Cell probe lower bounds

Ian Mertz

June 21, 2018

1 The cell probe model

In previous lectures, we've seen a number of different data structures that have allowed us to efficiently compute queries and updates for ADTs such as predecessor. For these upper bounds, our notion of "efficient" has been in a very classical sense, defined as the worst-case (or possibly amortized) runtime needed to solve the supported operations, where the data is split into words of size b and most standard operations on words, such as bitshift or multiplication, only take one time step, while well known operations like sorting take time proportional to their best classical algorithm. This is commonly referred to as the *RAM* model.

We take reading or writing a word from the data structure, which we call a *cell*, as an atomic operation, regardless of the word size b and, thus, even if we relax or strengthen the computation model to something other than the RAM model, it's clear that a lower bound on the number of cells we need to probe (i.e. read or write) translates to a lower bound on the runtime regardless of which model we choose. This motivates the *cell probe model of complexity*, wherein the runtime of an algorithm is simply defined as the number of distinct cells the algorithm probes in the course of its execution. In many cases, we can imagine storing extra data to reduce the number of cells we need to probe. Even with this idea we can often prove a cell probe lower bound on dynamic data structures as this extra data often requires maintenance itself. Furthermore, most of the known upper bounds on cell probe complexity are actually upper bounds in the RAM model. Thus despite the strength of the cell probe model, it does not seem unreasonable—and in fact has proven very useful—to consider data structure problems in this framework.

The cell probe model was introduced by Yao [6] to discuss the time complexity of searching for a given item in a sorted list of size n over the integers of size at most 2^b. The simplest method to solve such queries would be to maintain the list in sorted order and then to perform binary search on a given query, giving an upper bound of $\log n$ even in the RAM model. However if all we have access to is the sorted list and no other memory, the paper's main result shows that $\log n$ is also a lower bound even in the cell probe model. Another result of the paper was that one can achieve huge savings in the cell probe model; for a more complicated data structure involving some cleverly encoded cells, two probes are sufficient for certain values of n. For a concrete example consider a list with just two distinct elements of $\{1, 2, 3\}$; clearly a sorted list requires probing both elements to check if 2 is in the list, while we could simply store the missing element instead. Of course for a static problem such as searching in a list, given enough space one could just encode the answer to every query and get a cell probe upper bound of 1; the result presented in [6] gives a polynomial-size data structure and thus achieving a $O(1)$ upper bound is non-trivial. For the rest of this lecture, we focus on dynamic systems, where we can achieve lower bounds without imposing any constraints on space.

We briefly lay out a roadmap for the rest of the lecture. In Section 2 we introduce the central problem for which cell probe lower bounds exist, and we give an overview of these lower bounds and their central technique. In Section 3 we give a proof from one of the papers as an example of how these lower bounds are proven, and we briefly address the differences in how these techniques are used between various papers. In Section 4 we show how the notion of reductions can be used to prove cell probe lower bounds, and in particular we prove a lower bound on a natural graph problem.

2 Prefix sum ADT

We focus now on a particular ADT called the *prefix sum* ADT, consisting of a list $A[1 \dots n]$ of n integers in $\{0, \dots, \Delta - 1\}$ with the following operations:

- $update(k, \delta)$: set $A[k]$ to be $(A[k] + \delta) \mod \Delta$ assuming $k \in \{1, \dots, n\}$, $\delta \in \{0, \dots, \Delta - 1\}$
- $query(k)$: return $(\sum_{i=1}^{k} A[i]) \mod \Delta$

In particular we refer to the special case when $\Delta = 2$ as the *parity prefix sum* ADT, and for simplicity we assume $update$ only takes the index k as an argument and flips it. Recall that b is the word size of our data structure, and so we make the further assumption that $\Delta \leq 2^b$ so that every element of A fits into a single word.

Given an arbitrary data structure implementing prefix sum, we denote by t_u and t_q the amortized probe complexity of $update$ and $query$ respectively. First we observe that there exists a trivial $t_u, t_q \leq \log n$ solution to the prefix sum ADT in the cell probe model that works even for the worst case runtime in the RAM model: we put a leaf-oriented balanced binary tree on top of the list A and at each internal node v we store the sum (mod Δ) of the leaves in the subtree rooted at v. An $update$ corresponds to updating each node on the path from the corresponding leaf to the root, and a $query$ corresponds to a sum of at most one node from each level of the tree.

The first lower bound in the cell probe model for a dynamic ADT was given by Fredman and Saks [1], where they showed $\max(t_u, t_q) = \Omega(\frac{\log n}{\log \log n + \log b})$ for the parity prefix sum ADT. More specifically they showed an alternating sequence of m $update$ and $query$ operations such that, with high probability, $t_q = \Omega(\frac{\log n}{\log t_u + \log \log n + \log b})$. Nearly twenty years later, Pătraşcu and Demaine [5] removed the $\log \log n$ by showing $\max(t_u, t_q) = \Omega(\frac{\log n}{\log b - \log \log \Delta})$, and again more specifically they show that $t_q(\log \frac{b}{\log \Delta} + \log \frac{t_u}{t_q}) = \Omega(\log n)$. We leave it as an exercise to the reader to see how these simpler lower bounds are implied by the more precise ones stated. This turns out to be tight, as they give a data structure that achieves $t_u = t_q = O(\frac{\log n}{\log b - \log \log \Delta})$ in the worst-case in the RAM model. This gives another case where we cannot get any additional power from only charging for cell probes.

Both [1] and [5] used a novel technique introduced in [1] called the *chronogram* technique. Roughly speaking we take a sequence of $update$ and $query$ operations whose arguments are chosen uniformly at random, and then divide the sequence into epochs. Each cell is given a timestamp indicating the last time it was updated. We then calculate, for each epoch i, the expected number of $query$ operations which involve a cell with a timestamp in epoch $i - 1$. Intuitively, if we are constantly needing to use freshly updated cells, then there should be a large number of cell probes needed overall. The chronogram technique continues to appear throughout the cell probe literature every year, and most recently (or notably) was used by Larsen et al [4] to eventually prove an $\Omega(\log^{1.5} n)$ lower bound for a different ADT called the *parity 2D range sum* ADT, the strongest known cell probe lower bound to date.

Here we make a note about the size of the output. Another result of [1] achieves a lower bound of $\Omega((\frac{\log n}{\log\log n})^2)$ for the case of $b = 1$ and $\Delta = n$ (for this result they break our assumption that $\Delta \leq 2^b$). This would seem to contradict our claim that the $\Omega(\log^{1.5} n)$ lower bound for parity 2D range sum of [4], a result from nearly thirty years later, is the strongest known lower bound in the cell probe model. However in a query to the parity prefix sum ADT we need only output a single bit, whereas for a query to general prefix sum we need $\log \Delta$ bits to specify the answer, and when $\Delta = O(n)$ this translates to $\log n$ bits of output. While this lower bound of [1] is still impressive, it's not clear a priori that we would need all $\Omega((\frac{\log n}{\log\log n})^2)$ probes in order to output a *single bit* of a query output, and in fact it may only take $O(\frac{1}{\log n}(\frac{\log n}{\log\log n})^2) = O(\frac{\log n}{(\log\log n)^2}) = o(\log n)$ probes per bit. Another way to interpret this observation is that it isn't clear, for a problem requiring many output bits for a query, whether the difficulty of the problem stems from the inherent difficulty of the problem or simply from the output size. This technicality is referenced as the central motivation in [4] and was posed in [3] five years prior, which showed $t_q = \Omega((\frac{\log n}{\log b + \log t_u})^2)$ for a different range counting problem, where the output is again of size $\Theta(\log n)$. The lower bound of [5] holds for all choices of Δ and thus namely gives a $\Omega(\log n)$ lower bound for parity prefix sum as long as $b = O(\log \Delta) = O(1)$.

3 The chronogram technique

In this section we present a different and slightly weaker version of the lower bound of [5], which still achieves $\Omega(\log n)$ for the case $b = O(\log \Delta)$. We state the theorem and then first present a brief sketch of the proof technique before presenting the proof in full.

Theorem 3.1 (Theorem 2.1, [5])**.** *Consider any cell-probe data structure for prefix sum ADT, and let t_u and t_q be the amortized cell probe complexity of $update(k, \delta)$ and $query(k)$ respectively. Then* $\min(t_u, t_q) \log \frac{\max(t_u, t_q)}{\min(t_u, t_q)} = \Omega(\frac{\log n}{b/\log \Delta})$.

Proof sketch. Given a data structure for prefix sum, we create a hard instance which consists of a sequence of blocks made up of update and query operations, all of which take random indices into A as arguments. We put a balanced (non-binary) tree on top of this sequence and for each node v in the tree we analyze the amount of information produced by random updates in left subtrees of v that are needed to compute the queries in right subtrees of v. We prove a technical lemma (Lemma 3.2) that shows that a constant fraction of the updates on the left each cause at least one distinct query on the right to be completely uncertain. This means that for each such $update$-$query$ pair, at least $\log \Delta$ bits, or equivalently $\frac{\log \Delta}{b}$ words, need to be transferred to the $query$. We charge this to the lowest common ancestor of the two subtrees, namely v. Summing up across all v on the same level we end up charging $\Omega(\frac{\log \Delta}{b})$ per $query$ for a constant fraction of all queries in the tree, and then summing across approximately $\log n$ levels of the tree gives us the lower bound. □

Proof. We first prove an auxilliary technical lemma from [5] as promised in our sketch.

Lemma 3.2 (Lemma 5.1, [5])**.** *Consider two adjacent intervals of operations such that the left interval contains at least L updates, the right interval contains at least L queries, and there are a total of $O(n^{1/3})$ operations in both intervals. Furthermore assume that all arguments for the updates and queries are chosen uniformly at random. Then $\mathbf{E}[c] = \Omega(\frac{L}{b/\log \Delta})$, where c is the number of cell probes executed during the right interval that read cells last written during the left interval.*

Proof. For the moment we assume that every index k into A chosen as an argument to an operation in either interval is unique. Let S be the set of all of these arguments, which has size $O(n^{1/3})$. Consider sets $U, Q \subseteq S$, which are the set of arguments of the updates from the left interval and queries from the right interval respectively. Note that S, U, and Q are all random subsets of $[n]$, and U and Q have size at least L. We define an *interleaving* of U and Q to be an ordered list $u_1 < q_1 < u_2 < q_2 < \ldots < u_I < q_I$ where $u_i \in U$ and $q_i \in Q$ for all $i \in [I]$. We define the *interleaving number* of to be the largest such I. Consider the ordered list of all arguments of U and Q, which has size at least $2L$. We count the expected number of times in the ordered list we have an adjacent pair of arguments where the first is from U and the second is from Q.

We claim that $\mathbf{E}[I|$ all arguments are unique$] = \Omega(L)$. To do this we view the sampling procedure for U and Q as follows: first, sample L distinct arguments from $\{1, \ldots, n\}$, which we call T. Now label each argument in T as an $update$ or a $query$ independently with probability $\frac{1}{2}$ each. Finally sample the remaining arguments in U and Q from $\{1, \ldots, n\}/T$ without replacement. Clearly this is the same as choosing the set of all arguments for both U and Q uniformly from the set of all subsets of $\{1, \ldots, n\}$ of size $|U| + |Q|$. We order T as $s_1 < s_2 < \ldots < s_L$, and consider all pairs s_{2k-1}, s_{2k} for $k = 1 \ldots L/2$. For any fixed pair the probability of s_{2k-1} being labeled $update$ and s_{2k} being labeled $query$ is $\frac{1}{4}$. Since all pairs are non-overlapping and all labels are independent, by linearity of expectation there are $\frac{1}{4} \cdot \frac{L}{2}$ such pairs in T. Thus $\mathbf{E}[I|$ all arguments are unique$] \geq \frac{L}{8} = \Omega(L)$.

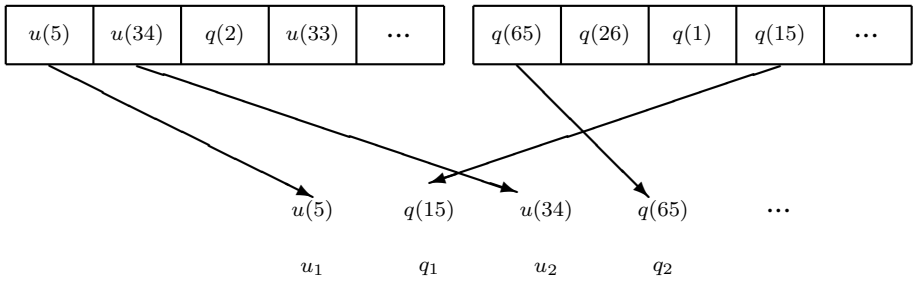

Figure 1: Interleaving between two intervals

Let $q_1 < \ldots < q_I$ be the arguments to the queries in our maximal interleaving of U and Q, and consider the value returned by $query(q_i)$ for any $i \in [I]$. Because a random update occured in the range (q_{i-1}, q_i), even conditioned on knowing $query(q_{i-1})$ there are Δ possibilities for $query(q_i)$. Thus letting $\overline{q} = \langle query(q_1), query(q_2), \ldots query(q_I) \rangle$ be the tuple of all answers to these queries, all Δ^I possibilities are equally likely, and thus we expect $\overline{q}$ to need at least $I \log \Delta = \Omega(L \log \Delta)$ bits to be represented. Thus, since a cell contains at most b bits of information and we need to query enough bits to capture all of $\overline{q}$, we find that $\mathbf{E}[c|$ all arguments are unique$] = \Omega(L\frac{\log \Delta}{b}) = \Omega(\frac{L}{b/\log \Delta})$ as required.

Finally we remove the condition that all arguments are unique. We first state two important lemmas from probability theory.

Lemma 3.3 (Birthday paradox)**.** *Given m elements chosen independently and uniformly at random from $\{1 \ldots n\}$, the expected number of elements which are not unique is asymptotically $\Theta(\frac{m^2}{n})$.*

Lemma 3.4 (Markov's inequality)**.** *For any random variable Y, $\Pr(|Y - \mathbf{E}[Y]| \geq t) \leq \frac{\mathbf{E}[Y]}{t}$.*

Applying the birthday paradox to the variable Y = number of args which are not unique, we see that $E[Y] = \Theta(\frac{(n^{1/3})^2}{n}) = \Theta(n^{-1/3}) = o(1)$. Thus by Markov's inequality with $t = \epsilon$ for some constant $\epsilon > 0$ we get that $\Pr(Y \geq \epsilon + o(1)) \leq \frac{o(1)}{\epsilon} = o(1)$, and so with probability $1 - o(1)$ there is less than one non-unique element in S. In other words, $\Pr(\text{all args are unique}) \geq 1 - o(1)$. Thus it follows that

$$\begin{aligned} \mathbf{E}[c] &\geq \mathbf{E}[c|\text{ all args are unique}] \cdot \Pr(\text{all args are unique}) \\ &= \Omega(\tfrac{L}{b/\log\Delta} \cdot (1 - o(1))) \\ &= \Omega(\tfrac{L}{b/\log\Delta}) \end{aligned}$$

which completes the proof. □

We now return to proving Theorem 3.1. Assume there exists a data structure for prefix sum for which an amortized $update$ probes t_u cells and an amortized $query$ probes t_q cells. Consider an instance which consists of m blocks for some parameter m to be chosen later, where each block has t_q random updates and t_u random queries, all in some arbitrary order. The amortized number of probes in each block is at most $2t_u t_q$, as each of the t_q updates takes at most t_u probes and each of the t_u queries takes at most t_q probes. We will show that the expected amortized number of probes per block is also at least $\Omega(\frac{\log_B n}{b/\log\Delta}\max(t_u, t_q))$, where $B = 2\max(\frac{t_u}{t_q}, \frac{t_q}{t_u})$. If this holds then $2t_u t_q = \Omega(\frac{\log_B n}{b/\log\Delta}\max(t_u, t_q))$, and rearranging using the change of base formula and facts about $\max$ and $\min$ completes the proof.

First observe that if $\max(t_u, t_q) = \Omega(n^{1/6})$ then $\min(t_u, t_q)\log\frac{\max(t_u,t_q)}{\min(t_u,t_q)} \geq \log\max(t_u, t_q) = \Omega(\log n)$, and thus we assume that $\max(t_u, t_q) = O(n^{1/6})$. We let $m = \Theta(n^{1/6})$, meaning that adding up over all blocks the total number of operations is $m(t_u + t_q) = O(n^{1/6} \cdot n^{1/6}) = O(n^{1/3})$. We put the blocks in chronological order and lay a balanced B-ary tree on top of it.

Assume for the moment that $t_u \geq t_q$. Fix any internal node v at level i and consider any child w of v among its rightmost $\frac{B}{2}$ children. Let $\ell = B^{i-1}$ be the number of blocks in the subtree rooted at w, and let $T_{v,w}$ be the interval defined by the leaves in this subtree, which contains ℓt_u queries. Take the union of all left siblings of w and define $S_{v,w}$ to be the interval defined by the leaves in this union, which contains at least $\frac{B}{2}\ell$ blocks because w is in the right half of v's children. Hence there are at least $\frac{2t_u/t_q}{2}\ell t_q = \ell t_u$ updates in $S_{v,w}$. Setting $L = \ell t_u$ we've defined a left interval $S_{v,w}$ with at least L updates and a right interval $T_{v,w}$ with L queries, and there are $O(n^{1/3})$ total operations in $S_{v,w} \cup T_{v,w}$ because there are at most that many operations in the whole tree. Thus we can apply Lemma 3.2 to find that in expectation there are $\Omega(\frac{\ell t_u}{b/\log\Delta})$ cell probes needed during the queries in $T_{v,w}$ probing locations last updated in $S_{v,w}$.

For convenient notation let $c_{v,w}$ be the number of cell probes executed during $T_{v,w}$ that read cells last written during $S_{v,w}$. Let c_i be the total, over all v on level i and right children w of v, of cell probes executed during $T_{v,w}$ that read cells last written during $S_{v,w}$. Let c be the total, over all v and right children w of v, of cell probes executed during $T_{v,w}$ that read cells last written during $S_{v,w}$. If we can get a lower bound on $\mathbf{E}[c]$ then this gives us a lower bound on the expected number of cell probes needed to compute all queries in the tree. Our application of Lemma 3.2 showed that $\mathbf{E}[c_{v,w}] = \Omega(\frac{\ell t_u}{b/\log\Delta})$.

Observe that the subtree rooted at w is disjoint from the subtree rooted at any other node, and thus if we consider some w' which is also a right child of v, $T_{v,w'}$ is completely disjoint from $T_{v,w}$. Likewise if we consider different nodes v' at level i and w', $T_{v',w'}$ is completely disjoint from $T_{v,w}$. We recall that all arguments to updates and queries are chosen independently and uniformly. Because our lower bound from Lemma 3.2 is an expectation our lower bound for any choice of v and w is an expectation.

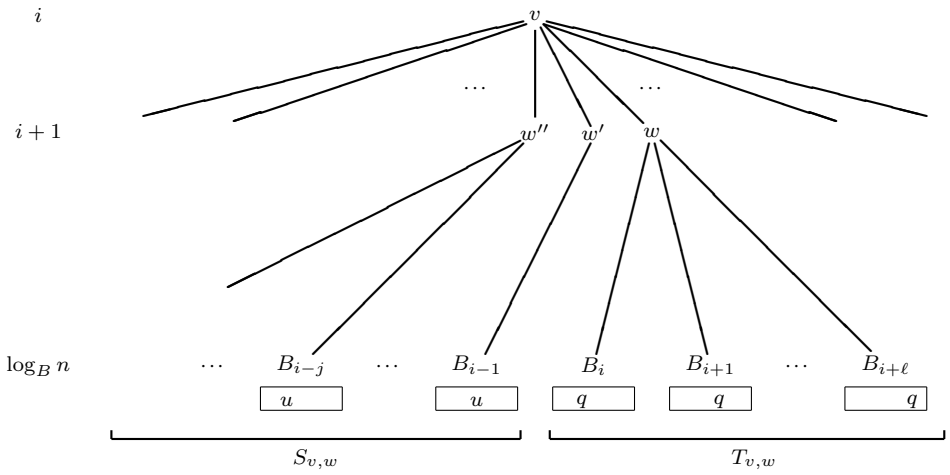

Figure 2: Tree and intervals for a given v, w

Therefore by linearity of expectation we can sum up all lower bounds for v at level i and w. The tree is balanced so $\ell = |T_{v,w}|$ is the same for all choices of v and w. Hence there are $\frac{m}{2\ell}$ choices for v and w. It follows that $\mathbf{E}[c_i] = \sum_{v,w} \mathbf{E}[c_{v,w}] = \frac{m}{2\ell} \cdot \Omega(\frac{\ell t_u}{b/\log \Delta}) = \Omega(\frac{m t_u}{b/\log \Delta})$.

We now consider some other level i'. Clearly the same argument gives a $\Omega(\frac{m t_u}{b/\log \Delta})$ expected lower bound as well. We claim that the bounds for levels i and i' can be summed up without double counting any information needed to be transferred to a given query q. Let v and w be the v and w from level i containing query q in our previous argument, and let v' and w' be the v and w from level i' containing q. Without loss of generality let $i < i'$. Then the subtree rooted at w' contains the entirety of the subtree rooted at v, and because w' is a left child of v' any update in a left child of v cannot appear in the subtree rooted at any left child of v' and vice versa. Thus we can add up over all $\log_B n$ levels of the tree to get $\mathbf{E}[c] = \sum_i \mathbf{E}[c_i] = \Omega(\frac{m t_u \log_B n}{b/\log \Delta})$, which is a lower bound on the expected cost of all blocks in the tree. Amortizing over all m blocks gives a cost of $\Omega(\frac{t_u \log_B n}{b/\log \Delta})$ per block.

If $t_u \leq t_q$ we have the exact same argument but instead we let w be a left child of v and consider all updates in w and all queries in right siblings of w, which gives an amortized expected cost of of $\Omega(\frac{t_q \log_B n}{b/\log \Delta})$ per block. Together we get an expected amortized lower bound of $\Omega(\frac{\log_B n}{b/\log \Delta} \max(t_u, t_q))$ per block as required, and the theorem follows. □

It is noted in [5] that this lower bound also implies $\max(t_u, t_q) = \Omega(\frac{\log n}{b/\log \Delta})$, which only differs from their main theorem by taking a logarithm in the denominator.

We briefly address the differences in the proofs of [1], [5], and [4] at a high level. The chronogram technique as it is introduced in [1] considers a fixed set of $t = \Theta(\frac{\log n}{\log b + \log\log n})$ epochs, the ith of which has size approximately $(\log n)^{t-i}$. Here updates and queries alternate, and it's shown that there are sufficiently many queries that require at least one cell probe to locations last updated in $\Theta(t)$ different epochs. This argument uses the fact that the epochs decrease by a logarithmic factor in size and thus later epochs cannot "invalidate" every update from the earlier epochs. This is in contrast to the proofs in [5] where the epochs vary depending on which level of the B-ary tree we are considering. However both proofs argue that the updates force the algorithm to make a certain number of cell queries by

a coding argument, claiming that the information obtained from too few cell probes is insufficient to answer all queries.

The fixed logarithmically-decreasing interval strategy is also used in [4] and before it in [3], but these papers use the 2D range query problem, where the object is a set of points in two dimensions with an $add(x, y)$ update and $query(x, y)$ which returns the number of points in the set which have both a smaller x and y coordinate than the given values. This problem is chosen so that they can argue that for a random query at the end of the last epoch, there is not just one $update$ per epoch that's needed but rather $O(\frac{\log n}{\log \log n})$ updates per epoch that require their own cell probes. The central improvements of [4] over [3] are to make the query output binary (see our discussion from the previous section) and change the coding argument.

4 Reducing to dynamic graph problems

Closely related to the prefix sum ADT are a number of classical and well-known graph problems. Of note is *dynamic graph connectivity*, consisting of a graph $G = (V, E)$ with the following operations:

- $insert(u, v)$: add the edge (u, v) to G
- $delete(u, v)$: remove the edge (u, v) from G
- $query(u, v)$: return true iff u and v are connected in G

We show a lower bound for dynamic graph connectivity by way of a reduction from parity prefix sum. Since Theorem 3.1 proves a lower bound on $\max(t_u, t_q)$, it is enough to show that we can maintain a graph such that updates and queries in parity prefix sum can each be computed by a constant number of inserts, deletes, and queries in dynamic graph connectivity.

Theorem 4.1 (Main result for connectivity, [2])**.** *Consider any cell-probe data structure for the dynamic graph connectivity problem, and let b be the word size, in bits, of a memory cell. Let t_u and t_q be the complexity of $update$ and $query$ for some given data structure for the parity prefix sum ADT, and let t'_i, t'_d, and t'_q be the cell probe complexity of $insert$, $delete$, and $query$ for the dynamic graph connectivity problem respectively. Then $t_u \leq 2(t'_i + t'_d + t'_q)$ and $t_q \leq t'_q$.*

Proof. Given an instance of parity prefix sum A, we build a graph as follows: we have $2n + 2$ vertices, labeled $s_0 \dots s_n, t_0 \dots t_n$. For all i, if $A[i] = 0$ we add the edges (s_{i-1}, s_i) and (t_{i-1}, t_i), and if $A[i] = 1$ we add the edges (s_{i-1}, t_i) and (t_{i-1}, s_i). We now claim that $query(k) = 0$ in the parity prefix sum instance iff $query(s_0, s_k)$ returns true, and $query(k) = 1$ iff $query(s_0, t_k)$ returns true. This can be easily seen by induction: we added the edge (s_0, s_1) iff $A[1] = 0$, and if $A[1] = 1$ then we added the edge (s_0, t_1) instead. For the induction step if $A[k+1] = 0$ then $query(k+1) = query(k)$, and our graph has the edges (s_k, s_{k+1}) and (t_k, t_{k+1}). Likewise if $A[k+1] = 1$ then $query(k+1) = \overline{query(k)}$, and our graph has the edges (s_k, t_{k+1}) and (t_k, s_{k+1}).

Thus we can compute a query in parity prefix sum with a single query in dynamic graph connectivity; given $query(k)$ we run $query(s_0, s_k)$ and output 0 iff the query returns true. For $update(k)$ we need to figure out if the edges going into level k cross between s to t or not, and so we perform $query(s_0, s_{k-1})$ and $query(s_0, s_k)$. If they are equal then using the same induction argument we know that we have edges (s_{k-1}, s_k) and (t_{k-1}, t_k) in the graph, and otherwise we have (s_{k-1}, t_k) and (t_{k-1}, s_k) instead. Thus if the queries return the same answer we delete (s_{k-1}, s_k) and (t_{k-1}, t_k) and insert (s_{k-1}, t_k) and (t_{k-1}, s_k), and otherwise we delete (s_{k-1}, t_k) and (t_{k-1}, s_k) and add (s_{k-1}, s_k)

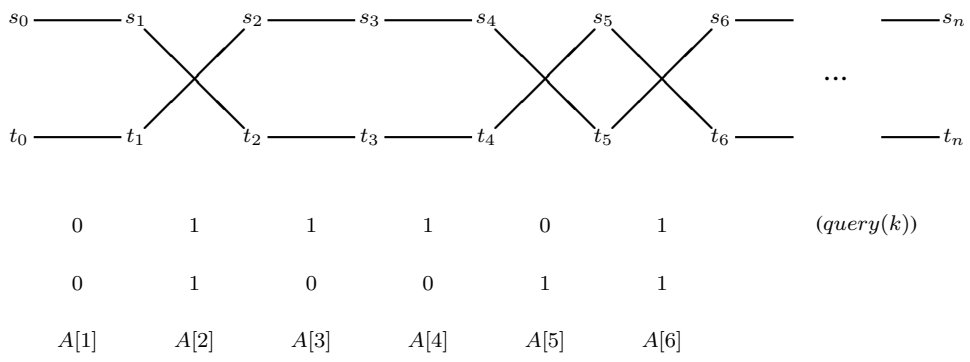

Figure 3: Reduction graph from parity prefix sum to dynamic connectivity

and (t_{k-1}, t_k). This flips which s_j and t_j are connected to s_0 for all $j \geq k$ and leaves them untouched for all $j < k$, which is correct as flipping k flips the sum for all $j \geq k$ and leaves them untouched for all $j < k$. □

Because Theorem 3.1 holds even if initially $A[i] = 0$ for all i, we can remove all preprocessing costs in the cell probe model, but this is not necessary for either lower bound.

References

[1] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989.

[2] M. R. Henzinger and M. L. Fredman. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22(3):351–362, 1998.

[3] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing (STOC)*, pages 85–94, 2012.

[4] Kasper Green Larsen, Omri Weinstein, and Huacheng Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In *Proceedings of the Fiftieth Annual ACM Symposium on Theory of Computing (STOC)*, 2018. To appear.

[5] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[6] Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, 1981.