CS 170 HW 8

Due **2021-10-25, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 (Max-Flow) A cohort of spies

A cohort of k spies resident in a certain country needs escape routes in case of an emergency. They will be travelling using the railway system which we can think of as a directed graph $G = (V, E)$ with V being the cities. Each spy i has a starting point $s_i \in V$, all s_i's are distinct. Every spy needs to reach the consulate of a friendly nation; these consulates are in a known set of cities $T \subseteq V$. In order to move undetected, the spies agree that at most c of them should ever pass through any one city. Our goal is to find a set of paths, one for each of the spies (or detect that the requirements cannot be met).

Model this problem as a flow network. Specify the vertices, edges and capacities, and show that a maximum flow in your network can be transformed into an optimal solution for the original problem. You do not need to explain how to solve the max-flow instance itself.

3 (Max-Flow LP) Min Cost Flow

In the max flow problem, we just wanted to see how much flow we could send between a source and a sink. But in general, we would like to model the fact that shipping flow takes money. More precisely, we are given a directed graph G with source s, sink t, costs l_e, capacities c_e, and a flow value F. We want to find a nonnegative flow f with minimum cost, that is $\sum_e l_e f_e$, that respects the capacities and ships F units of flow from s to t.

(a) Show that the minimum cost flow problem can be solved in polynomial time.

(b) Show that the shortest path problem can be solved using the minimum cost flow problem

(c) Show that the maximum flow problem can be solved using the minimum cost flow problem.

4 Faster Maximum Flow

In the class, we see that the Ford-Fulkerson algorithm computes the maximum flow in $O(mF)$ time, where F is the max flow value. The run-time can be very high for large F. In this question, we explore a polynomial-time algorithm whose run-time does not the depend on

the flow value. Recall that Ford-Fulkerson provides a general recipe of designing max flow algorithm, based on the idea of an *augmenting path* in a residual graph:

Algorithm 1 Ford-Fulkerson

while there exists an augmenting path in G_f **do**
  Find an arbitrary augmenting path P from s to t
  Augment flow f along P
  Update G_f

This problem asks you to consider a specific implementation of the algorithm above, where each iteration we find the augmenting path with the smallest number of edges and augment along it.

Algorithm 2 Fast Max Flow

while there exists an augmenting path in G_f **do**
  Find the augmenting path P from s to t with the smallest number of edges
  Augment flow f along P
  Update G_f

We first show that each iteration can be implemented efficiently. Then we analyze the number of iterations required to terminate. Throughout, we define the shortest path from u to v as the path from u to v with the smallest number of edges (instead of the smallest sum of edge capacities). Consequently, we define distance $d(u, v)$ from u to v as the number of edges in the shortest path from u to v.

(a) Show that given G_f, the augmenting path P from s to t with the smallest number of edges can be found in $O(m + n)$ time.
Hint: Use the most basic graph algorithm you know.

(b) Show that the distance from s to v in G_f never decreases throughout the algorithm, for any v (including t).
Hint: Consider what augmentation does to the "forward edges" going from u to u', where $d(s, u') = d(s, u) + 1$.

(c) Show that with every m flow augmentations, the distance from s to t must increase by (at least) 1.
Hint: Observe that each flow augmentation removes at least one forward edge.

(d) Conclude by proving that the total number of flow augmentations this algorithm performs is at most $O(mn)$. Analyze the total run-time of the algorithm.
Hint: You may observe that the distance from s to t can increase at most $O(n)$ times throughout the algorithm. If you use this fact, explain why it holds.

5 Jumbled Sequences

You are measuring temperatures in a few different places, and receiving temperatures remotely. You have received a sequence of n temperatures, from a total of k thermometers;

unfortunately, they are all jumbled together, and you don't know which temperatures are from which thermometer.

Thankfully, you don't really care, and so are content to simply extract k disjoint 'plausible' subsequences from your original sequence of measurements (disjoint meaning each element in the original sequence belongs to at most 1 subsequence). A subsequence is 'plausible' if it satisfies 3 conditions:

1. it consists of elements from the original sequence in their original order
2. any two consecutive elements in the subsequence have no more than $2k$ elements between them
3. the first element in the subsequence has no more than $2k$ elements before it in the original sequence, and the last element has no more than $2k$ elements after it
4. any two consecutive elements in the subsequence differ in value by no more than 2

As an example, if you receive the sequence $(10, 8, 3, 9, 4, 9, 5, 6)$, you can extract 2 disjoint plausible subsequences from it: $(10, 8, 9, 9)$ and $(3, 4, 5, 6)$. Note that $(10, 8, 6)$ is not a plausible subsequence, because there are 5 elements between 8 and 6 in the original sequence.

Give an algorithm which, given a *specific* k, determines in $O(nk^2)$ time whether it is possible to find k disjoint plausible subsequences.

Describe your algorithm and give a runtime analysis; no proof of correctness is necessary.