Lecture - 18

Friday, 2 September 2016 (16:15 - 17:05)

Randomized Median, QuickSelect

Median of a set A of n numbers $\{a_1, a_2, \ldots, a_n\}$ is the kth largest element where $k = (n+1)/2$ if n is odd and $k = n/2$ if n is even. A more general problem is to find kth largest element in the given set where k can range from 1 to n. This problem is called *Selection* or *Quickselect*. So we see that finding median is a special case of Selection. Similarly, finding minimum and maximum is a special case of selection where k is 1 and n respectively.

We see that it is easy to implement selection by first sorting the given array and then finding the desired element as per k. Since sorting takes $O(nlogn)$ time, the complexity of selection thus implemented will be $O(nlogn)$. Our aim is to come up with an $O(n)$ algorithm for selection.

The algorithm that we will now discuss is based on divide and conquer technique, which chooses an element $a_i \in A$. This element is called *pivot element*. It then forms two sets A_l and A_g such that all the elements of A that are lesser than a_i go to A_l and all the elements of A that are greater than a_i go to A_g. The sizes of A_l and A_g indicate which of the two subsets has the desired element. We, therefore, discard one of the two subsets and continue with the other, thereby, reducing the size of the problem. Following is the Selection algorithm:

Algorithm 1: Selection Algorithm for finding kth largest element in an array

```
1  Selection(A,k);
   Input  : The array $A$
   Output: $k$th largest integer
2  for each element $a_j$ of $A$ do
3      Put $a_j$ in $A_l$ if $a_j < a_i$ ;
4      Put $a_j$ in $A_g$ if $a_j > a_i$ ;
5  end
6  if $|A| = k - 1$ then
7      The pivot element $a_i$ is the $k$th largest element
8  else
9      if $|A| >= k$ then
10         The $k$th largest element is in $A_l$ ;
11         Recursively call Selection$(A_l, k)$
12     else
13         The $k$th largest element is in $A_g$ ;
14         Recursively call Selection$(A_g, k - 1 - l)$ where $l = |A_l|$
15     end
16 end
```

Analysis of Selection:
We can observe that the running time of *Selection* depends on how we choose the pivot element.

Best Case: If the pivot element always divides the array into two equal parts, then at every iteration, we discard half of the array and hence the running time of the algorithm is given by the following recurrence relation:

$$T(n) = T(n/2) + cn \tag{1}$$

where $T(n)$ is the time to sort n elements. The recurrence has a solution $T(n) = O(n)$. Therefore, we see that the best case will happen if the pivot element divides the array into two parts at every iteration.

Worst Case: Worst case of the algorithm will happen when the pivot is always the minimum or the maximum element of the array, in which case, at every iteration the array size is reduced only by one. In such a case, the algorithm follows the following recurrence:

$$T(n) = T(n-1) + cn \tag{2}$$

This recurrence has a solution $T(n) = O(n^2)$.

So we see that in the worst case, the algorithm gives a running time of $O(n^2)$, which is even more than in the case of sorting the array and then finding the kth element. So it all depends on the choice of the pivot element.

Let us see if randomization can help us in reducing the time complexity of the algorithm.

Randomized Version:
In this case, we choose the pivot element a_i uniformly at random from the set A. We know that the worst case of the algorithm happens when the pivot is either the maximum or the minimum element of the set. Let us see when the pivot is chosen uniformly at random, what is the probability of getting the worst case time complexity.

Let E be the worst case runtime of Selection algorithm.
Let E_k be the event that the maximum or the minimum element of the array is chosen as pivot when the current array size is k.
Then, E can be defined as

$$E = \cup_{i=1}^{n} E_i \tag{3}$$

Our aim is to find the value of $P(E)$.
We have,

$$P(E) = P(\cup_{i=1}^{n} E_i)$$

Since we make indpendent choices at each level, we can write,

$$P(E) = P(\cup_{i=1}^{n} E_i) = \prod_{i=1}^{n} P(E_i)$$

We know that $P(E_1) = 1$ and $P(E_i) = 2/i$, we get,

$$P(E) = \prod_{i=1}^{n} P(E_i) = \prod_{i=1}^{n} \frac{2}{i}$$

$$P(E) = \frac{2^{n-1}}{n!}$$

which is the probability of worst case when we randomly select the pivot element, which is a very small value.

Now let us analyse the expected running time of the randomized selection.

1. We can think of the algorithm as a chain of recursive calls. We may divide the whole functioning of the algorithm into a sequence of phases, where a phase consists of multiple recursive calls. The work done in each phase is equal to the sum of the work done in the recursive calls. Our aim is to divide into phases intelligently. Before that, we define an element to be a central element, if at least 1/4th of the total elements are greater than it and at least 1/4th of the total elements are smaller than it. It gives us half of the array elements to be central elements. So, the probability that we choose an elemengt and it comes out to be a central element is 1/2. Therefore, to make sure that in a phase, we will choose some central element, we will need to make 2 recursive calls (i.e. 1/probability).

2. Let us number the phases from $0, 1, 2, \ldots$. Since the pivot element is going to be the central element, and hence 1/4th of the array is going to be discarded at each step, in phase k, the array size will be at most $n(3/4)^k$. And hence, the total number of phases will be at most $\log_{4/3} n$.

3. Let X_k be the random variable equal to the number of recursive calls in phase k . If X denotes the random variable that stores the number of steps taken by the algorithm, we can write

   $$X = X_0 + X_1 + \ldots$$

   .

   Number of steps required in each call $= cn(\frac{3}{4})^k$
   Number of calls in each phase $= 2$
   Total work done in each phase $= 2cn(\frac{3}{4})^k$

4. Expected value of X, i.e. $E[X]$ is given by:

$$E[X] = \sum_{k=0}^{\log_{4/3} n} E[X_k] \tag{4}$$

$$E[X] = \sum_{k=0}^{\log_{4/3} n} 2cn(\frac{3}{4})^k \tag{5}$$

$$E[X] <= 2cn \sum_{k=0}^{\infty} (\frac{3}{4})^k \tag{6}$$

$$E[X] <= 2cn \frac{1}{1 - 3/4} \tag{7}$$

$$E[X] <= 8cn \tag{8}$$

Therefore, the expected running time of Selection in case the pivot element is a *central* element, is $O(n)$.