Fun with Dynamic Programming

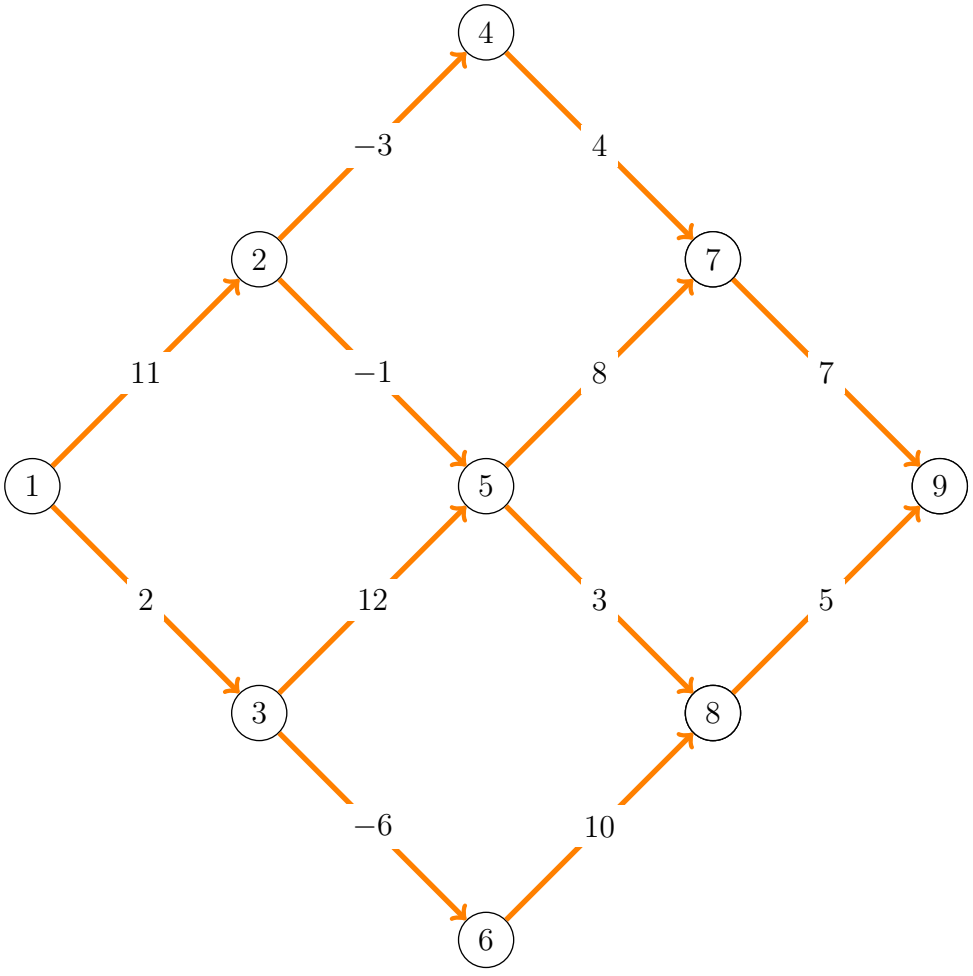
Due: 11:00pm on November 4th, 2022

100 points

1. **(70 points)** - The Mountain Marathon people have contacted you. They've mapped out a potential set of routes but want to maximize the number of participants. This problem is best seen with an example. The file you read in will be formatted in the following fashion:
   turnPenalty numberOfNodes $e_{1,2}$ $e_{1,3}$ ... $e_{n-2,n}$ $e_{n-1,n}$

   The input file will always be formatted correctly, i.e. the input will always be a diamond shape that is k by k. Assume the input file looks like the following:
   -20 9 11 2 -3 -1 12 -6 4 8 3 10 7 5

   The graph looks like the following:



The edge weights represent the number of extra runners the planners expect to get if they choose that edge to be in the final route. The edge weight may be negative because a particular path may be undesirable. The goal of your program is to maximize the path from 1 to n. Some requirements your program must account for:

(a) You must always go "right." All edges are directed edges and lead to the "right" in the above graph. For example, you can go from vertex 2 to vertex 5. However, you are not allowed to go from vertex 5 to vertex 3.

(b) Runners generally hate turns during a race. Whenever you change direction, you incur a penalty based on what the input file says. There is no penalty for deciding whether to start by going from 1 to 2 or going from 1 to 3. However, if you choose to go from 1 to 2 and decide to change the direction and go from 2 to 5, you incur a penalty. You can think of the direction from 1 to 2 as going "up" and you turned to go "down" to 5.

There are 6 paths in the above example. The following are the number of extra runners for each path:

(a) $\{1, 2, 4, 7, 9\} = -1$. This was obtained by adding 11 + (-3) + (-20) + 4 + 7. The -20 is because of the turn at vertex 4.

(b) $\{1, 2, 5, 7, 9\} = -35$. This was obtained by adding 11 + (-20) + (-1) + (-20) + 8 + (-20) + 7. The -60 is because of the turn at vertex 2, vertex 5 and vertex 7.

(c) $\{1, 2, 5, 8, 9\} = -22$.

(d) $\{1, 3, 5, 7, 9\} = -11$.

(e) $\{1, 3, 5, 8, 9\} = -38$.

(f) $\{1, 3, 6, 8, 9\} = -9$.

Your final output should be -1. You do not need to print out the path but only need to print out the maximum weighted path. The answer may be different if a different penalty were used. You should not assume the turn penalty is negative. The turn "penalty" may be a positive number because the planners may assume that runners want a lot of turns.

The only requirement for this problem is that it must run quickly on a very large input file where the number of nodes exceeds 1,000,000. There are no other specific requirements for this problem.

2. **(30 points) - For this problem, you must submit a problem individually.** Do not discuss your solutions with anyone else. You must write up your solutions to this assignment using LaTeX. You must also upload a separate file with your code.

   The "Rod cutting" problem is a classic optimization problem in Computer Science. In this problem, pretend that you work for a company that buys long steel rods and

cuts them into shorter rods, which it then sells. Each cut is free. Your company wants to know the best way to cut up the rods.

Therefore, we assume that we know, for each $i = 1, 2, \ldots$, the price p_i in dollars that your company charges for a rod of length i inches. Rod lengths are always an integral number of inches.

Given a rod of length N inches, and a table of prices p_i, for $i = 1, 2, \ldots, N$, determine the maximum revenue $r[N]$ obtainable by cutting up the rod and selling the pieces. Note that if the price p_N for a rod of length N is large enough, then an optimal solution may require no cutting at all.

For the rest of this assignment, you must design, implement and evaluate the efficiency of a dynamic programming solution for a variation of the standard Rod cutting problem. In this variation, we will assume that you are given G an amount of gold coating material that can be used to coat G inches in gold, where G is a fixed constant, independent of N. Coating a cut segment in gold will double the payoff that you receive for that segment. However, fractional portions of segments cannot be coated. That is, segments must be entirely coated, or not coated at all. You can't coat half a segment, or three-quarters of a segment, etc. For example, if $G = 3$, then you can coat three one-inch segments, one two-inch segment and a one-inch segment, or one three-inch segment. While you aren't required to coat any cut segments in gold, the overall goal is to compute the maximum revenue.

See the rubric at the end of this document for more specific requirements. Note that, in addition to the requirements listed in the rubric, I will also be grading your implementation based on its asymptotic worst case running time. The questions begin on the next page.

(a) As a warm-up, what is the optimal solution for the following instance of the problem with $N = 10$, $G = 4$ and prices: $[1, 14, 35, 44, 45, 60, 91, 110, 117, 158]$? Give the revenue as well as describe where the cuts should be made and which segments should be coated in gold.

Put your answer here.

(b) Start by developing a valid recurrence that characterizes the optimal solution for a given rod of length N, with prices p_i, for $i = 1, 2, \ldots, N$ and G inches worth of gold coating, from which a corresponding algorithm can be derived. Justify all aspects of the recurrence.

Hint: Use the recurrence that we developed in class for the standard rod cutting problem as a starting point and modify it to satisfy the constraints for this version of the problem.

This problem corresponds to Criterion 1: Design. Note that the rubric refers to "pseudocode" but what I'm looking for in your response to this question is the full, valid recurrence, with all aspects justified. The idea is that a full, valid recurrence would easily translate to an algorithm.

Put your answer here.

(c) Give an efficient implementation of your solution as a Java method, which I will test. Your code must compile and cannot crash under any circumstances. Your algorithm should not run in exponential time, but you will lose more points if it's not even correct. You must use the following starter code:

```
/**
Returns the maximum revenue given a rod of length len using
at most gold amount of coating.
@param prices the prices table
@param len the length of the rod
@param gold the number of inches worth of gold coating
*/
static int cut_rod_gold(int[] prices, int len, int gold) {
  // This method may call subsequent (private) helper methods.
  // Put your answer here.
}
```

Do not use any `Lists`, `Sets`, `Maps`, or any other data structure besides just primitive `int` arrays. If you violate this condition, then you will automati-

cally receive 0 points for this part. This problem corresponds to Criterion 2: Implementation.

(d) Finally, evaluate the performance of your algorithm from question 2c. Give the best possible upper bounds on the worst-case running time and space complexity of your algorithm in terms of O notation. Justify your answer using appropriate mathematical formalism that we've previously discussed. (e.g., Master theorem, summations, etc.). This problem corresponds to Criterion 3: Evaluation.

Put your answer here.

Rubric for Evaluation: Your solutions to problems 2b, 2c and 2d will be graded based on the following rubric.

	Exemplary	**Satisfactory**	**Marginal**	**Deficient**
Criterion 1: Design	The given design technique is correctly applied and the algorithm is correct in all test cases.	The given design technique is correctly applied and the algorithm is incorrect in at least one test case.	The given design technique is incorrectly applied.	The specification of the pseudocode for the algorithm is missing, incomplete or obviously incorrect for at least one instance of the problem.
Criterion 2: Implementation	Compiles, uses the correct method header, runs without crashing on all test cases and returns the optimal solution on all test cases.	Compiles, uses the correct method header, runs without crashing on all test cases but does not return the optimal solution on at least one test case.	Compiles, uses the correct method header, but crashes (at runtime) on at least one test case.	Does not compile, or does not use the given method header.
Criterion 3: Evaluation	The upper bounds given (for space and time) are tight and rigorously justified.	The upper bounds given (for space and time) are tight, but with insufficient justification.	Valid asymptotic upper bounds (for space and time) are given, but the bounds are not tight, or tight bounds are given with no justification.	Asymptotic upper bounds are invalid, missing for either time or space, or do not apply standard conventions for asymptotic notation, e.g., having additional lower-order terms or constants inside the asymptotic operator.