CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2021**

Homework #3

Deadline: Monday, March 22, 11:59 PM

Homework Policy

- If you leave a question completely blank, you will receive 25% of the grade for that question. This however does not apply to the extra credit questions.
- 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 (video recordings, notes, textbook, 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.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "use DFS (or BFS) to find all vertices reachable from a given vertex s in a graph G in $O(n+m)$ time". You are **strongly encouraged to use graph reductions** instead of designing an algorithm from scratch whenever possible (even when the question does not ask you to do so explicitly).
- Remember to always **prove the correctness** of your algorithms and **analyze their running time** (or any other efficiency measure asked in the question).
- The "Challenge yourself" and "Fun with algorithms" are both extra credit. These problems are significantly more challenging than the standard problems you see in this course (including lectures, homeworks, and exams). As a general rule, only attempt to solve these problems if you enjoy them.

Problem 1. Recall the job scheduling problem from the lectures: we have a collection of n processing jobs and the length of job i, i.e., the time to process job i, is given by $L[i]$. This time, you are given a number M and you are told that you should finish all your processing jobs between time 0 and M; any job not fully processed in this window then should be paid a penalty that is the same across all the jobs. The goal is to find a schedule of the jobs that minimizes the penalty you have to pay, i.e., it minimizes the number of jobs not fully processed in the given window.

Design a greedy algorithm that given the array $L[1:n]$ of job lengths and integer M, finds the scheduling that minimizes the penalty in $O(n \log n)$ time. **(25 points)**

Example: Suppose the length of the jobs are $[7, 3, 9, 2, 4]$ and $M = 7$. Then, one optimal solution is to run the jobs $[3, 4]$ in the window $[0:7]$ and then pay a penalty of 3 for the remaining jobs. Note that we could have alternatively picked the jobs $[3, 2]$ or $[2, 4]$ also but still had to pay a penalty of 3.

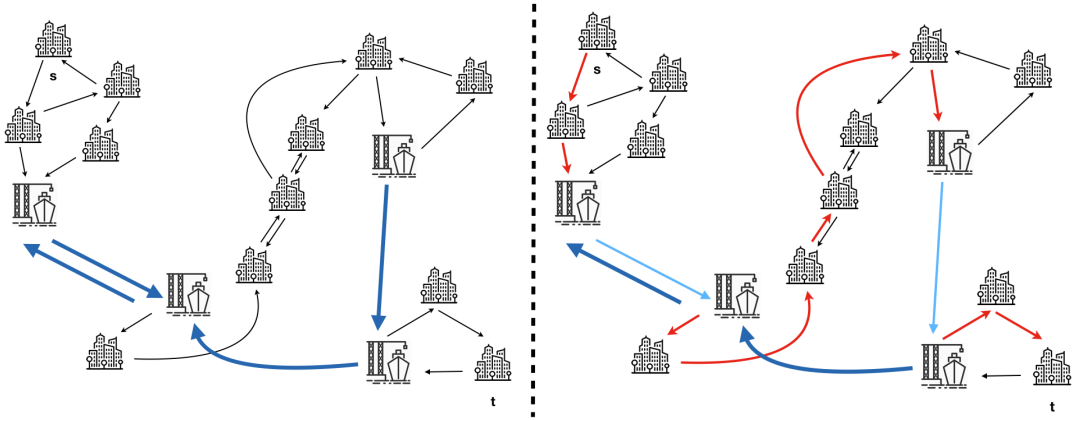
Simple bonus credit: Can you design an algorithm that instead runs in $O(n+M)$ time? **(+5 points)**

Problem 2. You are stuck in some city s in a far far away land and you know that your only way out is to reach another city t. This land consists of a collection of c *cities* and p *ports* (both s and t are cities). The cities in the land are connected by one-way *roads* to other cities and ports and you can travel these roads as many times as you like. In addition, there are one-way *shipping routes* between certain ports. However,

unlike the roads, you need a ticket to use these shipping routes and you only have 3 tickets; so effectively you can use at most 3 shipping routes in your journey.

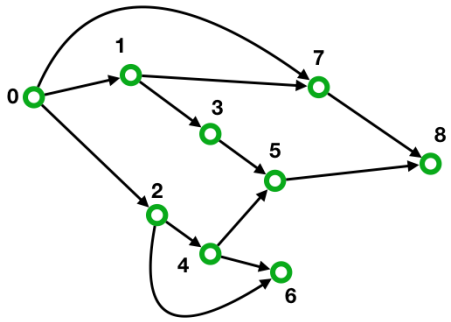
Assume the map of this land is given to you as a graph with $n = c + p$ vertices corresponding to the cities and ports and m directed edges showing one-way roads and shipping routes. Design an algorithm that in $O(n + m)$ time outputs whether or not it is possible for you to go from city s to city t in this land following the rules above, i.e., by using any number of roads but at most 3 shipping routes. **(25 points)**

Example: An example of an input map (on the left) and a possible solution (on the right) is the following:



Problem 3. We are given a directed acyclic graph (DAG) $G = (V, E)$ and two vertices s and t in this DAG. Design a dynamic programming algorithm that in $O(n + m)$ time, decides if the *number* of different paths from s to t is an *odd* number or an *even* one. You can assume that the number of paths from a vertex to itself (i.e., when $s = t$) is one and thus the correct answer in this case is *odd*. **(25 points)**

Example. Consider the following DAG:



- The answer for $s = 1$ and $t = 5$ is *odd* (there is just one path $1 \to 3 \to 5$).
- The answer for $s = 1$ and $t = 2$ is *even* (there is no path from 1 to 2).
- The answer for $s = 2$ and $t = 6$ is *even* (there are two paths: $2 \to 6$ and $2 \to 4 \to 6$).
- The answer for $s = 0$ and $t = 8$ is *even* (there are 4 paths in total).

Problem 4. You are given a 3D-matrix $A[1:n][1:n][1:n]$ with entries in $\{0,1\}$. You start from position $A[1][1][1]$ in this matrix and you can only move as follows:

- if you are at position $A[i][j][k] = 0$, then you can only move to either

$$A[i+1][j][k] \quad \text{or} \quad A[i][j+1][k];$$

- if you are at position $A[i][j][k] = 1$, then you can only move to either

$$A[i][j+1][k] \quad \text{or} \quad A[i][j][k+1];$$

In either of the cases, you cannot make a move that takes you outside the boundary of the matrix. The goal is to find a longest sequence of valid moves in this matrix starting from $A[1][1][1]$.

Design an $O(n^3)$ time algorithm that outputs the length of the longest sequence of moves. **(25 points)**

Challenge Yourself. A *bridge* in an undirected connected graph $G = (V, E)$ is any edge e such that removing e from G makes the graph disconnected. Design an $O(n + m)$ time algorithm for finding *all* bridges of a given graph with n vertices and m edges. **(+10 points)**

Fun with Algorithms. We are given an undirected connected graph $G = (V, E)$ and vertices s and t. Initially, there is a robot at position s and we want to move this robot to position t by moving it along the edges of the graph; at any time step, we can move the robot to one of the neighboring vertices and the robot will reach that vertex in the next time step.

However, we have a problem: at every time step, a subset of vertices of this graph undergo maintenance and if the robot is on one of these vertices at this time step, it will be destroyed (!). Luckily, we are given the schedule of the maintenance for the next T time steps in an array $M[1:T]$, where each $M[i]$ is a linked-list of the vertices that undergo maintenance at time step i.

Design an algorithm that finds a route for the robot to go from s to t in at most T seconds so that at no time i, the robot is on one of the maintained vertices, or output that this is not possible. The runtime of your algorithm should ideally be $O((n + m) \cdot T)$ but you will receive partial credit for worse runtime also.

(+10 points)