Assignment 3

Introduction

Your task is to implement Prim’s algorithm for undirected graphs, using a binomial heap as the basis for a priority queue.

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 vertices (optionally followed by a weight) followed by a semicolon. A vertex 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 `prim` 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 ;
$ prim g1
[output appears here]
```

Your program should interpret the graph description as an undirected graph and should report a forest of minimum spanning trees that cover all vertices.

Execution

If more than one edge is eligible to be added at any given point, chose the edge with the smallest vertex, breaking further ties with the smallest remaining vertex. For example, if 13 is the smallest edge weight among the remaining edges and there are these edges with weight 13 available:

```
3 5 13 ;
2 8 13 ;
2 9 13 ;
```

then the 2-8 edge would be added first, followed by the 2-9 edge, followed by the 3-5 edge.

Output

The output of your program should be a spanning forest, with each tree displayed as a breadth-first traversal along with its weight. Trees are presented in increasing order, based upon the smallest vertex in the tree. Here is an example display:
For each tree in the forest, you should perform a breadth-first (level-order) traversal, rooted at the smallest vertex. Each level of the traversal starts with the level number (level 0 is the first level) and a colon and is followed by a comma separated list of vertex descriptions. A vertex description is the vertex followed by its breadth-first-search predecessor (in parentheses) followed by the weight of the edge from the predecessor to the vertex in question. The vertex descriptions in a level are to be ordered by increasing vertex number. Each level is terminated by a semicolon. You must follow the format exactly as diff will be used to assess your output.

Program organization

You must implement the your binomial heap as a separate module named `binheap.c` and `binheap.h`. You must define the following methods:

- `binheap *newBinHeap(int (*)(void *,void *))`;
- `void insertBinHeap(binheap *,void *)`;
- `void *extractBinHeap(binheap *)`;

Your binomial heap module is to be generic. That is to say, you can store any kind of (pointer) data in the heap. The constructor `newBinHeap` takes in a comparison function, which will allow it to work on any kind of data. A comparator for an `edge` class might will look something like this:

```c
int edgeComparator(void *e1,void *e2)
{
    //cast from the generic type to the actual type
    edge *edge1 = (edge *) e1;
    edge *edge2 = (edge *) e2;

    //return a positive number if edge1's weight > edge2's
    //return a zero if they are equal
    //return a negative number if if edge1's weight < edge2's
    ...
}
```

The edge comparator function should, of course, reside in your `edge` class, not in your `binheap` class.

A call to the `binheap` constructor might look like:

```c
binheap *h = newBinHeap(edgeComparator);
```

Other details

You must implement your program in C. 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 prim`, `make test`, and `make clean`. The `make` and `make prim` commands must compile your program with no errors or warnings and it must compile with the highest level of error checking (the `-Wall` and `-Wextra` options). The `make test` command should run your program through some test files of your choosing. The `make clean` command should remove all intermediate files, including the executable `prim`. 
In order for your program to run on a randomly created graph, if an edge is given more than once, ignore subsequent occurrences. 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 (unless you preprocess the input to determine its extent) or any other data whose size changes depending upon the input. You many not use built-in data structures. and all structures must run as efficiently as commonly expected.

**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.

**Handing in results**

For the preliminary test, delete all intermediate files and executables. Then, send me all the files in your directory by running the command:

```
submit cs201 lusth prelim
```

For your final submission, use the command:

```
submit cs201 lusth assign3
```

Again, your implementation may be developed on other hardware and operating systems, but it must also compile and run cleanly and correctly on a Linux system. You may submit as many times as you like, up to the deadline.