CS 161 (Stanford, Winter 2022) 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 Fear of Negativity

Do our graph algorithms work when the weights are negative? Let's answer that in this problem. Assume that the graph is directed and that all edge weights are integers.

1.1 Negative Cycles (1 pt.)

A "negative cycle" is a cycle where the sum of the edge weights is negative. Why can't we compute shortest paths in a graph with negative cycles? **[We are expecting:** An informal explanation.**]**

1.2 Negative Dijkstra (2 pt.)

Even if a graph contains no negative cycles (but still contains negative edges) we might still be in trouble. Please draw a graph G, which contains both positive and negative edges but does not contain negative cycles, and specify some source $s \in V$ where `Dijkstra(G, s)` does not correctly compute the shortest paths from s. **[We are expecting:** A graph G with no more than 4 vertices (including a source node s), and an example of a shortest path from s that is not correctly computed using Dijkstra's algorithm.**]**

1.3 A Fix for Negative Dijkstra? (4 pt.)

Consider the algorithm `Negative-Dijkstra` for computing shortest paths through graphs with negative edge weights (but without negative cycles). This algorithm adds some number to all of the edge weights to make them all non-negative, then runs `Dijkstra` on the resulting graph, and argues that the shortest paths in the new graph are the same as the shortest paths in the old graph.

Negative–Dijkstra(G, s):

```
minWeight = minimum edge weight in G
for e in E: # iterate through all edges in G
    modifiedWeight(e) = w(e) − minWeight
modifiedG = G with weights modifiedWeight
T = Dijkstra(modifiedG, s) # run Dijkstra with modifiedWeight to get a SSSP Tree
update T to use weights w that corresponds to graph G
return T
```

(Note that an "SSSP tree", or a "single-source-shortest-path tree", is analogous to a breadth-first-search tree in that paths in the SSSP tree correspond to shortest paths in the graph. Here we assume that Dijkstra's algorithm has been modified to output such a tree.) **Prove or disprove:** Negative-Dijkstra *always* computes single-source shortest paths correctly in graphs with negative edge weights. **[We are expecting:** To prove the algorithm correct, show that for all $u \in V$, a shortest path from s to u in the original graph lies in T. To disprove the algorithm, exhibit a graph with negative edges, but no negative cycles, where Negative-Dijkstra outputs the wrong "shortest" paths, and explain why the algorithm fails.**]**

1.4 Negative Prim? (4 pt.)

Since Prim's algorithm is very similar to Dijkstra, we want to now consider a similar algorithm `Negative-Prim` for computing minimum spanning tree in graphs with negative edge weights. Again, this algorithm adds some number to all of the edge weights to make them all non-negative, then runs `Prim's algorithm` on the resulting graph, and argues that the Minimum Spanning Tree in the new graph are the same as the MST in the old graph. You can assume that all the edge weights are unique integers.

```
Negative–Prim(G, s):
    minWeight = minimum edge weight in G
    for e in E: # iterate through all edges in G
        modifiedWeight(e) = w(e) − minWeight
    modifiedG = G with weights modifiedWeight
    T = Prim(modifiedG, s) # run Prim's algorithm starting from s
    update T with edges that corresponds to graph G
    return T
```

[We are expecting: Either an informal explanation of why `Negative-Prim` computes the correct MST, or a counter-example of an undirected graph with negative edge weights where `Negative-Prim` does not output the correct minimum spanning tree, as well as an explanation of why it is a valid counter-example.**]**

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.

2 Max-Cut

In this question we'll try to come up with algorithms for the **Max-Cut** problem, which is just like Min-Cut but with the opposite objective: we're given an *undirected, unweighted* graph $G = (V, E)$, and our goal is to find a partition of the vertices into subsets $S, V \setminus S$ that maximizes the *number of edges* going from S to $V \setminus S$. We've given two possible algorithms for the Max-Cut problem in 2.1 and 2.2. For each one, give a short explanation of why it does or doesn't find the max-cut.

2.1 Modified Ford-Fulkerson (3 pt.)

Assume that all edges have weight 1. Enumerate over all candidate pairs of (s, t). For each pair find the *minimum* s-t flow, using the idea that a MinFlow corresponds to a MaxCut (consider the MinCut = MaxFlow theorem we saw in class, just reversing min and max). **[We are expecting:** If the algorithm is correct, an informal explanation. If the algorithm is incorrect, an counter-example and an explanation of why it is a counter-example.**]**

2.2 Modified BFS (3 pt.)

Initialize two empty sets S_1 and S_2. Run BFS on the graph starting at a random node, adding the start node to S_1. Then at each step of BFS, add the current node to the opposite set as its parent (the node it was discovered from). Terminate once all nodes have been discovered, and return $\{S_1, S_2\}$ as the cut. **[We are expecting:** If the algorithm is correct, an informal explanation. If the algorithm is incorrect, an counter-example and an explanation of why it is a counter-example.**]**

2.3 Greedy Algorithm (6 pt.)

Next you will design a **greedy** algorithm that runs in time $O(m+n)$ on G, an undirected and unweighted graph, and returns a cut of size at least $1/2$ times the maximum cut.
[We are expecting: An English description of your algorithm, an informal justification of correctness, and a runtime analysis.**]** *[Hint: You can always return a cut with at least* $1/2$ *of all the edges in the graph.]*

3 Plucky's Subway Adventure

Plucky is planning to visit her very large family this weekend. She realizes that she need to visit every single subway stations to visit everyone from her family. She obtained a subway map where each station are represented as a vertex and she sees that there are subway lines connecting all the stations to form a 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 is planning 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. That is, if she can travel through any path P in the subway system, where $\max(\{w_e | e \in P\}) \leq x$.

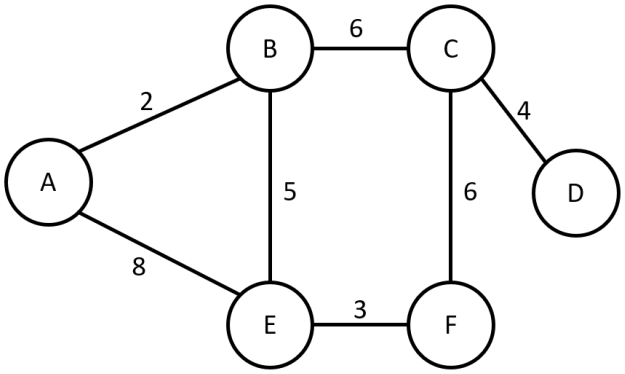

Figure 1: In a graph like this, Plucky need to buy a \$6 ticket to travel to all the stations, she will be able to travel freely through any edges except for $\{A, E\}$ with her ticket.

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 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. [Hint: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).] **[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 (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 2 (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.**]**

4.4 Node capacities (5 pt.)

A directed graph $G' = (V', E')$ has integer node capacities $n_c(v)$ for each node $v \in V'$, in addition to integer edge capacity for every edge $e \in E'$. The flow through a node v cannot be more than v's node capacity. In other words, the sum of all the flow into v will be at most $n_c(v)$, and the sum of all the flow out of v will be at most $n_c(v)$. Let the maximum node capacity be M_n and the maximum edge capacity be M_e. Design an $O((|V'| + |E'|) \cdot \max\{M_n, M_e\})$-time algorithm that finds the value of the maximum flow

from $s \in V'$ to $t \in V'$. **[We are expecting:** An English description of your algorithm, an informal explanation of why it works, and a runtime analysis.**]** *[Hint: Create a new graph G'' that only has edge capacities and does not have node capacities, like the flow networks we saw in lecture. Then run an algorithm you learned in lecture to get the maximum flow.]*