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 end with a description of the Cell-Probe model as a concrete model to asses and measure the complexity of queries and prove lower bounds about them.

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 $\mathbf{Query}(\sigma_1, .., \sigma_{m-1}; \sigma_m)$ 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.

y_1	y_2	....	y_s

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 being larger than the node, and the left subtree being 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

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] $t_U = O(\log_2(n))$, $t_Q = O(\log_2(n))$

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 3 (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 4 (Weak Prefix-Sum Lower Bound). *For Array-Prefix Sum queries,* $t_Q \geq \frac{\log_2(n)}{\log_2(\frac{t_U \cdot w}{l})}$

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

- *FREDMAN-SAKS Chronogram*: Constructing a chronogram to partition and initialize $A[0], .., A[n-1]$ by using over-writes to geometrically decreasing buckets of the array (iterated over a finite number of epochs), and then choosing a query $q = \sigma_i \sim_r [n]$.
- *Lower bound CC_ε(Compute($A[0]..A[n-1]$))*: Partitioning the epochs $U_1, .., U_j$ in the chronogram into 3 buckets A, B, C, where Alice gets buckets A, B, C, Bob gets buckets A, C, and constructing a one-way communication protocol in which Alice sends Bob the minimal number of bits possible so that Bob can compute the correct values of the original array (in the bucket B). Here, $j \approx \frac{\log_2(n)}{\log_2(\beta)} = \#$ of epochs, where $\beta \geq t_U$.

To make our ultimate claim, we will first define 2 sets and state a helper lemma, the claim to which will imply our **Theorem-4** (with a little more analysis).

4.1 Fredman-Saks Chronogram

Let the number of epochs $j \approx \frac{\log_2(n)}{\log_2(\beta)}$ and $\beta \geq t_U$.
Construct the j epochs in the following manner:

- U_0: $A[i] \leftarrow \Delta_{i,1}$, $\forall i \in \{0, .., (n-1)\}$
- U_1: $A[\beta \cdot i] \leftarrow \Delta_{i,2}$, $\forall i \in \{0, .., (\frac{n}{\beta} - 1)\}$

- U_1: $A[\beta \cdot i] \leftarrow \Delta_{i,3}$, $\forall i \in \{0, .., (\frac{n}{\beta^2} - 1)\}$
- ...
- U_1: $A[\beta \cdot i] \leftarrow \Delta_{i,j}$, $\forall i \in \{0, .., (\frac{n}{\beta^j} - 1)\}$

We then select the query $q \sim_r [n]$. Assume that our cell-probe is now j-colored, such that we color every cell that was updated in a particular epoch with a color. Note that our epochs are geometrically decreasing in size, but we will rewrite certain cells multiple times, and therefore, we will be interested in the color assigned to a particular cell y_i in the last epoch to update it. We now introduce a few elementary definitions:

Definition 5. $C(i) = \{t \mid \text{Color of } y_t = i\}$

Definition 6. $p_(q) =$ Set of cells probed on query q

4.2 One-way Communication analysis

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

Lemma 7. $\forall i \in [j], \underset{U_1,..,U_m,q}{E}[C(i) \cap p(q)] = \Omega(1)$, *provided* $\beta \geq \frac{t_U \cdot w}{l \cdot \varepsilon}$

The proof for this lemma follows from an analysis of the one-way communication protocol, based on choosing $A = U_1, .., U_{i-1}$, $B = U_i$ and $C = U_{i+1}, .., U_j$. We let the status of the array at $U_{i-1} = D$ and $U_i = D^{'}$. Initially, Bob sets $D = D^{'}$, but then runs the query function q (some fixed number of times: $O(\frac{1}{\varepsilon^2})$ so as to obtain a fixed error ε) and updates the cells in U_i.
Bob is allowed to make errors, but since Alice knows A, B, C, she can run the same query q and predict for what $j \in U_i$ will Bob make an error. This is *precisely* the set of bits: $A_{j_1}, .., A_{j_m} \in U_i$ she sends to Bob (in addition to the information about the updates in C) so that he can correctly construct B. Since Bob knows A and C, and the query function is adaptive, with Alice's correcting bits he can construct B.
The probability that Bob makes an error is: $Pr[error] \leq 2\varepsilon \frac{n}{\beta^i}$
Note that as B is chosen to be U_i independently, it has maximum entropy even when conditioned on A and C. So:
$H(B \mid A, C) = H(B) = |U_i| \cdot l = \frac{nl}{\beta^i}$
The last remaining part of the puzzle is the number of bits that Alice sends to Bob, i.e., the message length:
$|m| = |C| \cdot t_U(w + O(\log_2(s)) +$ correcting bits
Here, the updates are the scaled by the size of each word that can be stored in the cell-probe, as well as some auxiliary bits for pointing into the correct indices. Note that, as $\beta > 1$, $|U_i| \geq \sum_{p=i+1}^{j} U_p$. Given that the length of the query is l, the total message length is the information about the updates plus the set of corrections:
$|m| = |C| \cdot t_U(w + O(\log_2(s)) + 2\varepsilon \frac{n}{\beta^i} \cdot l = O(\frac{n \cdot t_U \cdot W}{\beta^{i+1}}) + 2\varepsilon \frac{n}{\beta^i} \cdot l$
By choosing $\beta \geq \frac{t_U \cdot W}{(1-2\varepsilon)l}$, we obtain $\underset{U_1,..,U_m,q}{E}[C(i) \cap p(q)] \geq \varepsilon$.
Since the expectation is over positive random variables, we see that this expectation is $\Omega(1)$, $\forall i \in [j]$.

To complete the proof, we use our inequality over β to obtain that:
$t_Q \geq \frac{\log_2(n)}{\log_2(\beta)}$
By making the ε sufficiently small, we can substitute $\beta \approx \frac{t_U \cdot w}{l}$. This implies that: $t_Q \geq \frac{\log_2(n)}{\log_2(\frac{t_U \cdot w}{l})}$
This proves the statement in **Theorem-4**.
QED