Project Report

Authors: Shanmukha Praveen Madasu, Narendra Kumar Vankayala

Shortest Path Algorithm:

Description:

To find the shortest paths from given source vertex to all other vertices in the given graph, we used Dijkstra's Algorithm. Here we generated a Shortest Path Tree with given source as root.

Below are the detailed steps used in our shortest path algorithm

1) Create a cloud which is a min-heap based priority queue. Using cloud as priority queue helps us to reduce the time complexity of the algorithm.

2) priority queue used here, doesn't have option to reduce already present node's key. So we are inserting duplicate key with reduced weight, this way while picking it will pick the minimum distance vertex.

3) used distance vector, which is used to store all vertexes distances which may subject to change. Later these may be revisited if there is any lesser distance than the available one.

4) parent vector is used to keep track of the current vertex's parent, used to print path.

5) Explored vector is used to keep track of already explored vertexes.

6) Initialize every vertex distance to infinite.

7) push the source node into the cloud and store its distance as '0'.

8) Traverse until the cloud is empty. We keep on adding to the cloud If we find any new adjacent vertex.

9) pick the top most node, as it is min-heap based priority queue, the least distance vertex will always be at the top of the heap.

10) Based on the direction of the graph (directed/undirected), divided algorithm into 2 parts. One will pick the adjacent edges from their corresponding vertex list.

11) Get all adjacent edges from the picked vertex. If this adjacent vertex is already explored just continue the loop, as we don't want to make any changes for the already explored vertex.

12) If the adjacent vertex is not yet explored, recalculate its distance if possible with any of the optimized path. Like, if the adjacent vertex distance is more than the combined distance of its parent's distance and its distance, then replace the current distance with the calculated distance value.

13) After calculating and assigning, push the vertex into the cloud with its new distance value.

14) If all its adjacent vertexes have been explored, then mark it as explored such that later we will not revisit that again.

15) All the above-mentioned steps will be same for the directed graph as well.

Data structures used:

1) `priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> cloud`

   This is a min-heap based priority queue.

2) `vector<int> dist (vertices, INT_MAX)`

   Dist is a vector used to store vertex distances.

3) `vector<int> parent (vertices, INT_MAX)`

   Parent is a vector used to store parent information

4) `vector<bool> explored (vertices, false)`

   Explored is a vector used to store vertexes which were explored

5) `list<pair<int, int>> *unDirectedEdges`

   unDirectedEdges is a list, used to store undirected edges information

6) `list<pair<int, int>> *directedEdges`

   directedEdges is a list, used to store directed edges information

Runtime:

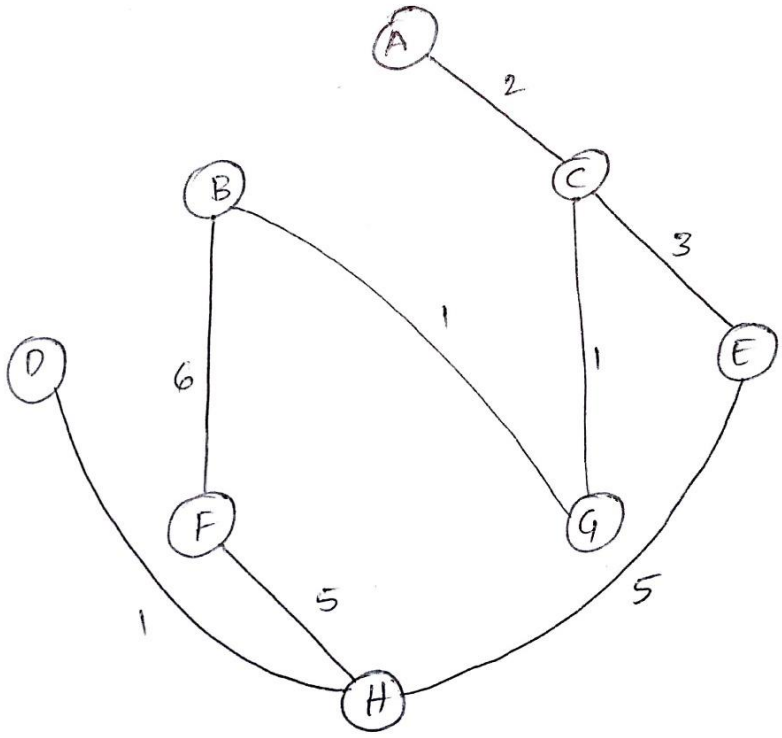
O(ElogV)

E – Number of Edges

V – Number of Vertices

Input:

Undirected Graph:

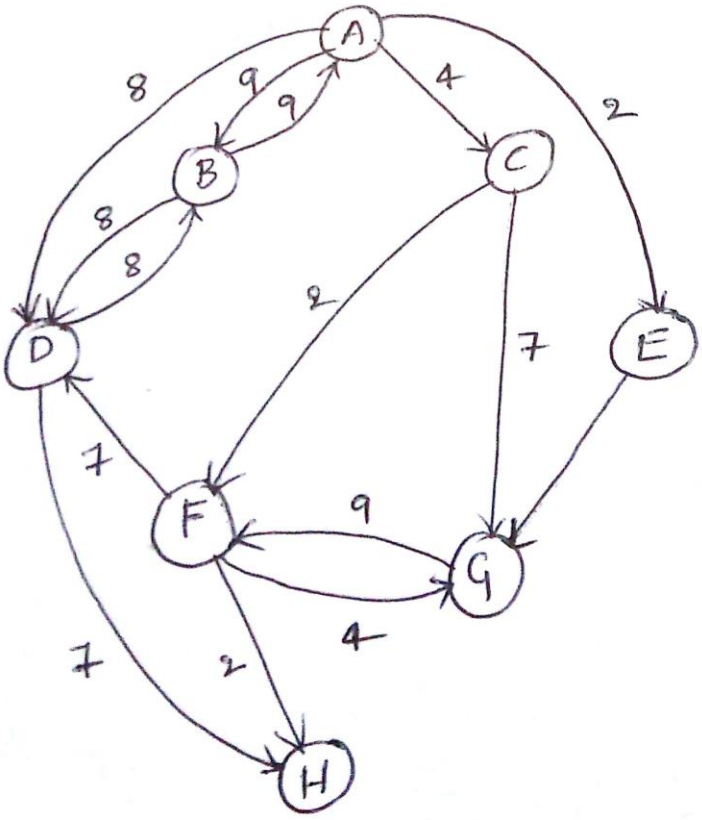



```
8 16 U
A C 2
B F 6
B G 1
C A 2
C E 3
C G 1
D H 1
E C 3
E H 5
F B 6
F H 5
G B 1
G C 1
H D 1
H E 5
H F 5
C
```

Output:

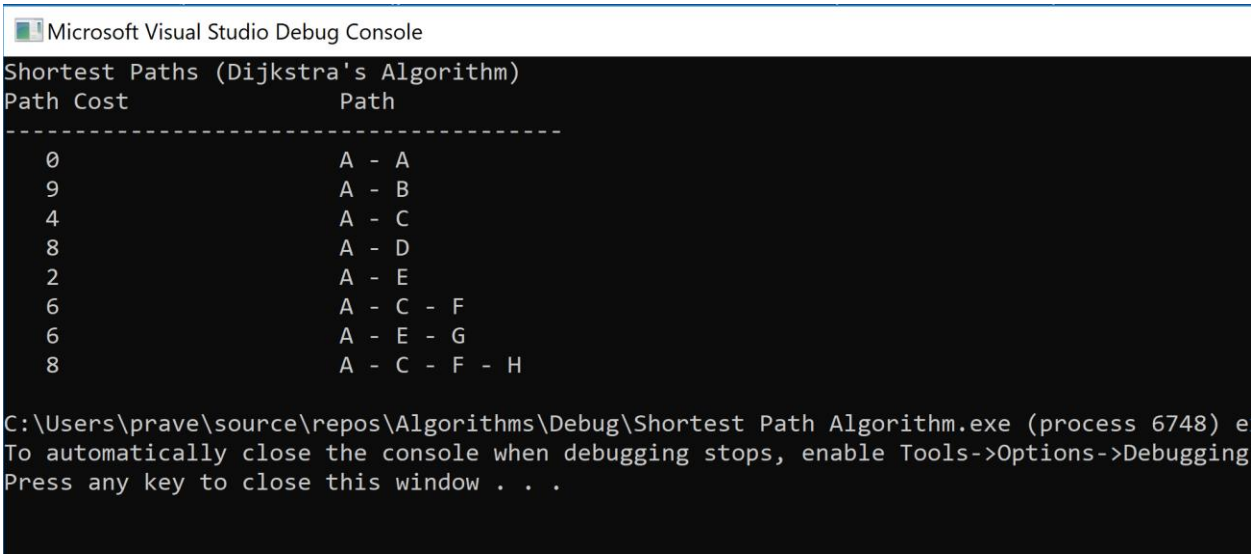
```
Microsoft Visual Studio Debug Console
src: C

Shortest Paths (Dijkstra's Algorithm)
Path Cost               Path
----------------------------------------
   2                    C - A
   2                    C - G - B
   0                    C - C
   9                    C - E - H - D
   3                    C - E
   8                    C - G - B - F
   1                    C - G
   8                    C - E - H
```

Directed Graph:



```
8 15 D
A B 9
A C 4
A D 8
A E 2
B A 9
B D 8
C F 2
C G 7
D B 8
D H 3
E G 4
F D 7
F G 4
F H 2
G F 9
A
```

Output:



Instructions to run this program:

Give the input information like no of vertices & edges information in the input file present in the same directory of the source file. And execute the source code in the visual studio IDE. If u want to execute in any other C++ compiler, just remove the first line

```
#include "pch.h"
```

this line is specific to the visual studio. Copy the sample input provided into the input text file present in the same source directory and execute.

Minimum Spanning Tree Algorithm:

Description:

Minimum spanning tree is a subset of edges of a connected, undirected, weight graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. We used prim's algorithm to find minimum spanning tree.

Below are the detailed steps used in our minimum spanning tree algorithm

1) Create a cloud which is a min-heap based priority queue. Using cloud as priority queue helps us to reduce the time complexity of the algorithm.

2) priority queue used here, doesn't have option to reduce already present node's key. So we are inserting duplicate key with reduced weight, this way while picking it will pick the minimum distance vertex.

3) used distance vector, which is used to store all vertexes distances which may subject to change. Later these may be revisited if there is any lesser distance than the available one.

4) parent vector is used to keep track of the current vertex's parent, used to print path.

5) Explored vector is used to keep track of already explored vertexes.

6) Total cost variable stores the total minimum spanning tree cost.

7) Initialize every vertex distance to infinite.

8) push the source node into the cloud and store its distance as '0'.

9) Traverse until the cloud is empty. We keep on adding to the cloud If we find any new adjacent vertex.

10) pick the top most node, as it is min-heap based priority queue, the least distance vertex will always be at the top of the heap.

11) Based on the direction of the graph (directed/undirected), divided algorithm into 2 parts. One will pick the adjacent edges from their corresponding vertex list.

12) Get all adjacent edges from the picked vertex. If this adjacent vertex is already explored just continue the loop, as we don't want to make any changes for the already explored vertex.

13) If the adjacent vertex is not yet explored, recalculate its distance if possible with any of the optimized path. Like, if the adjacent vertex distance is more than its distance, then replace the current distance with the new distance value.

14) After calculating and assigning, push the vertex into the cloud with its new distance value.

15) If all its adjacent vertexes have been explored, then mark it as explored such that later we will not revisit that again.

Data structures used:

1) `priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> cloud`

   This is a min-heap based priority queue.

2) `vector<int> dist (vertices, INT_MAX)`

   Dist is a vector used to store vertex distances.

3) `vector<int> parent (vertices, INT_MAX)`

   Parent is a vector used to store parent information

4) `vector<bool> explored (vertices, false)`

   Explored is a vector used to store vertexes which were explored

5) `list<pair<int, int>> *unDirectedEdges`

   unDirectedEdges is a list, used to store undirected edges information

6) `list<pair<int, int>> *directedEdges`

   directedEdges is a list, used to store directed edges information

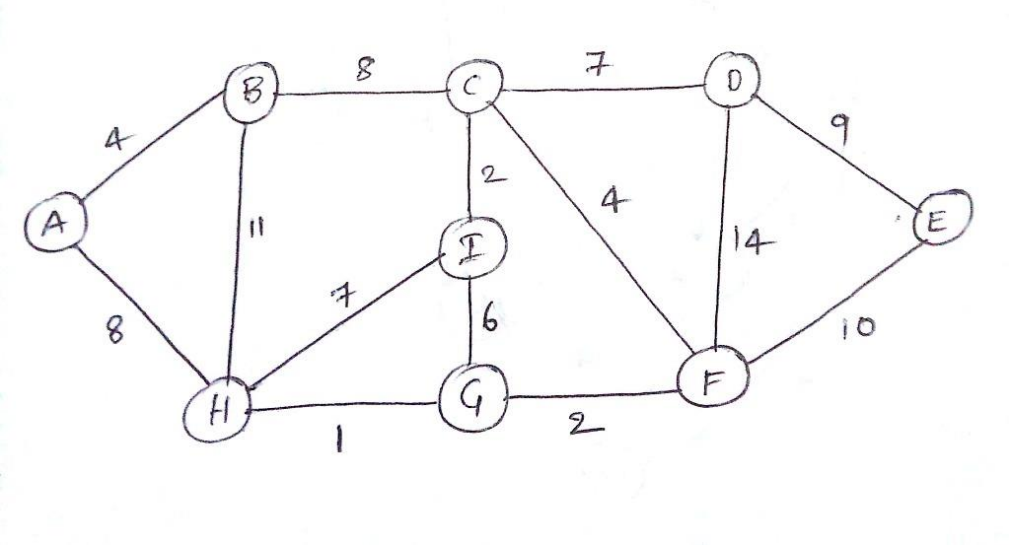
Runtime:

O(ElogV)

E – Number of Edges

V – Number of Vertices

Input:




```
9 14 U
A B 4
A H 8
B C 8
B H 11
C D 7
C I 2
C F 4
D E 9
D F 14
E F 10
F G 2
G H 1
G I 6
H I 7
A
```

Output:

```
Prims Algorithm
Edges        Distances
---------------------
A - A           0
A - B           4
B - C           8
C - D           7
D - E           9
C - F           4
F - G           2
G - H           1
C - I           2
Total Cost: 37
```

Instructions to run this program:

Give the input information like no of vertices & edges information in the input file present in the same directory of the source file. And execute the source code in the visual studio IDE. If u want to execute in any other C++ compiler, just remove the first line

```
#include "pch.h"
```

this line is specific to the visual studio. Copy the sample input provided into the input text file present in the same source directory and execute.