Time-Space Trade-Offs for Predecessor Search*

Mihai Pătraşcu
mip@mit.edu

Mikkel Thorup
mthorup@research.att.com

March 10, 2006

Abstract

We develop a new technique for proving cell-probe lower bounds for static data structures. Previous lower bounds used a reduction to communication games, which was known not to be tight by counting arguments. We give the first lower bound for an explicit problem which breaks this communication complexity barrier. In addition, our bounds give the first separation between polynomial and near linear space. Such a separation is inherently impossible by communication complexity.

Using our lower bound technique and new upper bound constructions, we obtain tight bounds for searching predecessors among a static set of integers. Given a set Y of n integers of ℓ bits each, the goal is to efficiently find $\text{PREDECESSOR}(x) = \max \ \{y \in Y \mid y \leq x\}$. For this purpose, we represent Y on a RAM with word length w using S words of space. Defining $a = \lg \frac{S}{n} + \lg w$, we show that the optimal search time is, up to constant factors:

$$\min \begin{cases} \log_w n \\ \lg \frac{\ell - \lg n}{a} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{\ell}{a}\right)} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\lg \frac{\ell}{a} \Big/ \lg \frac{\lg n}{a}\right)} \end{cases}$$

In external memory ($w > \ell$), it follows that the optimal strategy is to use either standard B-trees, or a RAM algorithm ignoring the larger block size. In the important case of $w = \ell = \gamma \lg n$, for $\gamma > 1$ (i.e. polynomial universes), and near linear space (such as $S = n \cdot \lg^{O(1)} n$), the optimal search time is $\Theta(\lg \ell)$. Thus, our lower bound implies the surprising conclusion that van Emde Boas' classic data structure from [FOCS'75] is optimal in this case. Note that for space $n^{1+\varepsilon}$, a running time of $O(\lg \ell / \lg \lg \ell)$ was given by Beame and Fich [STOC'99].

*An extended abstract of this paper appears in the Proceedings of the 38th ACM Symposium on Theory of Computing (STOC'06).

1 Introduction

In this paper we provide tight trade-offs between query time and space of representation for static predecessor search. This is one of the most basic data structures, and the trade-off gives the first separation between linear and polynomial space for *any* data structure problem.

1.1 The Complexity-Theoretic View

Yao's cell-probe model [21] is typically the model of choice for proving lower bounds on data structures. The model assumes the memory is organized in w-bit cells (alternatively called words). In the case of static data structures, one first constructs a representation of the input in a table with a bounded number of cells S (the space complexity). Then, a query can be answered by probing certain cells. The time complexity T is defined to be the number of cell probes. The model allows free nonuniform computation for both constructing the input representation, and for the query algorithm. Thus, the model is stronger than the word RAM or its variants, which are used for upper bounds, implementable in a programming language like C. In keeping with the standard assumptions on the upper bound side, we only consider $w = \Omega(\lg n)$.

Typically, lower bounds in this model are proved by considering a two-party communication game. Assume Bob holds the data structure's input, while Alice holds the query. By simulating the cell-probe solution, one can obtain a protocol with T rounds, in which Alice sends $\lg S$ bits and Bob replies with w bits per round. Thus, a lower bound on the number of rounds translates into a cell-probe lower bound.

Intuitively, we do not expect this relation between cell-probe and communication complexity to be tight. In the communication model, Bob can remember past communication, and answer new queries based on this. Needless to say, if Bob is just a table of cells, he cannot remember anything, and his responses must be a function of Alice's last message (i.e. the address of the cell probe). By counting arguments, it can be shown [12] that the cell-probe complexity can be much higher than the communication complexity, for natural ranges of parameters. However, a separation for an explicit problem has only been obtained in a very restricted setting. Gál and Miltersen [11] showed such a separation when the space complexity is very close to minimum: given an input of n cells, the space used by the data structure is $n + o(n)$.

Besides the reduction to communication complexity, and the approach of [11] for very small space, there are no known techniques applicable to static cell-probe complexity with cells of $\Omega(\lg n)$ bits. In particular, we note that the large body of work initiated by Fredman and Saks [9] only applies to *dynamic* problems, such as maintaining partial sums or connectivity. In the case of static complexity, there are a few other approaches developed specifically for the bit-probe model ($w = 1$); see [14].

In conclusion, known lower bound techniques for cell-probe complexity cannot surpass the communication barrier. However, one could still hope that communication bounds are interesting enough for natural data structure problems. Unfortunately, this is often not the case. Notice that polynomial differences in S only translate into constant factors in Alice's message size. In the communication game model, this can only change constant factors in the number of rounds, since Alice can break a longer message into a few separate messages. Unfortunately, this means that communication complexity cannot be used to separate, say, polynomial and linear space. For many natural data-structure problems, the most interesting behavior occurs close to linear space, so it is not surprising that our understanding of static data-structure problems is rather limited.

In this work, we develop a new lower-bound technique, the *cell-probe elimination lemma*, targeted specifically at the cell-probe model. Using this lemma, we obtain a separation between space $n^{1+o(1)}$ and space $n^{1+\varepsilon}$ for any $\varepsilon > 0$. This also represents a separation between communication complexity and cell-probe complexity with space $n^{1+o(1)}$. Our lower bounds hold for predecessor search, one of the most natural and well-studied problems.

Our lower bound result has a strong direct sum flavor, which is interesting in its own right. Essentially, we show that for problems with a certain structure, a data structure solving k independent subproblems with space $k \cdot \sigma$ cannot do better than k data structures solving each problem with space σ.

1.2 The Data-Structural View

Using our lower bound technique and new upper bound constructions, we obtain tight bounds for predecessor search. The problem is to represent an ordered set Y, such that for any query x we can find efficiently $\texttt{PREDECESSOR}(x) = \max\ \{y \in Y \mid y \leq x\}$. This is one of the most fundamental and well-studied problems in data structures. For a comprehensive list of references, we refer to [4]; here, we only describe briefly the best known bounds.

1.2.1 The Upper-Bound Story

We focus on the static case, where Y is given in advance for preprocessing. For example, we can sort Y, and later find the predecessor of x by binary search using $O(\lg n)$ comparisons, where $n = |Y|$.

On computers, we are particularly interested in integer keys. Thereby we also handle, say, floating point numbers whose ordering is preserved if they are cast as integers. We can then use all the instructions on integers available in a standard programming language such as C, and we are no longer limited by the $\Omega(\lg n)$ comparison based lower bound for searching. A strong motivation for considering integer keys is that integer predecessor search is asymptotically equivalent to the IP look-up problem for forwarding packets on the Internet [7]. This problem is extremely relevant from a practical perspective. The fastest deployed software solutions use non-comparison-based RAM tricks [6].

More formally, we will represent Y on a unit-cost word RAM with a given word length w. We assume each integers in Y has ℓ bits, and that $\lg n \leq \ell \leq w$. On the RAM, the most natural assumption is $\ell = w$. The case $w > \ell$ models the external memory model with $B = \lfloor \frac{w}{\ell} \lfloor$ keys per page. In this case, the well-known (comparison-based) B-trees achieve a search time of $O(\log_B n)$. For the rest of the discussion, assume $w = \ell$.

Using the classic data structure of van Emde Boas [19] from 1975, we can represent our integers so that predecessors can be searched in $O(\lg \ell)$ time. The space is linear if we use hashing [20].

In the 1990, Fredman and Willard [10] introduced fusion trees, which requires linear space and can answer queries in $O(\log_\ell n)$ time. Combining with van Emde Boas' data structure, they got a search time of $O(\min\{\frac{\lg n}{\lg \ell}, \lg \ell\})$, which is always $O(\sqrt{\lg n})$.

In 1999, Beame and Fich [4] found an improvement to van Emde Boas' data structure bringing the search time down to $O(\frac{\lg \ell}{\lg \lg \ell})$. Combined with fusion trees, this gave them a bound of $O(\min\{\frac{\lg n}{\lg \ell}, \frac{\lg \ell}{\lg \lg \ell}\})$, which is always $O(\sqrt{\frac{\lg n}{\lg \lg n}})$. However, the new data structure of Beame and Fich uses quadratic space, and they asked if the space could be improved to linear or near-linear.

As a partially affirmative answer to this question, we show that their $O(\frac{\lg \ell}{\lg \lg \ell})$ search time can be obtained with space $n^{1+1/\exp(\lg^{1-\varepsilon}\ell)}$ for any $\varepsilon > 0$. However, we also show, as our main result, that with closer to linear space, such as $n \lg^{O(1)} n$, one cannot in general improve the old van Emde Boas bound of $O(\lg \ell)$.

1.2.2 The Lower-Bound Story

Ajtai [1] was the first to prove a superconstant lower bound for our problem. His results, with a correction by Miltersen [13], can be interpreted as saying that there exists n as a function of ℓ such that the time complexity for polynomial space is $\Omega(\sqrt{\lg \ell})$, and likewise there exists ℓ a function of n making the time complexity $\Omega(\sqrt[3]{\lg n})$.

Miltersen [13] revisited Ajtai's work, showing that the lower bound holds in the communication game model, and for a simpler colored predecessor problem. In this problem, the elements of Y have an associated color (say, red or blue), and the query asks only for the color of the predecessor in Y. This distinction is important, as one can reduce other problems to this simpler problem, such as existential range queries in two dimensions [15] or prefix problems in a certain class of monoids [13]. Like previous lower bound proofs, ours also holds for the colored problem, making the lower bounds applicable to these problems.

Miltersen, Nisan, Safra and Wigderson [15] once again revisited Ajtai's proof, extending it to randomized algorithms. More importantly, they captured the essence of the proof in an independent *round elimination lemma*, which forms a general tool for proving communication lower bounds. Our cell-probe elimination lemma is inspired, at a high level, by this result.

Beame and Fich [4] improved the lower bounds to $\Omega(\frac{\lg \ell}{\lg\lg \ell})$ and $\Omega(\sqrt{\frac{\lg n}{\lg\lg n}})$ respectively. Sen and Venkatesh [16] later gave an improved round elimination lemma, which can reprove the lower bounds of Beame and Fich, but also for randomized algorithms. Analyzing the time-space trade-offs obtained by these proofs, one obtains $\Omega(\frac{\lg n}{\lg w}, \frac{\lg \ell}{\lg\lg S})$, where S is the space bound, and possibly $w > \ell$.

1.3 The Optimal Trade-Offs

Define $\lg x = \lceil \log_2(x+2) \rceil$, so that $\lg x \geq 1$ even if $x \in [0,1]$. Assuming space S, and defining $a = \lg \frac{S}{n} + \lg w$, we show that the optimal search time is, up to constant factors:

$$\min \begin{cases} \log_w n \\ \lg \frac{\ell - \lg n}{a} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{\ell}{a}\right)} \\ \dfrac{\lg \frac{\ell}{a}}{\lg\left(\lg \frac{\ell}{a} \Big/ \lg \frac{\lg n}{a}\right)} \end{cases} \tag{1}$$

The upper bounds are achieved by a deterministic query algorithm on a RAM. The data structure can be constructed in expected time $O(S)$ by a randomized algorithm, starting from a sorted list of integers. The lower bounds hold for deterministic query algorithms answering the colored predecessor problem in the cell-probe model. When $S \geq n^{1+\varepsilon}$ for some constant $\varepsilon > 0$, the lower bounds also hold in the stronger communication game model, even allowing randomization with two-sided error.

1.3.1 External Memory and Branch One

To understand the first branch of the trade-off, first consider the typical case on a RAM, when a word fits exactly one integer, i.e. $w = \ell$. In this case, the bound is $\log_\ell n$, which describes the performance of fusion trees [10].

To understand the case $w > \ell$, consider the external memory model with B words per page. This model has as a nonuniform counterpart the cell-probe model with cells of size $w = B\ell$. Observe that only the first branch of our trade-off depends on w. This branch is $\log_w n = \frac{\lg n}{\lg B + \lg \ell} = \Theta(\min\{\log_\ell n, \log_B n\})$. The first term describes the performance of fusion trees on a RAM with ℓ-bit words, as noted above. The second term matches the performance of the B-tree, the fundamental data structure in external memory.

Thus, we show that it is always optimal to either use a standard B-tree, or the best RAM algorithm which completely ignores the benefits of external memory. The RAM algorithm uses ℓ-bit words, and ignores the grouping of words into pages; this algorithm is the best of fusion trees and the algorithms from branches 2–4 of the trade-off. Thus, the standard comparison-based B-tree is the optimal use of external memory, even in a strong model of computation.

1.3.2 Polynomial Universes: Branch Two

For the rest of the discussion, assume the first branch (B-trees and fusion trees) does not give the minimum. Some of the most interesting consequences of our results can be seen in the very important special case when integers come from a polynomial universe, i.e. $\ell = O(\lg n)$. In this case, the optimal complexity is $\Theta(\lg \frac{\ell - \lg n}{a})$, as given by the second branch of the trade-off.

On the upper bound side, this is achieved by a simple elaboration of van Emde Boas' data structure. This data structure gives a way to reduce the key length from ℓ to $\frac{\ell}{2}$ in constant time, which immediately implies an upper bound of $O(\lg \ell)$. To improve that, first note that when $\ell \leq a$, we can stop the recursion and use complete tabulation to find the result. This means only $O(\lg \frac{\ell}{a})$ steps are needed. Another trivial idea, useful for near-linear universes, is to start with a table lookup based on the first $\lg n$ bits of the key, which requires linear space. Then, continue to apply van Emde Boas for keys of $w - \lg n$ bits inside each subproblem, giving a complexity of $O(\lg \frac{w - \lg n}{a})$.

Quite surprisingly, our lower bound shows that van Emde Boas' classic data structure, with these trivial tweaks, is optimal. In particular, when the space is not too far from linear (at most $n \cdot 2^{\lg^{1-\varepsilon} n}$) and $\ell \geq (1+\varepsilon)\lg n$, the standard van Emde Boas bound of $\Theta(\lg \ell)$ is optimal. It was often conjectured that this bound could be improved.

Note that with space $n^{1+\varepsilon}$, the optimal complexity for polynomial universes is constant. However, with space $n^{1+o(1)}$, the bound is $\omega(1)$, showing the claimed complexity-theoretic separations.

1.3.3 The Last Two Branches

The last two branches are relevant for superpolynomial universes, i.e. $\ell = \omega(\lg n)$. Comparing the two branches, we see the third one is better than the last one (up to constants) when $a = \Omega(\lg n)$. On the other hand, the last branch can be asymptotically better when $a = o(\lg n)$. This bound has the advantage that in the logarithm in the denominator, the factor $\frac{a}{\lg n}$, which is subconstant for $a = o(\lg n)$, is replaced by $1/\lg \frac{a}{\lg n}$.

The third branch is obtained by a careful application of the techniques of Beame and Fich [4], which can improve over van Emde Boas, but need large space. The last branch is also based on these techniques, combined with novel approaches tailored for small space.

1.3.4 Dynamic Updates

Lower bounds for near-linear space easily translate into interesting lower bounds for dynamic problems. If inserting an element takes time t_u, we can obtain a static data structure using space $O(n \cdot t_u)$ by simply simulating n inserts and storing the modified cells in a hash table. This transformation works even if updates are randomized, but, as before, we require that queries be deterministic. This model of randomized updates and deterministic queries is standard for hashing-based data structures. By the discussion above, as long as updates are reasonably fast, one cannot in general improve on the $O(\lg \ell)$ query time. It should be noted that van Emde Boas data structure can handle updates in the same time as queries, so this classic data structure is also optimal in the typical dynamic case, when one is concerned with the slowest operation.

1.4 Contributions

We now discuss our contributions in establishing the tight results of (1). Our main result is proving the tight lower bounds for $a = o(\lg n)$ (in particular, branches two and four of the trade-off). As mentioned already, previous techniques were helpless, since none could even differentiate $a = 2$ from $a = \lg n$.

Interestingly, we also show improved lower bounds for the case $a = \Omega(\lg n)$, in the classic communication framework. These improvements are relevant to the third branch of the trade-off. Assuming for simplicity that $a \le w^{1-\varepsilon}$, our bound is $\min\{\frac{\lg n}{\lg w}, \frac{\lg w}{\lg\lg w + \lg(a/\lg n)}\}$, whereas the best previous lower bound was $\min\{\frac{\lg n}{\lg w}, \frac{\lg w}{\lg a}\}$. Our improved bound is based on a simple, yet interesting twist: instead of using the round elimination lemma alone, we show how to combine it with the *message compression lemma* of Chakrabarti and Regev [5]. Message compression is a refinement of round elimination, introduced by [5] to prove a lower bound for the approximate nearest neighbor problem. Sen and Venkatesh [16] asked whether message compression is really needed, or one could just use standard round elimination. Our result sheds an interesting light on this issue, as it shows message compression is even useful for classic predecessor lower bounds.

On the upper bound side, we only need to show the last two branches of the trade-off. As mentioned already, we use techniques of Beame and Fich [4]. The third bound was anticipated[1] by the second author in the concluding remarks of [18]. The last branch of (1), tailored specifically for small space, is based on novel ideas.

1.5 Direct-Sum Interpretations

A very strong consequence of our proofs is the idea that sharing between subproblems does not help for predecessor search. Formally, the best cell-probe complexity achievable by a data structure representing k independent subproblems (with the same parameters) in space $k \cdot \sigma$ is asymptotically equal to the best complexity achievable by a data structure for one subproblem, which uses space σ. The simplicity and strength of this statement make it interesting from both the data-structural and complexity-theoretic perspectives.

At a high level, it is precisely this sort of direct-sum property that enables us to beat communication complexity. Say we have k independent subproblems, and total space S. While in the communication game Alice sends $\lg S$ bits per round, our results intuitively state that $\lg \frac{S}{k}$ bits are

[1] As a remark in [18, Section 7.5], it is stated that "it *appears* that we can get the following results…", followed by bounds equivalent to the third branch of (1).

sufficient. Then, by carefully controlling the increase in k and the decrease in key length (the query size), we can prevent Alice from communicating her entire input over a superconstant number of rounds.

A nice illustration of the strength of our result are the tight bounds for near linear universes, i.e. $\ell = \lg n + \delta$, with $\delta = o(\lg n)$. On the upper bound side, the algorithm can just start by a table lookup based on the first $\lg n$ bits of the key, which requires linear space. Then, it continues to apply van Emde Boas for δ-bit keys inside each subproblem, which gives a complexity of $O(\lg \frac{\delta}{a})$. Obtaining a lower bound is just as easy, given our techniques. We first consider $n/2^\delta$ independent subproblems, where each has 2^δ integers of 2δ bits each. Then, we prefix the integers in each subproblem by the number of the subproblem (taking $\lg n - \delta$ bits), and prefix the query with a random subproblem number. Because the universe of each subproblem ($2^{2\delta}$) is quadratically bigger than the number of keys, we can apply the usual proof showing the optimality of van Emde Boas' bound for polynomial universes. Thus, the complexity is $\Omega(\lg \frac{\delta}{a})$.

2 Lower Bounds for Small Space

2.1 The Cell-Probe Elimination Lemma

An abstract decision data structure problem is defined by a function $f : D \times Q \to \{0, 1\}$. An input from D is given at preprocessing time, and the data structure must store a representation of it in some bounded space. An input from Q is given at query time, and the function of the two inputs must be computed through cell probes. We restrict the preprocessing and query algorithms to be deterministic. In general, we consider a problem in conjunction with a distribution $\mathcal{D}$ over $D \times Q$. Note that the distribution need not (and, in our case, will not) be a product distribution. We care about the probability the query algorithm is successful under the distribution $\mathcal{D}$ (for a notion of success to be defined shortly).

As mentioned before, we work in the cell-probe model, and let w be the number of bits in a cell. We assume the query's input consists of at most w bits, and that the space bound is at most 2^w. For the sake of an inductive argument, we extend the cell-probe model by allowing the data structure to publish some bits at preprocessing time. These are bits depending on the data structure's input, which the query algorithm can inspect at no charge. Closely related to this concept is our model for a query being successful. We allow the query algorithm not to return the correct answer, but only in the following very limited way. After inspecting the query and the published bits, the algorithm can declare that it cannot answer the query (we say it *rejects* the query). Otherwise, the algorithm can make cell probes, and at the end it must answer the query correctly. Thus, we require an a priori admission of any "error". In contrast to models of silent error, it actually makes sense to talk about tiny (close to zero) probabilities of success, even for problems with boolean output.

For an arbitrary problem f and an integer $k \leq 2^w$, we define a direct-sum problem $\bigoplus^k f : D^k \times ([k] \times Q) \to \{0, 1\}$ as follows. The data structure receives a vector of inputs $(d^1, \dots, d^k)$. The representation depends arbitrarily on all of these inputs. The query is the index of a subproblem $i \in [k]$, and an element $q \in Q$. The output of $\bigoplus^k f$ is $f(q, d^i)$. We also define a distribution $\bigoplus^k \mathcal{D}$ for $\bigoplus^k f$, given a distribution $\mathcal{D}$ for f. Each d^i is chosen independently at random from the marginal distribution on D induced by $\mathcal{D}$. The subproblem i is chosen uniformly from $[k]$, and q is chosen from the distribution on Q conditioned on d^i.

Given an arbitrary problem f and an integer $h \leq w$, we can define another problem $f^{(h)}$ as follows. The query is a vector $(q_1, \dots, q_h)$. The data structure receives a regular input $d \in D$, and integer $r \in [h]$ and the prefix of the query $q_1, \dots, q_{r-1}$. The output of $f^{(h)}$ is $f(d, q_r)$. Note that we have shared information between the data structure and the querier (i.e. the prefix of the query), so $f^{(h)}$ is a partial function on the domain $D \times \bigcup_{i=0}^{t-1} Q^i \times Q$. Now we define an input distribution $\mathcal{D}^{(h)}$ for $f^{(h)}$, given an input distribution $\mathcal{D}$ for f. The value r is chosen uniformly at random. Each query coordinate q_i is chosen independently at random from the marginal distribution on Q induced by $\mathcal{D}$. Now d is chosen from the distribution on D, conditioned on q_r.

We give the $f^{(h)}$ operator precedence over the direct sum operator, i.e. $\bigoplus^k f^{(h)}$ means $\bigoplus^k \left[f^{(h)}\right]$. Using this notation, we are ready to state our central cell-probe elimination lemma:

Lemma 1. *There exists a universal constant C, such that for any problem f, distribution $\mathcal{D}$, and positive integers h and k, the following holds. Assume there exists a solution to $\bigoplus^k f^{(h)}$ with success probability δ over $\bigoplus^k \mathcal{D}^{(h)}$, which uses at most $k\sigma$ words of space, $\frac{1}{C}(\frac{\delta}{h})^3 k$ published bits and T cell probes. Then, there exists a solution to $\bigoplus^k f$ with success probability $\frac{\delta}{4h}$ over $\bigoplus^k \mathcal{D}$, which uses the same space, $k\sqrt[h]{\sigma} \cdot Cw^2$ published bits and $T-1$ cell probes.*

2.2 Setup for the Predecessor Problem

Let $P(n, \ell)$ be the colored predecessor problem on n integers of ℓ bits each. Remember that this is the decision version of predecessor search, where elements are colored red or blue, and a query just returns the color of the predecessor. We first show how to identify the structure of $P(n,\ell)^{(h)}$ inside $P(n, h\ell)$, making it possible to apply our cell-probe elimination lemma.

Lemma 2. *For any integers $n, \ell, h \geq 1$ and distribution $\mathcal{D}$ for $P(n,\ell)$, there exists a distribution $\mathcal{D}^{*(h)}$ for $P(n, h\ell)$ such that the following holds. Given a solution to $\bigoplus^k P(n, h\ell)$ with success probability δ over $\bigoplus^k \mathcal{D}^{*(h)}$, one can obtain a solution to $\bigoplus^k P(n,\ell)^{(h)}$ with success probability δ over $\bigoplus^k \mathcal{D}^{(h)}$, which has the same complexity in terms of space, published bits, and cell probes.*

Proof. We give a reduction from $P(n,\ell)^{(h)}$ to $P(n, h\ell)$, which naturally defines the distribution $\mathcal{D}^{*(h)}$ in terms of $\mathcal{D}^{(h)}$. A query for $P(n,\ell)^{(h)}$ consists of $x_1, \dots, x_h \in \{0,1\}^\ell$. Concatenating these, we obtain a query for $P(n, h\ell)$. In the case of $P(n,\ell)^{(h)}$, the data structure receives $i \in [h]$, the query prefix $x_1, \dots, x_{i-1}$ and a set Y of ℓ-bit integers. We prepend the query prefix to all integers in Y, and append zeros up to $h\ell$ bits. Then, finding the predecessor of x_i in Y is equivalent to finding the predecessor of the concatenation of $x_1, \dots, x_h$ in this new set. $\square$

Observe that to apply the cell-probe elimination lemma, the number of published bits must be just a fraction of k, but applying the lemma increases the published bits significantly. We want to repeatedly eliminate cell probes, so we need to amplify the number of subproblems each time, making the new number of published bits insignificant compared to the new k.

Lemma 3. *For any integers $t, \ell, n \geq 1$ and distribution $\mathcal{D}$ for $P(n,\ell)$, there exists a distribution $\mathcal{D}^{*t}$ for $P(n \cdot t, \ell + \lg t)$ such that the following holds. Given a solution to $\bigoplus^k P(n \cdot t, \ell + \lg t)$ with success probability δ over $\bigoplus^k \mathcal{D}^{*t}$, one can construct a solution to $\bigoplus^{kt} P(n,\ell)$ with success probability δ over $\bigoplus^{kt} \mathcal{D}$, which has the same complexity in terms of space, published bits, and cell probes.*

Proof. We first describe the distribution $\mathcal{D}^{*t}$. We draw $Y_1, \dots, Y_t$ independently from $\mathcal{D}$, where Y_i is a set of integers, representing the data structures input. Prefix all numbers in Y_j by j using $\lg t$

bits, and take the union of all these sets to form the data structure's input for $P(nt, \ell + \lg t)$. To obtain the query, pick $j \in \{0, \ldots, t-1\}$ uniformly at random, pick the query from $\mathcal{D}$ conditioned on Y_j, and prefix this query by j. Now note that $\bigoplus^{kt} \mathcal{D}$ and $\bigoplus^{k} \mathcal{D}^{*t}$ are really the same distribution, except that the lower $\lg t$ bits of the problems index for $\bigoplus^{kt} \mathcal{D}$ are interpreted as a prefix in $\bigoplus^{k} \mathcal{D}^{*t}$. Thus, obtaining the new solution is simply a syntactic transformation. □

Our goal is to eliminate all cell probes, and then reach a contradiction. For this, we need the following impossibility result for a solution making zero cell probes:

Lemma 4. *For any $n \geq 1$ and $\ell \geq \log_2(n+1)$, there exists a distribution $\mathcal{D}$ for $P(n, \ell)$ such that the following holds. For all $(\forall) 0 < \delta \leq 1$ and $k \geq 1$, there does not exist a solution to $\bigoplus^k P(n, \ell)$ with success probability δ over $\bigoplus^k \mathcal{D}$, which uses no cell probes and less than δk published bits.*

Proof. The distribution $\mathcal{D}$ is quite simple: the integers in the set are always 0 up to $n-1$, and the query is n. All that matters is the color of $n-1$, which is chosen uniformly at random among red and blue. Note that for $\bigoplus^k P(n, \ell)$ there are only k possible queries, i.e. only the index of the subproblem matters.

Let $\mathbf{p}$ be the random variable denoting the published bits. Since there are no cell probes, the answers to the queries are a function of $\mathbf{p}$ alone. Let $\delta(p)$ be the fraction of subproblems that the query algorithm doesn't reject when seeing the published bits p. In our model, the answer must be correct for all these subproblems. Then, $\Pr[\mathbf{p} = p] \leq 2^{-\delta(p)k}$, as only inputs which agree with the $\delta(p)k$ answers of the algorithm can lead to these published bits. Now observe that $\delta = \mathbf{E}_p[\delta(p)] \leq \mathbf{E}_p\left[\frac{1}{k}\log_2 \frac{1}{\Pr[\mathbf{p}=p]}\right] = \frac{1}{k}H(\mathbf{p})$, where $H(\cdot)$ denotes binary entropy. Since the entropy of the published bits is bounded by their number (less than δk), we have a contradiction. □

2.3 Showing Predecessor Lower Bounds

Our proof starts assuming that we for any possible distribution have a solution to $P(n, \ell)$ which uses $n \cdot 2^a$ space, no published bits, and successfully answers all queries in T probes, where T is small. We will then try to apply T *rounds* of the cell-probe elimination from Lemma 1 and 2 followed by the problem amplification from Lemma 3. After T rounds, we will be left with a non-trivial problem but no cell probes, and then we will reach a contradiction with Lemma 4. Below, we first run this strategy ignoring details about the distribution, but analyzing the parameters for each round. Later in Lemma 5, we will present a formal inductive proof using these parameters in reverse order, deriving difficult distributions for more and more cell probes.

We denote the problem parameters after i rounds by a subscript i. We have the key length ℓ_i and the number of subproblems k_i. The total number of keys remains n, so the have n/k_i keys in each subproblem. Thus, the problem we deal with in round $i+1$ is $\bigoplus^{k_i} P(\frac{n}{k_i}, \ell_i)$, and we will have some target success probability δ_i. The number of cells per subproblem is $\sigma_i = \frac{n}{k_i}2^a$. We start the first round with $\ell_0 = \ell, \delta_0 = 1, k_0 = 1$ and $\sigma_0 = n \cdot 2^a$.

For the cell probe elimination in Lemma 1 and 2, our proof will use the same value of $h \geq 2$ in all rounds. Then $\delta_{i+1} \geq \frac{\delta_i}{4h}$, so $\delta_i \geq (4h)^{-i}$. To analyze the evolution of ℓ_i and k_i, we let t_i be the factor by which we increase the number of subproblems in round i when applying the problem amplification from Lemma 3. We now have $k_{i+1} = t_i \cdot k_i$ and $\ell_{i+1} = \frac{\ell_i}{h} - \lg t_i$.

When we start the first round, we have no published bits, but when we apply Lemma 1 in round $i+1$, it leaves us with up to $k_i \sqrt[h]{\sigma_i} \cdot Cw^2$ published bits for round $i+2$. We have to choose t_i large enough to guarantee that this number of published bits is small enough compared to the number of subproblems in round $i+2$. To apply Lemma 1 in round $i+2$, the number of published bits must be at most $\frac{1}{C}(\frac{\delta_{i+1}}{h})^3 k_{i+1} = \frac{\delta_i^3 t_i}{64Ch^6} k_i$. Hence we must set $t_i \geq \sqrt[h]{\sigma_i} \cdot 64C^2 w^2 h^6 (\frac{1}{\delta_i})^3$. Assume for now that $T = O(\lg \ell)$. Using $h \leq \ell$, and $\delta_i \geq (4h)^{-T} \geq 2^{O(\lg^2 \ell)}$, we conclude it is enough to set:

$$(\forall) i: \qquad t_i \geq \sqrt[h]{\frac{n}{k_i}} \cdot 2^{a/h} \cdot w^2 \cdot 2^{\Theta(\lg^2 \ell)} \tag{2}$$

Now we discuss the conclusion reached at the end of the T rounds. We intend to apply Lemma 4 to deduce that the algorithm after T stages cannot make zero cell probes, implying that the original algorithm had to make more than T probes. Above we made sure that we after T rounds had $\frac{1}{C}(\frac{\delta_T}{h})^3 k_T < \delta_T k_T$ published bits, which are few enough compared to the number k_T of subproblems. The remaining conditions of Lemma 4 are:

$$\ell_T \geq 1 \qquad \text{and} \qquad \frac{n}{k_T} \geq 1 \tag{3}$$

Since $\ell_{i+1} \leq \frac{\ell_i}{2}$, this condition entails $T = O(\lg \ell)$, as assumed earlier.

Lemma 5. *With the above parameters satisfying (2) and (3), for $i = 0, \ldots, T$, there is a distribution $\mathcal{D}_i$ for $P(\frac{n}{k_i}, \ell_i)$ so that no solution for $\bigoplus^{k_i} P(\frac{n}{k_i}, \ell_i)$ can have success probability δ_i over $\bigoplus^{k_i} \mathcal{D}_i$ using $n \cdot 2^a$ space, $\frac{1}{C}(\frac{\delta_i}{h})^3 k_i$ published bits, and $T - i$ cell probes.*

Proof. The proof is by induction over $T - i$. A distribution that defies a good solution as in the lemma is called difficult. In the base case $i = T$, the space doesn't matter, and we get the difficult distribution directly from (3) and Lemma 4. Inductively, we use a difficult distribution $\mathcal{D}_i$ to construct a difficult distribution $\mathcal{D}_{i-1}$.

Recall that $k_i = k_{i-1} t_{i-1}$. Given our difficult distribution $\mathcal{D}_i$, we use the problem amplification in Lemma 3, to construct a distribution $\mathcal{D}_i^{*t_{i-1}}$ for $P(\frac{n}{k_i} \cdot t_{i-1}, \ell_i + \lg t_{i-1}) = P(\frac{n}{k_{i-1}}, \ell_i + \lg t_{i-1})$ so that no solution for $\bigoplus^{k_{i-1}} P(\frac{n}{k_{i-1}}, \ell_i + \lg t_{i-1})$ can have success probability δ_i over $\bigoplus^{k_{i-1}} \mathcal{D}_i^{*t_{i-1}}$ using $n \cdot 2^a$ space, $\frac{1}{C}(\frac{\delta_i}{h})^3 k_i$ published bits, and $T - i$ cell probes.

Recall that (2) implies $k_{i-1} \sqrt[h]{\sigma_{i-1}} \cdot C w^2 \leq \frac{1}{C}(\frac{\delta_i}{h})^3 k_i$, hence that $k_{i-1} \sqrt[h]{\sigma_{i-1}}$ is less than the number of bits allowed published for our difficult distribution $\mathcal{D}_i^{*t_{i-1}}$. Also, recall that $\sigma_j k_j = n \cdot 2^a$ for all j. We can therefore use the cell probe elimination in Lemma 1, to construct a distribution $\left(\mathcal{D}_i^{*t_{i-1}}\right)^{(h)}$ for $P(\frac{n}{k_{i-1}}, \ell_i + \lg t_{i-1})^{(h)}$ so that no solution for $\bigoplus^{k_{i-1}} P(\frac{n}{k_{i-1}}, \ell_i + \lg t_{i-1})^{(h)}$ can have success probability $\delta_{i-1} \geq h\delta_i$ over $\bigoplus^{k_{i-1}} \left(\mathcal{D}_i^{*t_{i-1}}\right)^{(h)}$ using $n \cdot 2^a$ space, $\frac{1}{C}(\frac{\delta_{i-1}}{h})^3 k_{i-1}$ published bits, and $T - i + 1$ cell probes. Finally, using Lemma 2, we use $\left(\mathcal{D}_i^{*t_{i-1}}\right)^{(h)}$ to construct the desired difficult distribution $\mathcal{D}_{i-1}$ for $P(\frac{n}{k_{i-1}}, h(\ell_i + \lg t_{i-1})) = P(\frac{n}{k_{i-1}}, \ell_{i-1})$. $\square$

The predecessor lower bound then follows by applying Lemma 5 with $i = 0$ and the initial parameters $\ell_0 = \ell, \delta_0 = 1, k_0 = 1$. We conclude that there is a difficult distribution $\mathcal{D}_0$ for $P(n, \ell)$ with no solution getting success probability 1 using $n \cdot 2^a$ space, 0 published bits, and T cell probes.

2.4 Calculating the Trade-Offs

In this section, we show how to choose h and t_i in order to maximize the lower bound T, under the conditions of (2) and (3). First, we show a simple bound on a recursion that shows up repeatedly in our analysis:

Lemma 6. *Consider the recursion $x_{i+1} \geq \alpha x_i - \gamma$, for $\gamma \geq 1$. As long as $i \leq \log_{1/\alpha}(\frac{x_0}{1+\gamma/(1-\alpha)})$, we have $x_i \geq 1$.*

Proof. Expanding the recursion, we have $x_i \geq x_0\alpha^i - \gamma(\alpha^{i-1} + \cdots + 1) = x_0\alpha^i - \gamma\frac{1-\alpha^i}{1-\alpha}$. For $x_i \geq 1$, we must have $x_0\alpha^i \geq 1 + \gamma\frac{1-\alpha^i}{1-\alpha}$, which is true if $x_0\alpha^i \geq 1 + \frac{\gamma}{1-\alpha}$. This gives $i \leq \log_{1/\alpha}(\frac{x_0}{1+\gamma/(1-\alpha)})$. □

We now argue that the bound for low space that we are trying to prove can only be better than the communication complexity lower bound when $\lg \ell = O((\lg\lg n)^2)$. This is relevant because our cell-probe elimination lemma is less than perfect in its technical details, and cannot always achieve the optimal bound. Fortunately, however, it does imply an optimal bound when ℓ is not too large, and in the remaining cases an optimal lower bound follows from communication complexity.

Remember that for space $O(n^2)$, communication complexity implies an asymptotic lower bound of $\min\{\frac{\lg n}{\lg w}, \frac{\lg(\ell/\lg n)}{\lg\lg(\ell/\lg n)}\}$. If $\lg \ell = \Omega((\lg\lg n)^2)$, this is $\Theta(\min\{\frac{\lg n}{\lg w}, \frac{\lg \ell}{\lg\lg \ell}\})$. For $a \leq \lg n$, we are trying to prove an asymptotic lower bound of $\min\{\frac{\lg n}{\lg w}, \frac{\lg(\ell/a)}{\lg(\lg\frac{\ell}{a}/\lg\frac{\lg n}{a})}\}$. If $\lg \ell = \Omega((\lg\lg n)^2)$, this becomes $\Theta(\min\{\frac{\lg n}{\lg w}, \frac{\lg \ell}{\lg\lg \ell}\})$, which is identical to the communication bound.

Polynomial Universes. Assume that $\ell \geq 3\lg n$. We first show a lower bound of $\Omega(\lg\frac{\lg n}{a})$, which matches van Emde Boas on polynomial universes. For this, it suffices to set $h = 2$ and $t_i = (\frac{n}{k_i})^{3/4}$. Then, $\frac{n}{k_{i+1}} = (\frac{n}{k_i})^{1/4}$, so $\lg\frac{n}{k_i} = 4^{-i}\lg n$ and $\lg t_i = \frac{3}{4}4^{-i}\lg n$. By our recursion for ℓ_i, we have $\ell_{i+1} = \frac{\ell_i}{2} - \frac{3}{4}4^{-i}\lg n$. Given $\ell_0 = \ell \geq 3\lg n$, it can be seen by induction that $\ell_i \geq 3\cdot 4^{-i}\lg n$. Indeed, $\ell_{i+1} \geq 3\cdot 4^{-i}\cdot\frac{1}{2}\lg n - \frac{3}{4}4^{-i}\lg n \geq 3\cdot 4^{-(i+1)}\lg n$. By the above, (3) is satisfied for $T \leq \Theta(\lg\lg n)$. Finally, note that condition (2) is equivalent to:

$$\begin{aligned} \lg t_i \;\; \geq \;\; \frac{1}{h}\lg\frac{n}{k_i} + \frac{a}{h} + \Theta(\lg w + \lg^2 \ell) \quad &\Leftrightarrow \quad \frac{3}{4}4^{-i}\lg n \;\geq\; \frac{1}{2}4^{-i}\lg n + \frac{a}{2} + \Theta(\lg w + \lg^2 \ell) \\ \Leftrightarrow \;\; T \;\leq\; \Theta\left(\lg\;\min\left\{\frac{\lg n}{a}, \frac{\lg n}{\lg^2 w}\right\}\right) &= \Theta\left(\min\left\{\lg\frac{\lg n}{a}, \lg\lg n\right\}\right) = \Theta\left(\lg\frac{\lg n}{a}\right) \end{aligned}$$

Here we have used $\lg w = O((\lg\lg n)^2)$, which is the regime in which our bound for small space can be an improvement over the communication bound.

Handling Larger Universes. We now show how one can take advantage of a higher w to obtain larger lower bounds. We continue to assume $w \geq 3\lg n$. Our strategy is to use the smallest t_i possible according to (2) and superconstant h. To analyze the recursion for ℓ_i, we just bound $t_i \leq n$, so $\ell_{i+1} \geq \frac{\ell_i}{h} - \lg n$. Using Lemma 6, we have $\ell_T \geq 1$ for $T \leq \Theta(\lg_h(\frac{w}{\lg n}))$. We also have the recursion:

$$\lg\frac{n}{k_{i+1}} = \lg\frac{n}{k_i} - \lg t_i = \left(1 - \frac{1}{h}\right)\lg\frac{n}{k_i} - \frac{a}{h} - O(\lg^2 w)$$

Again by Lemma 6, we see that $\frac{n}{k_T} \geq 1$ if:

$$T \leq \Theta\left(\lg \frac{\lg n}{h \cdot (\frac{a}{h} + \lg^2 w)} \Big/ \lg \frac{1}{1 - \frac{1}{h}}\right) = \Theta\left(h \lg \frac{\lg n}{a + h \lg^2 w}\right) = \Theta\left(\min\left\{h \lg \frac{\lg n}{a}, h \lg \frac{\lg n}{h \lg^2 w}\right\}\right)$$

As mentioned before, the condition $\ell_T \geq 1$ in (3) implies $T = O(\lg w)$, so we can assume $h = O(\lg w)$. Remember that we are assuming $\lg w = O((\lg \lg n)^2)$, so the second term in the min is just $\Theta(h \lg \lg n)$. Then, the entire expression simplifies to $\Theta(h \lg \frac{\lg n}{a})$.

The lower bound we obtain is be the minimum of the bounds derived by considering ℓ_i and k_i. We then choose h to maximize this minimum, arriving at:

$$\Theta\left(\max_h \min\left\{\frac{\lg(w/\lg n)}{\lg h}, h \lg \frac{\lg n}{a}\right\}\right)$$

Clearly, the $\Omega(\lg \frac{\lg n}{a})$ bound derived previously still holds. Then, we can claim a lower bound that is the maximum of this and our new bound, or, equivalently up to constants, their sum:

$$\begin{aligned}
& \lg \frac{\lg n}{a} + \max_h \min\left\{\frac{\lg(w/\lg n)}{\lg h}, h \lg \frac{\lg n}{a}\right\} = \max_h \min\left\{\lg \frac{\lg n}{a} + \frac{\lg(w/\lg n)}{\lg h}, (h+1) \lg \frac{\lg n}{a}\right\} \\
\geq \; & \max_h \min\left\{\frac{\lg(w/\lg n) + \lg(\lg n/a)}{\lg h}, h \lg \frac{\lg n}{a}\right\} = \max_h \min\left\{\frac{\lg(w/a)}{\lg h}, h \lg \frac{\lg n}{a}\right\}
\end{aligned}$$

We choose h to balance the two terms, so $h \lg h = \frac{\lg(w/a)}{\lg(\lg n/a)}$ and $\lg h = \Theta(\lg \lg \frac{w}{a} - \lg \lg \frac{\lg n}{a})$. Then the bound is $\Omega(\frac{\lg(w/a)}{\lg \lg(w/a) - \lg \lg(\lg n/a)})$.

Handling Smaller Universes. Finally, we consider smaller universes, i.e. $w < 3 \lg n$. Let $w = \delta + \lg n$. We start by applying Lemma 3 once, with $t = n/2^{\delta/2}$. Now we are looking at the problem $\bigoplus^t P(2^{\delta/2}, \frac{3}{2}\delta)$. Observe that the subproblems have a universe which is cubic in the number of integers in the subproblem. Then, we can just apply our strategy for polynomial universes, starting with $\ell_0 = \frac{3}{2}\delta$ and $n_0 = 2^{\delta/2}$. We obtain a lower bound of $\Omega(\lg \frac{\delta}{a}) = \Omega(\lg \frac{w - \lg n}{a})$.

3 Proof of Cell-Probe Elimination

We assume a solution to $\bigoplus^k f^{(h)}$, and use it to construct a solution to $\bigoplus^k f$. The new solution uses the query algorithm of the old solution, but skips the first cell probe made by this algorithm. A central component of our construction is a structural property about any query algorithm for $\bigoplus^k f^{(h)}$ with the input distribution $\bigoplus^k \mathcal{D}^{(h)}$. We now define and claim this property. Section 3.1 uses it to construct a solution for $\bigoplus^k f$, while Section 3.2 gives the proof.

We first introduce some convenient notation. Remember that the data structure's input for $\bigoplus^k f^{(h)}$ consists of a vector $(d^1, \ldots, d^k) \in D^k$, a vector selecting the interesting segments $(r^1, \ldots, r^k) \in [h]^k$ and the query prefixes Q^i_j for all $j \in [r^i - 1]$. Denote by $\mathbf{d}, \mathbf{r}$ and $\mathbf{Q}$ the random variables giving these three components of the input. Also let $\mathbf{p}$ be the random variable representing the bits published by the data structure. Note that $\mathbf{p}$ can also be understood as a function $\mathbf{p}(\mathbf{d}, \mathbf{r}, \mathbf{Q})$. The query consists of an index i selecting the interesting subproblem, and a vector $(q_1, \ldots, q_h)$ with a query to that subproblem. Denote by $\mathbf{i}$ and $\mathbf{q}$ these random variables. Note that in our probability space $\bigoplus^k f^{(h)}$, we have $\mathbf{q}_j = \mathbf{Q}^{\mathbf{i}}_j, (\forall) j < \mathbf{r}^{\mathbf{i}}$.

Fix some instance p of the published bits and a subproblem index $i \in [k]$. Consider a prefix $(q_1, \ldots, q_j)$ for a query to this subproblem. Depending on $q_{j+1}, \ldots, q_h$, the query algorithm might begin by probing different cells, or might reject the query. Let $\Gamma^i(p; q_1, \ldots, q_j)$ be the set of cells that could be inspected by the first cell probe. Note that this set could be $\varnothing$, if all queries are rejected.

Now define:

$$\varepsilon^i(p) = \begin{cases} 0 & \text{if } \Gamma^i(p; \mathbf{Q}^i) = \varnothing \\ \Pr\left[|\Gamma^i(p; \mathbf{q}_1, \ldots, \mathbf{q}_{\mathbf{r}^i})| \geq \frac{\min\{\sigma, |\Gamma^i(p; \mathbf{Q}^i)|\}}{\sqrt[h]{\sigma}} \mid \mathbf{i} = i \right] & \text{otherwise} \end{cases} \tag{4}$$

The probability space is that defined by $\bigoplus^k \mathcal{D}^{(h)}$ when the query is to subproblem i. In particular, such a query will satisfy $\mathbf{q}_j = \mathbf{Q}^i_j, (\forall) j < \mathbf{r}^i$, because the prefix is known to the data structure. Note that this definition completely ignores the suffix $\mathbf{q}_{\mathbf{r}^i+1}, \ldots, \mathbf{q}_h$ of the query. The intuition behind this is that for any choice of the suffix, the correct answer to the query is the same, so this suffix can be "manufactured" at will. Indeed, an arbitrary choice of the suffix is buried in the definition of Γ^i.

With these observations, it is easier to understand (4). If the data structure knows that no query to subproblem i will be successful, $\varepsilon_i = 0$. Otherwise, we compare two sets of cells. The first contains the cells that the querier might probe given what the data structure knows: $\Gamma^i(p, \mathbf{Q}^i)$ contains all cells that could be probed for various $\mathbf{q}^i_{\mathbf{r}^i}$ and various suffixes. The second contains the cells that the querier could choose to probe considering its given input $\mathbf{q}^i_{\mathbf{r}^i}$ (the querier is only free to choose the suffix). Obviously, the second set is a subset of the first. The good case, whose probability is measured by ε_i, is when it is a rather large subset, or at least large compared to σ.

For convenience, we define $\varepsilon^*(p) = \mathbf{E}_{i \leftarrow [k]}[\varepsilon^i(p)] = \frac{1}{k} \sum_i \varepsilon^i(p)$. Using standard notation from probability theory, we write $\varepsilon^i(p \mid E)$, when we condition on some event E in the probability of (4). We also write $\varepsilon^i(p \mid X)$ when we condition on some random variable X, i.e. $\varepsilon^i(p \mid X)$ is a function $x \mapsto \varepsilon^i(p \mid X = x)$. We are now ready to state our claim, to be proven in Section 3.2.

Lemma 7. *There exist* $\mathfrak{r}$ *and* $\mathfrak{Q}$*, such that* $\mathbf{E}_{\mathbf{d}}[\varepsilon^*(\mathbf{p}(\mathfrak{r}, \mathfrak{Q}, \mathbf{d}) \mid \mathbf{r} = \mathfrak{r}, \mathbf{Q} = \mathfrak{Q}, \mathbf{d})] \geq \frac{\delta}{2h}$.

3.1 The Solution for $\bigoplus^k f$

As mentioned before, we use the solution for $\bigoplus^k f^{(h)}$, and try to skip the first cell probe. To use this strategy, we need to extend an instance of $\bigoplus^k f$ to an instance of $\bigoplus^k f^{(h)}$. This is done using the $\mathfrak{r}$ and $\mathfrak{Q}$ values whose existence is guaranteed by Lemma 7. The extended data structure's input consists of the vector $(d^1, \ldots, d^k)$ given to $\bigoplus^k f$, and the vectors $\mathfrak{r}$ and $\mathfrak{Q}$. A query's input for $\bigoplus^k f$ is a problem index $i \in [k]$ and a $q \in Q$. We extend this to $(q_1, \ldots, q_h)$ by letting $q_j = \mathfrak{Q}^i_j, (\forall) j < \mathfrak{r}^i$, and $q_{\mathfrak{r}^i} = q$, and manufacturing a suffix $q_{\mathfrak{r}^i+1}, \ldots, q_h$ as described below.

First note that extending an input of $\bigoplus^k f$ to an input of $\bigoplus^k f^{(h)}$ by this strategy preserves the desired answer to a query (in particular, the suffix is irrelevant to the answer). Also, this transformation is well defined because $\mathfrak{r}$ and $\mathfrak{Q}$ are "constants", defined by the input distribution $\bigoplus^k \mathcal{D}^{(h)}$. Since our model is nonuniform, we only care about the existence of $\mathfrak{r}$ and $\mathfrak{Q}$, and not about computational aspects.

To fully describe a solution to $\bigoplus^k f$, we must specify how to obtain the data structure's representation and the published bits, and how the query algorithm works. The data structure's representation is identical to the representation for $\bigoplus^k f^{(h)}$, given the extended input. The published bits for $\bigoplus^k f$ consist of the published bits for $\bigoplus^k f^{(h)}$, plus a number of published cells from the data structure's representation. Which cells are published will be detailed below. We publish the cell address together with its contents, so that the query algorithm can tell whether a particular cell is available.

The query algorithm is now simple to describe. Remember that $q_1, \dots, q_{\mathfrak{r}^i - 1}$ are prescribed by $\mathfrak{Q}$, and $q_{\mathfrak{r}^i} = q$ is the original input of $\bigoplus^k f$. We now iterate through all possible query suffixes. For each possibility, we simulate the extended query using the algorithm for $\bigoplus^k f^{(h)}$. If this algorithm rejects the query, or the first probed cell is not among the published cells, we continue trying suffixes. Otherwise, we stop, obtain the value for the first cell probe from the published cells and continue to simulate this query using actual cell probes. If we don't find any good suffix, we reject the query. It is essential that we can recognize success in the old algorithm by looking just at published bits. Then, searching for a suffix that would not be rejected is free, as it does not involve any cell probes.

Publishing cells. It remains to describe which cells the data structure chooses to publish, in order to make the query algorithm successful with the desired probability. Let p be the bits published by the $\bigoplus^k f^{(h)}$ solution. Note that in order to make the query (i, q) successful, we must publish one cell from $\Gamma^i(p; \mathfrak{Q}^i, q)$. Here, we slightly abuse notation by letting $\mathfrak{Q}^i, q$ denote the $\mathfrak{r}^i$ entries of the prefix $\mathfrak{Q}^i$, followed by q. We will be able to achieve this for all (i, q) satisfying:

$$\Gamma^i(p; \mathfrak{Q}^i) \neq \varnothing \qquad \text{and} \qquad |\Gamma^i(p; \mathfrak{Q}^i, q)| \geq \frac{\min\{\sigma, |\Gamma^i(p; \mathbf{Q}^i)|\}}{\sqrt[h]{\sigma}} \tag{5}$$

Comparing to (4), this means the success probability is at least $\varepsilon^*(p \mid \mathbf{r} = \mathfrak{r}, \mathbf{Q} = \mathfrak{Q}, \mathbf{d} = (d_1, \dots, d_k))$. Then on average over possible inputs $(d_1, \dots, d_k)$ to $\bigoplus^k f$, the success probability will be at least $\frac{\delta}{2h}$, as guaranteed by Lemma 7.

We will need the following standard result:

Lemma 8. *Consider a universe $U \neq \varnothing$ and a family of sets $\mathcal{F}$ such that $(\forall) S \in \mathcal{F}$ we have $S \subset U$ and $|S| \geq \frac{|U|}{B}$. Then there exists a set $T \subset U, |T| \leq B \ln |\mathcal{F}|$ such that $(\forall) S \in \mathcal{F}, S \cap T \neq \varnothing$.*

Proof. Choose $B \ln |\mathcal{F}|$ elements of U with replacement. For a fixed $S \in \mathcal{F}$, an element is outside S with probability at most $1 - \frac{1}{B}$. The probability all elements are outside S is at most $(1 - \frac{1}{B})^{B \ln |\mathcal{F}|} < e^{-\ln |\mathcal{F}|} < \frac{1}{|\mathcal{F}|}$. By the union bound, all sets in $\mathcal{F}$ are hit at least once with positive probability, so a good T exists. ☐

We distinguish three types of subproblems, parallel to (5). If $\Gamma^i(p; \mathfrak{Q}^i) = \varnothing$, we make no claim (the success probability can be zero). Otherwise, if $|\Gamma^i(p; \mathfrak{Q}^i)| < \sigma$, we handle subproblem i using a local strategy. Consider all q such that $|\Gamma^i(p; \mathfrak{Q}^i, q)| \geq \frac{|\Gamma^i(p; \mathfrak{Q}^i)|}{\sqrt[h]{\sigma}}$. We now apply Lemma 8 with the universe $\Gamma^i(p; \mathfrak{Q}^i)$ and the family $\Gamma^i(p; \mathfrak{Q}^i, q)$, for all interesting q's. There are at most 2^w choices of q, bounding the size of the family. Then, the lemma guarantees that the data structure can publish a set of $O(\sqrt[h]{\sigma} \cdot w)$ cells which contains at least one cell from each interesting set. This means that each interesting q can be handled successfully by the algorithm.

We handle the third type of subproblems, those with $|\Gamma^i(p; \mathfrak{Q}^i)| \geq \sigma$, in a global fashion. Consider all "interesting" pairs (i, q) with $|\Gamma^i(p; \mathfrak{Q}^i, q)| \geq \sigma^{1-1/h}$. We now apply Lemma 8 with the universe consisting of all $k\sigma$ cells, and the family being $\Gamma^i(p; \mathfrak{Q}^i, q)$, for interesting (i, q). The cardinality of the family is at most 2^w, since i and q form a query, which takes at most one word. Then by Lemma 8, the data structure can publish a set of $O(k \sqrt[h]{\sigma} \cdot w)$ cells, which contains at least one cell from each interesting set. With these cells, the algorithm can handle successfully all interesting (i, q) queries.

The total number of cells that we publish is $O(k \sqrt[h]{\sigma} \cdot w)$. Thus, we publish $O(k \sqrt[h]{\sigma} \cdot w^2)$ new bits, plus $O(k)$ bits from the assumed solution to $\bigoplus^k f^{(h)}$. For big enough C, this is at most $k \sqrt[h]{\sigma} \cdot Cw^2$.

3.2 An Analysis of $\bigoplus^k f^{(h)}$: Proof of Lemma 7

Our analysis has two parts. First, we ignore the help given by the published bits, by assuming they are constantly set to some value p. As $\mathbf{r}^i$ and $\mathbf{Q}^i$ are chosen randomly, we show that the conditions of (4) are met with probability at least $\frac{1}{h}$ times the success probability for subproblem i. This is essentially a lower bound on ε^i, and hence on ε^*.

Secondly, we show that the published bits do not really affect this lower bound on ε^*. The intuition is that there are two few published bits (much fewer than k) so for most subproblems they are providing no information at all. That is, the behavior for that subproblem is statistically close to when the published bits would not be used. Formally, this takes no more than a (subtle) application of Chernoff bounds. The gist of the idea is to consider some setting p for the published bits, and all possible inputs (not just those leading to p being published). In this probability space, ε^i are independent for different i, so the average is close to ε^* with overwhelmingly high probability. Now pessimistically assume all inputs where the average of ε^i is not close to ε^* are possible inputs, i.e. input for which p would be the real help bits. However, the probability of this event is so small, that even after a union bound for all p, it is still negligible.

We now proceed to the first part of the analysis. Let $\delta^i(p)$ be the probability that the query algorithm is successful when receiving a random query for subproblem i. Formally, $\delta^i(p) = \Pr[\Gamma^i(p; \mathbf{q}) \neq \varnothing \mid \mathbf{i} = i]$. We define $\delta^i(p \mid E), \delta^i(p \mid X)$ and $\delta^*(\cdot)$ similar to the functions associated to ε^i. Observe that the probability of correctness guaranteed by assumption is $\delta = \mathbf{E}_{\mathbf{r},\mathbf{Q},\mathbf{d}}[\delta^*(\mathbf{p}(\mathbf{r}, \mathbf{Q}, \mathbf{d}) \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})]$.

Lemma 9. *For any i and p, we have $\varepsilon^i(p) \geq \frac{\delta^i(p)}{h}$.*

Proof. Let us first recall the random experiment defining $\varepsilon^i(p)$. We select a uniformly random $r \in [h]$ and random $q_1, \ldots, q_{r-1}$. First we ask whether $\Gamma^i(p; q_1, \ldots, q_{r-1}) = \varnothing$. If not, we ask about the probability that a random q_r is good, in the sense of (4). Now let us rephrase the probability space as follows: first select $q_1, \ldots, q_h$ at random; then select $r \in [h]$ and use just $q_1, \ldots, q_r$ as above. The probability that the query $(q_1, \ldots, q_h)$ is handled successfully is precisely $\delta^i(p)$. Let's assume it doesn't. Then, for any r, $\Gamma^i(p; q_1, \ldots, q_{r-1}) \neq \varnothing$ because there is at least one suffix which is handled successfully. We will now show that there is at least one choice of r such that q_r is good when the prefix is $q_1, \ldots, q_{r-1}$. When averaged over $q_1, \ldots, q_{r-1}$, this gives a probability of at least $\frac{\delta^i(p)}{h}$

To show one good r, let $\phi_r = \min\{|\Gamma^i(p; q_1, \ldots, q_{r-1})|, \sigma\}$. Now observe that $\frac{\phi_1}{\phi_2} \cdot \frac{\phi_2}{\phi_3} \cdot \cdots \cdot \frac{\phi_{h-1}}{\phi_h} = \frac{\phi_1}{\phi_h} \leq \phi_1 \leq \sigma$. By the pigeonhole principle, $(\exists) r : \frac{\phi_r}{\phi_{r+1}} \leq \sigma^{1/h}$. This implies $|\Gamma^i(p; q_1, \ldots, q_r)| \geq \frac{\min\{\sigma, |\Gamma^i(p; q_1, \ldots, q_{r-1})|}{\sqrt[h]{\sigma}}$, as desired. $\square$

Note that if the algorithm uses zero published bits, we are done. Thus, for the rest of the analysis we may assume $\frac{1}{C}(\frac{\delta}{h})^3 k \geq 1$. We now proceed to the second part of the analysis, showing that ε^* is close to the lower bound of the previous lemma, even after a union bound over all possible published bits.

Lemma 10. *With probability at least* $1 - \frac{\delta}{8h}$ *over random* $\mathbf{r}, \mathbf{Q}$ *and* $\mathbf{d}$*:* $(\forall)p: \ \varepsilon^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \geq \frac{\delta^*(p)}{h} - \frac{\delta}{4h}$

Proof. Fix p arbitrarily. By definition, $\varepsilon^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) = \frac{1}{k}\sum_i \varepsilon^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$. By Lemma 9, $\mathbf{E}[\varepsilon^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})] = \varepsilon^i(p) \geq \frac{\delta^i(p)}{h}$, which implies $\varepsilon^*(p) \geq \frac{\delta^*(p)}{h}$. Thus, our condition can be rephrased as:

$$\frac{1}{k}\sum_i \varepsilon^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \geq \mathbf{E}\left[\frac{1}{k}\sum_i \varepsilon^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})\right] - \frac{\delta}{4h}$$

Now note that $\varepsilon^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$ only depends on $\mathbf{r}^i, \mathbf{Q}^i$ and $\mathbf{d}^i$, since we are looking at the behavior of a query to subproblem i for a fixed value of the published bits; see the definition of ε^i in (4). Since $(\mathbf{r}^i, \mathbf{Q}^i, \mathbf{d}^i)$ are independent for different i, it follows that $\varepsilon^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$ are also independent. Then we can apply a Chernoff bound to analyze the mean $\varepsilon^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$ of these independent random variables. We use an additive Chernoff bound [2]:

$$\Pr_{\mathbf{r}, \mathbf{Q}, \mathbf{d}}\left[\varepsilon^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) < \varepsilon^*(p) - \frac{\delta}{4h}\right] < e^{-\Omega(k(\frac{\delta}{h})^2)}$$

Now we take a union bound over all possible choices p for the published bits. The probability of the bad event becomes $2^{\frac{1}{C}(\frac{\delta}{h})^3 k} e^{-\Omega((\frac{\delta}{h})^2 k)}$. For large enough C, this is $\exp(-\Omega((\frac{\delta}{h})^2 k))$, for any δ and h. Now we use that $\frac{1}{C}(\frac{\delta}{h})^3 k \geq 1$, from the condition that there is at lest one published bit, so this probability is at most $e^{-\Omega(Ch/\delta)}$. Given that $\frac{h}{\delta} \geq 1$, this is at most $\frac{\delta}{8h}$ for large enough C. □

Unfortunately, this lemma is not exactly what we would want, since it provides a lower bound in terms of $\delta^*(p)$. This probability of success is measured in the original probability space. As we condition on $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$, the probability space can be quite different. However, we show next that in fact δ^* cannot change too much. As before, the intuition is that there are too few published bits, so for most subproblems they are not changing the query distribution significantly.

Lemma 11. *With probability at least* $1 - \frac{\delta}{8}$ *over random* $\mathbf{r}, \mathbf{Q}$ *and* $\mathbf{d}$*:* $(\forall)p: \ \delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \leq \delta^*(p) + \frac{\delta}{4}$

Proof. The proof is very similar to that of Lemma 10. Fix p arbitrarily. By definition, $\delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$ is the average of $\delta^i(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})$. Note that for fixed p, δ^i depends only on $\mathbf{r}^i, \mathbf{Q}^i$ and $\mathbf{d}^i$. Hence, the δ^i values are independent for different i, and we can apply a Chernoff bound to say the mean is close to its expectation. The rest of the calculation is parallel to that of Lemma 10. □

We combine Lemmas 10 and 11 by a union bound. We conclude that with probability at least $1 - \frac{\delta}{4}$ over random $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$, we have that $(\forall)p$:

$$\left.\begin{array}{l} \varepsilon^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \geq \frac{\delta^*(p)}{h} - \frac{\delta}{4h} \\ \delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) \leq \delta^*(p) + \frac{\delta}{4} \end{array}\right\} \Rightarrow \varepsilon^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) - \frac{\delta^*(p \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})}{h} \geq -\frac{\delta}{2h} \tag{6}$$

Since this holds for all p, it also holds for $p = \mathbf{p}$, i.e. the actual bits $\mathbf{p}(\mathbf{r}, \mathbf{Q}, \mathbf{d})$ published by the data structure given its input. Now we want to take the expectation over $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$. Because

$\varepsilon^*(\cdot), \delta^*(\cdot) \in [0,1]$, we have $\varepsilon^*(\cdot) - \frac{1}{h}\delta^*(\cdot) \geq -\frac{1}{h}$. We use this as a pessimistic estimate for the cases of $\mathbf{r}, \mathbf{Q}$ and $\mathbf{d}$ where (6) does not hold. We obtain:

$$\mathbf{E}\left[\varepsilon^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d}) - \frac{\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})}{h}\right] \geq -\frac{\delta}{2h} + \frac{\delta}{4} \cdot \left(-\frac{1}{h}\right) = -\frac{3\delta}{4h}$$
$$\Rightarrow \quad \mathbf{E}\big[\varepsilon^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})\big] \geq \frac{1}{h}\mathbf{E}\big[\delta^*(\mathbf{p} \mid \mathbf{r}, \mathbf{Q}, \mathbf{d})\big] - \frac{3\delta}{4h} = \frac{1}{h}\delta - \frac{3\delta}{4h} = \frac{\delta}{4h}$$

4 Communication Lower Bounds

4.1 Protocol Manipulations

To obtain our improved lower bounds for large space, we use two-party communication complexity. In this section, we state the protocol manipulation tools that we will use in our proof. We allow protocols to make errors, and look at the error probability under appropriate input distributions. Thus, as opposed to our lower bounds for small space, we also obtain lower bounds for randomized algorithms with bounded error. We define an $[A; m_1, m_2, m_3, \dots]$-protocol to be a protocol in which Alice speaks first, sending m_1 bits, Bob then sends m_2 bits, Alice sends m_3 bits and so on. In a $[B; m_1, m_2, \dots]$-protocol, Bob begins by sending m_1 bits.

For a communication problem $f : A \times B \to \{0,1\}$, define a new problem $f^{A,(k)}$ in which Alice receives $x_1, \dots, x_k \in A$, Bob receives $y \in B, i \in [k]$ and $x_1, \dots, x_{i-1}$, and they wish to compute $f(x_i, y)$. This is similar to our definition for $f^{(t)}$, except that we need to specify that Alice's input is being multiplied. We define $f^{B,(k)}$ symmetrically, with the roles of Alice and Bob reversed. Finally, given a distribution $\mathcal{D}$ for f, we define $\mathcal{D}^{A,(k)}$ and $\mathcal{D}^{B,(k)}$ following our old definition for $\mathcal{D}^{(k)}$.

The first tool we use is round elimination, which, as mentioned before, has traditionally been motivated by predecessor lower bounds. The following is a strong version of this result, due to [16]:

Lemma 12 (round elimination [16]). *Suppose $f^{(k),A}$ has an $[A; m_1, m_2, \dots]$-protocol with error probability at most ε on $\mathcal{D}^{A,(k)}$. Then f has a $[B; m_2, \dots]$-protocol with error probability at most $\varepsilon + O(\sqrt{\frac{m_1}{k}})$ on $\mathcal{D}$.*

As opposed to previous proofs, we also bring message compression into play. The following is from [5], restated in terms of our $f^{A,(k)}$ problem:

Lemma 13 (message compression [5]). *Suppose $f^{(k),A}$ has an $[A; m_1, m_2, \dots]$ protocol with error probability at most ε on $\mathcal{D}^{A,(k)}$. Then for any $\delta > 0$, f has an $[A; O(\frac{1+(m_1/k)}{\delta^2}), m_2, \dots]$-protocol with error probability at most $\varepsilon + \delta$ on $\mathcal{D}$.*

Since this lemma does not eliminate Alice's message, but merely reduces it, it is used in conjuction with the message switching technique [5]. If Alice's first message has a bits, we can eliminate it if Bob sends his reply to all possible messages from Alice (thus increasing his message by a factor of 2^a), and then Alice includes her first message along with the second one (increasing the second message size additively by a):

Lemma 14 (message switching). *Suppose f has an $[A; m_1, m_2, m_3, m_4, \dots]$-protocol. Then it also has a $[B; 2^{m_1} m_2, m_1 + m_3, m_4, \dots]$-protocol with the same error complexity.*

Message compression combined with message switching represent, in some sense, a generalization of the round elimination lemma, allowing us to trade a smaller k for a larger penalty in Bob's messages. However, the trade-off does yield round elimination as the end-point, because message compression cannot reduce Alice's message below $\Omega(\delta^{-2})$ for any k. We combine these two lemmas to yield a smooth trade-off (with slightly worse error bounds), which is easier to work with:

Lemma 15. *Suppose $f^{(k)}$ has an $[A; m_1, m_2, m_3, m_4, \dots]$-protocol with error ε on $\mathcal{D}^{A,(k)}$. Then for any $\delta > 0$, f has a $[B; 2^{O(m_1/(k\delta^4))} m_2, m_1 + m_3, m_4, \dots]$-protocol with error probability $\varepsilon + \delta$ on $\mathcal{D}$.*

Proof. If $\frac{m_1}{k} \le \delta^2$, we can apply the round elimination lemma. Then, Alice's first message is ommitted with an error increase of at most δ. None of the subsequent messages change. If $\frac{m_1}{k} \ge \delta^2$, we apply the message compression lemma, which reduces Alice's first message to $O(\frac{1+(m_1/k)}{\delta^2})$ bits, while increasing the error by δ. Since $\frac{m_1}{k} \ge \delta^2$, the bound on Alice's message is at most $O(\frac{m_1}{k\delta^4})$. Then, we can eliminate Alice's first message by switching. Note our bound for the second message from Alice is loose, since it ignores the compression we have done. □

4.2 Application to Predecessor Search

Theorem 16. *Consider a solution to colored predecessor search in a set of n ℓ-bit integers, which uses space $n \cdot 2^a$ in the cell-probe model with cells of w bits. If $a = \Omega(\lg n)$ and the query algorithm has an error probability of at most $\frac{1}{3}$, the query time must satisfy:*

$$T = \Omega\left(\min\left\{\lg_w n,\ \frac{\lg(\ell/a)}{\lg\lg(\ell/a) + \lg(a/\lg n)}\right\}\right)$$

Proof. We consider the communication game in which Alice receives the query and Bob receives the set of integers. Alice's messages will have $\lg(n \cdot 2^a) = \Theta(a)$ bits, and Bob's w bits. The structure of our proof is similar to the application of the cell-probe elimination lemma in Section 2. By Lemma 2, we can identify the structure of $P(n,\ell)^{A,(h)}$ in $P(n, h\ell)$. Then, we can will apply our Lemma 15 to eliminate Alice's messages. Now, we use Lemma 3 to identify the structure of $P(n,\ell)^{B,(t)}$ in $P(n \cdot t, \ell + \lg t)$. Note that $P(n,\ell)^{B,(t)}$ is syntactically equivalent to our old $\bigoplus^t P(n,\ell)$, except that Alice also receives a (useless) prefix of Bob's input. Now we apply the round elimination lemma to get rid of Bob's message.

Thus, after eliminating a message from each player, we are left with another instance of the colored predecessor problem, with smaller n and ℓ parameters. This contrasts with our cell-probe proof, which couldn't work with just one subproblem, but needed to look at all of them to analyzing sharing. Our strategy is to increase the error by at most $\frac{1}{9T}$ in each round of the previous argument. Then, after T steps, we obtain an error of at most $\frac{1}{3} + \frac{1}{9} < \frac{1}{2}$. Assuming we still have $n \ge 2$ and $\ell \ge 1$, it is trivial to make the answer to the query be either red or blue with equal probability. Then, no protocol with zero communication can have error complexity below $\frac{1}{2}$, so the original cell-probe complexity had to be greater than T.

As explained in Section 2.3, the proof should be interpreted as an inductive argument in the reverse direction. Assuming we have a distribution on which no protocol with i rounds can have error less than ε, our argument constructs a distribution on which no protocol with $i+1$ rounds can have error less than $\varepsilon - \frac{1}{9T}$. At the end, we obtain a distribution on which no protocol with T rounds can have error $\frac{1}{3}$, implying the cell-probe lower bound.

It remains to define appropriate values h and t which maximize our lower bound T by the above discussion. After step i, Alice's message will have size $(i+1) \cdot (a + \lg n) = O(aT)$, because we have applied message switching i times (in the form of Lemma 15). Applying Lemma 15 one more time with $\delta = \frac{1}{18T}$, we increases Bob's next message to $w \cdot 2^{O(aT^5/h)}$. We now apply round elimination to get rid of Bob's message. We want an error increase of at most $\frac{1}{18T}$, adding up to at most $\frac{1}{9T}$ per round. Then, we set t according to:

$$O\left(\sqrt{\frac{w \cdot 2^{O(aT^5/h)}}{t}}\right) \le \frac{1}{18T} \quad \Rightarrow \quad t = w \cdot O(T^2) \cdot 2^{O(aT^5/h)}$$

Let n_i and ℓ_i denote the problem parameters after i steps of our argument. Initially, $n_0 = n, \ell_0 = \ell$. By the discussion above, we have the recursions: $\ell_{i+1} = \frac{\ell_i}{h} - \lg t$ and $\lg n_{i+1} = \lg \frac{n_i}{t} = \lg n_i - \lg t$.

We have $\lg t = O(\lg w + \lg T + \frac{aT^5}{h})$. Since we want $\ell_T \ge 1$, we must have $T \le \lg \ell \le \lg w$, so $\lg t$ simplifies to $O(\lg w + \frac{aT^5}{h})$. Now the condition $n_T \ge 2$ implies the following bound on T:

$$T < \frac{\lg n - 1}{\Theta(\lg w + \frac{aT^5}{h})} = \Theta\left(\min\left\{\frac{\lg n}{\lg w}, \frac{h \lg n}{aT^5}\right\}\right)$$

To analyze the condition $\ell_T \ge 1$, we apply the recursion bound of Lemma 6, implying $T < \lg_h(\frac{\ell}{\Theta(\lg t)})$. This is satisfied if we upper bound $\lg t$ by $O((a + \lg w) \cdot (1 + \frac{T^5}{h}))$, and set:

$$T < \Theta\left(\frac{\lg(\frac{\ell}{a+\lg w}) - \lg(\frac{T^5}{h})}{\lg h}\right) = \Theta\left(\frac{\lg(\frac{\ell}{a+\lg w})}{\lg h}\right) - O(\lg T) \quad \Rightarrow \quad T < \Theta\left(\frac{\lg(\frac{\ell}{a+\lg w})}{\lg h}\right)$$

Thus, our lower bound is, up to constant factors, $\min\{\frac{\lg n}{\lg w}, \frac{h \lg n}{aT^5}, \frac{\lg(\ell/(a+\lg w))}{\lg h}\}$. First we argue that we can simplify $a + \lg w$ to just a in the last term. If $a = \Omega(\lg w)$, this is trivial. Otherwise, we have $a = O(\lg w)$, so $\lg n = O(\lg w)$. But in this case the first term of the min is $O(1)$ anyway, so the other terms are irrelevant.

It now remains to choose h in order to maximize the lower bound. This is achieved when $\frac{h \lg n}{aT^5} = \frac{\lg(\ell/a)}{\lg h}$, so we should set $\lg h = \Theta(\lg \lg \frac{\ell}{a} + \lg \frac{a}{\lg n} + \lg T)$. The $\lg T$ term can be ignored because $T = O(\lg \frac{\ell}{a})$. With this choice of h, the lower bound becomes, up to constants, $\min\{\frac{\lg n}{\lg w}, \frac{\lg(\ell/a)}{\lg\lg(\ell/a) + \lg(a/\lg n)}\}$. $\square$

5 Upper Bounds

We are working on the static predecessor problem where we are first given a set Y of n keys. The predecessor of a query key x in Y is the largest key in Y that is smaller than or equal to x. If x is smaller than any key in Y, its predecessor is $-\infty$, representing a value smaller than any possible key. Below each key is assumed to be a non-negative ℓ-bit integer. We are working on a RAM with word length $w \ge \ell$. The results also apply in the stronger external memory model where w is the bit size of a block. The external memory model is stronger because it like the cell-probe model does not count computations.

For $n \le s$ and $\lg n \le \ell \le w$, we will show represent n ℓ-bit keys using $O(s\ell)$ bits of space where

$s \geq n$. With $a = \lg \frac{sw}{n}$, we will show how to search predecessors in time

$$O\left(\frac{\lg n}{\lg w}\right) \tag{7}$$

$$O\left(\lg \frac{\ell - \lg n}{a}\right) \quad \text{if} \quad \ell = O(\lg n) \tag{8}$$

$$O\left(\frac{\lg \frac{\ell}{a}}{\lg \frac{a \lg \frac{\ell}{a}}{(\lg n)}}\right) \quad \text{if} \quad a \geq \lg n \quad \text{and} \quad \ell = \omega(\lg n) \tag{9}$$

$$O\left(\frac{\lg \frac{\ell}{a}}{\lg \frac{\lg \frac{\ell}{a}}{\lg \frac{\lg n}{a}}}\right) \quad \text{if} \quad a \leq \lg n \quad \text{and} \quad \ell = \omega(\lg n) \tag{10}$$

Contents Below, we first obtain (7) using either B-trees or the fusion trees of Fredman and Willard [10]. Next we use (7) to increase the space by a factor w so that we have $O(2^a \ell)$ bits of space available per key. Then we prove (8) by a slight tuning of van Emde Boas' data structure [19]. This bound is tight when $w = O(\lg n)$. Next, elaborating on techniques of Beame and Fich [4], we will first show (9) and then (10) in the case where $w \geq 2 \lg n$.

5.1 Preliminaries

In our algorithms, we will assume that w, ℓ, and a are powers of two. For the word length w, we note we that can simulate up to twice the word length implementing each extended word operation with a constant number of regular operations. Hence, internally, our algorithms can use a word length rounded up to the nearest power of two without affecting the asymptotic search times. Concerning the parameters ℓ and a, we note that it does not affect the asymptotics if they change by constant factors, so we can freely round ℓ up to the nearest factor of two and a down to the nearest factor of two, thus accepting a larger key length and lesser space for the computations.

The search times will be achieved via a series of reductions that often reduce the key length ℓ. We will make sure that each reduction is by a power of two.

We will often allocate arrays with m entries, each of ℓ bits. These occupy $m\ell$ consecutive bits in memory, possibly starting and ending in the middle of words. As long as $\ell \leq w$, using simple arithmetic and shifts, we can access or change an entry in constant time. In our case, the calculations are particularly simple because ℓ and w are powers of two.

We will use product notation for concatenation of bit strings. Hence xy or $x \cdot y$ denotes the concatenation of bit strings x and y. As special notation, we define $-\infty \cdot x = x \cdot -\infty = -\infty$.

Finally, we define $\log = \log_2$. Note that this is different from lg which is the function used in our asymptotic bounds.

5.2 Fusion or B-trees

With Fredman and Willard's fusion trees [10], we immediately get a linear space predecessor search time of $O\left(\frac{\lg n}{\lg \ell}\right)$. If $w \leq \ell^2$, this implies the $O\left(\frac{\lg n}{\lg w}\right)$ search time from (7). Otherwise, we use a

B-tree of degree $d = w/\ell$. We can pack the d keys in a singe word, and we can then search a B-tree node in constant time using some of the simpler bit manipulation from [10]. This gives a search time of $O\left(\frac{\lg n}{\lg d}\right)$ which is $O\left(\frac{\lg n}{\lg w}\right)$ for $w \geq \ell^2$. Thus we achieve the search time from (7) using only linear space, or $O(n\ell)$ bits.

We note that the B-tree solution is simpler in the external memory model where we do not worry about the actual computations.

5.3 Inceasing the space

We will now use (7) to increase the space per key by a factor w. We simply pick out a set Y' of $n' = \lfloor n/w \rfloor$ equally spaced keys so that we have a segment of less than w keys between consecutive keys in Y'. We will first do a predecessor search in Y', and based on the result, do a predecessor search in the appropriate segment. Since the segment has less than w keys, by (7), it can be searched in constant time.

Thus we are left with the problem of doing a predecessor search in Y'. For this we have $O(s\ell)$ bits, which is $O(s\ell/(n/w)) = O(sw\ell/n) = O(2^a\ell)$ bits per key. Moreover we note that replacing n by $n' = \lfloor n/w \rfloor$ does not increase any of the bounds (8)-(10). Hence it suffices to prove the these bounds (8)-(10) assuming that we $O(n2^a\ell)$ bits of space available.

5.4 A tuned van Emde Boas bound for polynomial universes

In this section, we develop a tuned version of van Emde Boas's data structure, representing n keys in $O(n2^a\ell)$ bits of space providing the search time from (8) of

$$O\left(\lg \frac{\ell - \lg n}{a}\right).$$

We shall only use this bound for polynomial universes, that is, when $\ell = O(\lg n)$.

5.4.1 Complete tabulation

The static predecessor problem is particularly easy when we have room for a complete tabulation of all possible query keys, that is, if we have $O(2^\ell \ell)$ bits of space. Then we can allocate a table $pred_Y$ that for each possible query key x stores the predecessor $pred_Y[x]$ of x in Y. If $x < \min Y$, $pred_Y[x] = -\infty$. For our bounds, we will use this as a base case if $\ell \leq a$.

Note that this simple base case is a prime example of what we can do when not restricted to comparisons on a pointer machine: we use the key as an addrees to a table entry and get the answer in constant time.

5.4.2 Prefixes tabulation

If our keys are too long for a complete tabulation, but not too much longer, it may still be relevant to use tabulation based on the first p bits of each key. Below it is understood that the prefix of a key is the first p bits and the suffix is the last $\ell - p$ bits. Let $Suff_Y[u]$ be the suffixes in Y of keys with prefix u. Also let $pred^{\neq}_Y[u]$ to denote the strict predessor in Y of u suffixed by zeroes. Here by strict predecessor, we mean an unequal predecessor. If no length p prefix in Y is smaller than u, $pred^{\neq}_Y[u] = -\infty$. The representation of Y now consists of the table that with each prefix

u associates $pred_Y^{\neq}[u]$ and a recursive representatation of $\mathit{Suff}_Y[u]$. Note that if $\mathit{Suff}_Y[u] = \emptyset$, the recursive representation returns $-\infty$ on any predecessor query.

We now have the following pseudo-code for searching Y:

> $\mathit{Pred}(x, Y)$
> $\quad (x_0, x_1) = (\mathit{prefix}(x), \mathit{suffix}(x))$
> $\quad y_1 = \mathit{Pred}(x_1, \mathit{Suff}_Y[x_0])$
> $\quad$ if $y_1 = -\infty$ then return $pred_Y^{\neq}[x_0]$.
> $\quad$ return $x_0 \cdot y_1$

Note in the above pseudo-code that the produre termintes as soon as it executes return statement. Hence the last statement is only executed if $y_1 \neq -\infty$. Also, as a rule of thumb, we use square brackets around the argument of a function that we can compute in constant time..

We shall use this reduction with $p \approx \lg n$ as the first step of our predecessor search. More precisely, we choose $p \leq \lg n$ such that the reduced length $\ell - p$ is a power of two less than $2(\ell - \lg n)$. The reduction adds a constant to the search time. It uses $O(2^p \ell) = O(n\ell)$ bits of space on the tables over the prefixes. The suffix of a key y appears in the subproblem $\mathit{Suff}_Y[u]$. Hence the subproblems have a total of n keys, each of length less than $2(\ell - \lg n)$.

5.4.3 Van Emde Boas' reduction

The essential component of van Emde Boas' data structure is a reduction that halves the key length ℓ. Trivially this preserves that key lengths are powers of two. To do this halving, we would like to use the reduction above with prefix length $p = \ell/2$. However, if ℓ is too large, we do not have $O(2^p \ell)$ bits of space for tabulating all prefixes. As a limited start, we can use hashing to tabulate the above information for all prefixes of keys in Y. Let U be the set of these prefixes. For all $u \in U$, as above, we store $pred_Y^{\neq}[u]$ and a recursive representation of $\mathit{Suff}_Y[u]$. We can then handle all queries x with a prefix in U. However, if the prefix x_0 of x is not in u, we need a way to compute $pred_Y^{\neq}[x_0]$.

To compute $pred_Y^{\neq}[x_0]$ for a prefix x_0 not in U, we use a recursive representation of U. Moreover, with each $u \in U$, we store the maximal key $max_Y[u]$ in Y with prefix u. Moreover, we define $max_Y[-\infty] = -\infty$. We now first compute the predecessor y_0 of x_0 in U, and then we return $max_Y[y_0]$.

The above reduction spends constant time on halving the key length but the number of keys may grow in that a key $x = x_0 x_1 \in Y$ has x_0 in the subproblem U and x_1 in the subproblem $\mathit{Suff}_Y[x_0]$. A general solution is that instead of recursing directly on a subproblem Z, we remove the maximal key treating it separately, thus only recursing on $Z^- = Z \setminus \{\max Z\}$.

In our concrete case, we will consider the reduced recursive subproblems $\mathit{Suff}_Y^-[u]$. We then have the following recursive pseudo-code for searching the predecessor of x:

> $\mathit{Pred}(x, Y)$
> $\quad (x_0, x_1) = (\mathit{prefix}(x), \mathit{suffix}(x))$
> $\quad$ if $x_0 \notin U$ then return $max_Y[\mathit{Pred}(x_0, U)]$
> $\quad$ if $x \geq max_Y[x_0]$ then return $max_Y[x_0]$
> $\quad y_1 = \mathit{Pred}(x_1, \mathit{Suff}_Y^-[x_0])$

if $y_1 = -\infty$ then return $pred^{\neq}_{Y}[x_0]$
return $x_0 y_1$

The key lengths have been halved to $\ell' = \ell/2$. We have $n - |U|$ half keys in the suffix subproblems $\mathit{Suff}^{-}_{Y}[x_0]$, and $|U|$ half keys in the prefix subproblem U, so the total number of keys is n.

As described above, the space used by the reduction is $O(n\ell)$ bits.

5.4.4 The final combination

To solve the predecessor search problem in $O(n2^a\ell)$ bits of space, we will first tabulate a prefix of length $p \leq \lg n$ as described in Section 5.4.2, thus reducing the key length to $\ell - p \leq 2(\ell - \lg n)$ whish is a power of two. Then we apply the van Emde Boas reduction recursively as described in Section 5.4.3, until we get down to a key length below a. This requires $\lg \frac{\ell - \lg n}{a}$ recursions. We do not recurse on empty subproblems. For these we know that the predecessor is always 0. Finally we use the complete tabulation on each subproblem as described in Section 5.4.1.

Since each reduction adds a constant to the search time, the search time of our solution is $O(\lg \frac{\ell - \lg n}{a})$. The first reduction uses $O(n\ell)$ bits of space, and the last uses $O(2^a\ell)$ bits of space per subproblem. Since the subproblems are non-empty, this is $O(n2^a\ell)$ bits of space in total. Each van Emde Boas recursion uses $O(n\ell)$ bits of space where ℓ is the current key length. Since ℓ is halved each time, the space of the first iteration with the original key length ℓ dominates. Thus we have proved:

Lemma 17. *Using $O(n2^a\ell)$ bits of space, we can represent n ℓ-bit keys so to we can search predecessors in $O(\lg \frac{\ell - \lg n}{a})$ time.*

5.5 Reduction à la Beame and Fich

In this section, we will derive better bounds for larger universes using a reduction very similar to one used by Beame and Fich [4]. Our version of the reduction is captured in the following proposition:

Proposition 18. *Let be given an instance of the static predecessor search problem with n keys of length ℓ. Choose integer parameters $q \geq 2$ and $h \geq 2$ where h divides ℓ. We can now reduce into subproblems, each of which is easier in one of two ways:*

length reduced *The key length in the subproblem is reduced by a factor h to ℓ/h, and the subproblem contains at most half the keys.*

cardinality reduced *The number of keys is reduced by a factor q to n/q.*

The reduction costs a constant in the query time. For some number m determined by the reduction, the reduction uses $O((q^{(2h)} + m)\ell)$ bits of space. The total number of keys in the cardinality reduced subproblems is at most $n - m$, and the total number of keys in the length reduced subproblems is at most m.

The original reduction of Beame and Fich [4, Section 4.2] is specialized towards their overall quadratic space solution, and had an assumption that $\ell \leq \ell/h$. They satisfy this assumption by first applying van Emde Boas' reduction $\lg h$ times. This works fine in their case, but here we consider solutions to the predecessor search problem where we get down to constant query time using large space, and then their assumption would be problematic.

5.5.1 Larger space

Recall that we are looking for a solution to the predecessor problem using $O(n2^a\ell)$ bits of space. In our first simple solution assumes $a \geq \lg n$ and $\ell = \omega(\lg n)$. With some h to be fixed later we apply Proposition 18 recursively with q fixed as $2^{a/(2h)}$. Here a and h are assumed powers of two. Then the bit space used in each recursive step is $O(2^a\ell)$. Since no subproblem has more than half the keys, the recursion tree has no degree 1 nodes. Hence we have at most $n-1$ recursive nodes, so the total space used in the recursive steps is $O(n2^a\ell)$ bits.

As described in Section 5.4.1, we can stop recursing when we get down to key length a, so the number of length reductions in a branch is at most $\lg_h \frac{w}{a}$. On the other hand, the number of cardinality reductions in a branch is at most $\lg_{2^{a/(2h)}} n = \frac{2h(\lg n)}{a}$. Thus, for $n \leq s$, the recursion depth is at most

$$\frac{\lg \frac{w}{a}}{\lg h} + \frac{2h(\lg n)}{a}.$$

This expression is minimized with

$$h = \Theta\left(\frac{a \lg \frac{w}{a}}{\lg n} / \lg \frac{a \lg \frac{w}{a}}{\lg n}\right),$$

and then we get a query time of

$$O\left(\frac{\lg \frac{w}{a}}{\lg \frac{a \lg \frac{w}{a}}{(\lg n)}}\right).$$

Except for the division of w by a in $\frac{w}{a}$, the above bound is equivalent to one anticipated without any proof or construction in [18]. we shall prove that this bound is tight.

5.5.2 Smaller space

We now consider the case where we start with a problem with $\lg n \geq a/2$ and $\ell = \omega(\lg n)$. We are now going to appy Proposition 18 recursively with a fixed value of h which is a power of two, but with a changing value of q, stopping when we get a subproblem with only one key, or where the key length is at most a. While $\lg n \geq a/2$, we use Proposition 18 recursively with $q = \lfloor n^{1/(4h)} \rfloor$. However, when we get down to $n \leq 2^{a/2}$ keys, we use $q = 2^{a/(4h)}$.

Lemma 19. *The above construction uses $O(n2^a\ell)$ bits of space and the search time is*

$$O\left(\frac{\lg \frac{\ell}{a}}{\lg \frac{\lg \frac{\ell}{a}}{\lg \frac{\lg n}{a}}}\right).$$

Proof. First we analyze the search time which is the recursion depth. Since we start with key length at most ℓ and finish if we get to a, the number of length reductions in a recursion branch is $O(\lg_h \frac{\ell}{a})$.

For the cardinality reductions, while $n \geq 2^{a/2}$, we note that it takes less than $4h$ reductions to get from n to $\sqrt{n}$ keys. More precisely, in each of these reductions, we have $q > \sqrt{n}^{1/(4h)}$, and then it takes less than

$$\lg_{\sqrt{n}^{1/(4h)}} \frac{n}{\sqrt{n}} = \frac{\lg \sqrt{n}}{\lg n^{1/(8h)}} = 4h$$

cardinality reductions to get down to $\sqrt{n}$ keys. Thus it takes $4h$ cardinality reductions to half $\log n$, so to get from the original value and down to $a/2$, we need at most $4h\lceil \lg \frac{\lg n}{a/2} \rceil = O(h \lg \frac{\lg n}{a})$ cardinality reductions. In the above argument, we have ignored that q is rounded down to the nearest integer. However, since h is a power of two, we can use the same argument to show that we can have at most $4h$ iterations while $\log n \in [2^i, 2^{i+1})$.

Finally, starting from $n \leq 2^{a/2}$ keys and using $q = 2^{a/(2h)}$, we use at most h cardinality reductions to get down to a single key. Thus, the total number of cardinality reducing reductions is $O(h \lg \frac{\lg n}{a})$. It follows that the total recursion depth is at most

$$O\left(\frac{\lg \frac{\ell}{a}}{\lg h} + h \lg \frac{\lg n}{a}\right)$$

The search time stated in the lemma is obtained setting

$$h = \frac{\lg \frac{\ell}{a}}{\lg \frac{\lg n}{a}} / \lg \frac{\lg \frac{\ell}{a}}{\lg \frac{\lg n}{a}}.$$

The bit space bound in each reductive step is $O((q^{(2h)} + m)\ell)$. We will add up each term separately over the whole recursion tree.

For the $O(m\ell)$ bound, we note that at least m keys get reduced to length $\ell/h \leq \ell/2$. Thus, the total bit length of the keys is reduced by at least $m\ell/2$, so we use $O(1)$ bits per key bit saved. Starting with $n\ell$ key bits, the total space used is $O(n\ell)$.

Finally, concerning the $O(q^{(2h)}\ell)$ bound, we have to cases. When $q = \lfloor 2^{(\lg \sqrt{n})/(4h)} \rfloor$, the bit space used is $O(\sqrt{n}\ell)$. This is $O(\ell/\sqrt{n})$ bits of space per key in the recursion. Following a key x down the branch, we know that the number of keys is halved in each step, and this means that the space assigned to x is increased by a factor $\sqrt{2}$. Thus, the total space assigned to x is dominated by the last recursion, hence $O(\ell)$ bits. Thus, over all the keys, we get $O(n\ell)$ bits of space for this case.

Finally, when $q = 2^{a/(2h)}$, the bit space is $O(2^a\ell)$, and then the at most $n-1$ recursive nodes give a bit space bound of $O(n2^a\ell)$. Thus the whole thing adds up to $O(n2^a\ell)$ bits of space, as desired. □

5.5.3 Proof of Proposition 18

In this section, we prove Proposition 18:

> *Let be given an instance of the static predecessor search problem with n keys of length ℓ. Choose integer parameters $q \geq 2$ dividing n and $h \geq 2$ dividing ℓ. We can now reduce into subproblems, each of which is easier in one of two ways:*
>
> **length reduced** *The key length in the subproblem is reduced by a factor h to ℓ/h, and the subproblem contains at most half the keys.*
>
> **cardinality reduced** *The number of keys is reduced by a factor q to n/q.*
>
> *The reduction costs a constant in the query time. For some number m determined by the reduction, the reduction uses $O((q^{(2h)} + m)\ell)$ bits of space. The total number of keys in the cardinality reduced subproblems is at most $n - m$, and the total number of keys in the length reduced subproblems is at most m.*

In the proof below we will ignore the requirement that a length reduced subproblem should contain at most half the keys. If one of these subproblems ends up with two many keys, we can just split it around the median, adding only a constant to the search time.

We will view each key x as a vector $x_1 \cdots x_h$ of h characters, each of $c = \ell/h$ bits. We now provide an alternative to the parallel hashing in [4, Lemma 4.1]. The most significant difference is that our lemma does not require a word length that is h times bigger than ℓ. Besides, the statement is more directly tuned for our construction.

Lemma 20. *Using $O(q^{2h}\ell)$ bits of space, we can store a set $Z = \{z^1, ..., z^q\}$ of q h-character keys so that given a query key x, we can in constant time find the number of whole characters in the longest common prefix between x and any key in Z.*

Proof. Andersson et al. [3, Section 3] have shown we in constant time can apply certain universal hash functions $H_1, ..., H_h$ in parallel to the characters in a word, provided that the hash values are no bigger than the characters hashed. Thus, for each i independently, and for any two different characters $x \neq y$, if the hash values are in $[m]$, then $\Pr[H_i(x) = H_i(y)] \leq 1/m$. Given $x = x_1 \cdots x_h$, we return $H_1(x_1) \cdots H_h(x_h)$ in constant time. However, the hashed key has the same length as the original key. More precisely, if the characters have c bits and the hashed characters are in $[2^b]$, then we have $c - b$ leading zeros in the representation of $H_i(x_i)$.

We will map each character to $b = 2 \lg q$ bits. We may here assume that $b < c$, for otherwise, we can tabulate all possible keys in $q^{2h}\ell$ bits of space. For each character position i, we have q characters z_i^j, and for random H_i these are all expected to hash to different values. In particular, we can choose an H_i without collisions on $\{z_i^j\}_{1 \leq j \leq q}$. Now if $x_i = z_i^j$ we have $H_i(x_i) = H_i(z_i^j)$ and there is no $z_i^{j'} \neq z_i^j$ with $H_i(x_i) = H_i(z_i^{j'})$.

Next, consider the set A of values $H_1(x_1) \cdots H_h(x_h)$ over all possible vectors $x = x_1 \cdots x_h$. These vectors are ch long, but since only the b least significant bits are used for the hash values of each character, there are at most 2^{bh} different values in A. Using the linear space 2-level hashing of Fredman et al. [8], we construct a hash table $\mathcal{H}$ over A using $O(2^{bh}\ell)$ bits of space. With the entry $\mathcal{H}(H_1(x_1) \cdots H_h(x_h))$, we store the key z^j so that $H_1(z_1^j) \cdots H_h(z_h^j)$ has the longest possible prefix with $H_1(x_1) \cdots H_h(x_h)$. The key z^j is found from x in constant time.

We now claim that no key $z^{j'}$ can agree with x in more characters than $z^{j'}$. Suppose for a contradiction that z^j agrees with x in the first $r - 1$ characters but not in character r, and that $z^{j'}$ agrees in the first r characters. Then $H_1(x_1) \cdots H_r(x_r) = H_1(z_1^{j'}) \cdots H_r(z_r^{j'})$. However, since H_r is 1-1 on $\{z_r^j\}_{1 \leq j \leq q}$, $H_r(z_r^j) \neq H_r(z_r^{j'}) = H_r(x_r)$.

All that remains is to compute the number of whole characters in the common prefix of x and z^j. This can be done by clever use of multiplication as described in [10]. A more practical solution based on converting integers and to floating point numbers and extracting the exponent is discussed in [17]. $\square$

Using Lemma 20, we can compute in constant time the longest common prefix, $\textit{comm\_pref}_Z[x]$, in whole characters, between x and any key in Z. Also, if $x \notin Z$, we can get the prefix $\textit{comm\_pref}_Z^+[x]$ that has one more character from x.

We are now return to the proof of Proposition 18 which is similar to the one in [4, Section 4]. Out of our original set Y of n keys, we pick a subset $Z = \{z^1, ..., z^q\}$ of q keys so that there is a key from Z among any sequence of $\lceil n/q \rceil$ consecutive keys from Y. We apply Lemma 20 to Z. Thereby we use $O(q^{2h}\ell)$ bits of space. We are going to consider two types of subproblems.

Cardinality reduced problems First we have the cardinality reduced subproblems. These are of the following type: we take a key from $Y \setminus Z$ and consider the prefix $v = \mathit{comm\_pref}^+(y, Z)$. Let $\mathit{Agree}_Y[v]$ denote the keys from Y that have prefix v. These keys are consecutive and they do not contain any key from Z, so $|\mathit{Agree}_Y[v]| < q$. We use 2-level hash table for the prefixes in $V = \{\mathit{comm\_pref}^+(y, Z) \mid y \in Y \setminus Z\}$. With $v \in V$, we store $\mathit{pred}_Y[v]$ and $\mathit{max}_Y[v]$ as defined in the previous section, that is, $\mathit{pred}_Y[v]$ is the strict predecessor in Y of v suffixed by zeros, and $\mathit{max}_Y[v]$ is the largest key in Y with prefix v. Finally, as the cardinality reduced subproblem, we have $\mathit{Agree}^-_Y[v] = \mathit{Agree}_Y[v] \setminus \{\max \mathit{Agree}_Y[v]\}$.

This above informatoin suffices to find the predecessor of any query key x with $\mathit{comm\_pref}^+(x, Z) = v$. The bit space used above is $O(|V|\ell)$. Each cardinality reduced subproblem $\mathit{Agree}^-_Y[v]|$ has at most $q - 2$ keys, and they add up to a total of $n - |Z| - |V|$ keys.

Length reduced subproblems For query keys x with $\mathit{comm\_pref}^+(x, Z) \notin V$, we will consult length reduced subproblems defined over the set U of prefixes of keys in Z. We will have a 2-level hash table over U. For each $u \in U$, let $\mathit{Next\_char}_Y[u]$ be the set of characters c such that uc is a prefix of a key in Y. We will have a length reduced subproblem over the characters in $\mathit{Next\_char}^-_Y[u] = \mathit{Next\_char}_Y[u] \setminus \{\max \mathit{Next\_char}^-_Y[u]\}$. As complimentary information, we store $\mathit{pred}_Y[v]$ and $\mathit{max}_Y[v]$.

Now, consider a query key x with $\mathit{comm\_pref}^+(x, Z) \notin V$. Let $u = \mathit{comm\_pref}(x, Z)$ and let d be the subsequent character in x, that is, $ud = \mathit{comm\_pref}^+(x, Z)$. Then $d \notin \mathit{Next\_char}_Y[u]$. Suppose x is between the smallest and the largest key in Y with prefix u. If c is the predecessor of d in $\mathit{Next\_char}^-_Y[u]$, then the predecessor of x in Y is the largest key with prefix uc. However, $uc \in V$, so the predecessor of x is the $\mathit{max}_Y[uc]$ stored under the length reduced subproblems.

The above length reduction used $O(\ell)$ bits for each $u \in U$ and $c \in \mathit{Next\_char}_Y[u]$. Consider $c \in \mathit{Next\_char}_Y[u]$. There can be at most U cases where $uc \in U$. Otherwise, we have $uc = \mathit{comm\_pref}^+(y, Z) \in V$. The total bit space of the length reduction is hence $O((|U| + |V|)\ell)$.

We will now prove that the total number of keys in the length reduced subproblems $\mathit{Next\_char}^-_Y[u]$ is at most $|Z| + |V|$. Above we saw that if a character $c \in \mathit{Next\_char}_Y[u]$ did not represented a prefix in U, it represented a prefix in V. Those representing prefixes in U can also be viewed as representing children in the trie over Z. The total number of such children is at most $|Z|$ plus the number of internal trie nodes, and since we for $\mathit{Next\_char}^-_Y[u]$ subtracted a node for each internal trie node u, we conclude that the total number of keys is the length reduced subproblems is bounded by $|Z| + |V|$.

Pseudo-code We now have the following recursive pseudo-code for searching the predecessor of x in Y:

$\mathit{Pred}(x, Y)$
  if $x \in Z$ return x
  let $ud = \mathit{comm\_pref}^+_Z(x)$ with d the last character
  if $ud \in V$ then
    if $x \geq \mathit{max}_Y[ud]$ then return $\mathit{max}_Y[ud]$)
    $y = \mathit{Pred}(x, \mathit{Agree}^-_Y[uc])$
    if $y = -\infty$ then return $\mathit{pred}^{\neq}_Y[ud]$
    return y
  if $x \geq \mathit{max}_Y[u]$ then return $\mathit{max}_Y[u]$

$c = \mathit{Pred}_Y(d, \mathit{Next\_char}^-_Y[u])$
if $c = -\infty$ then return $\mathit{pred}^{\neq}_Y[u]$
return $\mathit{max}_Y[uc]$

Final analysis This almost finishes the proof. Let $m = |Z| + |V|$. Then we have at most $n - m$ keys in cardinality reduced subproblems and at most m keys in length reduced subproblems.

The total bit space used is $O(q^{2h}\ell)$ for the implication of Lemma 20, $O(|V|\ell)$ for the cardinality reduction, and $O((|U| + |V|)\ell)$ for the length reduction. Here $O(|U|) = O(hq) = O(q^{2h})$ and $|V| < m$, so the total bit space is $O((q^{2h} + m)\ell)$. **This completes the proof of Proposition 18.** $\square$

References

[1] Miklós Ajtai. A lower bound for finding predecessors in Yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988.

[2] Noga Alon and Joel Spencer. *The Probabilistic Method.* John Wiley, 2nd edition, 2000.

[3] Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. Sorting in linear time? *Journal of Computer and System Sciences*, 57(1):74–93, 1998. See also STOC'95.

[4] Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002. See also STOC'99.

[5] Amit Chakrabarti and Oded Regev. An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In *Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 473–482, 2004.

[6] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *Proc. ACM SIGCOMM*, pages 3–14, 1997.

[7] Anja Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *Proc. IEEE INFOCOM*, pages 1193–1202, 2000.

[8] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. See also FOCS'82.

[9] Michael L. Fredman and Michael E. Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989.

[10] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. See also STOC'90.

[11] Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. In *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 332–344, 2003.

[12] Peter Bro Miltersen. The bit probe complexity measure revisited. In *10th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 662–671, 1993.

[13] Peter Bro Miltersen. Lower bounds for Union-Split-Find related problems on random access machines. In *26th ACM Symposium on Theory of Computing (STOC)*, pages 625–634, 1994.

[14] Peter Bro Miltersen. Cell probe complexity - a survey. In *19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 1999. Advances in Data Structures Workshop.

[15] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998. See also STOC'95.

[16] Pranab Sen and S. Venkatesh. Lower bounds for predecessor searching in the cell probe model. *arXiv:cs.CC/0309033*. See also ICALP'01, CCC'03, 2003.

[17] Mikkel Thorup. On RAM priority queues. *SIAM Journal on Computing*, 30(1):86–109, 2000. See also SODA'96.

[18] Mikkel Thorup. Space efficient dynamic stabbing with fast queries. In *Proc. 35th ACM Symposium on Theory of Computing (STOC)*, pages 649–658, 2003.

[19] Peter van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977. Announced by van Emde Boas alone at FOCS'75.

[20] Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983.

[21] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981. See also FOCS'78.