CSc 220: Algorithms
Homework 8 Solutions

Problem 1: A min-max spanning tree of an undirected weighted graph \( G \), is a spanning tree \( T \) for \( G \) that minimizes the weight of the "heaviest" edge in \( T \). In other words: define \( \text{maxw}(T) = \max_{e \in T}[w(e)] \) (rather than the sum of the weight of the edges in \( T \) as in the notion of minimum spanning tree we saw in class).

- Prove that a minimum spanning tree is also a min-max spanning tree;
- Show that you can find a min-max tree in linear time \( O(|V| + |E|) \)

Solution:

- Suppose that a minimum spanning tree \( T \) is not a min-max spanning tree. Let \( e \) be an edge that has the largest weight \( w_e \) in \( T \). Since \( T \) is not a min-max spanning tree, there exists a min-max spanning tree \( T' \) whose edges all have smaller weight than \( w_e \). Remove \( e \) from \( T \): this will disconnect \( T \) in two subtrees. Now replace \( e \) with the edge in \( T' \) that connects those two subtrees (since \( T' \) is a spanning tree, such an edge must exists). Let \( T'' \) be the resulting tree: the total weight of \( T'' \) is less than that of \( T \), contradicting our assumption that \( T \) is a minimum spanning tree.
- To find our linear time algorithm now we proceed as follows. Let \( n = |V| \) and \( m = |E| \). We first give a procedure \( \text{CHECK}(G, b) \) that on input a value \( b \) determines if there is a spanning tree with edges all weight less than \( b \). This procedure simply remove all edges whose weights are greater than \( b \) to obtain \( G' \). Then, using BFS or DFS, checks if the graph \( G' \) is connected. Note that this procedure runs in \( O(n + m) \).

Now let’s proceed to the algorithm to find the min-max spanning tree.

- We find (in \( O(m) \) time) the median of the weights in \( G \), call it \( b \).
- Then we run \( \text{CHECK}(G, b) \). If the resulting graph \( G' \) is still connected then we know that the min-max spanning tree must be in the remaining subgraph. So we recurse on \( G' \).
- If the resulting graph is disconnected, running \( \text{CHECK}(G, b) \) yields a spanning forest (collection of trees) on \( G' \). We reduce \( G' \) by consolidating each tree in the forest into a single "super-node". We then consolidate the edges coming out of that tree to a node \( u \) in \( G - G' \) into a single edge. That is given a tree \( T \) in \( G' \), let \( u_T \) be the resulting supernode: for every node \( v \in G - G' \) there is an edge from \( u_T \) to \( v \) if and only if there is an edge \( (u, v) \) with \( u \in T \). The weight on that edge is taken as the smallest weight among all the edges of the form \( (u, v) \) with \( u \in T \), i.e. \( w(u_T, e) = \min_{e \in T}[w(u, v)] \). Let \( G'' \) be the graph resulting from adding the supernodes and adjacent edges to \( G' \). We now recurse on \( G'' \).

This procedure will find the "optimal" \( b \) for the min-max tree. It will also run in \( O(n + m) \) time since each time we recurse with cut the number of edges in at least half.

Problem 2: We saw in class Dijkstra’s algorithm is asymptotically faster than the Bellman-Ford algorithm to find shortest paths, but it requires all edges to be non-negative.

- Show an example of a graph with some edges with negative weights where Dijkstra’s algorithm fails to find a shortest path;
- Let \( G = (V, E) \) be a weighted graph with weight function \( w \) and some edges with negative weights. Let \( W \) be the minimum weight in the graph with \( w < 0 \). Then for each edge \( e \in E \) change the weight function to \( w'(e) = w(e) - W \). Note that now all edges are non-negative according to the weight function \( w' \). Now run Dijkstra’s algorithm on this graph. What happens? Is the shortest path according to \( w' \) be the same as the one according to \( w \). Why?
- Let $G = (V, E)$ be a weighted graph with weight function $w$ and some edges with negative weights. Let $W$ be the minimum weight in the graph with $w < 0$. Then for each edge $e \in E$ change the weight function to $w'(e) = w(e) - W$. Note that now all edges are non-negative according to the weight function $w'$. Now run Dijkstra’s algorithm on this graph. What happens? Is the shortest path according to $w'$ the same as the one according to $w$? Why?

**Solution:**

- If you run Dijkstra on the graph below the algorithm will assign distance 2 to node $c$. That’s because $a$ is removed from the priority queue before $b$ with "temporary" distance 1 while the correct distance is $-1$. The edge $(a, c)$ is relaxed only when $a$ is removed from the queue, giving $c$ the "permanent" distance 2. Later in the algorithm (when $b$ comes out of the queue) the edge $(b, a)$ is relaxed giving $a$ the correct distance $-1$ but at that point is too late to fix the distance of $c$ since the edge $(c, a)$ will not be revisited.

![Graph Diagram]

- In the graph below (with negative edges) the shortest path from $s$ to $c$ is the path $s \rightarrow a \rightarrow b \rightarrow c$ of weight 0. After you add 1 to all the edges that path has weight 3, while the path $s \rightarrow c$ has weight 2 and is the new shortest path. Therefore the transformation does not preserve shortest paths.

![Graph Diagram]

**Problem 3:** After graduating from City College you moved to a successful career as a data analyst on Wall Street. You notice that there is a potential fortune to be made in currency trading by analyzing real-time currency exchange rates to find a way to turn $1 into more than $1 by a sequence of trades. For example suppose that $1 can buy 1.3EUR and that 1EUR buys .9 Swiss Francs, and that 1 Swiss Franc buys .86$ then by engaging in those trades you would end up with $1.0062.

Given $n$ currencies $c_1, \ldots, c_n$ and a table of mutual exchange rates, show an algorithm that finds such a sequence of trades, i.e. a sequence of currencies $c_i_1, \ldots, c_i_k$ such that one unit of currency $c_i_1$ is transformed into $k > 1$ units of the same currency, after being traded according to the sequence $c_i_1, \ldots, c_i_k$.

Analyze the correctness and running time of your algorithm.
Therefore we can run Bellman-Ford to detect if such a cycle exists. Since this is a complete graph it has $m = n^2$ edges, therefore the running time is $O(n^3)$. 