CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman
Assignment 8
Due Friday April 10, by 3pm

A Nation Divided … by Zombies!

Admit it, we all love zombies. Maybe it's because they don't actually exist, and we don't actually have to worry about navigating life with the undead. But, imagine for a second an alternate universe where they do exist and they have attacked – creating mayhem throughout the country, knocking down communications towers and taking control of bridges and highways. One could imagine a resourceful zombie coalition making it impossible to travel between major cities, isolating human survivors in small districts around the country with no safe means of reaching other districts. The US would become a collection of small outposts, where cities within a district could be reached from within the district, and district residents would need to be careful about travel even within their district. Knowing the shortest path between cities to avoid being attacked would be paramount for survival.

What your program needs to do:

Good news: We won't be using json for this assignment. The specific cout statements that COG expects will be added to the end of this writeup.

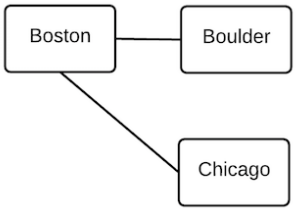
Build a graph. There is a file on Moodle called zombieCities.txt that contains the names of 10 cities and the distances between them. Cities that still have roads connecting them that aren't controlled by zombies have a positive distance in the file. Cities that have been cutoff from each other have a -1 as their distance. When the user starts the program, read in the cities and distances from the text file and build a graph where each city is a vertex, and the adjacent cities are stored in an adjacency list for each vertex.

Use a command-line argument to handle the filename.

For example, this data:

	Boston	Boulder	Chicago
Boston	0	2000	982
Boulder	2000	0	-1
Chicago	982	-1	0

would generate this graph:



The vertices in the graph are Boston, Boulder, and Chicago. The adjacent vertices for Boston are Boulder and Chicago. The adjacent vertex for Boulder is Boston, and the adjacent vertex for Chicago is Boston.

Display a menu. Once the graph is built, your program should display a menu with the following options:

1. **Print vertices**
2. **Find districts**
3. **Find shortest path**
4. **Find shortest distance**
5. **Extra credit**
6. **Quit**

Menu Items and their functionality:

1. **Print vertices.** If the user selects this option, the vertices and adjacent vertices should be displayed. The district ID should also be included in the display.

   An example of how the output should be formatted is shown here:
   1:Boston->Boulder**Chicago
   1:Boulder->Boston
   1:Chicago->Boston

   The 1 shown is the district ID. District IDs should all be initialized to -1. If you call print vertices before finding districts, your display would look like:
   -1:Boston->Boulder**Chicago
   -1:Boulder->Boston
   -1:Chicago->Boston

2. **Find districts.** If the user selects this option, you need to do a breadth-first search of the graph to determine the connected components in the graph. The connected components are the vertices that are connected, either directly or through another vertex. For example, in the Boulder, Boston, Chicago graph shown above, these three cities are all in the same connected component even though there isn't an edge connecting Chicago and Boulder. There is a path between these two cities that goes through Boston.

   In your graph, add a parameter to each vertex to store a district ID. The ID should be an integer, 1 to n, where n is the number of districts discovered in

the graph, you will not know this value ahead of time. To get the correct, expected district ID for each vertex, make sure you read in the zombieCities.txt file in order so that your vertices are set up in alphabetical order.

When assigning district IDs, start at the first vertex and find all vertices in the connected component for that vertex. This is district 1. Next, find the first vertex alphabetically that is not assigned to district 1. This vertex is the first member of district 2, and you can repeat the breadth-first search to find all vertices connected to this vertex. Repeat this process until all vertices have been assigned to a district. This may also require an additional data structure to track this information.

You do not need to print anything for this menu option. To verify that district IDs have been assigned, call print vertices again.

3. **Find shortest path.** If the user selects this option, they should be prompted for the names of two cities. Your code should first determine if they are in the same district. If the cities are in different districts, print "No safe path between cities". If the cities are in the same district, run a breadth-first search that returns the number of edges to traverse along the shortest path, and the names of the vertices along the path. For example, to go from Boulder to Chicago in the example graph, you would print:

   2, Boulder, Boston, Chicago

4. **Find shortest distance.** If the user selects this option, they should be prompted for the names of two cities, just as in the shortest path menu option. If the cities are in different districts, print "No safe path between cities". If the cities are in the same district, run Dijkstra's algorithm to find the shortest weighted path between the cities. Print the distance between the cities, and the names of the cities on the path between them. (Zombies like cities, so a longer, weighted path that avoids cities might be preferred to a shorter path that travels through more cities.) For example, to go from Boulder to Chicago in the example graph, assuming the edge weights were 50 each, you would print:

   100, Boulder, Boston, Chicago

5. **Extra credit.** Order has been restored, and it's time for a road trip. You have compiled a list of the 10 best places to visit in the Western US, given in bestPlaces.txt on Moodle. Leaving from Boulder, calculate the shortest distance between all 10 places and then returning home to Boulder. This is the horribly inefficient exhaustive search algorithm that tries all possible paths and returns the shortest one.

How to submit your assignment
To submit your work, zip all files together and submit them to COG as Assignment8.zip. If you do not get your assignment working on COG, you will have the option of a grading interview.

Appendix A – cout statements

1. Print menu

```
cout << "======Main Menu=====" << endl;
cout << "1. Print vertices" << endl;
cout << "2. Find districts" << endl;
cout << "3. Find shortest path" << endl;
cout << "4. Find shortest distance" << endl;
cout << "5. Extra credit" << endl;
cout << "6. Quit" << endl;
```

2. Print vertices

```
cout<< vertices[i].district <<":"
<<vertices[i].name<<"-->";
for each adjacent vertex:
       cout<<vertices[i].adj[j].v->name;
       if (j != vertices[i].adj.size()-1)
           cout<<"***";
```

3. Find districts

   Nothing to print.

4. Find shortest path

```
cout << "Enter a starting city:" << endl;
cout << "Enter an ending city:" << endl;
```

   One or both cities not found:
```
cout << "One or more cities doesn't exist" << endl;
```

   Cities in different districts:
```
cout << "No safe path between cities" << endl;
```

   Districts not set yet:
```
cout << "Please identify the districts before checking
distances" << endl;
```

   Algorithm successful:
```
cout << distance;
for all cities in path
```

```
    cout << "," << path[j]->name;
cout << endl;
```

5. Find shortest distance

   Same as shortest path.

6. Extra credit

```
cout << "Enter a starting city:" << endl;
cout << distance;
for all cities in path
    cout << "," << path[j]->name;
cout << endl;
```

7. Quit

```
cout << "Goodbye!" << endl;
```