The Batched Predecessor Problem in External Memory

Michael A. Bender[1 2], Martin Farach-Colton[2 3], Mayank Goswami[4],
Dzejla Medjedovic[1], Pablo Montes[1], and Meng-Tsung Tsai[3]

[1] Stony Brook University, Stony Brook NY 11794, USA
{bender, dmededovic, pmontes}@cs.stonybrook.edu
[2] Tokutek, Inc.
[3] Rutgers University, Piscataway NJ 08855, USA
{farach, mtsung.tsai}@cs.rutgers.edu
[4] Max-Planck Institute for Informatics, Saarbrücken 66123, Germany
gmayank@mpi-inf.mpg.de

Abstract. We give lower bounds for the batched predecessor problem in external memory. Given the underlying *sorted* set S of size n, and a *sorted* query Q of size $x \leq n^c$, $0 \leq c < 1$, we study tradeoffs between the searching cost, and the cost to preprocess S. We give lower bounds in three external memory models: the I/O comparison model, I/O pointer-machine model, and the indexability model.

Our results show that in the comparison I/O model, the batched predecessor problem needs $\Omega(\log_B n + 1/B)$ I/Os per element if the preprocessing is bounded by a polynomial; however, with exponential preprocessing, the problem can be solved faster, in $\Theta((\log_2 n + 1)/B)$. We also present the tradeoff that quantifies the minimum preprocessing required for a given searching cost.

In the pointer-machine model, we show that with $O(n^{4/3-\varepsilon})$ preprocessing for any ε bounded above 0 by a constant, the optimal algorithm cannot perform asymptotically faster than a B-tree. In the more general indexability model, we exhibit the tradeoff between the redundancy r and access overhead α of the optimal indexing scheme, showing that to report all query answers in $\alpha(x/B)$ I/Os, $r = (n/B)^{\Omega(B/\alpha^2)}$.

We also devise algorithms and indexing schemes that either always match or nearly match the stated lower bounds.

1 Introduction

A ***static dictionary*** is a data structure to represent a sorted set $S = \{s_1, s_2, \ldots, s_n\}$ subject to the following operations:

SEARCH(q, S):	Return TRUE if $q \in S$ and FALSE otherwise.
PREDECESSOR(q, S):	Return $\max_{s_i \in S}\{s_i < q\}$.

The traditional static dictionary can be extended to support batched operations. Let $Q = \{q_1, \ldots, q_x\}$, where Q is sorted. Then, the ***batched predecessor*** problem can be defined as follows:

BATCHEDPRED(Q, S):	Return $A = \{a_1, \ldots, a_x\}$, where $a_i =$ PREDECESSOR(q_i, S).

In this paper, we prove lower bounds on the batched predecessor problem in ***external memory*** [1], that is, when the dictionary is too large to fit into main memory. We study tradeoffs between the searching cost, and the cost to preprocess the underlying set S. We present our results in three models: the comparison-based I/O model [1], the pointer-machine I/O model [20], and the indexability model [12, 13].

We focus on query size $x \leq n^c$, for constant $c < 1$. Thus, the query Q can be large, but is still much smaller than the underlying set S. This query size is interesting because Q is small enough that scanning S is inefficient, and so are buffering techniques [2,7,8] (due to expensive flushes at the end). On the other hand, Q is too large for the B-tree to be an obvious choice.

For these values of x, the cost to sort Q is subsumed by the query cost. The same does not always hold for the cost to sort S, but we can consider sorting S as the part of the preprocessing phase. This scenario is consistent with a semi-offline version of the problem, where the set S is static but queries arrive in batches.

Surprisingly, our results show that the batched predecessor problem in external memory cannot be solved asymptotically faster than $\Omega(\log_B n + 1/B)$ I/Os per element if the preprocessing is bounded by a polynomial. The common wisdom is that one uses buffer trees to deal with batched queries or inserts (see e.g., [4,23]), but our bounds imply that buffer trees may not work efficiently in this setting. We also show that the problem *can* be solved asymptotically faster, in $\Theta((\log_2 n + 1)/B)$ I/Os if we impose no constraints on preprocessing. This stands in contrast to single predecessor queries, where one search costs $\Omega(\log_B n)$ even if preprocessing is unlimited.

Single and batched predecessor problem in RAM. In the comparison model, a single predecessor can be found in $\Theta(\log n)$ using binary search. The batched predecessor problem is solved in $\Theta(x \log(n/x) + x)$ by combining merging and binary search [6, 15, 16]. The bounds for both problems remain tight for any preprocessing budget.

Pătraşcu and Thorup [17] give tight lower bounds for single predecessor queries in the cell-probe model. We are unaware of prior lower bounds for the batched predecessor problem in the pointer-machine and cell-probe models.

Although batching does not help algorithms that rely on comparisons, Karpinski and Nekrich [14] give an upper bound for this problem in the word-RAM model (bit operations are allowed), which achieves $O(x)$ for all batches of size $x = O(\sqrt{\log n})$ ($O(1)$ per element amortized) with superpolynomial preprocessing.

Batched predecessor problem in external memory. Dittrich et al. [9] consider multisearch problems where queries are simultaneously processed and satisfied by navigating through large data structures on parallel computers. They give a lower bound of $\Omega(x \log_B(n/x) + x/B)$, but under stronger assumptions: no duplicates of nodes are allowed, the ith query has to finish before the $(i+1)$st query starts, and $x < n^{1/(2+\varepsilon)}$, for a constant $\varepsilon > 0$. They do not study the tradeoffs between preprocessing and queries.

Several data structures have been proposed that take advantage of the large block size to buffer insertions in order to amortize their cost [2, 7, 8]. In all of these, queries can also be buffered and lazily pushed down the tree if we allow the data structures to answer queries at a later time. How long it takes for queries to trickle down the tree depends on the size of the query set. It is possible, therefore, that if the query set is not

large enough, too many queries will get stuck in the buffers, and then flushing them is expensive, as each query has to at least finish their walk down a root-to-leaf path in order to find the answer.

Goodrich et al. [11] present a general method for performing x simultaneous external memory searches in $O((n/B + x/B)\log_{M/B}(n/B))$ I/Os when x is large. When x is small his technique achieves $O(x\log_B(N/B))$ I/Os with a modified version of the parallel fractional cascading technique of Tamassia and Vitter [21].

Results

We first consider the **comparison-based I/O model** [1]. In this model, the problem can not be solved faster than $\Omega(\log_B n + 1/B)$ I/Os per element if preprocessing is polynomial. That is, batching queries is not faster than processing them one by one. With exponential preprocessing, the problem can be solved faster, in $\Theta((\log_2 n+1)/B)$ I/Os per element. We generalize to show the tradeoff a query-preprocessing tradeoff.

Next we study the **pointer-machine I/O model** [20], which is less restrictive than the comparison I/O model in main memory, but more restrictive in external memory.[5] We show that with preprocessing at most $O(n^{4/3-\varepsilon})$ for constant $\varepsilon > 0$, the cost per element is again $\Omega(\log_B n)$.

Finally, we turn to the more general **indexability model** [12, 13]. This model is frequently used to describe reporting problems, and it focuses on bounding the number of disk blocks that contain the answers to the query subject to the space limit of the data structure; the searching cost is ignored. Here, the ***redundancy*** parameter r measures the number of times an element is stored in the data structure, and the ***access overhead*** parameter α captures how far the reporting cost is from the optimal.

We show that to report all query answers in $\alpha(x/B)$ I/Os, $r = (n/B)^{\Omega(B/\alpha^2)}$. Note that the lower bounds in this model also hold in the previous two models. This result shows that it is impossible to obtain $O(1/B)$ per element unless the space used by the data structure is exponential, which corresponds to the situation in RAM, where exponential preprocessing is required to achieve $O(1)$ per element amortized [14].

The rest of this section formally outlines our results.

Theorem 1 (Lower bound, unrestricted preprocessing, comparison I/O model). *Solving the batched predecessor problem for a query set of size x on an underlying set of size n of size requires*

$$\Omega\left(\frac{x}{B}\log\frac{n}{xB} + \frac{x}{B}\right)$$

I/Os in the worst-case.

We obtain the lower bound by demonstrating that an algorithm can learn up to B bits per I/O. This is in contrast to $\log B$ that an algorithm learns from a block of a B-tree, and also in contrast to $\Theta(B\log(M/B))$ bits per I/O, bound associated with buffer tree performance, which we show is not achievable.

We give an algorithm that matches the bound of Theorem 1: the algorithm processes the query in batches of size B, by performing binary search on B elements at once.

[5] An algorithm can perform arbitrary computations in RAM, but a disk block can be accessed only via a pointer that has been seen at some point in past.

Theorem 2 (Search-preprocessing tradeoff, comparison I/O model). *Given a batch of size x and the underlying set S of size n, where $x \leq n^c$, $0 \leq c < 1$, solving the batched predecessor problem in $O((x \log_{B/j+1} n)/j)$ I/Os requires $\Omega(n^{\Omega(j)})$ I/Os for preprocessing.*

This tradeoff captures the notion that to learn up to $j \log(B/j + 1)$ bits in one I/O, the algorithm needs to spend $\Omega(n^{\Omega(j)})$ in preprocessing (substituting j by 1 and B yields two ends of the spectrum.) This shows that even to obtain a performance that is only twice as fast as that of a B-tree, quadratic preprocessing is necessary.

Theorem 3 (Lower bound, pointer-machine I/O model). *Assume that the graph $\mathcal{G} = (V, E)$ corresponding to a data structure for the batched predecessor problem satisfies $|V| = O(n^{4/3-\varepsilon})$ for any ε bounded above 0 by a constant. Then the data structure needs to visit $\Omega(x \log_B(n/x)+x/B)$ nodes in order to answer predecessors of elements in Q in the worst case.*

Theorem 4 ($r - \alpha$ tradeoff, indexability model). *Any indexing scheme for the batched predecessor problem with access overhead $\alpha \leq \sqrt{B}/4$ has redundancy r satisfying*

$$\log r = \Omega\left(\frac{B}{\alpha^2} \log \frac{n}{B}\right).$$

2 Batched Predecessor in the I/O Comparison Model

This section analyzes the batched predecessor problem in the I/O comparison model. We first give two lower bounds for the problem if preprocessing is unrestricted, and then study the tradeoff between preprocessing and the optimal number of memory transfers.

2.1 Lower Bounds

The first lower bound is derived using standard information-theoretic techniques used to prove comparison-based lower bounds [1, 10]. In this argument, we analyze the number of ways to interleave two sorted lists of sizes x and n, and the number of interleavings that can be eliminated in one I/O (proof in Appendix A).

Lemma 1. *Any algorithm that solves* BATCHEDPRED(Q, S) *requires $\Omega\left(\frac{x}{B} \log_{M/B} \frac{n}{x} + \frac{x}{B}\right)$ I/Os in the worst case.*

The same lower bound can be derived by relating the number of comparisons needed to solve the batched predecessor problem in RAM with the number of I/Os required to achieve this many comparisons, as described in [3].

Even though the standard information-theoretic argument gives a tight bound in RAM, in external memory it is possible to obtain a stronger lower bound which is optimal assuming unrestricted preprocessing.

One can prove that searching for elements of Q one by one on a B-tree built on S is asymptotically optimal when x is very small (e.g., $x = O(1)$). Similarly, a simple scan-and-merge algorithm is optimal when Q and S are roughly equal in size (e.g., when

$x = \Theta(n)$) as one lower bound for the problem is x/B. Therefore, there exist few cases when the lower bound given by Lemma 1 is tight, but it is not tight in general, as we show in Theorem 1.

Definition 1 (*Search interval*). *We define the search interval $S_i = [s_i, s_j]$ for an element q_i as the narrowest interval where q_i belongs in the final sorted order, given the information learnt by the algorithm at a given point.*

Proof of Theorem 1. Assume that the following information is given for free:

1. For all q_i, $|S_i| = n/x$. That is, all elements in Q have been given the first $\log x$ bits of information about where they belong in S.
2. For all i and j ($1 \leq i \neq j \leq x$), $S_i \cap S_j = \emptyset$. That is, all search intervals are disjoint.

We assume that the adversary transfers all elements of Q between main memory and disk for free. This is equivalent to allowing all elements of Q to reside in main memory at all times while still having the entire memory free for other manipulations. Storing Q in main memory does not require the release of any additional information, as the sorted order of Q is already known.

Now we only consider the inputs of S. Denote a block being input as $b = (b_1, \ldots, b_B)$. Observe that, for a given b_i ($1 \leq i \leq B$), there can be at most one element $q_j \in Q$ such that $b_i \in S_j$. The element b_i acts as a **pivot** and helps q_j learn at most one bit of information—by shrinking S_j to its left or its right half.

Given that a single pivot can give at most one bit of information, the entire block b can give at most B bits to the elements of Q. After a block gives B bits, it becomes useless for the rest of the algorithm.

We require the algorithm to identify the final block in S where each q_i belongs. Thus, the total number of bits that the algorithm needs to learn to solve the problem is $\Omega(x \log(n/xB))$. Therefore, along with the scan bound to output the answer, the minimum number of block transfers required to solve the problem is $\Omega\left(\frac{x}{B} \log \frac{n}{xB} + \frac{x}{B}\right)$. □

We devise a matching algorithm (assuming $B \log n < M$), which has $O(n^B)$ preprocessing cost. This algorithm is not practical due to large preprocessing costs, but gives evidence that that the lower bound from Theorem 1 is tight.

Optimal Algorithm. The algorithm processes Q in batches of size B, one batch at a time. A single batch is processed by simultaneously performing binary search on all elements of the batch until they find their final positions in n.
PREPROCESSING: The algorithm produces all $\binom{n}{B}$ possible blocks. The algorithm also preprocesses a perfectly balanced binary search tree T on S. Note that the former takes at most $B\binom{n}{B}$ I/Os, which is $O(n^B)$, while the latter has a linear cost. The $\binom{n}{B}$ blocks are laid out in a lexicographical order in external memory, and it takes $B \log n$ bits to address the location of any block.
QUERYING: The algorithm has the following steps:

– Do a linear merge of Q and ℓth level of T, where $\ell = \lceil \log x \rceil$.

– For each batch $C_i = \{q_{iB+1}, q_{iB+2}, \ldots, q_{(i+1)B}\}$, set $j = iB + 1$, input the batch C_i, and do the following:
  - Bring in a block $b = (p_j, p_{j+1}, \ldots, p_{j+B-1})$, where p_k is a median of S_k(the current search interval of q_k). In other words, for each element of the batch, there is one pivot in the block that is a median of its search interval. Compare each $p_j \in b$ with q_j and after the comparison, adjust (shrink) S_j.
  - Use a string of B bits to represent whether the B elements in a batch go into the left/right half of their respective search intervals. Use this string to decide which block to bring in next.
  - Repeat the steps above until every element's search interval is of size at most B.

2.2 Preprocessing-Searching Tradeoffs

In this section we give a lower bound on the batched predecessor problem with restrictions on the number of I/Os allowed in preprocessing.

We prove Theorem 2 by proving Theorem 5:

Definition 2 (*j-parallelization I/O*). *A block transfer of a block b_i that contains elements of S is a j-parallelization I/O (or a j-parallelized I/O) if during this block transfer j distinct elements of Q acquire bits of information.*

Theorem 5. *For $x \leq n^{1-\varepsilon}$ ($0 < \varepsilon \leq 1$), any algorithm that solves* BATCHEDPRED(Q, S) *in at most $(cx \log n)/(j \log(B/j + 1)) + x/B$ I/Os in the worst case requires at least $\left(\varepsilon j n^{1/2}/2cB\right)^{\varepsilon j/2c}$ I/Os in preprocessing.*

Proof Sketch. The proof is by a deterministic adversary argument. In the beginning, the adversary partitions S into x equal-sized chunks $C_1, \ldots, C_x$, and places each query element into a separate chunk (i.e., $S_i = C_i$). This way each element learns $\log x \leq (1 - \varepsilon) \log n$ bits of information. As before, the adversary gives the inputs of elements in Q for free, thereby stipulating that all bits must be learned through inputs of elements of S. Then each element is additionally given half of the number of bits that remain to be learned. This leaves us with another $T \geq (\varepsilon x \log n)/2$ total bits yet to be discovered.

Disregarding the additional linear term (which we can do provided that $n \geq 2^{1/c}$, where c is a positive constant), to learn T bits in at most $(cx \log n)/(j \log(B/j + 1))$ I/Os, there must be at least one I/O in which the algorithm learns at least $(j \log(B/j + 1))/a$ bits, where $a = 2/\varepsilon$. If there are multiple I/Os with these properties, we only consider the last such I/O during the algorithm runtime. Denote the contents of the block as $b_i = (p_1, \ldots, p_B)$. Proofs for the lemmas and observations below can be found in Appendix A.

Observation 6 *The maximum number of bits an I/O can learn while parallelizing g elements is $g \log(B/g + 1)$.*

Lemma 2. *The I/O b_i parallelizes at most j/a elements.*

We focus our attention on an arbitrarily chosen group of such j/a elements which, without loss of generality, we call $q_1, \ldots, q_{j/a}$.

Observation 7 *For each element q_u that learns new information during the transfer of b_i, there must exist at least one pivot $p_v \in b_i$, such that $p_v \in S_u$.*

Consider the vector $V = (S_1, S_2, \ldots, S_{j/a})$ where the ith position denotes the search interval of q_i right before the input of b_i.

Thus far, each element of Q has acquired at least half of the total number of bits it needs to learn. Having learned at least $(\log n)/2$ bits per element, the total number of distinct choices for any position in the vector is at least $n^{1/2}$. Thus, we have the following observation:

Observation 8 *The total number of distinct choices for V at the time of parallelization is at least $C = n^{j/2a}$.*

Observation 9 *For each choice of a vector, there must exist a (not necessarily distinct) block with pivots $p_{i,1}, p_{i,2}, \ldots, p_{i,j/a}$, such that $p_{i,k} \in S_k$.*

Lemma 3. *A manufactured block can cover at most $(aB/j)^{j/a}$ distinct vector choices.*

As a consequence, the minimum number of blocks the algorithm needs to manufacture is at least $n^{j/2a}/(aB/j)^{j/a} = \left(n^{1/2}/aB/j\right)^{j/a}$. Substituting for the value of a, we get that the minimum preprocessing is at least $\left(\varepsilon j n^{1/2}/2cB\right)^{\varepsilon j/2c}$.

Algorithms. An algorithm that runs in $O((x \log n)/j \log(B/j+1)+x/B)$ I/Os follows an idea similar to the optimal algorithm for unrestricted preprocessing. The difference is that we preprocess $\binom{n}{j}$ blocks, where each block correspond to a distinct combination of some j elements. The block will contain B/j evenly spaced pivots for each element. The searching algorithm uses batches of size j.

3 Batched Predecessor in the I/O Pointer-Machine Model

The upper bounds in the comparison model show that unless we somehow restrict the choices for the next block to be brought into main memory, the lower bound cannot be improved. In contrast to the comparison model, in the external pointer machine model there are limits to the number of next blocks a data structure can read, although it is allowed any kind of computation in RAM. Thus it is natural to analyze the batched predecessor problem in the I/O pointer-machine model.

We show that if the preprocessing time is $\mathcal{O}(n^{4/3-\varepsilon})$ for any ε bounded above 0 by a constant, then there exists a query set Q of size x such that reporting BATCHED-PRED(Q, S) requires $\Omega(x/B + x \log_B n/x)$ I/Os. Before proving our theorem, we give a brief description of the model.

External pointer machine model. The external pointer machine model [20] is a generalization of the pointer machine model introduced by Tarjan, and many results in range reporting have been obtained in this model. In this model the data structure is

modeled as a directed graph $\mathcal{G} = (V, E)$. The node set V represents the set of blocks; each block contains at most B elements and at most B pointers to other blocks. This graph is formed by the data structure during the preprocessing phase. Given a query, the data structure starts at any node $v \in V$, and follows a pointer contained in the block corresponding to v. At any stage, after having accessed nodes $v_1, ...v_{t-1}$, at time t the data structure can follow any pointer seen so far; i.e., a pointer contained in some v_i, $1 \leq i \leq t-1$. Note that this gives the data structure more power, as opposed to allowing it to only follow pointers in main memory or from the last visited node.

The choice of the next block can be any function of the elements in the blocks accessed so far. In order to answer the query(which is a subset of the input set), the data structure must visit a set of blocks containing all the output elements.

Proof Sketch. We now give a brief overview of our proof; complete proofs of the lemmas and the theorem are placed in the appendix.

The algorithm preprocesses S and builds a data structure comprised of n^k blocks, where k is a constant to be determined later. We use the directed graph $G = (V, E)$ to represent the n^k blocks and their associated directed pointers. Every algorithm that answers BATCHEDPRED(Q, S) begins at the start node a in V, walks along the directed edges in E, and ends when the elements in the visited nodes form a superset of the answer set A. Note that the start node a is in general different for different instances of the problem, and is chosen by the data structure.

To prove a lower bound on BATCHEDPRED(Q, S), we will 1) pick a suitable A(corresponding to the query Q) and 2)show that every subset $W \subseteq V$ where the set of elements in W is a superset of A needs to be connected by a large subgraph. Any data structure has to therefore traverse one of these large subgraphs for the chosen A, giving us our lower bound.

We define the notion of distances before proceeding to the proof. In a single run of the data structure, we only focus on those nodes in the graph that are reachable from the start node a, as the other nodes cannot help the data structure in any way. In what follows, H will denote a graph that is star-shaped with respect to a node a. Let $d_H(u, v)$ be the length of the shortest path from node u to node v in H. In this notation, the graph H and the path within the graph can be directed or undirected. Furthermore, let

$$\Lambda_H(u, v) = \min_{w \in V} \left(d_H(w, u) + d_H(w, v) \right). \tag{1}$$

Thus, $\Lambda_H(u, v) = d_H(u, v)$ for undirected graphs, but not for directed graphs.

Let $H = (V, E)$ be a directed graph that is star shaped with respect to a. For each $W \subseteq V$, define $f_H(W)$ to be the minimum number of nodes in any connected subgraph $\tilde{H}$ such that 1) the node set of $\tilde{H}$ contains $W \cup \{a\}$ and 2) $\tilde{H}$ contains a path from a to each $v \in W$.

Observe that $f_H(\{u, v\}) \geq \Lambda_H(u, v)$ (The two quantities may not be the same because $\tilde{H}$ must contain a, which may not be equal to w, above.)

The following lemma gives a more general lower bound for $f_H(W)$.

Lemma 4. *For any directed graph $G = (V, E)$ and any $W \subseteq V$ of size $|W| \geq 2$, $f_G(W) \geq r|W|/2$, where $r_W = \min_{u,v \in W, u \neq v} \Lambda_G(u, v)$.*

In our setting, we will try to find a query set Q such that any superset W containing Q has a large r_W. The query set Q will be an independent set of a certain kind, that we define next. For a directed graph $G = (V, E)$ and an integer $r > 0$, we say that a set of nodes $I \subseteq V$ is ***r-independent*** if $\Lambda_G(u, v) > r$ for all $u, v \in I$ where $u \neq v$. The next lemma(similar to the famous theorem by Turán on sizes of independent sets) guarantees a substantial r-independent set.

Lemma 5. *Given a directed graph $G = (V, E)$, where each node has out-degree at most $B \geq 2$, there exists an r-independent set I of size at least $\frac{|V|^2}{|V|+4r|V|B^r}$.*

Apart from r-independence, we want another condition on our query set Q; namely that we do not want elements of Q to occur in too many blocks, in order to control the possible choices of the explored set W that contains Q. We define the ***frequency*** of an element e to be the number of times that e appears, summed over all n^k blocks. Because there are n^k blocks and each block has at most B elements, the average frequency is $\mathcal{O}(n^{k-1}B)$. We say that an element has ***low frequency*** if its frequency is at most twice the average. We show that there exists an r-independent set $\mathcal{I}$ of size n^{ε}(here ε depends on r) such that no two blocks(corresponding to two nodes in this set) share the same low-frequency element. We will then construct our query set Q using this set of of low frequency elements in this r-independent set. (Our construction does not work if the query set contains high frequency elements, because high frequency elements might be placed in every block.)

We next show that if two blocks share the same low-frequency element, then they are at most ρ-apart, for some $\rho < r/2$. In the proof, we need to add some additional edge to closely connect the blocks that share the same low-frequency element. Lemma 6 is used to show how to connect them with small number of edges.

Lemma 6. *For any $k > 0$ and $m > k$ there exists an undirected k-regular graph H of size m having diameter $\log_{k-1} m + o(\log_{k-1} m)$.*

Consider two blocks B_1 and B_2 in the r-independent set $\mathcal{I}$ above, and let a and b be two low-frequency elements such that $a \in B_1, b \notin B_1$ and $a \notin B_2, b \in B_2$. Any other pair of blocks $B_1^{'}$ and $B_2^{'}$ that contain a and b respectively must be at least $(r - 2\rho)$-apart, since $B_i^{'}$ is at most ρ-apart from B_i. By this argument, every working set W whose elements comprise a superset of Q is $(r - 2\rho)$-apart. Now by Lemma 4, we get a lower bound of $O((r - 2\rho)|W|)$ on the query complexity of Q. We choose $r = c_1 \log_B(n/x)$ and get $\rho = c_2 \log_B(n/x)$ for appropriate constants $c_1 > 2c_2$(this is the part where we require the assumption that $k < 4/3$, where n^k was the size of the entire data structure). We then apply Lemma 5 to obtain that $|W| = \Omega(x)$.

Theorem 10. *Consider any data structure for* BATCHEDPRED(Q, S) *of size $O(n^k)$, where $k = 4/3 - \varepsilon$ for any ε bounded above 0 by a constant. There exists a query set Q of size x such that* BATCHEDPRED(Q, S) *takes $\Omega(x/B + x \log_B n/x)$ I/Os.*

4 Batched Predecessor in the Indexability Model

This section analyzes the batched predecessor problem in the indexability model [12, 13].

The indexability model is used to describe reporting problems by focusing on the number of blocks that an algorithm must access to report all the query results. Lower bounds on the query time are obtained solely based on how many blocks were preprocessed, and the search cost is ignored—the blocks containing the answers are told to the algorithm for free.

A ***workload*** is given by a pair $\mathcal{W} = (S, Q)$, where S is the set of n input objects, and Q is a set of subsets of S—the (output to the) queries. An ***indexing scheme*** for a given workload $\mathcal{W}$ is given by a collection $\mathcal{B}$ of B-sized subsets of S such that $S = \cup\mathcal{B}$; each $b \in \mathcal{B}$ is called a block.

An indexing scheme has two parameters associated to it. The first parameter, called the ***redundancy*** represents the average number of times an element is replicated (i.e., an indexing scheme with redundancy r uses $r\lceil n/B \rceil$ blocks). The second parameter is called the ***access overhead***. Given a query Q, the query time is $\min\{|\mathcal{B}^{'}| : \mathcal{B}^{'} \subset \mathcal{B}, Q \subset \cup\mathcal{B}^{'}\}$, as this is the minimum number of blocks that contain all the answers to the query. If the size of Q is x, then the best indexing scheme would require a query time of $\lceil x/B \rceil$. The access overhead of an indexing scheme is the maximum ratio of the query time of the indexing scheme and this "ideal" query time. In other words, an indexing scheme with access overhead a requires $a\lceil x/B \rceil$ block reads to answer a query of size x in the worst case.

Every lower bound in this model applies to our previous two models as well. To show the tradeoff between a and r, we will use the Redundancy Theorem from [13,19], which we restate here for completeness:

Theorem 11 (Redundancy Theorem [13, 19]). *For a workload $\mathcal{W} = (S, Q)$ where $Q = \{q_1, \cdots, q_x\}$, let $\mathcal{I}$ be an indexing scheme with access overhead $A \leq \sqrt{B}/4$ such that for any $1 \leq i, j \leq x$, $i \neq j$, $|q_i| \geq B/2$ and $|q_i \cap q_j| \leq B/(16A^2)$. Then the redundancy of $\mathcal{I}$ is bounded by $r \geq \frac{1}{12n}\sum_{i=1}^{x} |q_i|$.*

Next we prove Theorem 4. Recall that Theorem 4 claims that any indexing scheme for the batched predecessor problem with access overhead $\alpha \leq \sqrt{B}/4$ has redundancy r satisfying $\log r = \Omega\left(\frac{B}{\alpha^2} \log \frac{n}{B}\right)$.

Proof of Theorem 4. A query in the batched predecessor problem corresponds to a set of x locations in the set S. Although the locations can occur with multiplicity more than 1(same predecessors), for the sake of a lower bound we assume that all the x predecessors, and hence these x locations, are distinct. To use the redundancy theorem, we want to create as many queries as possible.

Call a family of k-element subsets of S β-sparse if any two members of the family intersect in less than β elements. The size $C(n, k, \beta)$ of a maximal β-sparse family is crucial to our analysis. For a fixed k and β this was conjectured to be asymptotically equal to $\binom{n}{\beta}/\binom{k}{\beta}$ by P. Erdös and J. Hanani and was later proven by V. Rödl in [18]. Thus, for large enough n, $C(n, k, \beta) = \Omega(\binom{n}{\beta}/\binom{k}{\beta})$. Note that $C(n, k, \beta)$ is never greater than this ratio, as there are at most $\binom{n}{k}$ sets of size k and a set of size k contains at least $\binom{k}{\beta}$ subsets of size β.

We now pick a $k = (B/2)$-element, $B/(16\alpha^2)$-sparse family of S, where α is the access overhead of $\mathcal{I}$. The result in [18] gives us that

$$C\left(n, \frac{B}{2}, \frac{B}{16\alpha^2}\right) = \Omega\left(\binom{n}{B/(16\alpha^2)} / \binom{B/2}{B/(16\alpha^2)}\right).$$

Thus, there are at least $(n/B)^{B/(16\alpha^2)}$ subsets of size $B/2$ such that any pair intersects in at most $B/(16\alpha^2)$ elements. The Redundancy Theorem then implies that the redundancy r is greater than or equal to $(n/B)^{\Omega(B/\alpha^2)}$, completing the proof. □

We next describe an indexing scheme that is off from the above lower bound by a factor α.

Theorem 12 (Indexing scheme for the batched predecessor problem). *Given any $\alpha \leq \sqrt{B}$, there exists an indexing scheme $\mathcal{I}_\alpha$ for the batched predecessor problem with access overhead α^2 and redundancy $r = O((n/B)^{B/\alpha^2})$*

Proof. Call a family of k-element subsets of S β-dense if any subset of S of size β is contained in at least one member from this family. Let $c(n, k, \beta)$ denote the minimum number of elements of such a β-dense family. In [18] it was proved that for a fixed k and β,

$$\lim_{n\to\infty} c(n, k, \beta)\binom{k}{\beta}\binom{n}{\beta}^{-1} = 1,$$

and thus, for large enough n, $c(n, k, \beta) = O(\binom{n}{\beta}/\binom{k}{\beta})$.

The indexing scheme $\mathcal{I}_\alpha$ consists of all sets in a B-element, (B/α^2)-dense family. By the above, the size of $\mathcal{I}_\alpha$ is $O((n/B)^{B/\alpha^2})$.

Given a query $Q = \{q_1, \cdots, q_x\}$ of size x, fix $1 \leq i < \lceil x/B \rceil$ and consider the B-element sets $C_i = \{q_{iB}, \cdots, q_{(i+1)B}\}$ (note that $C_{\lfloor x/B \rfloor}$ may have less than B elements). Since we are only designing an indexing scheme, we are told all the blocks in $\mathcal{I}_\alpha$ that contain predecessors of elements in C_i. Let $P_i = \{p_{iB}, \cdots, p_{(i+1)B}\}$ be the set of predecessors, which is part of the output we need to produce. By construction, there exists a block in $\mathcal{I}_\alpha$ that contains a $1/\alpha^2$ fraction of P_i. Thus in at most α^2 I/Os we can output P_i, by reporting B/α^2 elements in every I/O. The total number of I/Os needed to answer the entire query Q is thus $\alpha^2\lceil x/B \rceil$, which proves the theorem.

References

1. Aggarwal, A., Vitter, Jeffrey, S.: The input/output complexity of sorting and related problems. Commun. ACM 31, 1116–1127 (September 1988)
2. Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica 37(1), 1–24 (2003)
3. Arge, L., Knudsen, M., Larsen, K.: A general lower bound on the I/O-complexity of comparison-based algorithms. In: In Proc. Workshop on Algorithms and Data Structures (WADS), LNCS 709. pp. 83–94. Springer-Verlag (1993)
4. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.S.: Theory and practice of I/O-efficient algorithms for multidimensional batched searching problems. In: Proc. 9th Annual ACM-SIAM Aymposium on Discrete Algorithms (SODA) (1998)

5. Bollobás, B., Fernandez de la Vega, W.: The diameter of random regular graphs. Combinatorica 2(2), 125–134 (1982)
6. Brodal, G., Fagerberg, R.: External memory algorithms and data structures. http://www.madalgo.au.dk/~gerth/emF03/ (2003)
7. Brodal, G.S., Fagerberg, R.: Lower bounds for external memory dictionaries. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 546–554 (2003)
8. Buchsbaum, A.L., Goldwasser, M., Venkatasubramanian, S., Westbrook, J.R.: On external memory graph traversal. In: Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 859–860 (2000)
9. Dittrich, W., Hutchinson, D., Maheshwari, A.: Blocking in parallel multisearch problems (extended abstract). In: Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures. pp. 98–107. SPAA '98, ACM, New York, NY, USA (1998), http://doi.acm.org/10.1145/277651.277676
10. Erickson, J.: Lower bounds for external algebraic decision trees. In: Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 755–761 (2005)
11. Goodrich, M.T., Tsay, J.J., Cheng, N.C., Vitter, J., Vengroff, D.E., Vitter, J.S.: External-memory computational geometry (1993)
12. Hellerstein, J.M., Koutsoupias, E., Papadimitriou, C.H.: On the analysis of indexing schemes. In: Proc. 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS). pp. 249–256. Tucson, Arizona (12–15 May 1997)
13. Hellerstein, J.M., Koutsoupias, E., Miranker, D.P., Papadimitriou, C.H., SAMOLADAS, V.: On a model of indexability and its bounds for range queries. JOURNAL OF THE ACM 49, 35–55 (2002)
14. Karpinski, M., Nekrich, Y.: Predecessor queries in constant time? In: Proc. 13th Annual European Conference on Algorithms (ESA). pp. 238–248 (2005)
15. Knudsen, M., Larsen, K.: I/O-complexity of comparison and permutation problems. Master's thesis, DAIMI (November 1992)
16. Knuth, D.E.: The Art of Computer Programming: Sorting and Searching, vol. 3. Addison Wesley (1973)
17. Pătraşcu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proc. 38th Annual ACM Symposium on Theory of Computing (STOC). pp. 232–240 (2006)
18. Rödl, V.: On a packing and covering problem. European Journal of Combinatorics 6(1), 69–78 (1985)
19. Samoladas, V., Miranker, D.P.: A lower bound theorem for indexing schemes and its application to multidimensional range queries. In: Proc. ACM Symposium on Principles of Database Systemsi (PODS). pp. 44–51 (1998)
20. Subramanian, S., Ramaswamy, S.: The p-range tree: A new data structure for range searching in secondary memory. In: Proc. Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 378–387 (1995)
21. Tamassia, R., Vitter, J.S.: Optimal cooperative search in fractional cascaded data structures. In: Algorithmica. pp. 307–316 (1990)
22. Tao, T., Vu, V.H.: Additive Combinatorics. Cambridge Studies in Advanced Mathematics (2009)
23. Verbin, E., Zhang, Q.: The limits of buffering: A tight lower bound for dynamic membership in the external memory model. In: Proc. 42nd ACM Symposium on Theory of Computing (STOC). pp. 447–456 (2010)

A Appendix

Proof of Lemma 1. The proof is by an information-theoretic argument. The total number of possible interleavings is $\binom{n+x}{x}$. One block transfer can reduce the number of candidate interleavings by at most a factor of $\binom{M}{B}$. As such, the lower bound is given by

$$\frac{\log \binom{n+x}{x}}{\log \binom{M}{B}} \geq \frac{x \log(n/x)}{\Theta(B \log(M/B))} = \Omega\left(\frac{x}{B} \log_{M/B} \frac{n}{x}\right).$$

Finally, $\Omega(x/B)$ is a lower bound due to the output size. □

Proof of Observation 6. Solving the following maximization program, where c_i is the number of pivots dedicated to the ith element parallelized,

$$\max \sum_{i=1}^{g} \log(c_i + 1) \text{ subject to } \sum_{i=1}^{g} c_i = B,$$

gives that for all i, $c_i = B/g$. □

Proof of Lemma 2. From our previous observation, we know that the most bits an I/O can learn while parallelizing $j/a - 1$ elements is $(j/a - 1) \log\left(B/(j/a - 1) + 1\right)$ bits. For all $a \geq 1$ and $j \geq 2$, the following condition holds:

$$\frac{j}{a} \log\left(\frac{B}{j} + 1\right) > \left(\frac{j}{a} - 1\right) \log\left(\frac{B}{j/a - 1} + 1\right).$$

Thus, we can conclude that with the block transfer of b_i, the algorithm must have parallelized strictly more than $j/a - 1$ distinct elements. □

Proof of Observation 9. Assume there exists a vector choice for which there is no block with a pivot from each search interval. The adversary will then make decisions consistent with assigning these search intervals to $q_1, \ldots, q_{j/a}$, and thus avoid parallelization. Note that no natural blocks already contain the combinations of these search intervals, and that such block must be manufactured in the preprocessing phase. □

Proof of Lemma 3. Call t_i the number of pivots in the block b that fall in distinct potential search intervals of the element q_i. Then we are trying to maximize $\prod_{i=1}^{j/a} t_i$ subject to $\sum_{i=1}^{j/a} t_i = B$. This gives that $t_i = aB/j$. By selecting aB/j pivots for each element, we cover $(aB/j)^{j/a}$ distinct vectors. □

Proof of Lemma 4. We abbreviate by setting $r = r_W$. Let H be the connected spanning subgraph of G corresponding to $f_G(W)$; that is, $|H| = f_G(W)$. By definition, H contains the start node a, node set W, and some directed path from a to each $v \in W$. Taking these directed path from a BFS starting at a to each $v \in W$ and removing the direction on each edge, we get an undirected tree T. Since T is a subgraph of H, $|T| \leq |H|$ and $|T| = |H|$ since H is the minimum connected spanning subgraph on W.

We show a lower bound on $|T|$. We find a minimum spanning tree to connect the nodes in W using the edges in graph T. Let α denote the number of nodes in this MST. Note that this MST might not be equal to T because it may be that $a \notin W$. Since T is also a tree spanning the node set W, $\alpha \leq |T|$. By definition, the distance $d_T(u, v) \geq \Lambda_G(u, v)$. Let β be the minimum number of nodes in a traveling salesman tour (assume that an edge can be traversed more than once), in graph T, visiting all nodes in W. Then,

$$\beta = d_T(v_1, v_2) + d_T(v_2, v_3) + \cdots + d_T(v_{|W|}, v_1) \geq r|W|, \tag{2}$$

for any ordering of nodes in W. Since the weight of an MST in an undirected graph is at least half the TSP tour length, $\alpha \leq \beta \leq 2\alpha$, we have $|T| \geq \alpha \geq \beta/2$ as desired. □

Proof of Lemma 5. Construct an undirected graph $H = (U, F)$ such that $U = V$ and $(u, v) \in F$ iff $\Lambda_G(u, v) \in [1, r]$. Then, H has at most $2r|V|B^r$ edges. By Turán Theorem [22], there exists an independent set of the desired size in H, which corresponds to an r-independent set in G, completing the proof. □

Proof of Lemma 6. In [5], Bollobás shows that a random k-regular graph has such small diameter with probability close to 1. Thus there exists some graph satisfying the constraints. □

Proof of Theorem 10. We partition S into S_ℓ and S_h by the frequency of elements in these n^k blocks and claim that there exists $Q \subseteq S_\ell$ such that query time for Q matches the lower bound.

Let S_ℓ be the set of elements of frequency no more than $2Bn^k/n$, i.e., twice of the average frequency. The rest of elements belong to S_h. By the Markov inequality, we have

$$|S_\ell| = \Theta(n) \text{ and } |S_h| \leq n/2. \tag{3}$$

Let $G(V, E)$ represent the connections between the n^k blocks such that each node represents a block and each directed edge represents a pointer. We partition V into V_1 and V_2 such that V_1 is the set of blocks containing no elements in S_h and $V_2 = V \setminus V_1$. Since each block can at most contain B elements in S_ℓ,

$$|V_1| = \Theta(n/B). \tag{4}$$

Then, we add some additional pointers to G and obtain a new graph G' such that, for each $e \in S_\ell$, for each $u, v \in V$, if u, v have element e in common, then $\Lambda_G(u, v)$ is small. We achieve this by, for each $e \in S_\ell$, we use a graph H_e to connect all the n^k blocks containing element e such that the diameter in H_e is small and the degree for each node in H_e is $\mathcal{O}(B^\delta)$ for some constant δ. By Lemma 6, the diameter of H_e can be as small as

$$\rho \leq \frac{1}{\delta} \log_B |H_e| + o(\log_B |H_e|) \leq \frac{k-1}{\delta} \log_B n + o(\log_B n). \tag{5}$$

We claim that the graph G' has an $(2\rho + \varepsilon_1)$-independent set of size n^{ε_2}, for some constants $\varepsilon_1, \varepsilon_2 > 0$. For the purpose, we construct an undirected graph $H(V_1, F)$ such

that $(u, v) \in F$ iff $\Lambda_G(u, v) \leq r$. Since the degree of each node in G' is bounded by $\mathcal{O}(B^{\delta+1})$, by Lemma 5, there exists a r-independent set I of size

$$\begin{aligned} |I| &\geq \frac{|V_1|^2}{|V_1| + 4r|V|\mathcal{O}(B^{r(\delta+1)})} \\ &\geq \frac{n^{2-k}}{4r\mathcal{O}(B^{r(\delta+1)+2})} \\ &= n^{\varepsilon_2} \end{aligned}$$

Then,

$$r = \frac{(2 - k - \varepsilon_2)}{\delta + 1} \log_B n + o(\log_B n). \tag{6}$$

To satisfy the condition made in the claim, we let $r > 2\rho$. Hence,

$$\frac{2 - k - \varepsilon_2}{\delta + 1} > 2\frac{k - 1}{\delta}. \tag{7}$$

Then, $k \to 4/3$ if δ is sufficiently large. Observe that, for each element $e \in S_\ell$, e is contained in at most one node in I; in addition, for every pair $e_1, e_2 \in S_\ell$ and e_1, e_2 contained in some node in I, then $\Lambda_G(u, v) \geq \varepsilon_1$ for $u \ni e_1, v \ni e_2$. Thus, by Lemma 4, we are done. □