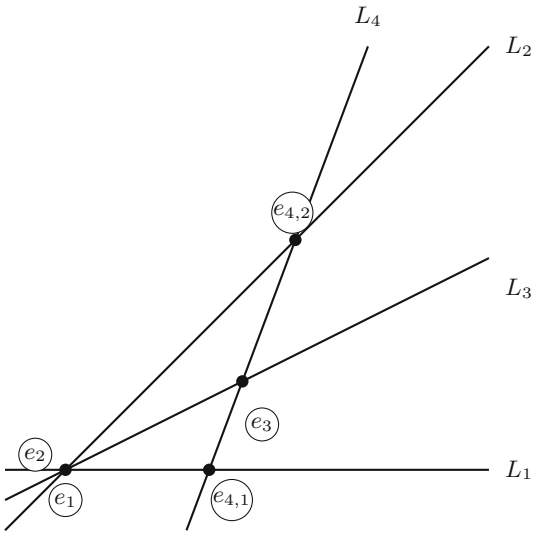


Fig. 3. Counterexample

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 $e_{4,2} = (s_4, x_{4,2}, y_{4,2}, i_{4,2})$.

5.2 The Contradiction

We can store the element e_1 of superblock s_1 in one of two tables $\mathcal{B}$ and $\mathcal{C}$. Let us assume that we store e_1 in table $\mathcal{B}$. 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 $\mathcal{B}$, and hence they must be stored in table $\mathcal{C}$.

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 $\mathcal{C}$, and hence must be stored in table $\mathcal{B}$. So, the other blocks of L_4 must be stored in table $\mathcal{C}$, 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 $\mathcal{B}$. So, all blocks on the line L_3 must now be store in table $\mathcal{C}$.

The element e_2 of the superblock s_2 falls on the line L_3 and hence must be stored in table $\mathcal{B}$. So, all blocks of line L_2 must be stored in table $\mathcal{C}$.

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

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

But we have started with the premise that e_1 is being stored in table $\mathcal{C}$, 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.

6 Conclusion

In this paper, 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 [8] about the existence of such a scheme that uses the ideas of blocks and superblocks due to Radhakrishnan *et al.* [9]. The technique used is that of arranging the blocks of a superblock in a two-dimensional grid, and grouping them along lines. We hope 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.

References

1. Alon, N., Feige, U.: 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, 4–6 January 2009, pp. 346–354 (2009)
2. Baig, M.G.A.H., Kesh, D.: Two new schemes in the bitprobe model. In: Rahman, M.S., Sung, W.-K., Uehara, R. (eds.) WALCOM 2018. LNCS, vol. 10755, pp. 68–79. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75172-6_7
3. Baig, M.G.A.H., Kesh, D., Sodani, C.: An improved scheme in the two query adaptive bitprobe model. CoRR abs/1812.04802 (2018). http://arxiv.org/abs/1812.04802
4. Garg, M.: The bit-probe complexity of set membership. Ph.D. thesis, School of Technology and Computer Science, Tata Institute of Fundamental Research, Homi Bhabha Road, Navy Nagar, Colaba, Mumbai 400005, India (2016)
5. Garg, M., Radhakrishnan, J.: 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, 4–6 January 2015, pp. 776–784 (2015)
6. Kesh, D.: 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, Ahmedabad, India, 11–13 December 2018, pp. 12:1–12:12 (2018)

7. Lewenstein, M., Munro, J.I., Nicholson, P.K., Raman, V.: Improved explicit data structures in the bitprobe model. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 630–641. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44777-2_52
8. Nicholson, P.K.: Revisiting explicit adaptive two-probe schemes. Inf. Process. Lett. **143**, 1–3 (2019)
9. Radhakrishnan, J., Raman, V., Srinivasa Rao, S.: Explicit deterministic constructions for membership in the bitprobe model. In: auf der Heide, F.M. (ed.) ESA 2001. LNCS, vol. 2161, pp. 290–299. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44676-1_24
10. Radhakrishnan, J., Shah, S., Shannigrahi, S.: Data structures for storing small sets in the bitprobe model. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6347, pp. 159–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15781-3_14