Re-examination Exercises

This document contains the list of exercises for the re-examinations of the 2014 offering of the course Efficient Algorithms and Problem Complexity. Please prepare your solutions as a single pdf document, and submit the document [here].

The deadline is 2014-04-09, 23:59 for all of them. Note that you only need to solve the exercises corresponding to those you failed on during the ordinary examination. Not all exercises are identical to the original ones, but Exercise \( n \) below corresponds to Exercise \( n \) in the original list of exercises. If you want me to grade your solution before the due date, please send me an e-mail and I will do that asap.

**Exercise 1 (due 2014-04-09, 23:59)**

For this exercise, recall that the binomial coefficient “\( n \) choose \( r \)” can be determined recursively by

\[
\binom{n}{r} = \begin{cases} 
1 & \text{if } r \in \{0, n\} \\
\binom{n-1}{r-1} + \binom{n-1}{r} & \text{otherwise}
\end{cases}
\]

for \( 0 \leq r \leq n \).

1. Write down the pseudocode of the algorithm that computes \( \binom{n}{r} \) according to the equations above and determine the recurrence relation that describes its running time in terms of the parameter \( n \) (i.e., disregard the parameter \( r \), which can be done since \( r \leq n \)).
2. Solve the recurrence relation.
3. Draw a computation tree and use it in order to explain why the algorithm is inefficient. (Hint: A reasonable case to study is the computation of \( \binom{6}{4} \).)
4. Use a dynamic programming approach to produce the pseudocode of an efficient solution to the problem and analyze its running time.

**Exercise 2 (due 2014-04-09, 23:59)**

Give an asymptotically optimal algorithm to sort \( n \) dates, all of which are drawn from the 20th century. Every date is represented in the format month-day-year format. Analyze the running time of the algorithm that you suggest. Why is it optimal?

**Exercise 3 (due 2014-04-09, 23:59)**

Consider a graph \( G = (V, E, w) \) where each edge represents a road and the corresponding weight gives the probability of not having an accident on that road when driving on it. We assume that these probabilities are independent of each other. The most reliable path between two nodes is defined as the (simple) path with the highest probability of not having accidents. Your task is to modify
Dijkstra's algorithm so that it can be used to find the most reliable paths from a given start node to the other nodes. (Hint: Recall that, when two events \( a \) and \( b \) are independent with probabilities \( P(a) \) and \( P(b) \), the probability of them occurring both is \( P(a \land b) = P(a) \cdot P(b) \).)

Here is the original algorithm as presented during the lecture:

\[
\text{Dijkstra}(\text{adj}) \quad \text{where} \quad V = \{1, \ldots, n\} \quad \text{and} \quad v_0 = 1, \text{using a minheap} \ h
\]

1: initialize \( h \) with \( \text{key}(1) = 0, \ u.\text{key} = \infty \) for \( u \in V \setminus \{1\} \)
2: \( \text{parent}(u) = \perp \) for all \( u \in V \)
3: \( u.\text{pending} \leftarrow \text{true} \) for all \( u \in V \)
4: \textbf{while} \( h \) not empty \textbf{do}
5: \( (u, \text{key}) \leftarrow h.\text{del}() \)
6: \( u.\text{pending} \leftarrow \text{false} \)
7: \textbf{for} \( (v, \text{weight}) \) in \( \text{adj}[u] \) \textbf{do}
8: \textbf{if} \( v.\text{pending} \land \text{key} + \text{weight} < v.\text{key} \) then
9: \( v.\text{key} \leftarrow \text{key} + \text{weight} \)
10: \( \text{parent}(v) \leftarrow u \)
11: \( h.\text{restoreHeap}(v) \)
12: \textbf{end if}
13: \textbf{end for}
14: \textbf{end while}
15: \textbf{return} \( \text{parent} \)

(1) Write down the modifications required for the given algorithm to find the safest paths. You are not required to re-write the algorithm. It suffices to indicate clearly which of the statements must be changed, and how.

(2) Apply your modified algorithm to the graph and the start node \( v_1 \). (Yes, the roads in your country are very dangerous!) Draw a table whose rows and columns are indexed by nodes and iterations of the outer loop, resp. For every node \( v \) and every iteration \( i \in \{0, \ldots, 1\} \) indicate the key of \( v \) (which should be ‘-’ if \( v \) has been removed from the heap) and its parent at the point in time where the while statement is reached. (Thus, \( i = 0 \) corresponds to the situation directly after initialization, \( i = 1 \) to the one after the first loop execution, and so on.)

(3) Based on (2), what is the most reliable path from \( v_1 \) to \( v_4 \)?

Exercise 4 (due 2014-04-09, 23:59)

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, which
would be a correct solution of the problem, cannot result from applying the
algorithm.

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

Exercise 5 (due 2014-04-09, 23:59)

A cyclic rotation of a string is obtained by chopping off a prefix and attaching
it at the end of the string. More formally, a cyclic rotation of a string v is
obtained if v can be decomposed into u = u1u2 such that v = u2u1. Clearly, in this case u is
also a cyclic rotation of v. For example, ALGORITHM and RITHMALGO are
cyclic rotations of each other. You are required to design a simple algorithm
called Cyclic_Rotation that uses string pattern matching in order to determine
in linear time whether a string is a cyclic rotation of another string.

(1) Given two strings u, v, construct strings u’, v’ such that v is a cyclic rotation
of u if and only if u’ matches v’. Explain why your method is correct and
illustrate this with the strings ALGORITHM and RITHMALGO.

(2) As a consequence, how efficiently can Cyclic_Rotation be implemented?

(3) Assume finally that you, instead of a pure yes/no answer, want to compute
a decomposition of u into u1 and u2 such that v = u2u1 in those cases where
v actually is a cyclic rotation of u. How can you do this?

Exercise 6 (due 2014-04-09, 23:59)

Recall the characterization of NP by means of polynomially bounded relations
and witnesses: a problem A is in NP if and only if there is a polynomially
bounded binary relation R ∈ P such that

A = \{x | \exists y: (x, y) \in R\}.

Explain why the relations used must be polynomially bounded for the result to
be correct. More precisely, argue why the existence of a binary relation R ∈ P
(which is not polynomially bounded) with

A = \{x | \exists y: (x, y) \in R\}

does not yield a polynomial-time nRAM that decides A if we apply the con-
struction in the proof of the characterization.

Exercise 7 (due 2014-04-09, 23:59)

Choose one (!) of the problems below and show that it is NP-complete.

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 \(x_2\)
Exercise 8 (due 2014-04-09, 23:59)

A friend of yours has defined a problem $A$ on natural numbers whose definition you do not quite understand. He then claims that $A$ is NP-hard and “proves” this claim by the following polynomial-time reduction $f$ from HAM to $A$: For a graph $G$, $r(G)$ is equal to the number of nodes plus the number of edges of $G$. Argue why this is not a reduction of HAM to $A$, regardless of the definition of $A$.

Exercise 9 (due 2014-04-09, 23:59)

Recall the problem CLIQUE:

Input: A graph $G = (V, E)$ and a number $k$.

Question: Does $G$ have a $k$-clique, i.e., a complete subgraph on $k$ nodes?

Assume that we find a quadratic algorithm that solves CLIQUE. Now we want to solve the function problem that, given a graph $G$, returns a $k$-clique of $G$, where $k$ is the largest number such that $G$ has a $k$-clique.

Describe a polynomial algorithm that solves this problem and analyze its running time.