CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2022**

Homework #3

Deadline: Thursday, March 24, 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. You are given a set of n (closed) intervals on a line:

$$[a_1, b_1], [a_2, b_2], ..., [a_n, b_n].$$

Design an $O(n \log n)$ time greedy algorithm to select the *minimum* number of points on the line between $[\min_i a_i, \max_j b_j]$ such that any input interval contains at least one of the chosen points.

Example: If the following 5 intervals are given to you:

$$[2, 5], [3, 9], [2.5, 9.5], [4, 8], [7, 9],$$

then a correct answer is: $\{5, 9\}$ (the first four intervals contain number 5 and the last contains number 9; we also definitely need two points since $[2, 5]$ and $[7, 9]$ are disjoint and no single point can take care of both of them at the same time).

Problem 2. You are given two arrays of positive integers $A[1:n]$ and $B[1:n]$. The goal is to re-order the arrays A and B so that the average difference between entries of A and B with the same index, after the re-ordering, is minimized. In other words, we want to change the orders of numbers in A and B, so that

$$\frac{1}{n} \cdot \sum_{i=1}^{n} |A[i] - B[i]|$$

is minimized.

Design an $O(n \log n)$ time greedy algorithm for this problem.

Example: If $A = [3, 2, 5, 1]$ and $B = [5, 7, 2, 9]$, we can re-order A to $[1, 2, 3, 5]$ and B to $[2, 5, 7, 9]$ to achieve

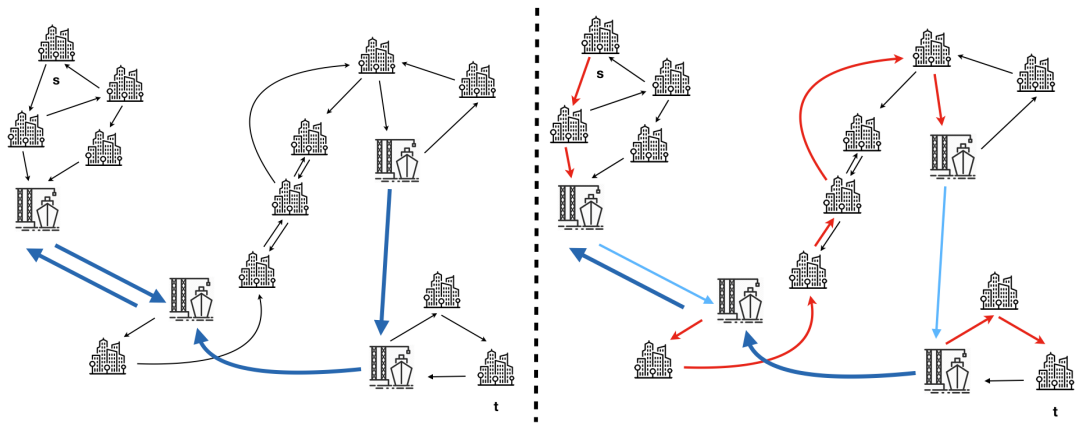
$$\frac{1}{4} \cdot \sum_{i=1}^{4} |A[i] - B[i]| = \frac{1}{4} \cdot (1 + 3 + 4 + 4) = 3,$$

which you can also check is the minimum value.

Problem 3. 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 10 tickets; so effectively you can use at most 10 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 10 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 4. This question is from your textbook (Question 11 in Chapter 05).

A number maze is a $k \times k$ grid of positive integers. A token starts in the upper left corner and your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For

example, if the token is on a square labeled 3, then you may move the token *exactly* three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board.

Design and analyze an algorithm that in $O(k^2)$ time determines if there is a way to move the token in the given number maze or not. **(25 points)**

Example: Given the number maze in the following figure, your algorithm should return *Yes*.

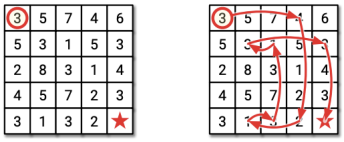


Figure 1: A 5×5 maze with the answer *Yes*.

Hint: Think of graph reductions – this problem is not that different from the fill coloring problem after all.

Challenge Yourself. Consider Problem 4 again, except this time, you are additionally allowed to change the value of *up to* 10 different cells in the maze, each by plus one or minus. You can decide how many cells from 0 to 10 you want to change, and which cells these are, as well as for each chosen cell whether you like to increase the value of the number inside by one, or decrease it by one. The goal is to see if there is a way of making a change as described above so that the maze can be solved or not. The output is *Yes*, along with the set of cells to change their value (and whether you increase or decrease them), if one can make proper changes to solve the maze, and *No* otherwise. **(+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)