Minimum Spanning Tree

By

Arun Cyril Jose

Undirected Graphs

- Undirected Graph is a **Tree** if there is exactly one simple path between any pair of vertices.

Trees

- $|E| = |V| - 1$
- Connected
- No Cycles

- Any of these two properties imply the third property, and signifies the Graph is a Tree.

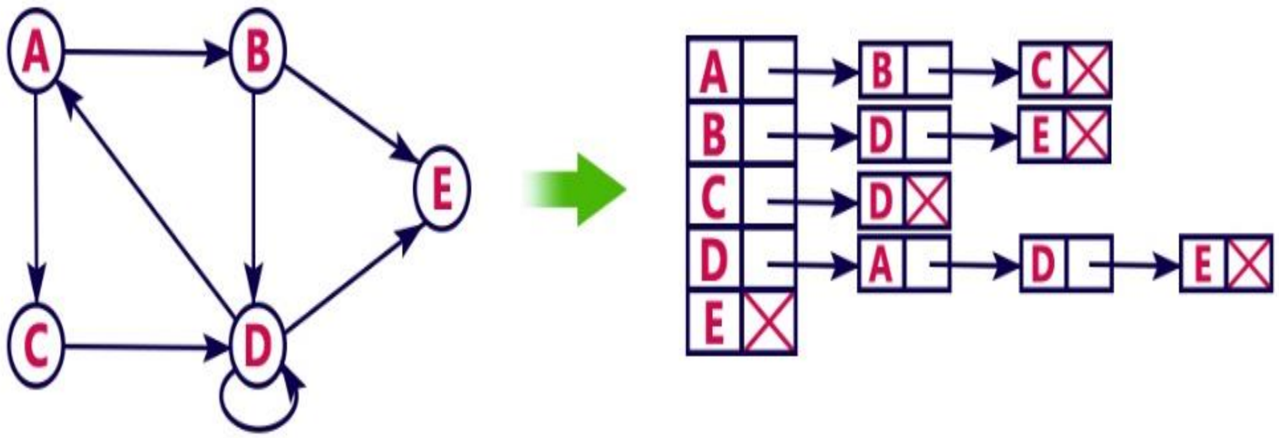
Graphs

- A **path** is **simple** if no vertices are repeated.
- A **path** forms a **cycle** if last vertex = first vertex.
- A **cycle** is **simple** if no vertices are repeated other than first and last.
- Two representations:
  - **Adjacency list** – Best suited for **Sparse** Graphs (few edges).
  - **Adjacency matrix** – Best suited for **Dense** graphs (lots of edges).

Graph Representations

- **Adjacency List**

- Requires Θ (V + E) space.



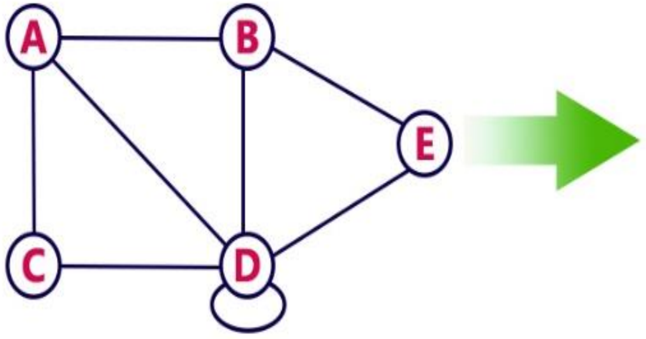
- Requires Θ (degree(u)) time to check if edge (u,v) is in E.

- Space efficient for Sparse Graphs.

Graph Representations

- **Adjacency Matrix**

- Given a Graph G = (V, E)
- Number the vertices 1, 2, …., |V|

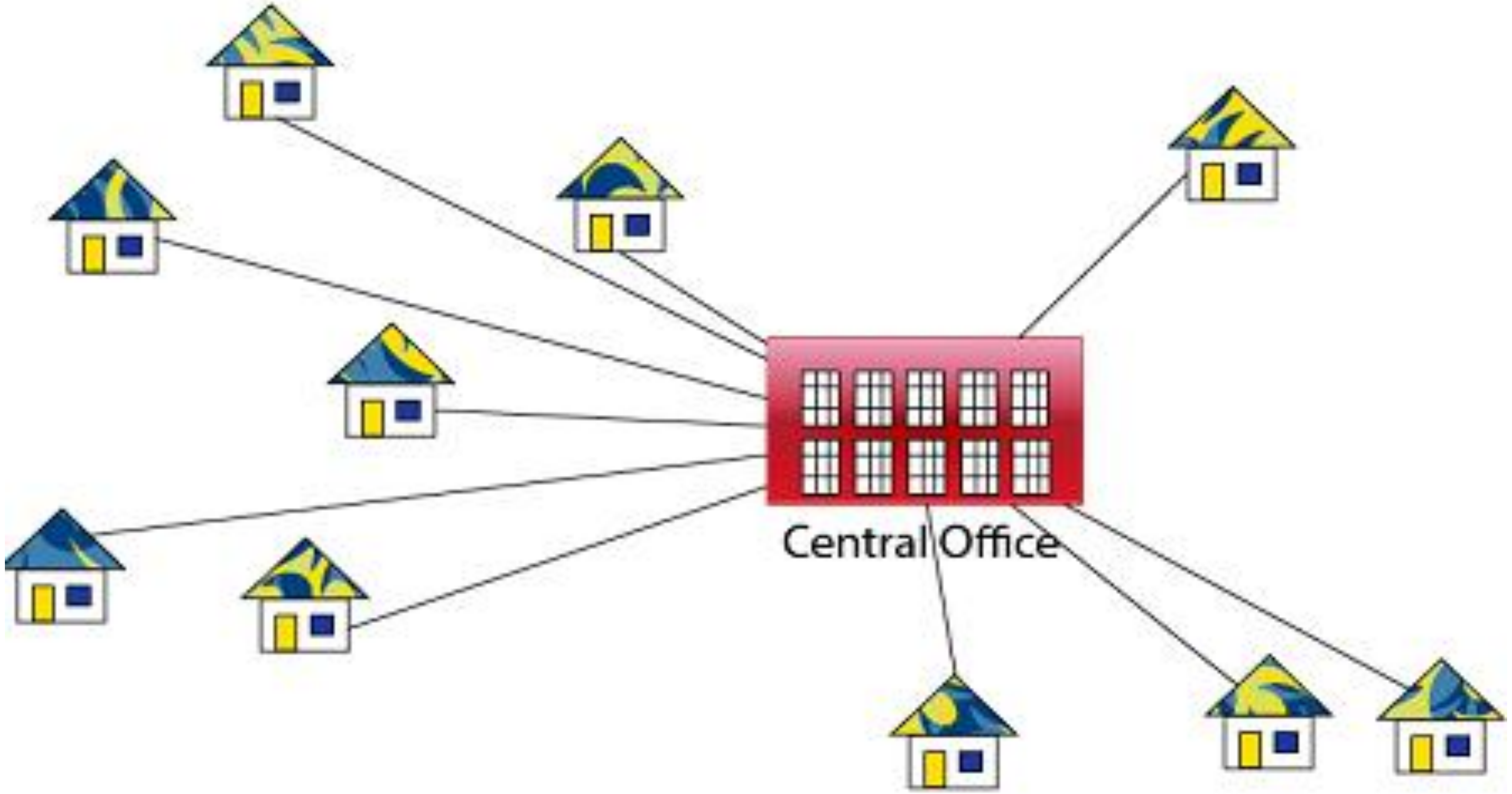


$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

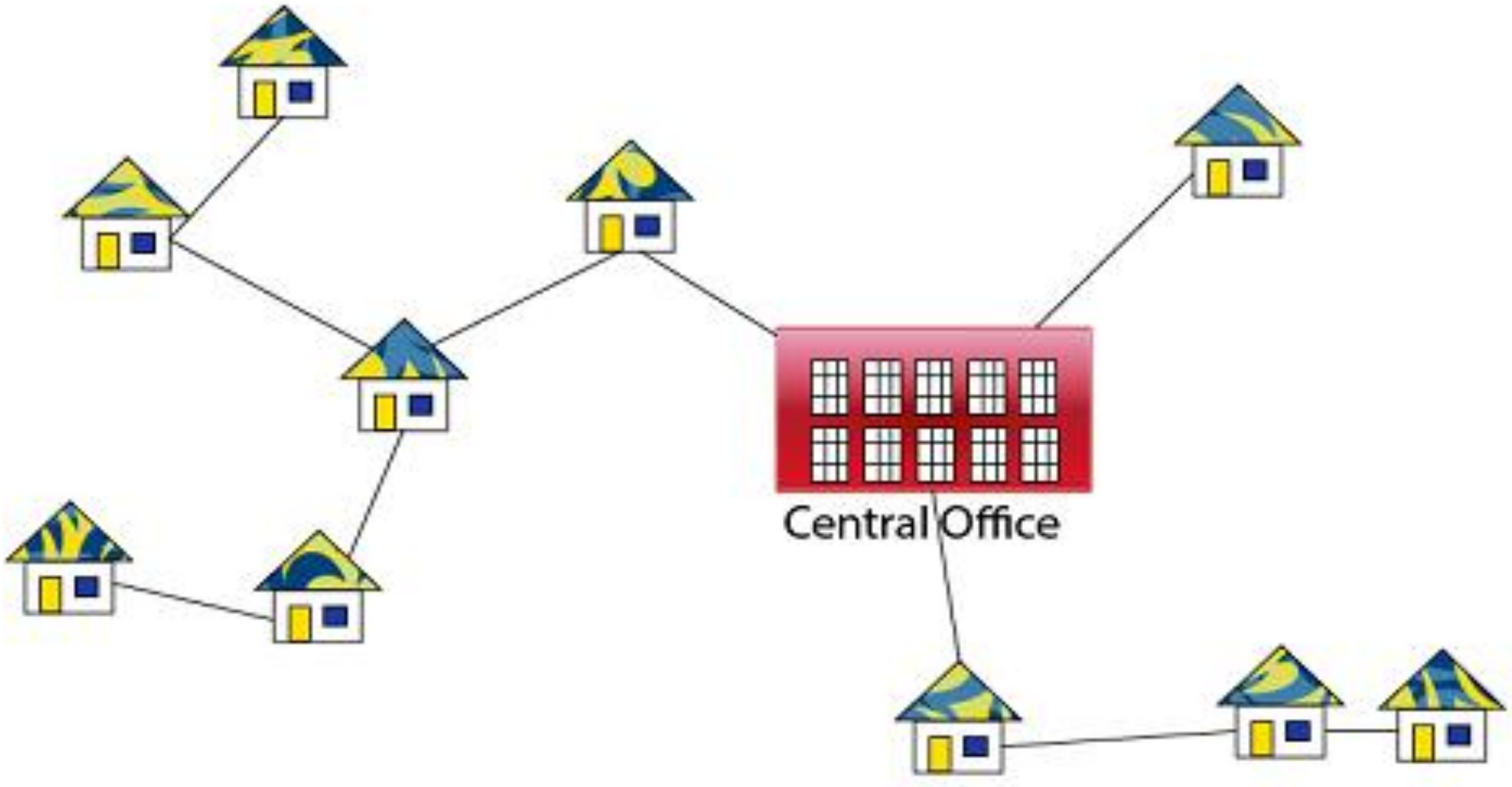
- We use a |V| x |V| matrix for Adjacency Matrix representation.
- Requires Θ (V^2) space.
- Requires Θ (1) time to check if (u,v) is in E.

Cable Laying Problem

Cable Laying Problem

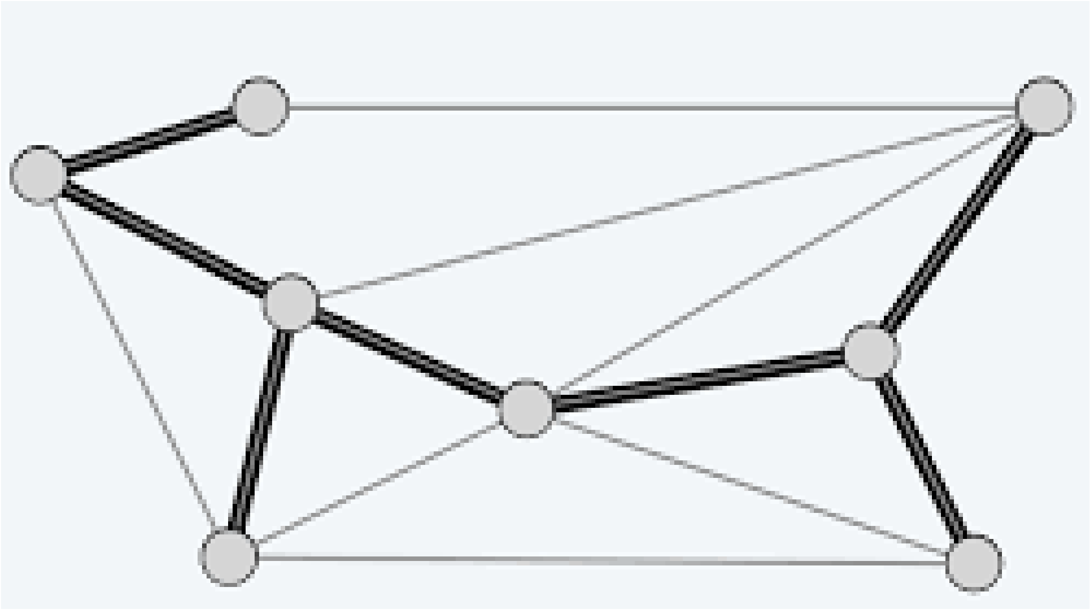


Cable Laying Problem



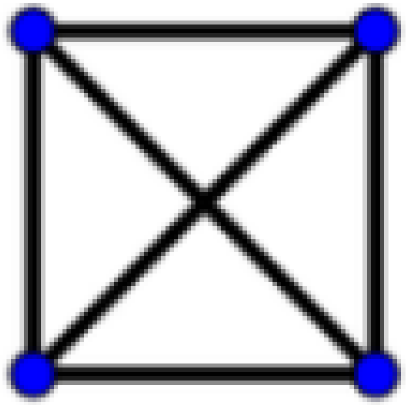
Spanning Tree

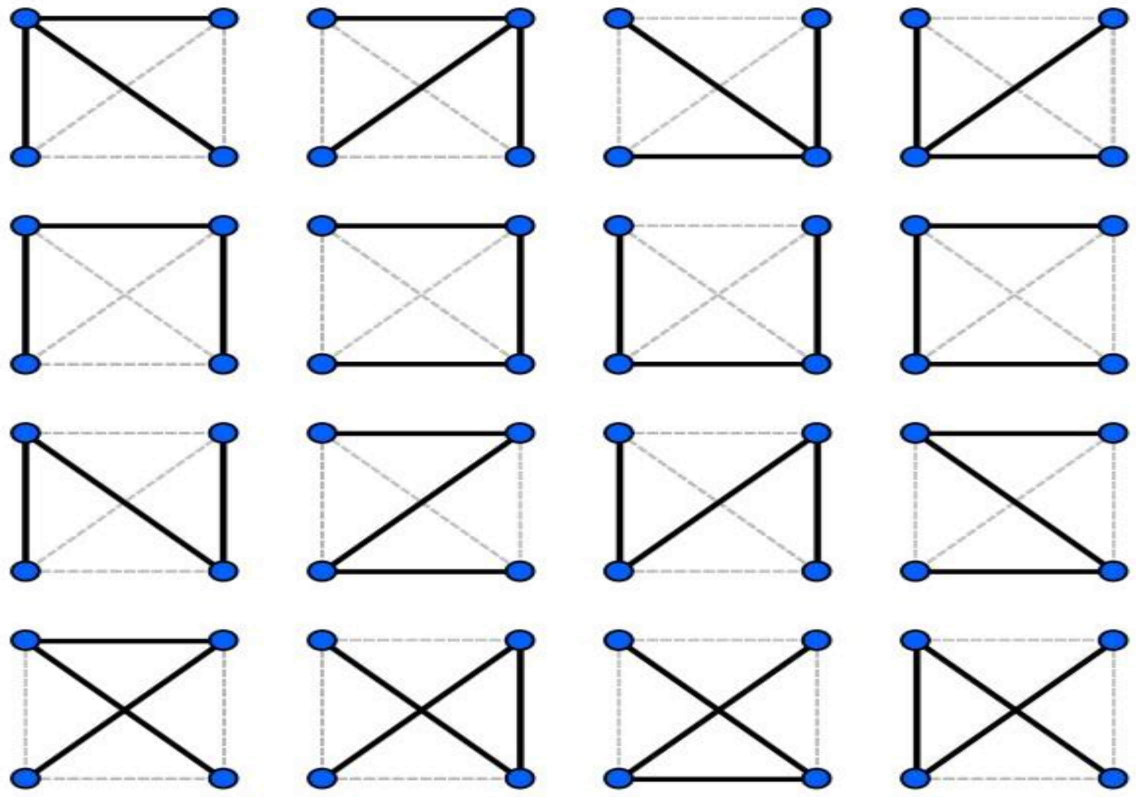
- A **spanning tree** of a graph is a tree containing all vertices from the graph.



Spanning Tree

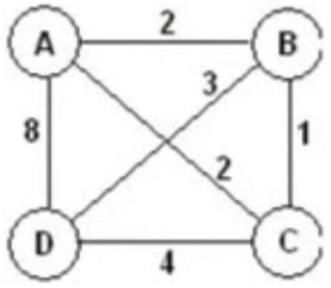
- A **spanning tree** of a graph is a tree containing all vertices from the graph.



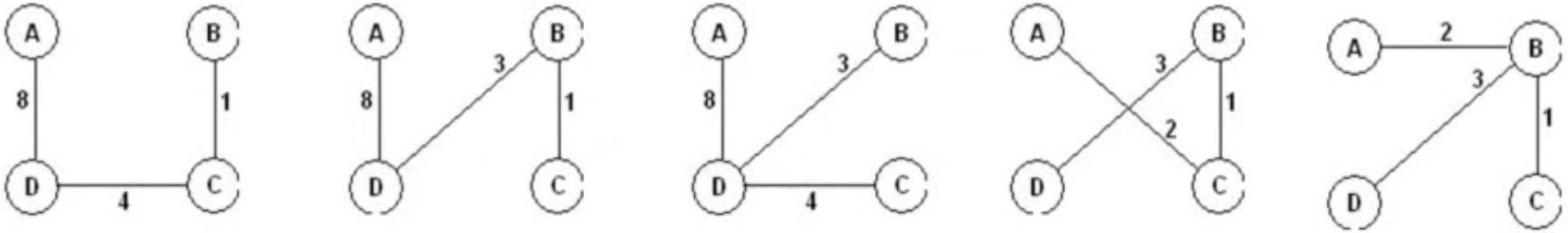


Minimum Spanning Tree

- A **minimum spanning tree** is a spanning tree, where the sum of the weights on the tree's edges are minimal.



- Has 16 possible Spanning Trees, some of them are,

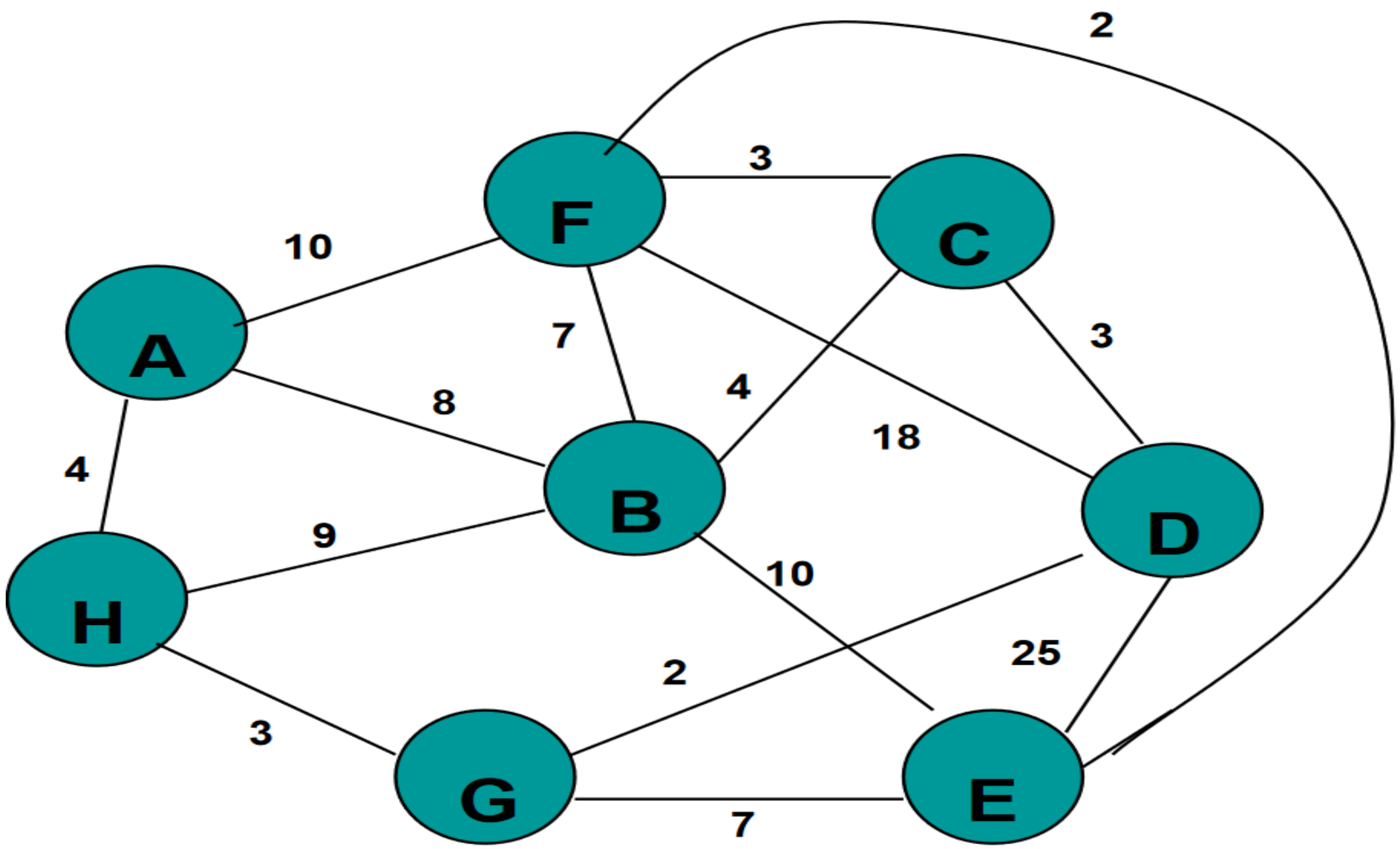


- Has two minimum cost spanning Tree, each with a cost of 6:

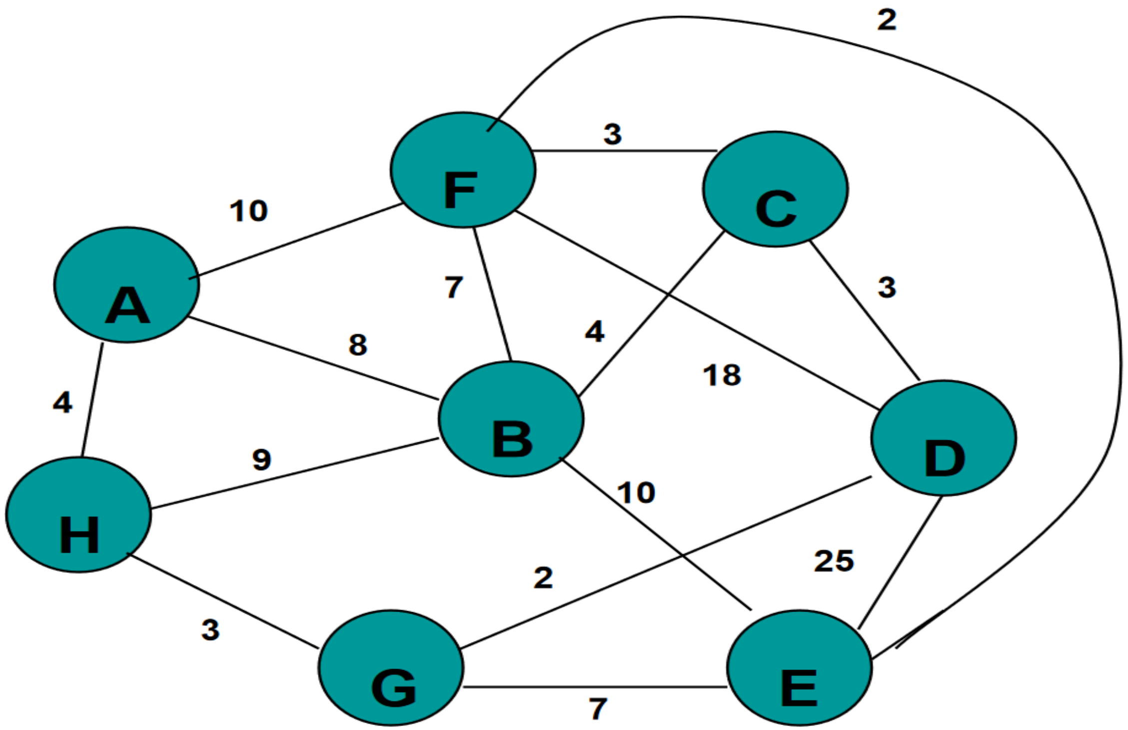
Prim's Algorithm

- Randomly pick a Vertex as the initial Tree T.
- Gradually expand into a Minimum Spanning Tree:
  - For each vertex that is not in T but directly connected to some node in T
    - Compute its minimum distance to any vertex in T.
  - Select the vertex that is closest to T
    - Add it to T.

Prim’s Algorithm



Prim’s Algorithm



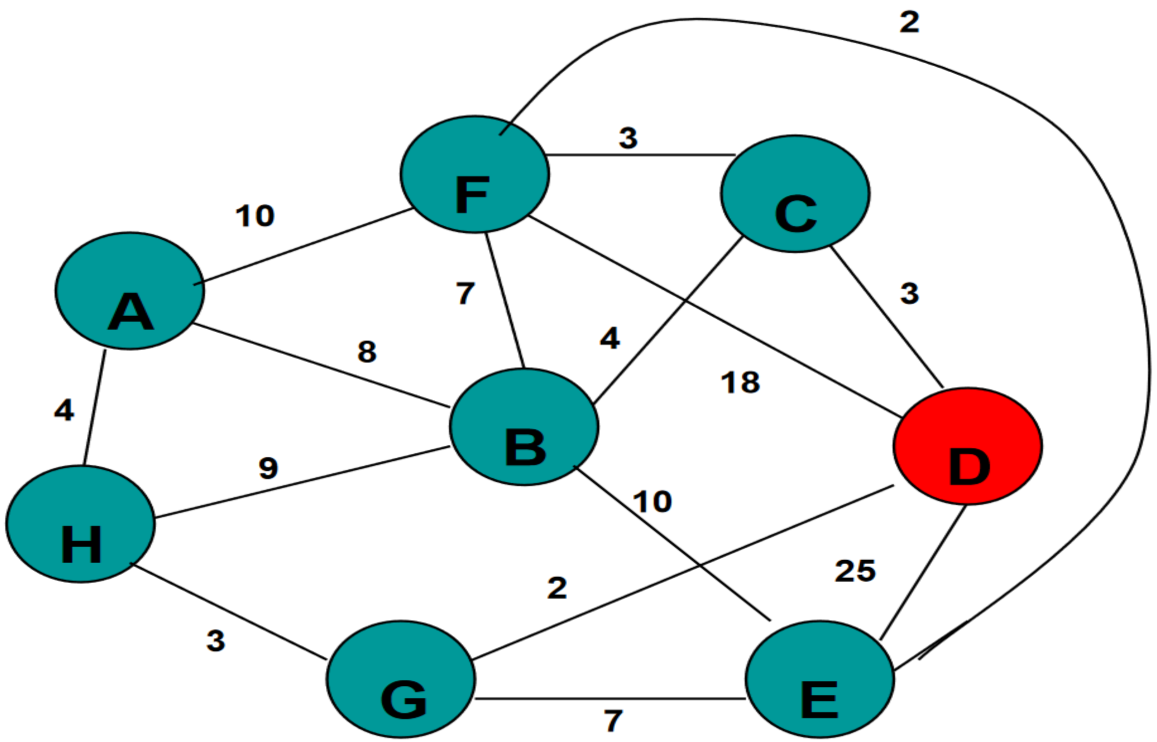
	K	d_v	p_v
A	F	∞	–
B	F	∞	–
C	F	∞	–
D	F	∞	–
E	F	∞	–
F	F	∞	–
G	F	∞	–
H	F	∞	–

K : whether in the tree

d_v : distance to the tree

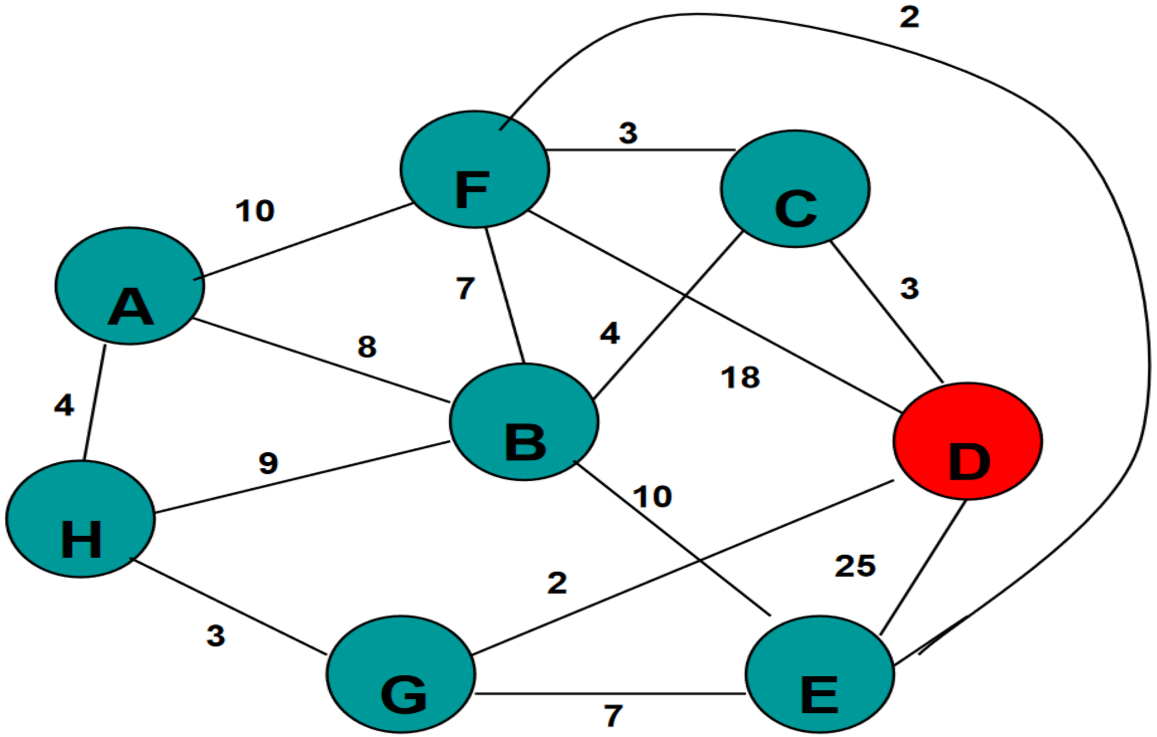
p_v : closest node that is in the tree

Prim’s Algorithm



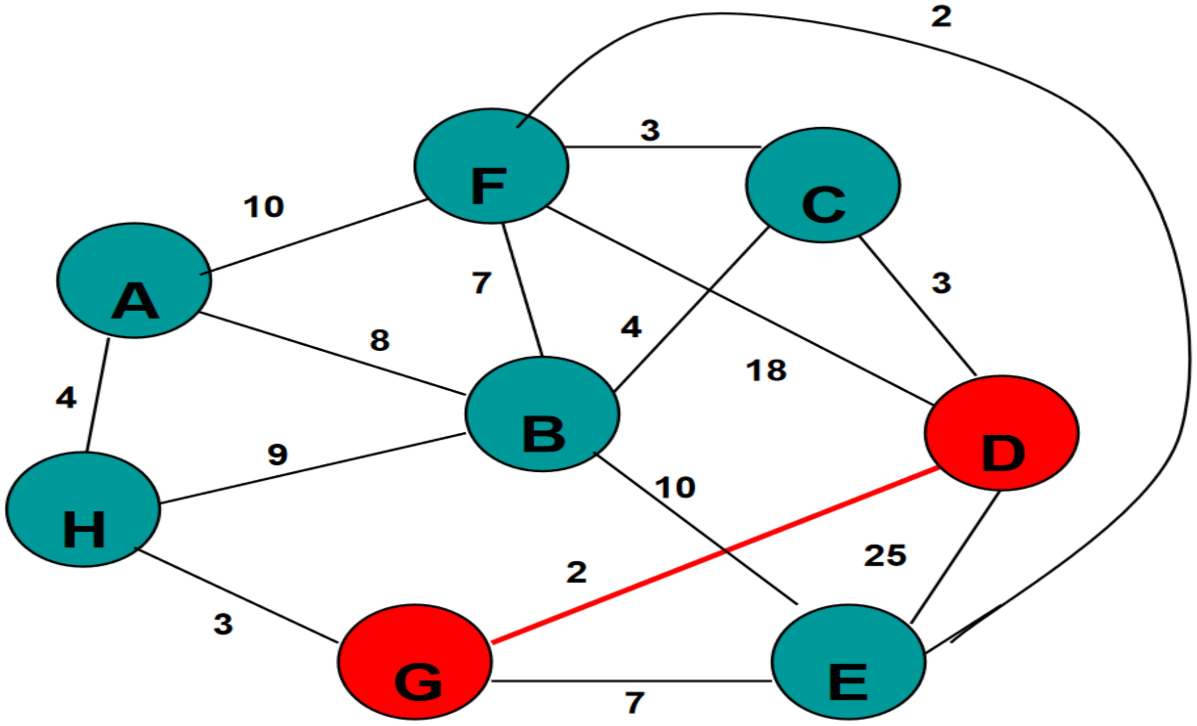
	K	d_v	p_v
A			
B			
C			
D	T	0	–
E			
F			
G			
H			

Prim’s Algorithm



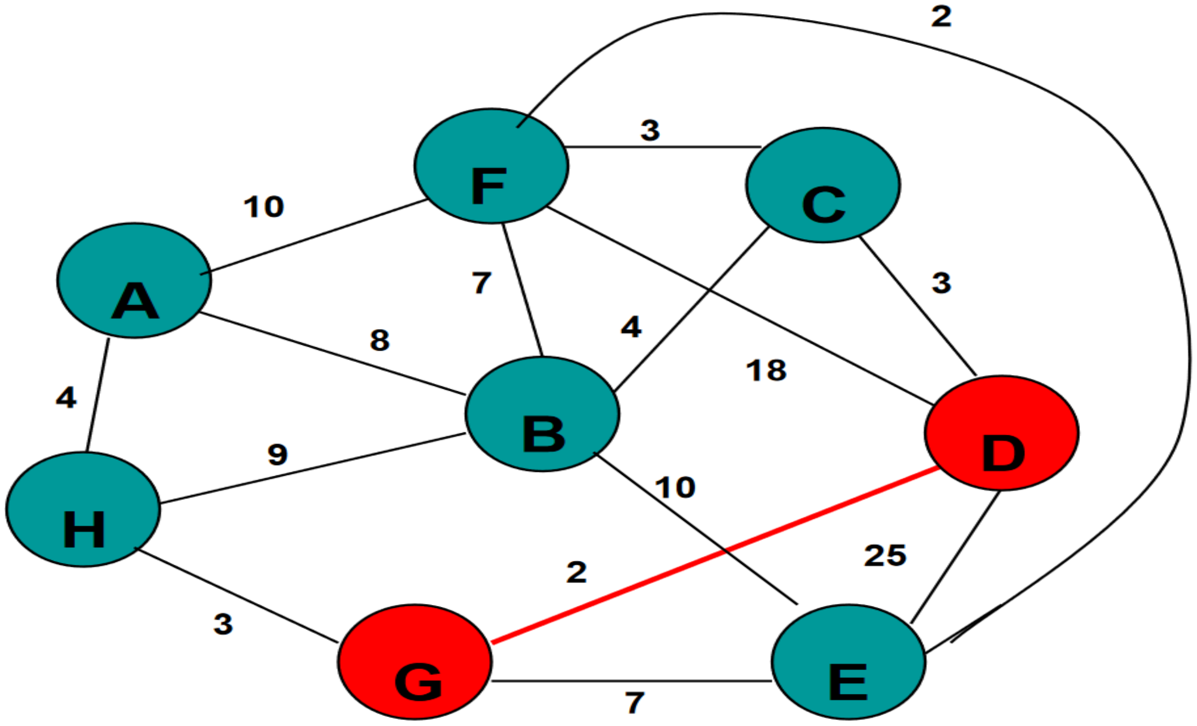
	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G		2	D
H			

Prim's Algorithm



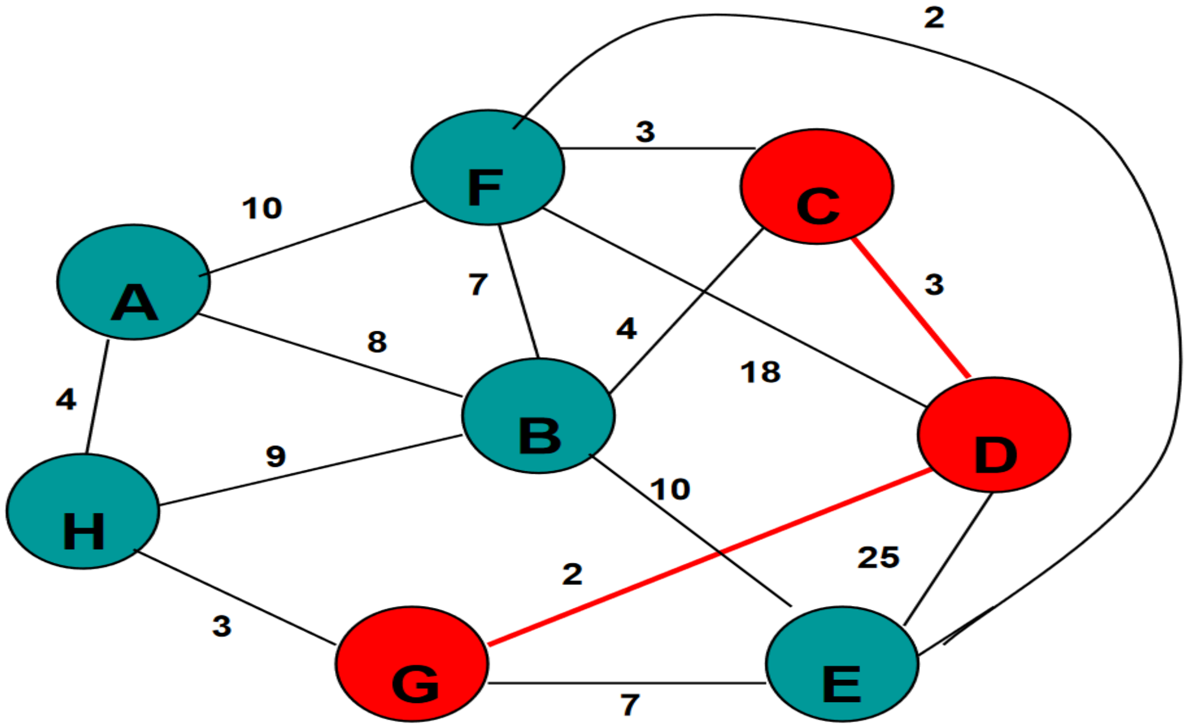
	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G	T	2	D
H			

Prim's Algorithm



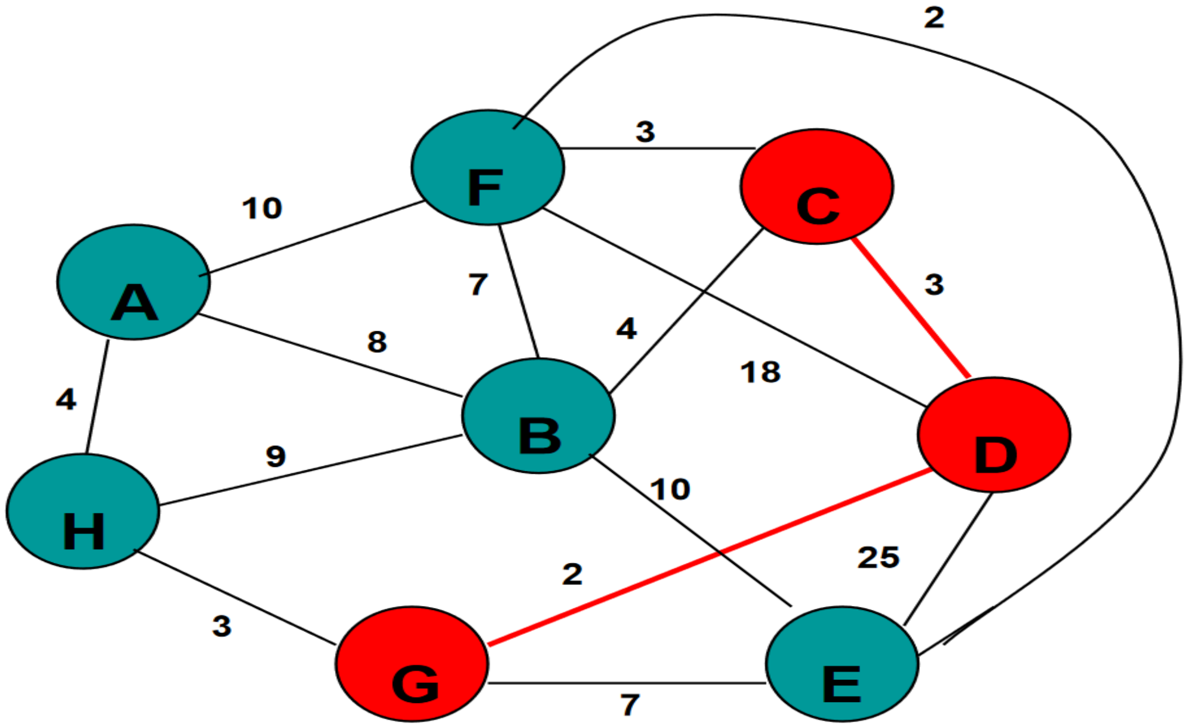
	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G

Prim’s Algorithm



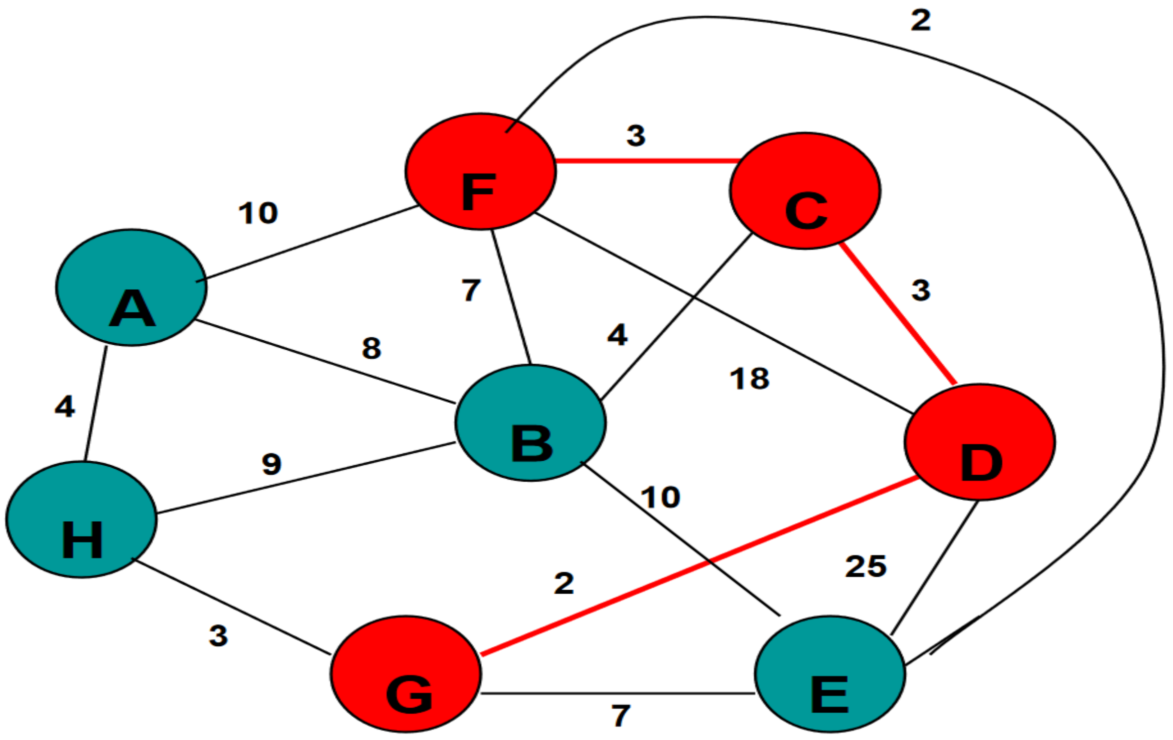
	K	d_v	p_v
A			
B			
C	T	3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G

Prim’s Algorithm



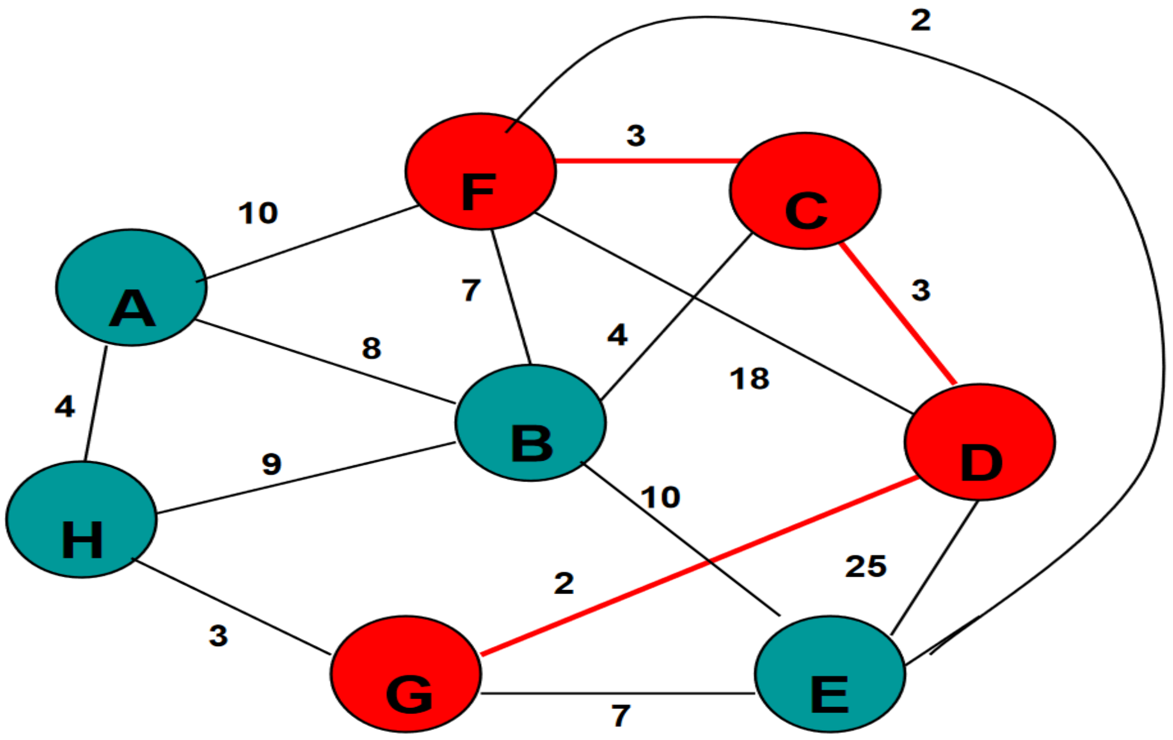
	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F		3	C
G	T	2	D
H		3	G

Prim's Algorithm



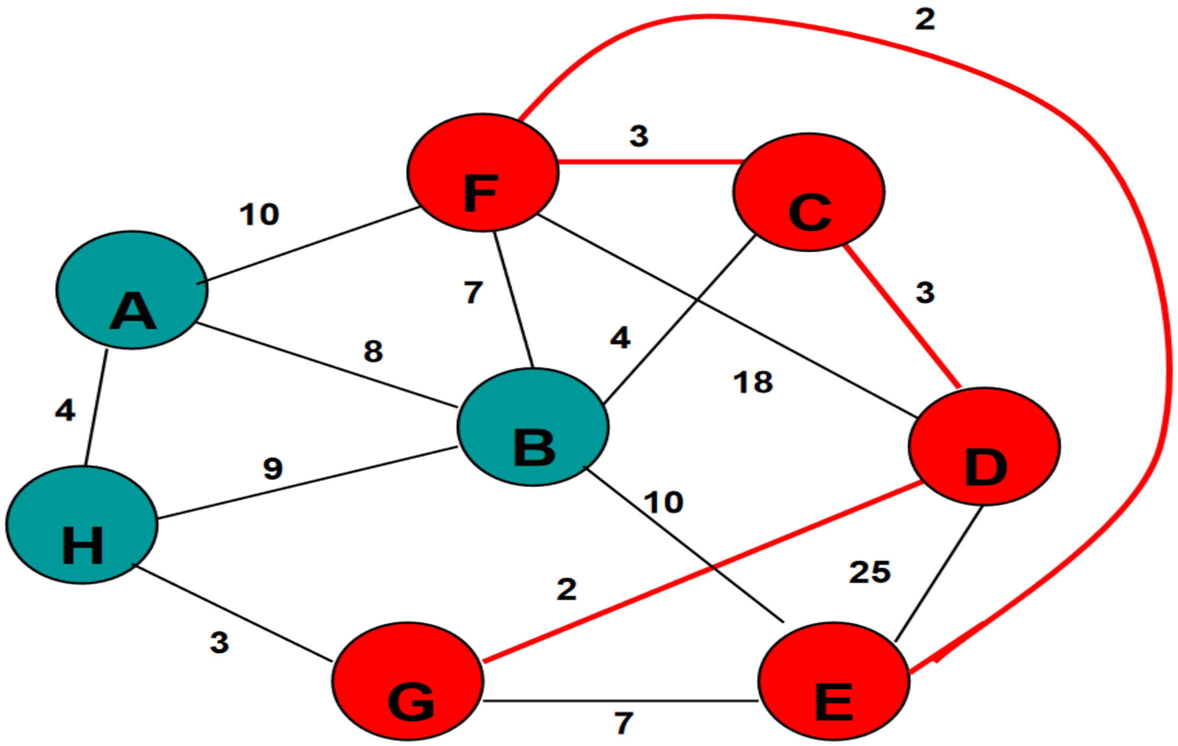
	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F	T	3	C
G	T	2	D
H		3	G

Prim's Algorithm



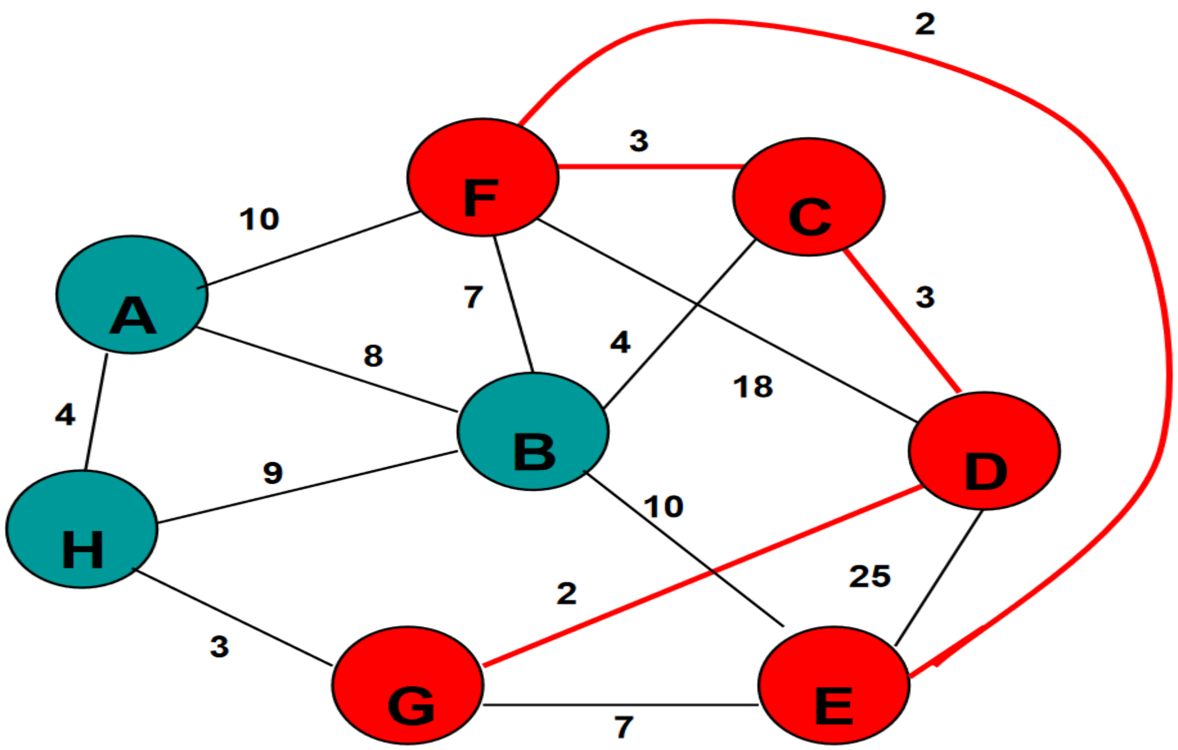
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E		2	F
F	T	3	C
G	T	2	D
H		3	G

Prim’s Algorithm



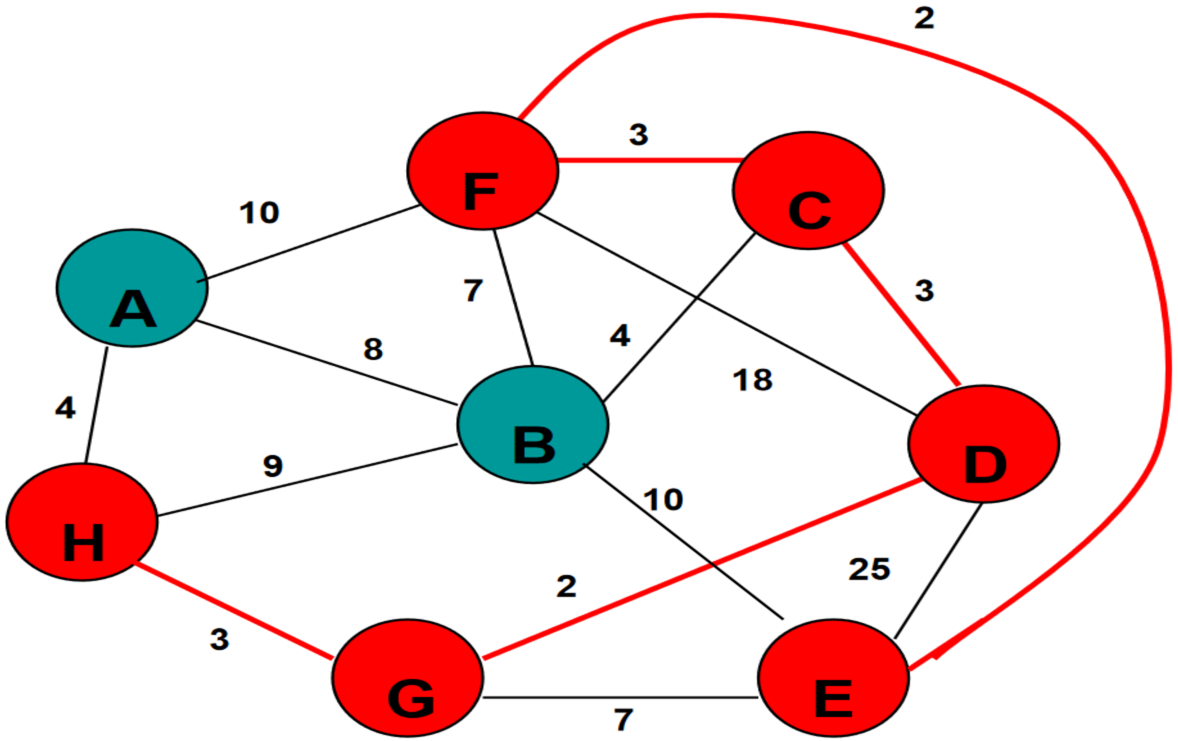
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

Prim's Algorithm



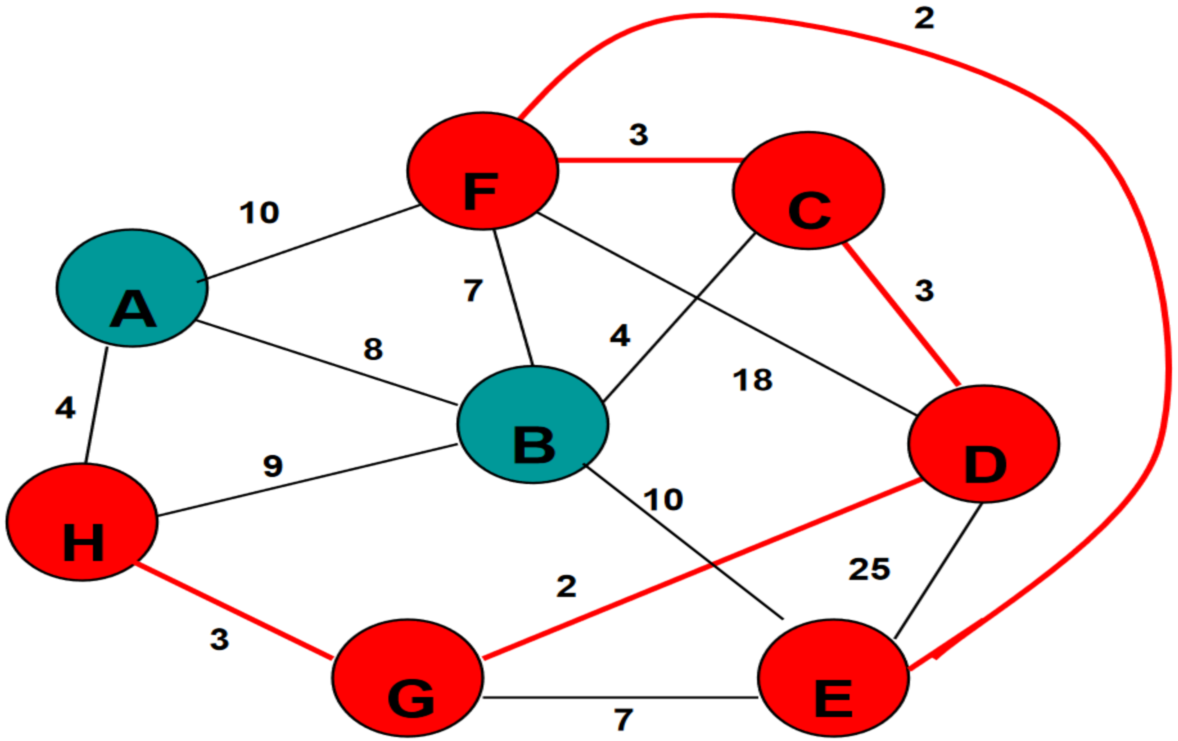
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

Prim’s Algorithm



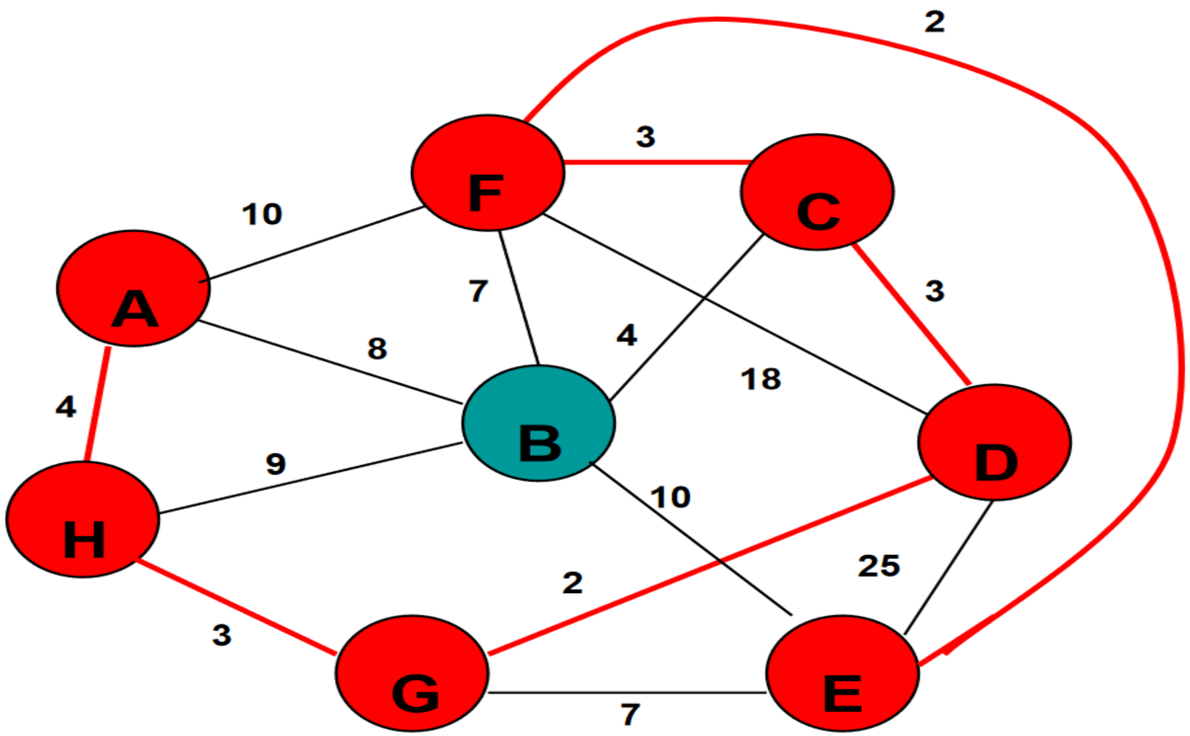
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Prim’s Algorithm



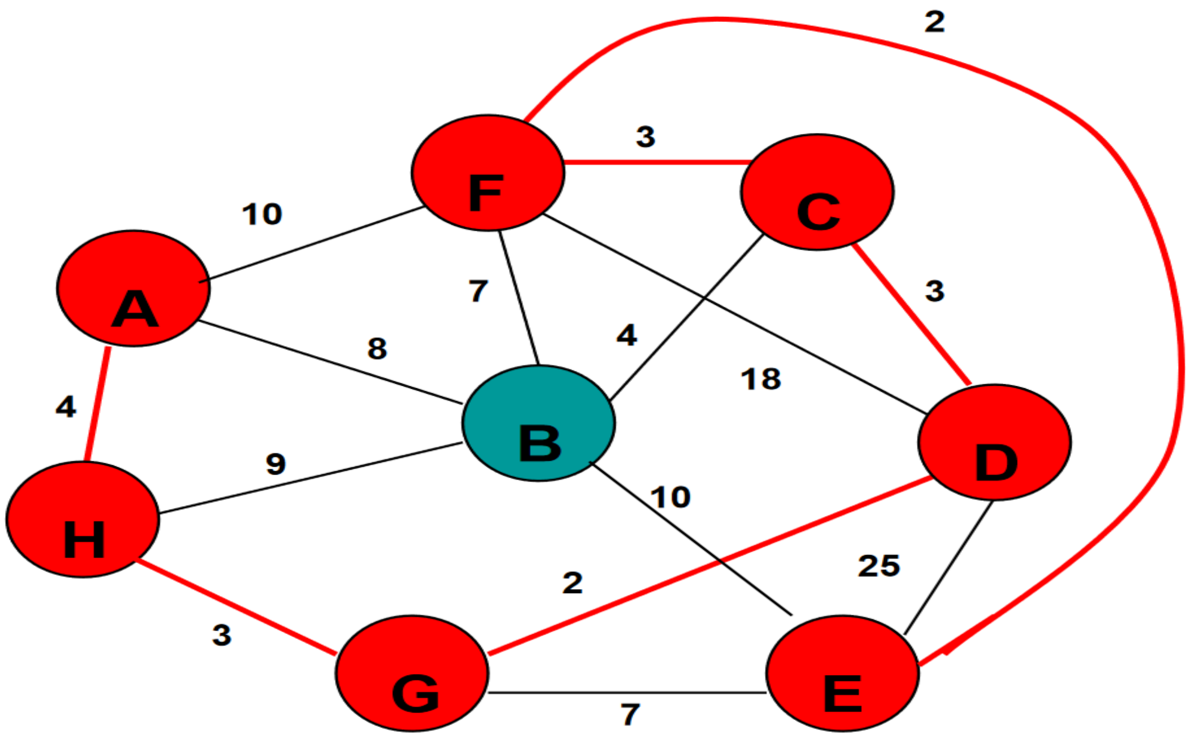
	K	d_v	p_v
A		4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Prim's Algorithm



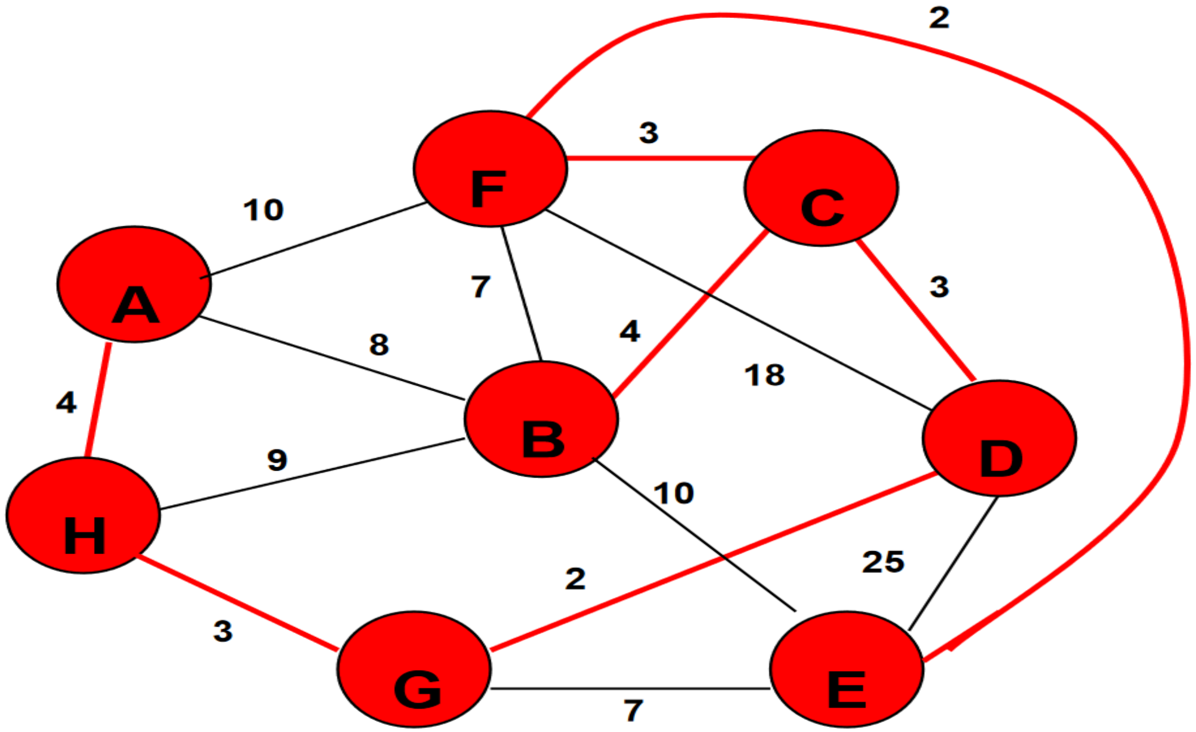
	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Prim’s Algorithm



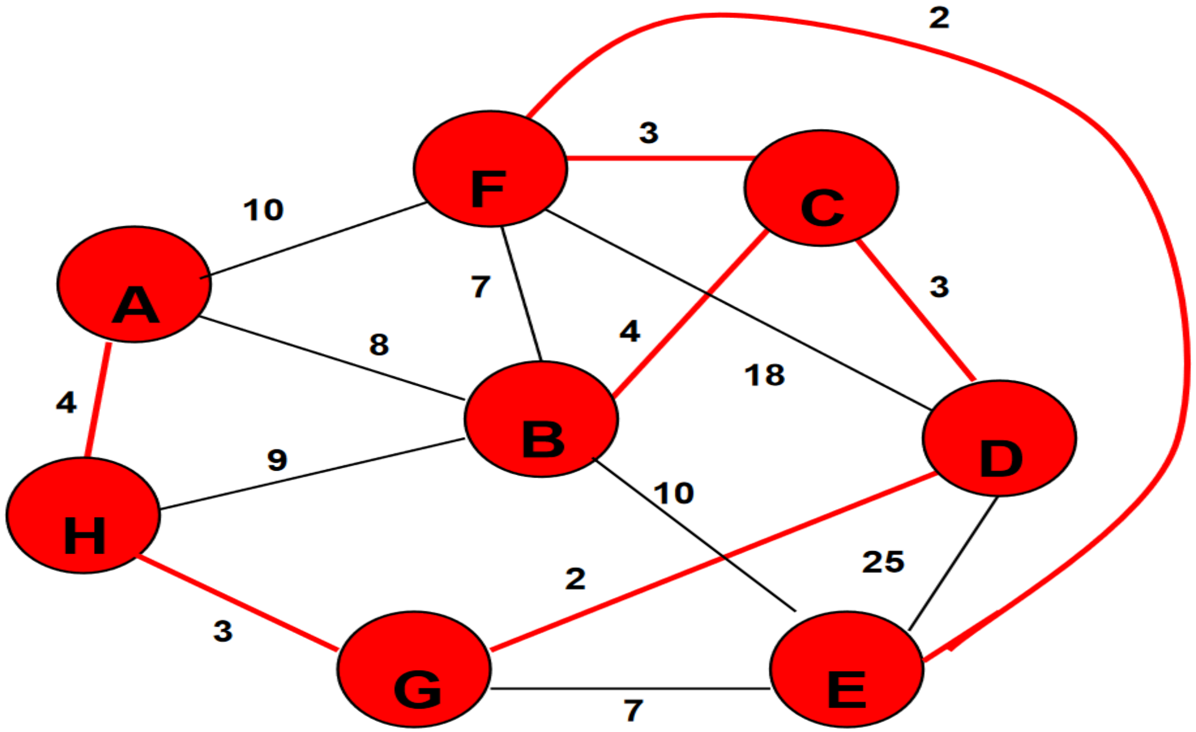
	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Prim’s Algorithm



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

Prim's Algorithm



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

Cost of Minimum Spanning Tree = **21**

Applications of Minimum-Cost Spanning Tree

- Build a cable network which joins ‘n’ locations with minimum cost.
- Logistical problems.
- Obtain independent set of circuit equations for an electrical network.
- Pattern recognition, minimum spanning tree can be used to find noisy pixels.

Algorithm Design Techniques

- Iterative method.
- Divide & Conquer method.
- Greedy method.

Algorithm Design Techniques

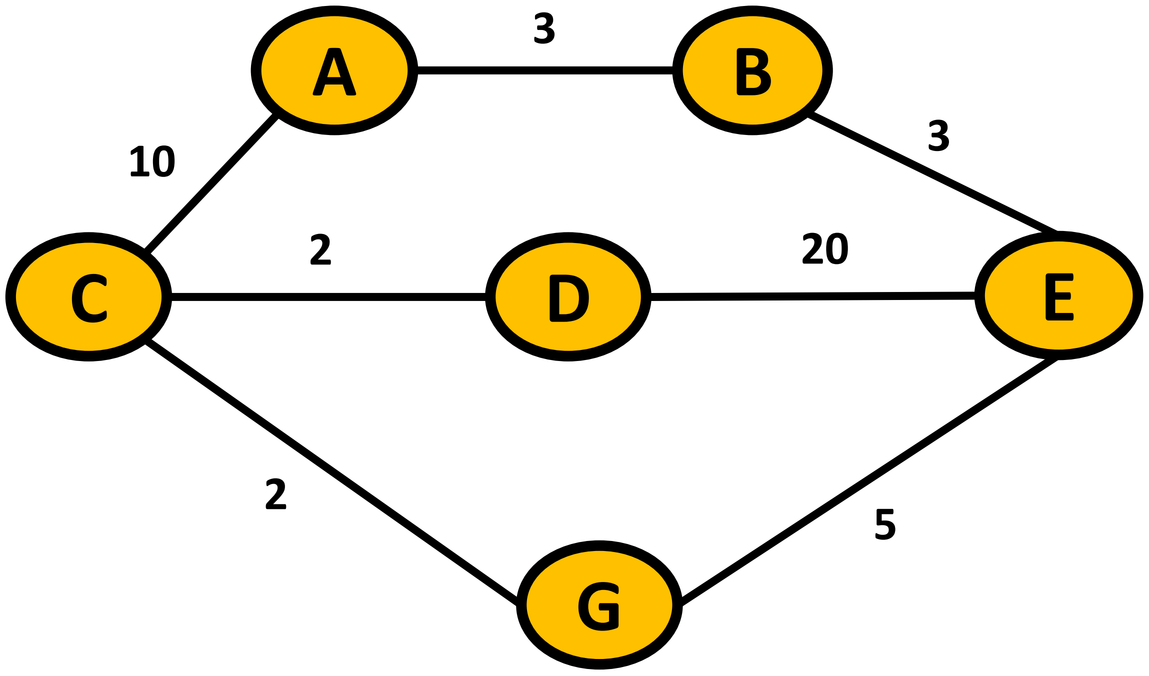
- Greedy method.
- {10, 5, 2, 1}

- May not always work.
- {18, 10, 1}

Algorithm Design Techniques

- Greedy method.

- May not always work.



Algorithm Design Techniques

- Iterative method.
- Divide & Conquer method.
- Greedy method.
- Dynamic Programming.

Dynamic Programming

- Dynamic Programming.
- Hope you remember what recursion is?
- Reuse the solutions of the sub-problem
  - Memoization [Top-Down]
  - Tabulation [Bottom-up]

Dynamic Programming Applications

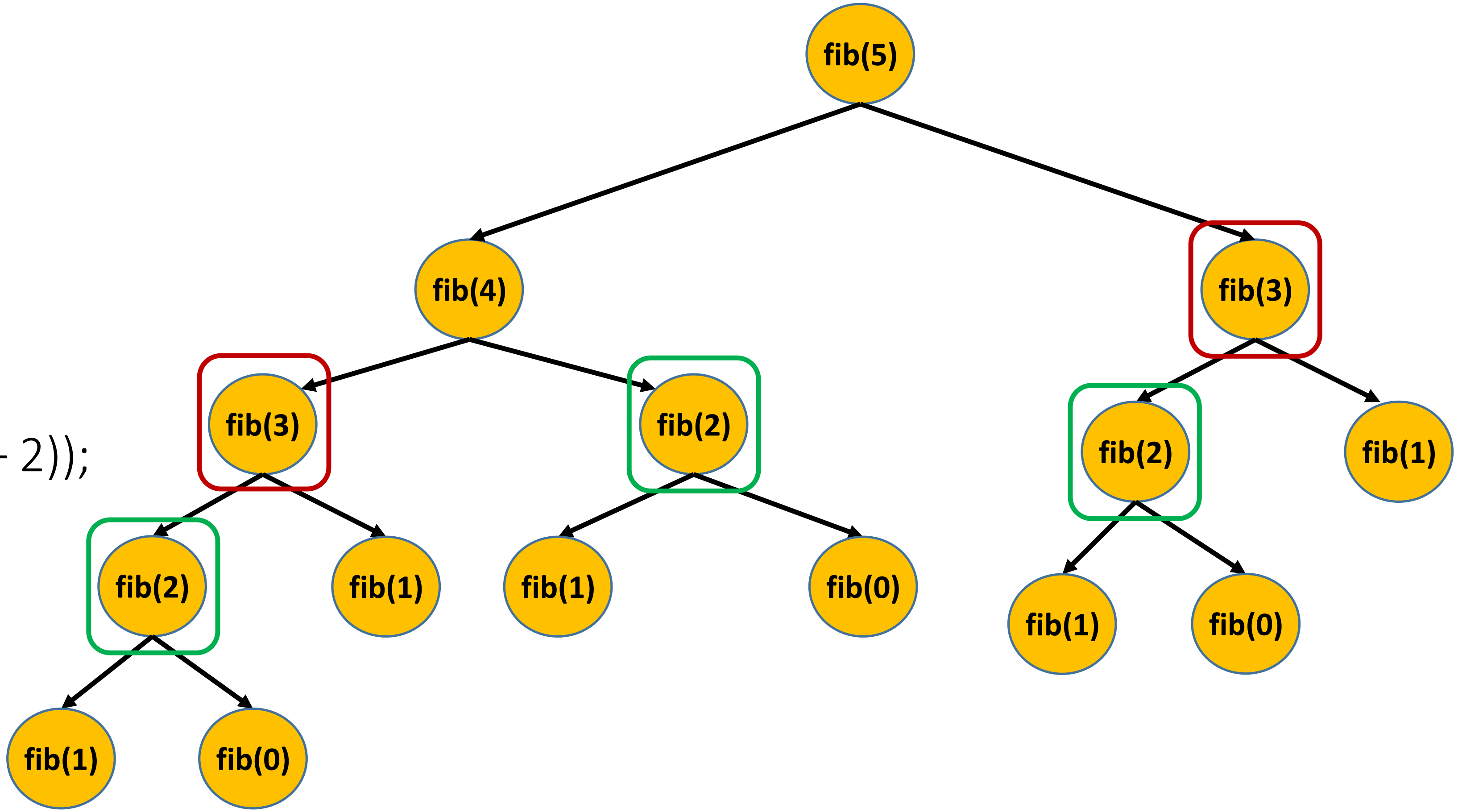
- Bellman Ford Algorithm
  - Single source shortest path algorithm
- Floyd Warshall Algorithm
  - All pair shortest path algorithm.
- Different utility:
  - Version control systems.
- Search close words.
  - Edit distance.

Dynamic Programming

- Fibonacci Series

<u>Snippet 5</u>

```
int fib (int n)
{
    if (n == 0 || n == 1)
        return n;
    return (fib (n – 1) + fib (n – 2));
}
```



Dynamic Programming: Fibonacci Series

Solution (Divide & Conquer)

```
int fib (int n)
{
    if (n == 0 || n == 1)
        return n;
    return (fib (n – 1) + fib (n – 2));
}
```

```
int memo [n] = { -1, -1, -1, ……… -1}
```

Solution (Dynamic Programming)

```
int fib (int n)
{
    if (memo[n] == -1)
    {   int res;
        if (n == 0 || n == 1);
            res = n;
        else
            res = (fib (n – 1) + fib (n – 2));
            memo [n] = res;
    }
    return memo [n];
}
```

Dynamic Programming: Fibonacci Series

-1	-1	-1	-1	-1	-1	initialize
-1	1	-1	-1	-1	-1	fib (1)
0	1	-1	-1	-1	-1	fib (0)
0	1	1	-1	-1	-1	fib (2)
0	1	1	2	-1	-1	fib (3)
0	1	1	2	3	-1	fib (4)
0	1	1	2	3	5	fib (5)