FTP_Alg
Selection in linear time
Optional part (it will be not examinated)

jungkyu.canci@hslu.ch

14. October 2024

Order Statistics

- For a given set of n points in a total ordered order (example real numbers), the ***i*–th order statistic** is the i–th smallest element (in statistics the i/n–quantile).
- For example the minimum is the first order statistic (i.e. $i = 1$) and the maximum is the n–th order statistic.
- In this lecture we consider the following so called **selection problem**:

Input A set P of n (distinct) points and an integer i, with $1 \leq i \leq n$.
Output The element x that is larger than exactly $i - 1$ other elements of P.

- We could solve the selection problem in $O(n \log n)$ running time, simply by sorting the set by using heapsort or merge sort.
- We will see an algorithm that solve selection problem with an running expected time $O(n)$.

Minimum/Maximum

We determine the minimum of a set of n (distinct) numbers by considering $n-1$ comparisons.

```
MINIMUM(A)
min = A[1]
for i = 2 to A.length
    if min > A[i]
        min = A[i]
return min
```

The code runs in $\Theta(n)$. Since we have to compare $n-1$ values, to be sure to have a minimum, we can not reduce the running time.
Similar arguments and code apply for the maximum.
The general selection problem seems to be more difficult than just finding the minimum of a set. But, we will see an algorithm, which solves general problem in $O(n)$ expected time.

Selection in expected linear time

In the algorithm below we use RANDOMIZED–PARTITION, already used in Quick sort, which produces two subarrays $A[p, \ldots, q-1]$ and $A[q+1, \ldots, r]$, such that all elements in $A[p, \ldots, q-1]$ are less of the pivot $A[q]$ and the one in $A[q+1, \ldots, r]$ are greater of the pivot. The pivot $A[q]$ is randomly chosen.

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]
q = RANDOMIZED-PARTITION(A, p, r)
k = q - p + 1
if i == k            // the pivot value is the answer
    return A[q]
elseif i < k
    return RANDOMIZED-SELECT(A, p, q - 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```

RANDOMIZED–SELECT: Running Time

- Since RANDOMIZED–SELECT use RANDOMIZED–PARTITION, then the running time in the worst case is $\Theta(n^2)$. The worst case is when the partition splits into two subarrays where one has no elements. But this is an unlikely situation.
- We will see that the expected running time for RANDOMIZED–SELECT is $\Theta(n)$.
- We denote by $T(n)$ the random variable running time for RANDOMIZED–SELECT. The goal is to prove that the expectation $E[T(n)]$ is $\Theta(n)$.
- Recall that RANDOMIZED–PARTITION divides the array into two subarrays $A[p, \ldots, q-1]$ and $A[q+1, \ldots, r]$.
- Let us denote X_k the random variable that has value 1 if $A[p, \ldots, q-1]$ contains exactly k points and 0 otherwise.

- The cardinality of $A[p, \ldots, q-1]$ is an integer i with $0 \leq i \leq n-1$, all with probability $1/n$. Thus $E(X_k) = 1/n$.
- If $X_k = 1$, then the problem is divided in a subproblem of size $k-1$ or $n-1-(k-1) = n-k$. Thus:

$$T(n) \leq \sum_{k=1}^{n} \left(X_k \cdot T(\max(k-1, n-k)) + O(n) \right).$$

- By considering the expectation we obtain:

$$\begin{aligned}
E[T(n)] &\leq \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k)) + O(n)] \\
&= \sum_{k=1}^{n} E[X_k] \cdot E[T(\max(k-1, n-k)) + O(n)] \\
&= \sum_{k=1}^{n} \frac{1}{n} \cdot E[T(\max(k-1, n-k)) + O(n)]
\end{aligned}$$

We have used the fact that X_k and $T(\max(k-1, n-k))$ are two independent random variables.

Now we observe that

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lceil n/2 \rceil \\ n-k & \text{if } k \leq \lceil n/2 \rceil \end{cases}$$

if n is even each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears twice in the sum and if n is odd they appears twice as well and $T(\lfloor n/2 \rfloor)$ appears once. Thus we have proven that

$$E[T(n)] = \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} \left(E[T(k)] + O(n) \right).$$

By induction one can prove that $E[T(n)] = O(n)$ (exercise).

SELECT Algorithm

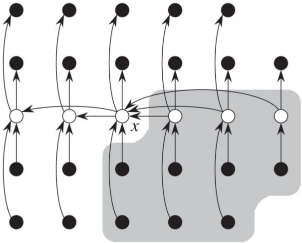
SELECT uses a deterministic partitioning algorithm PARTITION from quick sort, but it takes the pivot element around an input parameter. Let n be as usual the cardinality of the set where we operate the selection. SELECT executes the following steps.

1. Divide the n numbers in $\lfloor n/5 \rfloor$ groups, each with 5 elements and one with the remaining r points. (Note that $0 \leq r \leq 4$).
2. Find the median of each group, by using insertion–sorting.
3. Use SELECT recursively to find the median x of the medians find in the previous step.
4. Use the modified PARTITION to split the input array around the median of medians x calculated in previous step. Let k the number obtained by adding one to the cardinality of the low side of the partition. So x is the k–order statistics.
5. If $i = k$ return x. Otherwise if $i < k$ use SELECT recursively on the low side. If $i > k$ on the high side.

SELECT Running Time

Step 1, Step 2 and Step 4 have $O(n)$ running time. Step 3 takes $T(\lceil n/5 \rceil$. So in order to determine the running time $T(n)$ we have to estimate the time in Step 5.

At least half of the median in Step 2 are greater than or equal to the median x of the medians. At least half of the $\lceil n/5 \rceil$ contribute with at least 3 elements that are greater than x, except at most for the group with less than 5 elements and the one containing x.



Threrefore the number of the elements greater than x is at least $3\left(\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2\right)$, which is greater than or equal $\frac{3n}{10} - 6$.

Thus Step 5 calls SELECT recursively on a problem of size at most

$$n - \left(\frac{3n}{10} - 6\right) = \frac{7n}{10} + 6$$

which is greater than $\frac{3n}{10} - 6$.
Therefore we have proven that for n big enough, we have

$$T(n) \leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n).$$

We can obtain the recurrence

$$T(n) \leq \begin{cases} = O(1) & \text{if } n < 140 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n \geq 140 \end{cases}$$

The number 140 is given so that we are able to prove by induction that the running time is $O(n)$ (exercise).

Reference

The material of these slides is taken form the book "Introduction to Algorithms" by de Cormen et al., Section 9.1, Section 9.2 and Section 9.3.