Advanced Data Structures Jan-Apr 2012

Lecture 18 — April 5, 2012

Lecturer: Abhishek Dang *Scribe: Sudeshna Kolay*

1 Overview

We have already seen a $O(\log^2 n)$ amortised deterministic algorithm for testing connectivity in fully dynamic graphs.

In this lecture, we will see that the fully dynamic connectivity problem requires $\Omega(\log n)$ time per operation, even with amortisation. This was studied in the paper [1].

2 Proof Strategy

We will first be showing that the Partial Sums problem requires $\Omega(\log n)$ time per operation, by exhibiting a problem instance that demands at least the lower bound on time. From there, using a notion of nondeterminism, we will look at an instance for dynamic connectivity in graphs where the lower bound has to hold.

3 Partial Sums Problem

Input : An arbitrary group G, array $A[1 \dots n]$ initialised to the group identity (say 0). **Operations allowed in query sequence:**

1. $Update(k, g)$ which modifies value of $A[k]$ by element $g \in G$
2. $Sum(k)$ which returns the partial sum of elements in indices $\leq k$.

Let $\delta = \log |G|$, $w =$ cell size.

We will describe the hard instance by a permutation π on $[n]$ and a sequence $(g_1, \dots g_n) \in G^n$. The g_i's are chosen uniformly at random. We will consider the query sequence $Sum(\pi(1)), Update(\pi(1), g_1)$, $\dots Sum(\pi(n)), Update(\pi(n), g_n)$. We will also think of the pair $Sum(\pi(i)), Update(\pi(i), g_i)$ to be occurring at time i, $i \in [n]$.

3.1 Information Transfer

Definition : The information transfer $IT(t_0, t_1, t_2)$, where $t_0 < t_1 < t_2$ is the number of cells read in [t_1,t_2] that were last written in [t_0,t_1].

For the time being let us fix all the elements of G that are updated before time t_0 and after time t_2. We will call this collection G'. Notice that this collection can be represented by a random variable X. Here, $X = G'$.

Definition : $A_{[t_1,t_2]}$ is the collection of partial sums queried for in the interval [t_1,t_2].

$A_{[t_1,t_2]}$ also becomes a random variable(an AND of random variables).

Claim : $H(A_{[t_1,t_2]}|G') \leq w + 2w.E[|IT(t_0,t_1,t_2)||G']$, where H is the entropy of the outputs $A_{[t_1,t_2]}$.

Proof of Claim : We had defined the entropy of a random variable as the expected length of encoding of the random variable. Also, it is known that Huffman coding is an optimal prefix-free encoding. From the construction of the Huffman code using frequencies, entropy of a random variable is the lower bound for the Huffman code over the support of the random variable. Thus, if we give a prefix-free encoding for $A_{[t_1,t_2]}$ where the number of bits required is the RHS of the claim, then we are done.

Consider the following encoding:

- First, $|IT(t_0,t_1,t_2)|$ is written down.
- The address, followed by the cell content, of each cell in the information transfer is written down.

Note that the cardinality of information transfer fixes the length of an encoding. So the encoding of two different values of $A_{[t_1,t_2]}$ will either differ in the cardinality of information transfer or will be different despite being of the same length. This ensures that the encoding is prefix-free. Also it is easy to see that the average lemgth of the encoding is the RHS of the claim.

Now we need to show that this is decodable(prefix-free implies uniquely decodable). Here, we are not bothered with the time required to calculate the outputs, we just want to make sure that the required data can be read from the data structure being maintained for the problem. Showing that the read instructions can be executed is good enough. For any algorithm, we simulate the data structure being used in the interval $[1, t_0 - 1]$. Then we skip simulation in $[t_0, t_1]$ and again simulate in $[t_1, t_2]$: Suppose the last write of a cell was at time t

- $t < t_0$: We read the data off the data structure maintained at time t_0.
- $t > t_1$: We can keep a record of what is happening in the interval $[t_1, t_2]$ and read off that record.
- $t_0 < t < t_1$ Then this cell participates in information transfer and therefore the data is present in the encoding itself.

This encoding gives us the required inequality.

3.2 Interleave

Definition : Let $(\pi(t_0), \ldots \pi(t_2))$ be the indices used in queries in the interval $[t_0, t_1]$. Suppose

L is the list of these indices after sorting. The interleave number $IL(t_0, t_1, t_2)$ is the number of transitions in L between a value $\pi(i)$, $i < t_1$, and a consecutive $\pi(j)$, $j > t_1$.

Lemma : $H(A_{[t_1,t_2]}|G') = \delta.IL(t_0, t_1, t_2)$

Proof : If two consecutive indices i and j of $L \cap [t_1, t_2]$ do not have an index from $[t_0, t_1]$ in L then the difference between the $Sum(i)$ and $Sum(j)$ is a constant and therefore the $Sum(j)$ does not add anything to the entropy function.

Otherwise, the $Sum(j)$ is a random variable depending on answers in $[t_0, t_1]$ that lie between i and j in L. Since this random variable is uniformly distributed in group G, it adds δ bits of entropy. This is done exactly at the interleaves. Hence the equality.

Using the above results we get the following corollary.

Corollary For a fixed G' : $E[|IT(t_0, t_1, t_2)||G'] \geq \frac{\delta}{2w}.(IL(t_0, t_1, t_2) - 1)$.

We can build a binary tree over the total time interval. Each node of the tree represents a time i. We recursively build a binary tree by adding a a parent to consecutive nodes that do not already have a common ancestor. We say node v corresponds to (t_0, t_1, t_2) when the right subtree rooted at v represents $[t_0, t_1]$ and the left subtree represents $[t_1, t_2]$.We define $IT(v)$ and $IL(v)$ accordingly.

Theorem 1 : For any algorithm, a fixed permutation π and query sequence , the expected running time of the algorithm, for a random sequence $(g_1, g_2, \ldots g_n)$, is at least $\frac{\delta}{2w}.\sum IL(v) - n$.

Proof : Any cell that is being read has been written to at a time before this. If the read and write does not happen at the same time, then there is a unique node in the binary tree such that this cell participates in information transfer from left-subtree to right subtree of the node. This node is in fact the lowest common ancestor of the read and write times. Thus the number of reads for the query sequence is at least $\sum IT(v)$. This clearly is a lower bound on the running time of the algorithm. Using the corollary and the lower bound on the running time of the algorithm, we get the lower bound on the expected running time of the algorithm.

If we can show that there is a permutation π for which the the total interleave is $\Omega(n \log n)$ then we are done, because in the given query sequence of length $2n$, even the amortised time per operation is $\Omega(\log n)$

3.3 Bit-Reversal Permutation

This permutation is actually a permutation on 0 to $(n-1)$. Here the permutation maps an integer i to the integer j representing the bit reversal of the binary representation of i. Notice that all numbers in the first half are mapped to even numbers and the rest to odd numbers. Now view the elements 0 to $(n-1)$ as points in time. On building the binary tree over this we see that at the root there ia perfect interleave between left subtree and right subtree. Now suppose we extend the permutation to one on $2n$ elements : $\pi' = (2\pi(1) - 1, \ldots, 2\pi(n) - 1, 2\pi(1), \ldots 2\pi(n))$. Then, looking at the binary tree built over these points in time, at the root, again the left subtree and right subtree have perfect interleave. Using this method of extension, we can get a permutation that has perfect interleave at every node of the binary tree built over the points in time. Therefore we have constructed the required permutation to show the lower bound on each operation time in Partial Sums.

4 Lower bound for Dynamic Connectivity

We want to show that there is an instance where the amortised lower bound for operations $insert()$, $delete()$ and $connected()$ is $\Omega(\log |V|)$, where V is the vertex set of the dynamic graph.

We will translate this instance to an instance of partial sums. Here, let the size of A, $n = \sqrt{V} - 1$. The group $G = S_{\sqrt{V}}$, which is the permutation group on $n+1$ elements. Our graph will be a $(n+1)x(n+1)$ grid, where at any point of time, the edges between consecutive columns represent a permutation of $n+1$.

Observation 1 : Nodes x and y in the first and k^{th} columns respectively are connected iff the partial sum $A[1].A[2] \dots A[k-1]$ takes x to y.

Observation 2 : It follows that there are at most $n+1$ disjoint paths in the graph being maintained, one starting from each node in the first column.

Here we will plug in the values of $\delta = \log |G|$ and $w = \log v$ to get the that a partial sums operation requires $\Omega(\sqrt{V}.\log V)$ cell probes for a query sequence as described earlier.

It is easy to see that $Update$ in partial sums can be implemented by $O(\sqrt{V})$ dynamic connectivity updates: updating a permutation in array $A[]$ is equivalent to deleting the existing $O(\sqrt{V})$ edges between the corresponding columns and adding $O(\sqrt{V})$ new edges. If we can argue something similar for Sum using $O(\sqrt{V})$ connectivity queries then we know from the previous lower bound that each dynamic operation also has a lower bound of $\Omega\frac{(\sqrt{V}.\log V)}{\sqrt{V}} = \Omega(\log V)$, even on amortisation. Since connectivity returns a boolean value and Sum returns a permutation with δ bits of entropy, it is not clear how to represent the latter using the former.

Instead we modify the Sum query for partial sums to $VerifySum(k, s)$. This query verifies if the partial sum upto k has value s. This can be done by checking for each node in the first column whether the output of the permutation s is the same as the output of the partial sum. Using the observations, this is same as checking for connectivity between each node in the first column and its mapping by s.

Thus, if we can give a lower bound for this modified partial sums problem we have the above lower bound for dynamic connectivity.

5 Lower bound for the modified Partial Sums problem

Here we introduce non-deterministic cell-probes. At each query many threads are created out of which exactly one accepts. While all threads are allowed to read only the accepting thread is allowed to write.

We will define the running time of a query as the number of cells read and written on by the accepting thread.

If there is a deterministic algorithm for $VerifySum$ running in time t then we have a non-deterministic algorithm for Sum, also in time t, which guesses the sum value and verifies.

We will see that for the non-deterministic algorithm for Sum too the previous lower bound holds for operation time. It follows from there that query time for $VerifySum$ has $\Omega(\log n)$ lower bound

and therefore query time for *connected* in dynamic graphs has $\Omega(\log V)$ lower bound. Then we are done with showing a lower bound for operations in the dynamic connectivity problem.

We need to bound the entropy of a nondeterministic query in terms of the information transfer.

For the relation between entropy and information transfer, we first define two things:

- $W(t_0, t_1)$ which is the number of cells written into by the accepting thread in interval $[t_0, t_1]$.
- $R(t_0, t_1)$ which is the number of cells read by the accepting thread during $[t_0, t_1]$, provided that these cells were last written into before time t_0.

So $IT(t_0, t_1, t_2) = W(t_0, t_1) \cap R(t_1, t_2)$.

Theorem 2 : $H(A_{[t_1,t_2]}|G') \leq w + 2w.E[|IT(t_0, t_1, t_2)| + |R(t_0, t_1)| + |W(t_1, t_2)||G']$.

If we look at the binary tree built of the time instances, $R + W$ at any level of the tree takes at most time T, which is the total time for the given algorithm. So the total value of $R + W$ over all nodes is $T. \log n$, since $\log n$ is the depth of the binary tree.

Then everything else follows through and we get a lower bound on the expected time of the algorithm in terms of total interleaves. Taking the bit-reversal permutation again gives a $\Omega(\log n)$ amortised time per operation.

Proof of Theorem 2: We again give the encoding argument as before. The cells read by rejecting threads are not in the information transfer by definition. So, when decoding we have to be careful while simulating rejecting threads. For example, when simulating a decoding thread, we may incorrectly think that a cell was not written during $[t_0, t_1]$. If we give the algorithm a version of the cell from before time t_0 then a rejecting thread could now turn into an accepting one, which is incorrect.

To avoid this, our encoding also keeps a dictionary record of cells in $W(t_0, t_1) \cup R(t_1, t_2) - IT(t_0, t_1, t_2)$, with an additional bit specifying whether it is from $W(t_0, t_1)$ (W) or $R(t_1, t_2)(R)$. Thus the information that a cell was written into in $[t_0, t_1]$ can be learnt from the encoding. Using Bloomier filters we can implement this in $O(|W(t_0, t_1)| + |R(t_1, t_2)|)$ bits of space.

Bloomier filter : Consider a set S from a universe U, where each element of S has r bits of associated data. We can construct a data structure occupying $O(|S|.r + \log\log|U|)$ bits of memory that answers query $Retrieve(x)$: if $x \in S$ then return the associated data; otherwise return an arbitrary value.

Construction for the Bloomier filter : We will look at the set of hash functions that map U to $[2|S|]$. If such a function h is injective on S then we can use an array with $2n$ locations of r bits each, to store the data associated with each $x \in S$ in $h(x)$. This guarantees correctness of *Retrieve*. let $|U| = m$, $|S| = n$.

There are $\binom{2n}{n}.n!.(2n)^{m-n}$ injective functions for a given S. So, if hash function h is chosen u.a.r for a fixed S, then it is injective on S with probability $\binom{2n}{n}.n!.(2n)^{m-n}/(2n)^m \geq 2^{-O(n)}$. We pick a u.a.r hash family $\mathcal{H}$ of size $2^{O(n)}. \log \binom{m}{n}$. The probability that there is no $h \in \mathcal{H}$ which is

injective on this S is $(1 - \frac{1}{2^{O(n)}})^{|\mathcal{H}|} < 1/\binom{m}{n}$. Then, taking union bound over all subsets S, there exists a family of hash functions, say $\mathcal{H}$, such that for each set S there is a hash function in the family that maps S injectively.

Time to answer the query is not of importance here. So we find out sucha hash family by brute force. The space required is the space for the table for elements in S and space for storing the hash family $\mathcal{H}$. This works out to be the space mentioned above.

Notice that $W(t_0, t_1) \cup R(t_1, t_2) - IT(t_0, t_1, t_2)$ is the universe and $W(t_0, t_1)$ is S, while there is only one bit of data associated with each element of the universe. Now the average number of bits for the total encoding is the RHS of the theorem.

It is only left to show that the outputs can be found out. As before we only need to retrieve the data queried for. Since *Update* still works the same as before we can execute it in the same way as before.Again, at time t, we analyse how to retrieve the data for the cell to be read depending on which group it belongs:

1. $W(t_1, t)$: Keep a record in the interval itself.
2. $IT(t_0, t_1, t_2)$, : This is in the encoding itself.
3. $W(t_0, t_1) - IT(t_0, t_1, t_2)$: if a query to the dictionary returns W then the answer cannot be R(property of the filter). If the answer is not R then the cell cannot be in $R(t_1, t_2)$ and hence this thread is not the accepting thread. So reject the thread immediately.
4. Anything else was written before t_0 and for that we can refer to the simulate the datastructure at that time.

Arbitrarily many reject threads are rejected in this process. The threads are either simulated correctly till they reject or they are rejected somewhere in the middle. Accepting threads are always simulated correctly.

So this gives us a correct decoding.

Hence we have given a lower bound for the partial sums problem that includes $VerifySum$ as a query, and thereby provided a lower bound for the dynamic connectivity problem.

References

[1] M. Patrascu and E. D. Demaine. *Logarithmic lower bounds in the cell-probe model.* SIAM Journal on Computing, 35(4):932-963, 2006.