**Representation and important applications**

**of graph theory**

**Third Year- Computer Science**

**Spring Semester**

**Submitted to:**

Dr. Mostafa Reda El-Tantawi

**Team members:**

|  |  |
| --- | --- |
| **Name** | **ID** |
| Ahmed Sayed Mansour | 20170022 |
| Ashraf Samir Ali | 20170053 |
| Abdelraman Bahig | 20170143 |
| Atef Magdy | 20170136 |
| Hatem Sayed | 20170084 |
| Ahmed Sayed Ebrahim | 20170021 |
| Mohamed Mostafa | 20170257 |

* **Minimum Spanning Tree problem:-**

Abstraction:

Minimum Spanning tree: Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree

Analysis:

we compared between The two algorithms which are used in minimum spanning tree and find that Both **Prim’s and Kruskal’s algorithm** finds the Minimum Spanning Tree and follow the Greedy approach of problem-solving, but there are few **major differences between them**.

|  |  |
| --- | --- |
| PRIM’S ALGORITHM | KRUSKAL’S ALGORITHM |
| It starts to build the Minimum Spanning Tree from any vertex in the graph. | It starts to build the Minimum Spanning Tree from the vertex carrying minimum weight in the graph. |
| It traverses one node more than one time to get the minimum distance. | It traverses one node only once. |
| Prim’s algorithm has a time complexity of O(V^2), V being the number of vertices. | Kruskal’s algorithm’s time complexity is O(logV), V being the number of vertices. |

So, we chose Kruskal Algorithm Based on his Time Complexity

Using Kruskal Algorithm:

Kruskal Implementation Pseudocode

KRUSKAL(G):

A = ∅

For each vertex v ∈ G.V:

MAKE-SET(v)

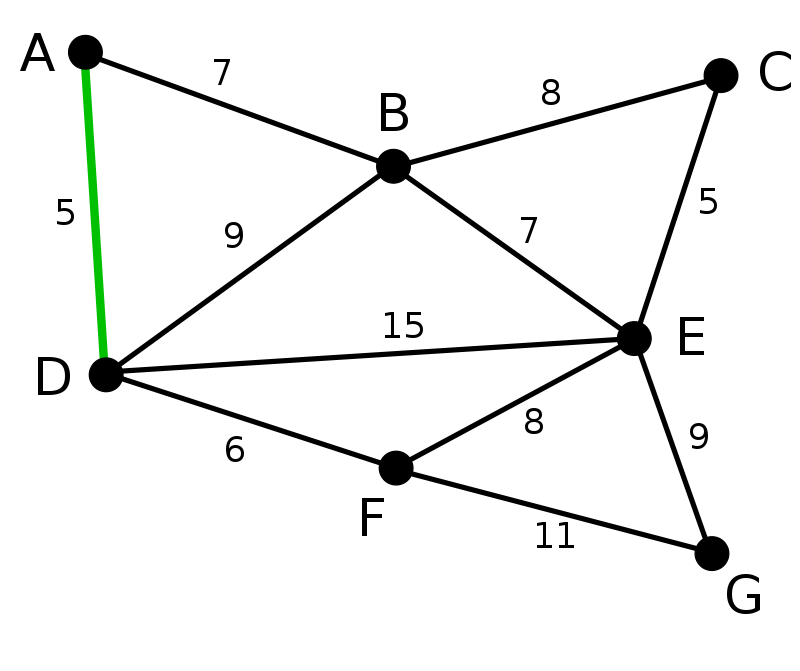
For each edge (u, v) ∈ G.E ordered by increasing order by weight (u, v):

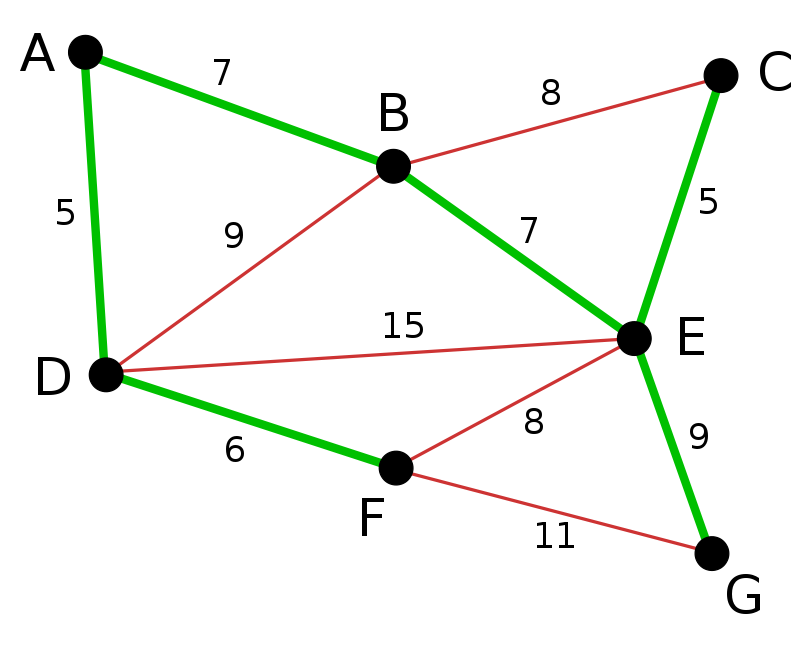
if FIND-SET(u) ≠ FIND-SET(v):

A = A ∪ {(u, v)}

UNION (u, v)

return A

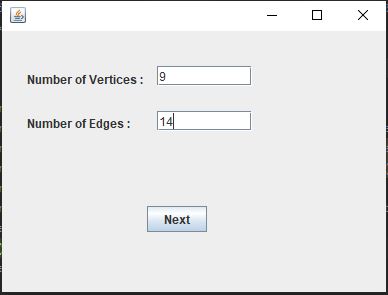




The Program (Run)

Input:

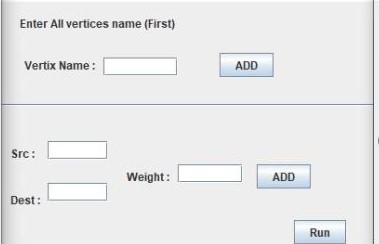
At the first should enter the number of Vertices and number of Edges you want (e.g. 4, 5)

****

Then Name your Vertices (e.g. 0, 1, 2, 3)



After that you can enter the source edge, destination edge and Weight (e.g. from 0 to 1, weight is 10). Do this as many times as the number of the Edges.



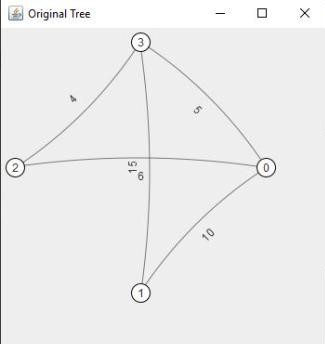
Note:

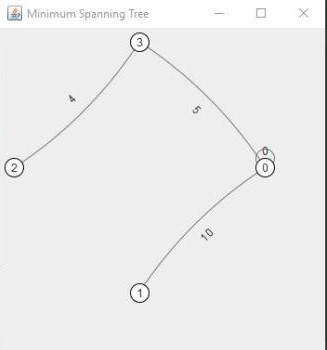
**Must fill the Vertices names with the number of Vertices you entered. You can’t enter the Edges unless you enter all Vertices name.**

Output:

Two Screens:

|  |  |
| --- | --- |
| Frist Screen  Original Graph | Second Screen  Minimum Spanning Tree |





Conclusion:

- Minimum Spanning tree used to reduce cost, time, weight etc. By finding the shortest path between any two nodes.

- Kruskal’s algorithm is an “Edge based algorithm “.

- Kruskal’s algorithm is a good method to find The Minimum Spanning Tree according to its Time Complexity O(logV).

References:

<https://www.ics.uci.edu/~eppstein/161/960206.html>

<https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/>

<https://www.javatpoint.com/kruskals-minimum-spanning-tree-algorithm>

* **Graph Representation :-**

**Abstraction:**

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.

Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.

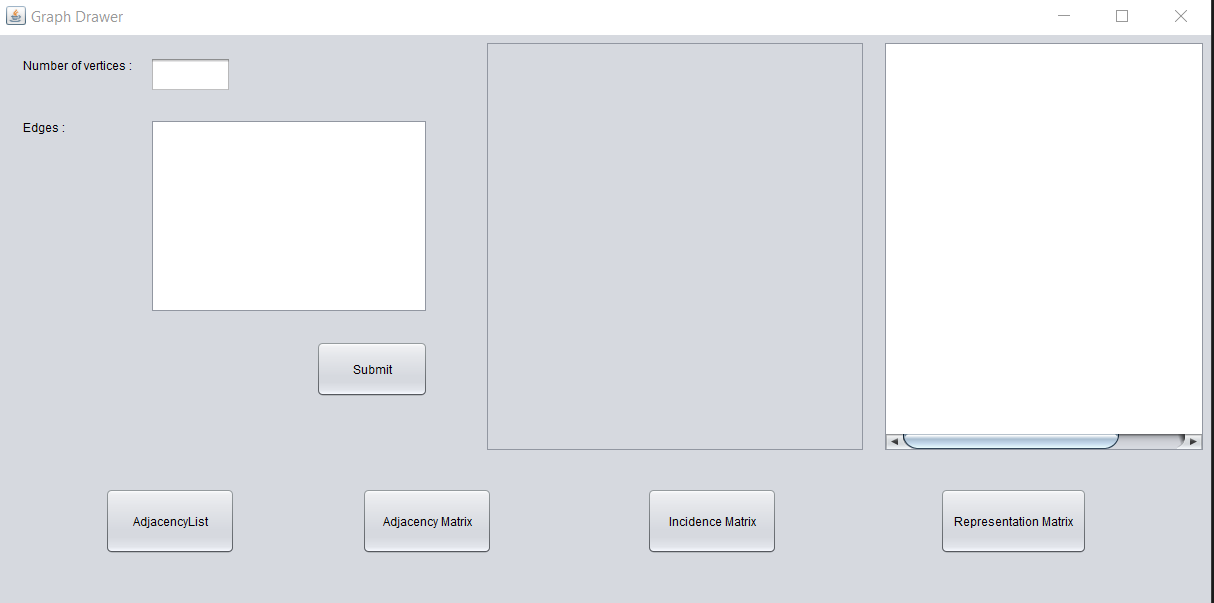
***How to use the program:***

4

3

2

1



*1. Enter number of vertices*

*2. Enter edges edge by edge*

*Edge input: Vertex ”space” Vertex  
Ex : 1 2*

*2 3*

*3. Click submit*

*4. Output appears.*

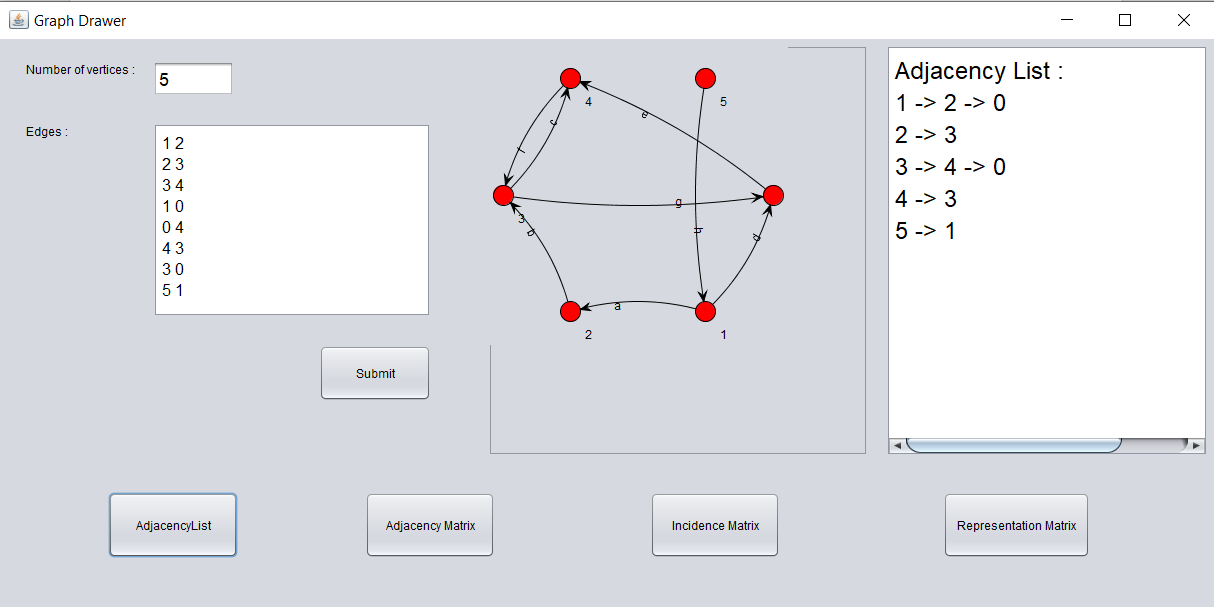
***Program in run:***

1

2

4

3



*1 - Adjacency List*

*2 - Adjacency Matrix*

*3 - Incidence Matrix*

*4 - Representation Matrix*

***Pseudo code for my program:***

*Adjacency List:*

Procedure Adjacency-List(maxN, E): // maxN denotes the maximum number of nodes

edge[maxN] = Vector() // E denotes the number of edges

for i from 1 to E

input -> x, y // Here x, y denotes there is an edge between x, y

edge[x].push(y)

edge[y].push(x)

end for

Return edge

*Adjacency Matrix:*

Procedure AdjacencyMatrix(N): //N represents the number of nodes

Matrix[N][N]

for i from 1 to N

for j from 1 to N

Take input -> Matrix[i][j]

endfor

endfor

*Incidence matrix:*

Begin

ed\_cnt := ed\_cnt + 1

inc\_matrix[u, ed\_cnt] := 1

inc\_matrix[v, ed\_cnt] := 1

End

References:

<https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>

<https://www.javatpoint.com/graph-and-graph-theory-introduction>