Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik, Anja Rey

Exercise Sheet 4

for the lecture on

Advanced Programming and Algorithms – Part II

This exercise sheet contains exercises for self-study and discussion.
If you would like feedback on your solutions, please upload a PDF via ILIAS until Monday, 6th April, 12:30 pm.

Discussion in the exercise classes from 6th until 10th May, 2024.

Problem 1 to hand in: *Greedy Proof Attempt: Find the Error*

Consider *Selection Sort*, a greedy sorting algorithm that runs in quadratic time in the input size (even if the input is already sorted).

```
selection_sort($A[0..n-1], n$):
1 for $j \leftarrow 0$ to $n-1$ do
2     find index min of the minimum element in $A[j..n-1]$
3     swap $A[j]$ and $A[\text{min}]$
4 return $A$
```

Take a look at the following proof attempt that claims to show why the algorithm works. Determine and explain three crucial errors. Repair the errors.

> **Loop invariant:** In loop iteration j, the current element $A[j]$ is swapped with the minimum of the remaining subarray $A[j..n-1]$.
>
> **Proof by induction over j:**
>
> **Base case:** For $j = 0$, the minimum index of the whole array $A[0..n-1]$ is found (line 2). This minimum swaps positions with $A[0]$ (line 3).
>
> **Induction step:** In later iterations, the minimum is only found in the remaining subarray $A[j..n-1]$ (line 2). This is sufficient, because the beginning of the array is already sorted, and therefore saves running time. The minimum is swapped with $A[j]$ (line 3) to save the minimum at the current position j.
>
> All in all, by this loop invariant we obtain the following **termination case:** After n steps, the array is completely sorted.

Problem 2 as a programming exercise: *Greedy Activity Selection*

The greedy activity selection algorithm from the Lecture 4 is presented with an algorithmically precise, but rather abstract pseudocode. For implementation this leaves many options open to interpretation. Implement this algorithm. Which data structure do you choose for the input? Which sorting algorithm do you use?

Problem 3 for discussion: *Greedy Algorithm Design: Storage*

Consider the following setting. We have got several moving boxes, all of the same size, but with different weights. The moving transporter can only carry boxes up to a certain weight. For the first transport, we want to load as many boxes as possible.

Formulate a greedy algorithm that, given the weights of the boxes, determines which boxes to load for the first transport to maximise the number of boxes.

- Formulate the idea and provide the algorithm in pseudocode.
- Analyse the asymptotic running time.
- Prove that the algorithm indeed computes an optimal solution.

Problem 4 for discussion: *Critical Path*

In a waterfall model for project management, we come across the following setting. We consider an acyclic weighted, directed graph with nodes representing milestones of the project and directed edges denoting jobs and dependencies between the milestones. The edge weights represent the times required to perform the jobs.

Given such a graph and two nodes s with in-degree 0 and t with out-degree 0, a *critical path* is a simple path from s to t with maximal length.

Reminder: A *topological order* in an acyclic directed graph is a sequence of nodes $v_1, \dots, v_n$ such that, if there is an edge from v_i to v_j, $1 \leq i, j \leq n$, then $i < j$. In other words, there cannot be a path from a later vertex in the sequence to an earlier one.

a) What is an interpretation of a critical path in such a project setting?

b) Recap: Which graph algorithms do we know to traverse a graph and determine shortest paths?

c) How can we find a topological order of the nodes and what is the connection to this setting?

d) Based on known graph algorithms, develop a (greedy) algorithm that, given the graph above and two nodes s and t, computes the length of a critical path from s to t.

e) Analyse the running time of this algorithm.

f) Discuss the advantages and disadvantages of such a model.