601.433/633 Introduction to Algorithms **Fall 2024**
Homework #8 **Due:** November 21, 2024, 9am

Bella Lu
Partners: Jecia Mao, Luna Liu

---

1 Removing Train Stations (50 points)

We discussed in class how one of the original motivations for studying max-flow and min-cut comes from WWII, and in particular the questions of "how much stuff can be moved from s to t along the rail network" and "what rail lines can we remove in order to destroy the ability to ship anything from s to t". Now suppose that instead of being able to remove rail lines, we have the ability to sabotage rail *stations*. Can we still find the best way to do this?
More formally, suppose that we are given a directed graph $G = (V, E)$, two vertices $s, t \in V$ with $(s, t) \notin E$, and a function $c : V \setminus \{s, t\} \to \mathbb{R}^{\geq 0}$. As usual, $|E| = m$ and $|V| = n$. We will call $c(v)$ the *cost* of v. A feasible solution is a subset $S \subseteq V \setminus \{s, t\}$ so that there is no path from s to t in $G - S$ (the graph obtained from G by removing all vertices in S and all edges incident on at least one vertex in S). Your goal is to find the feasible solution S which has minimum cost, i.e., which minimizes $\sum_{v \in S} c(v)$.
Design an algorithm for this problem which runs in $O(mn)$ time. You may assume that $m \geq n$ and that there is an algorithm for maximum flow which runs in $O(mn)$ time (as we discussed in class: Orlin's algorithm). Prove running time and correctness.

Algorithm

Step 1:
Create a new graph, G', with same vertices and edges in G, then set every edge in the graph to weight of infinity.

Step 2:
Split each vertex in G' into two nodes (one that accepts incoming edges and one that connects to outgoing edges, respectively v_{in} and v_{out}). The edge weight between every split vertex is $c(v)$. This will be the new graph, which we can call G'.

Step 3:
Run Orlin's algorithm and get the final residual path and the maximum flow.

Step 4:
From the residual graph we can find vertices in G' that are reachable and not by s by doing a run of DFS. This will form a edge cut set. These edges correspond to vertices in the original graph G that was split. We will get a set of vertices whose one split node was reachable in the residual graph but the other one not. This set of vertices will be our final answer.

Correctness

We can first prove that each edge cut in G' will correspond exactly to removing the vertex in G. We will never choose an edge with infinite weight in our modified graph, so each edge we decide to cut in the residual graph will always correspond to a vertex in the original graph. Each (v_{in}, v_{out}) edge contains exacly the weight of each vertex v, c(v). And therefore the maximum flow we get after running Orlin's will reflect the vertex weights in the original graph.
When we remove a vertex (v) in the original graph G, we would also destroy all incoming/outgoing edges attached to the vertex, therefore cutting the flow at that vertex. This corresponds exactly to cutting the (v_{in}, v_{out}) edge in the modified graph G' because if we cut the edge there it also cuts the flow at v_{in} and v_{out}, which is equivalent to cutting out the vertex v.

Since the min edge cut set must be a subset of the set of vertices we can remove from the original graph to destroy the ability to get from s to t, the vertex cut set in G has capacity at least its corresponding edge cut set in G'.

We have proven that the set of vertices we get from running Orlin's on our modified graph G' is a feasible solution in our original problem, and the edge cut set in G' has capacity at most its corresponding vertex cut set in G. Because of that, the solution of G', which is correct in giving the min cut after running Orlin's, also gives the correct solution to our original problem.

Runtime

Creating the new graph, G', takes $O(m+n)$.
Orlin's algorithm runs in $O(mn)$.
Getting the set of vertices from the residual graph requires a run of DFS, which takes $O(m+n\log n)$ with a Fibonacci Heap.

Orlin's algorithm dominates, so the final run time is $O(mn)$.

2 Linear Programming (50 points)

Let's consider linear programming formulations of the minimum spanning tree problem. We now have an (undirected) graph $G = (V, E)$ and weights $w : E \to \mathbb{R}^+$. As we discussed in class, one way of phrasing the minimum spanning tree problem is as finding the minimum cost connected subgraph which spans all nodes. This interpretation naturally gives rise to a straightforward LP relaxation which requires every cut to have at least one edge crossing it (fractionally). More formally, suppose that we have a variable x_e for every edge e, and consider the following linear program. For all $S \subset V$, let $E(S, \bar{S})$ denote the edges with exactly one endpoint in S and exactly one endpoint not in S.

$$\begin{aligned} \min & \sum_{e \in E} w(e) x_e \\ \text{subject to} & \sum_{e \in E(S,\bar{S})} x_e \geq 1 && \forall S \subset V : S \neq \emptyset \\ & x_e \geq 0 && \forall e \in E \end{aligned}$$

Note that there are an exponential number of constraints, but let's not worry about that.

(a) (25 points) Prove that the optimal value of this LP is *at most* the weight of the minimum spanning tree.

We will prove that a MST is feasible in our LP.

By definition of a MST, we are finding a subset of edges that minimums the possible total edge weight. This is precisely what the LP is minimizing, which is $\sum_{e \in E} w(e) x_e$.

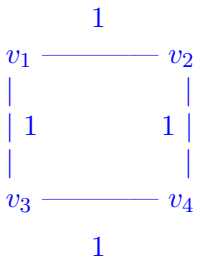
Again, by definition of a MST, for each partition of the graph into two, there exist at least 1 way of getting between the two partitions. Thus, the sum of the number of edges from one cluster to another must be at least 1. This satisfies the constraints that $\sum_{e \in E(S,\bar{S})} x_e \geq 1$ in this LP.

In an MST, the edges of the graph are either in the MST or not, thus each x_e will always be either 1 or 0, for edge that is in the MST or not. Thus satisfies the constraint that $x_e \geq 0$ in this LP.

Thus, since a MST is a feasible solution of our LP, the optimal value of the LP will always be the same or even lower.

(b) (25 points) Unlike the examples in class, this is not an exact formulation of the MST problem. Find a graph $G = (V, E)$ and weights $w : E \to \mathbb{R}^+$ such that the optimal LP value is strictly less than the weight of the MST (and prove this).

Concider this graph:



There are 4 vertices and 4 edges, each with weight 1.

MST will give us an objective function value of 3 because each edge can only be included or not, and every edge must be reachable. This makes us have to include 3 edges. Since each has weight 1, the function value is 3.

Another feasible solution to this LP is having each x_e as $1/2 \geq 0$. This would ensure that $\sum_{e \in E(S,\bar{S})} x_e = 1$, which satisfies the constraint and is therefore feasible. This solution gives a LP value of $4 \cdot (1/2) = 2$, which is less than the MST value.

Since there exists another feasible solution that is strictly smaller than the value MST gives us, the optimal solution must only be at most the value of that alternative solution, which means that the optimal LP value is strictly less than the weight of the MST.