RWTH Aachen **SS 2012**
Lehrgebiet Theoretische Informatik Problem Set 3
Langer—Reidl—Rossmanith—Sikdar 30.4.2012

Analysis of Algorithms — Tutorial

Problem 3-1

We analysed Quicksort on inputs that consisted of pairwise distinct keys in random order. Let us now see what happens if the input is ordered in reverse direction.

a) Analyse C_N for the case that the array contains $a[i] = -i$ for $i = 1, \ldots, N$. As always, we suppose that $a[0]$ contains a sentinel element that is smaller than all other keys.

b) Accordingly, analyse E_N.

Solution:

A close look at the algorithm reveals that the pivot element will always be the minimum or the maximum of the yet-to-be sorted portion of the array. Therefore, the partition of the N elements will always result in one empty partition and one of size $N-1$. Therefore we obtain

$$\begin{aligned} C_N &= N + 1 + C_{N-1} \\ &= \sum_{k=M+1}^{N+1} k \\ &= \frac{(N+1)(N+2) - M(M+1)}{2} \end{aligned}$$

In the end we are left with an array that is sorted correctly in the index interval $[0, L]$ and $[R, N]$ where $R - L \leq M$ (in the case of $N < M$ we have $L = 0, R = N$) and still in descending order in the interval $[L+1, R-1]$. In the sorted portions of the array the inner loop of the selection sort is not executed at all and therefore do not contribute to E_N. Let us say that the unsorted portion of the array has size $N' \leq M$. The selection sort itself moves the element at position i of the unsorted portion by $N' - i - 1$ steps, therefore

$$E_N = \sum_{k=0}^{N'-1} k = \frac{N'(N'-1)}{2} \leq \frac{M(M-1)}{2}$$

Problem 3-2

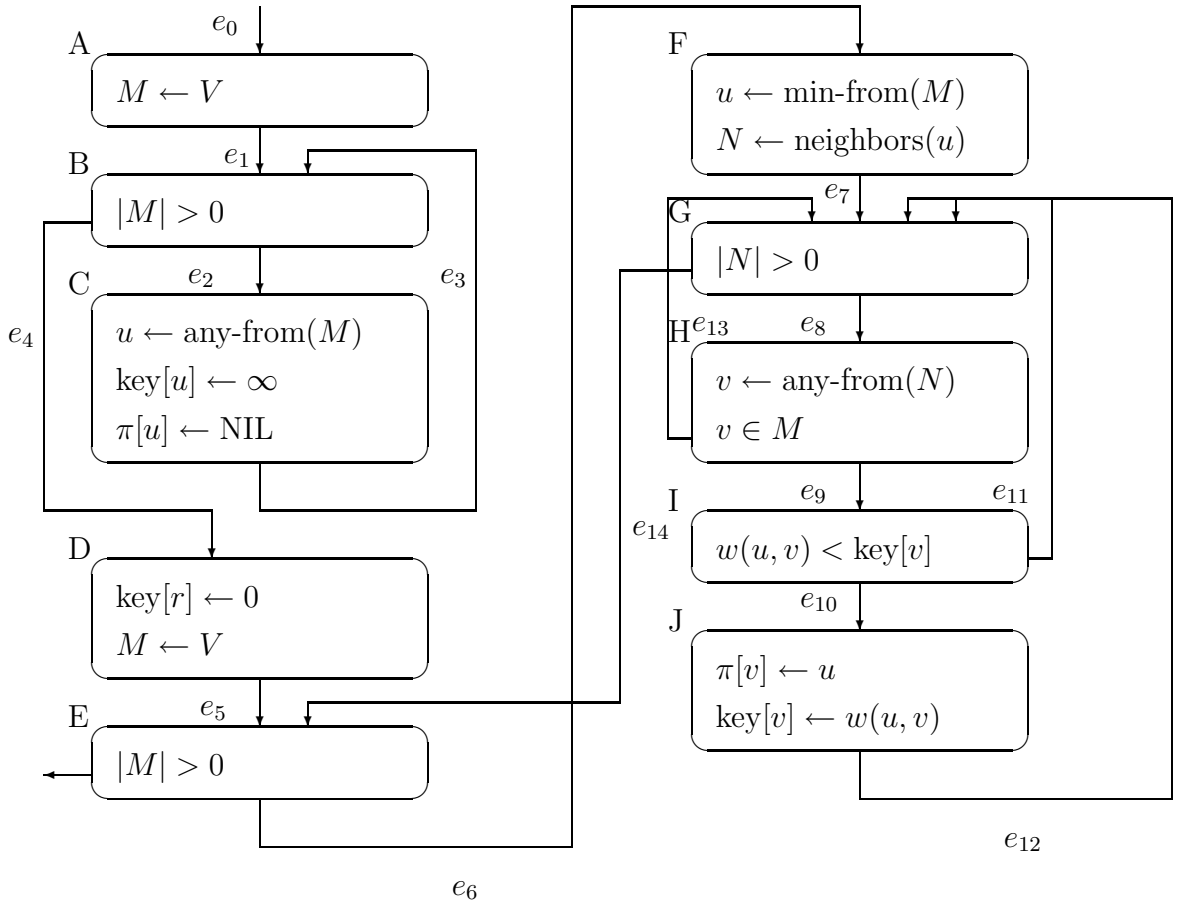
In this exercise, we consider Prim's Algorithm, which computes a minimum spanning tree. The input to this algorithm is a graph $G = (V, E)$, a weight function on the edges $w : E \to \mathbf{R}$ and a starting node r.

```
 1  for each u ∈ V do
 2      key[u] ← ∞
 3      π[u] ← NIL
 4  key[r] ← 0
 5  M ← V
 6  while (M ≠ ∅) do
 7      u ← min-from(M)
 8      for each v ∈ neighbors(u) do
 9          if (v ∈ M) ∧ (w(u, v) < key[v]) then
10              π[v] ← u
11              key[v] ← w(u, v)
```

Construct the control flow graph, a spanning tree in the control flow graph, the fundamental cycles, a corresponding linear system of equations and a solution to this system.

Solution:

The flow diagram is depicted below. The **for**-loops were changed, since the initializing and iteration condition must be separated.



We choose the spanning tree $e_1, e_2, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}$. This yields the following fundamental cycles:

$$
\begin{aligned}
C_0 &= e_0 + e_1 + e_4 + e_5 \\
C_3 &= e_3 + e_2 \\
C_{11} &= e_{11} + e_8 + e_9 \\
C_{12} &= e_{12} + e_8 + e_9 + e_{10} \\
C_{13} &= e_{13} + e_8 \\
C_{14} &= e_{14} + e_6 + e_7
\end{aligned}
$$

We now use standard linear algebra to find a good set of blocks whose number of visits we need to compute: By E_i, $0 \leq i \leq 14$, we denote the number of times the program flow visits the edge e_i. With each fundamental cycle above we identify a vector C_i. Then the E_i can be written as a linar combination of the fundamental cycles, i.e.,

$$
(C_0, C_3, C_{11}, C_{12}, C_{13}, C_{14}) \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \end{pmatrix} = \begin{pmatrix} E_0 \\ E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \\ E_6 \\ E_7 \\ E_8 \\ E_9 \\ E_{10} \\ E_{11} \\ E_{12} \\ E_{13} \\ E_{13} \end{pmatrix}
$$

for appropriate values of $\lambda_1, \ldots, \lambda_6$. We select six independent rows and obtain the equation

$$
\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \end{pmatrix} = \begin{pmatrix} E_0 \\ E_2 \\ E_{11} \\ E_{12} \\ E_{13} \\ E_{14} \end{pmatrix} = \begin{pmatrix} 1 \\ C \\ I - J \\ J \\ H - I \\ G - H \end{pmatrix}.
$$

This means, we only need to compute the values of C, G, H, I, J for a complete analysis ($E_0 = 1$ is trivally known), and then all other values can be derived: We see that $E_0 = E_1 = E_4 = E_5$, $E_2 = E_3$, $E_6 = E_7 = E_{14}$, $E_{10} = E_{12}$. This implies $A = 1$, $B = E_1 + E_3 = C + 1$, $D = E_4 = 1$, $E = E_5 + E_{14} = 1 + G - H$, $F = E_6 = G - H$.

Homework Assignment 3-1 (10 Points)

Quicksort has been analysed by us for different inputs: In the lecture we considered random permutations and in Problem set 3 this week inputs that already ordered — a case that is particularly bad for Quicksort. In this exercise we look at another interesting special case:

a) Analyse C_N for the case that the array contains $a[i] = k$ for $i = 1, \ldots, N$, where k is some natural number (i.e., all keys are identical). We suppose that $a[0]$ contains a sentinel element that is smaller than all other keys, e.g., $a[0] = 0$ or $a[0] = -\infty$.

b) Analyse D_N and E_N under the same conditions. For simplicity, you may assume $N \in \{\, 2^j - 1 \mid j \in \mathbf{N} \,\}$.

Solution:

As the comparisons in the inner while loop of the partition phase always evaluate to false, the pointers i, j move in lockstep towards each other and cross when $i = \lceil N/2 \rceil$. The left partition then has size $\lfloor N/2 \rfloor$ and the right $\lfloor (N-1)/2 \rfloor$, from which we obtain the recurrence

$$C_N = N + 1 + C_{\lfloor N/2 \rfloor} + C_{\lfloor (N-1)/2 \rfloor}$$

Let us, for simplicity, assume that $N = 2^p - 1$ for some integer p. Then the recurrence simplifies to

$$\begin{aligned}
C_N &= N + 1 + 2C_{(N-1)/2} \\
&= N + 1 + \sum_{k=1}^{d} 2^d \left(\frac{N + 1 - 2^{d+1}}{2^d} + 1 \right) \\
&= N + 1 + \sum_{k=1}^{d} (N + 1 - 2^{d+1} + 2^d) \\
&= N + 1 + d(N+1) - \sum_{k=1}^{d} 2^d \\
&= N + 1 + d(N+1) - 2^{d+1}
\end{aligned}$$

where d is the point where the recursion stops (i.e. $N \leq M$). A quick calculation yields the value of d:

$$\begin{aligned}
&\frac{N}{2^d} \leq M \\
\Leftrightarrow\ &\log(N/M) \leq d
\end{aligned}$$

Which in turn gives us

$$C_N = N + 1 + \log(N/M)(N-1)$$

Finally, $D_N = E_N = 0$ as the inner loop of the selection sort is never reached in an array where all elements are the same.

Homework Assignment 3-2 (10 Points)

Consider the following program:

```
int selsort(int a[ ], int n) {
    for(int i = 0; i < n; i++) {
        int min = i;
        for(int j = i; j < n; j++) {
            if(a[j] < a[min]) {
                min = j;
            }
        }
        int temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
}
```

The input to this program is an array $a[0, \ldots, n-1]$ that contains n pairwise distinct integer keys in random order.

a) Explain how this program sorts the given array.

b) Analyse how often each instruction of the program is executed on average depending on n.

c) There is only one instruction whose analysis is not trivial. Which one is it?

Make a table for small values of n by hand that lists the results for this instruction. Compare the table entries with the results from your closed formula that you obtained in b).

Solution:

(The original program contained a small error with is fixed in this version)

The algorithm is, as indicated by the name, selection sort: it finds the minimal element of the yet-to-be-sorted part and appends it to the sorted part by exchange.

For part b), the outer loop is execute n times, therefore each statement not in the inner loop is executed that often. The if-statement is executed exactly $\frac{n(n+1)}{2}$ times, which leaves the min $= j$ statement.

The first comparison ($j = i$ to i) always fails, therefore the expected number of executions (cf. Problem 1-2) then is

$$\sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \frac{1}{j-i+1} = \sum_{i=0}^{n-1} \sum_{j=1}^{n-i-1} \frac{1}{j+1} = \sum_{i=0}^{n-1} (H_{n-i} - 1) = n(H_{n-1} - n)$$