Overview: You are to finish implementing `Kruskal.java`, a Java implementation of Kruskal’s algorithm, which computes the minimum spanning tree for a given input graph with weighted edges. The input reading phase is already done for you. The remaining steps are left to you: they involve sorting the edges, and then selecting the output edges using a union-find data structure, as further detailed in the commentary of `Kruskal.java`.

Compared to previous homework assignments, there is much less structure here—it is up to you design and implement this structure on your own. As always, I’m available for any questions that you may have.

I/O Format: In both the input and the output, the first line has the format:

\[ \langle N \rangle \langle M \rangle \]

That is two integers in decimal: the number of vertices \( N \) and the number of edges \( M \). Each subsequent line (of either the input or output) describes a weighted edge, in this format:

\[ E \langle u \rangle \langle v \rangle \langle w \rangle \]

That is the letter “E”, followed by a space and then the three integers \( u \), \( v \), and \( w \). Here \( u \) and \( v \) are the vertex id’s (integers in the range 0 to \( N-1 \)) of the two endpoints of the edge, and \( w \) is the weight of the edge (an arbitrary integer).

Note that the provided “graph generator” program `GraphGenerator.java` produces random output graphs in the above format.

Getting Started: As usual, you start by entering these commands at a Math/CS lab machine:

```
cd /home/cs171000/share/hw7
make copy
cd ~/cs171/hw7
```

You should only modify `Kruskal.java`. As usual, the Makefile defines some useful actions:

```
make compile compile Kruskal.java
make run run Kruskal.java on some known test graphs
make turnin try to turnin
```

Carefully read the comments in `Kruskal.java`. These comments outline a suggested approach to completing the assignment.

Grading: You must successfully “make turnin” to get a non-zero mark.

Note that to get the expected output, you must use a stable sorting algorithm (so edges with the same weight will stay in their input order). Your program should work correctly on graphs which are connected; for a small bit of extra credit (5 points), your code should also work on graphs that are not connected. In that case the “minimum spanning tree” is actually a minimum weight forest, spanning each component of the graph. Your program should be reasonably fast (just make reasonable choices for your sorting algorithm and your union-find structure).

As usual, your work is governed by the Emory Honor Code and the Math/CS SPCA.