CS 325 Group Project 4

Robert Detjens, Srikar Valluri, Felix Brucker

Scenario

In class, we have seen multiple algorithms for computing minimum spanning trees. In this assignment, we design algorithms for computing the second and third minimum spanning trees. For example, the following figure shows a graph, and its first, second, and third minimum spanning trees.

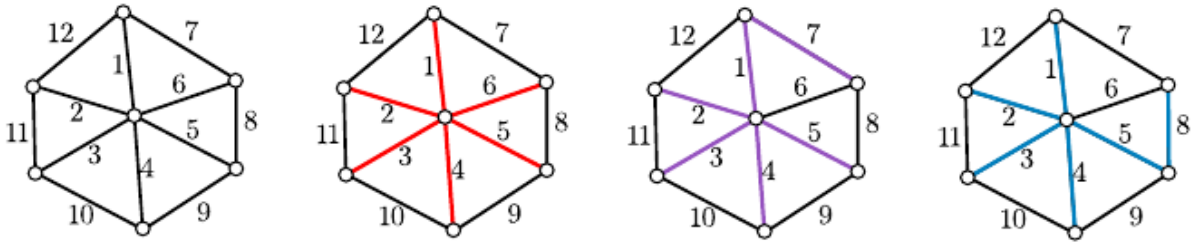

Figure 1: Example of first, second, and third MSTs

Algorithm Description

The base algorithm used is Prim's algorithm. An arbitrary node is chosen as the starting point. Starting with it, nodes are added to a singular forest of vertices. Each step of the algorithm, all the edges connected to but not in the forest are evaluated. The one with the smallest weight is chosen and the node it leads to is added to the forest using a priority queue. This continues until the queue is exhausted, at which point all nodes have been visited and added to the MST.

The program in its entirety calculates the first MST using Prim's algorithm. To calculate the second and third MSTs, another approach is used. The second and third MSTs will be one edge different than the first MST, so this algorithm calculates all MSTs one edge different from the first. The resulting MSTs are ranked by total weight and the two lightest are the second and third MSTs respectively.

Runtime Analysis

When run on the graph and memoized with the queue, Prim's algorithm has a known runtime of $O(E \log V)$, where E is the number of edges and V is the amount of vertices in the graph. The program is deployed once for every edge in the first minimum spanning tree, plus one time to get the first MST. The amount of repetition is V (recalculate Prim's for each edge in the first MST), so the big O of the entire program is $O(VE \log V)$.

Proof of Correctness

Prim's algorithm is known to correctly find the first minimum spanning tree.

To prove that the second and third MSTs are one edge different from the first, consider the following:

Let e_1 and e_2 be some edges in the first MST. These edges are known to be the minimum edges in their local area of the graph due to them being in the MST.

Let e_1' and e_2' be adjacent edges to e_1 and e_2 respectively. e_1' and e_2' are known to not be in the MST and carry a higher weight than e_1 or e_2.

Replacing e_1 with e_1' will increase or maintain the total weight of the MST by forcing the use of a non-optimal path. Replacing e_2 with e_2' will also increase or maintain the total weight of the MST by forcing the use of a non-optimal path.

Replacing both e_1 and e_2 will increase the total weight by more than either single replacement, as there are now two non-optimal edges in the new tree.

Every spanning tree 1-edge-different from the MST is calculated. Of the resultant trees, the one with the least difference between edge and replacement is chosen for the second MST. The one with the second least difference is the third MST. These MSTs must be correct because all possibilities are searched.