

## Programming Assignment 2

Assigned: Nov 21

Due: Dec 7 (Fri, 11:59 PM)

---

Your second programming assignment is to implement **Kruskal's** algorithm for Minimum Spanning Trees, using the Union-Find data structure for efficient cycle-detection.

**Kruskal's Algorithm**

The input graph for your program is given as a **list of edges**, with their weights, as shown below. The format lists one edge per line:

```
(4,6) 20
(1,3) 5
(2,9) 18
etc.
```

Recall that the algorithm begins by sorting the edges in **increasing** order of weight, and then one-by-one scans each edge to determine whether it should be added to the MST. A new edge  $(u, v)$  is added to the MST if and only if  $u$  and  $v$  belong to different subtrees. If so, the edge  $(u, v)$  is added to the MST **and** the subtrees containing  $u$  and  $v$  are “unioned”. If  $u$  and  $v$  were already in the same subtree, then the edge  $(u, v)$  is not included in the MST.

**Union-Find Structure**

In order to efficiently check whether or not two nodes  $u$  and  $v$  are in the subtree, you will use the Union-Find data structure. Each set will represent a subtree in the current group of subtrees, created by Kruskal's algorithm. In particular, initially each node is its own subtree, as a singleton. As new edges are added to MST, subtrees containing the endpoints of those edges are merged, using the UNION operation. When the algorithm needs to decide if  $u$  and  $v$  are in same or different subtrees, it performs  $\text{FIND}(u)$  and  $\text{FIND}(v)$  to decide this.

You must implement the Union-Find data structure as described in class (or textbook) with **path compression and union-by-rank**.

**Sorting of edges**

You can use either a system provided  $O(n \log n)$  sorting routine, or use the binary Heap structure from Programming Assignment 1 to extract the edges in the sorted order. But be sure that the sorting takes  $O(m \log m)$  time, if there are  $m$  edges being sorted.

**Issues to deal with**

Make sure your program gracefully deals with incorrect data. For instance,

1. If an edge has the format  $(u, u)$  (that is, both endpoints are the same), then it should NOT be included in the MST construction. But the program should not abort; it should continue without this edge.
2. The vertex indices should be positive integers; anything else should be reported as an error, but the program should continue with the rest of the input.
3. The edge weights can be arbitrary real numbers (floats). It is okay for an edge to have a NEGATIVE weight. Convince yourself that Kruskal's algorithm works correctly even with negative edge weights.
4. What happens if the input graph is NOT connected? That is, upon termination you still do not have a spanning tree. Report an error for this at the end.

## Program Output and Grading

- Please make sure that the format of your program output matches that of the sample output EXACTLY.
- (50 pts) Your program should read input from a file and print progress and diagnostics for each edge. That is,
  1. if the edge is added to MST, it should say so.
  2. if the edge creates a cycle, and not included in MST, it should say so.
  3. if the edge gives any error (due to self-loop, invalid node ID etc), report it.
  4. The following example shows what your output should look like:
 

```
Edge (1,4) successfully inserted
Edge (4,6) creates cycle
Edge (7,2) Invalid Node ID
```
- (50 pts) Upon termination, your program should
  1. list out the edges of the MST (one edge per line), and the total weight of the MST;
  2. or, if there was an error (e.g. MST not found), report the error.
  3. The following example shows what your output should look like:
    - If the tree is successfully built, list the edges of the minimum spanning tree, one per line, followed by the weight of the tree on the last line:
 

```
4 6
1 3
2 9
47
```
    - If there was an error, write out an ERROR LINE:
 

```
ERROR: MST not found.
```