Algorithm Design and Analysis
CSE222 Winter '23

Tutorial 8

The following two properties of MSTs and Graph Traversals would be required for most of the problems below.

i. *Cut Property.* For any edge e, if there exists a cut $S, V \setminus S$ such that e is the cheapest edge (or one of the cheapest edges in case edge costs are not distinct) crossing the cut, then e is necessarily part of the MST of G. This was proved in the class.

ii. *Cycle Property.* If any edge e in G is the most expensive edge in *some cycle* in G, then e cannot be in the MST of G.

Why is this true ? Suppose not - that is e is the most expensive edge of some cycle C and yet it is in the MST. Now consider the cut formed by the connected components upon removing e from the MST. By the Double Cross Lemma (done in class), there has to be another e' that is crossing this cut in G and belongs to the cycle C. Remove e from and add e' to the MST. Clearly this produces another spanning tree - why ? Because each side of the cut forms connected components even after removing e and hence adding e' connects everybody. Further, it is cheaper than the MST - hence a contradiction.

Problem 0 Prove the cycle property.

Problem 1 Given a connected graph $G = (V, E)$ with costs c_e on the edges and a spanning tree $T \subset E$, the *bottleneck* of T is the cost of the most expensive edge in T. The following problem is about finding the minimum bottleneck spanning tree (MBST) of G. For each of the following, either prove the statement or show a counter-example.

a. The MST of G is always an MBST of G.

Solution. Let MST be $T^\star$ and MBST be $\hat{T}$. Suppose the above is not true for the sake of contradiction. Hence the most expensive edge in $T^\star$, say e, is not part of the $\hat{T}$ and further, all the edges in $\hat{T}$ are cheaper than e. Thus, $\hat{T} \cup \{e\}$ has a cycle for which e is the most expensive edge. But then by Cycle property, e cannot be part of $T^\star$ leading to a contradiction.

b. An MBST of G is not necessarily an MST of G

Solution. Easy examples, try to find yourself.

Give a $O(|E| \log |V|)$ time algorithm to find a spanning tree whose bottleneck is minimum.

Problem 2 Suppose you are given a connected graph $G = (V, E)$ with costs c_e on the edges and a *minimum* spanning tree $T \subset E$. Give an $O(|V| + |E|)$ algorithm to find a minimum spanning tree when

a) The cost of a given edge e is decreased.

Solution sketch.

- Case I : If e is part of MST, then the old MST is still the MST. To see this, recall the cut property of edges and MSTs of a graph as done in the lecture. The converse of this is also true. That is, any edge in MST is necessarily one of the cheapest edges crossing some cut - why ? This follows from Prim's algorithm's correctness since every time it adds an edge to the tree, it is indeed (one of) the cheapest edges crossing some cut. Here e is such an edge. Now if the cost of e decreases, it clearly still remains the cheapest edge crossing the corresponding cut. Hence, the MST will remain unchanged.
- Case II : If e is not part of MST, then find the unique fundamental cycle formed by MST union e. Remove the largest cost edge from that cycle. All these can be done using bfs/dfs etc. The correctness follows from cycle property which states - The largest cost edge of any cycle in the graph G is not part of the MST.

b) The cost of a given edge e is increased.

- If e is not part of the MST, it can not become part of new MST
- If e was part of MST - First let us understand what needs to be done. Suppose you remove the edge e from MST. Note that the two components that are formed as a result essentially form the cut which certifies the existence of e in the MST (by the Cut Property and Prim's Algorithm). Now all you need to do is choose the minimum cost edge that is crossing this cut and plug it back (note that this could be the edge e itself). This can be done in linear time as follows. Firstly, the connected components can be be found by running two BFS/DFS - one from each of the endpoints of e. Then just find the minimum edge crossing this cut and add it to T - clearly this can be done in $O(E)$ time.

Problem 3 Suppose that an n-vertex graph $G = (V, E)$ contains two vertices s and t such that the distance between them is strictly greater than $\frac{n}{2}$.

(a) Prove that there exists a vertex u that is neither s nor t such that deleting u destroys all the s-t paths in G.

(b) Give an $O(m + n)$-time algorithm to find such a vertex v.

Solution:

(a) The proof idea is based on a counting argument. We have to exploit the property of BFS carefully. If the BFS algorithm is started from the vertex s, then it computes a value $Layer(v)$ (as discussed in the lecture) for all the vertices $v \in V$. This value $Layer(v)$ contains the distance from s to v, i.e. the number of edges in a shortest path from s to v in G. Note that $Layer(s) = 0$ and $Layer(t) > n/2$. Let us now consider the vertices that occur with Layer-values $1, 2, \ldots, \lfloor n/2 \rfloor$ of the BFS tree T_s starting from s, i.e. all the vertices that are of distance at most $n/2$ from s. At least one such vertex has to be deleted in order to destroy all s-t paths. Suppose for the sake of contradiction that there is no vertex u such that deleting u destroys all the s-t paths in G.
Case 1: n is odd. Since $\lfloor n/2 \rfloor + 1 = \lceil n/2 \rceil$, it holds that $Layer(t) \geq \lceil n/2 \rceil$. Moreover, $2\lfloor n/2 \rfloor = n - 1$. By our assumption, for every $1 \leq i \leq \lfloor n/2 \rfloor$, there are 2 vertices x, y in

the BFS-tree T_s such that $Layer(x) = Layer(y) = i$. Then, the total number of vertices that occur with Layer-values $1, 2, \ldots, \lfloor n/2 \rfloor$ of the BFS tree T_s is at least $n/2-1$ excluding the vertices s and t. This contradicts that the total number of vertices is n.

Case 2: n is even. The proof idea is similar as before.

(b) Using the proof idea above the algorithm is simple. The main steps are the following.

- Invoke $BFS(G, s)$ starting from s and compute an array contaning the $Layer(v)$ values for all $v \in V$.
- For every $1 \leq i \leq \lfloor n/2 \rfloor$ check if only one vertex $u \in V \setminus \{s, t\}$ satisfies $Layer(u) = i$. This can be done in $O(n)$-time by putting a counter array. As soon as such a vertex u is found, return that vertex u.

Note that this entire process takes $O(n + m)$-time.