CS 670 (Spring 2025) — Assignment 1

Due: 02/10/2025

In all homeworks, you can use all facts from class and from the textbook without reproving them.

Note: the solutions to many homework problems can probably be found fairly easily online or in textbooks, not to mention being answered by Chatbots and other AI tools. Recall that using these resources to solve homework problems is considered cheating, and will result in corresponding sanctions.

(1) [15 points]
This problem deals with the concept of matroids. Recall that a *matroid* is a set system $(E, \mathcal{I})$, where $\mathcal{I} \subseteq 2^E$, with the following properties:

- $\emptyset \in \mathcal{I}$.
- If $S \in \mathcal{I}, S' \subseteq S$, then $S' \in \mathcal{I}$.
- If $S, S' \in \mathcal{I}, |S| < |S'|$, then there exists an $e \in S' \setminus S$ such that $S \cup \{e\} \in \mathcal{I}$.

(a) Prove that for any given undirected graph G, the following is a matroid: E is the set of all edges, and $\mathcal{I}$ is the set of all acyclic edge sets of G.

(b) Prove that the following is a matroid: You have k disjoint sets $S_1, S_2, \ldots, S_k$, and $E = \bigcup_i S_i$. Furthermore, you are given k integers $n_1, n_2, \ldots, n_k$, and one more integer n. $\mathcal{I}$ consists of all subsets $S \subseteq E$ such that $|S \cap S_i| \leq n_i$ for all i, and $|S| \leq n$.

(c) Now, give each element e of the matroid a weight $w_e \geq 0$. (You can assume that all weights are distinct.) The goal is to find a set $S \in \mathcal{I}$ maximizing $\sum_{e \in S} w_e$. (Generalization of a maximum spanning tree.)
Consider the following natural adaptation of Kruskal's algorithm: Sort all elements e by decreasing weights. Greedily (in the sorted order) add element e to the current set S if $S \cup \{e\} \in \mathcal{I}$; otherwise, discard e. Prove that this algorithm finds a maximum weight set in $\mathcal{I}$.

(2) [10 points]
In class, we mentioned that Dijkstra's Algorithm cannot be used as is for graphs with negative edge weights. Here, we will explore a way to make this work, so long as the input graph G has no negative cycles, i.e., cycles C with $\sum_{e \in C} w_e < 0$. (If you have negative cycles, then the notion of "shortest path" is not even well defined.)
Our input is a (let's say directed) graph $G = (V, E)$ with edge weights w_e, which could be positive or negative. Add a new node s and connect it to each original node $v \in V$ with a directed edge of weight $w_{(s,v)} = 0$. Now use some shortest-path algorithm that works with negative edge weights (such as Bellman-Ford, which most of you know, and we will probably briefly review in about two weeks) to find the shortest-path distance from s to each node v. Call this distance b_v; it could be negative. Now, change the weights of the edges $e = (u, v) \in E$ in the original graph to $w'_e = w_e + b_u - b_v$.

(a) Prove that the new weights w'_e are all non-negative.

(b) Prove that for each pair (u, v) of nodes in the original graph, if we compute the shortest-path distance from u to v with respect to the new weights w'_e, and then subtract $b_u - b_v$ from this distance, we get the shortest-path distance from u to v with respect to the original edge weights w_e.
(In case you're wondering what's the point if we're going to call Bellman-Ford anyway: Bellman-Ford is slower than Dijkstra. By calling Bellman-Ford once, we can now compute the shortest paths for all $\Theta(n^2)$ pairs using n calls to Dijkstra, which is faster.)

(3) [10 points]
Here is yet another Minimum Spanning Tree algorithm. As always, $G = (V, E)$ is an undirected connected graph with non-negative edge weights w_e, and you can assume that all w_e are distinct.
One iteration of the algorithm is illustrated in Figure 1.

Algorithm 1 Yet Another Minimum Spanning Tree Algorithm (YAMSTA)

1: **while** there is more than one node remaining **do**
2: Each node v chooses its cheapest incident edge e_v.
3: Contract all chosen edges e_v, resulting in each component (of the graph with edges e_v) becoming one new node. (The new node inherits all edges incident on any of its nodes, but only keeps the cheapest among parallel edges.)
4: Output the set of all edges e_v (across all iterations).

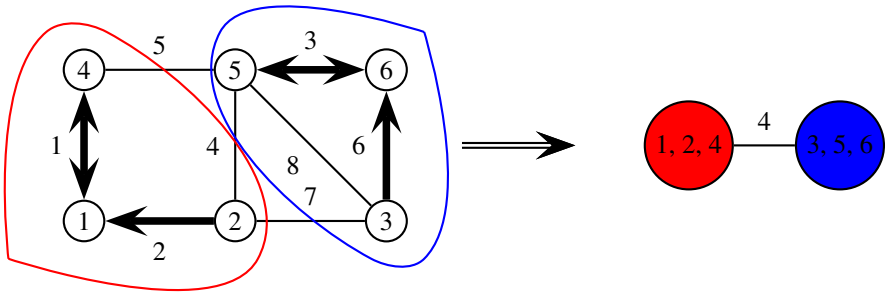

Figure 1: An illustration of a contraction step of the algorithm. The arrows go from the node selecting the edge to the other endpoint. Bold edges are selected, others not. The colored blobs show the contracted components.

(a) Prove that in each iteration of this algorithm, the set of edges e_v that are contracted is acyclic.

(b) Prove that the algorithm computes a minimum spanning tree.

(c) Explain (and analyze) how to implement the algorithm to run in time $O(m \log m)$.

(4) [10 points]
When defining Fibonacci Heaps, we had the rule that once a node lost its *second* child, it itself was removed from the tree (together with all its subtrees). Suppose we changed this rule to "once a node loses its third child, it is removed from the tree". Does the amortized asymptotic time per operation stay the same? In other words, does a sequence of b inserts, c deletes-mins, and d decrease operations still take time $O(b + d + c \cdot \log n)$? Resolve this question by doing exactly one of the following two:

(a) proving that the time stays the same. If you do this, you can use without proof the parts of the analysis that stay the same; you only need to focus on the parts that change, and explain how those changes affect the analysis.

(b) giving a sequence of operations for which the asymptotic (that is, big-Oh) time is longer. If you do this, you should of course also explain why your sequence of operations causes an asymptotic slowdown.

(5) [0 points]
Chocolate Problem (1 chocolate bar): On most/all of the homeworks, we will also have a chocolate problem. Chocolate problems are significantly harder, and they do not per default affect your grade. The reward is actual chocolate.[1] Chocolate problem solutions should be submitted separately from the rest of your solution, since David (not Yusuf) will grade them. You are welcome to solve chocolate problems as a team — in that case, you split the chocolate reward. Optionally, you can also designate a chocolate problem as a substitute for one regular problem; in that case, your score for the chocolate problem will be substituted for the regular problem.[2] Obviously, if you substitute the chocolate problem score for the regular problem, you cannot solve the chocolate problem as a team.

[1] If you have preferences/allergies, feel free to note them on your solution.
[2] This is so that you don't have to solve more problems which might be a bit tedious. Remember that chocolate problems are much harder.

The chocolate problem for Homework 1 is the following, somewhat motivated by a recent pandemic you may have heard about. You are given an undirected graph (social network) in which each edge $e = (v, v')$ has an interval $I_e = [\ell_e, u_e]$ on it. The meaning is that you *know* that v and v' met at some point during the interval I_e, but have no idea when exactly. You also know that some source node s started with some infectious disease, and that a set $P \subseteq V$ ended up with the disease, while a disjoint set $N \subseteq V$ definitely did not catch the disease. (There may be nodes that are neither in P nor in N.) The disease has the property that when two nodes v, v' meet, one of whom has the disease, it is transmitted to the other, who then becomes infectious one time unit later. More precisely, when v meets v' at time t and v was already infectious, then v' will be infectious at every time $t' \geq t + 1$, but not earlier (unless v' was already infected earlier by someone else, of course).

Give (and analyze, i.e., prove correctness and running time) a polynomial-time algorithm which is given G, P, N and the I_e, and either outputs meeting times $t_e \in I_e$ for each edge that explain the observations, or (correctly) concludes that it is impossible that all of P got the disease while no one in N did.