COMS-E6998: Information Theory in Computer Science **Fall 2017**

Lecture 9: Data Structures (I)

Lecturer: Omri Weinstein *Scribes: Sam Buchanan, Michael Hamilton*

In this lecture we begin our study of data structures. In particular, we will be interested in proving lower bounds on the types of the data structures possible. We begin with an example problem to set the tone for the rest of the lecture.

Example 9.0.1 (Shortest Path)**.** *Given a graph $G = (V, E)$ we wish to compute a data structure to handle shortest path queries over G. In particular we would like a data structure DS which has small space S and which a query algorithm $\mathcal{P}$ can answer questions of the form "What is the shortest path between u and v for $u, v \in V$". Here are two simple data structures for this problem.*

- *Pre-compute the shortest path between every pair of nodes u, v and store the answers in a look up table. Such a table enables $O(1)$ query time (by looking in the table) and requires space $O(|V|^2)$ to store the path length for every pair of nodes.*
- *Store only the underlying graph G. To find the shortest path between u, v the query algorithm runs a classical algorithm (for example Dijkstra's) requiring $O(|V| \log(E))$ query time. The space it takes to store the graph is simply $O(|V| + |E|)$.*

Example 9.0.1 demonstrates the time-space trade-off that occurs when designing data structures. In the example, a look-up tables gives constant query time, but at the cost of quadratic space. On the other hand, doing no pre-processing to the graph gives slower query time $O(|V| \log(|E|))$ but gains in terms of memory. Note that we did not consider the computational time it takes to generate/create the look-up table. In the model of data structures we study, we will allow *arbitrary* computational power in both preprocessing and in interpreting queries. The only limitations in the model will be the space the data structure uses and the number of queries necessary to garner a solution.

In this course we will study two kinds of data structures, static and dynamic.

Definition 9.0.2 (**Static Data Structure**)**.** *A static data structure is given all information up front and supports only queries*

Definition 9.0.3 (**Dynamic Data Structure**)**.** *A dynamic data structure must also support insertions into and deletions from the data structure.*

In this lecture we will focus only on static data structures. There are many static models for (integer) databases, we will work in the most general model for a data base so that our lower bounds are as strong as possible, and hold for as many implementations as possible. We call this model the cell and probe model.

9.1 Introduction to the Static Cell Probe Model

In the cell and probe model we model memory as a size S array and imagine a generic CPU querying the array at various points. The CPU is computational omnipotent, it can perform any task it likes with the information from it's queries. Lets formally define our notation,

1. Let ω be the *word-size*, the number of bits of an element in the data base. Let $\mathcal{U}$ be the universe of words. $\mathcal{U} = \{1, 2, \ldots, u-1\}$ where $u - 1 = 2^\omega - 1$, the maximum expressible integer in that word size.

2. Let S be the array of words (ω bit integers) the data structure contains. Let the $|S| = n$.
3. Let $\mathcal{Q}$ be the set of possible queries on the data base. We will use $q \in \mathcal{Q}$ to represent a single query
4. Let $\mathcal{P}(q, S)$ be the query problem. For example in the *predecessor* problem query the question is, given S and a query $q \in \mathbb{N}$, return the maximum element in S. that is less than q. In our notation, $Pred(q, S) = sup\{s \in S, s \leq q\}$. We will use t to denote the maximum query time $\mathcal{P}$ take on any element $q \in \mathcal{Q}$.
5. We will refer to the data structure as a triple, **(S,t,u)-DS**, it can be thought of as consisting of two parts:
   (a) **Preprocessing** Function $f : \{0,1\}^n \to [w]^S$ which builds the data structure.
   (b) **Query** A function $\mathcal{P}^i : [\omega^{<i}] \to [S]$ which encodes which cell to query next given a length i querying history. $\mathcal{P}$ is an arbitrary function of both the history and the query. Let t be the maximum number of queries D makes.

In the next section we highlight an interesting data structure for solving the Predecessor problem using Van Emde Boas trees.

9.2 The Predecessor Problem via Van Emde Boas (vEB) Trees

In this section we consider upper bounds on the space/time trade-off for data structures which solve the Predecessor problem. In the predecessor problem we're given a set of keys and asked to find the closest predecessor on the line, in S, to our query q. There's been much study on data structures for Predecessor since it has significant industry application. Notably in routing tables for UP packets. Essentially this problem is a nearest neighbor search problem in 1-dimension, the line. In fact, if you solve Predecessor and it sister problem Successor (i.e. find the first element in S that succeeds the query q), the minimum of Pred(q,S), Succ(q,S) is exactly the distance to a point in S.

Predecessor Problem

- **Input:** A set of keys $S \subset \mathcal{U}$, $|S| = n$. We assume $u = 2^\omega = n^{O(1)}$
- **Query:** For any $q \in \mathcal{U}$, find $Pred_{\omega,n}(q) = \max\{x \in S | x \leq q\}$

There is a simple $(S, \log(n), n^{O(1)}) - DS$ for this problem by simply keeping the key set S as a sorted array. The query time for compute a predecessor query given S sorted is simply a binary search on the array, thus $t = \log(n)$. It turns out we can substantially improve our query time by accepting polynomially more space u (recall the interesting case is $|S| = n^{O(1)}$).

Theorem 9.2.1 (vEB Trees). *There exists a deterministic DS for $Pred_{\omega,n}$ with query time $t = O(\log(\omega))$ $= O(\log(\log(u)) = O(\log(\log(n)$ and space $O(\mathcal{U})$.*

The main idea will be to exploit whats known as the *self reducibility* of the Predecessor problem. That is, you can express the Predecessor in terms of smaller Predecessor problems. In particular, imagine cutting up the universe into u/L contiguous blocks of size L (see Figure 9.1). Given a query q, the predecessor of q is either inside $[L\lfloor q/L \rfloor, L\lceil q/L \rceil]$ or it's the maximum element in the first non-empty block preceding block $\lfloor q/L \rfloor$. The recurrence that captures this simple observation is

$$\mathcal{P}(u) \leq \max\{\mathcal{P}(L), \mathcal{P}(\frac{u}{L})\} + O(1) \tag{9.1}$$

Where we abuse notation and use $\mathcal{P}(x)$ to mean the query time to solve a Predecessor problem over a universe of size x. Note if we pick $L = \sqrt{u}$, Equation 9.1 becomes $\mathcal{P}(u) = \mathcal{P}(\sqrt{u}) + O(1)$.

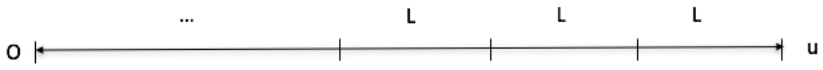

Figure 9.1: The universe of integers $\mathcal{U}$ [u] divided into blocks of size L.

Proof. We will sketch the construction of a vEB tree in accordance with the discussion above. See Figure 9.2 for a depiction of such a tree.

- Let $q \in 2^{\omega} = [u]$ be the query.
- Decompose universe $[u]$ into $\sqrt{u}$ blocks of size $\sqrt{u}$, in each block B_i store it's maximum element.
- Store a *summary* vEB data structure over the $\sqrt{u}$ elements corresponding to where a block B_i is empty (abusing notation slightly, $B_i \in \{0, 1\}$ for $i \in [\sqrt{u}]$). Call this summary vEB DS V.summary.
- On query q, look at the first $\omega/2$ to find its block number, call it B_q. If $B_q = \emptyset$, then $Pred(q) < \min\{\text{Element} in B_q\}$, and recurse through the summary structure to find first non-empty block (recall we've stored the max of each block). Otherwise, recurse on block B_q to find predecessor.

Thus the vEB tree solves predecessor queries like the recurrence in Equation 9.1. □

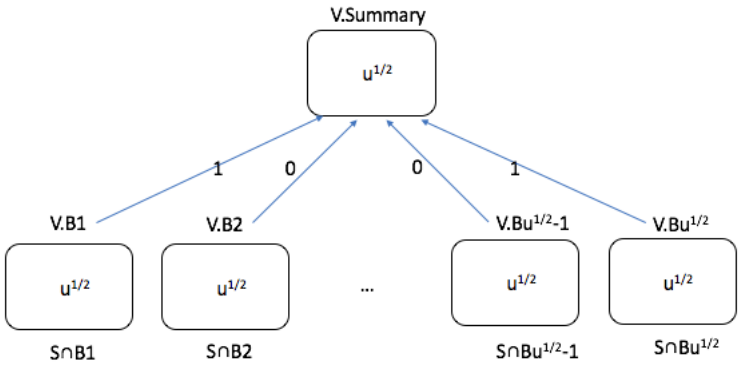

Figure 9.2: A vEB tree on universe of integers $\mathcal{U}$ = [u]

We close this section with some remarks about our upper bound.

Remark 9.2.2. *There exist linear space vEB trees ($S = O(n)$) which solve Predecessor with high probability using query time $t = O(\log\log(n))$ via hashing.*

Remark 9.2.3. *There are other data structures that give strong and slightly different trade-offs on the time and space required. Notably Fusion Trees see [FW93] which use space $O(n)$ and query time $t = O(\log(\omega)^n)$.*

9.3 Lower Bounds for the Predecessor Problem

Our goal will be to prove lower bounds $|S|$ and t for (S,t,U)-DS's that solve problems of interest. As is standard in this class, we will exploit a connection between data structures and communication complexity.

9.3.1 Relation to (One Way) Communication Complexity

There is a natural reduction to an asymmetric communication complexity problem for any (s, t, w) data structure [Mil+98]. As a concrete example, if the problem data is encoded as a graph $G = (V, E)$, we give Bob the graph G and Alice a query $\mathcal{Q}$, say some pair of vertices in V for finding a shortest path in G. Bob constructs the (s, t, w) data structure from his input graph, and Alice simulates the query by sending a message for each cell that the query would have requested in the cell probe model; Bob responds in turn with whatever occupies the address Alice asked for. This is diagrammed in figure 9.3. Thus Alice sends $t \log s$ bits, and Bob sends tw bits. Proving a lower bound on the amount of communication Alice needs to send to Bob to decide $\mathcal{Q}(q, S)$ implies a lower bound on t for size s data structures: in particular, if Alice sends $\Omega(A)$ bits and Bob sends $\Omega(B)$ bits, then $t \geq \min\{A/\log s, B/w\}$.

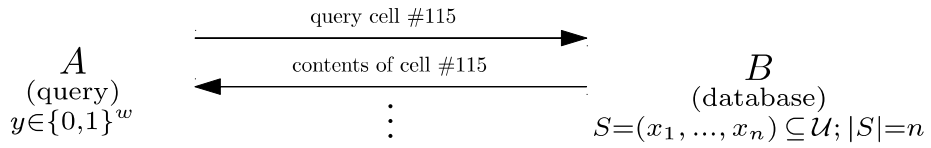

Figure 9.3: Schematic showing communication complexity reduction from data structure problem.

This can be quite bad: if Alice's query is of size $\Theta(n)$, then she sends $\Theta(\log n)$ bits, which implies a $\Omega(1)$ lower bound on t. Better bounds can be obtained using direct-sum-type approaches [PT06]. Ideally we would like to achieve better: something like $\Omega(n^\varepsilon)$ for $\varepsilon > 0$, or $s \geq n^{1+\varepsilon}$. But achieving such bounds has proved challenging for general data structures. Therefore for the remainder of the lecture, we focus on lower bounds for the predecessor problem, which we have shown above has specific fast data structures. We will prove matching lower bounds for our results above in the most interesting regimes $n \gg w$ or $w \gg n$.

9.4 Predecessor Search LB via Round Elimination

In this section, we will prove the following theorem.

Theorem 9.4.1. *For every (s, t) data structure for $\mathsf{pred}_{w,n}$ with $s = n^{O(1)}$, one has*

$$t = \Omega\left(\min\left\{\frac{\log w}{\log\log w}, \log_w n\right\}\right).$$

In the two ordinary regimes where either the word size is large compared to the set size or vice versa, this theorem asserts the essential optimality of fusion trees and van Emde Boas trees (respectively). The proof will be based on the results of Sen and Venkatesh [SV03], and it makes crucial use of the round elimination lemma introduced by Miltersen et al. [Mil+95].

More recently, Patrascu and Thorup proved a more complete characterization of the tradeoff between the two bounds in our theorem as one interpolates between the two aforementioned regimes of (w, n) [PT06]. The following is a corollary of their main theorem: for any (s, t) data structure, $s = \tilde{O}(n)$ implies

$$t = \Omega\left(\frac{\log w}{\log\left(\frac{\log w}{\log\log n}\right)}\right).$$

To prove the theorem, we will study a simplification of the predecessor problem and use ideas from communication complexity. The simplification is to the colored predecessor problem, denoted $\overset{\oplus}{\mathsf{pred}}_{w,n}$; here, each of the n items in S is assigned one of two colors, and the goal is to output the color of the predecessor of the queried point. This implies a lower bound on $\mathsf{pred}_{w,n}$.

Let us first make a definition that captures the asymmetry present in data structures problems in the setting of a two-party communication problem. Below, we assume that Alice sends messages of length a, and Bob sends messages of length b.

Definition 9.4.2. *For any two party CC problem of computing $f(A, B)$, let $f^{(k)}$ denote the problem where Alice gets $A_1, A_2, \ldots, A_k$, Bob gets B, some index $i \in [k]$, and $A_{<i}$, and the goal is to output $f(A_i, B)$ with a protocol where Alice speaks first.*

The basic idea of the proof is sketched as follows. The family of communication problems we have just defined has an interesting characteristic: Alice needs to know the index i to compute $f^{(k)}$ exactly, but Bob holds i and Alice speaks first. Thus her first message is wasted in terms of the total length of the protocol. The round elimination lemma will allow us to use this idea to reduce any t round protocol for $f^{(k)}$ where Alice speaks first to a $t-1$ round protocol where Bob speaks first that has probability of error not much larger than that of the original t round protocol (depending on s). Then we will exploit the self-reducibility of the (colored) predecessor problem as we did in the upper bound proofs, along with a clever embedding of the problem into the setting of the above definition that uses ideas from vEB trees and fusion trees, to argue that the t-round protocol can be reduced to a 0-round protocol with probability of success greater than $\frac{1}{2}$. But with zero rounds of communication, Alice can do no better than probability of success $\frac{1}{2}$ (using a coin flip), and this contradiction will establish the theorem.

We now formalize the round elimination lemma.

Lemma 9.4.3 (Round Elimination)**.** *For any f a two-party CC problem, suppose there exists a protocol for $f^{(k)}$ with error δ in t messages where Alice speaks first. Then there exists a protocol with $t-1$ messages and error probability at most $\delta + O(\sqrt{a/k})$ for f where Bob speaks first.*

Proof Sketch. If $I \in_R [k]$ is a uniform random variable denoting the index Bob holds, then $I(M_A; A_I | A_{<I}) \leq a/k$. Now use Pinsker's inequality to get

$$\mathop{\mathbb{E}}_{A_1,\ldots,A_k}\left[\|(M_A|\bar{A}) - (M_A|A_{<i})\|_1\right] \leq O(\sqrt{I(M_A; A_I|A_{<I})}) \leq O(\sqrt{a/k}).$$

We can use this fact to construct the protocol where Bob speaks first having the desired error. The full proof can be found as Lemma 4 in [SV03]. □

Proof (of Theorem 9.4.1). We will show that $t = \Omega(\min\{\log_a w, \log_b n\})$ in the setting of the theorem (for colored predecessor). For observe that when our data structure uses polynomial space, $a = O(\text{poly}\log n)$ and $b = w$ as usual, and plugging this into this bound yields

$$t = \Omega\left(\min\left\{\frac{\log w}{\log\log n}, \log_w n\right\}\right).$$

Then note that this bound is maximized when these two terms are equal; this can be shown to imply $\log\log n \sim \log\log w$, and hence the bound stated in our theorem.

Now suppose we have a t-round protocol for colored predecessor. Our goal is to apply $2t$ round eliminations to produce a zero-round protocol with error probability at most $1/3$.

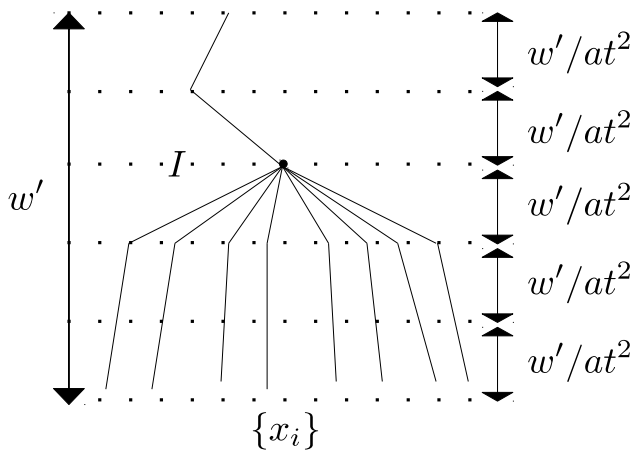

Figure 9.4: Representation of elimination of Alice's round; note the similarity to vEB trees. The database contains items that only differ within the I-th block of the partitioned words; we don't care about their values after block I, because the predecessor problem's solution depends solely on the I-th block. This picture makes it clear how the word size changes after the reduction.

First, we eliminate an Alice $\to$ Bob round. Suppose that Alice's input A has w' bits. To apply the round elimination lemma, we introduce the following distribution: we break the input into k chunks of size $\Theta(at^2)$, which we denote $A_1, \ldots, A_k$, and we suppose that the data structure for colored predecessor that Bob holds contains items that only differ on bits in block $I \in_R [k]$; see figure 9.4. Bob also holds $A_{<I}$. This is a valid distribution on inputs, and it induces a $f^{(k)}$ problem as we have defined them. By assumption Alice and Bob can compute the color of the predecessor of A in the dataset B in t rounds; but in our setting, the self-reducibility of the predecessor problem implies that computing $f(A, B)$ is equivalent to computing the predecessor of A_I, which is the same as computing $f^{(k)}(A_I, B')$, where B' is an appropriate restriction of the data structure B that Bob holds. Thus round elimination is applicable. By the round elimination lemma, this leads to an error increase of $O(1/t)$, and we can choose the constant on at^2 so that the error increase is ε/t for some fixed constant ε. (We will take this as an implicit truth in the discussion to follow.) To wrap up, we just need to identify how the restriction changes the problem parameters (n, w). More precisely, the data structure B' contains words of size $w'/\Theta(at^2)$.

Now we consider elimination of a Bob $\to$ Alice round. Suppose the number of elements in the data structure B that Bob holds is n', each with word length w'. As before, we break B into $k = \Theta(bt^2)$ chunks, so that once the round elimination lemma is shown to be applicable we have another $O(1/t)$ increase in the error. The distribution we define on the inputs in this setting uses the same idea as in the Alice to Bob round: we think of Bob's input B being split into n'/bt^2 subtrees, each of which encodes a distinct predecessor problem (see figure 9.5; Alice is given exact knowledge of the prefix tree, and we define her query to choose a random block amongst the k chunks Bob has set up. Computation of the color of the predecessor of Alice's query can be accomplished as soon as Bob knows which block is relevant; thus, we have again a $f^{(k)}$ problem (which solves the original f problem, since Alice knows the prefix tree), and the round elimination lemma is applicable. In this case, the reduced problem has word length $w' - \log(bt^2)$ and data structure size n'/bt^2. Observe that each Alice $\to$ Bob round elimination reduces the word length by much more, so that this reduction is ultimately inconsequential.

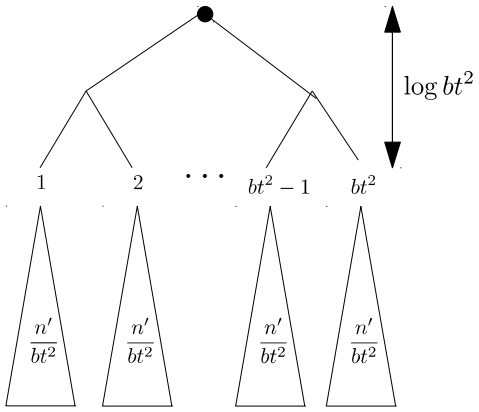

Figure 9.5: Representation of elimination of Bob's round; note the similarity to fusion trees.

Now we have a scheme to continually eliminate rounds in the protocol. A full round elimination leads to

$$w' \to w'/at^2 - \log(bt^2) \approx w'/at^2$$
$$n' \to n'/bt^2.$$

We run into trouble when the word length hits $\log w$ (or, say, 2) or when the number of elements in the dataset hits 2, since at this point we can no longer divide up Alice/Bob's inputs. t round eliminations produces a protocol with probability of success greater than $\frac{1}{2}$, since we can choose constants in the round eliminations appropriately and we incur $O(1/t)$ error increases with each round elimination. But this is absurd, because such a protocol has 0 rounds and cannot beat random guessing. Therefore our scheme must not have eliminated all the messages. This implies

$$t = \Omega\left(\min\left\{\log_{at^2} w, \log_{bt^2} n\right\}\right).$$

We claim that $at^2 = O(a^3)$, because there exists a data structure that achieves query time $O(\log n)$, namely binary balanced search trees, and because $a = \Theta(\log n)$ to store the data, which has size $n^{O(1)}$. Furthermore, we claim that $bt^2 = O(b^3)$, since $t = O(\log w)$ by the existence of vEB trees, and because $b = w$. Thus the bound we have obtained is the same as

$$t = \Omega\left(\min\left\{\log_a w, \log_b n\right\}\right),$$

which is what was claimed. □

References

[FW93] Michael L Fredman and Dan E Willard. "Surpassing the information theoretic bound with fusion trees". *J. Comput. System Sci.* (Dec. 1993).

[Mil+95] Peter Bro Miltersen et al. "On Data Structures and Asymmetric Communication Complexity". *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*. STOC '95. New York, NY, USA: ACM, 1995, pp. 103–111.

[Mil+98] Peter Bro Miltersen et al. "On Data Structures and Asymmetric Communication Complexity". *J. Comput. System Sci.* 57.1 (Aug. 1998), pp. 37–49.

[SV03] Pranab Sen and S Venkatesh. "Lower bounds for predecessor searching in the cell probe model" (Sept. 2003). arXiv: `cs/0309033 [cs.CC]`.

[PT06] Mihai Patrascu and Mikkel Thorup. “Time-Space Trade-Offs for Predecessor Search” (Mar. 2006). arXiv: `cs/0603043 [cs.CC]`.