Technische Universität München
Institut für Informatik
Scientific Computing in Computer Science

WS 2013/14
Worksheet
2.12.2013

Fundamental Algorithms 7

Exercise 1

Modify the algorithm SeqSearch, such that it works on sorted arrays (i.e., stops the search as soon as the elements become too large). Make a reasonable assumption for the probability that x is found at position i or not found at all, and give an estimate of the average number of comparisons that are required. How does the complexity differ from the "regular" SeqSearch algorithm?

Solution:

A modified version of SeqSearch is

```
SeqSearchMod (A:Array[1..n], x:Element) : Integer {
    for i from 1 to n do {
        if x >= A[i] then {
            if x = A[i] then return i;
            else return NIL;
        }
    }
    return NIL;
}
```

Note that this version only requires one (not two as straightforward approaches require) comparisons per loop iteration. Only the last iteration (if successful) requires two comparisons.

We use the same assumptions as in the lecture:

- x occurs either once or not at all in A
- The probability that x=A[i] is independent of the position i, which we denote by p

Thus, we can follow

- $p \leq 1/n$ in general;
  $\Rightarrow$ probability that $x \notin A$: $(1 - np)$
- $p = 1/n$, if $x \in A$

Nothing changed so far. Number of comparisons

- in case of success: $(i + 1)$
- in case of non-succes: average number of comparisons (if numbers uniformly distributed) $n/2$.

Therefore expected number of comparisons:

$$\bar{C}(n) = \sum_{i=1}^{n} p(i+1) + (1 - np)n/2 = p\left(\frac{n(n+1)}{2} + n\right) + (1 - np)n/2$$

Again, if $x \in A$: $p = 1/n$

$$\bar{C}(n) = \frac{n(n+1)}{2n} + 1$$

In terms of the average number of comparisons, we obtain even one more comparison as in the "uninformed" setting, if $x \in A$.

Exercise 2

For the algorithm BinarySearch, as discussed in the lectures, formulate a recurrence equation for the number of comparisons and solve the recurrence to estimate the time complexity of BinarySearch.

Solution:

Simplifications:

- We assume $n = 2^k$ for some $k \in \mathbb{N}$, thus we can neglect $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ as usual.

There are two comparisons per function call. The recurrence formula is

$$T(n) = T(\frac{n}{2}) + 2$$

There are plenty of ways to solve the recurrence formula:

1. Direct:

$$\begin{aligned} T(n) &= T(n/2) + 2 \\ &= T(n/4) + 2 + 2 \\ &= T(n/2^3) + 2 + 2 + 2 \\ &= \dots \\ &= T\left(\frac{2^k}{2^k}\right) + 2k \\ &= 2 + 2k = 2(k+1) \in \Theta(k) = \Theta(\log n) \end{aligned}$$

2. Master Theorem: $a = 1$, $b = 2$, $f(n) = 2$

$$\begin{aligned} & f(n) = 2 \in \Theta(n^{\log_2 1}) = \Theta(n^0) \\ \Rightarrow \quad & T(n) \in \Theta(n^{\log_2 1} \log n) = \Theta(\log n) \end{aligned}$$

3. Substitution method ...