Johan Karlander, KTH, CSC

Teoritenta i Algoritmer (datastrukturer) och komplexitet
för KTH DD1352–2352 2012-05-31, klockan 14.00–17.00

No aids are allowed. 12 points are required for grade E, 15 points for grade D and 18 points for grade C.

If you have done the labs you can get up to 4 bonus points. If you have got bonus points, please indicate it in your solutions.

The solutions can be written in Swedish.

1. (8 p)

   Are these statements true or false? For each sub-task a correct answer gives 1 point and an answer with convincing justification gives 2 points.

   a. Dijkstra's algorithm cannot generally be used with negative edge weights.
   b. There are known efficient algorithms for deciding if a graph is 3-colorable.
   c. The problem of deciding if a Turing machine M run on input x halts in less than 1000 steps is, in fact, decidable.
   d. Let us assume that Divide and Conquer-algorithm has a time complexity $T(n)$ given by the equation

   $$T(n) = 4T(\frac{n}{4}) + \Theta(n)$$

   Then the time complexity is smaller than $\Theta(n^2)$.

   Solution:

   a. True
   b. False
   c. True
   d. True

2. (3 p)

   Let us assume that we have a sequence S of n integers. We want to find the longest strictly increasing subsequence of S. The numbers does not have to be consequitive. For instance, if:

   $S = \{3, 0, 5, 2, 3, 3, 7, 1, 8, 0\}$

   the longest strictly increasing subsequence has length 5. Use Dynamic Programming to solve this problem. More exactly,

a. Define suitable subproblems.

b. Define an array containing the solutions to the subproblems.

c. Find a recursion formula that gives the solution.

Observe that you don't really have to find the longest subsequence, just the length of it.

Solution:

For a suitable solution see lecture notes to lecture 5.

3. (4 p)

The following is a description of Kruskal's algorithm for finding minimal spanning trees in a graph:

```
KRUSKAL(V, E, w)
(1)   A ← ∅
(2)   Sort E in increasing weight
(3)   foreach (u, v) ∈ E (in the sorted order)
(4)       if (u, v) does not form a cycle with any of the edges in A
(5)           A ← A ∪ {(u, v)}
(6)   return A
```

a. It can be shown that the step " if (u, v) does not form a cycle with any of the edges in A" can be decided by an algorithm $F(A, (u, v))$ that runs in time $|A|$. What is then the time complexity of Kruskal's algorithm in the form given above?

b. There is a modification of the algorithm with better time complexity. This modification handles the step " if (u, v) does not form a cycle with any of the edges in A" in a better way. How is that done? What time complexity do we get?

Solution:

a. We can set $O(|A|) = O(|E|)$. Then we get the complexity $O(|E|^2)$.

b. See lecture notes to lecture 3. We get the complexity $O(|E| \log |E|) = O(|E| \log |V|)$

4. (3 p)

In this problem we study reductions between different NP-problems. We will consider

HAMILTONIAN CYCLE

Input: An undirected graph G.
Goal: Is there a Hamiltonian cycle in G?

TSP (Decision form)

Input: A complete undirected graph G with weights $w(e)$ on each edge. A number K.
Goal: Is there a Hamiltonian cycle in G of length $\leq K$?

Here is an attempt to make a reduction from HAMILTONIAN CYCLE to TSP:

Let G be an instance of HAMILTONIAN CYCLE. We then give set $G' = G$, set $w(e) = 1$ for all edges e in G' and set $K = |V(G)|$ and give (G', w, K) as instance to TSP.

Informally, we take the graph G, add edge weights of size 1 to all edges and set K to the number of nodes in G.

However, this correction is not a correct one. Explain why and suggest how we can make a correct reduction from HAMILTONIAN CYCLE to TSP.

Solution:

The problem is that G' is not generally a complete graph so we have failed to give a correct instance to TSP. We can correct this by letting G' be a complete graph with the same set of nodes as G and set $w(e) = 1$ if $w \in G$ and $w(e) = |V| + 1$ if $w \notin G$ and, as before, setting $K = |V|$.

5. (2 p)

In this problem we study an optimization variant of SUBSET SUM. We are given a set $x_1, x_2, ...x_n$ of positive integers and a positive integer M. We can assume that at least one of the x_i are $\leq M$. We are trying to find a subset sum $\leq M$ but as large as possible, i.e. smaller than or equal to M but as close to M as possible.

Now look this greedy algorithm:

First sort the integers in increasing order. Then we find the largest k such that $\sum_{i=1}^{k} x_i \leq M$. We then return this value.

a. Show by an example that this algorithm sometimes fail to give the optimal solution.
b. There might, however, be some hope that the algorithm could work as an approximation algorithm, i.e., there might be some $B > 1$ such that the algorithm approximates within B. Decide if this is the case either by giving a B and showing that the algorithm approximates within B or proving that there is no B that works.

Solution:

We can take the very simple example of the numbers $1, 2$ and $M = 2$. The algorithm obviously fails to find the trivial solution. There can be no approximation quotient

B. Take $1, B+1$ and $M = B+1$. Then obviously $OPT = B+1$ and $APP = 1$ so $\frac{OPT}{APP} = B+1 > B$.