

Graph

Graph

A graph is a data structure that represents data using nodes as vertices and an edge between two vertices. If a graph is denoted as G , then we can describe G , as $G = (V, E)$. Where V is the set of vertices and E is the set of edges. The following is an illustration of graph:

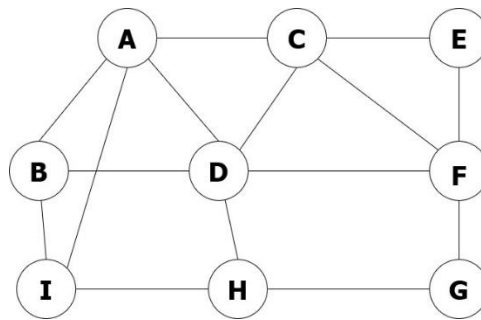


Figure 1: Graph (Undirected)

The graph has 9 vertices and 15 edges. So, this graph can be defined as:

$G = (V, E)$ where-

$V = \{A, B, C, D, E, F, G, H, I\}$;

$E = \{\{A, B\}, \{A, C\}, \{A, D\}, \{A, I\}, \{B, D\}, \{B, I\}, \{C, D\}, \{C, E\}, \{C, F\}, \{D, F\}, \{D, H\}, \{E, F\}, \{F, G\}, \{G, H\}, \{H, I\}\}$;

Undirected graph: A graph is called undirected graph if and only if none of the edges of the graph has any direction. The graph shown above is an undirected graph as none of the 15 edges has any direction.

Directed graph: A graph is called directed graph if and only if all the edges of the graph has a specific direction. The following graph is a directed graph:

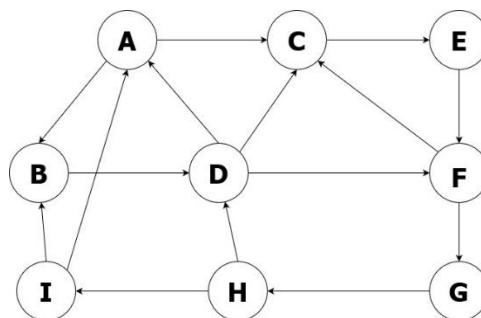


Figure 2: Graph (Directed)

Degree of vertex: The Degree of vertex for a vertex is the number of edges incident to that vertex. The degree of a vertex v in a graph G , written as $d(v)$. For the undirected graph in figure 1, the degree of the vertex D is 5, as it has 5 edges associated with it. For the directed graph in figure 2, the in-degree of the vertex D is 2 and out-degree of the vertex D is 3, as it has 2 edges coming into it and 3 edges going out from it.

Unweighted graph: A graph is an unweighted graph if no value as weight is associated with the edges of that graph. The graph in figure 1 is an undirected unweighted graph and the graph in figure 2 is a directed unweighted graph.

Weighted graph: A graph is a weighted graph if a value as weight is associated with the edges of that graph. The followings are illustration of weighted graph:

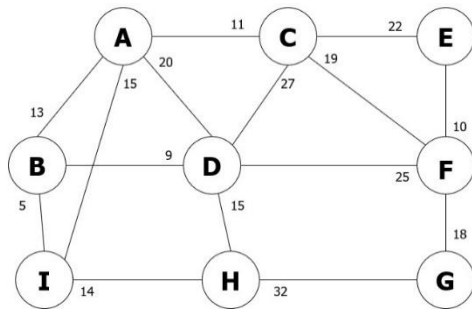


Figure 3: Undirected Weighted Graph

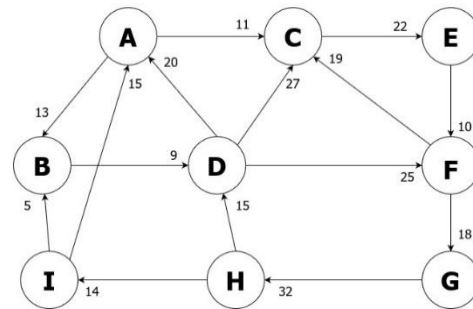


Figure 4: Directed Weighted Graph

Path: A Path is a sequence of vertices where each pair of successive vertices are connected with an edge.

Cycle: A cycle is a path where the first and the last vertices are same.

Cyclic Graph: A cyclic graph is a graph that contains at least one cycle.

Acyclic Graph: An acyclic graph is a graph that does not contain any cycle.

Spanning Subgraph: A spanning subgraph of a graph that contains all the vertices of that graph.

Spanning Tree: A spanning tree is a spanning subgraph of a graph that contains all the vertices of that graph and does not contain any cycle.

Minimum Spanning Tree: A minimum spanning tree is a spanning tree containing the edges with minimum weight. Prim's algorithm and Kruskal's algorithm can be used to construct a MST from a graph.

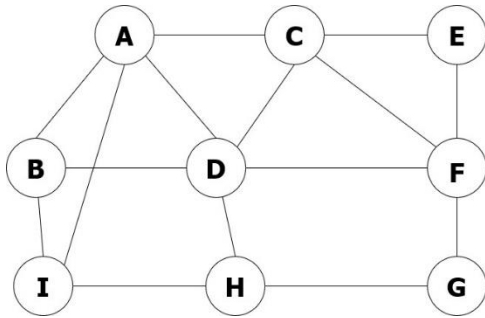
Graph Traversal: A vertex of a graph can be searched by using Breadth First Search (BFS) algorithm and Depth First Search (DFS) Algorithm. These algorithms can also be used for traversing a graph.

Graph Representation

A graph can be represented with both by a 2D array and a linked list. The array representation of a graph is called adjacency matrix. It is also known as Incidence Matrix. The linked list representation of a graph is called adjacency list.

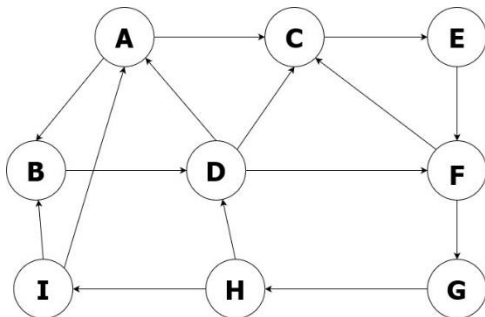
Adjacency Matrix

Undirected Graph



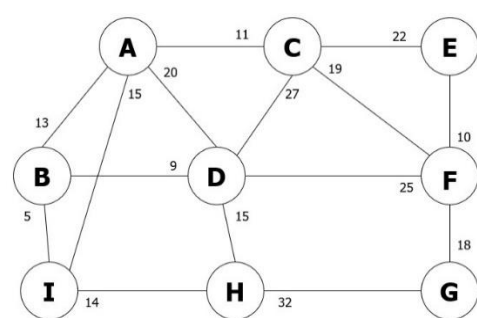
		A	B	C	D	E	F	G	H	I
		0	1	2	3	4	5	6	7	8
A	0	0	1	1	1	0	0	0	0	1
B	1	1	0	0	1	0	0	0	0	1
C	2	1	0	0	1	1	1	0	0	0
D	3	1	1	1	0	0	1	0	1	0
E	4	0	0	1	0	0	1	0	0	0
F	5	0	0	1	1	1	0	1	0	0
G	6	0	0	0	0	0	1	0	1	0
H	7	0	0	0	1	0	0	1	0	1
I	8	1	1	0	0	0	0	0	1	0

Directed Graph



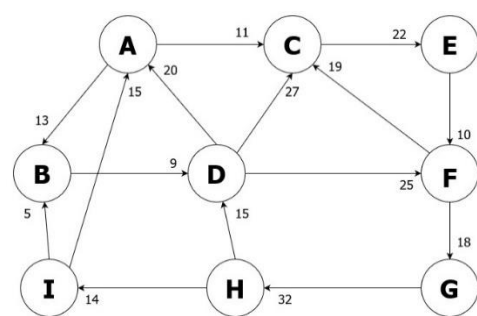
		A	B	C	D	E	F	G	H	I
		0	1	2	3	4	5	6	7	8
A	0	0	1	1	0	0	0	0	0	0
B	1	0	0	0	1	0	0	0	0	0
C	2	0	0	0	0	1	0	0	0	0
D	3	1	0	1	0	0	1	0	0	0
E	4	0	0	0	0	0	1	0	0	0
F	5	0	0	1	0	0	0	1	0	0
G	6	0	0	0	0	0	0	0	1	0
H	7	0	0	0	1	0	0	0	0	1
I	8	1	1	0	0	0	0	0	0	0

Undirected Weighted Graph



		A	B	C	D	E	F	G	H	I
		0	1	2	3	4	5	6	7	8
A	0	0	13	11	20	0	0	0	0	15
B	1	13	0	0	9	0	0	0	0	5
C	2	11	0	0	27	22	19	0	0	0
D	3	20	9	27	0	0	25	0	15	0
E	4	0	0	22	0	0	10	0	0	0
F	5	0	0	19	25	10	0	18	0	0
G	6	0	0	0	0	0	18	0	32	0
H	7	0	0	0	15	0	0	32	0	14
I	8	15	5	0	0	0	0	0	14	0

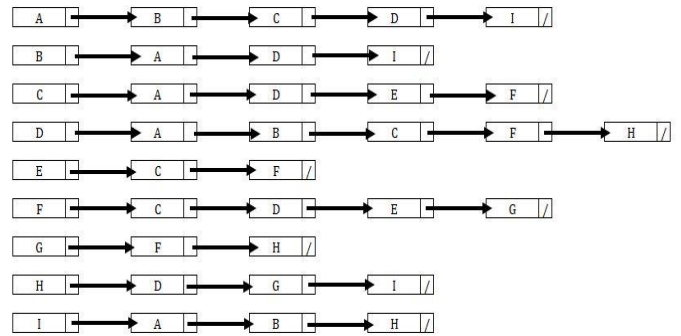
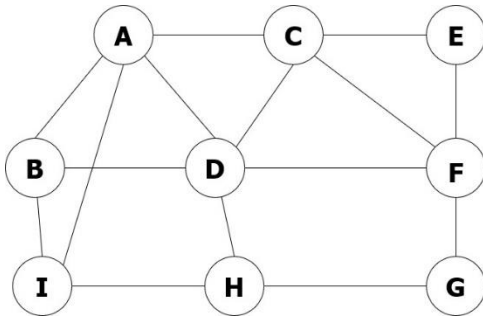
Directed Weighted Graph



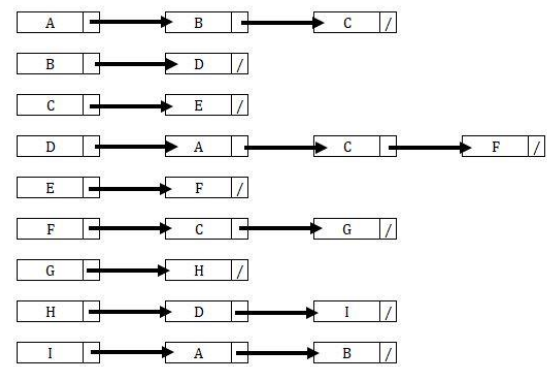
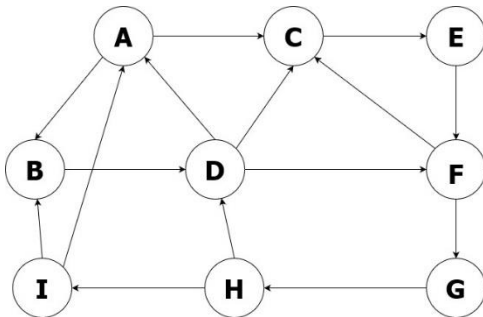
		A	B	C	D	E	F	G	H	I
		0	1	2	3	4	5	6	7	8
A	0	0	13	11	0	0	0	0	0	0
B	1	0	0	0	9	0	0	0	0	0
C	2	0	0	0	0	22	0	0	0	0
D	3	20	0	27	0	0	25	0	0	0
E	4	0	0	0	0	0	10	0	0	0
F	5	0	0	19	0	0	0	18	0	0
G	6	0	0	0	0	0	0	0	32	0
H	7	0	0	0	15	0	0	0	0	14
I	8	15	5	0	0	0	0	0	0	0

Adjacency List

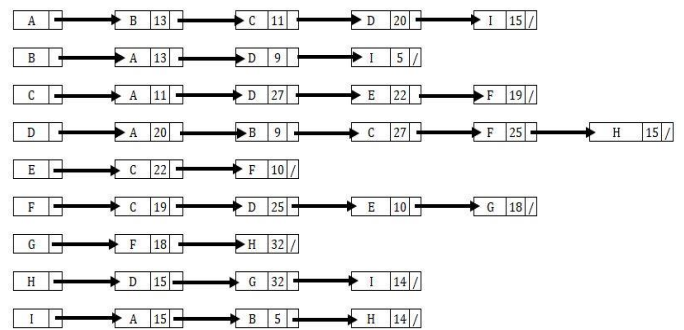
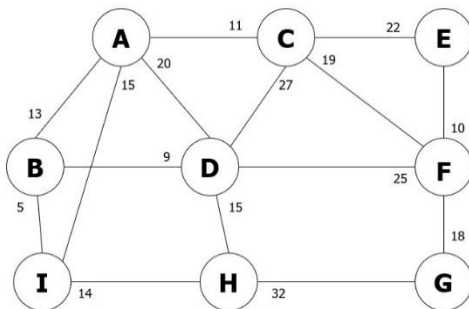
Undirected Graph



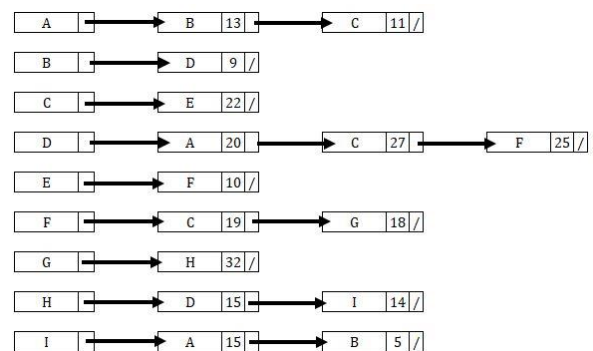
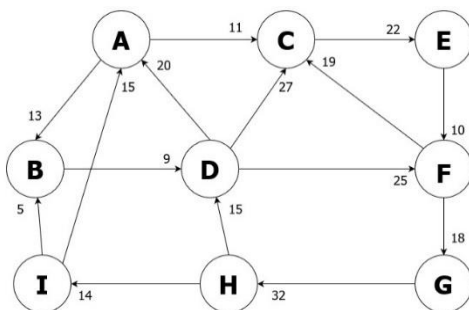
Directed Graph



Undirected Weighted Graph



Directed Weighted Graph



Pseudocode to Create Graph (Array Representation)

Input and Initializations: int v , int e .

Process (Undirected Unweighted Graph):

1. Declare an array $G[v][v]$ and assign 0 in all indexes.
2. Declare two variables int **from** and int **to**.
3. Enter the value of **from** and **to**.
4. The value of $G[\text{from}][\text{to}]$ will be 1. The value of $G[\text{to}][\text{from}]$ will be 1.
5. Repeat (2), (3) and (4) for all the edges.

Process (Directed Unweighted Graph):

1. Declare an array $G[v][v]$ and assign 0 in all indexes.
2. Declare two variables int **from** and int **to**.
3. Enter the value of **from** and **to**.
4. The value of $G[\text{from}][\text{to}]$ will be 1.
5. Repeat (2), (3) and (4) for all the edges.

Process (Undirected Weighted Graph):

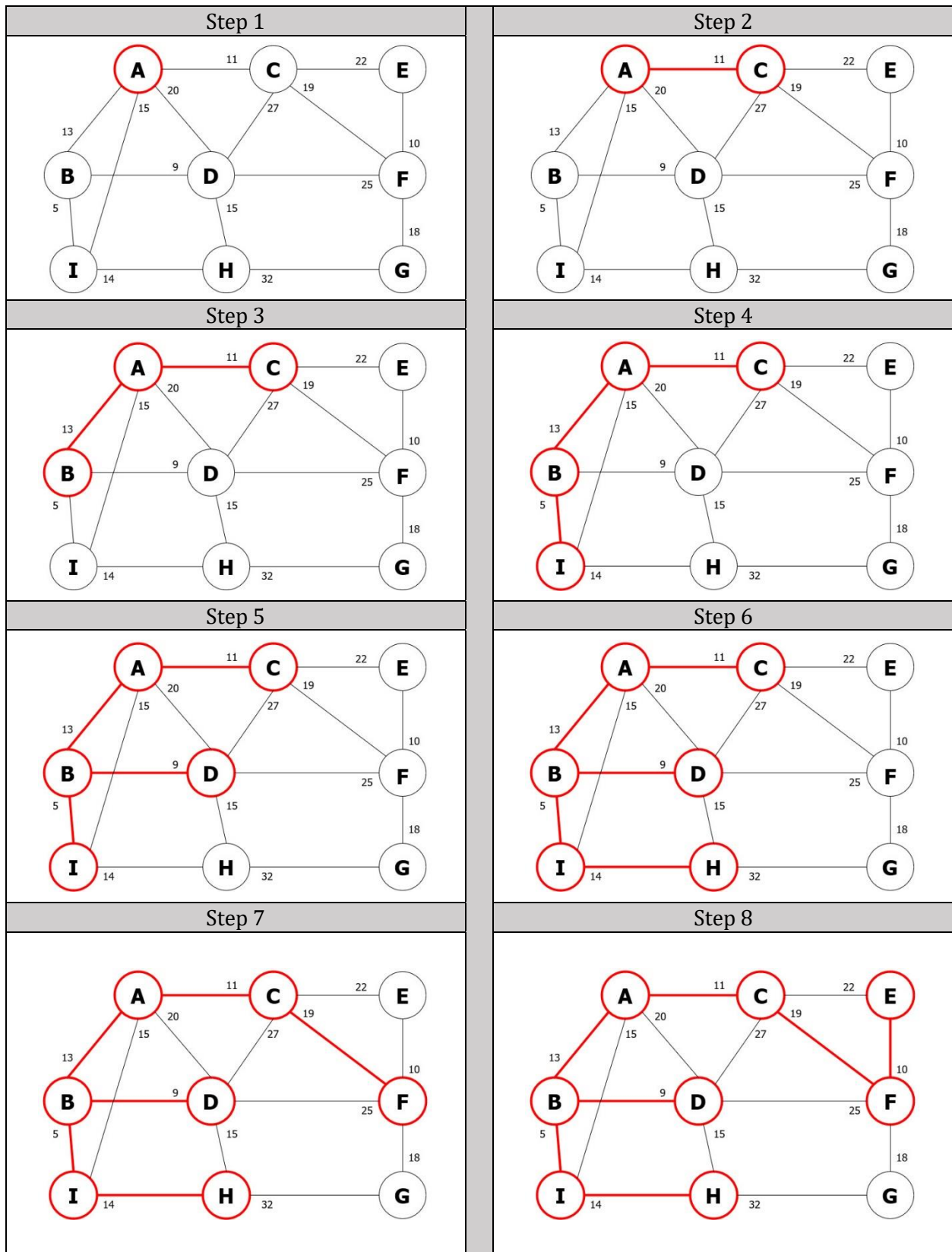
1. Declare an array $G[v][v]$ and assign 0 in all indexes.
2. Declare three variables int **from**, int **to** and int **w**.
3. Enter the value of **from**, **to** and **w**.
4. The value of $G[\text{from}][\text{to}]$ will be **w**. The value of $G[\text{to}][\text{from}]$ will be **w**.
5. Repeat (2), (3) and (4) for all the edges.

Process (Directed Weighted Graph):

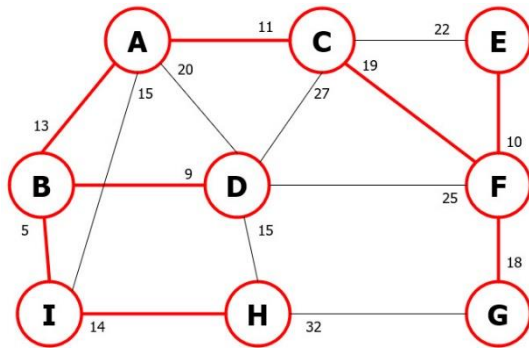
1. Declare an array $G[v][v]$ and assign 0 in all indexes.
2. Declare three variables int **from**, int **to** and int **w**.
3. Enter the value of **from**, **to** and **w**.
4. The value of $G[\text{from}][\text{to}]$ will be **w**.
5. Repeat (2), (3) and (4) for all the edges.

Output: The graph $G[v][v]$.

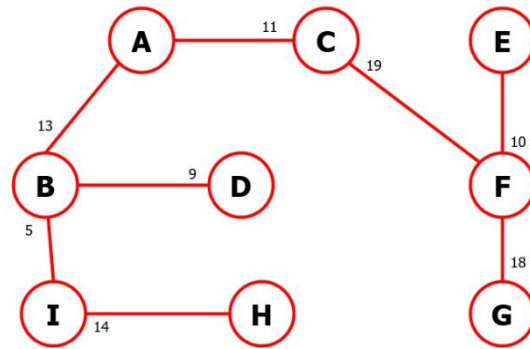
Simulation of Prim's Algorithm



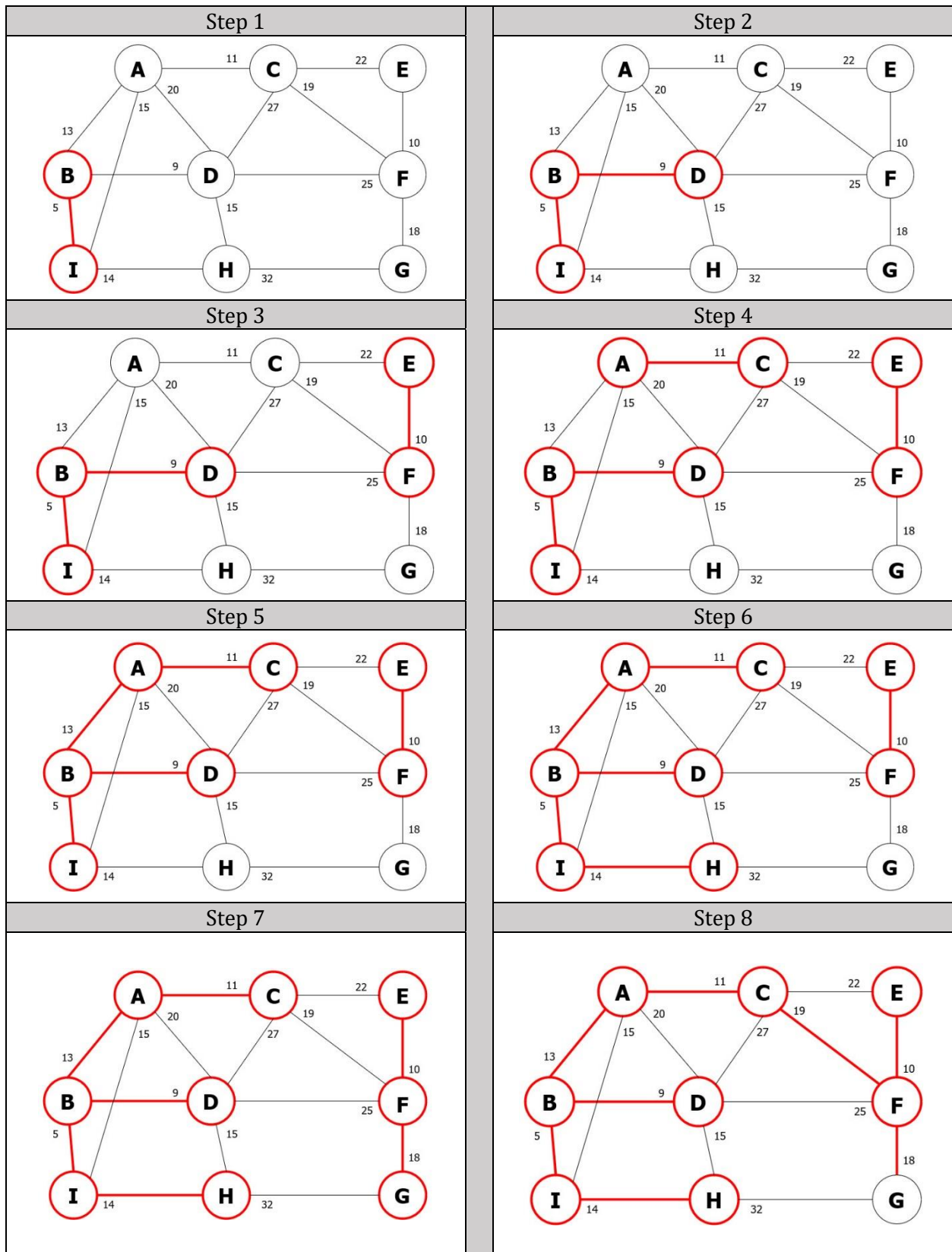
Step 9



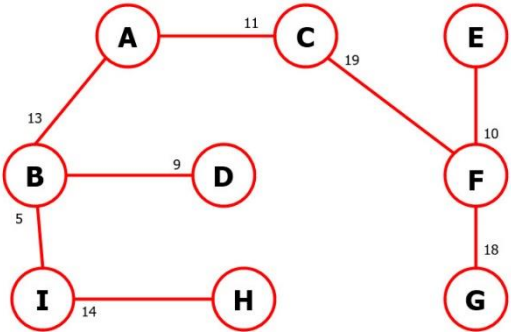
Step 10



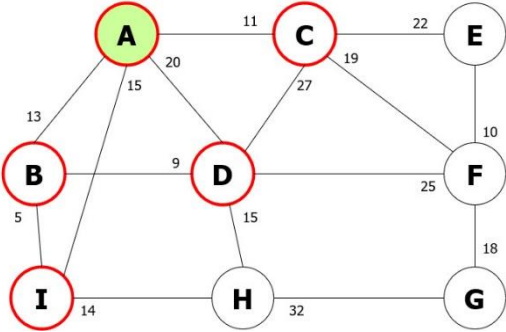
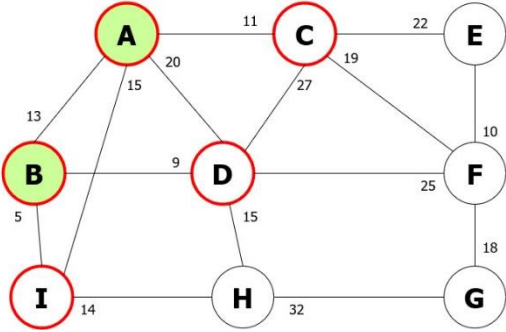
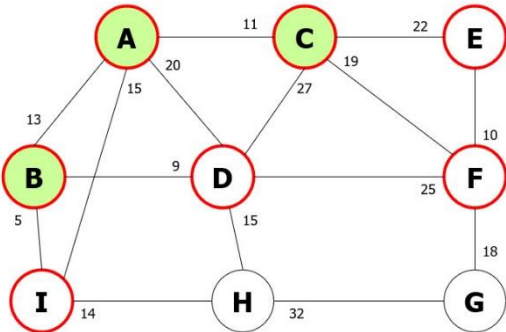
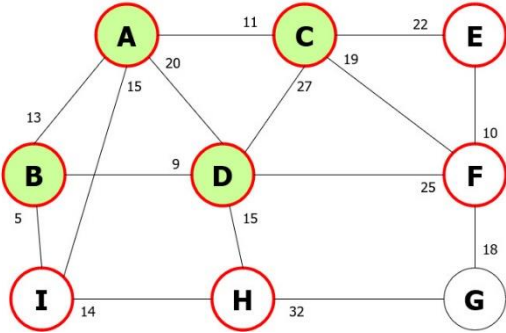
Simulation of Kruskal's Algorithm

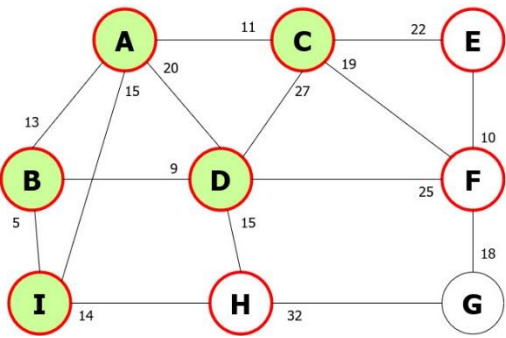
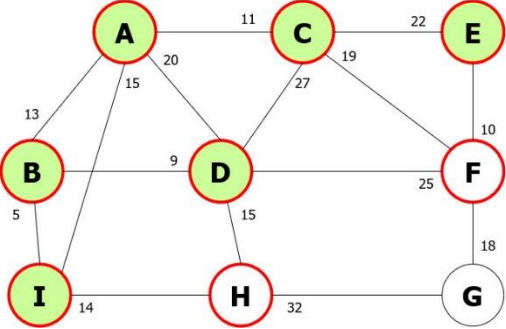
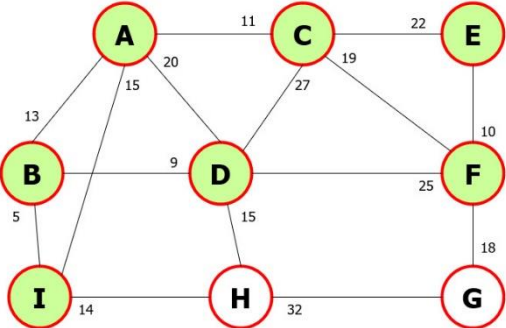
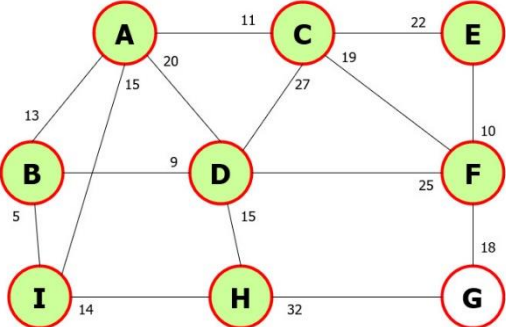


Step 9

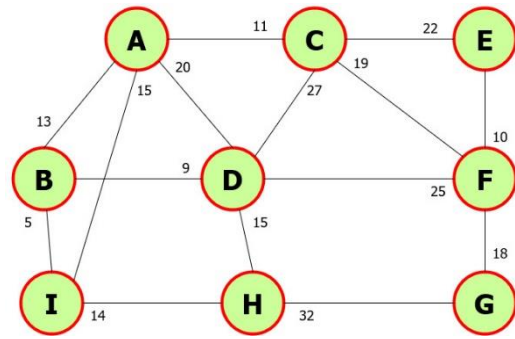


Simulation of BFS

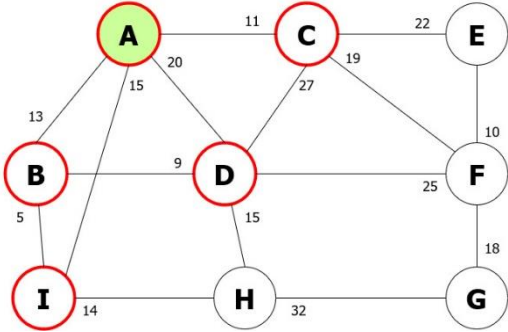
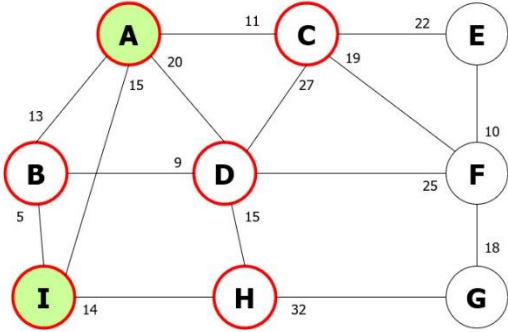
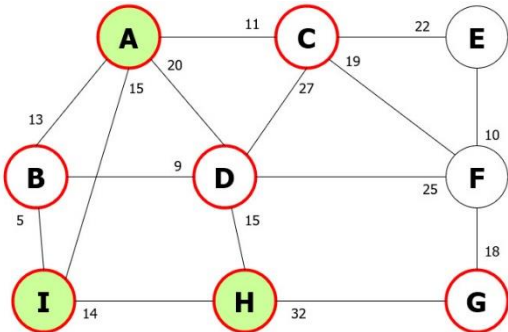
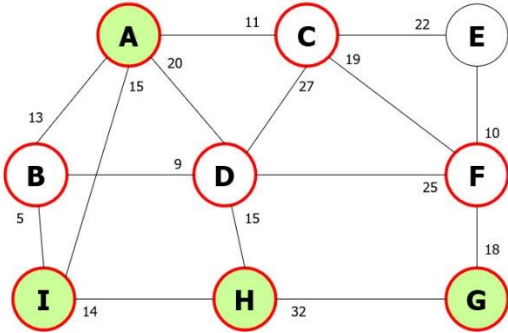
Vertex Visited	Vertex in Queue	Graph
A	B, C, D, I	
A, B	C, D, I	
A, B, C	D, I, E, F	
A, B, C, D	I, E, F, H	

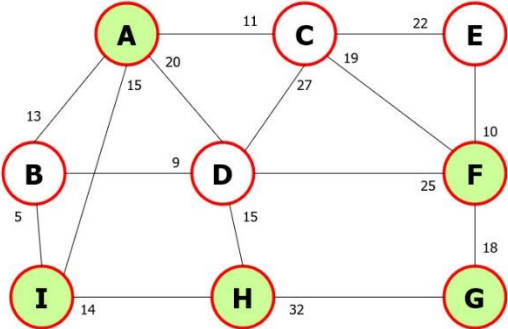
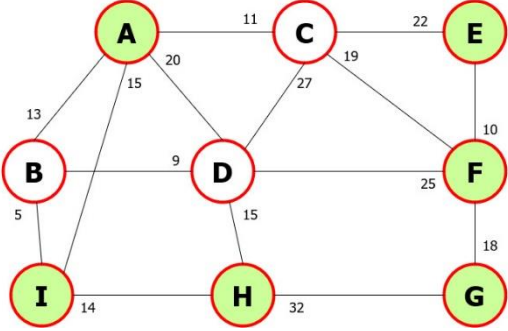
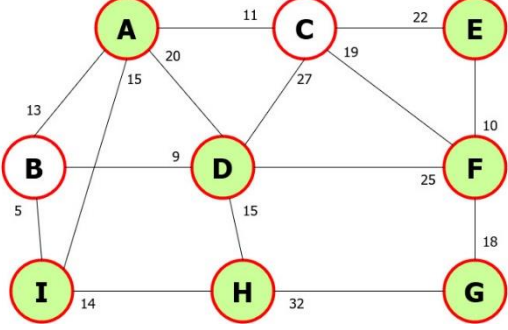
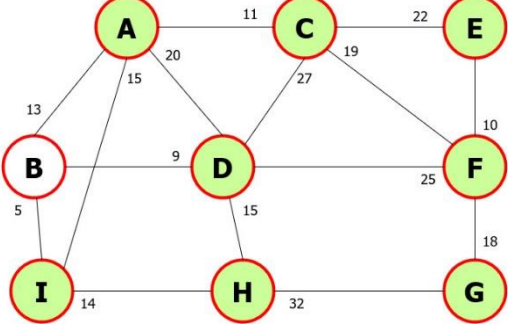
A, B, C, D, I	E, F, H	
A, B, C, D, I, E	F, H	
A, B, C, D, I, E, F	H, G	
A, B, C, D, I, E, F, H	G	

A, B, C, D, I, E, F, H, G



Simulation of DFS

Vertex Visited	Vertex in Stack	Graph
A	B, C, D, I	
A, I	B, C, D, H	
A, I, H	B, C, D, G	
A, I, H, G	B, C, D, F	

A, I, H, G, F	B, C, D, E	
A, I, H, G, F, E	B, C, D	
A, I, H, G, F, E, D	B, C	
A, I, H, G, F, E, D, C	B	

A, I, H, G, F, E, D, C, B

