COSC 6320: Data Structures and Algorithms – Programming Assignment 3

Aisha Farooque (PSID: 1809992)

October 23, 2022

**Leetcode Profile Link:** <https://leetcode.com/u1809992/>

**Introduction**

At the very core of the algorithm is computing the MST of each graph three times:

1. MST with all edges
2. MST by forcing the inclusion of the edge
3. MST by forcing the exclusion of the edge

Critical edges are defined as edges which, if excluded, will increase a graph’s MST weight. Pseudocritical edges are defined as edges which, if excluded, do not affect a graph’s MST weight. Cases 2 and 3 defined above determine which category an edge will fall into depending on the weight in Case 1. That is:

1. If , then
2. If , then

**Algorithm Explanation**

Boruvka’s algorithm for MST is a greedy algorithm that works for connected graphs with distinct weighted edges. The core idea is to connect vertices using the shorted edges between each subset/component. Before running Boruvka’s algorithm, the code to find critical and pseudocritical edges also runs a “connected-components check”. Checking ensure that the graph is connected which guarantees that Boruvka’s will never fail or be stuck in an infinite loop.

In the beginning, all vertices are separated components/forest of their own, i.e., each vertex is its own parent. A vertex’s parent helps recognize which subset they belong to which is helpful when performing edge contraction.

For each edge in the graph, we use a vertex’s parent to perform a find operation and compute which subset they belong to. If the two vertices in an edge, u and v, belong to different subsets, then we update their closest edge depending on the weights connecting these two subsets. Then iterating through vertex, we will consider each edge, find it’s cheapest edge into another subset and contract them into one subset. Every time an edge is contracted, the number of subsets, which initially equal to the number of vertices, is reduced by one. The edge is also included in the MST.

We will repeat these operations until we have one subset left, which will be the minimum spanning tree. We will exit the loop and return the weight of the MST. e

**Algorithm Correctness**

The algorithm outputs the critical and pseudocritical edges as defined in the rules in the Introduction section. For each rule, we’re computing the MST. We can prove the algorithms correctness by leveraging the cut property. Consider a minimum weighted edge, , on an iteration *i*.

Let *c* denote a set connected forest of edges with minimum weighted edges crossing the cut. If our current edge *e* is the one to be skipped, then *c* will include cheap edges from . If the current edge is the one to be forcibly picked, then *c* will include *e* and then add minimum weighted edges later.

Thus, according to the cut property, *e* must be an edge included in the MST. Since Boruvka’s algorithm only works on distinctly weighted edges, we are guaranteed that the MST is unique. Every time an e is included in the MST, the number of sets inside *c* is reduced by one.

We will repeat contraction of the edges until we’re left with exactly one set inside *c*. The edges selected by the algorithm will be the MST.

Using the weight of the resulting MST, we can separate edges into critical and pseudocritical categories as defined previously.

Algorithm Complexity

Each iteration of Boruvka’s algorithm reduces the number of connected sets by half, therefore, the bigger while-loop runs in O(log(v)) time where v is the number of vertices.

Inside each iteration of the while loop, we have two for loops that run for all edges and all vertices, which makes the resulting time equal O(e \* v \* log(v)) where e is the number of edges.

The algorithm is called twice for each edge, once to pick an edge and the other to skip the edge. This makes the running time equal O(2e \* e \* v \* log(v)). Therefore, the overall runtime of the algorithm is O(3e \* v \* log(v)) = O(e \* v \* log(v)).

Pseudocode

Code

**Leetcode Submission**

Note – Leetcode runtime is wildly unreliable and produces different runtimes for the same piece of code leading to inconsistency. This seems to be a problem with other users as well, <https://leetcode.com/discuss/general-discussion/136683/different-run-time-with-same-code>.

For example, here’s the output when the same code is run within the same time frame without modification.

Table

Description automatically generated

If running the code for the first time produces a TLE, please try again. Chances are that the code was executed on a slow server.

Submission Link - <https://leetcode.com/submissions/detail/833241183/>

Leetcode Profile Link - <https://leetcode.com/u1809992/>

