Obligatory Exercises

This document contains the list of obligatory exercises of the course Efficient Algorithms and Problem Complexity, to be solved by students who did not pass upon first attempt. Please prepare your solutions as a single pdf document containing solutions to those exercises you did not successfully solve before, and submit that document here. Note that some of the exercises have been changed while others have been kept. In either case, the idea is that the $n$th exercise should be solved if you have missed the $n$th original exercise. The due date for all exercises is 31 March 2013. Please notify me by e-mail as soon as you have uploaded your solutions.

Exercise 1

The Cantor set is a mathematically highly interesting set of points in the unit interval $[0, 1]$. It is obtained by starting with the line segment $[0, 1]$, taking away the middle third (except the endpoints), so that the two line segments $[0, 1/3]$ and $[2/3, 1]$ remain, and repeating this process over and over again, always removing the middle thirds of the $2^n$ line segments obtained in the previous step. The Cantor set is the set of points that remains in the limit.

For this exercise, we want to look at lists of integers instead of sets of line segments. For simplicity, we consider only nonempty lists. Given such a list $l$, we want to consider only the first and last thirds, disregarding the elements in between. As in the original Cantor set, we want to repeat this recursively. In other words, we first discard the middle third of the entire list, then the middle thirds of the two remaining sub-lists, and so on. However, because lists are finite, we do not have to continue this process ad infinitum as in the real Cantor set, but can stop when arriving at single list items. Let us denote the resulting list by $C(l)$.

(1) Give a precise formal definition of $C(l)$, where you denote a list consisting of items $a_0, \ldots, a_n$ as $[a_0, \ldots, a_n]$. (The items $a_0$ and $a_n$ should correspond to the points 0 and 1 in the geometric setting.)

(2) Write down the pseudocode for an algorithm that takes $l$ as input and builds $C(l)$ while sorting it at the same time. Thus, the output is $C(l)$ sorted in ascending order. For this, identify a suitable standard algorithm and modify it slightly.

(3) Determine the running time of this algorithm by means of the Main Recurrence Theorem.

Exercise 2

This exercise is derived from Exercises 6.2.17-23 in the textbook.

Let $m3quicksort(a, i, j)$ be the version that uses the median of the three array
cells $a[i], a[(i + j)/2]],$ and $a[j]$ in order to partition $a[i, \ldots, j], \text{if } i < j \text{ (instead of always using } a[i] \text{ for that purpose).}$

(1) Write down suitable pseudocode for $m3\text{quicksort.}$ (If $m3\text{quicksort}$ uses the procedure $\text{partition}$ from the textbook without changes, you do not need to include that one.)

(2) Determine the recurrence relation that describes the running time of the algorithm $m3\text{quicksort}, \text{if it is applied to a sorted array of } n \text{ pairwise distinct elements.}$ Argue convincingly why this recurrence relation is correct (pay particular attention to the recursive calls in your argumentation!), and use it to determine the running time of the algorithm on this type of input.

Exercise 3

Explain Prim’s algorithm by applying it to an example graph. The graph should be small, yet interesting enough to result in a non-trivial run of the algorithm. Show the contents of the adjacency lists, the ‘parent’ array, and the minheap

- directly after initialization,
- at some point during the execution of the algorithm, illustrating how a node is added to the MST by showing the situation before and after, and
- upon termination.

Explain briefly what is going on.

Exercise 4

Give an example that illustrates Huffman codes and Huffman’s algorithm, consisting of the following parts:

(1) a set (or multiset) of frequencies,
(2) an optimal Huffman frequency tree that could result from applying Huffman’s algorithm to these frequencies, and
(3) another optimal Huffman frequency tree that could never result from applying Huffman’s algorithm to these frequencies, regardless of the specific implementation of the data structures used. Explain why this tree cannot result from the algorithm.

Note that you have to choose the frequencies in a way that makes (3) possible.

Exercise 5

Give a version of Floyd’s algorithm (written down in pseudocode) that works on digraphs $G = (V, E)$ and computes, for all pairs $(u,v)$ of nodes, the set

$$\text{OnAll}(u,v) = \{ w \in V \mid w \text{ occurs on every path from } u \text{ to } v \}.$$ 

Make plausible that the algorithm works correctly.

As an example, in the graph

```
  1  2  3  4  5  6
  1---2---5
       |  |
       |  6
       |---3
```
we have

\[
\begin{align*}
\text{OnAll}(5,5) &= \{5\}, \\
\text{OnAll}(5,6) &= \{5,3,6\}, \\
\text{OnAll}(6,1) &= \{1,6\}, \text{ and} \\
\text{OnAll}(1,6) &= \{1,2,3,4,5,6\}.
\end{align*}
\]

Note especially the last one! Logically, if there is no path at all between two
nodes, then every node lies on each of these (nonexisting) paths.

Hints:

1. While the original algorithm starts out with the \(n \times n\) adjacency matrix \(A\) of \(G\), your algorithm will have to use \(A\) to initialize another \(n \times n\) matrix, which it then iteratively modifies similarly to the way in which Floyd’s algorithm modifies \(A\). Make sure to choose the initial values carefully. (Have a look at the example above and ask yourself what are the nodes that lie on all those paths from \(u\) to \(v\) that do not pass any other node.)

2. Apart from the initialization, the only change required concerns the update

\[
\]

in the body of the main loop. Find suitable replacements for the operations \(\min\) and \(+\).

Exercise 6

Let \(s\) be the Knuth-Morris-Pratt shift table for a pattern \(p[0, \ldots, m]\) of length \(m + 1\). Thus, for \(0 \leq k \leq m\), \(s(k - 1)\) is the shift amount to be used when a mismatch occurs at position \(k\) in the pattern. Now, consider the shift table \(s’\) for \(pp\).

(1) What is the relation between \(s’(k - 1)\) and \(s(k - 1)\) for \(0 \leq k \leq m\)?

(2) For \(0 \leq k \leq m\), which upper and lower bounds can you give for \(s’(m + k)\)?

(Hint: A good lower bound will depend on \(s\). The lower bound ‘1’ is not
good enough to qualify as a correct answer.)

Motivate your answers convincingly!

Exercise 7

By definition, a decision problem is NP-complete if it is in NP and all other
problems in NP can be reduced to it by a reduction that runs in polynomial time. Explain why it is important to require that the reduction runs in polynomial time, as opposed to accepting any computable reduction. (Hint: Argue that the notion of NP-completeness that is obtained by dropping the polynomial-time requirement would be useless for learning anything about the difference between \(P\) and \(NP\).)

Exercise 8

Let \(M\) be a deterministic random access machine \(M\) that decides a problem \(A\).

For every input \(x\), let \(used_M(x)\) denote the set of addresses of those registers of \(M\) that occur at least once as the target of some write instruction during the computation that processes \(x\). If the computation consists of \(l\) steps, it can be represented as a sequence of \(l + 1\) configurations \(C_0, \ldots, C_l\), where

- \(C_0\) is the initial configuration,
- \(C_l\) is the final one, and
- each configuration is of the form \(C_i = (\kappa_i, R_i)\), where \(\kappa_i\) is the value of the program counter (the number of the next instruction to be executed),
and a mapping $R_i: \text{used}_M(x) \rightarrow \mathbb{N}$, where $R_i(j)$ is the contents of register $R_j$ at that point in time.

Now, for $m \in \mathbb{N}$, let $|m|$ denote the number of bits of the binary representation of $m$. (Thus, $|0| = 1$ and $|m| = \lfloor \log m \rfloor + 1$ for $m > 0$.) We say that the memory space consumed by $M$ with input $x$ is

$$\text{space}_M(x) = \max_{0 \leq i \leq l} \{ \sum_{j \in \text{used}_M(x)} |R_i(j)| \}.$$ 

Thus, $\text{space}_M(x)$ is the maximum number of bits ever stored in memory during the computation, not counting the bits that are only required for storing the input $x$. We say that $M$ runs in logarithmic space if $\text{space}_M(x) \in O(\log |x|)$ (where $|x|$ denotes the size of $x$). The class $\mathsf{L}$ is the set of all decision problems that can be solved in logarithmic space.

Prove that, if $M$ runs in logarithmic space, then it runs in polynomial time. Thus, $\mathsf{L} \subseteq \mathsf{P}$. (Hint: Ask yourself how many configurations $C_0,\ldots,C_l$ a computation of $M$ may consist of, and thus how many steps there can be in such a computation.)

Exercise 9

Choose one (and only one!) of the problems below and show that it is NP-complete. (Do not forget that NP-completeness requires containment in NP!)

**TWICE-3SAT**

**Input:** A propositional formula $\varphi$ in conjunctive normal form, such that each clause consists of exactly three literals (as in 3SAT).

**Question:** Does $\varphi$ have at least two different satisfying assignments?

**OMNI-SAT**

**Input:** A propositional formula $\varphi$ in conjunctive normal form (as in SAT) such that there is at least one variable that occurs in every clause of $\varphi$.

**Question:** Is $\varphi$ satisfiable?

---

1 For example, $(x_1 \lor x_2 \lor \neg x_4) \land (\neg x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_7)$ is an allowed input because $x_2$ occurs in all three clauses.