Assignment 3

Introduction
Your task is to implement Kruskal’s algorithm using disjoint sets. The disjoint set version must implement union by rank as well as path compression during find.

Input
Your program will process a file containing a description of a graph. A graph description contains an arbitrary number of edge descriptions. An edge description consists of two vertex descriptions (optionally followed by a weight) followed by a semicolon. A vertex description is simply a non-negative integer. If a weight is omitted, a weight of 1 should be assumed. A weight is a positive integer.
The file should be free format; whitespace may appear anywhere. Here a sample graph description:

    1 5 ;
    2
    10
    23 ;
    214 33 1
    ;

which is equivalent to:

    1 5 ;
    2 10 23 ;
    214 33 1 ;

There are six vertices, named 1, 2, 5, 10, 33, and 214, and three edges, 1 to 5, 2 to 10 and 214 to 33, with weights 1, 23, and 1, respectively.

The name of your executable should be kruskal and the name of the file describing the graph will be passed to your program as a command line argument, as in:

    $ cat g1
    1 2 1 ;
    2 3 2 ;
    3 1 3 ;
    $ kruskal g1
    [output appears here]

Your program should be capable of interpreting the graph description as either a directed or undirected graph. Switching between the two interpretations is to be accomplished by providing the command line option -u, which signifies the graph should be undirected. The default behavior should be a directed graph interpretation.

Your program should report a minimum spanning tree covering all vertices reachable from the root. The root is specified by the -r option. For example, to specify vertex 10 as the root, kruskal would be called similarly to:

    kruskal -u -r 10 graph

where graph is the name of the file containing the graph description. If the -r option is not given, the root should default to the first vertex mentioned in the graph description. Options are given prior to the file name of the graph and can occur in any order.

Output
The output of your program should be a spanning tree displayed as an adjacency list. The adjacency list should be sorted both vertically and horizontally. If some vertices are not reachable, your program should report this situation with the message:

    n vertices were not reachable

with n being the number of unreachable vertices. The total weight of the tree should also be displayed. Here is an example display:
$ cat g2
10 0 x;
6 7 x;
5 9 x;
4 8 x;
1 6 x;
0 6 x ; 0 5 x ; 0 4 x; 0 3 x; 0 1 x;
$ kruskal -r 0 g2
A minimum spanning tree is:

0: 1,3,4,5
1: 6
3:
4: 8
5: 9
6: 7
7:
8:
9:
10: 0

The total weight of the tree is 47.
3 vertices were not reachable from root 0.
$

If all vertices are reachable, the message should be similar to:

All vertices were reachable from root 0.

Implementation details
You may implement your program in any language you wish, as long as I can compile or interpret your source code on Beastie. You must provide a makefile which responds properly to the command make. Your program must compile cleanly and with no warnings or errors and it must be compiled with the highest level of error checking (the -Wall option for gcc and g++). You may develop on any system you wish but your program will be compiled and tested on hopper. Only the most foolish student would not recompile and thoroughly test the implementation on a Linux system. If the language you chose does not provide for a direct executable, provide an appropriate shellscript named kruskal which runs your program.

In order for your program to run on a randomly created graph, if an edge is given more than once, ignore subsequent occurrences. For undirected graphs, if a \u; v edge is given, ignore subsequent \u; v and \v; u edges, since the presence of a \u; v edge implies the presence of a \v; u edge.

Documentation
All code you hand in should be attributed to its author. Comment sparingly but well. Do explain the purpose of your program. Do not explain obvious code. If code is not obvious, consider rewriting the code rather than explaining what is going on through comments.

Grading
Your implementation is worth 100 points. Implementations that do not pass the test program will not be graded. Points will be deducted for sloppy formatting, insufficient or overly verbose documentation, compiler warnings, run-time crashes, and other such transgressions. I will not be programming for speed, but for simplicity and elegance and you should have no trouble reaching this goal.

Submitting the project
When you are ready to send me your implementation, delete all intermediate files and executables. Then send me all these files in your directory by running the command:

submit cs201 lusth assign3

Again, your implementations may be developed on other hardware and operating systems, but they must also compile and run cleanly and correctly on hopper.