lab_graphs Gory Graphs

Due: Sunday, December 6 at 11:59 PM
Doxygen for lab_graphs

Assignment Description

In this lab you will:

- Use a traversal to find a particular edge in a graph.
- Find the shortest path between two vertices.
- Implement Kruskal's minimum spanning tree algorithm.

Checking Out The Code

Get the code in the usual way.

DisjointSets

Note that your `DisjointSets` class needs to be finished in order to do this lab. If you haven't finished MP7.1, you should work on that now instead.

After running `svn up`, copy your `dsets` from `mp7` to `lab_graphs`:

```
TERMINAL
cd lab_graphs
cp ../mp7/dsets.cpp .
cp ../mp7/dsets.h .
```

This lab has a graph library already implemented for you; it is your job to write some functions that make use of it. These functions are contained in the namespace `GraphTools`, and you can implement them by editing `graph_tools.h` and `graph_tools.cpp`. These are the only files that will be used for grading this lab. `demo.cpp` shows you how that graph class can be used, and `tests.cpp` calls your functions and creates output images. Some code to create some specific graphs is located in `premade_graphs.cpp`. The functions there will create graphs that represent maps of the United States, Europe, and Japan. You can have `tests.cpp` call your functions on these map graphs, or on random graphs.

The source code in this lab is documented heavily. Use this to your advantage. Above each function (including the ones you need to write), is a comment describing what the function does, its inputs, and its outputs. Hints and notes are included in the documentation above the functions you will write.

For a list of files and their descriptions, see the Doxygen for this lab.

The Graph Library

`demo.cpp` shows several ways that the `Graph` class and its functions can be used. Use this opportunity to familiarize yourself with the available functions. All functions are very similar (if not identical) to those described in lecture. By default, running

```
TERMINAL
make graphdemo
./graphdemo
```

will print two graphs to your terminal and put additional graph image files in the `images/` directory.

To help you debug your code, each edge is colored according to its edge label when graphs are printed as PNGs. Edge labels can be set with the `setEdgeLabel()` function. The coloring scheme is as follows:

```
"UNEXPLORED" -> black (default)
"MIN"        -> blue  (solution)
"MST"        -> blue  (solution)
"MINPATH"    -> blue  (solution)
"CROSS"      -> red
"DISCOVERY"  -> green
"VISITED"    -> grey
```

So if this line appears in your code, `graph.setEdgeLabel(u, v, "MIN")`, the edge between vertices `u` and `v` would appear blue to denote that the edge is the minimum weighted edge (i.e., if you were doing `findMinWeight()`).

Please note that the default edge label and vertex label is empty string. If you are doing a traversal or need to rely on the labels for any reason, you should initialize them to some value or consider empty string as `"UNEXPLORED"`.

Another useful function for debugging is `snapshot()`, a member function of the `Graph` class. By default, `tests.cpp` print out a picture of your graph after you're done with it, but what if you wanted to see how your graph labels are changing as you traverse it? For example,

```C++
C++
// do a BFS on the graph g

// setup snapshot feature with your image title
g.initSnapshot("myBFS");

while(...)
{
    // traverse the graph
    g.snapshot();
    // label edges, etc
    // ...
}
```

will create a PNG for each iteration of the `while` loop, so you can see how the traversal progresses. Ideally, edges will change from `"UNEXPLORED"` to `"DISCOVERY"`, `"CROSS"`, or `"VISITED"`. You will be able to see this by watching the edges change color while flipping through the generated files `myBFS0.png`, `myBFS1.png`, `myBFS2.png`, etc in `images/`.

One last bit of information: if you run

```
make tidy                                                          TERMINAL
```

all the PNG files in `images/` will be deleted. This is useful because they can accumulate fast, especially if you are liberally using `snapshot()`.

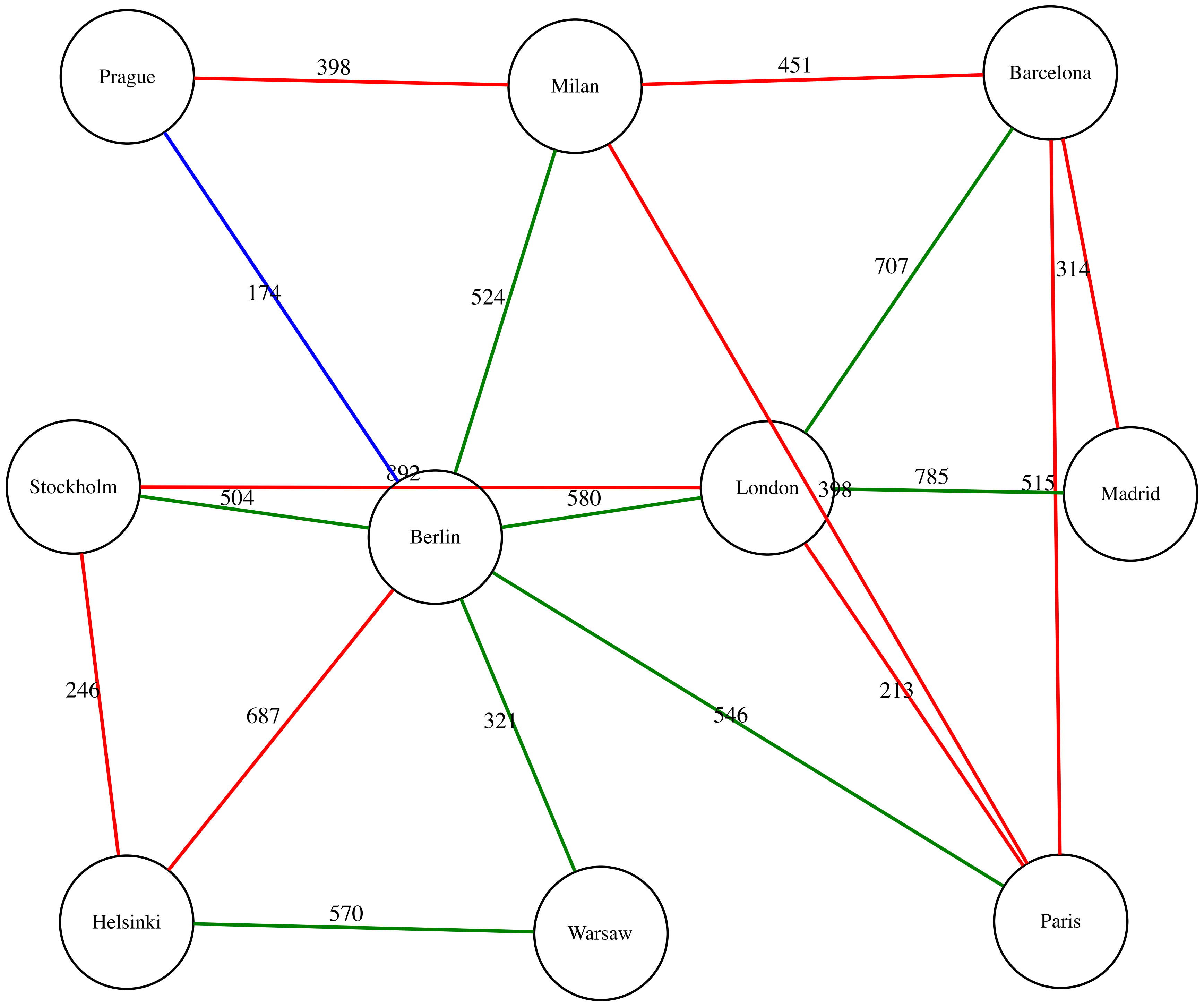
`int GraphTools::findMinWeight(Graph & graph)`

Given the map graphs of the U.S., Europe, and Japan, which two cities are closest in each one?

Answer

- U.S.: Champaign and Chicago (126 miles).
- Europe: Prague and Berlin (174 miles).
- Japan: Tokyo and Omiya (15 miles).

(Note traversal edges are also colored in this solution).



To test your code, run:

```
./lab_graphs weight europe
```

to test on the Europe map (you can also use “us” or “japan”) or

```
./lab_graphs weight random
```

to test on a random graph or

```
./lab_graphs weight random 8 47
```

to test on a random graph with 8 vertices and random seed 47.

`int GraphTools::findShortestPath(Graph & graph, Vertex start, Vertex end)`

What is the minimum number of layovers/train exchanges between two cities if the only flights/routes possible are represented by edges on the graph?

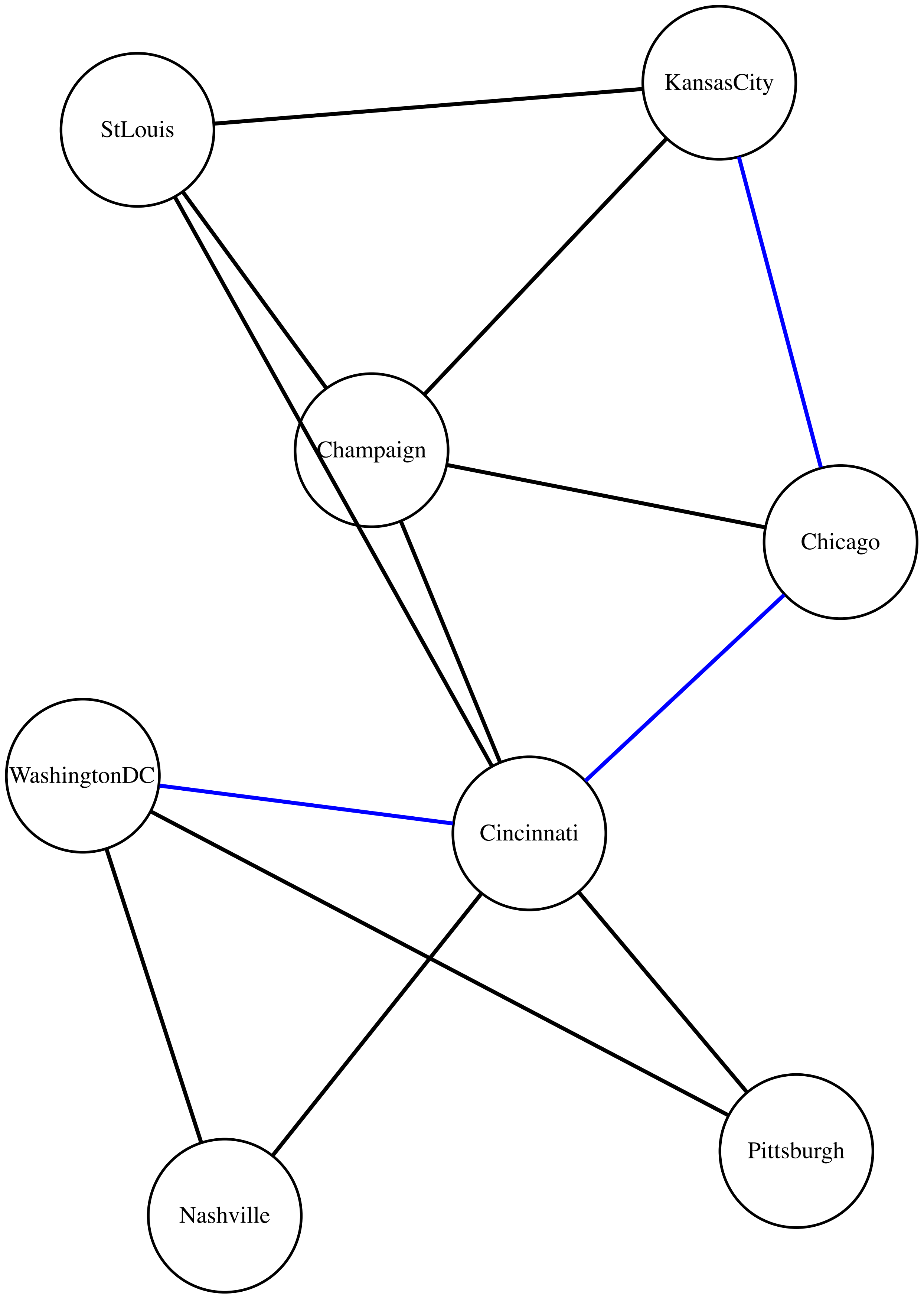
Answer

- Minimum path from KansasCity to WashingtonDC: 3 edges.
- Minimum path from London to Prague: 2 edges.
- Minimum path from Nagoya to Hitachinaka: 2 edges.

Note

Your graph may differ (say, `KansasCity->StLouis->Cincinnati->WashingtonDC`), but the minimum number of edges is the same: 3.

We will take it into consideration when grading that there may be multiple paths of minimum length.



For this task, you must return the length of the shortest path (in edges) between two vertices. You must also color the edges in the shortest path blue by labeling them `"MINPATH"`.

To test your code, run

```
./lab_graphs path europe
```

to test on the Europe map (you can also use “us” or “japan”) or

```
./lab_graphs path random
```

to test on a random graph or

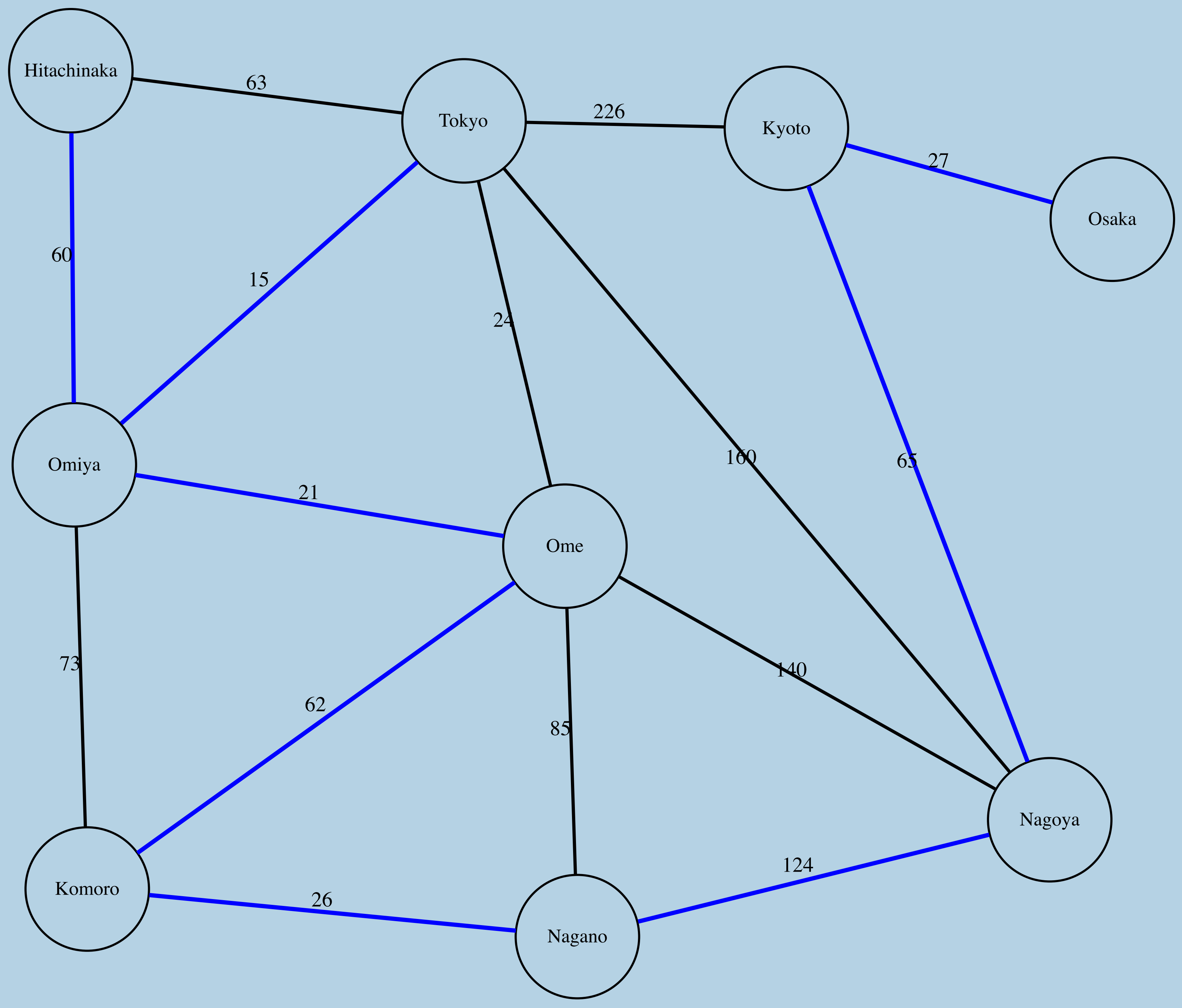
```
./lab_graphs path random 8 47
```

to test on a random graph with 8 vertices and random seed 47.

`void GraphTools::findMST(Graph & graph)`

What path can you create between all cities in each graph with the minimum total distance?

Answer



For this task, you must label the edges of the minimum spanning tree as `"MST"` in the `Graph g` using Kruskal's Algorithm.

A spanning tree connects all vertices of a graph, without creating cycles. A **minimum** spanning tree does the same thing, but the edges it uses to connect the vertices are of minimal weight. In other words, a minimal spanning tree uses the lightest edges possible to connect all the vertices in a graph.

In class we learn about two algorithms that accomplish this: Prim's Algorithm and Kruskal's Algorithm. For this lab, we will use Kruskal's algorithm, because we already have the tools necessary to implement it. Incredibly, we can find and label the minimum spanning tree in less than 40 lines of code! Let's take a look at the algorithm:

1. Get a list of all edges in the graph and sort them by increasing weight.
2. Create a disjoint sets structure where each vertex is represented by a set.
3. Traverse the list from the start (i.e., from lightest weight to heaviest).
    - Inspect the current edge. If that edge connects two vertices from different sets, union their respective sets and mark the edge as part of the MST. Otherwise there would be a cycle, so do nothing.
    - Repeat this until $n - 1$ edges have been added, where n is the number of vertices in the graph.

To test your code, run:

```
./lab_graphs kruskal europe
```
TERMINAL

to test on the Europe map (you can also use "us" or "japan") or

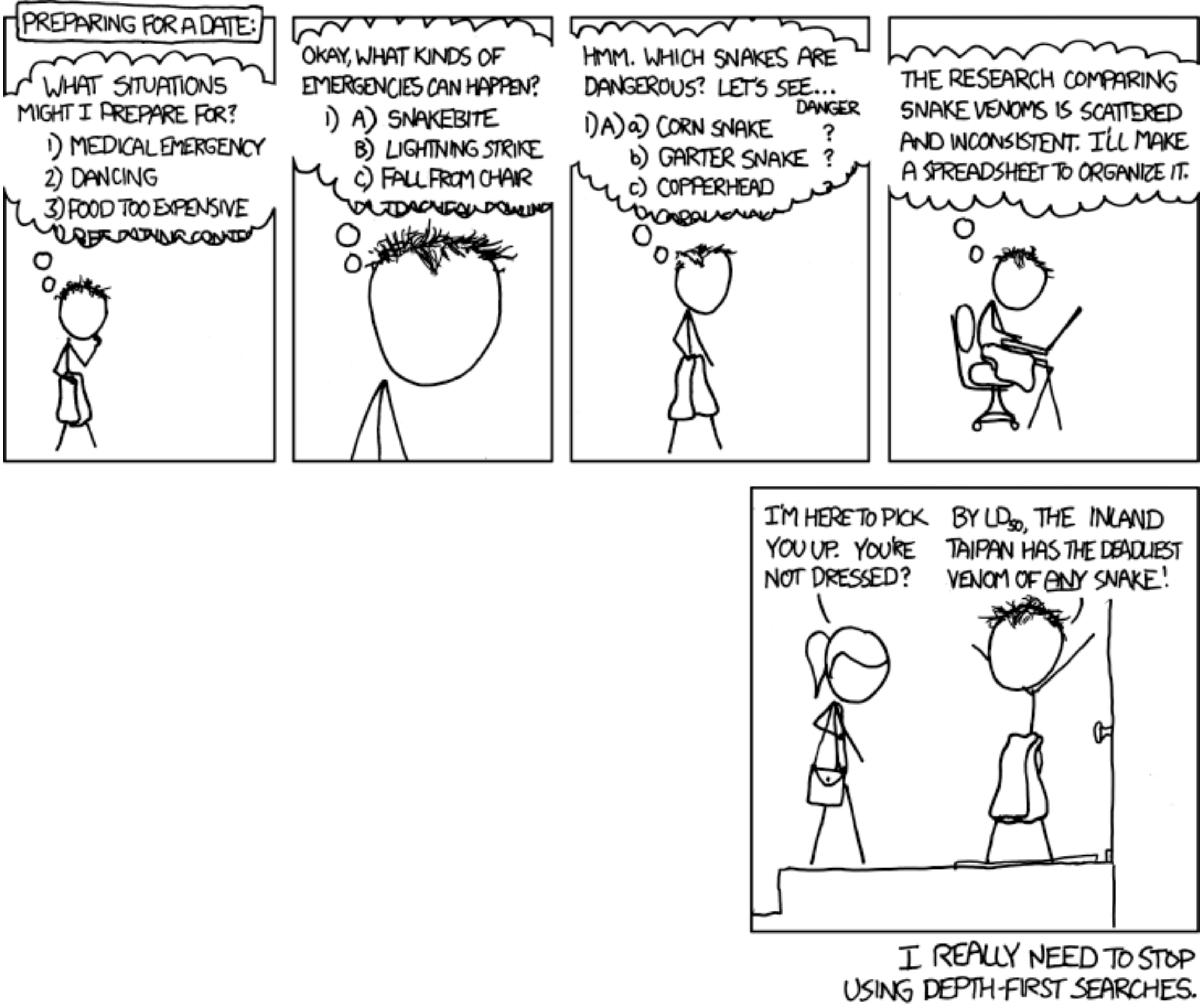
```
./lab_graphs kruskal random
```
TERMINAL

to test on a random graph or

```
./lab_graphs kruskal random 8 47
```
TERMINAL

to test on a random graph with 8 vertices and random seed 47.

Good luck!


PREPARING FOR A DATE:
WHAT SITUATIONS MIGHT I PREPARE FOR?
1) MEDICAL EMERGENCY
2) DANCING
3) FOOD TOO EXPENSIVE
OKAY, WHAT KINDS OF EMERGENCIES CAN HAPPEN?
1) A) SNAKEBITE
B) LIGHTNING STRIKE
C) FALL FROM CHAIR
HMM. WHICH SNAKES ARE DANGEROUS? LET'S SEE...
DANGER
1) A) a) CORN SNAKE ?
b) GARTER SNAKE ?
c) COPPERHEAD
THE RESEARCH COMPARING SNAKE VENOMS IS SCATTERED AND INCONSISTENT. I'LL MAKE A SPREADSHEET TO ORGANIZE IT.
I'M HERE TO PICK YOU UP. YOU'RE NOT DRESSED?
BY LD50, THE INLAND TAIPAN HAS THE DEADLIEST VENOM OF ANY SNAKE!
I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

DFS (XKCD #761)