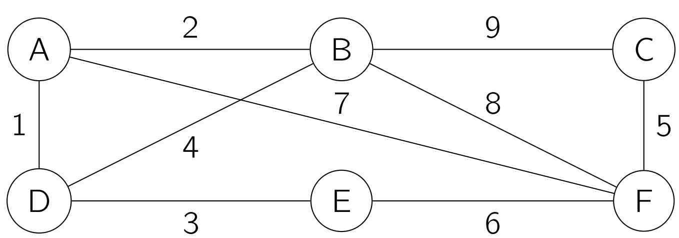
CS 161 (Stanford, Winter 2023) Homework 8

Style guide and expectations: Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material.
What we expect: Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Exercises. The following questions are exercises. We suggest you do these on your own. As with any homework question, though, you may ask the course staff for help.

1 Spanning Tree Algorithms

Consider the graph G below.



1.1 Prim (1 pt.)

In what order does Prim's algorithm add edges to an MST when started from vertex C?

[We are expecting: An ordered list of edges.**]**

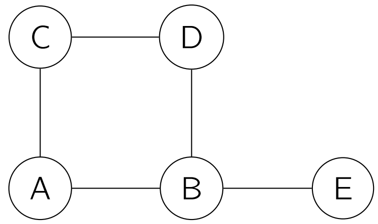
1.2 Kruskal (1 pt.)

In what order does Kruskal's algorithm add edges to an MST?

[We are expecting: An ordered list of edges.**]**

2 Min Cuts and Max Flows

Consider the following graph (assume all edges have weight 1):



2.1 Min Cut (1 pt.)

Recall that the weight of a cut is the sum of the weights of the edges which cross the cut. (If there is a cut between vertex sets X and Y, the weight of the cut is the sum of the weight of the edges which have one endpoint in X and the other in Y).

How can we partition all of the edges of the graph above into two non-empty sets X and Y so that the cut between the two is minimized? (This is known as a global min-cut).

[We are expecting: Two sets of vertices X and Y, along with the weight of the cut between them. No explanation is required.**]**

2.2 Max Flow (1 pt.)

What is the maximum flow from C to B? From C to E?

[We are expecting: Just the maximum flows. No justification is required.**]**

Problems. The following questions are problems. You may talk with your fellow CS 161-ers about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

3 Plucky's Subway Adventure

Plucky is planning to visit her very large family this weekend. She realizes that she needs to visit every single subway station to visit everyone from her family. She obtained a subway map where each station is represented as a vertex and she sees that there are subway lines connecting all the stations to form an undirected graph $G = (V, E)$.

The subway system in her town has a peculiar pricing system. Each edge in the subway graph has a weight that represents how expensive it is to travel between the two nodes it connects.

Plucky plans to buy a special student ticket marked for x dollars that allows her to travel for **unlimited trips** between any two stations that takes no more than x dollars to travel. In other words, she can travel through any path P in the subway system, as long as $\max\{w_e \mid e \in P\} \leq x$.

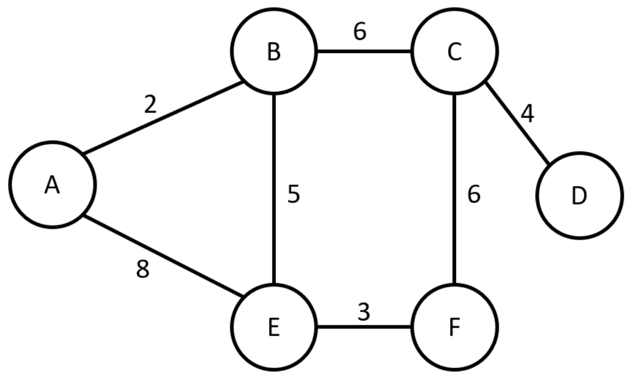


Figure 1: In a graph like this, Plucky needs to buy a \$6 ticket to travel to all the stations; she will be able to travel freely through any edges except for $\{A, E\}$ with her ticket.

Plucky wants to get the cheapest ticket while visiting all the stations. Plucky realizes that she will be able to do so by finding a spanning tree T of G that minimizes the quantity

$$x = \max_{e \in T} w_e,$$

Let us call this spanning tree a minimum-maximum tree since it minimizes the largest edge in the tree.

3.1 MST (6 pt.)

Prove that a minimum spanning tree in G is always a minimum-maximum tree. We will provide two hints, which suggest two *separate* ways to prove this statement. *DO NOT* try to use both hints within the same proof.

Hint 1: Suppose toward a contradiction that T is an MST but not a minimum-maximum tree, and say that T' is a minimum-maximum tree. Try to come up with a cheaper MST than T (and hence a contradiction).

Hint 2: Use (without proof) the fact that any MST can be created by Kruskal's algorithm.

[We are expecting: A rigorous proof.**]**

3.2 The other way around (3 pt.)

Show that the converse to the last part is not true. That is, minimum-maximum tree is not necessarily a minimum spanning tree.

[We are expecting: A counter-example, with an explanation of why it is a counter-example. **]**

4 Max-Flow

Let $G = (V, E)$ be a flow network with source $s \in V$, sink $t \in V$, and edge capacities for each edge $e \in E$. All edge capacities are positive *integers.* We can represent a flow by a 1-indexed array F, where $F[i]$ is the flow through edge $E[i]$ for $1 \leq i \leq |E|$.

4.1 Flow verification (5 pt.)

Given G and F, design an $O(|V|+|E|)$-time algorithm to determine if the flow F is a maximum flow in G.

[We are expecting: An English description of your algorithm, an informal explanation of why it works, and a runtime analysis.**]**

[Hint: remember to check that F is a valid flow.]

4.2 Flow update I (5 pt.)

Suppose that the capacity of a single edge $e = (u, v) \in E$ is increased by 1. Given G, its maximum flow F before the update, and e, design an $O(|V| + |E|)$-time algorithm to update F so that it is still the maximum flow of G after the update to e.

[We are expecting: An English description of your algorithm, an informal explanation of why it works, and a runtime analysis.**]**

4.3 Flow update II (5 pt.)

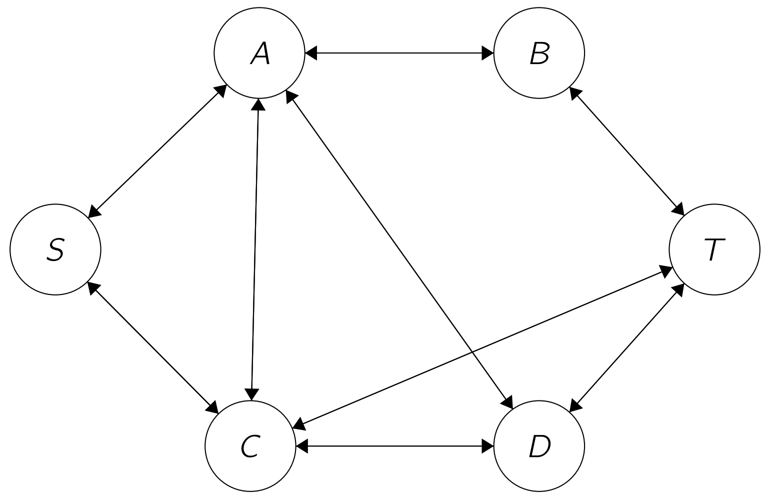
Suppose that the capacity of a single edge $e = (u, v) \in E$ is decreased by 1. Given G, its maximum flow F before the update, and e, design an $O(|V| + |E|)$-time algorithm to update F so that it is still the maximum flow of G after the update to e.

[We are expecting: An English description of your algorithm, an informal explanation of why it works, and a runtime analysis.**]**

5 Truculent Terrapins

Toby the Terrapin has two children who need to get to tap dancing class, but they often quarrel, and so have trouble traveling together. The possible routes that these two terrapins can take to class are represented via an undirected, unweighted graph G. Both of Toby's children start at node s, and they need to finish at node t without having used any of the same edges on their path.

An example graph:



The paths $[S, A, B, T]$ and $[S, C, D, T]$ are valid for both 5.1 and 5.2.

The paths $[S, A, C, T]$ and $[S, C, D, T]$ are valid for 5.1 but not 5.2.

The paths $[S, A, D, T]$ and $[S, C, D, T]$ are valid for neither 5.1 nor 5.2.

5.1 Find Two Paths, No Overlapping Edges (5 pt.)

Help Toby design an algorithm to find paths for each of his two children. Your algorithm should modify the graph and call Ford-Fulkerson as a subroutine. Your algorithm should either return the list of vertices visited by the two separate paths, or -1 if no two paths which meet the requirements exist.

[We are expecting: How you will modify the graph, how you will use Ford-Fulkerson, and a justification as to why your algorithm always finds a path for each of Toby's two children if one exists.**]**

5.2 Finding Two Paths, No Overlapping Nodes (5 pt.)

Toby finds that his children are still quarreling, because even if they don't use the same edges, if they ever end up at the same node they will get into an argument.

Help Toby design a new algorithm that finds two paths from s to t for his children that do not share any nodes (except for s and t). Your algorithm should first modify the graph so that each node can only be passed through once, and then call Ford-Fulkerson as a subroutine. Your algorithm should either return the list of vertices visited by the two separate paths, or -1 if no two paths which meet the requirements exist.

[We are expecting: How you will modify the graph, how you will use Ford-Fulkerson, and a justification as to why your algorithm always finds a path for each of Toby's two children if one exists.**]**

5.3 Extending the Algorithm (2 pt.)

Lucky the Lemur has 17 children, and he wants to put his children on separate paths to tap dancing as well. (He wants to use Toby's more strict requirement that none of his children ever visit the same node). Extend the algorithm you wrote in part 5.2 to find n separate paths from s to t such that no two of them visit the same node (except for s and t). (Again, the algorithm should return -1 if it is impossible to create n such paths).

[We are expecting: How you will modify your previous approach, and a justification as to why your algorithm always finds a path for each of Toby's two children if one exists.**]**