Esau Hervert
eah170630

CSE 5343
July 24, 2019

Assignment 5

Problem

Implement Dijkstra's algorithm to find the SPT. Make a graph with at least 10 nodes and 15 weighted edges. Print the graph as a adjancency Matrix. After you run the program to find SPT, print the SPT also as an adjancency matrix.

Submit the code and submit the screenshots.

Graph

Here is the following graph that I found the SPT for.

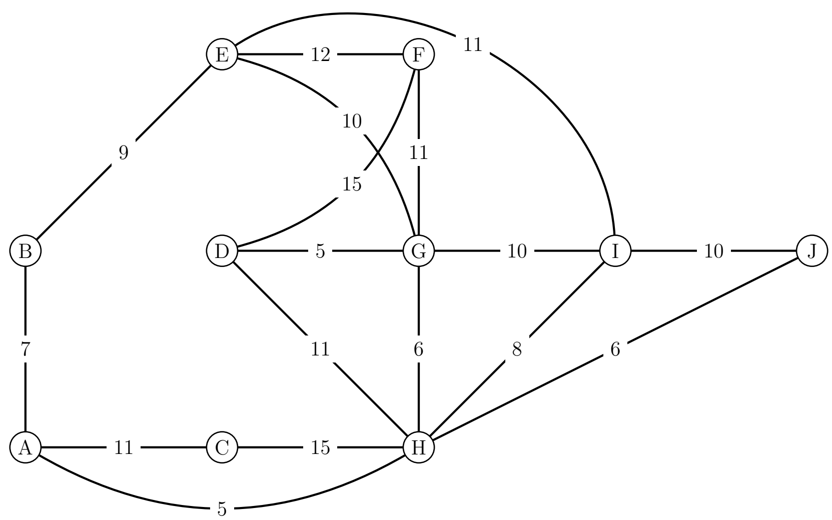


Figure 1: Graph

The Adjancency Matrix is as follows:

$$\begin{bmatrix}
0 & 7 & 11 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\
7 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 15 & 5 & 11 & 0 & 0 \\
0 & 9 & 0 & 0 & 0 & 12 & 10 & 0 & 11 & 0 \\
0 & 0 & 0 & 15 & 12 & 0 & 11 & 0 & 0 & 0 \\
0 & 0 & 0 & 5 & 10 & 11 & 0 & 6 & 10 & 0 \\
5 & 0 & 15 & 11 & 0 & 0 & 6 & 0 & 8 & 6 \\
0 & 0 & 0 & 0 & 11 & 0 & 10 & 8 & 0 & 10 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 10 & 0
\end{bmatrix}$$

Shortest Path Tree

Doing Dijkstra's Algorithm by hand, I found that the SPT from vertex A is as follows:

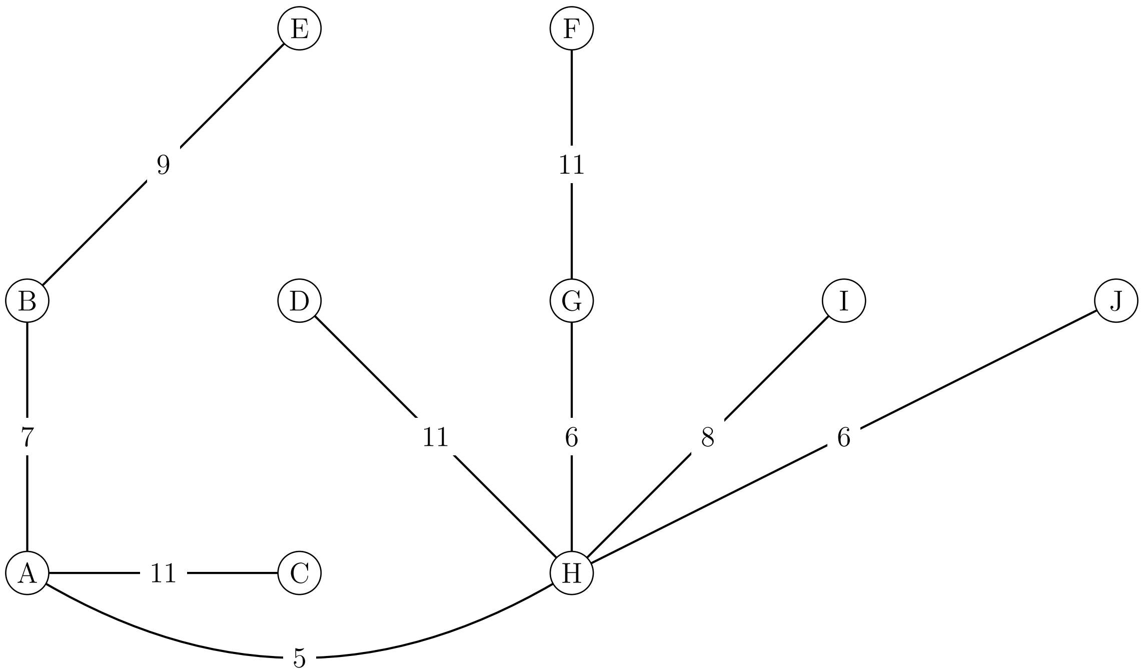


Figure 2: Graph

The Adjancency Matrix is as follows:

$$\begin{bmatrix} 0 & 7 & 11 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 7 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 11 & 0 & 6 & 0 & 0 \\ 5 & 0 & 0 & 11 & 0 & 0 & 6 & 0 & 8 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 \end{bmatrix}$$

Code

Code 1: SPT_Functions.h

```
#ifndef SPT_Functions
#define SPT_Functions

#include <iostream>
#include <queue>

using namespace std;

// Due to the fact that passing a queue through
// a function does not change the queue, we need
// to declare a structure so that we can modify
// the queue and return the modified queue.
struct Return {
        queue <char> Queue;
        char character;
};

// We will use this so that we can kinda have a
// "Global variable" but it be dependent on the
// Graph given.
int maxvalue(int A[10][10]) {
        int max = 0;

        for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                        if (A[i][j] > max) {
                                max = A[i][j];
                        }
                }
        }
        return max;
}

// Will be used to get the index of a character.
int getIndex(char c, char * vertices) {
        for (int i = 0; i < 10; i++) {
                if (vertices[i] == c) {
                        return i;
                }
        }
        return -1;
}

// This function will take a queue with some distances associated
// with it and return a smaller queue with the smallest element
// removed.
```

```
// I choose this over a priority queue do to the approach I took
// in implementing Dijkstra's Algorithm.
Return emptymin(queue <char> Q, int *distance, char *vertices) {

        // So we can dequeue but still hold values.
        queue <char> temp;

        // Initialize the first element to be the smallest.
        int minindex = getIndex(Q.front(), vertices);
        int min = distance[minindex];
        char c;
        int i = 0;

        // Traverse the queue to find the smallest value.
        while (!Q.empty()) {
                c = Q.front();
                i = getIndex(c, vertices);
                Q.pop();
                temp.push(c);


                if (distance[i] < min) {
                        min = distance[i];
                        minindex = i;
                }
        }

        // vertex of the smallest value.
        char minchar = vertices[minindex];

        // Restore the queue without the smallest value.
        while (!temp.empty()) {
                c = temp.front();
                temp.pop();
                if (c == minchar) {
                        continue;
                }
                Q.push(c);

        }

        // Return the vertex and the modified queue.
        Return R;
        R.Queue = Q;
        R.character = minchar;
        return R;
}

// This function will determine whether or not a vertex is
// incident to another using the Adjacency Matrix.
```

```
bool Incident(int A[10][10], int v, int n) {
        if (A[v][n] == 0) { return false; }
        else { return true; }
}

// Here is were we do the bulk of the work.
void DijkstraSPT(int A[10][10], int s, char* vertices) {

        // This queue will be used to travel through the graph.
        // We will update based on distance.
        queue <char> Q;

        // This array will contain the parent of a given vertex.
        int parent[10] = { -1,-1,-1,-1,-1,-1,-1,-1,-1,-1 };

        // This will be the distances. Here -1 represents infinity.
        int distance[10] = { -1,-1,-1,-1,-1,-1,-1,-1,-1,-1 };

        // Some local variables.
        int dist = 0;
        char u;
        int uindex;
        int max = maxvalue(A);
        int min;
        int minindex;
        Return R;

        // Start by adding the initial vertex.
        Q.push(vertices[s]);

        // Make the distance at the starting node to be 0.
        distance[s] = 0;

        // Go through the graph until everything has been checked.
        while (!Q.empty()) {
                // Take out the smallest element in the queue
                // and modify the queue.
                R = emptymin(Q, distance, vertices);
                u = R.character;
                Q = R.Queue;
                uindex = getIndex(u, vertices);

                // Some initialized values.
                min = max + 1;
                minindex = -1;

                // Here will will visit the neighbouring vertices of the
                // current node and update the distances accordingly.
                for (int n = 0; n < 10; n++) {
                        if (Incident(A, uindex, n)) {
```

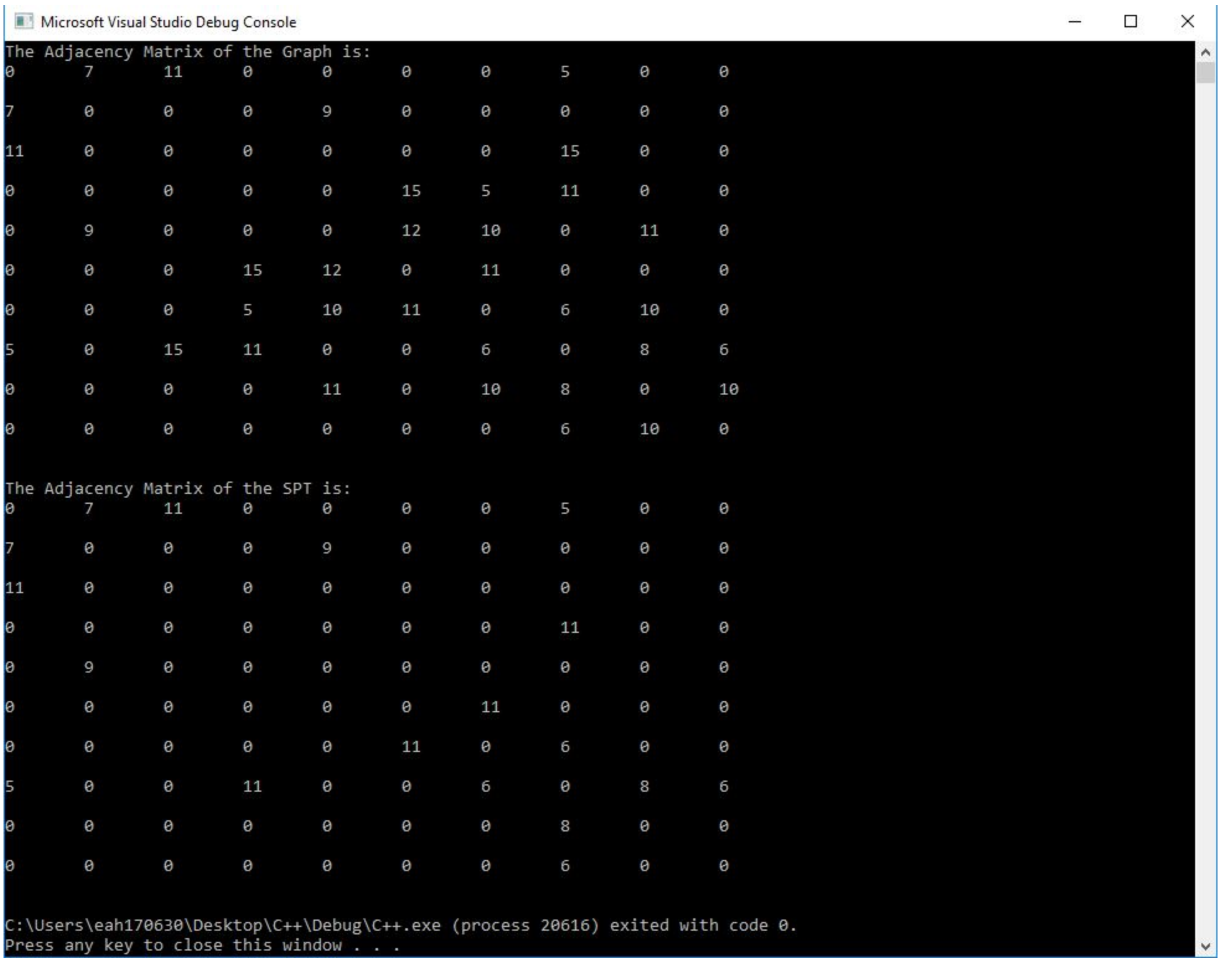
```
                                dist = distance[uindex] + A[uindex][n];
                                // If the new distance is smaller, update it.
                                if (dist < distance[n] || distance[n] < 0) {
                                        distance[n] = dist;
                                        parent[n] = uindex; // Update parent.
                                        Q.push(vertices[n]);
                                }
                        }
                }
        }

        // Using the parent array, we modify the Adjacency Matrix
        // to now be the Adjacency Matrix of the SPT.
        for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                        if (parent[j] == i || parent[i] == j) {
                                continue;
                        }
                        else {
                                A[i][j] = 0;
                        }
                }
        }
}

// This function will print the Adjacency Matrix of a graph.
void print(int A[10][10]) {
        for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                        cout << A[i][j] << "    ";
                }
                cout << endl << endl;
        }
}
#endif SPT_Functions
```

Code 2: Assignment_5_SPT.cpp

```
#include "SPT_Functions.h"

#include <iostream>
#include <queue>

using namespace std;

int main() {

        // Here is the Adjacency Matrix of the graph.
        int Graph[10][10] = {
        {0,7,11,0,0,0,0,5,0,0},
        {7,0,0,0,9,0,0,0,0,0},
        {11,0,0,0,0,0,0,15,0,0},
        {0,0,0,0,0,15,5,11,0,0},
        {0,9,0,0,0,12,10,0,11,0},
        {0,0,0,15,12,0,11,0,0,0},
        {0,0,0,5,10,11,0,6,10,0},
        {5,0,15,11,0,0,6,0,8,6},
        {0,0,0,0,11,0,10,8,0,10},
        {0,0,0,0,0,0,0,6,10,0} };

        cout << "The Adjacency Matrix of the Graph is: " << endl;

        print(Graph); // Print the Adjacency Matrix.

        char Vertices[10] = { 'A','B','C','D','E','F','G','H','I','J' };

        cout << endl;

        DijkstraSPT(Graph, 0, Vertices);

        cout << "The Adjacency Matrix of the SPT is: " << endl;

        print(Graph);


}
```

Results

The following is the run I did. The first matrix is the Adjacency Matrix of the graph and the second matrix is the Adjacency Matrix of the SPT.



```
Microsoft Visual Studio Debug Console
The Adjacency Matrix of the Graph is:
0       7       11      0       0       0       0       5       0       0

7       0       0       0       9       0       0       0       0       0

11      0       0       0       0       0       0       15      0       0

0       0       0       0       0       15      5       11      0       0

0       9       0       0       0       12      10      0       11      0

0       0       0       15      12      0       11      0       0       0

0       0       0       5       10      11      0       6       10      0

5       0       15      11      0       0       6       0       8       6

0       0       0       0       11      0       10      8       0       10

0       0       0       0       0       0       0       6       10      0


The Adjacency Matrix of the SPT is:
0       7       11      0       0       0       0       5       0       0

7       0       0       0       9       0       0       0       0       0

11      0       0       0       0       0       0       0       0       0

0       0       0       0       0       0       0       11      0       0

0       9       0       0       0       0       0       0       0       0

0       0       0       0       0       0       11      0       0       0

0       0       0       0       0       11      0       6       0       0

5       0       0       11      0       0       6       0       8       6

0       0       0       0       0       0       0       8       0       0

0       0       0       0       0       0       0       6       0       0


C:\Users\eah170630\Desktop\C++\Debug\C++.exe (process 20616) exited with code 0.
Press any key to close this window . . .
```

Figure 3: Results