Homework 9

Due Wednesday, April 13th at 11:59 pm ET on Gradescope

Solution guidelines. For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and pseudocode (*),
2. a proof of correctness,
3. an analysis of the asymptotic running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class, e.g. correctness of subroutines, running time of subroutines. You should be as clear and concise as possible in your write-up of solutions.

(*) It is fine if the English description concentrates on the high level ideas and doesn't include all the details. But the reader should not have to figure out your solution solely based on the pseudocode. You can also add comments to your pseudocode, in fact that is best practice. FYI we will share a document on good pseudocode style with you and it will also be discussed in labs and lecture.

Format for writing a solution for a DP problem. Dynamic programming solutions follow a pretty strict pattern. Here we give you a guide for what should be included in a full credit solution.

1. Clearly state what the subproblems are and what corresponds to the final solution. *For WIS write OPT(i) is the value of the max compatible schedule if we consider the first i jobs* $1, 2, \ldots i$ *in increasing order of finish time. OPT(n) is the solution on the entire input.* Note, that for some problems OPT will have 2 variables, e.g knapsack problem.
2. Write and explain the recursive formula to compute OPT(i). A proper proof would include induction on i (or on both variables). However, for these kind of problems it is pretty clear what the inductive steps and assumption are. For this reason we accept if you say that assuming the formula is correct for every lower index, then my formula is correct because XYZ. Don't forget about the border cases in your formula.
3. clearly state the dimensions of the memoization table and what its values are, e.g. M[i][j] = OPT(i,j).
4. Brief pseudocode, short and clean is better. It should be clear from your code what the dimensions of the table are, how you initialize the border case, in what order you iterate over the table to fill the values (both recursive and bottom-up are valid) and how the recursive formula is used. Any preprocessing steps, e.g. order the jobs, should be written too.
5. running time analysis.

Problem 1. *Pizza buffet (10 points)*

It's your birthday, and your friends take you to your favorite pizza buffet for lunch. There are n different types of pizza, and in true all-you-can-eat fashion the restaurant can indefinitely replenish any pizza type that runs low. A slice of pizza type i has a value $f_i > 0$ of how much it will fill you up, and a happiness value $h_i > 0$ you will obtain by eating it (even boring old cheese pizza is better than nothing!). Your stomach has a total capacity of C. Obviously you want to maximize the happiness you gain from the pizza you eat.

1. A greedy strategy is to always choose whichever slice of pizza that you have capacity for that will increase your happiness the most. Show an example where this fails to maximize happiness.

2. Design a dynamic programming algorithm to decide which pizza slices to choose based on the capacity of your stomach. For full credit, your algorithm should run in $O(nC)$ time.

   (a) Identify the subproblems in your solution. It's often helpful to think about the recursive formula: what subproblem is it that you make the recursive calls to?

   (b) Write the recursive formula to compute the optimal value for each subproblem. Explain why your formula works.

   (c) Write the dynamic programming algorithm using memoization for efficiency.

   (d) Analyze the running time.

   (e) Write the backtracking algorithm to find which pizza slices you should eat.

Problem 2. *Cell towers (10 points)*

A company operating cell towers needs to install towers along a long country road connecting two cities. Help them find the best location for the towers! Each installed tower covers an area of 5 miles in radius (i.e. 5 miles on either side of the tower). There needs to be cell coverage along the entire road. There is infrastructure to install a tower at every milestone along the road. The total distance between the cities is n miles. (You are allowed to place a tower in either city.) The cost of installing towers varies by location. The costs of the different locations are given to you in an array `cost[ ]` of the length $n + 1$. (The costs are positive numbers.) You need to find a set of building sites such that the entire road has cell coverage and the total cost is minimized.

1. Observe that it is possible that the minimum cost solution consists of stretches of road that are covered by multiple towers. Prove however, that in any *optimal* solution, each stretch is covered by at most 2 towers.

2. Consider a greedy algorithm where we add locations to our solution set one at a time. We consider the stretches of road (the road between two neighboring tower locations) in increasing order of distance from the first city. Each time we encounter a so far uncovered road stretch r_i (that is between miles $i - 1$ to i), we will add a tower location j that (1.) covers r_i (2.) has the best cost-coverage ratio. That is, we choose a location j such that the relative cost $\frac{\texttt{cost[j]}}{\texttt{additional miles}}$ of every additional mile covered by this tower is minimized.

   Show an example where this algorithm fails.

3. Here is the pseudocode for a memoized algorithm to find the *minimum cost* of locations. (Note that it does not give you the actual set of locations.)

```
Algorithm 1: DPMinCostTowers(cost, n)
1 $M \leftarrow$ empty array of length $n$;
2 for $i = 0 \ldots n$ do
3     $M[i] \leftarrow$ {some appropriate formula or small block of pseudocode to be defined by you};
4 return $M[n]$;
```

What does the entry `M[i]` contain (that is, state the subproblem for which it provides a solution)?

Write the recursive formula to compute `M[i]` in line 3 of the algorithm. (Write the formula only, no need for explanation.)

4. Write an algorithm that takes the table M filled in by Algorithm 1 and returns a list of the locations where towers should be built.