Johan Karlander, KTH, EECS

Teoritenta i Algoritmer (datastrukturer) och komplexitet
för KTH DD2350–2352 2018-05-31, klockan 14.00–17.00
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.

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.

   TRUE. You can find a simple example with three nodes a, b, c and edges $(a, b), (a, c), (b, c)$ with $w(a, b) = 3, w(a, c) = 2, w(b, c) = -2$ where Dijkstra would give the wrong distance from a to c.

   (b) There are known efficient algorithms for deciding if a propositional logic formula ϕ is a *tautology* (always true) or not.

   FALSE. A formula ϕ is a tautology if and only if $\neg\phi$ is not satisfiable. So we can reduce the problem to SAT.

   (c) The problem of deciding if a Turing machine M run on input x halts in exactly 1000 steps is undecidable.

   (d) FALSE. We can just run the algorithm for 1000 steps and see if it has halted. This is clearly a decidable problem.

   (e) If a problem can be solved by an algorithm that uses polynomial time, it can be solved by an algorithm that uses polynomial space.

   TRUE. In fact, the polynomial time algorithm must be polynomial space too, since it cannot change more than one bit at each time step.

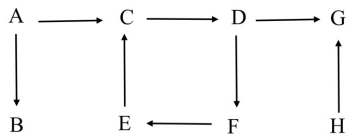
2. (3 p)

   Describe how Prim's algorithm works. This is an *efficient* algorithm. Explain what an efficient algorithm is and why this algorithm is efficient.

   Solution: See course book or lecture notes. The time complexity is $O(|E| \log |V|)$. That an algorithm is efficient means that it's time complexity is polynomial in input size. In this problem we can take the input size as $Max(|V|, |E|)$. Then, clearly, this algorithm is efficient.

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:

(a) C,D,E,F (X must be NP-Complete if there is a directed path from X to C and a durected oath from C to X.)

(b) B, G, H. (X can be outside NP if there is no edge from X to any NP problem.)

(c) A, B, H. (X can be in P if there is no edge from any NP-Complete problem to X.)

4. (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 do not have to be consecutive. 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 that contains 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.

5. (3p)

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 $< M$. We are trying to find a subset sum $\geq M$ but as small as possible, i.e. lager 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 $\sum_{i=1}^{k-1} x_i < M$ and $\sum_{i=1}^{k} x_i \geq M$. We then return $\sum_{i=1}^{k} x_i$.

(a) Show that this algorithm can be implemented as an algorithm that is polynomial in the input size.

(b) Show by an example that this algorithm sometimes fail to give the optimal solution.

(c) 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 the algorithm is an approximation algorithm with $B = \frac{3}{2}$ or not.

Solution: We can use a for-loop and compute the partial sums. We then compare the sums to M. Adding and comparing are $O(\log M)$ operations. So the time complexity will be $O(n \log M)$.
If we set $M = 10$ and have $x_1 = 1, x_2 = 10$, the algorithm will not generate the optimal solution.
If we set $M = 10$ and have $x_1 = 9, x_2 = 10$, the algorithm returns the number 19. The optimal value is 10. So for this instance we get $B = \frac{19}{10} > 1,5$. So this example show that we cannot always approximate within 1,5.