COMS 4231: Analysis of Algorithms I, Fall 2020

Problem Set 3, due Friday October 9, 11:59pm on Gradescope.

Please follow the homework submission guidelines posted on Courseworks.

- *In all problems that ask you to give an algorithm, include a justification of the correctness of the algorithm and of its running time.*
- *All time bounds refer to worst-case complexity, unless specified otherwise.*

Problem 1 [12 points] We are given a line L that represent a long hallway in an art gallery and a sorted set $X=\{x_1, x_2, \ldots, x_n\}$ of real numbers that specify the positions of paintings in this hallway. Suppose that a guard can protect all the paintings within distance 1 or less from his or her position (on both sides). Design an O(n)-time algorithm for finding a placement of guards that uses the minimum number of guards to protect all the paintings. Prove the optimality of your algorithm.

Problem 2 [22 points] Do Problem 16-2 (Scheduling) in CLRS, page 447. Make your algorithms as efficient as you can.

Problem 3 [22 points] Consider the following generalization of the activity selection problem. There are n activities where each activity a_i has a given starting time s_i, finishing time f_i , and positive weight w_i . Two activities a_i,a_j overlap if the corresponding open intervals (s_i, f_i), (s_j, f_j) overlap (intersect). We want to select a subset Q of non-overlapping activities with maximum total weight $w(Q) = \sum_{a_i \in Q} w_i$.

1. [6 points] Give counterexamples showing that the greedy algorithm with either of the following two selection criteria does not produce always optimal solutions:
   (i) Select an activity of largest weight.
   (ii) Select an activity with the earliest finishing time.

2. [16 points] Give an $O(n \log n)$-time algorithm that computes an optimal solution, i.e., a subset of non-overlapping activities with maximum total weight. Your algorithm should compute both the optimal weight and an optimal set of activities.

Problem 4 [20 points] A *palindrome* is a string that reads the same forward and backward; for example "racecar" and "lol" are palindromes. Give an $O(n^2)$-time algorithm to find the longest palindrome that is a subsequence of a given string of length n (if there are many such palindrome subsequences of maximum length, the algorithm can return anyone of them).

Problem 5 [24 points] An undirected graph $G=(N,E)$ is called *bipartite* if its set N of nodes can be partitioned into two subsets N_1, N_2 ($N_1 \cap N_2 = \emptyset$, $N_1 \cup N_2 = N$) so that every edge connects a node of N_1 with a node of N_2.

a. [6 points] Prove that if a graph contains a cycle of odd length then it is not bipartite.

b. [12 points] The converse to part (a) is also true, i.e., if a graph is not bipartite then it contains a cycle of odd length. In this part you will prove this algorithmically.
Give an algorithm, which takes as input a graph and either (i) determines that the graph is bipartite and outputs a bipartition of the nodes into two subsets N_1, N_2 such that every edge connects a node of N_1 with a node of N_2, or (ii) determines that the graph is not bipartite and outputs a cycle of odd length.
The algorithm should run in time O(n+e), where n is the number of nodes and e is the number of edges; the graph is given by its adjacency list representation.

c. [6 points] We are given a set of *non-equality* constraints of the form $x_i \neq x_j$ over a set of Boolean variables $x_1, x_2, ..., x_n$. We wish to determine if there is an assignment of Boolean values 0,1 to the variables that satisfies all the constraints, and compute such a satisfying assignment if there is one. Show that this problem can be solved in time O(n+m), where n is the number of variables and m is the number of constraints.

Optional Extra Credit Problem. [20 points] Many optimization problems on trees can be solved efficiently using dynamic programming. The method works in general as follows. The tree is rooted at some node and is processed bottom-up, from the leaves to the root, to compute the optimal value, and record sufficient information to recover an optimal solution. At each node v of the tree, a subproblem is solved for the subtree rooted at the node v, using the solutions to the subproblems for the subtrees rooted at the children of v (or sometimes even lower descendants). The algorithm can be usually written conveniently as a top-down recursive algorithm.

In this problem you will use this approach to solve the maximum weight independent set problem for trees. If T is a tree, and S is a subset of nodes of T, we say that S is an *independent set* of T if it does not contain any two adjacent nodes. The *maximum weight independent set problem* for trees is the following problem: we are given a tree T, where every node v has a given positive weight $w(v)$, and the problem is to compute an independent set S of T with the maximum possible total weight, $w(S) = \sum_{v \in S} w(v)$. You can assume that the tree T has been rooted at some node, and the input includes for every node v: the parent $p(v)$ of v, a list $C(v)$ of the children of v, and the (positive integer) weight $w(v)$ of v.

1. For each node v, let $T[v]$ be the subtree of T rooted at v, let $\alpha(v) = \max\{ w(S) \mid S$ is an independent set of $T[v]\}$, and let $\beta(v) = \max\{ w(S) \mid S$ is an independent set of $T[v]$ and $v \notin S\}$. Give (and justify) a recurrence relation that expresses the parameters $\alpha(v)$, $\beta(v)$ for v, in terms of the weight $w(v)$ of v, and the values of the parameters α, β for the children of v.

2. Give a O(n)-time algorithm for the maximum weight independent set problem for trees, where n is the number of nodes of the input tree. The algorithm should compute not only the maximum weight, but also an optimal solution itself (an independent set of maximum weight).