CSC2420 - Algorithm Design, Analysis and Theory **Fall 2022**

Lecture 10: Sublinear Algorithms

Lecturer: Allan Borodin *Scribe: Kevin Gao*

We consider two contexts in which randomization is usually necessary. In particular, sublinear time algorithms and the streaming algorithms. An algorithm is sublinear time if its running time is $o(n)$ for input of length n. The algorithm can give an answer without reading the entire input.

To achieve sublinearity, we almost always have to use randomness to sample parts of the inputs. There will usually be a tradeoff between the accuracy of the solution and the time bound.

10.1 Sublinar without Randomness

Suppose we are given a finite metric space M where the input is given as n^2 distance values $d(x_i, x_j)$. The problem is to compute the diameter D of the metric space, i.e. the maximum distance between any two points.

For the max diameter problem, there is a simple $O(n)$ algorithm algorithm. Choose an arbitrary point $x \in M$ and compute $D = \max_j d(x, x_j)$. By the triangle inequality, D is a 2-approximation of the diameter.

10.2 Searching in Sorted Linked List

Suppose we have an array $A[i]$ for $1 \leq i \leq n$ where each $A[i]$ is a pair (x_i, p_i) with $x_1 = \min\{x_i\}$ and p_i being a pointer to the next smallest value in the linked list. That is, $x_{p_i} = \min\{x_j \mid x_j > x_i\}$. For simplicity we are assuming all x_j are distinct. We would like to determine if a given value x occurs in the linked list and if so, output the index j such that $x = x_j$.

We cannot use binary search because although we have pointers to the next smallest value, the array itself is not sorted. We present a $\sqrt{n}$ time algorithm for searching in this type of anchored sorted linked list.

```
1  $R = \{j_i\}$ be $\sqrt{n}$ randomly chosen indices plus the index 1
2  access $A[j_i]$ to determine $k$ such that $x_k$ is the largest of the accessed array elements
   less than or equal to $x$
3  search forward $2\sqrt{n}$ steps in the linked list to find $x$
```

Theorem 10.1 (Chazelle, Liu, Magen)**.** *This is a one-sided error algorithm. The algorithm will fail to return j such that $x = A[j]$ with probability at most* $1/2$.

10.3 Estimating Average Degree in Graph

Given a graph $G = (V, E)$ with $|V| = n$, we want to estimate the average degree d of the vertices. We want to construct an algorithm that approximates the average degree within a factor less than $(2 + \epsilon)$ with probability at least 3/4 in time $O\left(\frac{\sqrt{n}}{\text{poly}(\epsilon)}\right)$. We assume that the graph is stored in a way such that the degree d_i of any vertex v_i can be accessed in one step.

The algorithm below is due to Feige.

```
1  sample $8/\epsilon$ random subsets $S_i \subseteq V$ each of size $\sqrt{n}/\epsilon^3$
2  compute the average degree $a_i$ of nodes in each $S_i$
3  return $\min\{a_i\}$
```

It turns out this algorithm can also be adapted to estimate the average of n numbers according to the following fundamental result in graph theory.

Theorem 10.2 (Erdos-Gallai)**.** *The sequence $d_1 \geq d_2 \geq \ldots \geq d_n \geq 1$ is a graph degree sequence if and only if $\sum_{i=1}^n d_i$ is even and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n d_i$.*

For the analysis, we will consider a slightly weaker result. The exact theorem as presented by Feige is as follows.

Theorem 10.3. *For any d_0 (minimum degree in the graph) and for $\rho = 2+\epsilon$, the Feige algorithm computes an estimation within a factor of ρ with high probbility and uses $O\left(\frac{1}{\epsilon}\sqrt{n/d_0}\right)$ degree queries.*

Feige's algorithm samples subsets to estimate the average degree. We might have over and under-estimates. But the following lemmas guarantee that with high probability these estimates will not be too bad.

Lemma 10.4.

$$\Pr[a_i < 1/2(1-\epsilon)d_{avg}] \leq \frac{\epsilon}{64}$$

Proof: We use the Chernoff bound. Recall that the Chernoff bound bounds the probability that a sum of independent random variables deviates from its mean. More formally, let $Z_1, \ldots, Z_s$ be a sequence of independent r.vs with $Z_j \in [0,1]$ and let $\mu = \mathbb{E}[\sum_j Z_j]$. Then,

$$\Pr\left[\sum_j Z_j \leq (1-\epsilon)\mu\right] \leq e^{-\epsilon^2 \frac{\mu}{2}}.$$

Let H be the $\sqrt{\epsilon' n}$ vertices of the highest degree. Here ϵ' will be chosen to satisfy $(1-\epsilon')(1/2-\epsilon') \leq (1/2-\epsilon)$. Assume that the random selection of nodes in the algorithm was restricted to just $L = V \setminus H$. By removing the high degree vertices from the random sampling, the probability of $d_{avg} \leq 1/2(1-\epsilon)d$ decreases. We claim that

$$\sum_{i \in L} d_j \geq \left(\frac{1}{2} - \epsilon'\right) \sum_{i \in V} d_i - \epsilon' n.$$

Thus, the average degree in L is at least $\frac{1}{2}(d_{avg} - \epsilon')$. So it remains to find a lower bound on the average degree of vertices in L. We use the following observations:

- d_H = minimum degree of any vertex in H
- X_j = degree of a vertex $v_j \in S_i$ which implies $X_j \in [1, d_H]$
- $Z_j = X_j/d_H$

Using the Chernoff bound and the bound for average degree of vertices in L, we have

$$\Pr[a_i < (1/2)(1-\epsilon)d_{avg}] \leq e^{-\frac{\epsilon^2 s \mathbb{E}[X_j]}{4d_H}}$$

If $s = |S_i|$ is sufficiently large (i.e. $s \geq \epsilon^2 \frac{d_H}{\mathbb{E}[X_j]}$), we are done. Otherwise, we consider the cases when $d_H < |H|$ and $d_H \geq |H|$ separately. ■

Lemma 10.5.

$$\Pr[a_i > (1+\epsilon)d_{avg}] \leq 1 - \frac{\epsilon}{2}$$

Proof: Consider any S_i. Letting X_j be the degree of vertex v_j. We have

$$\mathbb{E}[X_j] = d_{avg}$$

and

$$\mathbb{E}[a_i] = \mathbb{E}\left[\frac{1}{|S_i|}\left(\sum_{j\in S_i} X_j\right)\right] = \frac{1}{|S_j|}\sum_{j\in S_i}\mathbb{E}[X_j] = d_{avg}$$

The lemma then follows from Markov's inequality. ■

Proof of the theorem:

Proof: The probability bound in Lemma 10.5 is amplified to any constant by repeated trials. For Lemma 10.4, we fall outside the desired bound if any of the repeated trials gives a very small estimate of the average degree but by the union bound, this is no worse than the sum of the probabilities for each trial. That is

$$\Pr[ALG < 1/2(1-\epsilon)d_{avg}] \leq \sum_{i=1}^{\epsilon/8}\left(\frac{\epsilon}{64}\right) = \frac{1}{8}$$

■

10.4 Input-Query Model

Sublinear algorithms almost always sample the input in some way. The nature of these queries will influence what kind of results can be obtained.

Feige shows that in the **input-query model** where the algorithms only makes "degree queries" (i.e. "what is the degree of v?"), any algorithm that achieves a $(2-\epsilon)$ approximation for any $\epsilon > 0$ requires time $\Omega(n)$. This is a lower bound on the time complexity for the average degree problem discussed earlier.

10.5 Property Testing

The most prevalent and useful aspect of sublinear time algorithms is for the concept of **property testing**. The idea is: Given an object G (e.g. function, graph, etc.), test whether or not G has some property P (e.g. linearity, bipartite, etc.).

The tester determines with sufficiently high probability if G has the property or is ϵ-far from having that property. The tester can choose either way if G does not have the property or is ϵ-close to having the property.

10.5.1 Tester for Linearity of a Function

Let $f : \mathbb{Z}_n \to \mathbb{Z}_n$. f is linear if $\forall x, y,\ f(x+y) = f(x) + f(y)$. This is also a test for group homomorphism.

f is said to be ϵ-**close** to linear if its values can be changed in at most a fraction ϵ of the function domain arguments (i.e. changing the values of at most ϵn elements of $\mathbb{Z}_n$) so as to make it a linear function. Otherwise, f is said to be ϵ-far from linear.

```
1 repeat $4/\epsilon$ times
2     choose $x, y \in \mathbb{Z}_n$ at random
3     if $f(x) + f(y) \neq f(x+y)$
4         output that $f$ is not linear
5 if all $4/\epsilon$ trials succeed, then output that $f$ is linear
```

10.5.2 Tester for Monotonicity

Given a list $A[i] = x_i$ for $i = 1, \ldots, n$ of distinct elements, determine if A is a **monotone list**. That is, if for all $i < j$, $A[i] < A[j]$ or is ϵ-far from being monotone in the sense that more than ϵn list values need to be changed in order for A to be monotone.

```
1 $I$ = randomly choose $2/\epsilon$ indices $i$
2 for $i \in I$
3 binary search $A$ for $x_i$
4 if binary search did not report $i$ as the position of $x_i$
5     report that list is not monotone
6 report that the list is monotone
```

The time complexity is $O(\log \frac{n}{\epsilon})$. Also, if A is indeed monotone, the algorithm clearly reports that A is monotone.

If A is ϵ-far from being monotone, then the probability that a random binary search will succeed is at most $(1-\epsilon)$ and hence the probability of the algorithm failing to detect non-monotonicity is at most $(1-\epsilon)^{2/\epsilon} \leq \frac{1}{e^2}$.

10.5.3 Graph Property Testing

Let $G = (V, E)$ with $n = |V|$ and $m = |E|$.

Dense model: Graphs represented by adjacency matrix. Say that graph is ϵ-far from having a property P if more than ϵn^2 matrix entries have to be changed so that graph has property P.

Sparse model / bounded degree model: Graphs represented by vertex adjacency lists. Graph is ϵ-far from property P is at least ϵm edges have to be changed.

10.5.4 Bipartite Testing in Bounded Degree Model

```
**repeat** $O(1/\epsilon)$
    randomly select a vertex $s \in V$
    **if** HAS-ODD-CYCLE($s$)
        **reject**
**accept**
```

10.6 Sublinear Space

We let USTCON (resp. STCON) denote the problem of deciding if there is a path from some specified node s to some specified target node t in an undirected (resp. directed) graph G.

Aleliunas et al (1979) showed that USTCON is in RSPACE($\log n$) using random walk, and after a sequence of partial results about USTCON. is in DSPACE($\log n$). It remains open if:

- STCON is in RSPACE($\log n$) or even DSPACE($\log n$), or equivalently, is NSPACE($\log n$) = DSPACE($\log n$)?
- STCON $\in$ RSPACE(S) or even DSPACE(S) for any $S = o(\log^2 n)$?
- RSPACE(S) = DSPACE(S)?

10.7 Streaming Model

In the **data stream model**, the input is a sequence A of input items $a_1, \dots, a_n$ which is assumed to be too large to store in memory. Let $a_i \in [1, D]$.

We usually assume that n is not known and one can think of this model as a type of online or dynamic algorithm. The space available $S(n, D)$ is some sublinear function. The input items stream by and one can only store information is space S.

10.7.1 Missing Item Problem

The missing item problem has a deterministic streaming algorithms. However, it will turn out that almost all of the other results in this area are for randomized algorithms.

Suppose we are given a stream $A = a_1, \dots, a_{m-1}$ and we are promised that the stream A is a permutation of $\{1, \dots, m\} - \{x\}$ for some integer x in $[1, n]$. The goal is to compute the missing x.

10.8 Majority Element and Misra-Gries Algorithm

For the **frequency moments problem**, let $A = a_1, \dots, a_m$ be a data stream with $a_i \in [n] = \{1, \dots, n\}$. Let m_i be the number of occurrences of the value i in the stream A. For $k \geq 0$, the kth frequency moment is

$$F_k = \sum_{i \in [n]} (m_i)^k$$

For the k-**heavy hitter problem**, we find the elements appearing more than m/k times in stream A. One relatively easy (but still very interesting) result is the **Misra-Gries algorithm** for computing k heavy hitters. As a special case, we have the majority problem (i.e. the k-hitter problem for $k = 2$)

Maintain a candidate for the majority element and a counter for that candidate. When the counter is empty, the next element in the stream becomes the candidate. Every time the next element in the stream is the candidate, increase the counter by 1. If the next element is not the candidate, decrease the counter by 1.

This algorithm can be visualized using a FIFO stack, although it does not need to be implemented using the stack.

```
MAJORITY($\sigma = (i_1, i_2, \ldots, i_m)$)
1  $element = i_1$
2  $count = 1$
3  for $i = 2$ to $m$
4      if $i_1$
```

Claim: If there is a majority element, then it has to be the current candidate.

The algorithm then runs a second pass over the elements to check if the candidate occurs more than $n/2$ times. The algorithm uses $O(\log n + \log D)$ and the time is $\log n$ per input element.

It can be shown that no one-pass streaming algorithm can return a truly majority element. So the second pass is necessary to verify a candidate.

The majority algorithm can be generalized to solve the k-heavy hitter problem for any small k. For k heavy hitter, we instead maintain $k - 1$ counters. In terms of space, we need $k - 1$ counters (as opposed to one counter for majority element) as well as the name of the candidate that each counter is keeping track of (say using $O(\log k)$ space).

Consider the following extension to the ϵ-approximation ϕ-hitters problem where the algorithm is required to return a set S of elements such that

- Every element in S appears at least ϕn times
- S does not contain any elements less than $(\phi - \epsilon)n$ times

The Misra-Gries algorithm can be extended to solve this problem by setting $k = 1/\epsilon$ and then outputing all elements a that occur at least $(\phi - \epsilon)n$ times.