CS-GY 6033: Design and Analysis of Algorithms I

Fall 2024

Instructor: Nairen Cao **NYU Tandon School of Engineering**

Problems	1	2	3	4	5 (ungraded)	Total
Max. Score	25	20	25	30	0	100

Table 1: Score Summary

Homework #6

Assigned: 11/26/2024 **Due: 12/16/2024**

General Instructions Write all solutions clearly, concisely, and legibly. It is okay to provide handwritten and scanned/photographed solutions as long as they are clearly legible. If typing, using LaTeX is recommended.

You should generally describe algorithms in words and high-level pseudocode. Specifically, emphasize the goal of each step in your algorithm rather than the coding details. After giving a high-level description, provide more detail on the implementation of each step where appropriate.

The purpose of having these problem sets is to enhance your mathematics and programming skills in the scope of designing algorithms. These problems may be quite challenging, so discussions with classmates are encouraged. However, you should spend at least half an hour thinking about the problems individually before discussing them. Although discussions (and consulting ChatGPT and WolframAlpha) are allowed, you need to write the solution in your own words and by yourself. Please acknowledge any person or reference you discuss with or consult from.

Special Instructions

- **Theme for Homework:** Shortest Paths, Max Flow.

Problem 6-1. Ups and Downs

Suppose we are given as input a directed graph $G = (V, E)$ with distinct real edge weights $w(u, v) \in \mathbb{R}$ (both positive and negative). We wish to find the shortest paths from a single source vertex $s \in V$.

Without any additional information, we would probably run Bellman-Ford. Suppose, however, that we given one additional restriction: the edge weights along any shortest path starting from s first strictly increase, then strictly decrease, then strictly increase. For example, the sequence of edge weights on a shortest path may be $\langle 3, 7, 15, 9, -3, 2, 8\rangle$, but they could not be $\langle 3, 7, 4, 8, 1\rangle$. (There may be other paths in the graph that do not obey this increasing-decreasing-increasing property. The restriction applies only to shortest paths.)

(a) Provide the best single-source shortest paths algorithm you can, assuming the graph observes the increasing-decreasing-increasing restriction on shortest paths.

Note that your algorithm should be much faster than Bellman-Ford, or it will not receive any credit. Hint: first think about how you would solve the problem if edge weights only

increase along shortest paths, then extend your algorithm to handle the actual problem statement. You will want to use sorting as a subroutine, so you should assume that you have an algorithm that sorts n elements in $O(n \log n)$ time.

(b) Analyze the running time of your algorithm.

(c) Explain how to test whether the shortest paths from s do in fact obey the increasing-decreasing-increasing property. How efficiently can you do that?

Problem 6-2. Edge-Disjoint Paths Problem

In graph theory, the Edge-Disjoint Paths Problem is to determine the maximum number of edge-disjoint paths between two given nodes in a graph. Two paths are considered edge-disjoint if they do not share any edges.

More formally, given a directed graph $G = (V, E)$, and two nodes s (source) and t (sink), find the maximum number of edge-disjoint paths from s to t.

Example of the Edge-Disjoint Paths Problem

Consider the following directed graph $G = (V, E)$ with source s and sink t. We aim to find the maximum number of edge-disjoint paths from s to t.

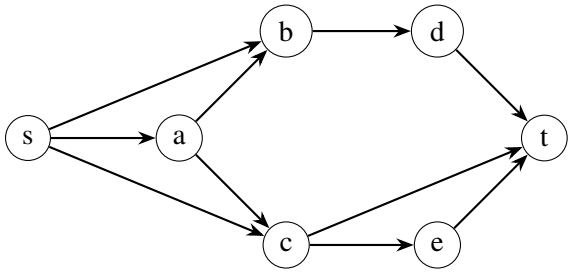

Figure 1: A directed graph with source s and sink t.

Result

In this example, there are 3 edge-disjoint paths:

1. $s \to b \to d \to t$
2. $s \to a \to c \to t$
3. $s \to c \to e \to t$

(a) Design an algorithm that computes the maximum edge-disjoint paths, you need to return the corresponding disjoint paths.

Problem 6-3. Scheduling classes

The registrar's office is having some difficulty optimizing class schedules for students. They've heard that you are a master at optimization problems, so they ask for your help.

The input to the problem is the following:

- An integer n specifying the number of students. For convenience, assume the students are identified by an id $1, 2, 3, \ldots, n$.
- An integer m specifying the number of courses. For convenience, assume the courses are identified by a course number $1, 2, \ldots, m$.

- For each student i, you are provided with a list S_i of 8 courses that student i wishes to take. For example, you might have $S_1 = [2, 4, 5, 7, 9, 10, 11, 13]$, which means that Student 1 is willing to take Course 2, Course 4, Course 5, Course 7, Course 9, Course 10, Course 11, or Course 13.
- For each course j, you are provided with a number C_j indicating the maximum number of students that can enroll in the course.

You'd like to assign students to classes to maximize the total enrollment across all classes. But there are a few constraints:

- each student can only be enrolled in a subset of his or her 8 selections[1],
- each student can only be enrolled in at most 5 courses total,
- and no course enrollment exceeds its capacity.

(a) Give an algorithm that uses max-flow as a black box to find an optimum solution for this problem. Your algorithm should output which students take which courses, not just determine the number of seats filled.

Note the problem is not stated as a graph problem. You have to explicitly describe (1) how to construct the corresponding flow network, and (2) how to transform a max-flow solution to a solution to the problem at hand.

(b) Analyze the worst-case running time of your algorithm, with respect to n and m, when using Ford-Fulkerson as the underlying max-flow algorithm.

Problem 6-4. Minimum Path Cover

Given a directed acyclic graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, the goal of the problem is to find the minimum number of robots needed to serve all vertices of the graph. Each robot can start and end at any vertex in G and can move only along the edges of the graph. Once a robot visits a vertex v, it can serve v. We allow multiple robots to visit the same vertex v.

For example, consider the example in Figure 2. We need 3 robots to serve all the vertices.

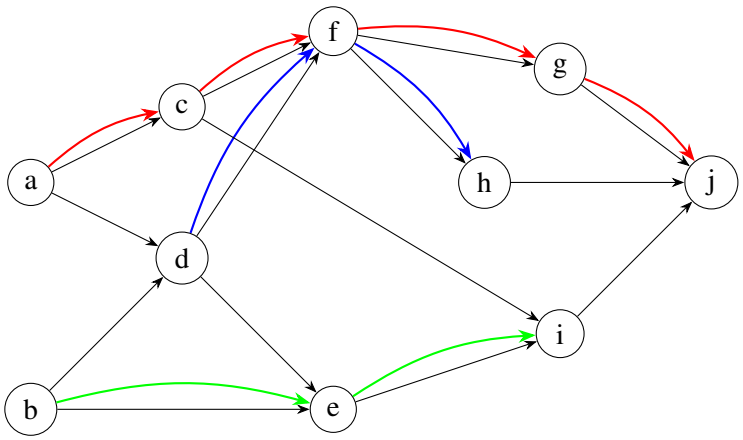

Figure 2: Using 3 robots to serve all the vertices in the graph. The paths for the three robots are $a \to c \to f \to g \to j$, $d \to f \to h$, and $b \to e \to i$.

(a) Now assume that we require each vertex can only be covered by one robot. Can you provide an algorithm using maximum flow? Describe how to construct the flow graph. You don't have to prove the correctness of your algorithm.

[1]You may assume that there are no time conflicts among those courses, so any subset of the 8 courses is feasible for any individual student.

(b) Solve the case where every vertex can be covered by multiple robots using the algorithm from the previous problem.

Problem 6-5. (Ungraded) Collecting Stars

As a CS major, you are certainly very good at Theoretically Collecting Stars (abbreviated as TCS). You are given a **directed**, weighted graph $G = (V, E, w)$. Additionally, you are given a subset of directed edges $S \subseteq E$ where for each edge $(u, v) \in S$ there is a star for you to collect. The function $w : E \to \mathbb{Z}_{>0}$ indicates the number of minutes required to travel along an edge. I.e., traveling through an edge (u, v) takes $w(u, v)$ minutes.

(a) Provide an algorithm to compute, for each pair $(u, v) \in V^2$, the shortest possible time (in minutes) to travel from u to v while collecting at least 2 stars. Note that traversing an edge in S more than once still yields only one star.

(b) Analyze the running time of your algorithm.