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-set.

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 ;
```

In this example 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 must 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 an undirected graph.

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 -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. In general, 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 breadth-first traversal, the weight of the tree and the number of unreachable vertices. Here is an example display:

```
$ cat g2
10 0 9 ;
6 7 11 ;
5 9 1 ;
```
You should perform a breadth-first (level-order) traversal of the spanning tree rooted at the starting vertex. Each level of the traversal should be preceded by the level number (starting at 0), followed by the vertices at the level. Each vertex in a level is followed by its predecessor (in parentheses) followed by the weight of the edge from the predecessor to the vertex in question, and terminated by a semicolon. The vertices in a level should be ordered by increasing edge weight. You must follow the format exactly as diff will be used to assess your output.

If all vertices are reachable, the last line should be:

```
unreachable: 0
```

**Program organization**

You must implement the disjoint set code as a separate class or module. You must also name your disjoint set methods or functions as follows:

- `findSet`
- `union` (or `unionSets`)
- `makeSet`

**Other details**

You may implement your program in any language you wish, as long as I can compile or interpret your source code on beastie.cs.ua.edu. Only the most foolish student would not recompile and thoroughly test the implementation on a Linux system.

You must provide a makefile which responds properly to the commands `make`, `make test`, and `make clean`. The `make` command must compile your program with no errors or warnings and it must compile with the highest level of error checking (the `-Wall` option for `gcc` and `g++`). The `make test` command should run your program through some test files of your choosing. The `make clean` command should remove all intermediate files, such as `.class`, `.o`, and `.pyc` files.

If the language you chose does not provide for a direct executable, provide an appropriate shellscript named `kruskal` which runs your program.

Here is an example for `Java`:

```
$ echo java -classpath classfiles main.Kruskal \$\* > kruskal
$ chmod +x kruskal
```

and here is an example for `Python`:

```
$ echo python3 kruskal.py \$\* > kruskal
$ chmod +x kruskal
```

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.

**Restrictions**

You cannot use any fixed-sized data structures for storing the graph or the disjoint set (unless you preprocess the input to determine its extent). You many not use built-in data structures for your disjoint set.
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 compile will not be graded.

A regrade will cost 20 points, unless the reason for the regrade is not your fault.

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 beastie.