Bocconi University — 30516: Theoretical Computer Science | Handout LN3
Professor Luca Trevisan | September 18, 2019

Notes for Lecture 3

October 7: corrected typo in pseudocode of "select" algorithm

Today we discuss an elegant and efficient divide-and-conquer algorithm for finding the median of a sequence of numbers, and we begin to talk about graphs

1 Median in Linear Time

The median of a sequence $A = A[1], \ldots, A[n]$ is the value that would appear in position $\lceil \frac{n}{2} \rceil$ after sorting the sequence. For example the median of $5, 1, 150, 7, 2$ is 5.

More generally, we say that a value has rank k in a sequence $A[1], \ldots, A[n]$ if, after sorting the sequence, we would find that value in position k. For example the minimum has rank 1. Note that a sequence can have repeated values, so the value of rank k could be the same as the value of rank $k + 1$.

Given a sequence $A[1], \ldots, A[n]$ and a parameter k, we would like to find the value of rank k in the sequence.

One possible approach is to sort the sequence and look up the value in the k-th position after sorting. This would take time $O(n \log n)$. We will describe a randomized algorithm whose expected running time is $O(n)$.

The divide-and-conquer idea is the following. Choose a position i in the sequence (we will discuss later how to choose it), and call $v = A[i]$ the value we see in this position. In time $O(n)$, count how many values in the sequence are smaller than v, equal to v, and bigger than v; call these counts s, eq and b, respectively.

If we sorted A, we would then see s elements smaller than v, then we would see v repeated eq times, and finally we would see b elements bigger than v. Now let us think of where the position k falls.

- If $s + 1 \leq k \leq s + eq$, then the rank k value is v and we are done.
- If $k \leq s$, then the rank k value is one of the values smaller than v. So we can construct, in $O(n)$ time, the sequence S of s elements smaller than v, and recursively look for the rank-k element of this sequence.
- If $k \geq s + eq + 1$, then the rank-k value of the sequence is bigger than v, and it has rank $k - eq - s$ among the elements of A bigger than v. Thus can construct the list B of elements of A bigger than v and recursively look for the value of rank $k - eq - s$ there.

Note that, no matter how we choose i, each recursive call either terminates with the correct output, or creates a recursive call with a smaller input, because both s and b are smaller than n.

To make the algorithm run as fast we possible, we want the recursive calls to be made on subsequences as small as possible. Although there is a clever way of choosing i that leads to an $O(n)$ time algorithm, we will describe a simple and very efficient randomized strategy: we just pick i uniformly at random between 1 and n. This leads to the following algorithm

select(A, k)

- Let n be the length of A and choose i uniformly at random in $\{1, \ldots, n\}$
- Let s, eq, b be, respectively, the number of elements of A that are smaller, equal, and bigger than $A[i]$
- if $s + 1 \leq k \leq s + eq$ then return $A[i]$
- else if $1 \leq k \leq s$ then construct sequence S of elements of A smaller than $A[i]$ and return select(S, k)
- else construct sequence B of elements of A bigger than $A[i]$ and
  return select $(B, k - s - eq)$

To analyze the running time, the first observation is that the algorithm takes $O(n)$ time plus the time that it takes to execute one recursive call.

In order to understand the size of the input in the recursive call, let us a call a choice of i *good* if both s and b are at most $3n/4$. It is not difficult to see that a random i has probability at least $1/2$ of being good, because if $A[i]$ has rank between $n/4$ and $3n/4$ then i is good, and there are $n/2$ choices of i with such property.

The next observation is that, on average, the number of recursive calls made until a good i is selected is at most 2, because when a good event happens with probability p the average number of attempts until the event happen is $1/p$.

Now let $\mathbb{E}T(n)$ be the expected running time of the algorithm on a worst-case sequence of length n. We will bound $\mathbb{E}T(n)$ by considering the expected amount of time spent on recursive calls until the first good i is selected, plus the remaining time. The expected amount time spent until a good i is selected is $O(n)$, because on average we have two recursive calls, each taking time $O(n)$; the remaining time is at most $\mathbb{E}T(3n/4)$, because after the first good i is selected the input size is at most $3n/4$.

Putting this observations together, we have

$$\mathbb{E}T(n) \leq O(n) + \mathbb{E}T(3n/4)$$

and we can use the master theorem to solve it to $\mathbb{E}T(n) = O(n)$.

2 Graphs

We discussed the definitions of directed graph, undirected graph, paths, and of connected components in undirected graphs, and their representation as adjacency matrices or adjacency lists, following Section 3.1 of Dasgupta. We also started to discuss the DFS algorithm, but we postpone all material about DFS to the notes for Lecture 4.