Johan Karlander, KTH, CSC

Teoritenta i Algoritmer (datastrukturer) och komplexitet för KTH DD1352–2352 2017-05-29, klockan 8.00–11.00 Solutions

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.

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. It is possible to tell in time $O(|E(G)|)$ if a graph is connected or not.

      TRUE. You can use DFS or BFS for this and they both have time complexity $O(|E|)$.

   b. Any problem that can be solved by a non-deterministic turing machine can also be solved by a deterministic turing machine.

      TRUE. A non-deterministic turing machine uses random choices during a computation. We can simulate all possible choices on a deterministic turing machine and see if some choice leads to an accepting computation.

   c. It is always impossible to find shortest paths in graphs when there exist edges with negative edge-weights.

      FALSE. It will only be problems for graphs with negative weight directed cycles.

   d. The problem PARTITIONING can be reduced to the problem SAT.

      TRUE. The problem PARTITIONING is an NP-problem and Cook's theorem tells us that all NP-problems can be reduced to SAT.

2. (3 p)

   Describe in detail how Mergesort works. Analyze the time complexity of the algorithm.

   Solution: See lecture notes or course book. For the complexity analysis you can either use the Master Theorem with $a = b = 2$ and $f(n) = n$ which gives time complexity $O(n \log n)$ or do a direct analysis.

3. (3 p) In this problem we consider an undirected, connected graph G. It has positive weights on all edges.

(a) Describe an algorithm for finding a minimal spanning tree in G.

Solution: Use Kruskal's algorithm or Prim's algorithm. You should provide a description of either of them.

(b) Let s be a vertex in G. Let T be a spanning tree in G such that for each vertex v in G, the unique $s - v$ path in T is a shortest path from s to v in G. Describe an algorithm that finds such a tree T.

Solution: Use Dijkstra's algorithm with s as start node. (You should describe the algorithm).

(c) Must T (as given in part b) be a minimal spanning tree? Give proof if you think so or give a counter example if you think it is not so.

Solution: The answer is NO. A very simple counter-example is to take a graph with three nodes s, a, b and set $w_{sa} = 2$, $w_{sb} = 2$, $w_{ab} = 1$. The shortest path tree will have edges $(s, a), (s, b)$ and weight 4, but the minimal spanning tree has eight 3.

4. (3 p) Carefully answer these questions about NP-problems:

(a) Explain why all problems in P are in NP.

Solution: There are several ways of showing this. One is to use the definition that problems in NP are problems that can be decided by non-deterministic turing machines that run in polynomial time. Problems in P can be decided by deterministic polynomial time turing machines. Since deterministic turings machines are special cases of non-deterministic turing machines, it follows that $P \subseteq NP$.

(b) Let us assume A is a problem that is known to be NP-complete and B is a problem that is in NP. If we want to show that B is NP-complete, should we make a reduction from A to B or from B to A? Explain why.

Solution: You should reduce from A to B. We know that B is in NP. We have to show that evere NP-problem C can be reduced to B. If we can show that $A \leq B$, then since $C \leq A$, we get $C \leq B$. Observe that it is pointless to show $B \leq A$ since we already know this from Cook's theorem.

(c) Explain why we demand that the reductions should have polynomial time complexity.

Solution: Let us assume that we allowed the reductions to be exponential. It can be shown that any NP-problem can be solved in exponential time. Let A be an NP-complete problem and B any problem in P. Let F be an algorithm for deciding A. Given an instance x to A we can define a reduction $R(x)$ like this: We run $F(x)$. If the answer us yes we output some yes-instance to B. If the answer is nu we output some no-instance to B. Some we can reduce A to B. In fact, we can reduce any NP-problem to any other NP-problem and any problem would be NP-complete. To avoid this anomalous behavior we demand that the reductions must be polynomial time.

5. (3p)

The problem INDEPENDENT SET is an NP-complete problem. The corresponding optimization problem is to find the size of a maximal independent set in a graph G. It can be shown that for general graphs this problem can not be approximated within any factor $1 < B < \infty$. But if we know that the graphs we consider have a certain *maximum degree* D (that is, D is a bound valid for all graphs we consider), we can construct an approximation algorithm that finds an independent set by choosing vertices with low degree. Describe such an algorithm in detail, analyze it's time complexity and show that it approximates within $D + 1$.

Solution: The idea is that you start with G and find a vertex v_1 of lowest degree, say δ. Remove this vertex and all it's neigbors. Call this new graph G'. Put δ into a set A. Repeat the procedure, and put chosen vertices into A, until we have removed all vertices. Then A will be our independent set.

How large can A be. This size is equal to the number m of times we remove a set. Let V be the number of vertices in G. At each step we remove at most $D + 1$ vertices. So $A = m \geq \frac{V}{D+1}$. Let Opt be the optimal size of an independent set. Then $Opt \leq V$. So we have $\frac{Opt}{D+1} \leq A \leq Opt$. So the algorithm approximates with $B = D + 1$.

Let $n = V$. We remove a set less than n times. We can assume that we have precomputed all degrees in time $O(n^2)$. In each step we have to find a node of lowest degree in time $O(n)$ and remove it and it's neighbors in time $O(n)$. The time complexity will be $O(n^2)$.