0368.416701 Sublinear Time Algorithms October 27, 2014

Lecture 1

Lecturer: Ronitt Rubinfeld *Scribe: Michal Kleinbort and Kiril Solovey*

1 Introduction

In many fields processing really big data, that is, huge data sets, is a basic requirement. The potentially accessible data is so big that one may not even access all of it, as it may take too much time. Additionally, it may even happen that once the data is accessed it changes. Therefore, the development of algorithms to handle such data, whose running time is sublinear in the input size, became a rapidly emerging topic in computer science.

Let us consider the following toy example: Suppose that we have a graph, whose nodes represents persons and there is an edge between two persons (nodes) who know each other. We say that the graph is *connected* if there is a path between every pair of nodes. The *distance* between two people is defined to be the shortest path between the two nodes, that is, the minimum number of edges to reach one from the other. The *diameter* is the maximum distance between any pair.

According to the "small world property", the diameter of the world population is 6. How could we experimentally verify that the property holds? The data is enormous, there are unknown groups of people on earth, and the set of people keeps changing since people are born and die all the time.

A linear time algorithm is too expensive, when the data is immense. Thus, we will not be able to answer questions that require a perfect knowledge regarding the properties of each an every member of the group. Instead, we should compromise and aim for a sublinear time algorithm that is not allowed to view all the data or even most of it, but in return is only required to return an approximate answer. We will ask different questions such as:

- Is there a large group of individuals whose diameter is at most 6?
- Is the average pairwise distances of a graph roughly 6?
- Approximately how many individuals on earth are left-handed?

It should be noted that most sublinear time algorithms are randomized and non-deterministic.

2 Sublinear time models

When discussing sublinear time algorithms, we assume two possible models for the input:

1. **Random access queries.** The data does not change and we have random access (constant time access) to each data entry.
2. **Samples of a distribution.** We can get a sample of the same distribution in one step (constant time). We will often assume in this case that the taken samples are independent. Alternatively, we can get a random word of the input in one step (constant time).

The first model can be viewed as taking random samples of a distribution.

3 Examples

3.1 Approximating the diameter of a point set

The following is a classical example of a deterministic sublinear-time approximation algorithm. It returns a 2-approximation.

Given m points, described by a distance matrix D, such that:

- D_{ij} is the distance from point i to point j
- D satisfies the triangle inequality, that is, $\forall i, j, k$, $D_{ij} + D_{jk} \geq D_{ik}$
- D satisfies symmetry, that is, $\forall i, j$, $D_{ij} = D_{ji}$

The size of the input is $n = m^2$.

Output: Let d be the diameter of the point set, i.e., the largest entry in D. The algorithm returns $D_{kl} \geq d/2$.

Algorithm

- Pick an arbitrary point k
- Pick l that maximizes D_{kl}
- Output D_{kl}

Time complexity: $O(m) = O(\sqrt{n})$

Correctness: Let i, j be two points for which $D_{ij} = d$ (recall that d is the diameter). The algorithm output D_{kl}.

Claim 1 $D_{kl} \geq D_{ij}/2$.

Proof By the triangle inequality it follows that

$$D_{ij} \leq D_{ik} + D_{kj},$$

and by symmetry ($D_{ki} = D_{ik}$) we know that

$$D_{ij} \leq D_{ik} + D_{kj} = D_{ki} + D_{kj}.$$

Since the algorithm picked the l such that $\forall i \in \{1, \ldots, m\}$, $D_{kl} \geq D_{ki}$ we conclude that

$$D_{ij} \leq D_{ik} + D_{kj} = D_{ki} + D_{kj} \leq 2D_{kl}.$$

■

3.2 Property testing

The following type of approximation algorithms is good for decision problems, where we output *Yes* for inputs that have a certain property and output *No* for those that do not have it. We would like to *quickly* answer a weaker question (a "gap problem"):

- If the input has the property - output *Yes*.
- If the input is far from having the property - output *No*.
- For inputs that are close to having the property but do not have it answer either *Yes* ot *No*.

It is important to understand the way the input is represented (for instance, if the input is a graph then is it given as an adjacency matrix or as an adjacency list?) and also to define the notion of distance (close/far to having the property).

3.2.1 A simple property tester: sortedness of a sequence

Suppose that we are given a list $y_1 y_2 \dots y_n$. We already know that it would not be able to tell whether it is sorted, as it would require a linear time. Instead, we wish to quickly decide whether the list is "close" to sorted:

Definition 2 *A list of size n is ϵ-close to sorted if at most ϵn of its entries can be deleted to make it sorted.*

We would like to be able to answer whether a given list is ϵ-close to sorted in time $O(\frac{1}{\epsilon}\log n)$, where we assume that the query time is $O(1)$. We mention that a more efficient algorithm for this problem does not exist. Notice that as we are dealing with randomized algorithms, we cannot expect that it will *always* return the correct answer. Instead, we require that a correct answer will be returned with high probability. It suffices to guarantee a probability of $3/4$, since it can be always amplified, using the Chernoff bound that will be described below, to a higher value.

Before describing the actual algorithm we first make a couple of failing attempts at solving the problem. Suppose that we pick a random i and check whether $y_i \leq y_{i+1}$. Restricting ourselves only to this routine will be very limiting as it is a challenge to find the exact "breakpoint". For instance, we might have the following sequence:

$$1, 2, \dots, \frac{n}{4}, 1, 2, \dots, \frac{n}{4}, 1, 2, \dots, \frac{n}{4}, 1, 2, \dots, \frac{n}{4}.$$

Another option would be for a query to pick random $i < j$ and check $y_i \leq y_j$. However, this method in itself can be rather limited as well, since we might have many short subsequences in which numbers are strictly decreasing. For instance, the following sequence is composed of $\frac{n}{4}$ subsequences of 4 decreasing elements:

$$4, 3, 2, 1, 8, 7, 6, 5, 12, 11, 10, 9, \dots, 4k, 4k-1, 4k-2, 4k-3.$$

One would need to be able to sample i, j inside such subsequences in order to detect discrepancies. In this case we will need to make as many as $\sqrt{n}$ queries, due to the *birthday paradox*.

Before moving on to the actual solution we make the assumption that the list is distinct ($x_i \neq x_j$). Such an assumption can always be made since every sequence $x_1, x_2, \dots, x_n$ can be treated as $(x_1, 1), (x_2, 2), \dots, (x_n, n)$. This transformation allows us to break ties without changing the order.

Algorithm. Perform the following test $O(1/\epsilon)$ times:

- Pick a random i
- Look at value of y_i
- Perform binary search for y_i in the list
- If the search does not end up in location i return "fail"

Obviously the running time is $O(\epsilon^{-1}\log n)$, but why does it work? First, note that if the list is sorted we will always return "success".

Correctness. We define index i to be *good* if binary search for y_i is successful. Our algorithm can be restated as picking $O(\epsilon^{-1})$ and returning "success" if they are all good. The main observation is that good elements form increasing sequences. Formally, suppose that $i < j$ are good. Denote by k the least common ancestor of i, j in the binary search. As the search for i went left, and the search for j went right we may conclude that $y_i < y_k < y_j$. Thus, if a list is ϵ-close to sorted then at least $(1-\epsilon)n$ elements are good. Suppose that at most $(1-\epsilon)n$ elements are good. The probability to only pick good i's in c/ϵ choices is bounded from above by

$$(1-\epsilon)^{c/\epsilon} \leq (\epsilon^{-1})^{c} << \frac{1}{4},$$

where we used the fact that $(1-\epsilon)^{\epsilon^{-1}} \sim \epsilon^{-1}$.

3.2.2 Another property tester: finding duplicates in a set of words

Given n words $x_1, x_2, \ldots, x_n$

Output Are all x_i distinct?

Complexity Requires $\Omega(n)$ queries, depending on the model of computation (for instance, assuming a comparison-based algorithm, $\Omega(n \log n)$ queries are required).

A weaker question Distinguish two cases:

- all words are distinct
- the number of distinct elements is smaller than $(1-\epsilon)n$.
- if neither case holds, output arbitrarily.

Proposed algorithm We know nothing about the order of the words, therefore one can show that the best we can do is take *several* independent random samples of the input. If we find a duplicate we return *No*, otherwise we return *Yes*.

Correctness For inputs where all words are distinct, the algorithm will always output *Yes*.
Suppose that the number of distinct elements is less than $(1-\epsilon)n$. How many samples are needed to detect a duplicate?
For example: for input $1, 1, 2, 2, \ldots, n/2, n/2$ in random order after $O(\sqrt(n))$ samples we will be able to detect duplicates (the birthday paradox).
Remark In order to analyze this number one as to define whether the sampling is with or without replacements. Additionally, the way the duplicates are distributed may have an impact on the value we are looking for (Detecting duplicates is easier when the input includes a single word that has many duplicates, as opposed to input with many repeating words).

Algorithm (Will be discussed in the next lecture)

4 Probabilistic toolkit

We review three fundamental inequalities that will come in handy throughout the course.

4.1 Markov's inequality

Let x be a positive random variable and denote by $E[x]$ its expectancy. Then the probability of exceeding the expectation by more than a factor of c is at most $1/c$. Formally,

$$\Pr[x > cE[x]] < \frac{1}{c}.$$

4.2 Chebyshev's inequality

Denote by B some upper bound on the standard deviation of x. The the probability of deviating from expectation by more than cB is at most $1/c^2$, that is

$$\Pr[|x - E[x]| > cB] < \frac{1}{c^2}.$$

4.3 Chernoff-Hoeffding bounds

We wish to bound the probability that the sum of independent variables deviates from its expectation. Let $S := \sum_{i=1}^{n} x_i$, where x_i's are independent 0/1 variables. Then

$$\Pr[S > (1+\delta)E[S]] < e^{-\delta^2 E[S]/3},$$
$$\Pr[S < (1-\delta)E[S]] < e^{-\delta^2 E[S]/2}.$$