COMS 4231: Analysis of Algorithms I, Fall 2021

Problem Set 4, due Friday November 12, 11:59pm on Gradescope.

Please follow the homework submission guidelines posted on Courseworks.

- *As usual, for each of the algorithms that you give, include an explanation of how the algorithm works and show its correctness.*
- *Make your algorithms as efficient as you can, state their running time, and justify why they have the claimed running time. All time bounds below 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, …, 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 at most 1 of 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 generalization of the activity selection problem, where each activity a_i has a given positive weight w_i , and 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. 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. Give an algorithm that computes an optimal solution, i.e., a subset of non-overlapping activities with maximum total weight. Make your algorithm as efficient as you can. Your algorithm should compute both the optimal weight and an optimal set of activities.

Problem 4. [22 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).

Problem 5. [22 points] You are given a large rectangular piece of cloth with dimensions X×Y, where X, Y are positive integers, which you want to cut into smaller rectangular pieces that you can sell. There are n possible types of rectangular pieces that can be sold directly, where the i-th type has dimensions $a_i \times b_i$ and has price p_i , for i=1,…,n. Assume that the a_i, b_i, p_i are all positive integers. Pieces of cloth can be rotated, thus a piece with dimensions $b_i \times a_i$ can be rotated and sold also for p_i. You are allowed to produce multiple pieces of the same type (or none). Pieces that do not conform to the dimensions of any one of the n types cannot be sold directly by themselves (i.e., have price 0).
Your cutting machine can cut any rectangular piece completely into two pieces, either vertically or horizontally (i.e. parallel to one of the sides), where the cost of a cut is equal to the length of the cut; thus for example, cutting a piece with dimensions 4×7 parallel to the side of length 7 costs 7 and yields two pieces of dimensions c×7 and (4-c)×7 for some c (of our choice) where $0< c < 4$.
The input to the problem consists of the dimensions X, Y of the given piece of cloth, and the n tuples (a_i, b_i, p_i) of the dimensions and prices of the n types of clothes that can be sold. Design an algorithm that computes the maximum possible net profit that can be gained by a strategy for cutting the given piece of dimensions X×Y into smaller pieces that can be sold. The net profit is the sum of the prices of all the pieces that are produced minus the sum of the costs of all the cuts. Express the running time of your algorithm in terms of n, X and Y. Your algorithm should run in time polynomial in n, X and Y.