Selection Problem · Week I

Data Structures: Theory and Practice

Monday 16th November, 2009

Johan Sejr Brinch Nielsen

Email: zerrez@diku.dk
Cpr.: 260886-2547

Dept. of Computer Science,
University of Copenhagen

1. Exercise 9.3-1

The running time of the randomized selection algorithm is described by the recurrence:

$$n < 140 : T(n) \leq O(1)$$

$$n \geq 140 : T(n) \leq T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) + O(n)$$

The running time of this algorithm is linear, since:

$$\frac{1n}{5} + \frac{7n}{10} + 6 = \frac{9n}{10} + 6$$

The combined sizes of the sub-problems are $\frac{9}{10}n + 6$, which is a fraction lower than n, for $n > 60$. For $n = 140$ this combined size is 132.

Groups of 7

When using groups of 7 the above equation for the running time with $n \geq 140$ becomes:

$$5\Big(\lceil \frac{1}{2} \lceil \frac{n}{7} \rceil \rceil - 2\Big) \geq \frac{5n}{14} - 10$$

Select is then called recursively on at most:

$$n - (\frac{5n}{14} - 10) = \frac{9n}{14} + 10$$

Which yields the running time:

$$T(n) = T(\frac{1n}{7}) + T(\frac{9n}{14} + 10) + O(n)$$

This choice of group size yields linear running time, since:

$$\frac{1n}{7} + \frac{9n}{14} + 10 = \frac{11}{14}n + 10$$

Again, the sub-problem is only a fraction of the original problem for $n \geq 47$.

Groups of 3

When using groups of 3 the above equation for the running time with $n \geq 140$ becomes:

$$1\Big(\lceil \frac{1}{2} \lceil \frac{n}{3} \rceil \rceil - 2\Big) \geq \frac{n}{6} - 2$$

Select is then called recursively on at most:

$$n - (\frac{n}{6} - 2) = \frac{5n}{6} + 2$$

Which yields the running time:

$$T(n) = T(\frac{1n}{3}) + T(\frac{5n}{6} + 2) + O(n)$$

This choice of group size yields linear running time, since:

$$\frac{1n}{3} + \frac{5n}{6} + 2 = \frac{7}{6}n + 2$$

Here the two sub-problems yields an even larger problem than the original. The recursion is no longer running in linear time.

2. Problem 9-4: Analysis of Randomized Selection

a) Number of comparisons

Let $E[X_{ijk}]$ be a measure for the number of expected comparisons between the ith and jth largest element, when locating the kth largest element.

I will now find an exact expression for $E[X_{ijk}$. First, observe that

- $k < i$ gives $i - k + 1$ elements (from k to i) that if chosen as pivot will cut z_i and z_j out of the problem (they are in the discarded subproblem). In this case, z_i and z_j will be compared if one of them are chosen as first pivot among $j - k + 1$ elements (from k to j).
- $i \leq k \leq j$ gives $j - i - 1$ elements that will cut either z_i or z_j out of the problem. In this case, z_i and z_j will be compared if one of them are chosen as pivot among $j - i + 1$ elements.
- $j < k$ is symmetric to the first case. z_i and z_j is compared if one of them are chosen as pivot among $k - i + 1$ elements.

Combined, these three cases yields the following number of expected comparisons:

$$E[X_{ijk}] = \frac{2}{\max(j - i + 1, j - k + 1, k - i + 1)}$$

Total Number of Comparisons

$$E[X_k] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{\max(j - i + 1, j - k + 1, k - i + 1)} =$$

$$\sum_{i=1}^{k} \sum_{j=k}^{n} \frac{2}{j - i + 1} + \sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} \frac{2}{k - i + 1} + \sum_{i=k+1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - k + 1} =$$

$$2\Big(\sum_{i=1}^{k} \sum_{j=k}^{n} \frac{1}{j - i + 1} + \sum_{i=1}^{k-2} \frac{k - i - 1}{k - i + 1} + \sum_{j=k+1}^{n} \frac{j - k - 1}{j - k + 1} \Big)$$

Clearly,

$$E[X_k] \leq 2\Big(\sum_{i=1}^{k} \sum_{j=k}^{n} \frac{1}{j - i + 1} + \sum_{i=1}^{k-2} \frac{k - i - 1}{k - i + 1} + \sum_{j=k+1}^{n} \frac{j - k - 1}{j - k + 1} \Big)$$

Linear Running Time

$$E[X_k] = 2\Big(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1} + \sum_{i=1}^{k-2}\frac{k-i-1}{k-i+1} + \sum_{j=k+1}^{n}\frac{j-k-1}{j-k+1}\Big) \leq$$

$$2\Big(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1} + \sum_{i=1}^{k-2}1 + \sum_{j=k+1}^{n}1\Big) \leq$$

$$2\Big(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1} + (k-2) + (n-k+1)\Big) \leq$$

$$2\Big(\sum_{i=1}^{\min(k,n-k)}\frac{1}{k} + \sum_{i=2}^{k}\sum_{j=k}^{i+k-1}\frac{1}{j-i+1} + \sum_{i=1}^{k-1}\sum_{i=k+i+1}^{n}\frac{1}{j-i+1} + n - 1\Big) \leq$$

$$2\Big(\frac{n}{2k} + \sum_{i=1}^{k-1}\sum_{i=k+i+1}^{n}\frac{1}{j-i+1} + \sum_{i=2}^{k}\sum_{i=k}^{i+k-1}\frac{1}{j-i+1} + n - 1\Big)$$

Well, that's all i got for now :-)

3. Selection Algorithm on Multi-Sets

One way to maintain the $O(n)$ running time on multi-sets (sets with non-distinct elements) is:

1. Find the median of the medians, as in the original algorithm
2. Partition the elements in two using the found median as pivot
3. If the left section (lower-than) is chosen, proceed as usual (no elements in this set is equal to the pivot)
4. If the right section (greater-than or equal-to) is chosen: 1) move all equal-to element to the beginning of the section with a linear scan and 2) if the needed median is available return it, otherwise recurse on the rest of the right section

Given the $<$ (less-than) operator one can test if an element a is equal to b by $!(a < b \vee b < a)$. When moving equal elements in the right section case, the equality of element e can be accomplished by $!(p < e)$, since $e \geq p$ is already known and

$$e \geq p \wedge !(p < e) \Rightarrow e \geq p \wedge p \leq e \Rightarrow p = e$$

This *trick* of locating equal elements ensures that recursion only occurs on either strictly lower or strictly greater elements (never equal). This maintains the upper bounds on the size of the remaining elements maintaining $O(n)$ running time.

4. Deterministic Linear-Time Selection Algorithm

I will now investigate the performance of deterministic linear-time selection algorithm. I benchmark this algorithm along with randomize selction (RND), STL selection, STL sort and selection using MergeSort.

These are my measured timings on random input:

Method	N = 2^{20}	N = 2^{22}	N = 2^{24}	N = 2^{26}
Det (g = 03)	0.095	0.36	1.41	5.59
Det (g = 07)	0.06	0.23	0.93	3.665
Det (g = 11)	0.055	0.22	0.875	3.56
Det (g = 15)	0.06	0.21	0.87	3.51
Det (g = 19)	0.055	0.225	0.885	3.605
Det (g = 23)	0.055	0.22	0.895	3.545
RND	0.01	0.06	0.23	1.03
STL sel	0.015	0.065	0.245	0.79
STL sort	0.12	0.52	2.275	9.855
mergesort	0.31	1.29	5.48	23.195

It seems clear, that the deterministic linear-time algorithm is not the fastest method. It does however beat other sorting-based algorithms. Of course the interesting observation is that the deterministic approach is slower than the randomized.

Just to be sure, I measured the timings on pre-ordered input too. I don't believe this will yield any significant change, since none of the algorithms have longer expected running time here:

Method	N = 2^{20}	N = 2^{22}	N = 2^{24}	N = 2^{26}
Det (g = 03)	0.09	0.36	1.4	5.615
Det (g = 07)	0.06	0.235	0.92	3.69
Det (g = 11)	0.055	0.225	0.89	3.575
Det (g = 15)	0.06	0.215	0.89	3.505
Det (g = 19)	0.055	0.225	0.93	3.61
Det (g = 23)	0.05	0.22	0.885	3.555
RND	0.01	0.06	0.24	1.015
STL sel	0.02	0.055	0.22	0.8
STL sort	0.115	0.51	2.295	10
mergesort	0.32	1.295	5.535	23.51

The overall result is the same. The deterministic method seems slower than the randomized. However this could be due to an inefficient implementation (they both have $O(n)$ theoretical running time). However, I would recommend the randomized version - it's simple to implement and seems very efficient. Of course, if the STL implementation is available this would be the choice.

5. Random-Sampling Selection Parameters

The random-sampling selection algorithm depends on the parameters α, β and γ. It works by performing a three-way partitioning on the input array, dividing it into 3 partitions. The hope is that the sought element ends up in the middle section, while the subsection defined by this section is easy. Hence,

some problems may arise:

1. The sought element is not in the middle section
2. The sought element is in the middle section, but this section is not small enough to solve efficiently

Since the parameters α, β and γ defines how the input should be divided, these greatly affects the performance of the algorithm.