Your assignment is to implement Kruskal’s algorithm for Minimum Spanning Trees using the Union-Find data structure for efficient cycle-detection.

Description
Kruskal’s algorithm begins by sorting the edges in increasing order of weight, and then scans edges in this order to determine whether they 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.

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 FIND(\(u\)) and 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.

You can use any sorting routine but be sure that it runs in \(O(m \log m)\) time, if there are \(m\) edges being sorted.

Negative weights and disconnected graph

1. The edge weights can be negative.

2. Report an error if the graph is not connected. That is, if upon termination the set of accepted edges do not form a spanning tree, report an error.

Input, Output and Grading

- Your programs will be autograded and should run on CSIL as \./prog2 < input.txt. So, please make sure that the format of your program output matches that of the sample output exactly.

- The first line of input contains two numbers \(n\), and \(m\), indicating the number of nodes, and edges, respectively. The next \(m\) lines provide the input graph as a list of edges, along with their weights. The format lists one edge per line starting at the second line. The node IDs are assumed to be 1, \ldots, \(n\). The following is a small example sample.txt:
4 6
1 2 10
2 3 9
1 4 4
2 4 7
1 3 5
3 4 2

• (70 pts) Your program should read input from stdin and print progress and diagnostics for each edge to stdout. That is,
  1. if the edge is added to MST, it should say so.
  2. if the edge creates a cycle, and it is not included in MST, it should say so.

• (30 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.

• The following example shows what your output should look like on the sample input:

```
user@csil~] ./prog2 < sample.txt
Edge (3,4) successfully inserted
Edge (1,4) successfully inserted
Edge (1,3) creates cycle
Edge (2,4) successfully inserted
 3 4
 1 4
 2 4
 13
```

• The autograder will additionally measure the time taken to execute the tests. Test cases will include sizeable input, and contain cases with and without MST's. We will be looking for reasonable speed here to check that your implementation is efficient (has properly implemented Kruskal and Union-Find).

Submission Instructions

1. Include a README file. This must contain the following information in the given format.
   
   CSIL login : [Your username here], UCSB Email : [Your ID]@umail.ucsb.edu
   
   This is needed to map your CSIL usernames to your records.
2. The recommended programming language is C++, however you may write your program in other languages. Always make sure to include a ‘makefile’, so that running ‘make’ compiles your source files and generates the executable ‘prog2’.

3. In order to electronically turnin your project files, use the following command:

    turnin prog2@cs130a LIST

where LIST is the list of all your files, i.e., your makefile, all your source and header files. (The turnin program is located in /usr/bin/turnin on csil.cs.ucsb.edu only.) If you turnin more than once then the version that will be graded is the last version you turned in before the deadline.

4. Late submissions will not be accepted!