Johan Karlander, KTH, EECS

Teoritenta i Algoritmer (datastrukturer) och komplexitet
för KTH DD2350–2352 2019-05-31, 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.

If you are registered on DD2350, 13 points are required for grade E. Bonus points from DD2350 are counted in the exam.

In all problems you can assume $P \neq NP$.

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) The problem of deciding if an undirected graph is bipartite is NP-Complete.

   FALSE. A variant of BFS solves this in polynomial time. Since $P \neq NP$ we get that an NP-Complete problem can not be solved in polynomial time. So this problem can not be NP-Complete.

   (b) If a Divide and Conquer-algorithm has a time complexity $T(n)$ given by the recursion formula

   $$T(n) = 3T(\frac{n}{3}) + cn$$

   then $T(n) \in O(n^3)$.

   TRUE. The Master Theorem gives that $T(n) \in O(n^{\log_3 3}) = O(n)$. Since $O(n) \subset O(n^3)$ we get $T(n) \in O(n^3)$.

   (c) It is possible to construct a Turing Machine that decides if there are negative cycles in a directed graph.

   TRUE. We can find algorithms that solve this problem. (For instance, one is given in the lecture notes.) Church's Thesis says that any algorithm can be implemented on a Turing Machine. So there must be a Turing Machine (perhaps a complicated one) that solves this problem.

   (d) All problems that are in NP are also in P.

   FALSE. In fact, it is the other way. All problems in P are also in NP.

2. (3 p)

   Let us assume that we have a directed graph G and we use the Ford-Fulkerson algorithm to find maximal flows between s and t in G.

(a) If all capacities for the edges are integers, then the maximum flow must have an integer value. Carefully explain why.

Solution: We can use induction. In fact we show something stronger. At each stage in the algorithm, the flow in each edge is an integer value. In the next stage we find an augmenting path and find the smallest capacity δ along this path. This capacity has the form $c(e) - f(e)$ or the form $f(e)$ for some edge e in the path. So δ must be an integer. The algorithm increases the flow with δ on the path. This means the the flow increases or decreases with δ on each edge in the path. So the flow is still an integer in each edge. Since the flow has integer value in each edge, the maximum flow given by the algorithm must also have integer value.

(b) We say that two directed paths are *edge disjoint* if there is no edge belonging to both paths. Let us assume that there is a set of m pairwise edge disjoint paths between s and t and m is maximal in this sense. Let us also assume that all edges have capacity 1. Then the maximum flow must have value m. Carefully explain why.

Solution: If we can find m pairwise edge disjoint paths we can obviously release a flow of size 1 on each path. This gives us a flow of size m. Let M be the size of a maximal flow. The obviously $m \leq M$. Now, on the other hand, assume that we run Ford-Fulkerson's algorithm and get a flow M. From a. we know that the flow in each edge must be 0 or 1. Remove all edges with flow 0. Among the remaining edges we still have a flow of size $M - 1$. We can obviously find a path from s to t in this graph. Remove all edges in this path. Then we still have a flow of size $M - 1$. (Here we use that the flow in each egde is 1.) Do this recursively. In this way we find a set of M edge disjoint paths. So $m \geq M$. This gives us $m = M$.

3. (3 p)

In the graph below the nodes are problems. An arrow like $A \to B$ indicates that there is a polynomial time reduction fram A to B. Observe that there could be more reductions than the ones indicated.

Let us assume that C is NP-Complete. Answer these questions:

a. Which problems must be NP-Complete?

b. Which problems could be outside NP?

c. Given P $\neq$ NP, which problems could then be in P?

Solution: The main facts to use are that if $A \to B$, then if B is in NP then A must be in NP and if A is NP-Hard then B must be NP-Hard

(a) C,D and E. These problems are such that you can find a path from C to them and a path from them to C. This guaranties that the problems are both in NP and NP-Hard.

(b) B and F. A problem could be outside NP if there is no path from the problem to a known NP-problem.

(c) A and F. A problem could be in P is there is no path from a known NP-Hard problem leading to the problem.

4. (3 p)

We can define a problem NEF (Not Equivalent Formulas) that takes two propositional logic formulas F_1 and F_2 as input. The goal is to decide if the formulas are not equivalent. So a yes answer means that the formulas are *not* equivalent.

(a) Show that this problem is in NP.

Solution: A Yes-certificate is a valuation (set of values for the variables) such that F_1 and F_2 have different truth values. Given a suggested valuation we can compute the truth values for the formulas in time that is polynomial in the sizes of the formulas. (But finding such a valuation could of course be more complicated.)

(b) Show how you can reduce SAT to this problem. More specific, given an instance ϕ to SAT, show how you can choose two formulas F_1, F_2 such that ϕ is satisfiable if and only if F_1 and F_2 are not equivalent.

Solution: Given ϕ we can set $F_1 = \phi$ and $F_2 = p_1 \wedge \neg p_1$. We see that this F_2 is unsatisfiable. So if ϕ is satisfiable then F_1 and F_2 are not equivalent and if F_1 and F_2 are not equivalent then $F_1 = \phi$ is satisfiable. (Observe that any unsatisfiable formula is equivalent to $p_1 \wedge \neg p_1$.)

Extra question for the sophisticated: What is the reason for choosing "not equivalent" instead of the perhaps more natural "equivalent". You don't have to answer this question and answering gives no points but it could be good to think about this.

Answer: If we use equivalent then we can find a valuation that is a No-certificate. But it is not at all obvious that we can find Yes-certificates. The problem with Equivalent belongs to a class called co-NP. It is not known if NP = co-NP or not.

5. (3p)

(a) In the course book and in the lecture notes there is a description of an *approximation algorithm* for the problem VERTEX COVER. Describe the algorithm. What is the approximation quotient?

Solution: For a description of the algorithm, see the lecture notes. The approximation quotient is $B = 2$.

(b) Instead of using this algorithm we might want to use another simple greedy algorithm: Given G, choose a vertex v of maximal degree. Put v in the vertex cover. Remove v and all edges on v. This gives us a new graph G'. Use the previous step recursively to get a vertex cover.

Solution: For instance, we can take the graph with $V = \{a, b, c, d, e, f, g\}$ and $E = \{(a, b), (a, c), (a, d), (b, e), (c, f), (d, g)$. In this case, the algorithm will choose a and 4 more vertices giving a vertex cover of size 4. But the optimal size of a vertex cover is 3.

Give an example when this simple algorithm does not give an optimal size vertex cover.

(c) A K-regular graph is a graph where each vertex has degree K. Let us assume that we have K-regular graphs with K *fixed*. Show that in this case the algorithm described in b. approximates VERTEX COVER within a factor B. Find a value for B as a function of K.

Solution: The essential thing here is to find a *lower bound* for the size of an optimal vertex cover. First, we observe that in a K-regular graph we have $|E| = \frac{K}{2}|V|$. Let OPT be the size of a minimal vertex cover. Since a vertex covers K edges (and no more) we have $OPT \geq \frac{1}{K}|E| = \frac{3}{2K}|V|$. Let APP be the size of the vertex cover given by our algorithm. We want to find an upp bound for $\frac{APP}{OPT}$. We have $\frac{APP}{OPT} \leq \frac{2K}{3}\frac{APP}{|V|}$. An obvious estimate is $APP \leq |V|$. This gives us $B \leq \frac{2K}{3}$. (Maybe stricter bounds can be found.)