Optimal Bounds for the Predecessor Problem

Paul Beame*
Computer Science and Engineering
University of Washington
Seattle, WA, USA 98195-2350
beame@cs.washington.edu

Faith E. Fich†
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 1A4
fich@cs.utoronto.ca

Abstract

We obtain matching upper and lower bounds for the amount of time to find the predecessor of a given element among the elements of a fixed efficiently stored set. Our algorithms are for the unit-cost word-level RAM with multiplication and extend to give optimal dynamic algorithms. The lower bounds are proved in a much stronger communication game model, but they apply to the cell probe and RAM models and to both static and dynamic predecessor problems.

1 Introduction

Many problems in computer science involve storing a set S of integers and performing queries on that set. The most basic query is the *membership query*, which determines whether a given integer x is in the set. A *predecessor query* returns the predecessor $pred(x,S)$ of x in S, that is, the largest element of the set S that is less than x. If there is no predecessor (which is the case when x is smaller than or equal to the minimum element of S), then a default value, for example, 0, is returned.

Predecessor queries can be used to efficiently perform range searches (i.e. find all elements of S between given integers x and x'). They can also be used to obtain certain information about arbitrary integers (for example, their rank in S) that is only stored for the elements of S. Priority queues can be implemented using data structures that support insertion, deletion, and predecessor (or, equivalently, successor) queries.

The *static dictionary* problem is to store a fixed set and perform membership queries on it; the *static predecessor* problem allows predecessor queries. If insertions and deletions may also be performed on the set, we have the *dynamic dictionary* problem and the *dynamic predecessor* problem, respectively.

The complexities of searching (for example, performing membership or predecessor queries) and sorting have been long and well understood, under the assumption that elements are abstract objects which may only be compared. But many efficient algorithms, including hashing, bucket sort, and radix sort, perform word-level operations, such as indirect addressing using the elements themselves or values derived from them.

Often, such algorithms are applied only when the number of bits to represent individual elements is very small in comparison with the number of elements in the set. Otherwise, those algorithms consume huge amounts of space. For example, van Emde Boas trees [25, 24] can be used to perform predecessor queries on any set of integers from a universe of size N in $O(\log\log N)$ time, but they require $\Omega(N)$ space.

However, there have been important algorithmic breakthroughs showing that such techniques have more general applicability. For example, with two level perfect hashing [16], any n element set can be stored in $O(n)$ space and constant time membership queries can be performed. Fusion trees fully exploit unit-cost word-level operations and the fact that data elements need to fit in words of memory to store static sets of size n in $O(n)$ space and perform predecessor queries in $O(\sqrt{\log n})$ time [18].

For the static predecessor problem, it has been widely conjectured that the time complexities achieved by van Emde Boas trees and fusion trees are optimal for any data structure using a reasonable amount of space. We prove that this is NOT the case. Specifically, we construct a new data structure that stores n element sets of integers from a universe of size N in $n^{O(1)}$ space and performs predecessor queries in $O\left(\min\left\{\log\log N/\log\log\log N, \sqrt{\log n/\log\log n}\right\}\right)$ time. Using recent generic transformations of Andersson and Thorup [4, 6], the algorithm can be made dynamic and the space

*Research supported by the National Science Foundation under grants CCR-9303017 and CCR-9800124

†Research supported by grants from the Natural Sciences and Engineering Research Council of Canada and Communications and Information Technology Ontario. Part of the research was conducted while visiting Laboratoire de Recherche en Informatique, Université de Paris–Sud, Orsay, France.

improved to $O(n)$.

We also obtain matching lower bounds, improving Miltersen's $\Omega(\sqrt{\log\log N})$ and $\Omega((\log n)^{1/3})$ lower bounds in the powerful communication game model [21, 22]. The key to our improved lower bounds is to use a more complicated input distribution. Unfortunately, this leads to a more complicated analysis. However, the understanding we obtained from identifying and working with a hard distribution in the communication game model directly led to the development of our algorithm. This approach has also been used to obtain an $\Omega(\log\log d/\log\log\log d)$ lower bound for the approximate nearest neighbour problem over the universe $\{0,1\}^d$ [9].

2 Related Work

The simplest model in which the dictionary and predecessor problems have been considered is the *comparison model* in which the only operations allowed that involve x are comparisons between x and elements of S. Using binary search, predecessor and membership queries can be performed in $O(\log n)$ time on a static set of size n stored in a sorted array. Balanced binary trees (for example, Red-Black trees, AVL trees, or 2-3 trees) can be used in this model to solve the dynamic dictionary and predecessor problems in $O(\log n)$ time. A standard information theory argument proves that $\log_2 n$ comparisons are needed in the worst case even for performing membership queries in a static set.

One can obtain faster algorithms when x or some function computed from x can be used for indirect addressing. If S is a static set from a universe of size N, one can trivially perform predecessor queries using N words of memory in constant time: Simply store the answer to each possible query x in a separate location. This works for more general queries and if, as with membership queries, the number of bits, k, needed to represent each answer is smaller than the number of bits b in a memory word, the answers can be packed b/k to a word, for a total of $O(Nk/b)$ words. When the only queries are membership queries, updates to the table can also be performed in constant time.

If the universe size N is significantly less than 2^b, where b is the number of bits in a word, then packed B-trees [18, 3, 7, 23] can be time and space efficient. Specifically, using branching factor $B \le b/(1+\log_2 N)$, insertions, deletions, and membership and predecessor queries can be performed in $O(\log n/\log B)$ steps using $O(n/B)$ words.

The most interesting data structures are those that work for an arbitrary universe whose elements can fit in a single word of memory and use a number of words that is polynomial in n, or ideally $O(n)$. The static dictionary problem has optimal constant-time data structures with these properties: Constant time membership queries can be obtained for any set of size n using an $O(n^2)$ word hash table and a hash function randomly chosen from a suitable universal family [8]. Fredman, Komlòs, and Szeméredi [16] improved the space to $O(n)$ using two level perfect hashing. Their original algorithm used multiplication and division of $\log_2 N$ bit words to compute the hash functions. Recently it was shown that hash functions of the form

$$h(x) = ax \mod 2^{\lceil \log_2 N \rceil} \operatorname{div} 2^{\lceil \log_2 N \rceil - r}$$

for $r \le \lceil \log_2 n \rceil$, suffice for implementing the two level perfect hashing data structure [11, 13]. Notice that the evaluation of such functions does not depend on constant time division instructions; a right shift suffices. These algorithms can be made dynamic with the same constant time for membership queries and with constant expected amortized update time [13, 14, 12], by rehashing when necessary (using randomly chosen hash functions). For the static problem, Raman [23] proves that it is possible to choose such hash functions deterministically in $O(n^2 \log N)$ time.

Although hashing provides an optimal solution to the static dictionary problem, it is not directly applicable to the predecessor problem. Another data structure, van Emde Boas (or stratified) trees [25, 24], is useful for both static and dynamic versions of the predecessor problem. These trees support membership queries, predecessor queries, and updates in $O(\log\log N)$ time. The set is stored in a binary trie and binary search is performed on the $\log_2 N$ bits used to represent individual elements. The major drawback is the use of an extremely large amount of space: $\Omega(N)$ words.

Willard's x-fast trie data structure [26] uses the same approach, but reduces the space to $O(n\log N)$ by using perfect hashing to represent the nodes at each level of the trie. In y-fast tries, the space is further reduced to $O(n)$ by only storing $\Theta(n/\log N)$ approximately evenly spaced elements of the set S in the x-fast trie. A binary search tree is used to represent the subset of $O(\log N)$ elements of S lying between each pair of consecutive elements in the trie. Both of Willard's data structures perform membership and predecessor queries in worst-case $O(\log\log N)$ time, but use randomization (for rehashing) to achieve expected $O(\log\log N)$ update time.

Fusion trees, introduced by Fredman and Willard [18] use packed B-trees with branching factor $\Theta(\log n)$ to store approximately one out of every $(\log_2 n)^4$ elements of the set S. The effective universe size is reduced at each node of the B-tree by using carefully selected bit positions to obtain compressed keys representing the elements stored at the node. As above, binary search trees are used to store the roughly equal-sized sets of intermediate elements and a total of $O(n)$ space is used. Membership and predecessor queries take $O(\log n/\log\log n)$ time. Updates take $O(\log n/\log\log n)$ amortized time. (The fact that this bound is amortized arises from the $O((\log n)^4)$ time bound to update a node in the B-tree.) This data structure forms the basis of their ingenious $O(n\log n/\log\log n)$ sorting algorithm. For $n \le (\log_2 N)^{(\log_2\log_2 N)/36}$, the time bounds for fusion trees can be improved to $O(\sqrt{\log n})$ while retaining $O(n)$ space. This is done by using branching factor $\Theta(2^{\sqrt{\log n}})$ in the B-tree and storing $2^{\Theta(\sqrt{\log n})}$ elements in each of the binary search trees. For the remaining range, $n > (\log_2 N)^{(\log_2\log_2 N)/36}$, Willard's y-fast tries have query time and expected update time $O(\log\log N) = O(\sqrt{\log n})$.

Andersson [3] uses similar ideas to construct another data structure with $O(\sqrt{\log n})$ time membership and predecessor queries, expected $O(\sqrt{\log n})$ update time, and $O(n)$ space. B-trees with branching factor $O(b)$ can also be used to obtain a static predecessor data structure with $O(1 + \log n/\log b)$ query time that can be constructed in $O(n^4)$ time and space [4] and a dynamic predecessor data structure with $O(1+\log n/\log b)$ query time and expected update time [23].

The methods used to make these algorithms dynamic depend on the precise details of the underlying static data structures. Andersson's exponential search trees [4] can be used to transform any static data structure that performs membership and predecessor queries in time $T(n)$ into a linear space dynamic data structure with query and amortized update time $T'(n)$, where $T'(n) \leq T(n^{k/(k+1)}) + O(T(n))$, provided the static data structure can be constructed in n^k time and space, for some constant $k \geq 1$. Essentially, this is a recursive tree with a root of degree $\Theta(n^{1/(k+1)})$ that uses the static data structure to implement the search at the root. Global and partial rebuilding are used to update the data structure. Combined with fusion trees, packed B-trees, and x-fast tries, exponential search trees can be used to obtain a solution to the dynamic predecessor problem that uses worst case search time and amortized update time $O(\min\{\sqrt{\log n}, \log\log n + \log n/\log b\})$ and $O(n)$ space. Very recently, Andersson and Thorup [6] have combined a variant of exponential search trees with eager partial rebuilding to improve the resulting dynamic data structure, achieving worst-case instead of amortized bounds for update time.

One of the most natural and general models for proving lower bounds for data structures problems, and one that is ideally suited for representing word-level operations, is the *cell-probe* model, introduced by Yao [29]. In this model, there is a memory consisting of *cells*, each of which is capable of storing some fixed number of bits. A cell-probe algorithm is a decision tree with one memory cell accessed at each node. The decision tree branches according to the contents of the cell accessed. We only count the number of memory cells accessed in the data structure; all computation is free. This means that no restrictions are imposed on the way data is represented or manipulated, except for the bound on the size of values that each memory cell can hold. Thus, lower bounds obtained in this model apply to all reasonable models of computation and give us insight into why certain problems are hard.

Ajtai, Fredman, and Komlos [1] showed that, if the word length is sufficiently large (i.e. $n^{\Omega(1)}$ bits), then any set of size n can be stored, using a trie, in $O(n)$ words so that predecessor queries can be performed in constant time in the cell probe model. On the other hand, Ajtai [2] proved that, if the word length is sufficiently small (i.e. $O(\log n)$ bits), and only $n^{O(1)}$ words of memory are used to represent any set of n elements, then worst-case constant time for predecessor queries is impossible.

Miltersen [21] observed that a cell-probe algorithm can be viewed as a two-party communication protocol [28] between a Querier who holds the input to a query and a Responder who holds the data structure. In each round of communication, the Querier sends the name of a memory cell to access and the Responder answers with the contents of that memory cell. The communication game model is more general, since the response at a given round can depend on the entire history of the computation so far. In the cell probe model, the response can depend only on which memory cell is being probed, so different probes to the same memory cell must always receive the same response. In fact, for many problems, the cell probe complexity is significantly larger than the communication complexity [20].

Miltersen [21] generalized Ajtai's proof to obtain an $\Omega(\sqrt{\log\log N})$ lower bound on time in this model for the problem of finding predecessors in a static set from a universe of size N. (Independent of our work, Xiao [27] has also improved this lower bound.) In [22], it was shown that for certain universe sizes, Ajtai's proof and its generalization in [21] also gives an $\Omega((\log n)^{1/3})$ lower bound on time. Furthermore, Miltersen [21] provides a general technique for translating time complexity lower bounds (under restrictions on memory size) for static data structure problems into time complexity lower bounds for dynamic data structure problems. In particular, he shows that the time to perform predecessor queries is $\Omega(\sqrt{\log\log N})$ if the time to perform updates is at most $2^{(\log N)^{1-\varepsilon}}$ for some constant $\varepsilon > 0$.

Although the cell probe model is useful for proving the most generally applicable data structure lower bounds, it does not permit one to analyze the particular instructions necessary for these algorithms.

Fich and Miltersen [15] have shown that, for the standard RAM model (which includes addition, multiplication, conditional jumps, and indirect addressing instructions, but not shifts, bitwise Boolean operations, or division), the complexity of performing membership queries in a set of size n stored using at most $N/n^{\Omega(1)}$ words (of unbounded size) requires $\Omega(\log n)$ time. Thus, for this model, binary search is optimal.

AC^0 RAMs allow conditional jumps and indirect addressing, as well as any finite set of AC^0 instructions (such as addition and shifts, but not multiplication or division). In this model, Andersson, Miltersen, Riis, and Thorup [5] proved that the time complexity of membership queries is $\Theta(\sqrt{\log n/\log\log n})$. Their algorithm uses $O(n)$ words (of $\log_2 N$ bits each) and their lower bound holds even if $2^{(\log n)^{O(1)}}$ words are allowed. It is intriguing that the somewhat unusual function describing the time complexity in this case is the same as the one that we derive in a different context.

3 An Optimal Algorithm

This section presents a new algorithm for the static predecessor problem that matches the time complexity lower bounds in section 4. The key contribution is a new technique that is a multi-way variant of binary search. At each step, it either reduces the number of elements in the set under consideration or the number of bits needed to represent individual elements in this set. This technique was motivated by our lower bound work – our first algorithm was for the restricted class of inputs used in our lower bound proof and these inputs provided us with key intuition.

Let S denote a set of $s \leq n$ strings of length $u \leq \log_2 n$ over the alphabet $[0, 2^k - 1]$ and let T denote the trie of branching factor 2^k and depth u representing this set. Each node at depth d of T corresponds to the length d prefix of some element of S, so T contains at most $us + 1$ nodes.

A node v in T is said to be *heavy* if the sub-trie rooted at v has at least $s/n^{1/u}$ leaves. Any ancestor of a heavy node is a heavy node. The root of T is always heavy. For $0 < d < u$, there are at most $n^{1/u}$ heavy nodes at depth d.

Lemma 1: If memory words contain $b \geq [2(u-1)^2 - 1]k$ bits, then there is a data structure of size $O(n^2)$ that can be constructed in time $O(n^2 k)$ and, given a string x of length u over the alphabet $[0, 2^k - 1]$, can determine, in constant time, the longest prefix of x that is a heavy node in T.

Proof Since T has at most $n^{1/u}$ heavy nodes at depth d, for $0 < d < u$, there are hash functions $h_d : [0, 2^k - 1] \to [0, 2^r - 1]$ of the form $h_d(z) = (a_d \times z) \bmod 2^k \operatorname{div} 2^{k-r}$, where $r \leq 2(\log_2 n)/u$ and $a_d \in [0, 2^k - 1]$, such that h_d is one-to-one on the subset

$$\{z \in [0, 2^k - 1] \mid yz \text{ is a heavy node at depth } d \text{ for some heavy node } y \text{ at depth } d-1\}.$$

Each of these $u - 1$ hash functions can be constructed deterministically in time $O(n^{2/u} k)$ [23]. Note that the function

$$y_1 \cdots y_d \to (h_1(y_1), \ldots, h_d(y_d))$$

is one-to-one on the set of heavy nodes of T at depth d.

Given a string $x = x_1 \cdots x_u$ of length u over the alphabet $[0, 2^k - 1]$, in constant time, we can construct a word containing the sequence of $u - 1$ hashed values, $h_1(x_1), \ldots, h_{u-1}(x_{u-1})$, in its $(u-1)r$ least significant bits, using the following algorithm.

Here, a string of length l over the alphabet $[0, 2^k - 1]$ is represented by $b - lk$ zero bits followed by the concatenation of the k-bit binary representations of each of the l letters. The symbol $\langle c \rangle$ denotes an element of $[0, 2^k - 1]$ and $a_i \times x_j$ denotes the string of length 2 over this alphabet formed by multiplying the values a_i and x_j.

- Shift x right k bit positions to obtain the string $x_1 \cdots x_{u-1}$.
- Multiply $x_1 \cdots x_{u-1}$ by the string $(\langle 0 \rangle^{2(u-2)} \langle 1 \rangle)^{u-1}$ to obtain the string $(\langle 0 \rangle^{u-2} x_1 \cdots x_{u-1})^{u-1}$.
- AND with the mask $(\langle 0 \rangle^{2u-3} \langle 2^k - 1 \rangle)^{u-1}$ to obtain the string $x_1 \langle 0 \rangle^{2u-3} x_2 \langle 0 \rangle^{2u-3} \cdots x_{u-2} \langle 0 \rangle^{2u-3} x_{u-1}$.
- Multiply by the string $a_1 \langle 0 \rangle a_2 \langle 0 \rangle \cdots a_{u-2} \langle 0 \rangle a_{u-1}$ to obtain the string $(a_1 \times x_1) \cdots (a_{u-1} \times x_1)(a_1 \times x_2) \cdots \cdots (a_{u-1} \times x_{u-2})(a_1 \times x_{u-1}) \cdots (a_{u-1} \times x_{u-1})$.
- AND with the mask $(\langle 0 \rangle^{2u-3} \langle (2^r - 1) 2^{k-r} \rangle)^{u-1}$, to obtain the string $\langle h_1(x_1) 2^{k-r} \rangle \langle 0 \rangle^{2u-1} \langle h_2(x_2) 2^{k-r} \rangle \langle 0 \rangle^{2u-1} \cdots \cdots \langle h_{u-2}(x_{u-2}) 2^{k-r} \rangle \langle 0 \rangle^{2u-1} \langle h_{u-1}(x_{u-1}) 2^{k-r} \rangle$.
- Multiply by the bit string $(0^{2uk-r-1} 1)^{u-1}$ and AND with the mask $1^{(u-1)r} 0^{(2(u-1)^2-1)k-(u-1)r}$ to obtain the bit string $h_1(x_1) \cdots h_{u-1}(x_{u-1}) 0^{(2(u-1)^2-1)k-(u-1)r}$.
- Finally, shift right $(2(u-1)^2 - 1)k - (u-1)r$ bit positions to obtain the $(u-1)r$ bit string $h_1(x_1) \cdots h_{u-1}(x_{u-1})$.

This sequence of $u - 1$ hashed values can be used to index an array that contains the name of the (unique) heavy node $y_1 \ldots y_d$ in T whose sequence of hashed values is the longest prefix of the given sequence. More formally, consider the $u - 1$ dimensional array V where $V(j_1, j_2, \ldots, j_{u-1}) = y_1 \cdots y_d$ if and only if $y_1 \cdots y_d$ is a heavy node in T, $h_1(y_1) = j_1, \ldots, h_d(y_d) = j_d$, and either $d = u - 1$ or $h_{d+1}(y_{d+1}) \neq j_{d+1}$ for all heavy nodes $y_1 \cdots y_d y_{d+1}$ in T. The array V has length $2^{r(u-1)} < n^2$. Each entry of V is in $[0, 2^{k(u-1)} - 1]$, so $O(n^2)$ words suffice to store V. To find the heavy node at greatest depth which is a prefix of x, it suffices to find the longest common prefix of $y_1 \cdots y_d$ and x, which can be computed in constant time.

The construction of the array V can be done by first explicitly constructing the trie T, which contains at most $O(nu)$ nodes. Depth first search can be used to determine which nodes are heavy. The heavy nodes are inserted into the array V, in order of increasing depth. □

Lemma 2: If memory words contain $b \geq [2(u-1)^2 - 1]k$ bits, $u^u \leq n$, $s \leq n^{a/u}$, and $k = u^c$, for non-negative integers $a \leq u$ and c, then there is a static data structure for representing a set of s integers from the universe $[1, 2^k]$ that supports predecessor queries in $O(a + c)$ time, uses $O(csn^2)$ words, and can be constructed in $O(kcsn^2)$ time.

Proof The proof is by induction on a and c. If $a = 0$ or $c = 0$, then $s \leq 2$ and it suffices to store the elements in a sorted table of size s. Therefore, assume that $a, c > 0$.

Consider any set S of s integers from $[1, 2^k]$ and let T denote the trie of branching factor $2^{k/u}$ and depth u representing it. For every node v in T, let $\min_S(v)$ denote the smallest element of S with prefix v and $\max_S(v)$ denote the largest element of S with prefix v. The data structure consists of the following parts:

- the data structure described in Lemma 1
- for each node v in T that either is heavy or has a heavy parent:
  - $\max_S(v)$, $\min_S(v)$, and $pred(\min_S(v), S)$

- for each heavy node v in T with at least two children:
  - a perfect hash table containing the non-heavy nodes that are children of v
  - the data structure for the set $S'_v = \{v' \in [0, 2^{k/u} - 1] \mid v \cdot v' \text{ is a child of } v\}$ of size at most s in a universe of size $2^{k'}$, where $k' = k/u \leq u^{c-1}$
- for each non-heavy node w in T with a heavy parent and at least two leaves in its subtree:
  - the data structure for the set $S'_w = \{v' \in S \mid w \text{ is a prefix of } v'\}$ of size at most $s/n^{1/u} \leq n^{(a-1)/u}$ in the universe of size 2^k.

To find the predecessor of $x \in [0, 2^k - 1]$ in the set S, first determine the longest prefix v of x that is a heavy node in T, as described in Lemma 1. Suppose that v is at depth d.

If v has at most one child, then
$$pred(x,S) = \begin{cases} pred(\min_S(v), S) & \text{if } x \leq \min_S(v) \\ \min_S(v) & \text{if } x > \min_S(v). \end{cases}$$

Now consider the case when v has at least two children. Determine whether some child of v is a prefix of x, using the hash table containing all of v's non-heavy children. By definition of v, if there is such a child, then it is not heavy.

If no child of v is a prefix of x, then
$$pred(x,S) = \begin{cases} pred(\min_S(v), S) & \text{if } x \leq \min_S(v) \\ \max_S(v \cdot pred(x_{d+1}, S'_v)) & \text{if } x > \min_S(v). \end{cases}$$
Find $pred(x_{d+1}, S'_v)$ using the data structure for the set S'_v. Note that $v \cdot pred(x_{d+1}, S'_v)$ is either a non-heavy child of the heavy node v or is itself heavy, so the largest element of S in the subtree rooted at this node is stored explicitly.

Now consider the case when some child w of v is a prefix of x. If w has exactly one leaf in its subtree, then
$$pred(x,S) = \begin{cases} pred(\min_S(w), S) & \text{if } x \leq \min_S(w) \\ \min_S(w) & \text{if } x > \min_S(w). \end{cases}$$
Otherwise,
$$pred(x,S) = \begin{cases} pred(\min_S(w), S) & \text{if } x \leq \min_S(w) \\ pred(x, S'_w) & \text{if } x > \min_S(w). \end{cases}$$
Find $pred(x, S'_w)$ using the data structure for the set S'_w.

Next, we analyze the storage requirements of this data structure. First consider the storage required at the top level of the recursion. There are most $un^{1/u} \leq n^{2/u}$ heavy nodes at that top level and at most $s \leq n$ non-heavy nodes that are children of heavy nodes associated with the top level, since the trie has at most s leaves. Except for the the data structure from Lemma 1 and the hash table containing v's non-heavy children, which require $O(n^2)$ words, there is only a constant amount of storage for each of these nodes. Thus the total storage for the top level of the data structure is $O(n^2)$. To bound the storage for the whole data structure we simply multiply this cost by the number of sets S'_v and S'_w that appear at all levels of recursion.

Observe that any set S'_w (or S'_v) can be identified with a partial sub-trie of the binary trie representing S that consists of a path from some node r in this trie to the node v and then the full sub-trie for S below w to some fixed depth so that the total depth of the partial sub-trie rooted at r is a power of u. Also note that, given the node w and this power of u, the set S'_w is uniquely determined. This immediately gives a bound of csk on the number of such sets since the binary trie for S has at most sk internal nodes and there are only c choices of the power of u. However, consider two nodes v and w with w below v on a path segment in the trie for S (a part where no branching occurs) and fix the choice of a power of u at which they are both candidates to have sub-data structures associated with them. If v and w are in different sub-data structures at this level of granularity then v has only one leaf in its sub-data structure, so it does not have an associated set S'_v. If v and w are in the same sub-data structure then they correspond to the same set of leaves so at most one of them will have an associated sub-data structure. (If v and w are both heavy then v will only have one child and if w is non-heavy then so is its parent because its parent is a descendant of v.) Since the trie for S has s leaves, it has at most $2s$ such segments; so, in total, there are at most $2s$ such nodes. This gives a bound of $2cs$ on the total number of sets that occur.

The construction cost analysis follows similarly. □

Theorem 3: There is a static data structure for representing n integers from the universe $[1,N]$ that takes $O\left(\min\left\{\log\log N/\log\log\log N, \sqrt{\log n/\log\log n}\right\}\right)$ time for predecessor queries and can be constructed in $O(n^4)$ time and space.

Proof If $n < 2^{(\log_2\log_2 N)^2/(\log_2\log_2\log_2 N)}$, then $(\log_2 n)/\log_2 b \leq (\log_2 n)/\log_2\log_2 N < \sqrt{2\log_2 n/\log_2\log_2 n} \leq \sqrt{2}\log_2\log_2 N/\log_2\log_2\log_2 N$. In this case, we use Andersson's static data structure [4] that supports predecessor queries in $O(1 + \log n/\log b)$ time and can be constructed in $O(n^4)$ time and space.

Now assume that $n \geq 2^{2(\log_2\log_2 N)^2/(\log_2\log_2\log_2 N)}$. Then $\sqrt{\log_2 n/\log_2\log_2 n} \geq \log_2\log_2 N/(\sqrt{2}\log_2\log_2\log_2 N)$. Let $u = 2(\log_2\log_2 N)/(\log_2\log_2\log_2 N)$, so $\sqrt{n} \geq u^u \geq \log_2 N$. The data structure in this case has two parts. The first part consists of the top $1 + 2\lceil\log_2 u\rceil$ levels of Willard's x-fast trie [26]. This reduces the problem of finding the predecessor in a set of size n from a universe of size N to finding the predecessor in a set of size at most n from a universe of size 2^k, where $k = (\log_2 N)/2^{1+2\lceil\log_2 u\rceil} \leq (\log_2 N)/2u^2 < u^{u-2}$ and $[2(u-1)^2 - 1]k < \log_2 N \leq b$. The data structure of Lemma 2 is used for each resulting subproblem. The total number of elements in all of these sets is at most $2^{1+2\lceil\log_2 u\rceil}n = O(u^2 n)$ and each set has size at most n.

By Lemma 2, the data structures for the subproblems use total space $O(u^3n^3) = O(n^4)$ and can be constructed in time $O(kn^3u^3) = O(n^4)$. The truncated x-fast trie uses space $O(n\log N) = O(n^2)$ and can be constructed in time $O(n^2(\log N)^2) = O(n^4)$. □

Combined with exponential search trees [4, 6], we get a linear size, dynamic data structure.

Corollary 4: The dynamic predecessor problem for a set of up to n integers from the universe $[1,N]$ can be solved on a RAM with $O(n)$ words of $\log_2 N$ bits, in time $O\left(\min\left\{\log\log n \log\log N/\log\log\log N, \sqrt{\log n/\log\log n}\right\}\right)$.

4 Lower Bounds for the Predecessor Problem

Let the *static* (N,n) *predecessor problem* be the static predecessor problem restricted to sets $S \subseteq [1,N]$ of size n. For technical reasons, we insist that, for such a data structure the answer be determined solely by the sequence of memory locations accessed and their contents. (This extra condition can be established using $O(n)$ additional memory cells and at most one additional time step: Simply add a perfect hash table for the set S and, when the value of the predecessor is determined, access the appropriate location in the hash table.)

Our results are more general than for the predecessor problem. In fact, our lower bounds also hold for the rank and even simpler problems, including prefix-parity (the lower order bit of the rank). In the full version of the paper, we prove, more generally, lower bounds for the prefix problem for any strongly-indecisive regular language [21]. Here, we give the argument for the predecessor problem since the details are somewhat easier in this case. Our main technical theorem is the following:

Theorem 5: There is a constant $c > 0$ such that if $(ckbt)^{4t} \leq n \leq (ckbt)^{8t}$ and $N \geq n^{(ckt)^t}$ then there is no t round cell-probe communication protocol for the static (N,n) predecessor problem using n^k memory cells of $b \geq 8$ bits each.

Before discussing the proof of this theorem, we derive its two main corollaries.

Theorem 6: Any cell-probe data structure for the static predecessor problem that stores any set S from a universe of size N using $|S|^{O(1)}$ memory cells of $2^{(\log N)^{1-\Omega(1)}}$ bits requires query time $\Omega(\log\log N/\log\log\log N)$ in the worst case.

Proof More precisely, we show that for any positive integer k, and any positive constant ε, there exists a function $n(N) \leq N$ such that any cell-probe data structure for the static $(N,n(N))$ predecessor problem using $(n(N))^k$ memory cells of $2^{(\log N)^{1-\varepsilon}}$ bits requires time $\Omega(\log\log N/\log\log\log N)$ per query.

Fix $k,\varepsilon > 0$ and choose the largest integer t such that $(\log N)^{\varepsilon} \geq (ckt)^{4t}$ where c is the constant from Theorem 5. Clearly $c' \log\log N/\log\log\log N \geq t \geq c'' \log\log N/\log\log\log N$ for some constants $c',c'' > 0$ depending only on k and ε (and the constant c). Let $b = 2^{(\log N)^{1-\varepsilon}}$ and set $n = (cbkt)^{4t}$. Then $(ckt)^t \leq (\log N)^{\varepsilon/4}$ and

$$n^{(ckt)^t} \quad \leq \quad \left[(\log N)^{\varepsilon} 2^{4c'(\log N)^{1-\varepsilon}\frac{\log\log N}{\log\log\log N}}\right]^{(\log N)^{\varepsilon/4}} \quad < N$$

for N sufficiently large. Therefore, by Theorem 5, any cell-probe data structure for the static (N,n) predecessor problem requires time at least $t+1$ using n^k memory cells of b bits each. $\square$

Theorem 7: Any cell-probe data structure for the static predecessor problem that stores any set S from a universe of size N using $|S|^{O(1)}$ memory cells of $(\log N)^{O(1)}$ bits requires query time $\Omega(\sqrt{\log|S|/\log\log|S|})$ in the worst case.

Proof More precisely, we show that for any positive integers k,k', there is a function $N(n)$ such that any cell-probe data structure for the static $(N(n),n)$ predecessor problem using n^k memory cells of $(\log N(n))^{k'}$ bits requires time $\Omega(\sqrt{\log n/\log\log n})$ per query.

Fix k and k' and consider the largest integer t for which $n \geq (ckt)^{4k't^2+4t}(\log n)^{4k't}$ where c is the constant from Theorem 5. Then $t \geq c'\sqrt{\log n/\log\log n}$ where $c' > 0$ is a constant depending only on k and k' (and the constant c). Set $N = n^{(ckt)^t}$ and $b = (\log N)^{k'} = [(ckt)^t \log n]^{k'}$. Clearly $n \geq (cbkt)^{4t}$ and, by the choice of t, one can also check that that $n \leq (cbkt)^{8t}$. Thus, by Theorem 5, any cell-probe data structure for the static (N,n) predecessor problem requires time at least $t+1$ using n^k memory cells of b bits each. $\square$

We now proceed to prove Theorem 5. Let $Z(N,n)$ denote the set of all subsets of $[1,N]$ of size n. We prove lower bounds on the complexity of the static (N,n) predecessor problem using an adversary argument. As discussed in the introduction, Miltersen [21] observed that one can phrase a static data structure algorithm in the cell-probe model in terms of a communication protocol between between two players: the Querier, who holds the input to a query, and the Responder, who holds the data structure. Each probe that the Querier makes to the data structure, a cell name, consists of $\log_2 m$ bits of communication, where m is the number of memory cells, and each response by the Responder, the contents of that named cell, consists of exactly b bits of communication. The number of rounds of alternation of the players is the time t of the cell-probe communication protocol. The technical condition that the sequence of locations and their values determine the answer is equivalent to the condition that the bits communicated alone determine the answer.

The lower bound, in the style of [19], works 'top down', maintaining, for each player, a relatively large set of inputs on which the communication is fixed. Unlike [19], we actually have non-uniform distributions on the Responder's inputs, so our notion of 'large' is with respect to these distributions. The distributions get simpler as the rounds of the communication proceed.

If $\mathcal{Z}$ is a probability distribution on a set Z and $B \subseteq Z$, we define $\mu_{\mathcal{Z}}(B) = \Pr_{\mathcal{Z}}[B]$. Let $\mathcal{U}(N,n)$ be the distribution which chooses a set $S \subseteq [1,N]$ of size n uniformly at random. The following is the base case of our lower bound.

Lemma 8: Let $N \geq n > 0$, $\alpha > 0$, and $\beta \geq e^{-\alpha n}$. Consider any set of positions $A \subseteq [1,N]$, with $|A| \geq \alpha N + 1$, and any collection of subsets $B \subseteq Z(N,n)$, with $\mu_{\mathcal{U}(N,n)}(B) \geq \beta$. Then there exist integers $a, a' \in A$ and a set $S \in B$ such that $pred(a,S) \neq pred(a',S)$.

Proof Observe that the only way that $pred(a,S)$ is the same for all $a \in A$ is if there is no element $j \in S$ with $\min(A) \leq j < \max(A)$. Since this region contains at least αn elements of $[1,N]$, this probability is at most $(1-\alpha)^n < e^{-\alpha n} \leq \beta$ since $\alpha > 0$. □

We define a sequence of distributions on $Z(N,n)$ and use it to demonstrate that no cell-probe communication protocol using n^k memory cells of b bits can solve the static (N,n) predecessor problem in t rounds. Given integers b, k, t, N, and n we will define two sequences of integers N_i and n_i for $i = 0, \ldots, t-1$ with $N_0 = N$, and $n_0 = n$. The general idea of the lower bound argument is to find, after each round, a portion of the Querier's and Responder's inputs on which the cell-probe communication protocol has made little progress. After i rounds, the possible values of the Querier's input will lie in an interval of length N_i and, within this interval, the Responder's input S will have at most n_i elements. Thus, the Responder's input can be viewed as an element of $Z(N_i, n_i)$.

More precisely, let b, k, t, N, and n be positive integers and define

- $\alpha = n^{-1/(4t)}$
- $u = 8kt$
- $r = 16bu/\alpha$
- $f = 8ru/\alpha = 128bu^2/\alpha^2$
- $N_0 = N$; $n_0 = n$ and
- for $i = 0, \ldots, t-1$, define $N_{i+1} = (N_i/f)^{1/u}$ and $n_{i+1} = n_i/(ru)$.

We say that the tuple of parameters (b,k,t,N,n) satisfies the *integrality condition* if $1/\alpha$ is an integer greater than 1 and, for every integer $i \in [0,t]$, N_i and n_i are integers and $N_i \geq n_i$.

If n is the $4t$-th power of an integer larger than 1, then $1/\alpha$ is an integer greater than 1 and f and r are also integers. Since $f \geq ru$ and $u \geq 1$, the condition $N_t \geq n_t$ is sufficient to imply that $N_i \geq n_i$ for $i \in [0,t]$. Furthermore, if N_t and n_t are both integers, then $n_i = (ru)^{t-i} n_t$ and $N_i = f^{(u^{t-i}-1)/(u-1)} N_t^{u^{t-i}}$ are integers for $i \in [0,t]$. In particular, the integrality condition will hold for (b,k,t,N,n) if n is the $4t$-th power of an integer larger than 1 and there are integers $N_t \geq n_t$ such that $n = (ru)^t n_t$ and $N = f^{(u^t-1)/(u-1)} N_t^{u^t}$.

Suppose that the integrality condition holds for (b,k,t,N,n). For each i, $i = t, \ldots, 0$, we define a probability distribution $\mathcal{Z}_i$ on $Z(N_i, n_i)$ inductively, beginning with $\mathcal{Z}_t$, which is the distribution $\mathcal{U}(N_t, n_t)$. For every $i < t$, each set in $Z(N_i, n_i)$ can be thought of as marking the leaves of a tree T_i with depth $u+1$, having fan-out f at the root and a complete N_{i+1}-ary tree of depth u at each child of the root. We choose a random element of $\mathcal{Z}_i$ as the set of marked leaves of the tree T_i, which we choose using the distribution $\mathcal{Z}_{i+1}$ as follows: First, choose r nodes uniformly from among all the children of the root. For each successively deeper level, excluding the leaves, choose r nodes uniformly among the nodes at that level that are not descendants of nodes chosen at higher levels. (Notice that, since the root of T_i has $f \geq ru$ children, it is always possible to choose enough nodes with this property at each level.) Independently, for each of these ru nodes, v, choose a set $S_v \in Z(N_{i+1}, n_{i+1})$ according to $\mathcal{Z}_{i+1}$. A leaf below v is marked if it is the rightmost descendant of the j-th child of v for some $j \in S_v$.

Lemma 9: Suppose (b,k,t,N,n) satisfies the integrality condition and $b \geq 3$. Let $A \subseteq [1, N_i]$ with $|A| \geq \alpha N_i$, and $B \subseteq Z(N_i, n_i)$ with $\mu_{\mathcal{Z}_i}(B) \geq \beta = 2^{-b-1}$. Suppose there is a $t-i$ round cell-probe communication protocol, using $m \leq n^k$ memory cells of b bits, that correctly computes $pred(j,S)$ for all $j \in A$ and $S \in B$. Then there exist $A' \subseteq [1, N_{i+1}]$ with $|A'| \geq \alpha N_{i+1}$, $B' \subseteq Z(N_{i+1}, n_{i+1})$ with $\mu_{\mathcal{Z}_{i+1}}(B') \geq \beta$ and a $t-i-1$ round cell-probe communication protocol, using m cells of b bits, that correctly computes $pred(j',S')$ for all $j' \in A'$ and $S' \in B'$.

Proof We first sketch the argument. This argument isolates a node v in T_i with the property that we can fix one round of communication in the original $t-i$ round cell-probe communication protocol to obtain a new $t-i-1$ round communication protocol that still works well in the subtree rooted at v.

To find this node v, we first find a level of the tree T_i from which to choose v. A node is a good candidate for v if there is some fixed communication c by the Querier such that many of its children (at least an α fraction) have descendants corresponding to inputs consistent with communication c. By a lemma of Ajtai [2], we show that, no matter how the communication of the Querier is determined, there is a level with many nodes that are good candidates for v. In fact, there are so many that there will be $\Omega(b)$ such nodes among the r nodes at that level involved in the definition of $\mathcal{Z}_i$.

By the bound on β, for at least one of these $\Omega(b)$ nodes, call it v, the μ-measure of the possible values for the portion of the set S that lies below v is at least 1/2. Now we fix v, fix the associated communication c of the Querier, and its most popular (with respect to the portion of the set S lying below v) response c' by the Responder. Since v was one of the nodes chosen by distribution $\mathcal{Z}_i$, only the rightmost descendants of its children are potentially marked, so the answer to the predecessor problem does not depend on which of those descendants below a given child of v is the input. Thus we identify the portion of the set S below v with an element in $Z(N_{i+1}, n_{i+1})$, and the input below v with the position of its ancestor among the children of v. It is not hard to see that we can remap the answers in a simple deterministic way using this reduction. We now follow through on this sketch, after stating a couple of preliminaries.

Preliminaries

The following form of the Chernoff-Hoeffding bound follows easily from the presentation in [10].

Proposition 10: Fix $H \subseteq U$ with $|H| \geq \rho|U|$ and let $S \subseteq U$ with $|S| = s$ be chosen uniformly at random. Then $\Pr[|H \cap S| \leq \rho s/4] \leq (\sqrt{2}/e^{3/4})^{\rho s} < 2^{-\rho s/2}$.

The next result is a small modification and rephrasing of a combinatorial lemma that formed the basis of Ajtai's lower bound argument in [2].

Suppose we have a tree T of depth d such that all nodes on the same level have the same number of children. For any node $v \in T$, let leaves(v) denote the set of leaves of T that are descendants of v and, for v not the root of T, let parent(v) denote the parent of v. Let $A(1),\ldots,A(m)$ be disjoint sets of leaves of T and let $A = \bigcup_{c=1}^{m} A(c)$. The leaves in $A(c)$ are said to have *colour c*. A non-leaf node v has *colour c* if leaves(v) contains a node in $A(c)$. For $c = 1,\ldots,m$, let $A'(c) = \{v \mid \text{leaves}(v) \cap A(c) \neq \phi\}$ denote the set of nodes with colour c. Note that the sets $A'(1),\ldots A'(m)$ are not necessarily disjoint, since a non-leaf node may have more than one colour.

A non-leaf node v is δ-*dense* (where $0 \leq \delta \leq 1$) if there is a colour c such that at least a fraction δ of v's children have colour c.

Lemma 11: (Ajtai[2]) Let T be a tree of depth $d \geq 2$ such that all nodes on the same level of T have the same number of children. Suppose that at least a fraction α of all the leaves in T are coloured (each with one of m colours). Then there exists a level ℓ, $1 \leq \ell \leq d-1$, such that the fraction of nodes on level ℓ of T that are δ-dense is at least $\dfrac{\alpha - m\delta^{d-1}}{d-1}$.

Finding the node v

We examine the behaviour of the Querier during the first round of the original cell-probe communication protocol to find a set of candidates for the node v. For each value of $j \in A$, the Querier sends one of m messages indicating which of the m memory cells it wishes to to probe. Colour the jth leaf of T_i with this message.

Since $|A| \geq \alpha N_i$, it follows from Ajtai's Lemma that there exists a level ℓ such that $1 \leq \ell \leq u$ and the fraction of α-dense nodes in level ℓ of T_i is at least $(\alpha - m\alpha^u)/u$. By the integrality condition, $\alpha \leq 1/2$. Furthermore, $u > 6$ and $m \leq n^k = \alpha^{-4kt} = \alpha^{-\frac{u}{2}}$. Therefore

$$(\alpha - m\alpha^u)/u \geq \alpha(1 - \alpha^{\frac{u}{2}-1})/u > 3\alpha/(4u).$$

We now argue that there is a sufficiently large set of candidates for v among the α-dense nodes at level ℓ and a way of marking the leaves of T_i that are not descendants of these candidates so that the probability of choosing a set in B remains sufficiently large.

Note that in the construction of $\mathcal{Z}_i$ from $\mathcal{Z}_{i+1}$, the r nodes chosen on level ℓ are not uniformly chosen from among all nodes on level ℓ. The constraint that these nodes not be descendants of any of the $r(\ell-1)$ nodes chosen at higher levels skews this distribution somewhat and necessitates a slightly more complicated argument.

Consider the different possible choices for the $r(\ell-1)$ nodes at levels $1,\ldots,\ell-1$ of T_i in the construction of $\mathcal{Z}_i$ from $\mathcal{Z}_{i+1}$. By simple averaging, there is some such choice with $\mu_{\mathcal{Z}'_i}(B) \geq \beta$, where $\mathcal{Z}'_i$ is the probability distribution obtained from $\mathcal{Z}_i$ conditioned on the fact that this particular choice occurred. Fix this choice.

Let R be the random variable denoting the set of r nodes chosen at level ℓ. Since the choice of nodes at higher levels has been fixed, there are certain nodes at level ℓ that are no longer eligible to be in R. Specifically, each of the r nodes chosen at level $h < \ell$ eliminates its $N_{i+1}^{\ell-h}$ descendants at level ℓ from consideration. In total, there are

$$\sum_{h=1}^{\ell-1} r \cdot N_{i+1}^{\ell-h} < 2r \cdot N_{i+1}^{\ell-1}$$

nodes eliminated from consideration at level ℓ. There are $fN_{i+1}^{\ell-1}$ nodes at level ℓ, so the fraction of nodes at level ℓ that are eliminated is less than $2r/f = \alpha/(4u)$. Thus, of the nodes at level ℓ that have not been eliminated, the subset D of nodes which are α-dense constitutes more than a fraction $3\alpha/(4u) - \alpha/(4u) = \alpha/(2u)$.

We may view the random choice R of the r nodes at level ℓ as being obtained by choosing r nodes randomly, without replacement, from the set of nodes at level ℓ that were not eliminated. Applying Proposition 10 with $\rho = \alpha/(2u)$ and $|R| = r$,

$$Pr[|D \cap R| \leq r\alpha/(8u)] < 2^{-r\alpha/(4u)} = 2^{-4b}.$$

Since $b \geq 1$, this probability is smaller than $\beta/2$. Let E be the event that at least $r\alpha/(8u) = 2b$ of the r elements of R are α-dense. Then $\mu_{\mathcal{Z}''_i}(B) \geq \beta - \beta/2 = \beta/2$, where $\mathcal{Z}''_i$ is the probability distribution obtained from $\mathcal{Z}'_i$ conditioned on the fact that event E occurred.

Assume that event E has occurred. Then $|D \cap R| \geq 2b$. Let V be the random variable denoting the first $2b$ nodes chosen for R that are also in D. By simple averaging, there is some choice for V with $\mu_{\mathcal{Z}'''_i}(B) \geq \beta/2$, where $\mathcal{Z}'''_i$ is the probability distribution obtained from $\mathcal{Z}''_i$ conditioned on the fact that this particular choice for V occurred. Fix some such choice.

Finally, consider the different possible choices W for the portion of the set S that marks leaves which are not descendants of nodes in V. By simple averaging, there is some choice for W with $\mu_{\mathcal{Z}^*_i}(B) \geq \beta/2$, where $\mathcal{Z}^*_i$ is the probability distribution obtained from $\mathcal{Z}'''_i$ conditioned on the fact that this particular choice for W occurred. Fix some such choice.

By construction, the distribution $\mathcal{Z}^*_i$ is isomorphic to a cross-product of $2b$ independent distributions, $\mathcal{Z}_{i+1}$, one for each of the nodes in V. Specifically, for each $v \in V$, the selection of the set of children of v that have marked descendants is made according to $\mathcal{Z}_{i+1}$ and only the rightmost descendants of such children are marked. For $v \in V$ and S chosen from $\mathcal{Z}_i^*$, let $\pi_v(S)$ denote the subset of $[1, N_{i+1}]$ indicating which children of v have marked descendants. In other words, $k \in \pi_v(S)$ if and only if the k'th child of v has a marked descendant. Let $B_v = \{\pi_v(S) \mid S \in B \text{ is consistent with } W\}$. Then

$$\beta/2 \le \mu_{\mathcal{Z}_i^*}(B) \le \prod_{v \in V} \mu_{\mathcal{Z}_{i+1}}(B_v).$$

Hence, there is some $v \in V$ such that

$$\mu_{\mathcal{Z}_{i+1}}(B_v) \ge (\mu_{\mathcal{Z}_i^*}(B))^{1/|V|} \ge (\beta/2)^{1/2b} = 2^{-(b+2)/(2b)} \ge 1/2,$$

since $b \ge 2$. Choose that node v.

Fixing a round of communication for each player

Since v is α-dense, there is some message c that the Querier may send in the first round such that $|A'|/N_{i+1} \ge \alpha$, where

$$A' = \{j' \in [1, N_{i+1}] \mid \text{the } j'\text{-th child of } v \text{ is coloured } c\};$$

i.e., there is some input j corresponding to a descendant of the j'-th child of v on which the Querier sends message c in the first round. We fix the message sent by the Querier in the first round to be c.

Fix a function ι that maps sets $S' \in B_v$ into sets $S \in B$ such that $\pi_v(\iota(S')) = S'$. In other words, $\iota(S')$ witnesses the fact that $S' \in B_v$.

There are only 2^b different messages the Responder can now send. Therefore, there is some fixed message c' for which

$$\mu_{\mathcal{Z}_{i+1}}(B') \ge 2^{-b-1} = \beta,$$

where B' is the collection of sets $S' \in B_v$ such that, in round one, given the input $\iota(S')$ and the query c, the Responder sends c'. We fix the message sent by the Responder in the first round to be c'.

Constructing the $t-i-1$ round protocol

Consider the following new $t-i-1$ round protocol: Given inputs $j' \in A'$ and $S' \in B'$, the Querier and the Responder simulate the last $t-i-1$ rounds of the original $t-i$ round protocol, using inputs $j \in A$ and $S = \iota(S') \in B$, respectively, where j is the index of some leaf in T_i with colour c that is a descendant of the j'-th child of node v. Note that it doesn't matter which leaf of colour c in the subtree rooted at the j'-th child of v is chosen. This is because every leaf in this subtree, except the rightmost leaf, is not marked so $pred(j,S)$ is the same no matter which leaf in the subtree is indexed by j.

It follows from the definitions of A' and B' that, for inputs j and S, the original protocol will send the fixed messages c and c' during round one. By construction, the new protocol computes $pred(j,S)$. If this value is not a descendant of v then this is interpreted as 0; otherwise, if it lies in the subtree below the i'-th child of v the answer is interpreted as i'. □

We now combine Lemma 8 and Lemma 9 to prove Theorem 5:

Proof of Theorem 5 Let $c = 64$. Suppose that there is a t round cell-probe communication protocol for the static (N,n) predecessor problem using $m \le n^k$ memory cells of b bits. Let $n' = (ckbt)^{4t} \le n$ and $k' = 2k$. Then $n' \le n$ and $m \le (n')^{k'}$.

Let $u = 8k't$, $\alpha = (n')^{-1/(4t)}$, $r = 16bu/\alpha$, $f = 128bu^2/\alpha^2$, $N_0 = N$, $n'_0 = n'$, and let $N_{i+1} = (N_i/f)^{1/u}$ and $n'_{i+1} = n'_i/(ru)$ for $i = 0, \ldots, t-1$.

Note that $f = 128b \times 64(k')^2t^2(n')^{1/2t} \le n$. One can now easily check that since $N \ge n^{(64kt)^t}$, $N_t \ge n \ge n_t$. As noted above, $N_t \ge N/(f^{u^t})$.

Therefore (b,k',t,N,n') satisfies the integrality condition and the algorithm works correctly for all inputs $j \in A = [1,N]$ and $S \in B = Z(N,n')$. Since $b \ge 2$, Lemma 9 can be applied t times to obtain $A' \subseteq [1,N_t]$ with $|A'| \ge \alpha N_t$, $B' \subseteq Z(N_t, n'_t)$ with $\mu_{\mathcal{Z}_t}(B') \ge \beta = 2^{-b-1}$, and a 0 round cell-probe communication protocol such that the protocol correctly computes $pred(j,S)$ for all $j \in A'$ and $S \in B'$. This implies that $pred(j,S)$ is the same for all $j \in A'$ and $S \in B'$.

Since $\alpha = (n')^{-1/(4t)}$ and $t \ge 1$, $\alpha^{1+t} \ge (n')^{-1/2}$. Also, $n' \ge (32bk't)^{4t}$ so

$$\begin{aligned} \alpha n'_t r &= \frac{\alpha n'}{(ru)^t} = \frac{n'\alpha^{1+t}}{(16bu^2)^t} \\ &\ge \frac{\sqrt{n'}}{[16b(8k't)^2]^t} = \frac{\sqrt{n'}b^t}{(32bk't)^{2t}} \ge b^t \ge b \end{aligned}$$

since $t \ge 1$. Therefore, $e^{-\alpha n'_t} \le e^{-b} < 2^{-b-1}$ since $b \ge 3$, and so by Lemma 8, there exist integers $a, a' \in A'$ and a set $S \in B'$ such that $pred(a,S) \ne pred(a',S)$. This is a contradiction. □

One can translate the arguments of this section to the dynamic case, using a translation argument given by Miltersen [21]. This requires work since the bound applies even without the polynomial restriction on the size of the data structure. The basic idea of Miltersen's translation is to observe that dynamic algorithms that have small cost per query and do not run for very long can access only a small number of memory cells from a moderate size set of potential memory cells. Using static dictionary techniques from [17], one can obtain an efficient solution to the static problem by beginning with the empty set and inserting elements one by one, recording the changes made to the memory in the dictionary. One then obtains:

Theorem 12: There is no cell-probe data structure for the dynamic predecessor problem on $[1,N]$ for sets of size at most n using $(\log N)^{O(1)}$ bits per memory cell, $n^{O(1)}$ worst-case time for insertions, and $o(\sqrt{\log n/\log\log n})$ worst-case query time.

We can obtain a lower bound solely in terms of the universe size with an even larger allowed word size. We do not state this version explicity but we state its extension to amortized costs provided the memory is not too large: If words are b bits long then a bound of $2^{O(b)}$ on the number of memory cells is reasonable; it is the number of different cells that can be accessed when performing indirect addressing.

Theorem 13: Any cell-probe data structure using $2^{O(b)}$ memory cells of $b = 2^{(\log N)^{1-\Omega(1)}}$ bits to solve the dynamic predecessor problem on $[1, N]$ using $2^{(\log N)^{1-\Omega(1)}}$ amortized time per update, requires $\Omega(\log\log N / \log\log\log N)$ worst-case time for queries.

Acknowledgements

We are grateful to Peter Bro Miltersen and Mikkel Thorup for helpful discussions.

References

[1] M. Ajtai, M. Fredman, and J. Komlós. Hash functions for priority queues. *Information and Control*, 63:217–225, 1984.

[2] Miklós Ajtai. A lower bound for finding predecessors in Yao's cell probe model. *Combinatorica*, 8:235–247, 1988.

[3] A. Andersson. Sublogarithmic seaching without multiplications. In *36th IEEE Annual Symposium on Foundations of Computer Science*, pages 655–665, Milwaukee, WI, 1995.

[4] A. Andersson. Faster deterministic sorting and seaching in linear space. In *37th Annual IEEE Symposium on Foundations of Computer Science*, pages 135–141, Burlington, VT, 1996.

[5] A. Andersson, P. B. Miltersen, S. Riis, and M. Thorup. Static dictionaries on AC^0 RAMs: query time $\Theta(\sqrt{\log n / \log\log n})$ is necessary and sufficient. In *37th Annual Symposium on Foundations of Computer Science*, pages 441–450, Burlington, VT, October 1996. IEEE.

[6] A. Andersson and M. Thorup. Exponential search trees for faster deterministic searching, sorting and priority queues in linear space. Manuscript.

[7] A. Brodnik, P.B. Miltersen, and I. Munro. Trans-dichotomous algorithms without multiplications–some upper and lower bounds. In *Proceedings of the 5th Workshop on Algorithms and Data Structures*, LNCS volume 1272, pages 426–439, Halifax, NS, Canada, 1997. Springer-Verlag.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[9] A. Chakrabarti, B. Chazelle, B. Gum, and A. Lvov. A good neighbor is hard to find. In *Proceedings of the Thirty First Annual ACM Symposium on Theory of Computing*, Atlanta, GA, May 1999.

[10] V. Chvátal. Probabilistic methods in graph theory. *Annals of Operations Research*, 1:171–182, 1984.

[11] M. Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS volume 1046, pages 569–580, Grenoble, France, February 1996. Springer-Verlag.

[12] M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger. Polynomial hash functions are reliable. In *Automata, Languages, and Programming: 19th International Colloquium*, LNCS volume 623, pages 235–246. Springer-Verlag, July 1992.

[13] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R Tarjan. Dynamic pertect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, 1994.

[14] M. Dietzfelbinger and F. Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Automata, Languages, and Programming: 17th International Colloquium*, LNCS volume 443, pages 6–17, Warwick University, England, July 1990. Springer-Verlag.

[15] F. Fich and P. B. Miltersen. Tables should be sorted (on random access machines). In *Proceedings of the 4th Workshop on Algorithms and Data Structures*, LNCS volume 995, pages 163–174. Springer-Verlag, 1995.

[16] M. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31:538–544, 1984.

[17] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 345–354, Seattle, WA, May 1989.

[18] M. Fredman and D. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993.

[19] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 539–550, Chicago, IL, May 1988.

[20] P. B. Miltersen. The bit probe complexity measure revisited. In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS volume 665, pages 662–671, Wurzburg, Germany, February 1993. Springer-Verlag.

[21] P. B. Miltersen. Lower bounds for Union-Split-Find related problems on random access machines. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 625–634, Montréal, Québec, Canada, May 1994.

[22] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

[23] R. Raman. Priority queues: Small, monotone, and trans-dichotomous. In *Proceedings of the 4th European Symposium on Algorithms*, LNCS volume 1136, pages 121–137. Springer-Verlag, 1996.

[24] P. Van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.

[25] P. Van Emde Boas, R Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

[26] D. E. Willard. Log-logarithmic worst case range queries are possible in space $\Theta(n)$. *Information Processing Letters*, 17:81–84, 1983.

[27] Bing Xiao. *New bounds in cell probe model*. PhD thesis, University of California, San Diego, 1992.

[28] A. C. Yao. Some complexity questions related to distributive computing. In *Conference Record of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 209–213, Atlanta, GA, April-May 1979.

[29] A. C. Yao. Should tables be sorted? *Journal of the ACM*, 28:615–628, July 1981.