CS 229r Information Theory in Computer Science April 16, 2019

Lecture 21

Instructor: Madhu Sudan *Scribe: Juspreet Singh Sandhu*

1 Overview

We define the (dynamic) Data-Structure model we wish to analyze, and prove a (weak) version of the prefix-sum lower bound. The references for the lecture are:

- Fredman & Saks, '89 - The majority of the proof in this lecture gives an analysis that proves a weaker version of Theorem-3 in this paper (for the *Partial sums* problem).
- Kasper Green Larsen, SITOC 2018, Part-1, Kasper Green Larsen, SITOC 2018, Part-2
- Patrascu & Demane, 2009

2 Dynamic Data-Structures & Cell-Probe Model

We lay down an (informal) definition of a dynamic Data-Structure, followed by the set of operations we wish to support. Then, we define the data-structure problem in its most abstract sense and state the description of the Cell-Probe model as a concrete model to asses and measure the complexity of queries. Finally, we end with a proof for a lower bound on the Array Prefix-Sum problem.

2.1 Dynamic Data-Structures

Informally, (dynamic) Data-Structures are objects that maintain data under dynamic updates; answer queries while using "small" amount of space, and process updates (and queries) efficiently.
Usually the operations we wish to support are:

- Update operations: *Insert(i, S)*, *Delete(i, S)*
- Query operation: *Query(i)*

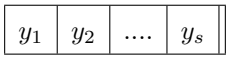
To that end, we define our Data-Structure problem in its most abstract sense as:

Definition 1 (Data-Structure Problem)**.** Given a sequence of query and update operations $\sigma_1, .., \sigma_m$, compute $\textbf{Query}(\sigma_1, .., \sigma_{m-1}; \sigma_m)$ efficiently.

The computation of the query function above may be seen as computing the state that the data-structure should be in as a function of its present state $s(\sigma_{m-1})$ and the last received query σ_m. We wish to find lower bounds (space and time) on the computation of this query function for different problems.

2.2 Cell-Probe Model

The Cell-Probe model was proposed by [Yao, '79] as a way to measure the time complexity of (dynamic) Data-Structures where the only cost payed is the total number of memory accesses.



There are a total of s-cells in the array, and every cell holds a number $y_i \in \{-2^w, 2^w\}$, which stores the data pertaining to location i and a pointer to the next cell $i+1$. Denote by l the length of a query σ_i. Then, we enforce that $w > l$ and $w > \log_2(s)$ (so that we can have pointers to any arbitrary cell).
It is important to note that every query σ_i will lead to a set of accesses (read/write): $i_1 \to i_2 \to ... \to i_t$. These accesses are allowed to be **adaptive**:

Definition 2 (Cell-Probe Adaptive Accesses)**.** $\forall \sigma_i$, the set of accesses $i_1 \to .. \to i_t$ are computed as: $i_k = g(i_{<k}; \sigma_i)$, $\forall k \in [t]$.

3 Examples & Array Prefix-Sum problem

Having defined our Data-Structure problem and the Cell-Probe model via which we will establish lower bounds on our query function, we state a few examples of problems for which we wish to establish lower bounds.

3.1 Set Membership

- *Given*: A set $S \subseteq [U] = \{-2^l, .., 2^l\}$, such that, $|S| \leq n$
- Query/Update operations:
  - *Insert(i, S)* $\to S \leftarrow S \cup \{i\}$
  - *Delete(i, S)* $\to S \leftarrow S - \{i\}$
  - *Query(i, S)* $\to \mathbb{1}_{i \in S}$

We can solve this using a Search tree with $O(n)$ Space Complexity and $O(\log_2(n))$ Time Complexity (simply use a balanced tree with the right subtree containing elements larger than the node, and the left subtree containing elements smaller than the node). We can also use a Hash-Function based approach that has $O(n)$ Space Complexity, and $O(1)$ Time Complexity (Armortized).

3.2 Partial Match Retrieval

- *Given*: A set $S \subseteq [U] = \{0,1\}^l$, such that, $|S| \leq n$
- Query/Update operations:
  - *Insert(i, S)* $\to S \leftarrow S \cup \{i\}$
  - *Delete(i, S)* $\to S \leftarrow S - \{i\}$
  - *Query(i, S)*, $i \in \{0, 1, *\}^l$:
    - 1, if $\exists y \in S$, st, $\forall i$ with $x_i \in \{0,1\}$, $x_i = y_i$
    - 0, otherwise

Exercise 3. *State the Set-Membership and Partial-Match Retrieval problems in the* ***Cell-Probe*** *model. More formally, define* $\textbf{\textit{Query}}_{SET-MEM}(\sigma_1, .., \sigma_{m-1}; \sigma_m)$ *and* $\textbf{\textit{Query}}_{PART-MATCH}(\sigma_1, .., \sigma_{m-1}; \sigma_m)$.

It turns out that outside of the trivial solution which requires $O(n)$ Space Complexity and Time Complexity (because of the Query function), getting a better solution is an open question:

Open Problem: *Does there exist a solution for Partial Match retrieval that requires* $O(poly(n))$ *Space Complexity and* $\tilde{O}(\log_2(n), l)$ Time Complexity ?

3.3 Array Prefix-Sum

- *Given/Goal*: Maintain an array $A = A[0], .., A[n-1]$
- Query/Update operations:
  - *Update(i, Δ)*: $A[i] \leftarrow \Delta$
  - *Query(i)*: return $\sum_{j \leq i} A[j]$

This problem presents a few solutions, all of which (as we shall see in Section-4) respect the [Fredman-Saks, '89] lower bound. A slight modification is that the paper defines the *Update* operation as addition as opposed to an over-riding operation. However, we assume that an update essentially over-rides the value at that position. We define t_U to be the Time Complexity of the update operations, and t_Q to be the Time Complexity of the query operations.

- [Array] $t_U = O(1)$, $t_Q = O(n)$
- [Binary Tree] $t_U = O(\log_2(n))$, $t_Q = O(\log_2(n))$

Exercise 4. *Implement the Binary-Tree solution to the* ***Array Prefix-Sum*** *problem. In particular, construct and analyze the Query(i) function.*
Hint: The Query(i) function can use a BST with array elements as leaves, and each node storing the sum of the elements of its subtree.

4 Array Prefix-Sum lower bound & Proof

We begin by stating the strong theorem (that we shall not prove), followed by the weaker version (that we shall prove).

Theorem 5 (Strong Prefix-Sum Lower Bound)**.** *For Array Prefix-Sum queries,* $t_U + t_Q \geq \frac{\log_2(n)}{\log_2(\frac{w}{l})}$

Theorem 6 (Weak Prefix-Sum Lower Bound)**.** *For Array-Prefix Sum queries,* $t_Q \geq \frac{\log_2(n)}{\log_2(\frac{t_U \cdot w}{l})}$

We first sketch an overview of how the proof will proceed, and then provide the details by splitting into 2 parts:

- **Fredman-Saks Chronogram**: In order to reason about the most adversarial queries, we construct queries that consist of j geometrically spaced batch updates (epochs) followed by a query to a random index i. These updates are chosen uniformly at random, and therefore, we argue that they are independent of each other, leading to a lower bound on t_Q that needs at-least $\Omega(1)$ probes to elements changed by each of the epochs and a total of j probes.
- **One-way Communication Protocol**: In order to show that the entropy of a randomly selected batch (epoch) is high even when we know every other epoch, we construct a one-way communication protocol (essentially a compressing algorithm) that violates Shannon's source coding theorem if we assume otherwise, and this leads to the lower bound above, which implies Theorem 6.

We will first define the Fredman-Saks chronogram, then define 2 important sets we have to reason about, and finally state a helper lemma (Lemma 9) which will imply Theorem 6.

4.1 Fredman-Saks Chronogram

Let the number of epochs be roughly $\frac{\log_2(n)}{\log_2(\beta)} = \log_\beta(n)$, where $\beta = \mathcal{O}(t_U)$. The reason for this choice of β will become clearer by the end of Section 4.2. Strictly speaking, we will want $\beta \geq \frac{t_U \cdot w}{l}$. However, for convenience we will swallow the word-length and element bit-length factors into the Big-O notation. We will also safely assume that $\beta \geq 1$. Therefore, we have:

$$j = \#(epochs) \approx \log_\beta(n) = \frac{\log_2(n)}{\log_2(\beta)} \approx \frac{\log_2(n)}{\log_2(\frac{t_U \cdot w}{l})}$$

Note that $\beta^j \approx \beta^{\log_\beta(n)} = n$. This observation is crucial so as to order the updates to be successively decreasing by a factor of β for a total run of j epochs, where the first epoch updates every element and the last epoch updates a single element. More formally, we construct the epochs in the following manner:

- U_j: $A[i] \leftarrow \Delta_{j,i}$, $\forall i \in \{0\}$
- U_{j-1}: $A[\frac{n}{\beta^{j-1}} \cdot i] \leftarrow \Delta_{j-1,i}$, $\forall i \in \{0, .., \beta^{j-1} - 1\}$
- U_{j-2}: $A[\frac{n}{\beta^{j-2}} \cdot i] \leftarrow \Delta_{j-2,i}$, $\forall i \in \{0, .., \beta^{j-2} - 1\}$
- ...
- U_2: $A[\frac{n}{\beta^2} \cdot i] \leftarrow \Delta_{2,i}$, $\forall i \in \{0, .., \beta^2 - 1\}$
- U_1: $A[\frac{n}{\beta} \cdot i] \leftarrow \Delta_{1,i}$, $\forall i \in \{0, .., \beta - 1\}$
- U_0: $A[i] \leftarrow \Delta_{0,i}$, $\forall i \in \{0, .., n - 1\}$

Two important points before we proceed further:
- We select a query $q \sim_r [n]$ (that is, we select an index uniformly at random to query) after the updates.
- Every update $\Delta_{l,i}$ is chosen uniformly at random from $\{-2^l, 2^l\}$, and represents the update at Epoch l to position i.

The cell-probe may now thought of as being j-colored, where every cell-probe is assigned a color corresponding to its last update. The epochs are deterministically spaced-out and geometrically decreasing in size, but each epoch (modulo the first) will rewrite certain cells. We are interested in the color assigned to a particular cell y_i in the last epoch to update it. In keeping with this, we introduce a few elementary definitions:

Definition 7. $C(i) = \{y_t \mid \text{Color of } y_t = i, t \in [2^w] = \mathcal{O}(n)\}$

Note that we has asserted $l = \log_2(n) = \Theta(w)$. This is a common setting, as we are merely asserting that we should be able to have a pointer to any address in the Data-Structure representation D (assuming it stores n elements) and that the word word size should be a constant multiple of the number of bits required to store an element.

Definition 8. $p(q) =$ Set of cells probed on query q

4.2 One-way Communication analysis

In order to prove our main theorem (Theorem), it suffices to prove the following helper lemma:

Lemma 9.

$$\forall i \in [j], \mathop{\mathbb{E}}_{U_1,..,U_j,q}[|C(i) \cap p(q)|] = \Omega(1)$$

provided $\beta \geq \frac{t_U \cdot w}{l}$

Proof: Before we formally begin a sketch of the analysis to prove this statement, observe that Theorem 6 indeed follows from Lemma 9:

$$t_Q \geq \mathbb{E}[|p(q)|] \geq \sum_{i=1}^{j} \mathop{\mathbb{E}}_{U_1,..,U_j,q}[|C(i) \cap p(q)|] = j \cdot \Omega(1) \geq (\frac{\log_2(n)}{\log_2(\frac{t_U \cdot w}{l})})$$

Intuitively, this statement should be true as an element $A[k]$ (for some $k \in [n-1]$) with color i depends on elements chosen uniformly at random ($\Delta_{k,x}, \forall k \in [i-1], x \in [n-1]$) in epochs $U_0, .., U_{i-1}$, and so we need all prior information. Furthermore, none of the cells ahead can give complete information, as the epochs decrease geometrically in size, and the total number of elements changed at all epochs after i are smaller in size compared to the ones colored at epoch i (property of geometric sums). Formally:

Note that after an update i, the number of cells changed are $C(i)$. Let $|U_i|$ denote the number of elements changed by epoch U_i. Then, it follows that:

$$|C_i| \leq (t_U \cdot w) \cdot |U_i| \leq (t_U \cdot w)\beta^i$$

where, we use $|U_i| = \beta^i$, by the Epoch construction. The size of all $C_{>i}$ updated by epochs $U_{>i}$ is bounded by their geometric sum as:

$$|C_{>i}| = \sum_{k=i+1}^{j} |C_k| = \mathcal{O}(\beta^{i-1} t_U \cdot w)$$

For sufficiently small β, we want to argue that the amount of information in U_i is more than the amount that anything ahead can tell us, asserting that we need information from $U_{<i}$. To this end, note that:

$$H(U_i) = \beta^i \cdot l$$

since each $\Delta(i,k)$ (with $k \in [n-1]$) was chosen uniformly at random. In fact, since U_i chosen in a way such that it is independent form the others:

$$H(U_i|(U - \{U_i\})) = \beta^i \cdot l$$

Now we consider an encoding argument which we construct as a one-way protocol (Π) between Alice (encoder) and Bob (decoder). In this game, Alice receives all the epochs $U = \{U_k\}_{k=1}^{\log_\beta(n)}$ and Bob receives all but one, say U_i. The goal is for Alice to communicate the minimum number of bits to help Bob construct U_i.
Now, let us we assume that $\exists\ \varepsilon$, such that:

$$\mathop{\mathbb{E}}_{U_1,..,U_j,q}[|C(i) \cap p(q)|] \leq \varepsilon$$

As we see below, with the above assumption we can construct Π in a manner, such that:

$$CC(\Pi) = |C_{>i}| < \beta^i \cdot l = H(U_i|U - \{U_i\})$$

(for large enough β) and that gives a contradiction, because Bob decodes U_i with fewer bits than its entropy (violating Shannon's source coding theorem).

4.2.1 Protocol Π

- *Alice*- Runs all queries to construct the color partitioning of elements: $U_i \in U \to \{C_1, .., C_{\log_\beta(n)}\}$.
- *Alice*- Sends all $C_{>i}$ to Bob. Note that this has cost $|C_{>i}| = \mathcal{O}(\beta^{i-1} t_U \cdot w) = \mathcal{O}(\beta^{i-1} t_U \cdot l)$.
- *Bob*- Runs all queries to enumerate a subset of the coloring partition: $\{U_1, ., U_{i-1}\} \cup \{U_{i+1}, ., U_{\log_\beta(n)}\} \to \{C_1, ., C_{i-1}\} \cup \{C_{i+1}, ., C_{\log_\beta(n)}\}$. Bob subtracts changes in C_{i-1} from C_{i+1} on relevant indices $\frac{n}{\beta^i}$ and takes a majority vote for every query to construct C_i.

- *Alice*- Knowing which votes Bob will fail on, merely sends the corrections on those votes to Bob, which allows him to reconstruct U_i completely. This has cost $\approx 2\varepsilon \cdot \beta^i \cdot l$ (see **1** below), which can be ignored for a sufficiently large choice of l.

There are 2 keys observations about Π:

1) The probability that a query q will probe a cell in C_i is low. By our assumption, Bob fails on $\leq \varepsilon \cdot n$ queries. Therefore, he is correct with probability $\geq (1 - 2\varepsilon)$, leading to a negligible overhead in communication cost for Alice.

2) The message length $CC(\Pi) = |C_{>i}| = \mathcal{O}(\beta^{i-1} t_U \cdot l)$ can be made smaller than $H(U_i|U - \{U_i\})$ for sufficiently large β. In particular, if $\beta \geq \frac{t_U \cdot w}{l} = \mathcal{O}(t_U)$, we have that:

$$CC(\Pi) = |C_i| < H(U_i|U - \{U_i\})$$

This yields a contradiction to the Shannon source coding theorem.

Therefore, $\mathop{\mathbb{E}}_{U_1,..,U_j,q}[|C(i) \cap p(q)|] \geq \varepsilon$.
As we showed in the beginning, this gives us the desired lower bound for Theorem 6.

QED.