CS/ECE 374 FALL 2018
Homework 9 Problem 1

Zhe Zhang (zzhan157@illinois.edu)
Ray Ying (xinruiy2@illinois.edu)
Anqi Yao (anqiyao2@illinois.edu)

Spanning trees have many nice algorithmic properties and are useful in a number of applications. For those interested, see the connection to abstract structures called matroids.

- Consider the following "local-search" algorithm for MST. It starts with an arbitrary spanning tree T of G. Suppose $e = (u, v)$ is an edge in G that is not in T. It checks if it can add e to T and remove an edge e' on the unique path $p_T(u, v)$ from u to v in T such that tree $T' = T - e' + e$ is cheaper than T. If T' is cheaper then it replaces T by T' and repeats. Assuming all edge weights are integers one can see that the algorithm will terminate with a "local-optimum" T which means it cannot be improved further by these single-edge "swaps". Assuming all edge weghts are distinct prove that a local-optimum tree is an MST. Note that you are not concerned with the running time here.
- We saw in lecture that Borouvka's algorithm for MST can be implemented in $O(m \log n)$ time where m is the number of edges and n is the number of nodes. We also saw that Prim's algorithm can be implemented in $O(m + n \log n)$ time. Obtain an algorithm for MST with running time $O(m \log \log n)$ by running Borouvka's algorithm for some number of steps and then switching to Prim's algorithm. This algorithm is better than either of the algorithms when $m = \Theta(n)$. Formalize the algorithm, specify the parameters and argue carefully about the implementation and running time details. No proof of correctness required but your algorithm should be clear.
- **Not to submit but encouraged to solve:** Let $G = (V, E)$ be an edge-weighted undirected graph. We are interested in computing a minimum spanning tree T of G to find a cheapest subgraph that ensures connectivity. However, some of the nodes in G are unrealiable and may fail. If a node fails it can disconnect the tree T unless it is a leaf. Thus, you want to find a cheapest spanning tree in G in which all the unreliable nodes (which is a given subset $U \subset V$) are leaves. Describe an efficient for this problem. Note that your algorithm should also check wither a feasible spanning tree satisfying the given constraint exists in G.

Solution:

1. Let T be the local-optimum spanning tree of G. We want to prove that the tree T is an MST. Because of the property of Spanning tree, T would contain all the vertices in G. Since the edge weights are all distinct, we guarantee to have $n-1$ safe edge that together build an MST. By the characteristic of tree, there will be only one path $p(u, v)$ from u to v for every pair of vertex in T, and also adding one edge e to T will create a cycle c that contains e. Since T cannot be improved by single-edge swaps, it means whatever edge e in G that is not in T added to T, the edge e will have the most largest weight in the cycle that contains e otherwise we can perform the swap again, so every edge in G not in T will be the unsafe edge. The number of $n-1$ edges in T will be the safe edge and hence, local-optimum spanning tree T will be the minimum spanning tree.

2. The running time for Borouvka's algorithm for MST is $O(mlogn)$ since we start with n components, so it runs in $log(n)$ time and we take m edges each iteration. As for Prim's algorithm, we have to check n vertices each iteration instead of edges. In order to have $O(mloglogn)$ running time, first, we have to perform the Borouvka's algorithm for $O(loglogn)$ steps, which won't help up finish the whole MST tree but as we put the small trees in a list L during the process of Borouvka's algorithm, we have all the small trees. Now, we can apply the Prim's algorithm. Treat the small tree as a vertex, or just call it "supervertex". Each edge that between vertex of two trees will now be the edge connecting the two supervectices. Building the graph will take linear time since the number of edge is the same and the number of vertex is the number of small trees. The running time for the Prim's algorithm on the second step will be $O(m + (n/logn)logn) = O(n + m)$. The running time for the first step is $O(mloglogn)$. So the total running time will be $O(mloglogn)$.

■