CS 224: Advanced Algorithms	Spring 2017
Lecture 27 — April 26, 2017	
Prof. Jelani Nelson	*Scribe: Jeffrey Cai*

THE DARK SIDE[1] (lower bounds). We will show e.g. link-cut trees are optimal up to $\lg \lg n$ factor: either link or cut has to take at least $\lg n / \lg \lg n$. It was later shown to be $\lg n$ in [PD04b] (lso ink-cut trees are optimal).

1 Models of computing

Lower bound in what model?

- *Upper bounds*: usually operate in word-RAM model. Instructions operate on values that are w bits long, e.g. $\texttt{ADD}(R_1, R_2, R_3)$ puts $\texttt{val}(R_2) + \texttt{val}(R_3)$ into register R_1. $\texttt{LOAD}(R_1, R_2)$ put contents of $\texttt{Mem}[\texttt{val}(R_2)]$ into R_1. Not all instruction sets are the same: you can simulate instructions with other instructions, but that might cause blowup in running time (e.g. does the architecture have a `POPCOUNT` instruction). If we can prove lower bounds against the strongest model possible, it will work for many different architectures.
- *Lower bounds*: we use the *Cell Probe Model* [Yao78]. In this model, we only charge for memory reads and writes. All other computation is free. Specifics: memory holds w-bit words. We may read/write one word at a time. Assume $w \geq \log_2 n$ (because we want indexing in constant time).

2 Dynamic partial sum

We have:

- An array $A[0...n-1]$, starts with $A[i] = 0$ for all i.
- Support:
  - $\texttt{update}(i, \Delta) : A[i] \leftarrow \Delta$
  - $\texttt{query}(i) : \sum_{j=0}^{i} A[j]$
- Algorithm: can solve with $t_u, t_q = O(\lg n)$ (where $t_u \coloneqq$ update time, $t_q \coloneqq$ query time).
- Maintain a binary search tree whose leaves are $0...n$. Each internal node and leaf has a blue number written on it. Queries are answered by traversing to the leaf and adding all the blue numbers.

[1] is a pathway to many abilities some would consider to be unnatural.

- (Imagine a diagram of a balanced BST with leaves numbered 0...7). Suppose someone says `update`$(2, \Delta)$. Start at the root and walk towards 2. At each step, we know that the entire right subtree is affected by this change. So what we do is, whenever we take a left-child step, we set the right child's blue number to Δ. At the end, when we reach the leaf node for 2, we also set its blue number to Δ. This can be easily modified to handle when we don't start with all zeros.

We now prove a lower bound. We will show that even if we only want

$$\sum_{j=1}^{i} A[j] \mod 2 \quad \left(\text{or } \bigoplus_{j=0}^{i} A[j] \right)$$

we have the bound. This is a strictly easier problem, so it's a stronger lower bound than what we want. We will use the "chronogram technique" [FS89], which is a kind of encoding/compression argument.

If I give you a uniformly random t-bit string, and you have some compression algorithm, the expected number of bits in the result has to be at least t bits. We will show that if a data structure for this problem is faster than our lower bound, then one can use it as a subroutine to construct a compression algorithm that's too good. Specifically, if $\max\{t_u, t_q\} < .0001 \lg n / \lg \lg n$, then we would be able to compress random t-bit strings into expected length $\ll t$:

$$\exists \text{Enc s.t.} \mathop{\mathbb{E}}_{X \sim \{0,1\}^+} |\text{Enc}(x)| \ll t$$

This is impossible by something known as Shannon's source coding theorem, which we won't cover today.

Chronogram method: Issue a sequence of n updates $U_n, U_{n-1}, \ldots, U_1$, then one query Q at the end.

$$U_n \quad U_{n-1} \quad \ldots \quad U_1 \quad Q \quad \text{(time flows left to right)}$$

Bin the updates into "epochs". Epoch 1 consists of the last β updates in time (so $U_1, \ldots, U_\beta$). Epoch 2 consists of the β^2 updates directly before Epoch 1, and so on. The "last" epoch contains β^T updates (so it includes $\ldots U_{n-1}, U_n$, and comes first in time). The number of epochs is $T = \Theta(\log_\beta n)$. Finally, Q will be a query to a uniformly random $i \in \{0, \ldots, n-1\}$.

Specifics of updates: we have a bit array A of size n. Epoch i will update indices of the form $j \cdot \frac{n}{\beta^i}$.

- The first epoch will update $0, n/\beta, 2n/\beta, \ldots$
- The second epoch will update $0, n/\beta^2, 2n/\beta^2, \ldots$
- The last epoch will update every index.

What are the values we update them to? They will be i.i.d. random bits. We use the notation $b_{i,1}, \ldots, b_{i,\beta^i}$ to denote these bit values, where $b_{i,j}$ is the value assigned to the entry $j \cdot (n/\beta^i)$ during epoch i.

For each cell c in memory, we will have a timestamp for c which is the index of the last epoch in which c was written by our data structure D. (Epochs later in time will overwrite the timestamp for epochs earlier in time.) Let C_i be the set of cells associated with epoch i.

Theorem 1.

$$t_q = \Omega\left(\frac{\lg n}{\lg(t_u \cdot w)}\right).$$

This implies for $w = \lg n$ that

$$\max\{t_u, t_Q\} = \Omega\left(\frac{\lg n}{\lg \lg n}\right).$$

Proof. We have, over any operation sequence,

$$\begin{aligned}
t_q &\geq \sum_{\text{cells } c} \mathbf{1}_{D \text{ reads } c \text{ to answer } Q} && \text{right side} = \#\text{ of reads} \\
\mathbb{E}[t_q] &\geq \mathbb{E} \sum_{\text{cells } c} \mathbf{1}_{D \text{ reads } c \text{ to answer } Q} \\
&= \mathbb{E} \sum_{i=1}^{T} |B \cap C_i| \\
&= \Omega(T) && \text{by Lemma 2, to be proved} \\
&= \Omega(\lg_\beta n) = \Omega(\lg n / \lg b).
\end{aligned}$$

□

Lemma 2 (Main Lemma)**.** *Let B be the (random) set of cells read by D to answer Q. The for all epochs i,*

$$\mathbb{E}|B \cap C_i| \geq \Omega(1), \quad (*)$$

when β is set to $(t_u \cdot w)^2$. That is, D has to read at least a constant number of cells from each epoch, in expectation.

Proof. Supose f.s.o.c there exists an i not obeying $(*)$. We will derive a compression scheme for $(b_{i,1}, \ldots, b_{i,\beta^i})$ into $\ll \beta^i$ bits. Both the encoder and the decoder will know:

- $b_{i',j}$ for all $i' \neq i$
- $R_1, ..., R_{\beta^i}$ a sequence of random indices, specifically: partition the indices $0, \ldots, n-1$ into β^i blocks of size n/β^i. Then R_j is chosen uniformly at random from block j.

Now suppose $\mathbb{E}|B \cap C_i| = \alpha$ (where $\alpha \leq \frac{1}{400}$). Then

$$\begin{aligned}
&\mathop{\mathbb{E}}_{j, R_1, \ldots, R_{\beta^i}} |B(R_j) \cap C_i| = \alpha = \frac{1}{\beta^i} \mathop{\mathbb{E}}_{R_1, \ldots, R_{\beta^i}} \sum_{j=1}^{\beta_i} |B(R_j) \cap C_i| \\
\implies &\mathop{\mathbb{E}}_{R_1, \ldots, R_{\beta^i}} \sum_{j=1}^{\beta^i} |B(R_j) \cap C_i| = \alpha \cdot \beta^i
\end{aligned}$$

Then by Markov's inequality, the probability that $\geq 4\alpha\beta^i$ of the R_j's read anything from C_i at all is $\leq \frac{1}{4}$. Hence, with probability $\geq \frac{3}{4}$, at most $4\alpha\beta^i \leq \frac{1}{100}\beta^i$ of the R_j's read from C_i. Call this "Event E".

Encoder: will do the following.

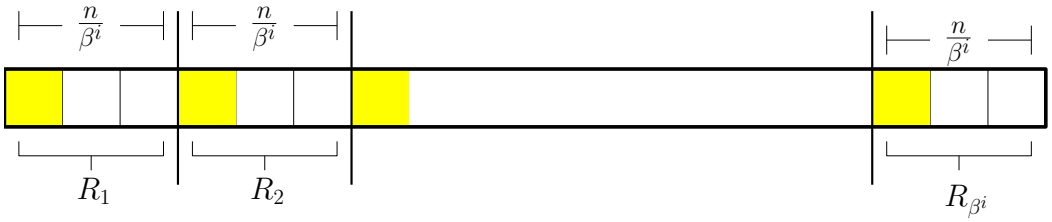

Figure 1: Above is the array A, and its entries are broken into contiguous blocks of size n/β^i. R_j is then chosen to be a uniformly random index in the jth block, i.e. R_j is chosen uniformly at random in the interval $[j \cdot (n/\beta^i), (j+1) \cdot (n/\beta^i))$. The entries in A colored in yellow correspond to the entries updated in epoch i.

- If event E doesn't hold, output 1 followed by $b_{i,1} \dots b_{i,\beta^i}$.
- If event E does hold, output 0 followed by all addresses and contents of $C_{i-1}, C_{i-2}, \dots, C_1$, and a description of which R_j are read from C_i, together with query answers for those R_j.

Then for the encoding length, we have:

$$\mathbb{E}(\text{encoding length}) = 1 + \tfrac{1}{4}\beta^i + \tfrac{3}{4}\left(\sum_{j=1}^{i-1}|C_j| \cdot (w+w) + \lg\binom{\beta^i}{\beta^i/100} + \frac{\beta^i}{100}\right)$$

- We have $\binom{n}{k} \le (en/k)^k$. Therefore, $\le \beta^i/100 \lg(100e) \ll \beta^i/5$.
- $|C_j| \le \beta^j t_u$.

The expected size of the encoding is less than $1 + (3/4)(\beta^i/4 + o(\beta^i))(3/4) + (1/4)\beta^i$ bits, thus by Shannon's source coding theorem, the entropy of the encoded message, M, is $H(M) < \beta^i/2$ bits.

Decoder: We're only halfway there, since we still need a way to (mostly) recover $(b_{i,j})_{j=1}^{\beta^i}$. We will show that a $(1-\beta)$-fraction of the bits $b_{i,1}, \dots, b_{i,\beta^i}$ are completely determined from M. If we wrote a 0-bit first, then trivially all bits are recovered. If we write a 1-bit first, we first obtain the set of queries that read inside C_i and their corresponding answers. From this, we can also deduce the queries that do not read inside C_i. Since the epochs $j \neq i$ are known to the decoder, he can execute the update algorithm of $\mathcal{D}$ on all updates preceeding epoch i. Now, since he knows which queries do not read in C_i, he can run $\mathcal{D}$'s query procedure on those queries. Whenever $\mathcal{D}$'s query procedure requests a memory cell, he checks whether the cell is included in the sets $C_{i-1}, \dots, C_1$ (which is in the encoding). If so, he has the contents and can continue. Otherwise, he knows they are not in sets $C_i, ..., C_1$ and therefore the contents must be the same as they were just before epoch i. But these contents are known because he just ran all updates preceeding epoch i. He can thus finish the query algorithm and now has the answer to every query $R_1, \dots, R_{\beta^i}$.

Now, let us remember which cells are updated in epoch i (the yellow cells in Figure 1). By the end of epoch i, the first yellow cell contains the value $b_{i,1}$, the second yellow cell contains $b_{i,2}$, etc. All the cells which are not yellow in between were written in previous epochs, and thus the decoder knows them. Therefore, by knowing all the answers to the $R_1, \dots, R_{\beta^i}$, the decoder can recover all the bits written in the yellow positions (since the answer to query R_j is the XOR of

all bits up to and including R_j, and the decoder knows all of those bits except for one, the one in yellow!). There is a slight catch: epochs $i-1, i-2, \ldots 1$ do overwrite some of the yellow cells with new bits, and thus for the blocks j where this overwriting happens, the decoder will not learn $b_{i,j}$ from the answers to the queries $R_1, \ldots, R_j$. However, note that the only indices that are overwritten in future epochs are the indices $0, n/\beta^{i-1}, 2n/\beta^{i-2}, \ldots = 0, \beta n/\beta^i, 2\beta n/\beta^i, \ldots$. In other words, only a β-fraction are overwritten, so the decoder does learn the $(1-\beta)$-fraction of the bits that weren't overwritten. But these are still $(1-\beta)\beta^i$ uniformly random bits, and any encoding to recover them should have expected encoding length at least $(1-\beta)\beta^i$, but we are achieving total expected encoding length $\mathbb{E}\,|M| < \beta^i/2$, which is still a contradiction. In other words, one can view this entire scheme as a compression scheme not for *all* the bits $b_{i,1}, \ldots, b_{i,\beta^i}$, but just the $(1-\beta)$-fraction that the decoder actually learns. Alternatively, the encoder can just always write $b_{i,1}, b_{i,\beta+1}, b_{i,2\beta+1}, \ldots, b_{i,(\beta^{i-1}-1)\beta+1}$ as part of the encoding as well, which only adds $\beta^{i-1} = o(\beta^i)$ to the encoding length, so that the decoder can in fact learn all the bits at the end of the day. □

3 Lower bound for dynamic connectivity

Given the lower bound of $\max\{t_u, t_q\} = \Omega(\lg n/\lg\lg n)$ of the previous section for dynamic partial sums over $\mathbb{Z}_2$, we can obtain the same lower bound for dynamic connectivity via a reduction due to [MSVT94]. In fact the reduction holds even if the underlying graph is promised to always be a collection of two vertex-disjoint trees, which implies optimality of the link-cut trees of Sleator and Tarjan [ST83]. In dynamic connectivity we wish to support the following operations, where the graph G starts as the empty graph on n vertices:

- `insert`(u, v): insert the edge (u, v) into G
- `delete`(u, v): delete the edge (u, v) from G, if it exists
- `connected`(u, v): return `True` if u, v are in the same connected component, and false otherwise

The reduction works as follows. Recall we want to solve dynamic partial sums over $\mathbb{Z}_2$, where there is some underlying array $A[0 \ldots n-1]$ and the answer to `query`(i) is $A[0] \oplus A[1] \oplus \cdots \oplus A[i]$. We maintain a graph G on vertex set $[2n+3]$ with the following edge set E:

$$E = \left(\bigcup_{i:A[i]=0} \{(2i+1, 2i+3), (2i+2, 2i+4)\} \right) \cup \left(\bigcup_{i:A[i]=1} \{(2i+1, 2i+4), (2i+2, 2i+3)\} \right).$$

Note that a change to some entry $A[i]$ corresponds to deleting and inserting $O(1)$ new edges in G (to be precise: two edge removals and two edge insertions into G). One can show that $A[0] \oplus A[1] \oplus \cdots \oplus A[i]$ equals 0 iff `connected`$(1, 2n+3)$ is `True` after inserting the edge $(2i+, 3, 2n+3)$ into G. Thus `query` in partial sums can be implemented by one edge insertion, one `connected` query, then one edge deletion in G. Thus the maximum operation time in dynamic connectivity must be at least that for partial sums over $\mathbb{Z}_2$, which is $\Omega(\lg n/\lg\lg n)$ as shown in the last section.

The intuition behind this reduction is the following. If all array entries A had been zero, then E would be $\{(1,3), (3,5), (5,7) \ldots\} \cup \{(2,4), (4,6), (6,8), \ldots\}$. That is, there would be an "even path"

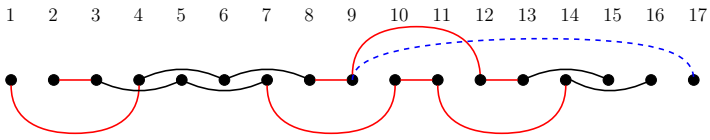

Figure 2: Example reduction from dynamic partial sums over $\mathbb{Z}_2$ to dynamic connectivity. Numbers in the top are vertex ID's. Red edges correspond to 1's in the array A, and black edges are 0's. The underlying array in this example is $A[0 \dots 6] = [1, 0, 0, 1, 1, 1, 0]$. Thus $n = 7$, so the total number of vertices in G is $2n + 3 = 17$. The blue dashed line is inserted before querying `connected`$(1, 17)$ to test whether $A[0] \oplus A[1] \oplus A[2] \oplus A[3] = 0$.

(only touching vertices with even ID's) and an "odd path" (only touching vertices with odd ID's), such that after every index i in A, the even path is always one step ahead of the odd path. These are the black edges in Figure 2. Whenever there is a 1 at some entry in A, it introduces the red edges into G which swap the even and odd paths. That is, it makes the even path the odd path, and the odd path the even path. Note that 1 and $2n + 3$ are both odd numbers, so had all entries of A been 0, they would both be in the odd path together. However, due to the paths swapping at 1's, we essentially want to check that the number of swaps from the beginning up to the edges added by $A[i]$ is even.

4 What's happened since?

- $\max\{t_q, t_u\} = \Omega(\lg n)$ for both dynamic connectivity and partial sums (over $\mathbb{Z}_{2^w}$) [PD04a].
- $\max\{t_q, t_u\} = \Omega((\lg n / \lg\lg n)^2)$ for dynamic weighted orthogonal range counting in $2D$, but only when the weights are $\lg^{2+\varepsilon} n$-bit integers [P07].
- $t_q = \Omega((\lg n / \lg(wt_u))^2)$ for dynamic range counting in $2D$ with $\lg n$-bit weights [L12].
- $\max\{t_u, t_q\} = \tilde{\Omega}(\lg^{1.5} n)$ for parity dynamic range counting in $2D$ [LWY17].
- Many lower bounds known for other problems, e.g. union-find, predecessor, dynamic min spanning forest, planarity testing, dynamic marked ancestor, several computational geometry problems, approximate nearest neighbor in ℓ_p for various p, etc. See [P11] and also the older survey of Miltersen [M99].

References

[FS89] Michael L. Fredman, Michael E. Saks. The Cell Probe Complexity of Dynamic Data Structures. *STOC*, pages 345–354, 1989.

[L12] Kasper Green Larsen. The cell probe complexity of dynamic range counting. *STOC*, pages 85–94, 2012.

[LWY17] Kasper Green Larsen, Omri Weinstein, Huacheng Yu. Crossing the Logarithmic Barrier for Dynamic Boolean Data Structure Lower Bounds. *CoRR* abs/1703.03575, 2017.

[M99] 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.

[MSVT94] Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, Roberto Tamassia. Complexity Models for Incremental Computation. *Theor. Comput. Sci.* 130(1), pages 203–236, 1994.

[P07] Mihai Pătraşcu. Lower bounds for 2-dimensional range counting. *STOC*, pages 40–46, 2007.

[P11] Mihai Pătraşcu. Unifying the Landscape of Cell-Probe Lower Bounds. *SIAM J. Comput.* 40(3), pages 827–847, 2011.

[PD04a] Mihai Pătraşcu, Erik D. Demaine. Tight bounds for the partial-sums problem. *SODA*, pages 20–29, 2004.

[PD04b] Mihai Pătraşcu, Erik D. Demaine. Lower bounds for dynamic connectivity. *STOC*, pages 546–553, 2004.

[ST83] Daniel Dominic Sleator, Robert Endre Tarjan. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.* 26(3), pages 362–391, 1983.

[Yao78] Andrew Chi-Chih Yao. Should Tables Be Sorted? *FOCS*, pages 22–27, 1978.