6.897: Advanced Data Structures Spring 2005

Lecture 17 — April 7, 2005

Lecturer: Mihai Pătraşcu *Scribe: Tim Abbott*

1 Overview

In the last lecture we saw how to solve marked ancestor using $O(\frac{\log n}{\log \log n})$ time for queries, and $O(\log \log n)$ time for updates. Today we give an essentially tight lower bound for the existential marked ancestor problem in the cell probe model, due to Alstrup, Husfeldt and Rauhe [1]. Existential marked ancestor queries require an answer as to whether or not the node has a marked ancestor; the updates of marking or unmarking a node are the same as before. Thus, existential queries are easier that the ones we considered before. Recall that in the cell probe model, the cost being bounded is the number of cell probes to $\Theta(\log n)$-bit cells. The argument we use is an interesting refinement of the Chronogram Technique, introduced by Fredman and Saks in [2].

2 A Lower Bound for Counting Ancestors

Recall that a data structure for the marked ancestor problem supports two types of operations on a static tree. One is an update, where we either set or unset the mark bit of a given node. The query on the data structure varies with the version, but in general we are given a leaf and asked to compute some property of the marked ancestors of that leaf (whether they exist, how many of them there are, the lowest one etc). We first consider the problem of counting the number of marked ancestors, modulo 2. It is much easier to obtain a lower bound for the counting problem than for the existential problem, and we use this opportunity to introduce the general technique.

Theorem 1. *Let t_u be the update time, and t_q be the query time. For a perfect tree of branching factor $B \geq t_u \log^2 n$, we have the tradeoff*

$$t_q = \Omega(\log_B n) = \Omega\left(\frac{\log n}{\log t_u + \log \log n}\right)$$

Note that for any $t_u = \lg^{O(1)} n$, we have that $t_q = \Omega(\frac{\log n}{\log \log n})$. Thus, our bound from last lecture was tight. Note also that the lower bound is linear in the height of the tree considered, so we are proving that simply scanning the root-to-leaf path is optimal for the complete tree. We will show the bound in the worst case; it also holds in the amortized case. Finally, observe that the theorem extends easily to binary trees, because we can embed our tree in a binary tree (ignore all levels nondivisible by $\lg B$).

2.1 A Hard Sequence of Operations

We first describe the hard sequence of operations on the data structure. The first thing we do is iterate through all the nodes of the tree, and at each node with probability $\frac{1}{2}$ set the mark bit to

one (otherwise set it to 0). We do this from the bottom of the tree upwards (so that all the nodes on a given level are visited before any nodes on higher levels). The result is that the list of n values of the mark bits is a uniformly random vector in $\{0,1\}^n$. Then, after all of those updates, query a random leaf.

Intuitively, an update wants to inform the leaves in its subtree about the node's bit (because queries come at the leaves). At the very least, it could try to inform its children about the bit, so that a query could ignore this node. However, we set B to be larger than t_u, so intutively, whatever propagation an update does, it is only useful with negligible probability. To solidify this intuition, note that it is easy to handle the case when we scan and update nodes beginning with higher levels. When updating a node, it can just obtain a partial count from the root to its parent (because updates from higher levels happened before), and calculate a partial count for itself. Then all operations take constant time.

2.2 Proof of the Lower Bound

Define *epoch* j to be the time during which the updates on level j were being executed. Thus, in epoch 0 the root was updated, and in epoch 1 the B elements at the second level were updated; and in general in epoch j, B^j updates were executed. Note that the epochs occured temporally in order of decreasing j.

Our strategy is to show that the query algorithm must read in expectation $\Omega(1)$ cells written during each epoch. In total over the $\log_B n$ epochs, we achieve the desired lower bound in expectation. This implies that it must hold in the worst case as well. The only variability that occurs here is in the state of each node, and the identity of the random leaf. Thus, the problem is determined by $u \in \{0,1\}^n$, the state of all the nodes in the tree, along with the value of the random query element q. We then wish to show that for the random u we've constructed, and random q, the problem is hard.

Fix a level j. During epoch j, there was a chunk of B^j updates. After those updates were completed, there were a series of smaller epochs, exponentially decreasing in size. Suppose that our query algorithm reads none of the cells written in epoch j. It can read any cells from other epochs; we will give it all cells from all other epochs for free. We argue that with constant probability, the algorithm does not have enough information to determine the correct answer. The cells that were written in an epoch $k > j$ cannot be useful, since they know nothing about the state of level j (which was updated in the future). The total number of cells written in epochs $k < j$ is at most

$$t_u(1 + B + B^2 + \cdots + B^{j-1}) = \frac{B^j - 1}{B - 1} t_u = t_u \cdot O(B^{j-1})$$

Since each cell has $\Theta(\log n)$ bits, this are a total of $O(t_u B^{j-1} \log n)$ bits of information that were written in these epochs. Now, the total amount of information revealed by the updates in epoch j is B^j bits, the information in the state of the nodes of level j. Thus the information the query algorithm possesses is only $O(B^j / \log n) = o(B^j)$, by the value we set of B; it follows that at least a constant proportion of the mark bits at nodes in level j cannot be known by the query algorithm.

Now, we need to be able to compute the number of marked nodes on the path from our random query element to the root. Thus, we must know the value of a random node at level j (since the level-j ancestor of a random node is random). Thus, with at least a constant probability, the query algorithm must read some cell from epoch j.

We know sketch how to formalize this proof idea, though we do not delve into the real combinatorial analysis. Let R_j be the set of cells written during epoch j, and $R_{<j} = \bigcup_{i<j} R_i$. We think of R_j as including pairs of cell address and cell contents.

Now consider the strings of update values which differ from u only on the j-th level nodes, and are indistinguishable from u by looking at more recent epochs. That is, we define the equivalence class:

$$[u_j] = \{w \in \{0,1\}^n \mid u, w \text{ differ only on level } j, R_{<j}(u) = R_{<j}(w)\}$$

Observe that $R_k(u) = R_k(w), (\forall)k > j$, because these epochs occured in the past, and the first difference between u and w happened on level j. Thus, u and w can only be distinguished by looking at R_j.

Now the proof proceeds as follows:

- as shown above, $|R_{<j}(u)| = t_u \cdot O(B^{j-1})$.
- there are 2^{B^j} choices of w which differ from u only on the j-th level.
- by fixing $R_{<j}(w) = R_{<j}(u)$, we are eliminating a fraction of about $2^{-O(t_u B^{j-1} \lg n)}$ from the choices of w, for an average u. So $|[u_j]| \geq 2^{\Omega(B^j)}$, with constant probability over a random u.
- if $|[u_j]|$ is large, it must mean that for a constant fraction of the nodes t on the j-th level, $(\exists)w \in [u_j] : w_t \neq u_t$. Call these "bad t's".
- picking a random leaf induces a random t. With constant probability, this is a bad t. The query must distinguish between u and w (because they yield different answers to the counting problem), but they are indistinguishable except by looking at R_j.

3 Nondeterminism

Looking over the argument we just used, our proof still works even if the query is allowed to be nondeterminstic in the complexity theoretic sense. One way to view nondeterminism is to imagine a prover with perfect information, which is revealing a minimum number of cells to convince you of the answer to the query. With constant probability, the prover still needs to show us at least one cell written during epoch j, for each j. This is because any (even all) cells from the other epochs are not enough to determine the answer, as we argued above.

This realization is bad news, since it means the proof technique cannot apply directly to the existential problem. In the nondeterminstic setting, the prover can simply demonstrate the value of a marked bit in the chain, and use only one cell to answer the query. The hardness in this case comes from co-nondeterministic complexity: to prove that there are no marked ancestors, the prover must be able to show that the mark bits at each node on the path from leaf to root are all false. Before we use this hardness intuition for a tight lower bound in the existential case, we first show how the power of nondeterminism can be used to our advantage, to reduce the counting problem to the existential one. This proves a weaker lower bound, but in a very surprising way.

Theorem 2. *The existential marked ancestor problem requires $\Omega(\sqrt{\log n})$ time per operation.*

Proof. Assume we have $t_u = t_q = o(\sqrt{\log n})$. We construct a nondeterministic algorithm for the marked ancestor counting problem. Remember that we showed this problem has $t_q = \Omega(\log_B n)$, for any $B \geq t_u \lg^2 n$. We use as our static input tree a balanced tree of branching factor $B = 2^{\sqrt{\log n}}$. Now, the height of such a tree is $h = \log_B n = \log n / \sqrt{\log n} = \sqrt{\log n}$. Build 2^h copies of the tree, each using the assumed existential marked ancestor algorithm. Each copy corresponds to a subset S of the levels. For the tree associated to subset S, we flip all the mark bits at the levels in S. On an update request, we use $O(2^h t_u) = o(2^{2\sqrt{\log n}})$ time to update all of the trees. On a query for the number of marked nodes above a leaf, we can nondeterministically guess the set of marked ancestors T, and then prove that the guess is correct. This can be done by an existential query in the tree with the levels in T flipped. This query will return false iff the ancestors on the levels T are exactly those marked in the original tree. Thus the query time here is $O(t_q) = o(\sqrt{\log n})$. Now, we arrive at a contradiction by comparing with our lower bound from before:

$$t_q = \Omega\left(\frac{\log n}{\sqrt{\log n} + \log\log n}\right) = \Omega(\sqrt{\log n})$$

□

4 A Tight Bound for Existential Queries

Theorem 3. *Given a perfect tree with branching factor $B \geq t_u \log^4 n$, then $t_q = \Omega(\log_B n)$, for existential marked ancestor queries.*

Proof. For some $u \in \{0,1\}^n$, we use the same sequence of updates as before, updating by levels starting from the bottom. However, instead of using a uniform distribution on the space of possible u's, we choose that $u_i = 1$ with probability $p = \frac{1}{\log n}$, and $u_i = 0$ otherwise. This implies that the expected number of total marked ancestors above a given leaf is $\frac{1}{\log n} \log_B n = \frac{1}{\log B} = o(1)$. Then with probability $1 - o(1)$ the path is empty, which, as we commented before, is difficult to demonstrate.

We're going to make a similar epoch argument as before. However, we cannot show that the algorithm needs to access a cell from each R_j, for reasons that will be explained below. Instead, we will need to construct sets $\overline{R_j} \supseteq R_j$, and we will show that the query needs a cell from each $\overline{R_j}$. The intuition behind these supersets will be seen later, but think of them as slightly larger than R_j. We will construct them iteratively, as follows. Consider

$$[u_j] = \{w \in \{0,1\}^n \mid u, w \text{ only differ on the } j\text{-th level, and } \overline{R_{<j}}(u) = \overline{R_{<j}}(w)\}$$

Notice that $[u_j]$ is an equivalence class, much similar to the equivalence classes considered in the previous proof. Later, we will argue that $|[u_j]|$ is large enough, so that for $\Omega(B^j)$ choices of nodes t on the j-th level, $(\exists) w \in [u_j]$ with $w_t \neq u_t$. Thus, for many t's, we cannot answer the query based on $\overline{R_{<j}}$.

Define a *fooling set* $F_j(u) \subseteq [u_j]$ such that the property described in the last paragraph is true: for $\Omega(B^j)$ choices of t, $(\exists) w \in F_j(u) : w_t \neq u_t$. Before, we argued that a big $[u_j]$ represents a fooling set. A crucial insight is that even very small subsets of $[u_j]$ can still be fooling sets. More precisely, one can show that there exists $F_j(u) \subset [u_j]$, with $|F_j(u)| = O(\lg n)$, having the fooling property. We only discuss the intuition behind the existence of small fooling sets. Because $|[u_j]|$ is almost as

large as possible, a random set from it will differ from u in a fraction of $\Omega(p)$ of the places (and these will be somewhat random). If one chooses $\frac{O(1)}{p}$ random sets from $[u_j]$, we will find sets which differ from u in a constant fraction of the possible places.

Now we are finally ready to define $\overline{R_j}$ based on $\overline{R_i}, i < j$:

$$\overline{R_j}(u) = \left(\bigcup_{w \in F_j(u) \cup \{u\}} R_j(w) \right) \setminus \overline{R_{<j}}(u)$$

An important property of this construction is that $\overline{R_j}(u)$ is disjoint from $\overline{R_k}(u)$ unless $j = k$. This follows from explicit exclusion of $\overline{R_{<j}}$ from $\overline{R_j}$.

The key property of the construction is that if $w \in F_j(u)$, then only cells in $\overline{R_j}(u)$ differ between executing the algorithm with u and with w. This is similar to the property we required in the old proof, forcing the query to read a cell from $\overline{R_j}$ to distinguish the update sequence u from some other plausible update sequence w.

To prove this key property, we analyze the possible classes of cells that could be used to distinguish an execution with input u and one with input $w \in F_j(u)$, assuming that we do not read any cell from $\overline{R_j}(u)$.

1. A cell in $\overline{R_i}(u)$, for $i < j$. By the definition of $[u_j]$, these are the same for u and w, so these cells cannot be useful in distinguishing u and w.

2. A cell in
$$\overline{R_j}(w) = \left(\bigcup_{v \in F_j(w) \cup \{w\}} R_j(v) \right) \setminus \overline{R_{<j}}(w)$$
Now, by the defintion of $[u_j]$, $\overline{R_{<j}}(w) = \overline{R_{<j}}(u)$. Furthermore, $[u_j] = [w_j]$ by transitivity of equivalence. The fooling set is a function only of the equivalence class, so $F_j(u) = F_j(w)$. But $w \in F_j(u)$, so $w \in F_j(w)$, so the union in the definition of $\overline{R_j}(u)$ includes all terms from the union in $\overline{R_j}(w)$. Thus, $\overline{R_j}(u) \supseteq \overline{R_j}(w)$, and we have already excluded from consideration cells in $\overline{R_j}(u)$.

3. A cell in $\overline{R_i}(u)$, for $i > j$. Supposed that this cell is in $\overline{R_m}(w)$. We have just shown that $m \neq j$, and further that m is not less than j (the first two cases), hence $m > j$. Thus, the cell must have been written before epoch j for both u and w. However, since u and w only differ on the j-th level, the values in this cell for the two instances must be equal.

Thus, if the query needs to distinguish between u and some $w \in F_j(u)$, it must read some element of $\overline{R_j}(u)$. Since picking a random leaf corresponds on the j-th level to picking a random node, and there exists a fooling w for a constant fraction of the nodes on the j-th level, this implies that we must read a cell in $\overline{R_j}(u)$ with probability $\Omega(1)$. The lower bound follows.

The only loose end is why $\overline{R_j}(u)$ really is small (implying that $[u_j]$ is large, which we assumed at the beginning). The idea is the $\overline{R_j}(u)$ includes cells written during a few alternative computation histories (namely, the ones in $F_j(u)$ and u itself). Because we could find very small fooling sets, this is just $O(\lg n)$ times more cells than R_j. □

Now we can explain why we needed to introduce the $\overline{R_j}$'s. Consider some w, an alternative to u on the j-th level. It is possible that u might write some cell in epoch $j+3$, say, that w also writes. But w might rewrite the same cell in epoch j. The problem here is that R_j does not contain this overwritten cell (u doesn't rewrite the cell, only w does). The query algorithm could detect that the write occured by attempting to read the value written in epoch $j+3$. Using the larger sets $\overline{R_j}(u)$ avoids this complication: we are including cells written not only by u, but also by some alternatives to u. Since the fooling sets are small, using these these larger sets $\overline{R_j}$ is not expensive, so that we can still obtain the tight bound.

References

[1] Stephen Alstrup, Thore Husfeldt, Theis Rauhe: *Marked Ancestor Problems*, Proc. 39th Annual Symposium on Foundations of Computer Science (FOCS), p.534, 1998.

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