CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2019**

Homework #3

Deadline: Thursday, October 24th, 11:59 PM

Homework Policy

- If you leave a question completely blank, you will receive 25% of the grade for that question.
- You are allowed to discuss the homework problems with other students in the class. **But you must write your solutions independently.** You may also consult all the materials used in this course (notes, textbook, further reading, etc.) while writing your solution, but no other resources are allowed.
- Do not forget to write down your name and whether or not you are using one of your two extensions. Submit your homework on Canvas. Note that if you have already used both your extensions and still submit this homework late, **your homework will not be graded**.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "use counting sort to sort the array $A[1:n]$ with positive integer values smaller than M in $O(n+M)$ time", or "sort the array in $O(n \log n)$ time using merge sort".
- Remember to **always prove the correctness** of your algorithms and **analyze their running time**.
- For the **dynamic programming algorithms** in this homework, it is sufficient to write your specification and the recursive formula for computing the solution (i.e., you do *not* need to write the final algorithm using either memoization or bottom-up dynamic programming). Just remember to prove the correctness of your recursive formula and analyze the runtime of the resulting algorithm.

Problem 1. Recall that in the knapsack problem, we are given n items with positive integer weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$, and a knapsack of size W; we want to pick a subset of items with maximum total value that fit the knapsack, i.e., their total weight is not larger than the size of the knapsack. In the class, we designed a dynamic programming algorithm for this problem with $O(nW)$ runtime. Our goal in this problem is to design a different dynamic programming solution.

Suppose you are told that the *value* of each item is a *positive integer* between 1 and some integer V. Design a dynamic programming algorithm for this problem with worst case $O(n^2 \cdot V)$ runtime. Note that there is no restriction on the value of W in this problem. **(25 points)**

Hint: Think of the following subproblems for your specification: what is the *minimum* size of a knapsack needed to collect j *values* from a subset of items $\{1, \ldots, i\}$? Having this, you can pick the largest value of j as the solution such that the answer to this problem is smaller than or equal to W.

Problem 2. You are given a set of n boxes with dimensions specified in the (length, width, height) format. A box i fits into another box j if *every* dimension of the box i is *strictly smaller* than the corresponding dimension of the box j. Your goal is to determine the *maximum length* of a sequence of boxes so that each box in the sequence can fit into the next one.

Design an $O(n^2)$ time dynamic programming algorithm that given the arrays $L[1:n]$, $W[1:n]$, and $H[1:n]$, where for every $1 \leq i \leq n$, $(L[i], W[i], H[i])$ denotes the (length, width, height) of the i-th box, outputs the length of the longest sequence of boxes $i_1, \ldots, i_k$ so that box i_1 can fit into box i_2, i_2 can fit into i_3, and so on and so forth. Note that a box i can fit into another box j if $L[i] < L[j]$, $W[i] < W[j]$, and $H[i] < H[j]$ (you are *not* allowed to rotate any box). **(25 points)**

Example: Given the boxes with dimensions $[(1,2,2),(5,3,2),(9,4,5),(2,3,4)]$, the correct answer is 3 as we can fit the box $(1,2,3)$ inside $(2,3,4)$, and $(2,3,4)$ inside $(9,4,5)$.

Problem 3. You have a bag of m cookies and a group of n friends. For each of your friends, you know the "greed factor" of your friend (denoted by g_i for your i-th friend): this is the minimum number of cookies you should give to this friend to make them stop complaining. Of course, you would like to find a way to distribute your cookies in a way to minimize the number of your friends that are still complaining.

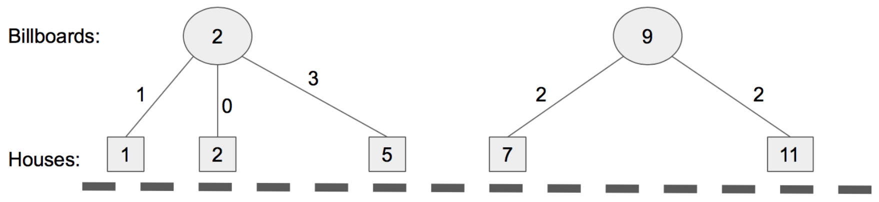
Design an $O(m+n)$ time greedy algorithm to find an assignment of the cookies to your friends so as the minimize the number of the friends that are still complaining: recall that a friend i stops complaining if we assign them c_i cookies and $c_i \geq g_i$. **(25 points)**

Hint: First design your greedy algorithm without worrying about its runtime. Then use your knowledge from the previous lectures (think of sorting algorithms) to make it run in $O(m+n)$ time.

Problem 4. There is a straight highway with n houses alongside it. You have developed a brilliant new product and would like to advertise it by placing billboards alongside this highway. Since constructing billboards is expensive, you would like to choose as few billboard locations as possible such that every house is within at most d units of a billboard.

Design an $O(n \log n)$ time greedy algorithm which given an array A of real numbers representing the locations of the houses alongside the highway (house one is placed on location $A[1]$, house two on $A[2]$, etc.), and a maximum distance d, outputs a *minimum* set of locations (i.e., real numbers) for placing the billboards such that each house is at most d units from some billboard. **(25 points)**

Example: If the houses are placed in locations $[1, 5, 2, 7, 11]$ and $d = 3$, we only need two billboards: we can place one at location 2 and another at location 9 – this way, the first three houses will be at distance at most 3 from the first bill board, and the last two will have distance at most 3 from the second billboard. See the figure below.



Challenge Yourself. Let us revisit Problem 4. Instead of minimizing the number of billboards so that every house is within distance d of some billboard, we now have a budget of b billboards in total. Our goal is again to place the b billboards in proper locations such that the *maximum* distance of every house from some billboard is *minimized*.

Design an algorithm for this problem that given the locations of the houses in the array $A[1:n]$ and the budget b, outputs an assignment of these billboards to the locations so that the maximum distance, among all houses, from their closest billboard is minimized.

Example: If the houses are placed in $[1, 5, 2, 7, 11]$ and $b = 2$, we can place one billboard at location 3 and another at location 9 and the answer would be 2 – every house is within distance 2 of some billboard. For the same houses, if instead we have $b = 1$, we can place a billboard in location 6 and have the distance of at most 5 for every house.