CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

Assignment 9 - Graphs

OBJECTIVES

1. **Build an undirected weighted graph**
2. **Perform Breadth First Traversal (BFT) and Depth First Traversal (DFT)**

Overview

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 (think disconnected components in a graph). You will have to build a graph with each vertex representing a city and each edge between vertices representing a route between them. The following structs will facilitate your graph.

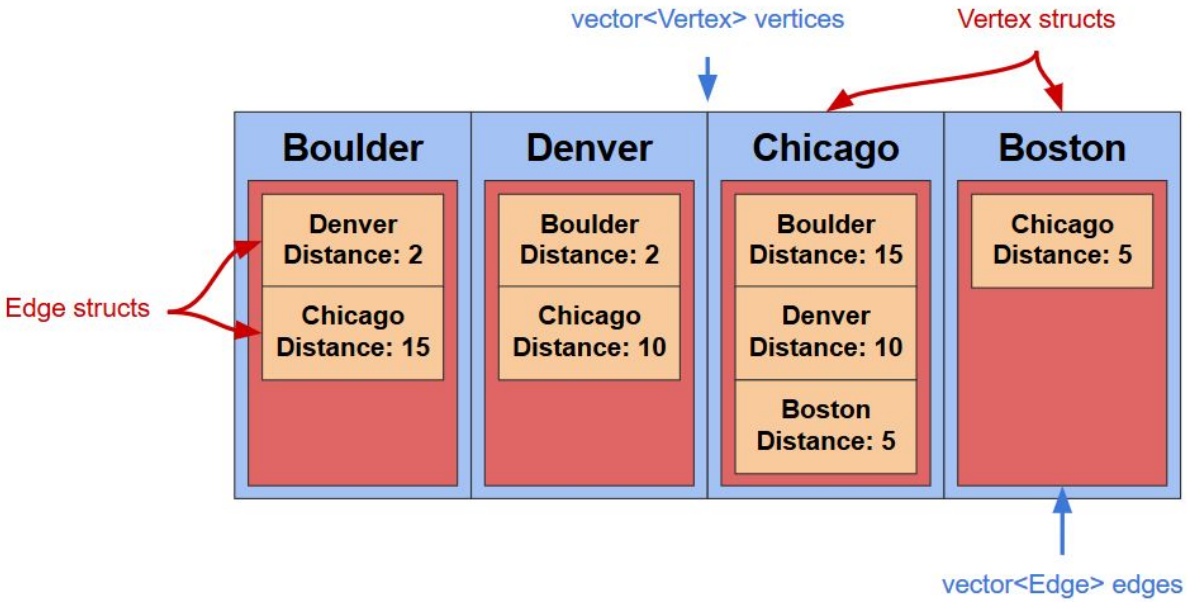
```
/* structure for edge connecting an adjacent vertex */
struct Edge
{
    vertex *v;
    int distance;
};

/*  structure for each vertex in the graph */
struct vertex
{
    std::string name;
    bool visited;
    std::vector<Edge> Edges; // stores edges to adjacent vertices
};
```

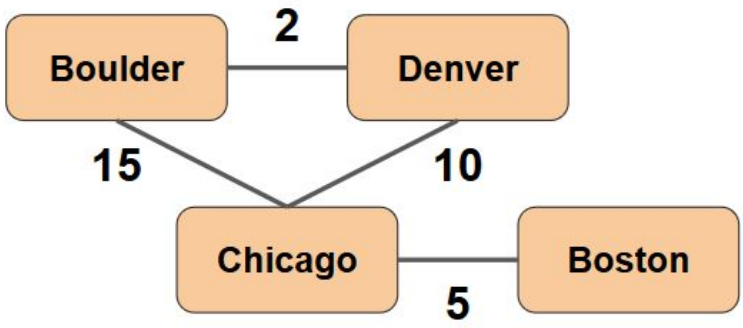
You will store the graph as a vector of *vertex* structs, where each city contains an adjacency list stored as a vector of *Edge* structs. This data structure is described in Graph.hpp and can be represented like this:

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki



Which would represent the following graph:



Graph Class

There is no need to manage dynamic memory in this assignment so you may leave your constructor/destructor empty.

void addVertex(std::string cityName)

➔ Create a new vertex with **name cityName** and push it back to your vector of vertices.

void addEdge(std::string city1, std::string city2, int distance)

➔ Establish a *single* edge from **city1** to **city2**. Create an edge with **distance** equal to **distance** and **v** equal to the address of the vertex with name **city2**. Then push back the edge to the **Edges** vector of the vertex with name **city1**.

void displayEdges()

➔ For each vertex in your **vertices** vector, print the city name and each city with which it is connected along with the distance in miles between them using the following example format. The graph below has three vertices that are all connected.

```
// Boulder has an edge to Denver with distance 2 and an
// edge to Chicago with distance 15
Boulder-->Denver (2 miles)***Chicago (15 miles)
Chicago-->Boulder (15 miles)***Denver (10 miles)
Denver-->Boulder (2 miles)***Chicago (10 miles)
```

void printDFT()

➔ Use a depth first traversal of the graph to print the names of every city beginning with the first vertex in your **vertices** vector. Use your helper functions **setAllVerticesUnvisited** and **DFT_traversal**. *Note that there may be disconnected components in the graph!*

void printBFT()

➔ Same as above with a breadth first traversal instead.

void setAllVerticesUnvisited()

➔ Loop through your vertices vector and set each vertex's **visited** field to false. This function should be called right before any traversal (BFT, DFT) that uses the *visited* member.

vertex *findVertex(std::string name)

➔ Return a pointer to the vertex with the specified **name**.

void BFT_traversal(vertex *v)

➔ Perform a breadth first traversal of the graph beginning with vertex **v**, printing the name of each vertex you visit.

```
cout << v->name << endl;
```

void DFT_traversal(vertex *v)

➔ Same as above with depth first traversal instead.

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

Driver

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 *simpleCities.txt*. This file is in the following format

```
cities,Boulder,Denver,Chicago,Boston,Austin
Boulder,0,2,15,-1,-1
Denver,2,0,10,-1,8
Chicago,15,10,0,5,-1
Boston,-1,-1,5,0,-1
Austin,-1,8,-1,-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 those cities

- A positive value represents the distance(in miles) between the row and the column city
- While the value -1 indicates that there is no path connecting the two cities.

Every time you read in a new edge with a distance greater than 0, use the following print statement:

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

After your graph is built, demonstrate your different traversal methods with the following lines of code:

```
cout << "----------------------------- " << endl
     << "Breadth First Traversal" << endl
     << "-----------------------------" << endl;
g.printBFT();

cout << "----------------------------- " << endl
     << "Depth First Traversal" << endl
     << "-----------------------------"<< endl;
g.printDFT();

cout << "----------------------------- " << endl
     << "Display Edges" << endl
     << "-----------------------------"<< endl;
```

```
g.displayEdges();
```