CS 3510 Section C: Design & Analysis of Algorithms September 19th 2023

HW 4: Dijkstra's, MST, Max Flow Min Cut

YOUR NAME HERE Due: September 24th 2023

- Please type your solutions using LaTeX or any other software. Handwritten solutions will not be accepted.
- Your algorithms must be in plain English & mathematical expressions, and the pseudo-code is optional. Pseudo-code, without sufficient explanation, will receive no credit.
- Unless otherwise stated, all logarithms are to base two.
- If we ask for a specific running time, a correct solution achieving it will receive full credit even if a faster solution exists.

1.) You are given a graph (V, E) and each edge in the graph is labeled either with a monkey or a banana. Since you like bananas, design an algorithm that finds the spanning tree that maximizes the number of banana edges.

Solution:

Including many banana edges as possible means including as few monkey edges as possible. We can mark monkey edges. We can have the banana edges incur a weight of 0 and the monkey edges incur a weight of 1. The MST of this graph will then be the spanning tree that includes as many banana edges as possible.

2.) A *feedback edge set* of an undirected graph $G = (V, E)$ is a subset $F \subseteq E$ such that *every* cycle in G contains at least one edge in F. Design an algorithm that takes as input a **connected** undirected weighted graph $G = (V, E)$ with all weights being positive and distinct, and returns a *minimum-weight* feedback edge set.

Justify correctness and running time, including explaining why your output is a feedback edge set (i.e. why every cycle contains edges on it), and why it is of minimum weight.

Hint: Think Kruskal's algorithm.

Solution: If we remove a set of edges belonging to the Maximum Spanning Tree in the graph, we will get all the edges in the minimum-weight feedback edge set.

1. Create a new graph G' by keeping the same vertices but multiplying the edge weights in E by -1.
2. Run Kruskal's algorithm on this new graph G'. The edges returned will be in the Maximum Spanning tree since the edges are negated.
3. Remove these edges from the original edge set E.

Proof of Correctness: Since we are running Kruskal's algorithm on the graph with negative edges, the lowest edge weights will be selected which are eventually a part of the maximum spanning tree. Removing these edges from E will give the minimum weight feedback edge set and would satisfy the given definition.

Time Complexity: The running time is $\mathcal{O}(|E| \log(|V|))$, as we loop through the edges to change the sign, which is linear in $\mathcal{O}(|E|)$, and then we run Kruskal's algorithm.

3.) Give an algorithm that finds shortest paths from a given vertex in a directed graph $G = (V, E)$ that contains exactly one negative weighted edge $e^* = (u, v)$. Assume that the graph does not contain any negative weight cycles. Your running time should be the same time complexity as Dijkstra's algorithm.

Solution: Since Dijkstra is a greedy algorithm and assumes that the lengths of the paths are always strictly positive, we cannot simply use Dijkstra with one negative weight. However, we can use the fact that shortest paths between a given vertex and all other vertices in the graph either utilize the negative weighted edge or they do not.

The only situation in which a path will utilize the negative weighted edge is if the path goes through u to v. Assuming the start vertex is k and $S_k[w]$ represents the shortest path to vertex w from k, we can express the shortest path for any vertex m:

$$\texttt{min}(S_k[m], S_k[u] + e^* + S_v[m])$$

This represents the only two options we have to reach vertex m, either directly from k, or through u and v, in which case we add the minimum distance from k to u, the negative weight edge, and the minimum distance from v to m. Therefore the algorithm will be as follows:

1. Run Dijkstra's algorithm from the given vertex k. This will output S_k, the set of shortest paths from k to every vertex which might or might not account for the negative weight edge.
2. Run Dijkstra's algorithm again from vertex v. This will output S_v.
3. Calculate the shortest path for each vertex m from k by calculating $\texttt{min}(S_k[m], S_k[u] + e^* + S_v[m])$.

The time complexity is $\mathcal{O}((|V|+|E|)\log|E|)+\mathcal{O}((|V|+|E|)\log|E|)+\mathcal{O}(|V|) = \mathcal{O}((|V|+|E|)\log|E|)$.

4.) You are given a flow network $G = (V, E)$ with source s and sink t. The maximum flow between s and t and the residual network have already been computed for you as f and G_f respectively. Suppose we generate a new graph G' by selecting some edge $e \in E$ and incrementing its capacity $c(e) = c(e) + 1$. Given G, f, G_f, e, *without recomputing the max flow of* G', design an algorithm that determines if f is a max flow of G'. Your algorithm should return a boolean and does not need to compute the max flow of G'.

Solution: Modify the residual network G_f by adding the weight of the edge to the forward edge and subtracting the backward residual edge weight. Then, run `Explore` from s to t. If there exists a path from s to t, then the minimum cut is no longer a cut, so the max flow will change - return False. If s is still disconnected from t, the max flow remains the same. Return True.