Applications of graphs

Graphs are constructed for various types of applications such as:

- In circuit networks where points of connection are drawn as vertices and component wires become the edges of the graph.
- In transport networks where stations are drawn as vertices and routes become the edges of the graph.
- In maps that draw cities/states/regions as vertices and adjacency relations as edges.
- In program flow analysis where procedures or modules are treated as vertices and calls to these procedures are drawn as edges of the graph.
- Once we have a graph of a particular concept, they can be easily used for finding shortest paths, project planning, etc.
- In flowcharts or control-flow graphs, the statements and conditions in a program are represented as nodes and the flow of control is represented by the edges.

Applications of graphs

- In state transition diagrams, the nodes are used to represent states and the edges represent legal moves from one state to the other.
- Graphs are also used to draw activity network diagrams. These diagrams are extensively used as a project management tool to represent the interdependent relationships between groups, steps, and tasks that have a significant impact on the project..

Applications of graphs

- **MINIMUM SPANNING TREE**
  - **KRUSKAL’S ALGORITHM**
  - **PRIM’S ALGORITHM**
- **SHORTEST PATH ALGORITHMS**
  - **FLOYD’S ALGORITHM / FLOYD WARSHALL’S ALGORITHM**

    (ALL PAIR SHORTEST PATH)
  - **DIJKSTRA’S ALGORITHM**

    (SINGLE SOURCE SHORTEST PATH)

Minimum Spanning Trees

A **Spanning Tree** is a tree formed from graph G using the edges that connect all the vertices of G

A **Minimum Spanning Tree (MST)** is a spanning tree with minimum weight or lowest total cost.

Algorithms to construct MST

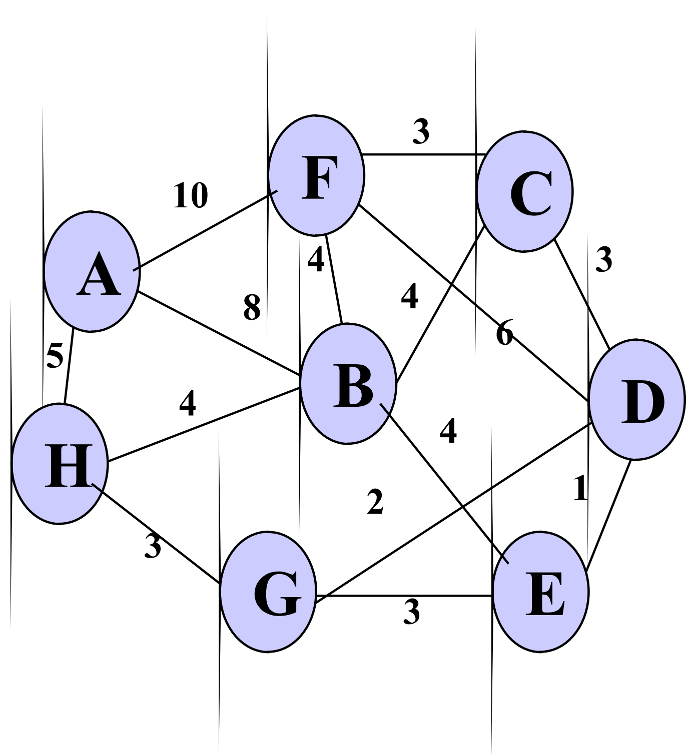
- **Kruskal’s Algorithm**
- **Prim’s Algorithm**

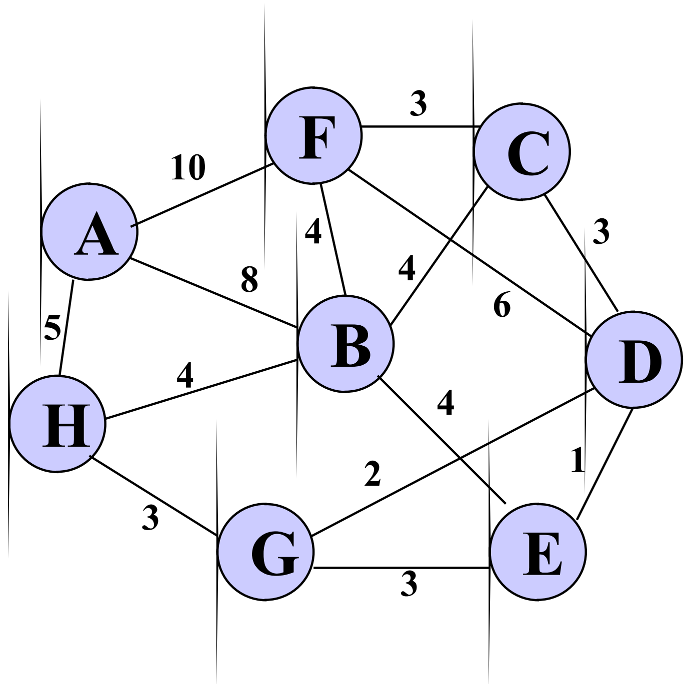
Kruskal's Algorithm

- Work with edges, rather than nodes
- Two steps:
  - Sort edges by increasing edge weight
  - Select the first $|V| - 1$ edges that do not generate a cycle

Kruskal's Algorithm

Consider an undirected, weight graph

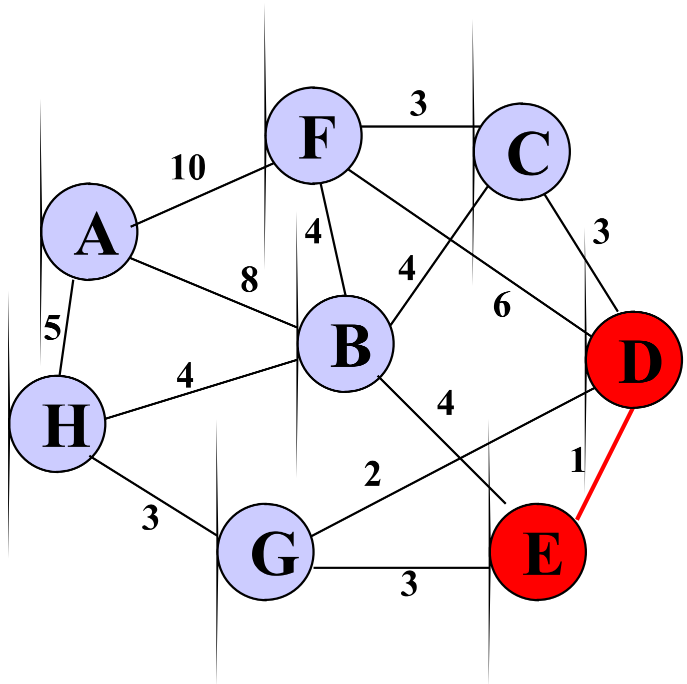




Sort the edges by increasing edge weight

edge	d_v	
(D,E)	1	
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

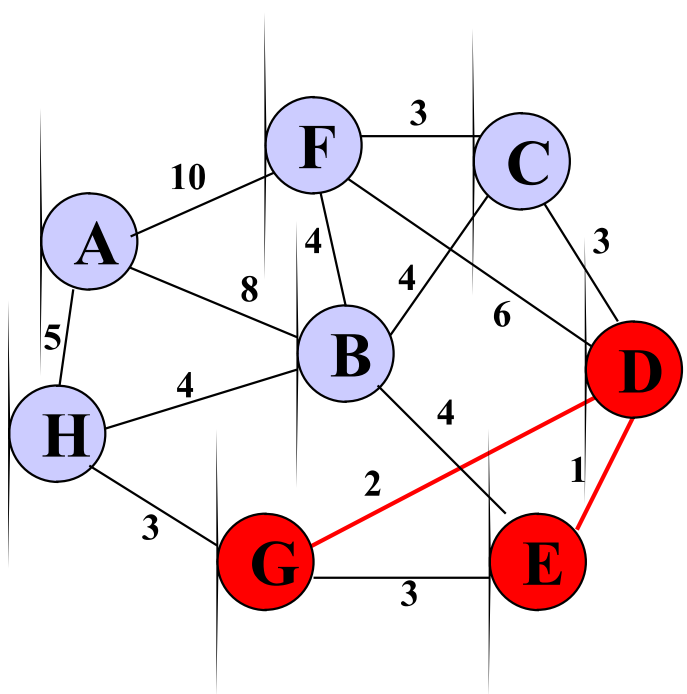
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

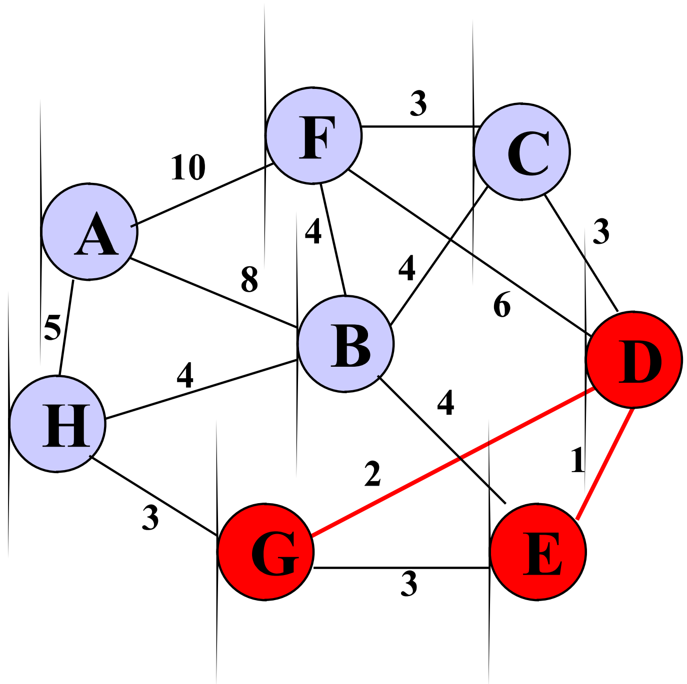
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

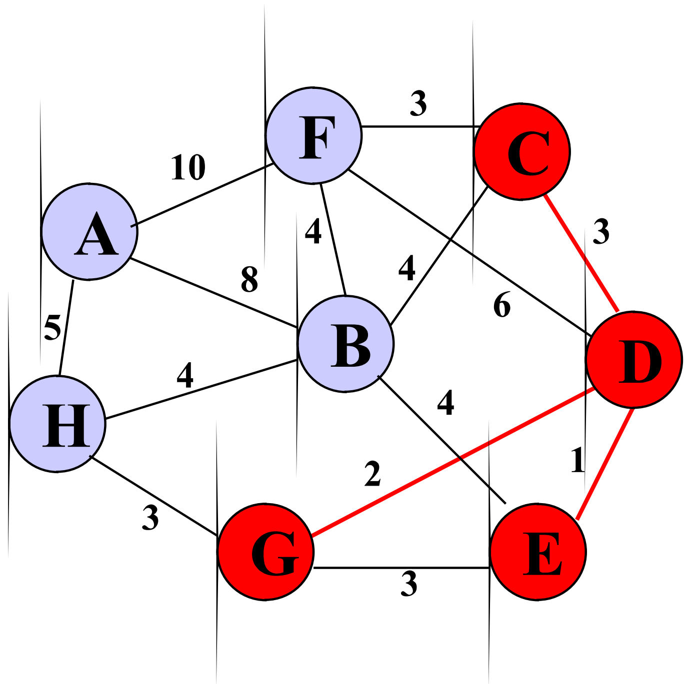


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	*X*
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

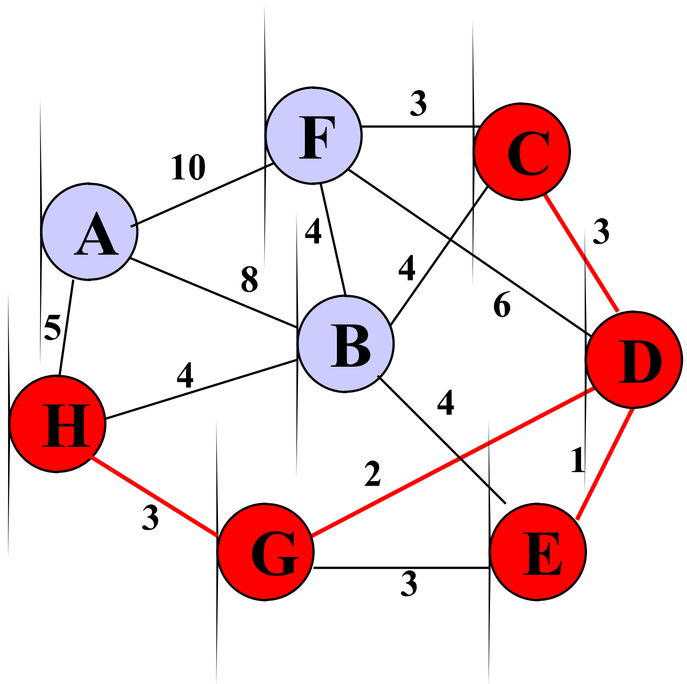
Accepting edge (E,G) would create a cycle



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	*X*
(C,D)	3	√
(G,H)	3	
(C,F)	3	
(B,C)	4	

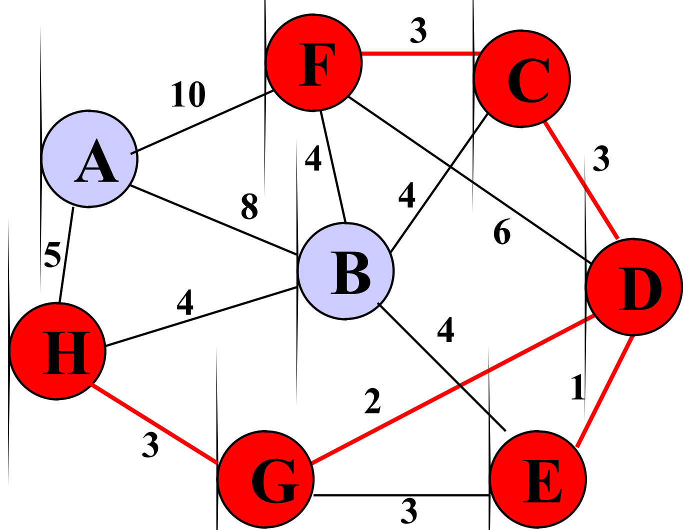
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	*X*
(C,D)	3	√
(G,H)	3	√
(C,F)	3	
(B,C)	4	

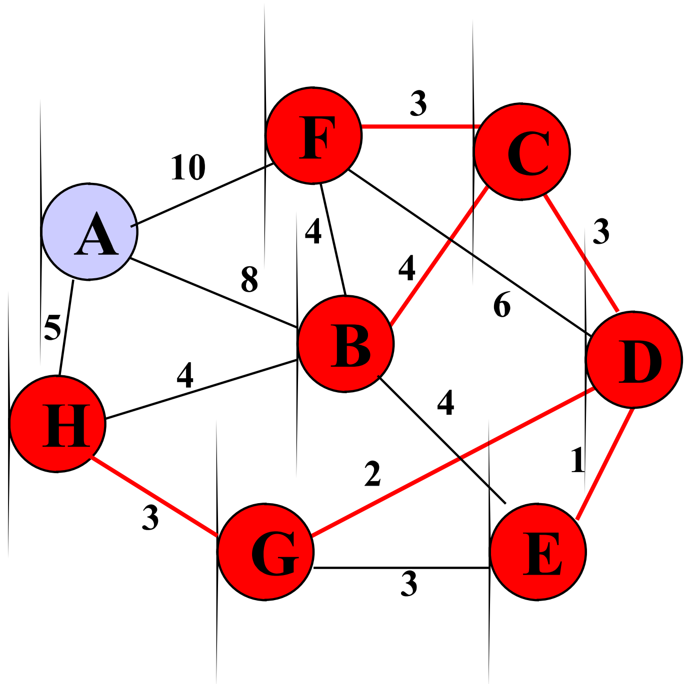
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	***X***
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	

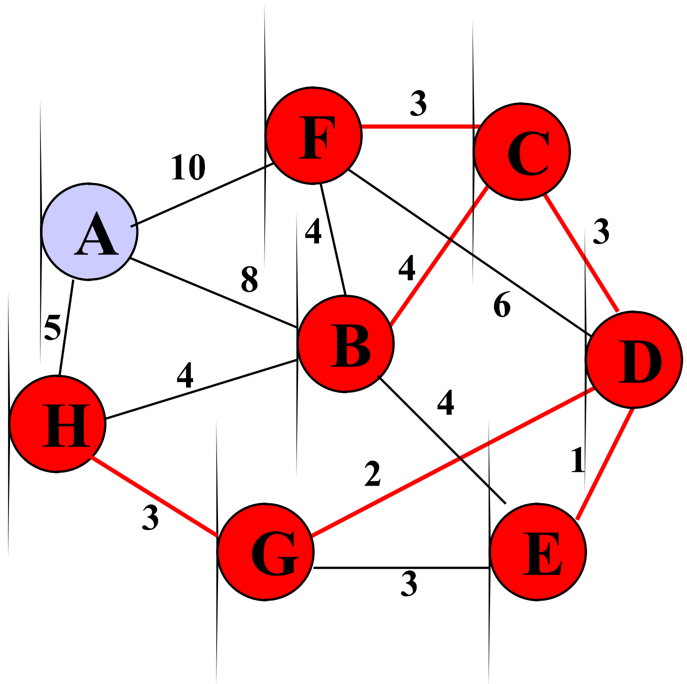
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	*X*
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

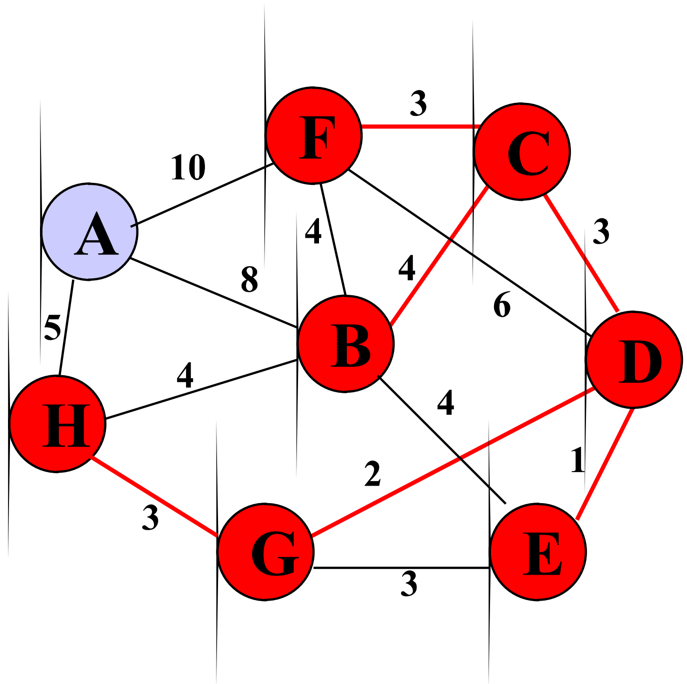
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	***X***
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

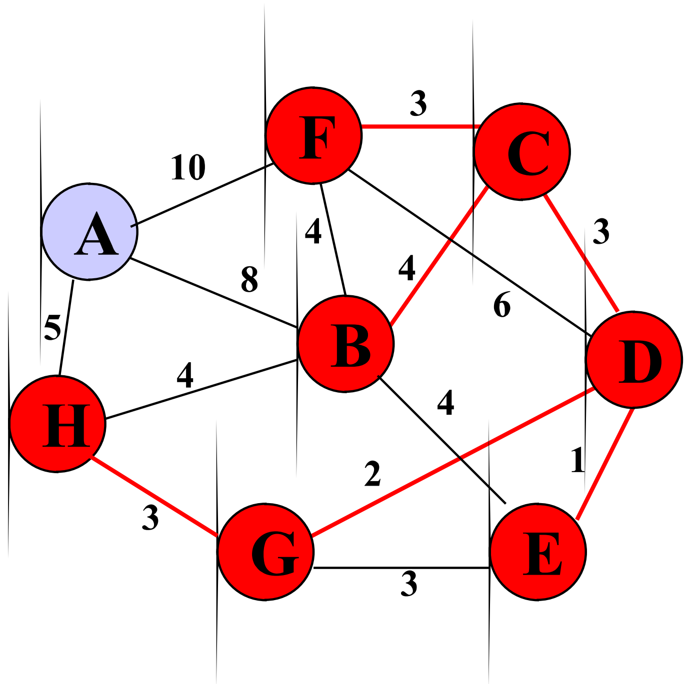
edge	d_v	
(B,E)	4	***X***
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	*X*
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

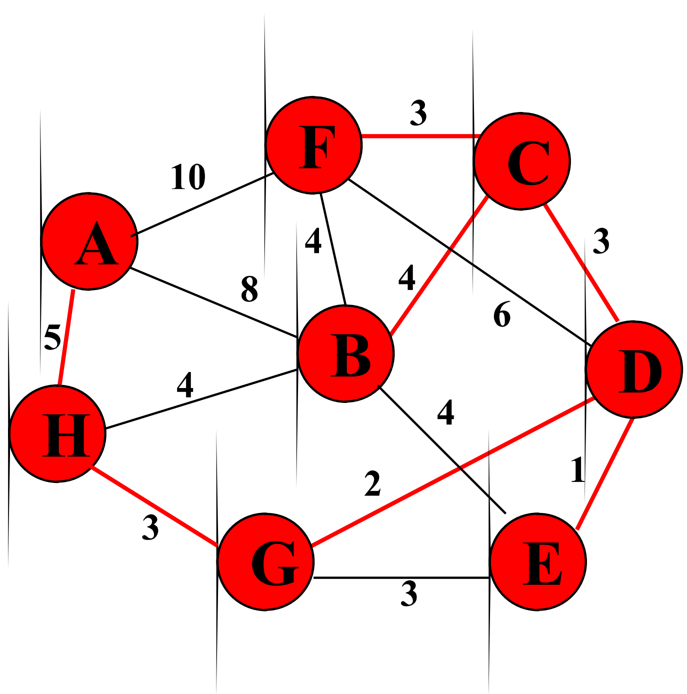
edge	d_v	
(B,E)	4	*X*
(B,F)	4	*X*
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	***X***
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

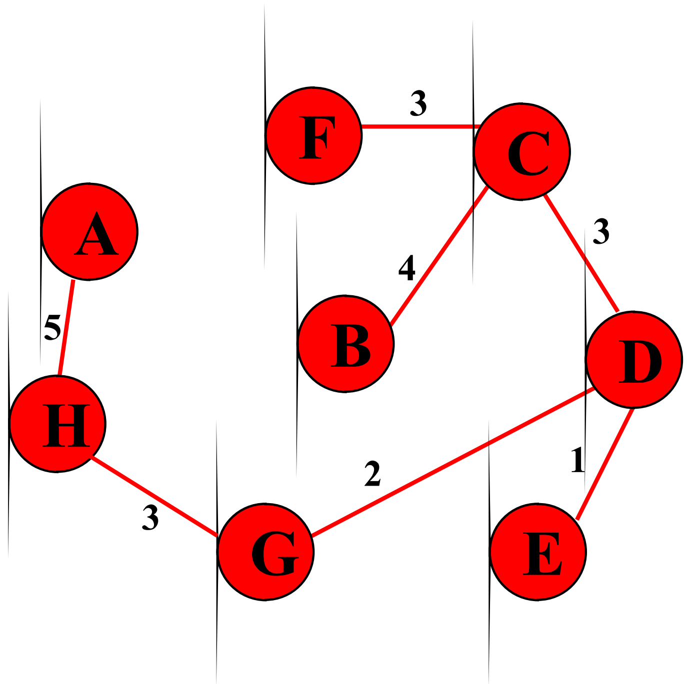
edge	d_v	
(B,E)	4	***X***
(B,F)	4	***X***
(B,H)	4	***X***
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first $|V|-1$ edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	***X***
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	***X***
(B,F)	4	***X***
(B,H)	4	***X***
(A,H)	5	√
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	*X*
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	*X*
(B,F)	4	*X*
(B,H)	4	*X*
(A,H)	5	√
(D,F)	6	
(A,B)	8	
(A,F)	10	

not considered

Done

Total Cost $= \sum d_v = 21$

KRUSKAL'S ALGORITHM

KRUSKAL'S ALGORTIHM

```
Step 1: Create a forest in such a way that each graph is a separate tree.
Step 2: Create a priority queue Q that contains all the edges of the graph.
Step 3: Repeat Steps 4 and 5 while Q is NOT EMPTY
Step 4:      Remove an edge from Q
Step 5:      IF the edge obtained in Step 4 connects two different trees, then
                    Add it to the forest (for combining two trees into one tree).
        ELSE
          Discard the edge
Step 6: END
```

Prim's Algorithm

New vertex add to the tree by choosing the edge (u,v) such that the cost of (u,v) is the smallest among all edges where u is in the tree and v is not

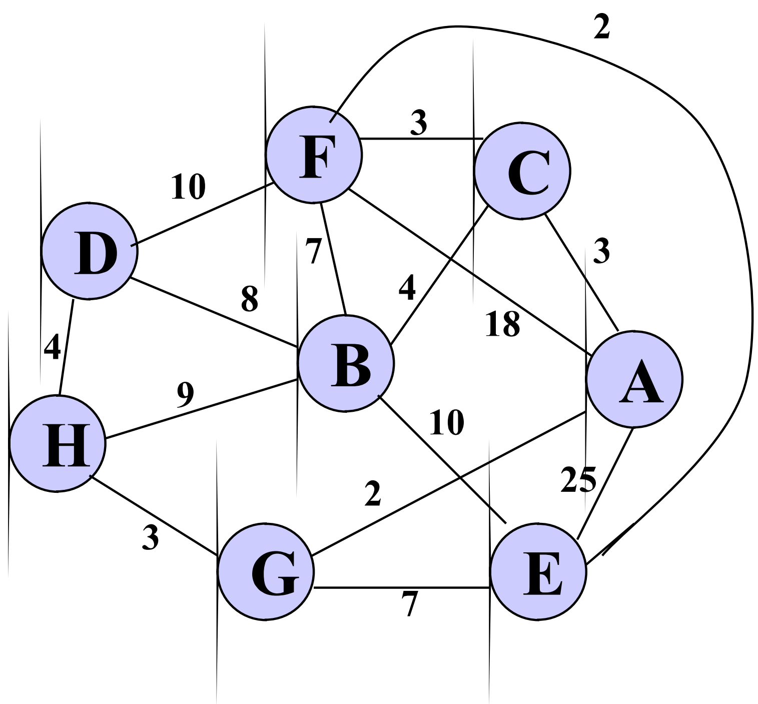
Vertex	*Known*	d_v	p_v
A			
B			
C			
D			
E			
F			
G			
H			

Known – T / F

d_v - ***Weight of the shortest edge connecting v to a known vertex***

p_v - ***Last vertex to cause a change in*** d_v

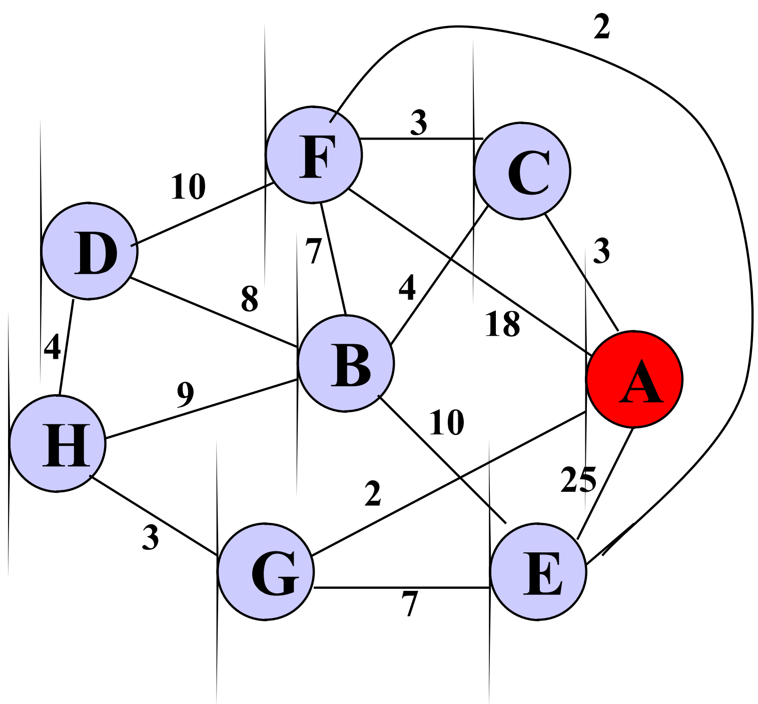
Step-1



Initialize Configuration

V	K	d_v	p_v
A	F	∞	–
B	F	∞	–
C	F	∞	–
D	F	∞	–
E	F	∞	–
F	F	∞	–
G	F	∞	–
H	F	∞	–

Step-2

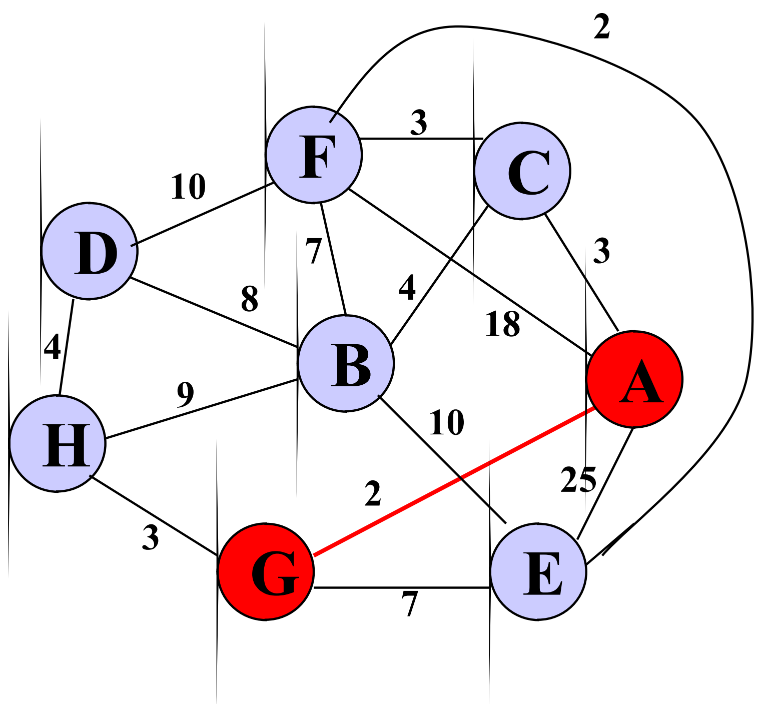


Start with any node, say A

After A is declared known

V	*K*	d_v	p_v
A	T	0	–
B	F	∞	–
C	F	3	A
D	F	∞	–
E	F	25	A
F	F	18	A
G	F	2	A
H	F	∞	–

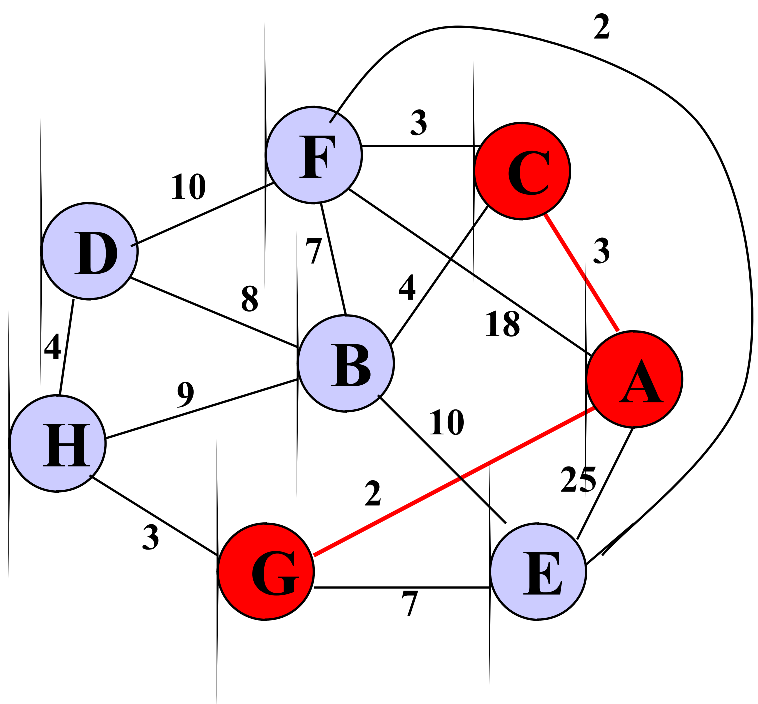
Step-3



After G is declared known

V	K	d_v	p_v
A	T	0	–
B	F	∞	–
C	F	3	A
D	F	∞	–
E	F	7	G
F	F	18	A
G	T	2	A
H	F	3	G

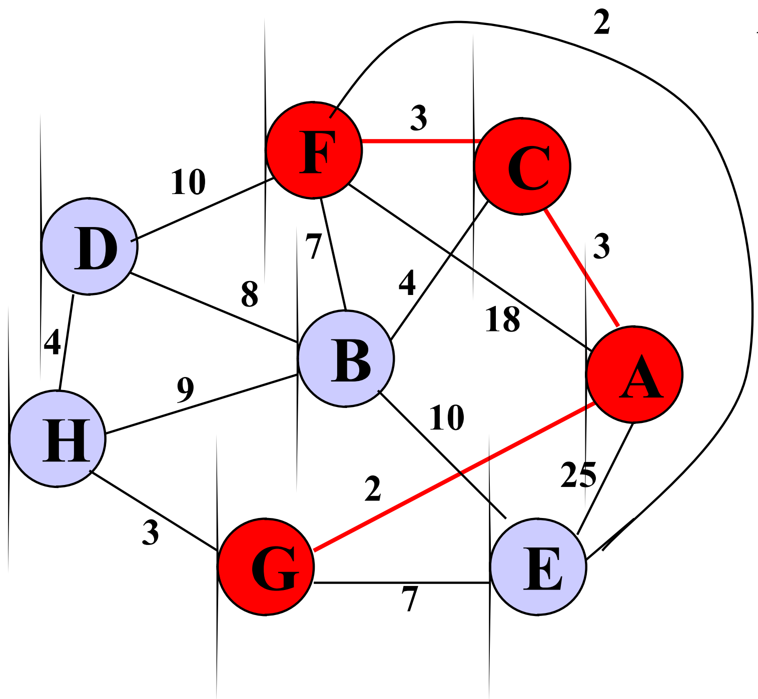
Step-4



After C is declared known

V	K	d_v	p_v
A	T	0	–
B	F	4	C
C	T	3	A
D	F	∞	–
E	F	7	G
F	F	3	C
G	T	2	A
H	F	3	G

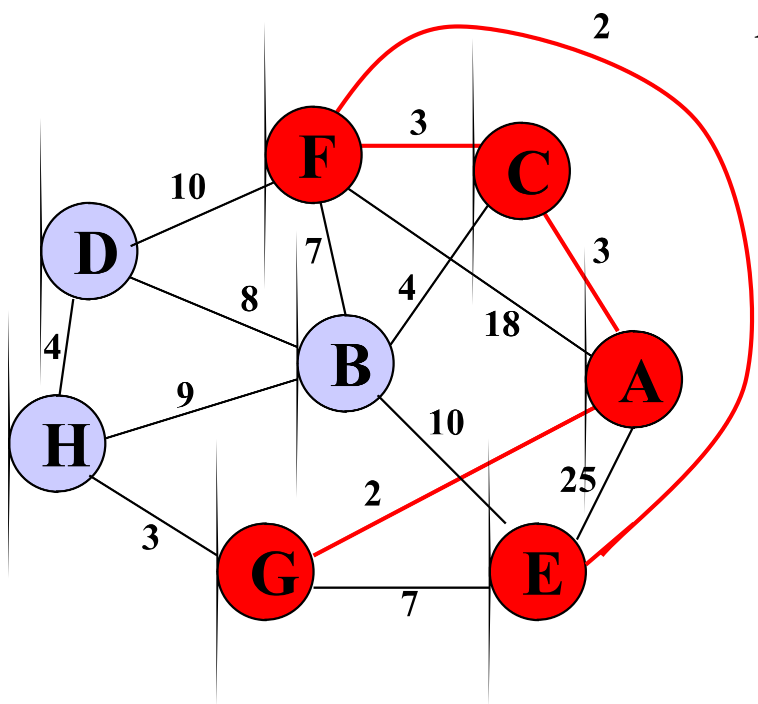
Step-5



After F is declared known

V	K	d_v	P_v
A	T	0	–
B	F	4	C
C	T	3	A
D	F	10	F
E	F	2	F
F	T	3	C
G	T	2	A
H	F	3	G

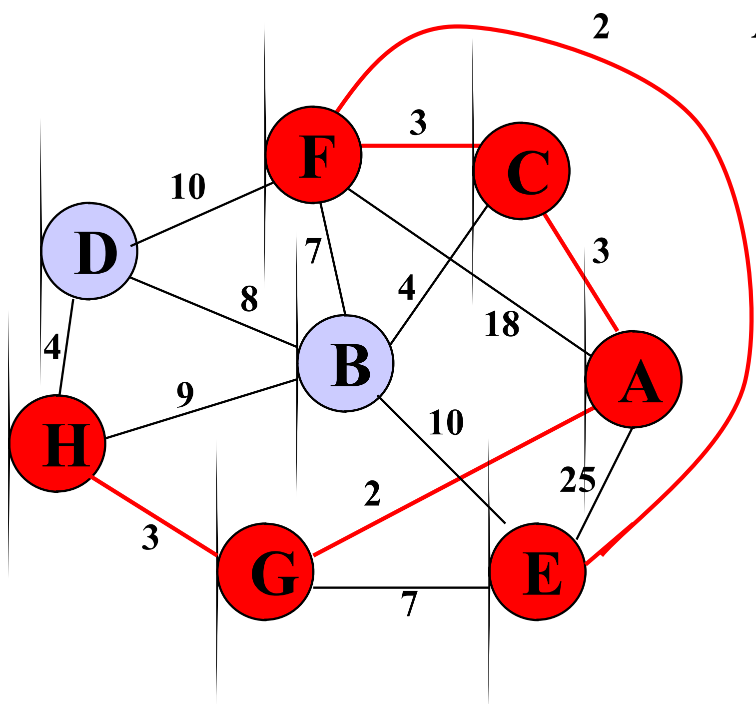
Step-6



After E is declared known

V	K	d_v	p_v
A	T	0	–
B	F	4	C
C	T	3	A
D	F	10	F
E	T	2	F
F	T	3	C
G	T	2	A
H	F	3	G

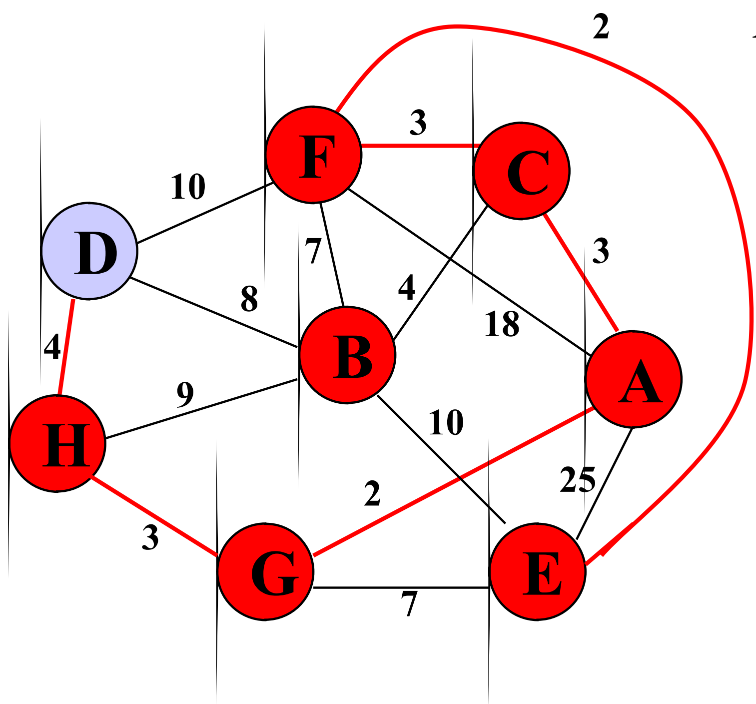
Step-7



After H is declared known

V	K	d_v	p_v
A	T	0	–
B	F	4	C
C	T	3	A
D	F	4	H
E	T	2	F
F	T	3	C
G	T	2	A
H	T	3	G

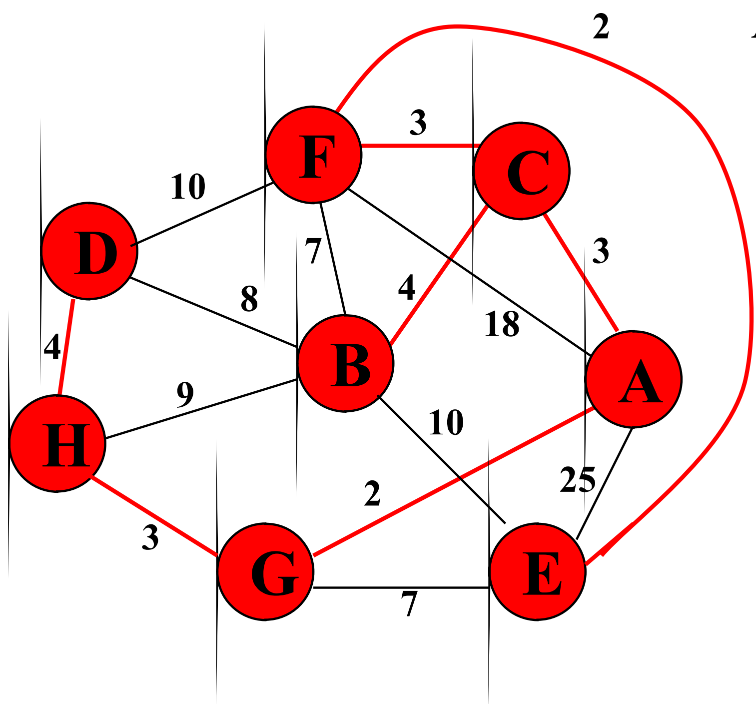
Step-8



After B is declared known

V	K	d_v	p_v
A	T	0	–
B	T	4	C
C	T	3	A
D	F	4	H
E	T	2	F
F	T	3	C
G	T	2	A
H	T	3	G

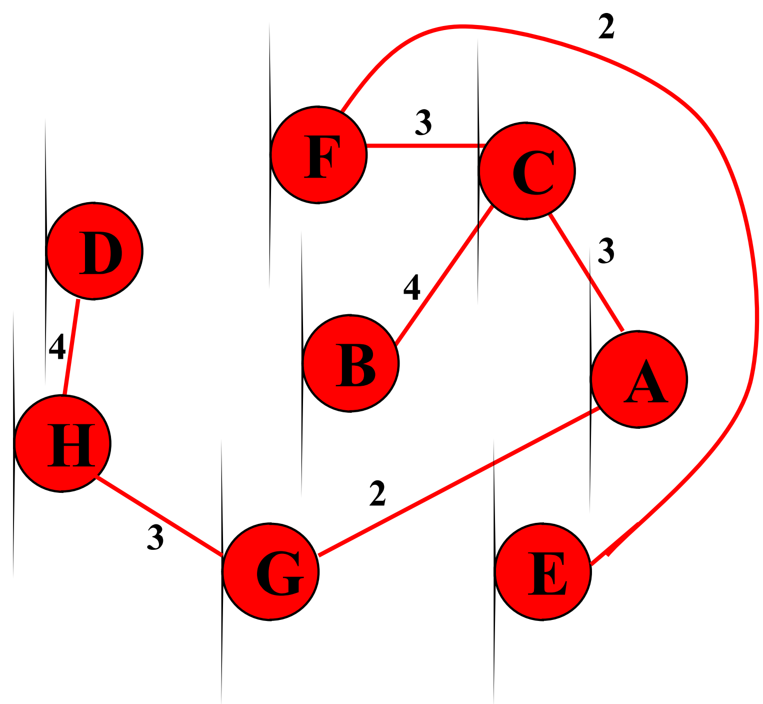
Step-9



After D is declared known

V	K	d_v	p_v
A	T	0	–
B	T	4	C
C	T	3	A
D	T	4	H
E	T	2	F
F	T	3	C
G	T	2	A
H	T	3	G

Step-10



Cost of Minimum
Spanning Tree = $\Sigma\, d_v$ = **21**

V	K	d_v	p_v
A	T	0	–
B	T	4	C
C	T	3	A
D	T	4	H
E	T	2	F
F	T	3	C
G	T	2	A
H	T	3	G

Done

PRIM'S ALGORITHM

Prim Algorithm

- **Step 1: Select a starting vertex**
- **Step 2: Repeat Steps 3 and 4 until there are fringe vertices**
- **Step 3: Select an edge e connecting the tree vertex and fringe vertex that has minimum weight**
- **Step 4: Add the selected edge and the vertex to the minimum spanning tree T**
- **[END OF LOOP]**
- **Step 5: EXIT**