CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 9, Nov 2018

Graphs

Background

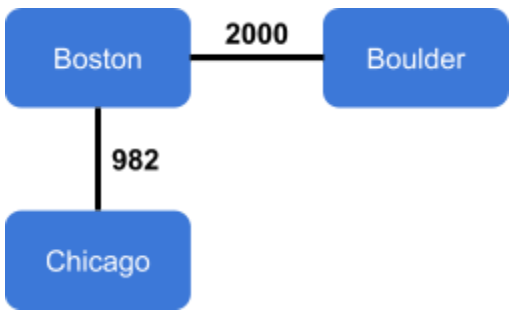
You were just a humble map programmer, but that was before the zombies attacked! Now, your skills are put to coordinating the survivors and finding routes between cities that haven't been overtaken. Some of the roads between these cities have been overrun and you will have to avoid them, which has caused the nation to be divided into multiple smaller districts.

Assignment

- You will be storing your graph data structure in a struct called Graph, which is defined in the Graph.hpp header file on Moodle. **Do not** modify this header file.
- You will also need to write a main function. We will assume your main function is written in a separate file while autograding. If you would like to write all of your code in one file, you will have to split it up when submitting it.
- Unlike previous assignments, your program will not have a menu. Instead, it should begin by reading data out of a file, where the name of this file is **passed as a command line argument**. An example file can be found on Moodle with the name *zombieCities.txt*. This file is in the following format:

```
cities,Boston,Boulder,Chicago
Boston,0,2000,982
Boulder,2000,0,-1
Chicago,982,-1,0
```

The first row and first column contain the name of each city in the graph. The rest of the values correspond to the distance between these cities - a positive value is the distance between the row and column city in miles, while the value -1 indicates that there is no path between the two cities at all. The above example corresponds to the following graph:

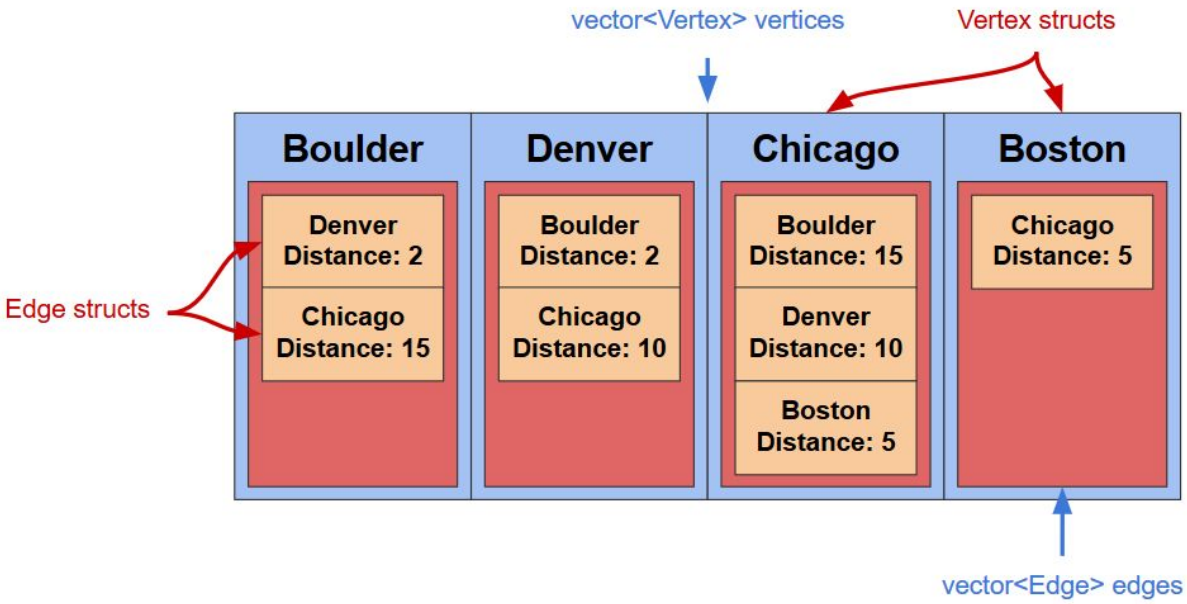


CSCI 2270 – Data Structures - Section 100

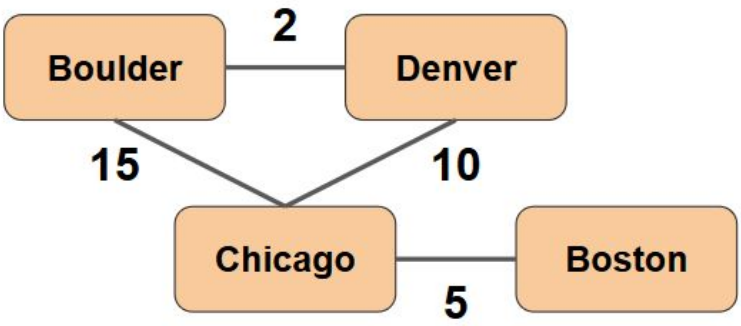
Instructor: Shayon Gupta

Assignment 9, Nov 2018

- You will store the contents of the text file in a graph data structure. You will store the graph as a vector of Vertex structs, where each one contains an adjacency list stored as a vector of Edge structs. This data structure is described in Graph.hpp and can be represented like this:



Which would represent the following graph:



Every time you read in a new edge with a distance greater than 0, you should print out the following statement to the user:

```
... Reading in <City> -- <Adjacent city> -- <Distance>
```

You can use the following print statement:

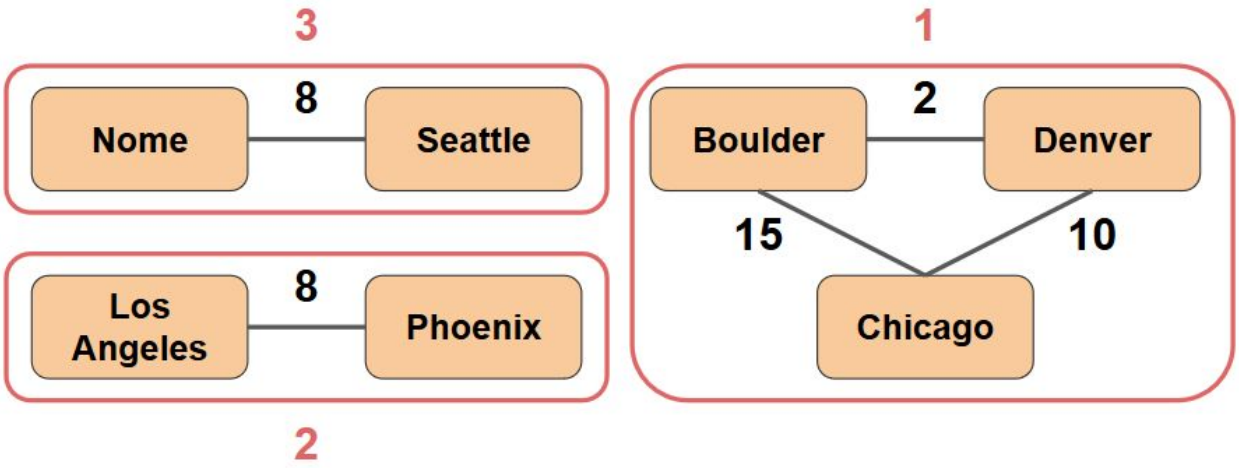
```
cout << " ... Reading in " << city << " -- "
     << connectedCity << " -- " << distance << endl;
```

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 9, Nov 2018

- After storing the graph within your data structure, you should find out all the unconnected *districts* in the graph. A *district* is a group of cities that are all connected, but none of them are connected to any other cities. For example, the graph below has three districts:



You should find these districts by iterating through every city in the graph. For each of these cities, you should perform a breadth-first search to determine every city that is part of the same district. Every time you find a new district, you should give it a district ID by setting the *districtID* variable that each Vertex struct contains to be a different number, starting at 1 and counting up from there. Examples of these IDs can also be seen above.

Hint: You can use the visited variable to tell whether any given city has already been visited by a breadth-first search.

Note: For autograding purposes, we ask that you iterate over the cities in the order that they are read in from the text file.

- After setting the district ID for each city, you should print out all the vertices and edges. It should print out each city, its district, each adjacent city, and the distance between them. Each adjacent city should be separated by three '*' characters. Use the following format, which is the expected output for the example graph above:

```
1:Boulder-->Denver (2 miles)***Chicago (15 miles)
1:Chicago-->Boulder (15 miles)***Denver (10 miles)
1:Denver-->Boulder (2 miles)***Chicago (10 miles)
2:Los Angeles-->Phoenix (8 miles)
2:Phoenix-->Los Angeles (8 miles)
3:Nome-->Seattle (12 miles)
3:Seattle-->Nome (12 miles)
```

- Finally, to test that you can write a depth-first search as well as a breadth-first search, the Moodle will have an additional question that does not tie into the rest of the assignment. It will require you to perform a depth-first search across the graph, starting at the first vertex in your vertices vector, while printing out the name of each node on a new line as you visit them.

  Note: For autograding purposes, please visit nodes in the order of their 'edges' list during your depth-first search.

- A list of the functions in the Graph class, as well as their expected behavior, is below:
  - **Constructor and destructor**
    These can be used if you want to initialize anything or free up any memory you allocate, but neither of those are necessary to complete the assignment.

  - **`void addVertex(string name)`**
    Add a vertex to the graph with the given name.

  - **`void addEdge(string v1, string v2, int distance)`**
    Add an edge going from *v1* to *v2* with a weight of *distance*.

  - **`void displayEdges()`**
    Print out the graph in the format detailed above.

  - **`void assignDistricts()`**
    Assign the districts to each node in the graph in the manner detailed above.

    Hint: Use the BFTraversalLabel *function to label each unvisited district. Also, call* setAllVerticesUnvisited *so that none of the cities are visited before your traversal.*

  - **`void printDFS()`**
    Print the names of the cities in the order they are visited during a depth-first traversal.

    Hint: It can be written recursively using the DFTraversal *helper function. Also, call* setAllVerticesUnvisited *so that none of the cities are visited before your traversal.*

  - **`void setAllVerticesUnvisited()`**
    Set the *visited* member of each vertex to be *false*. This should be called right before any traversal that uses the *visited* member.

- **`vertex *findVertex(string name)`**
  This helper function should return the address of the vertex with the given name.

- **`void BFTraversalLabel(string startingCity, int distID)`**
  This helper function should set the district ID of each city that is reachable from *startingCity* to be *distID*. It should be called by the *assignDistricts* function.

- **`void DFTraversal(vertex *v)`**
  This helper function should perform a depth-first traversal of the graph, starting from v, and print out each node as it is visited.

Submission

Submit your code on Moodle by following the directions on Assignment 9 Submit. You must submit an answer to be eligible for interview grading!