6.896 Sublinear Time Algorithms March 8, 2007

Lecture 10

Lecturer: Ronitt Rubinfeld *Scribe: Khanh Do Ba*

1 Recap

1.1 Algorithm and notation

LZ77(w)

$t \leftarrow 1$
repeat
    find longest substring $w_t ... w_{t+\ell-1}$ s.t. $\exists$ index $p < t$ with $w_p ... w_{p+\ell-1} = w_t ... w_{t+\ell-1}$
    if none **then**
        next symbol $= w_t$
        $t \leftarrow t + 1$
    else
        next symbol $= (p, \ell)$
        $t \leftarrow t + \ell$
until $t > n \ \ (= |w|)$

Recall the following notation.

$n_\ell(w)$ = # compressed segments of length ℓ in w, not counting alphabet symbols and last compressed segment

$C_{LZ}(w)$ = # symbols in compressed string (# pairs + # alphabet symbols)

$d_\ell(w)$ = # distinct substrings of length ℓ

1.2 Main theorem

Recall the main theorem, which forms the basis of our LZ-Approx algorithm.

Theorem 1 *For any integer $\ell_0 > 1$, let*

$$m = m(\ell_0) = \max_{\ell \in [\ell_0]} \frac{d_\ell(w)}{\ell}.$$

Then

$$m \le C_{LZ}(w) \le 4\left(m \log \ell_0 + \frac{n}{\ell_0}\right).$$

Last lecture, we already proved the first inequality. We also claimed that

(*) For every $\ell \in \{1, \ldots, \ell_0/2\}$ (wlog, assume ℓ_0 is even),

$$\sum_{k=1}^{\ell} n_k \le 2(m+1) \sum_{k=1}^{\ell} \frac{1}{k}.$$

(**) For every $\ell \ge 1$,

$$\sum_{k=\ell+1}^{n} n_k \le \frac{n}{\ell+1}.$$

and proved (**), as well as that (*) and (**) together imply the second inequality. Today, it remains to prove (*).

2 Proof of (*) and Theorem 1

Proof We reexpress the summation as

$$\sum_{k=1}^{\ell} kn_k = A + B,$$

where $A = \#$ locations within $\{\ell, \ldots, n-\ell\}$ that lie in compressed substrings of length $\leq \ell$, and $B = \#$ locations within $[n] \setminus \{\ell, \ldots, n-\ell\}$ that lie in compressed substrings of length $\leq \ell$. Clearly, $B \leq 2\ell$. Let $C = \#$ distinct length-2ℓ substrings, so that $C \leq 2m\ell$ by the definition of m. It remains to show that $A \leq C$ by giving a one-to-one mapping $f : A \to C$.

Define f as follows. For each location $j \in A \subset \{\ell, \ldots, n-\ell\}$, let $f(j)$ be the substring of w of length 2ℓ ending at location $j + \ell$, i.e., $w_{j-\ell+1}..w_{j+\ell}$. Clearly f is well-defined, so we are left to show that it is one-to-one.

Now, let us call a substring $w_t..w_{t+k-1}$ "new" if there exists no $p < t$ such that $w_p..w_{p+k-1} = w_t..w_{t+k-1}$. It follows that if $w_t..w_{t+k-1}$ is a compressed substring of length k, then since LZ always adds the longest previously seen substring, adding any suffix would make it new. Moreover, as long as we add some suffix, we can add as much of a prefix as we like and the substring would still be new. In other words, a substring which entirely contains a compressed substring, plus at least one location after it, must be new. In particular, $f(j)$ is new for any $j \in A$, since the compressed segment containing j (which is of length $\leq \ell$) can start no further left than $w_{j-\ell+1}$ and end no further right than $w_{j+\ell-1}$, hence is (right-hand-side) strictly contained in $f(j) = w_{j-\ell+1}..w_{j+\ell}$. This completes the proof of the theorem. ■

3 Models of Computation for Sublinear Algorithms

Previously, we studied inputs that were black-box probability distributions, out of which we were only allowed to take independent samples. In that case we wanted algorithms sublinear in the domain size. Now, our model allows random access to the input. That is, we are given a large input string and wish to only read a sublinear amount of it. For example, this was the case in our LZ-Approx algorithm, where we read random *segments* of the input, instead of single, independent locations.

Question: must it always be randomized and/or approximate?

3.1 Deterministic?

Consider the problem of computing the diameter of a set of points.

Given:	$m \times m$ distance matrix D, symmetric and satisfying triangle inequality
Output:	diameter $d = \max_{ij} D_{ij}$

We wish to compute d in time sublinear in the input size $n = m^2$. Consider the following deterministic algorithm.

1 Pick x arbitrarily
2 Pick y to maximize D_{xy}
3 Output D_{xy}

Claim 1 $d/2 \leq D_{xy} \leq d$.

Proof Let i, j be such that $D_{ij} = d$. Then by triangle inequality,

$$\begin{aligned} d = D_{ij} &\leq D_{ix} + D_{xj} \\ &\leq D_{xy} + D_{xy} \\ &\leq 2D_{xy}, \end{aligned}$$

so $D_{xy} \geq d/2$. The other inequality is obvious. ■

This is provided as an example of a nontrivial sublinear *deterministic* algorithm, but in general, we will almost always deal with randomized approximation algorithms.

3.2 Property testing

The general idea in property testing is as we have seen: if the data has the property, we pass; if the data is "far" from having the property, we fail; if it is in between, we make no guarantee. The important variable here is the definition of "far."

3.3 A property-tester for connectivity

Given a graph $G = (V, E)$, where $|V| = n$, $|E| = m$, multiple edges are allowed, self-loops are not, degree is bounded by d, and G is given in adjacency list format, we must tell whether G is connected. More precisely, if G is connected, Pr[output pass] $\geq 2/3$, and if G is ε-far from connected, Pr[output fail] $\geq 2/3$.

We attempt one definition of being ε-far from connected.

Definition 1 *G is ε-far from connected if we must add at least εdn edges to make it connected.*

A potential problem is adding these edges can drastically change the degree bound d. Alternately, let us define

Definition 2 $P_{n,d} = \{G' : G'$ *is a connected n-vertex graph with max degree* $\leq d\}$.

Definition 3 *G is ε-far from $P_{n,d}$ if we must add/delete at least εdn edges to make it a member of $P_{n,d}$.*

For this algorithm, we will use Definition 1. The key idea is as follows. If G is far from connected, then it has a lot of connected components, which means it has a lot of small connected components, which means a random vertex is reasonably likely to land in a small connected component.

Conn-Tester(G)
1 Choose $m = O\left(\frac{1}{\varepsilon d}\right)$ vertices uniformly at random
2 **for** each vertex s **do**
3   Run BFS until (a) $> \frac{2}{\varepsilon d}$ new nodes explored or (b) s is found to be in a component of size $\leq \frac{2}{\varepsilon d}$
4 **if** (b) ever happens **then** FAIL **else** PASS

The number of queries is

$$O\left(\frac{1}{\varepsilon d} \cdot \frac{2}{\varepsilon d} \cdot d\right) = O\left(\frac{1}{\varepsilon^2 d}\right),$$

where $\frac{1}{\varepsilon d}$ is the number of starting vertices, $\frac{2}{\varepsilon d}$ is the number of BFS steps, and d is the cost of a BFS step.

3.4 Correctness of algorithm

Theorem 2 *The algorithm is a property tester for connectivity. More precisely,*

(1) If G is connected then Conn-Tester(G) passes.

(2) If G is ε-far from connected, then $\Pr[$*Conn-Tester(G) fails*$] \geq 2/3$.

Proof (1) is obvious. For (2), observe that if G is ε-far from connected, then it has $> \varepsilon dn$ connected components. A direct consequence of this is that G must have $\geq \frac{\varepsilon dn}{2}$ components of size $\leq \frac{2}{\varepsilon d}$. In particular, each of these components has at least one vertex, so the number of vertices in small ($\leq \frac{2}{\varepsilon d}$) components is at least $\frac{\varepsilon dn}{2}$. It follows that

$$\Pr[\text{random } s \in V \text{ lies in component of size} \leq \tfrac{2}{\varepsilon d}] \geq \frac{\varepsilon dn/2}{n} = \frac{\varepsilon d}{2}.$$

Thus, using the standard approximation $(1 - 1/x)^x \approx 1/e$,

$$\Pr[\text{no } s \text{ lies in small component}] \leq \left(1 - \frac{\varepsilon d}{2}\right)^{c/\varepsilon d} \approx e^{-c/2} \leq 1/3$$

for large enough c. ■