CS 59000 CTT Current Topics in Theoretical CS	September 13, 2012
Lecture 8	
Lecturer: Elena Grigorescu	*Scribe: Vivek Patel*

1 Introduction

Recall that in previous lectures we've seen a couple of examples of testable graph properties. Our goal in this lecture shall be to analyze the testability of some non-graph properties. We will look at properties of lists of real numbers, namely at the property of being sorted (say, in non-decreasing order). In the end we will mention how the ideas here extend to testing monotonicity of functions defined over some discrete domain (say $f : \{0,1\}^n \to \{0,1\}$).

The aim here is to show a test which takes time $O(\log n)$ and decides whether the list is correctly sorted or it far from sorted (meaning that many of the entries of the list need to be modified in order to obtain a sorted list).

In practice one could use a quick test like this one to check if it is worth running some sorting algorithm on the data(in some applications, few outliers would not be that bad, but if there is a large fraction of outliers than one should resort). In such applications, if there are only few outliers then running a very fast test would save the effort of running a typical sorting algorithm of super-linear complexity.

Example: Say we have a list of n integers { 1,2,5,1,0,1...20 }. An efficient sorting algorithm like Quick sort would take $O(n \log n)$ time to sort the list, but if there are fewer than .1% elements that are out of order we might not care and do not need to sort. We would like to have a test which can test if the given list is sorted in time much smaller than linear.

2 Reviewing the 'Witness lemma'

We've seen the next lemma many times already (the upper bound part) and we used the lower bound in the previous lecture, but we recall it here again as we will be using it a few times today.

Lemma 1 (Witness Lemma) *1. (upper bound) If a test T is s.t.* $\Pr[T \textit{finds witness}] = p < 2/3$, *then if the test T' is obtained by running T for $s = 2/p$ times we have*

$$Pr[T' \textit{ finds witness}] > \frac{2}{3}$$

2. (lower bound) If test T'' runs T for $s < 1/(2p)$ times then the

$$Pr[T'' \textit{ catches witness}] < \frac{2}{3}$$

Proof Since $Pr[\text{T catches witness}] = p$ we have Pr[T' does not find a witness in any of its "s" steps] = $(1-p)^s$. Since $(1-p) < e^{-p}$, raising both sides to the power of s we get $(1-p)^s < e^{-p.\frac{2}{p}} = e^{-2} < \frac{1}{3}$.

To show the second part, we use $(1-p) \geq e^{-2p}$, if $0 \leq p \leq \frac{3}{4}$. As $\frac{3}{4} > \frac{2}{3}$,

$$\begin{aligned} \Pr[\text{T" does not find a witness in any of its "s" steps}] &= (1-p)^s & (1) \\ &> e^{-2p \cdot \frac{1}{2p}} = e^{-1} > \frac{1}{3}. & (2) \end{aligned}$$

■

We use the lemma to analyze the query complexity of our algorithms. The upper bound says that $2/p$ queries are enough, and the lower bound says that $1/(2p) = \Omega(1/p)$ queries are neccessary to get the correct answer w.p. $> 2/3$.

3 Some simple tests with high query complexity

Recall that we are looking for a test for sortedness. The next two tests are natural but we'll see that they require a linear number of queries.

3.1 Test 1

Consider the following test

Algorithm 1 Consecutive pair sort testing

Input: A list of n numbers, represented as $L = \langle \ell(1), \ell(2), \ldots, \ell(n) \rangle$ (where $\ell : [n] \to R$ represents labels of the indices)
Pick an index i which chosen uniformly from $[n]$
if $\ell(i) > \ell(i+1)$ then reject
else accept

Why is this a bad test? Clearly if it is given as input a sorted list the test always accepts. But what if it is given a list that is far from sorted? Does it detect a violating consecutive pair on any list with many outliers?

We'll show that there is a list that is far from sorted yet the test only detects a consecutive unsorted pair with very small probability, and so it needs to make many queries to boost this probability to 2/3.

Example: $L = \langle \frac{n}{2}, \frac{n}{2}+1, \frac{n}{2}+2 \ldots n, 1, 2, 3, ...n/2 \rangle$

Then the list is very far from being sorted: $dist(L, sortedness) = \frac{1}{2}$, as we would need to completely modify either the fist half or the second half to get to a list that is fully sorted.

Analyzing the test above, we get $\Pr[\text{test rejects}] = \frac{1}{n-1}$. Hence, by the witness lemma above, the number of queries required in order to catch a violating witness with probability $\frac{2}{3}$ is $(n-1)/2 = \Omega(n)$ which is too much.

3.2 Test2

Algorithm 2 Random index sort testing

Input: A list of n numbers, represented as $L = \langle \ell(1), \ell(2), \ldots, \ell(n) \rangle$
Pick indices $i < j$ which are chosen uniformly from $[n]$.
if $\ell(i) > \ell(j)$ then reject
else accept

As before, we'll find a string that is far from sorted yet the test is unable to find a violating pair except with some small probability.

Example: A list that interleaves two sorted lists: $L = \langle \mathbf{1}, 0, \mathbf{2}, 1, \mathbf{3}, 2, \mathbf{4}, 3, ... \rangle$.

Note that $dist(L, sortedness) = \frac{1}{2}$ since among each consecutive pair of numbers $(2i+1, 2i+2)$ one of the labels need to be changed to get a non-decreasing sequence.

We have

$$\Pr[\text{ test rejects }] = \frac{\#\text{ pairs } i < j \text{ with } \ell(i) > \ell(j)}{\#\text{ pairs } i, j} \quad (3)$$

$$= \frac{\frac{n}{2}}{\binom{n}{2}} = \frac{1}{n-1}. \quad (4)$$

(where we used that index $2i+1$ is out of order only with index $2i+2$.) Hence again we need $\Omega(n)$ queries.

4 A good test

In this section we'll show a test that works with only $O(\log n)$ queries.

Theorem 2 *There exists a test that given ϵ, can test if a list of size n is sorted by making only $O(\frac{\log n}{\epsilon})$ queries.*

The Idea: View the n indices of the list as vertices of a directed line L_n, with edges directed from vertex i to $i+1$, for $i \in [n-1]$. We'll add 'shortcut' edges between some pairs (i, j) (from i to j, if $i < j$) which will decrease the distance between vertices. After the addition of the set of shortcuts we get a new graph H with the following properties:

1. The number of edges in H is $n \log n$
2. For any indices $i < j$ we have a directed path of length at most 2 between them in H.

Such a graph is called a 2-*transitive closure (TC) spanner.*

4.1 How to build H

The graph H we get from the previous stage has n nodes. Pick the mid- point $v_{mid} = \frac{n}{2}$ and mark it on the graph . For every vertex to the left of the mid point add a directed edge connecting that vertex to the mid point. For every vertex to the right of the midpoint add a

directed edge from the mid point to it. Hence the distance between any vertex $i < v_{mid}$ and $j > v_{mid}$ is at most 2. We then do this recursively on the segments to the left and to the right of the mid point. Therefore, $\forall i, j$ there exists a level of recursion when i and j are separated by a midpoint at that level. Then both i, j are connected to the midpoint at that level and so they are within distance 2 from each other. This concludes that H is a TC-spanner. Finally, counting the number of edges of the graph H, note that Level 1 of recursion we add at most n edges. Similarly, at each other level we add at most n more edges. Since there are $\log n$ levels, we added at most $n \log n$ edges to obtain H.

4.2 The test

Algorithm 3 Test 3

Input: A list of n numbers, represented as $L = \langle \ell(1), \ell(2), \ldots, \ell(n) \rangle$
Pick an edge (i, j) (where $i < j$) uniformly from H
if $\ell(i) > \ell(j)$ reject
else accept

We'll next analyze this test.

Suppose list is ϵ-far from sorted. Call an edge *bad* if $i < j$, but $\ell(i) > \ell(j)$. Then Pr[test 3 finds a bad edge] $= \frac{\#\text{ bad edges}}{|H|}$.

In what follows we'll count the number of bad edges is H.

Lemma 3 *The number of bad edges in H is $\geq \frac{\epsilon}{2} n$.*

Notice that Lemma 3 immediately implies Theorem 2, since $|H| = n \log n$ and so it is enough to run the basic test $O(\log n/\epsilon)$ times to find a bad edge w.p. 2/3.

Proof [of Lemma3] Call a *bad vertex* one that is adjacent to a bad edge. Otherwise we call it a *good vertex.*

We make use of 2 more claims to prove the lemma.

Claim 4 *The good vertices are already sorted.*

Claim 5 *If set $S \in [n]$ is sorted then it is enough to change $n - |S|$ labels in order to get a sorted List.*

We can now complete the proof of the lemma using the claims. Recall that at least ϵn labels need to be changed to get a sorted list. By claim 4 all good vertices are ordered, and by Claim 5 it is enough to change the labels of the bad vertices to get a sorted list. This implies that the number of bad vertices must be $\geq \epsilon n$ Since a bad edge is adjacent to at most 2 bad vertices there must be at least $\frac{\epsilon}{2} n$ bad edges.

It remains to prove the claims.

Proof [of Claim4] We need to show that if $i < j$ then $\ell(i) < \ell(j)$. Since H is a 2- TC Spanner i, j are at a distance of at most 2 in H. If the distance is 1, that is (i, j) is an edge, since both i, j are good then we must have $\ell(i) < \ell(j)$. If the distance between i j is 2, then we must have the path as $(i, k)(k, j)$ in H. Again it must be that both the edges are good, so we have $\ell(i) \leq \ell(k)$ and $\ell(k) \leq \ell(j)$ hence we have $\ell(i) < \ell(j)$. ■

Proof [of Claim 5] Notice that we can relabel the bad vertices to take the value of the closest good vertex and obtain an ordered list. ■

■

5 Testing monotonicity of functions

The above test can be viewed as a test for the monotonicity of a function defined over the domain L_n and range R. One can generalize this to other discrete domains. For example, a well studied property is monotonicity of functions defined over the hypercube $\{0,1\}^n$, that is, given a function $f : \{0,1\}^n \to \{0,1\}$ is it monotone or far from any monotone function? Here the partial order on $\{0,1\}^n$ is given by set inclusion.