CSCI 310 – 02 (Fall 2019)

Programming Foundations

Programming Assignment 9

DUE: Fri, Dec 6, 11:59 PM (`turnin` time)

Specifications

Consider a number maze represented as a two dimensional array of numbers consisting of only the numbers between 0 and 9. The maze can be traversed following any orthogonal direction (*i.e.*, north, south, east and west). Considering that each cell represents a cost, then finding the minimum cost to travel the maze from one entry point to an exit point may pose you a reasonable challenge, although there must be a graph algorithm we have learned that we can apply to solve this problem. Your task is to find the minimum cost value to go from the *top–left* corner to the *bottom–right* corner of a given number maze

0	3	1	2	9
7	3	4	9	9
1	7	5	5	3
2	3	4	2	5

Figure 1: Sample number maze.

of size $N \times M$, where N denotes the number of rows and M denotes the number of columns. Additionally, $1 \leq N, M \leq 999$. Note that the solution for the given example is 24.

Although this problem can be solved without the use of any classes, you will be required to modify your `Graph struct` from our lab sessions and transform it into a `template`–ized `Graph class` (you may want to refer to Carrano's `GraphInterface` abstract class for ideas. You also need to define a `WeightedVertex class` that will be the type of vertex for your `Graph class`.

The driver for the program will be the following:

```
#include <iostream>

using namespace std;

#include "WeightedVertex.h"  // Your weighted vertex class
#include "Graph.h"           // Your graph class
#include "numMaze.h"         // For readMaze() and getSSSP()

int main()
{
    Graph< WeightedVertex > G("Number Maze");
    readMaze(G);
    // cout << G << endl; // For debugging only

    set<WeightedVertex> V=G.getVertices();
    map<WeightedVertex,int> distance=getSSSP( G , *(V.begin()) );

    // cout << "Distances from " << *(V.begin()) << ": ";
    // for( auto v : V )  // For debugging only
    //     cout << distance[v] << " ";
    // cout << endl;

    cout << distance[*(V.rbegin())] << endl;

    return 0;
}
```

Input

All input comes from standard input, so using `cin` should be sufficient. The first number indicates the number of rows, N, and the second number indicates the number of columns, M. The next $N \times M$ numbers (between 0 and 9) indicate the values at each cell of the maze. Each value is separated by whitespace. Assume that all input is correct, so need for error–checking.

Sample Input

Here is the input corresponding to the number maze given in Figure 1:

```
4 5
0 3 1 2 9
7 3 4 9 9
1 7 5 5 3
2 3 4 2 5
```

And here is another simple number maze:

```
3 3
1 2 3
4 5 6
7 8 9
```

Output

The output consists of a single number: the cost of the minimum cost path from the *top–left* corner to the *bottom–right* corner of the given maze. Altough there may be more than one path from the *top–left* corner to the *bottom–right* corner of a maze, the minimum cost should be the same. This is why we are not interested in the path, as there could be several solutions that generate the same minimum cost. While you are debugging your solution, you may want to devise a way to generate the path to check your answer.

All output goes to standard output.

Sample Output

For the sample number maze given in Figure 1, your program should produce the following output:

```
24
```

Can you determine the path? It is 0–3–1–4–5–4–2–5 for a total cost of 24.

For the second simple maze (and assuming you uncommented the use of the overloaded `operator<<` in line #14 and the loop in lines #18 through #21 of the provided function `main()` for debugging purposes, then the output may look something like this:

```
Number Maze has 9 vertices and 24 edges:
  V={1,2,3,4,5,6,7,8,9}
  E={(1,2),(1,4),(2,1),(2,3),(2,5),(3,2),(3,6),(4,1),(4,5),(4,7),(5,2),(5,4),(5,6),(5,8),
     (6,3),(6,5),(6,9),(7,4),(7,8),(8,5),(8,7),(8,9),(9,6),(9,8)}
Distances from 1: 1 3 6 5 8 12 12 16 21
21
```

Can you determine the path? It is 1–2–3–6–9 for a total cost of 21.

Additional Requirements

You are to supply a file named `numMaze.h` that implements the following functions:

1. `void readMaze( Graph<WeightedVertex> G )` – (see line #14) reads in the maze from standard input and uses a `Graph` object to represent it. The vertices of type `WeightedVertex` are read first, then the vertices are connected based on the maze configuration.

2. `map<WeightedVertex,int> getSSSP( Graph<WeightedVertex> G , const WeightedVertex src )` – (see line #19) performs a modified version of Dijkstra's SSSP algorithm so that it works on a *vertex–weighted graph* instead of the standard algorithm that works for *edge–weighted graphs*. Note that the total weight of the path is the sum of the weights of all the vertices along that path. The shortest distances from the indicated source to all the other vertices in the graph are returned as a `map` object.

Note that at least one member function is implied in the function `main()` given above. Make sure you have a good understanding of the design of your Graph class before you begin implementation.

Deliverables

Your submission will consist of the following files, submitted using the Department of Computer Science's `turnin` facility:

- `WeightedVertex.h` – header file for the required `WeightedVertex` class
- `WeightedVertex.cpp` – implementation file for the required `WeightedVertex` class
- `Graph.h` – header file for the required `Graph` class or struct
- `Graph.cpp` – implementation file for the required `Graph` class or struct
- `numMaze.h` – header file containing the implementations for the functions `readMaze()` and `getSSSP()` described in the Specifications and Additional Requirements sections.

Note that the following files are available from our *CSCI 310 Lab Files folder* and on `turnin`:

- `numMaze.cpp`

Sample test files have also been included in our *CSCI 310 Lab Files folder* for your reference.

We want you to develop good code documentation habits. Source code solutions submitted without any meaningful documentation will receive a total score of zero (0). You may refer to the *Google C++ Style Guide* section on source code comments as a guide.

Be sure to also review and adhere to the **Coding Standards** for this course.