Johan Karlander, KTH, CSC

Lösningar till Teoritenta i Algoritmer (datastrukturer) och komplexitet för KTH DD1352–2354 2010-12-13 klockan 14.00–17.00

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. If an algorithm runs in time $O(2^{\log n})$ it actually runs in polynomial time.

   b. There are problems that cannot be solved by any Turing machine.

   c. For all $K > 1$, the problem of deciding if a graph can be colored with K colors is NP-complete.

   d. If we have a 3-SAT formula with 5 clauses, we can decide in polynomial time if it is satisfiable.

   Solution:

   a. True. In fact, the laws of the logarithm shows that $2^{\log n} = n^{\log 2}$.

   b. True. For instance, the Halting problem is undecibable.

   c. False. We can use the BFS-search to tell if a graph can be 2-colored.

   d. True. At most 15 variables can occur in the formula. So there are just 2^{15} possible assignments to consider.

2. (4 p)

   During the course we have studied some important classes of algorithms. Three of these are Divide and Conquer Algorithms, Greedy Algorithms and Dynamic Programming Algorithms. Give non-trivial examples of each of these three types of algorithms and describe them in detai. For each example, explain what makes it such an algorithm. (That is, for your example of a greedy algorithm you should explain exactly what makes it a greedy algorithm, and so on.)

   Solution: There are lot of possible solutions. Natural examples would be: Divide and Conquer: Mergesort Greedy: Kruskal's algorithm. Dynamic Programming: Dijkstra's algorithm or Bellman-Ford's algorithm.

3. (4 p)

   Let us assume that Q is a problem known to be NP-complete. We also assume that $P \neq NP$ so that $Q \notin P$. Let A be another problem.

a. Carefully explain why the existence of a reduction $Q \leq A$ shows that $A \notin P$ but $A \leq Q$ does not necessarily show that $A \notin P$.

b. Here is an *attempt* to reduce from SUBSET SUM to TSP. (Travelling Salesperson Problem.) Given an instance $\{a_1, a_2, ..., a_n\}$, M of SUBSET SUM we construct a complete graph G with n nodes. We choose n edges arbitrarily and give them weights $a_1, a_2, ..., a_n$. All other edges are given weight 0. We then define the instance of TSP as G, M.
Is this reduction correct or not?
Motivate your answer.

Solution:

a. We know that Q cannot be decided in polynomial time. If $Q \leq A$ and A could be decided in polynomial time then we could take an instance x of Q, reduce it to an instance $R(x)$ of A and decide in polynomial time if $R(x)$ is a yes-instance. Then we could decide in polynomial time if x is a yes-instance of Q. Impossible! On the other hand, if Q is NP-complete, we know that any problem in NP can be reduced to Q. So even problems in P can be reduced to Q since $P \subseteq NP$.

b. The reduction is not correct. SUBSET SUM will have a yes-instance if there are numbers with sum M. The TSP-instance will be a yes-instance only if there are edges in a Hamiltonian cycle with sum M. And, of course, it is simple to find examples when the first instance (SUBSET SUM)is a yes-instance and the second (TSP) is not.

4. (4 p)

When we are faced with a hard optimization problem, NP-complete or not, we can try to design an approximation algorithm. In this problem I will sketch an algorithm for the optimization variant of the NP-Complete problem CLIQUE. The algorithm is as follows: Given G, find a node v with highest degree. Put v into a set S. Remove v and all its non-neighbors from G. Repeat this step on the remaining graph until there are no nodes left. When the algorithm ends, S will be our clique.

a. Express the algorithm in pseudo code and calculate the time-complexity.

b. What is an approximation quotient? Explain what it would mean if the algorithm had an approximation quotient 2.

c. In fact, the algorithm does not have an approximation quotient 2. Give an example which shows this.

Solutions:

a. The time complexity will be $O(n^2)$, where n is the number of nodes. This will probably only be true if we use adjacency list and use a heap to keep track of the nodes with higheest degrees.

b. If our algorithm returns a clique of size APP and the optimal cliquue-size is OPT, then $\frac{OPT}{APP} \leq 2$.

c. Take a graph which has two parts. The first is a clique of size M. The second is a "star" with a node connected to K other nodes. (So this part just have K edges.) If $K > M - 1$ and $M > 2$ we get $APP = 2$ and $OPT = M$ so $\frac{OPT}{APP} = \frac{M}{2}$. So if $M \geq 5$ we get $\frac{OPT}{APP} > 2$.