Minimum Spanning Trees

Definition

- A Minimum Spanning Tree (MST) is a subgraph of an undirected graph such that the subgraph spans (includes) all nodes, is connected, is acyclic, and has minimum total edge weight

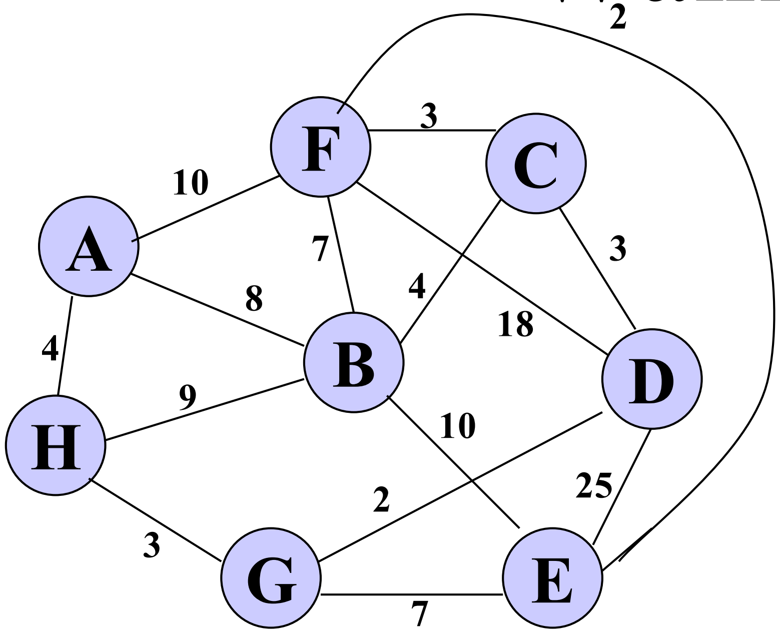
Algorithm Characteristics

- Both Prim’s and Kruskal’s Algorithms work with undirected graphs
- Both work with weighted and unweighted graphs but are more interesting when edges are weighted
- Both are greedy algorithms that produce optimal solutions

Prim’s Algorithm

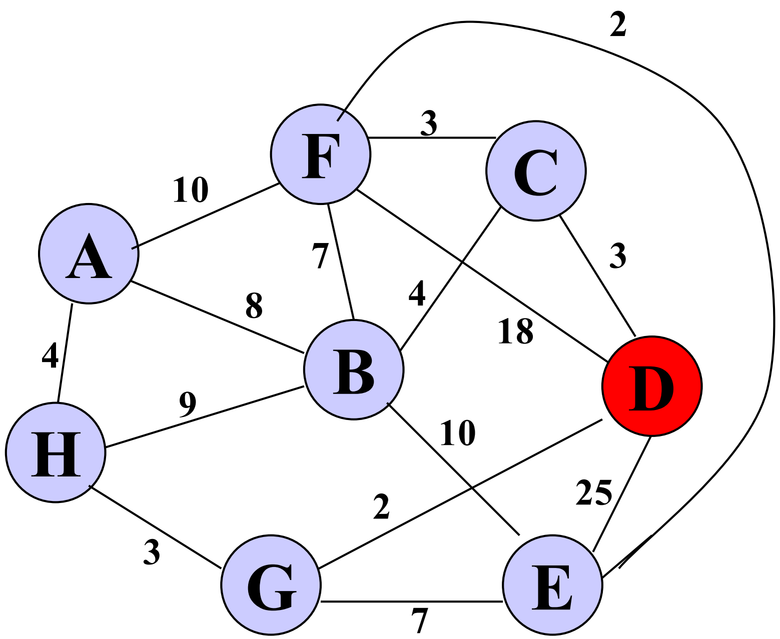
- Similar to Dijkstra’s Algorithm (shortest path algorithm) except that d_v records edge weights, not path lengths

Walk-Through



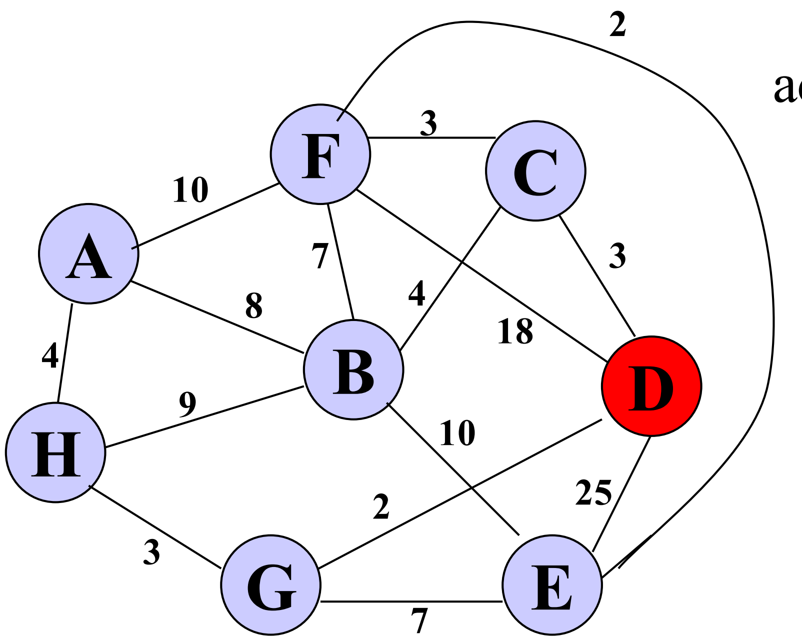
Initialize array

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



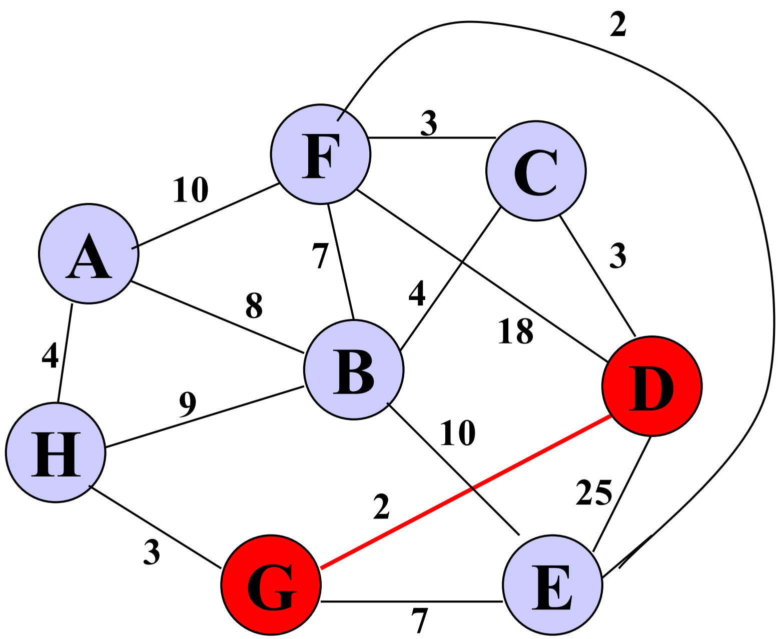
Start with any node, say D

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



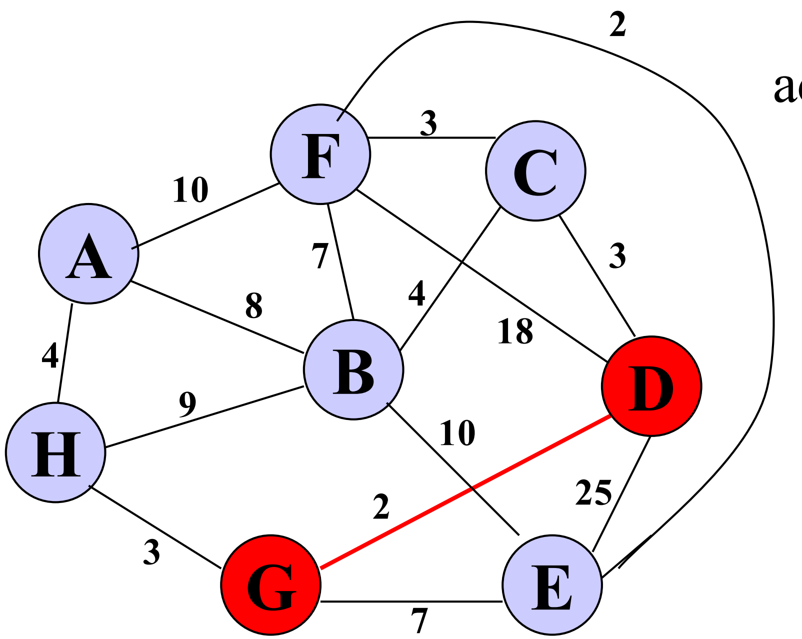
Update distances of adjacent, unselected nodes

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



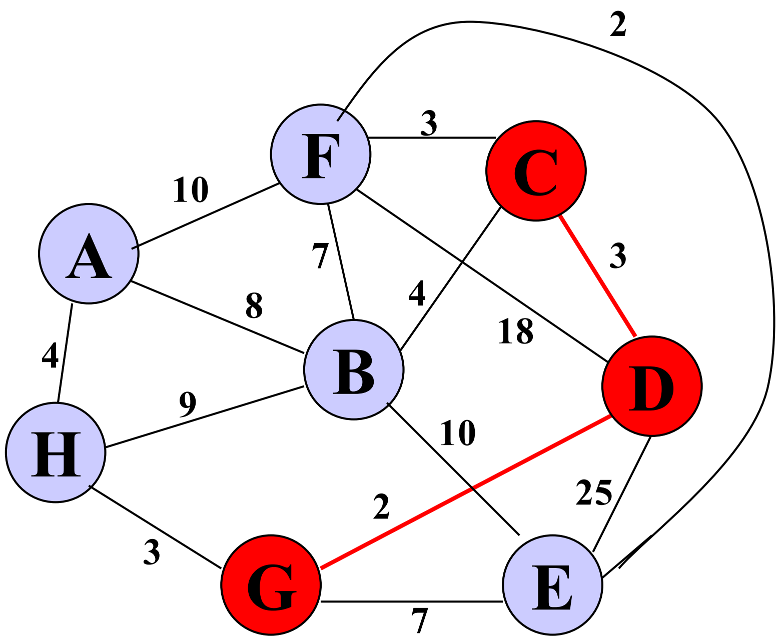
Select node with minimum distance

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



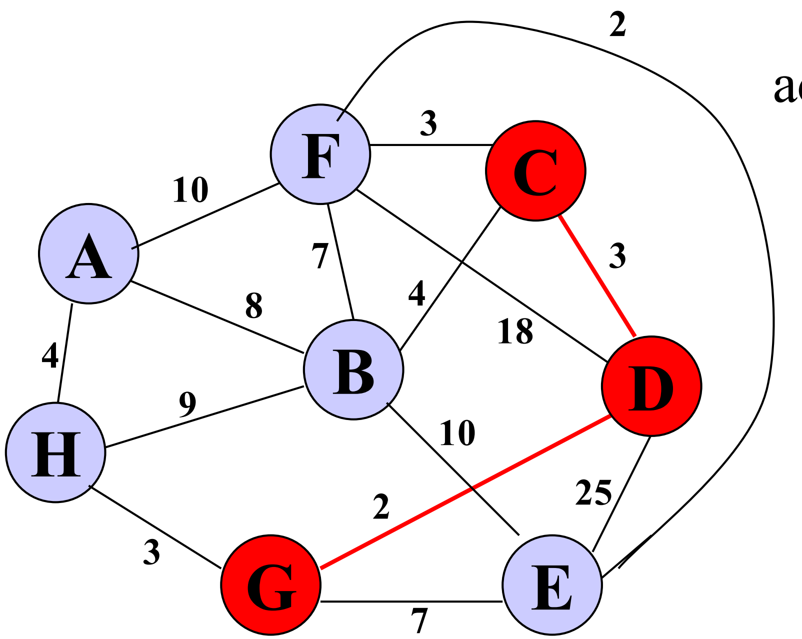
Update distances of adjacent, unselected nodes

	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



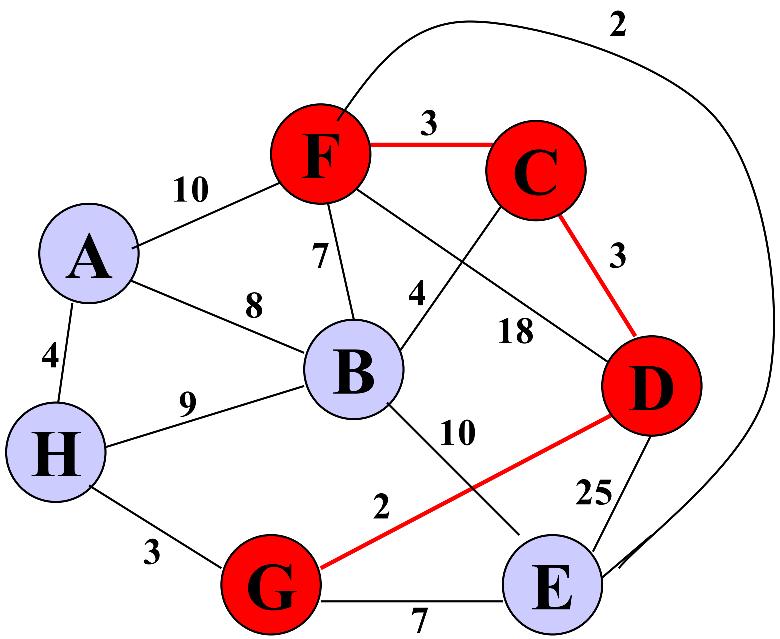
Select node with minimum distance

	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



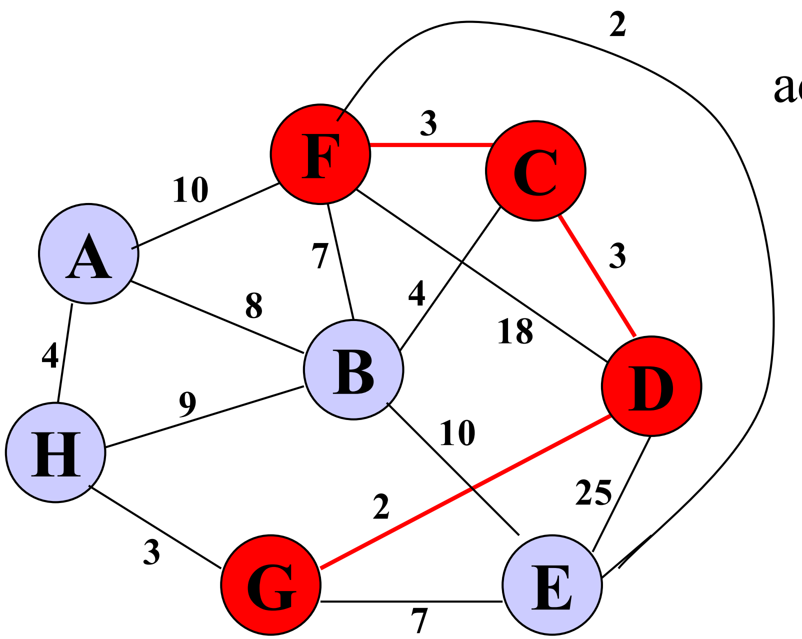
Update distances of adjacent, unselected nodes

	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



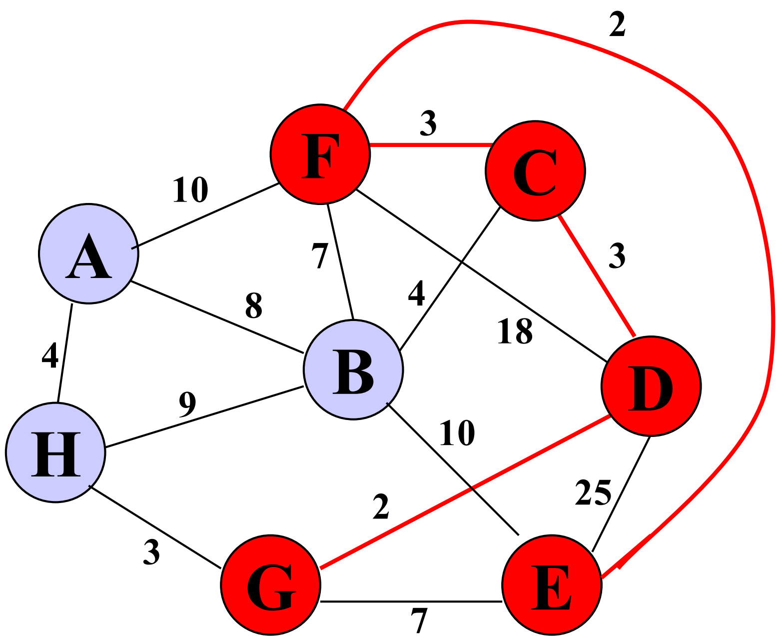
Select node with minimum distance

	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



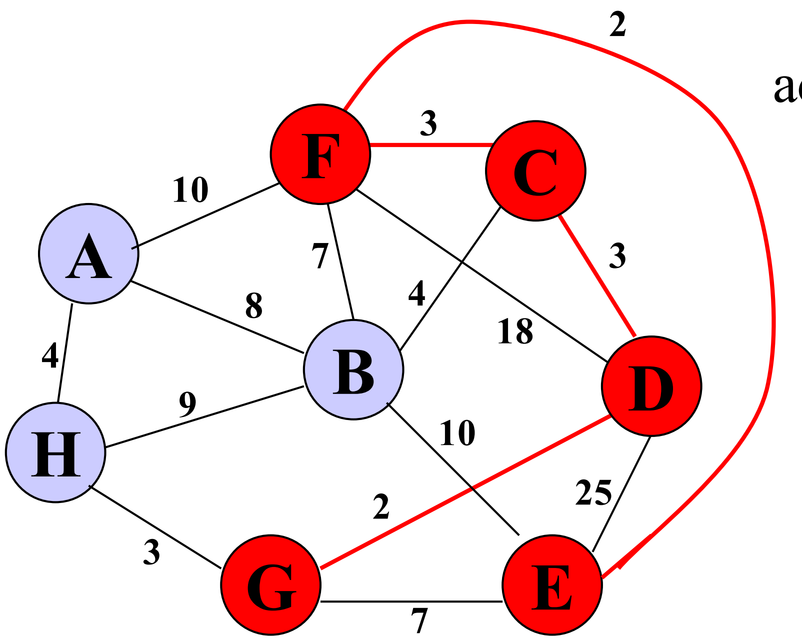
Update distances of adjacent, unselected nodes

	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



Select node with minimum distance

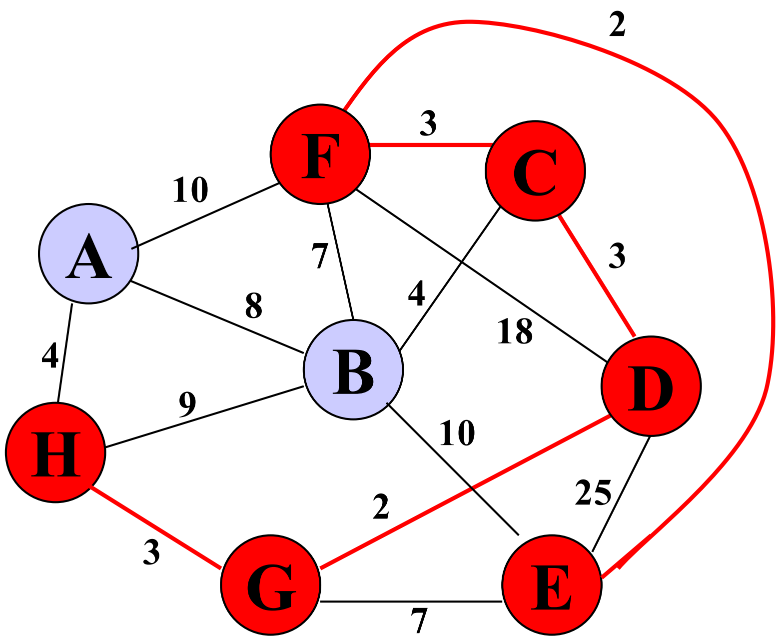
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G



Update distances of
adjacent, unselected nodes

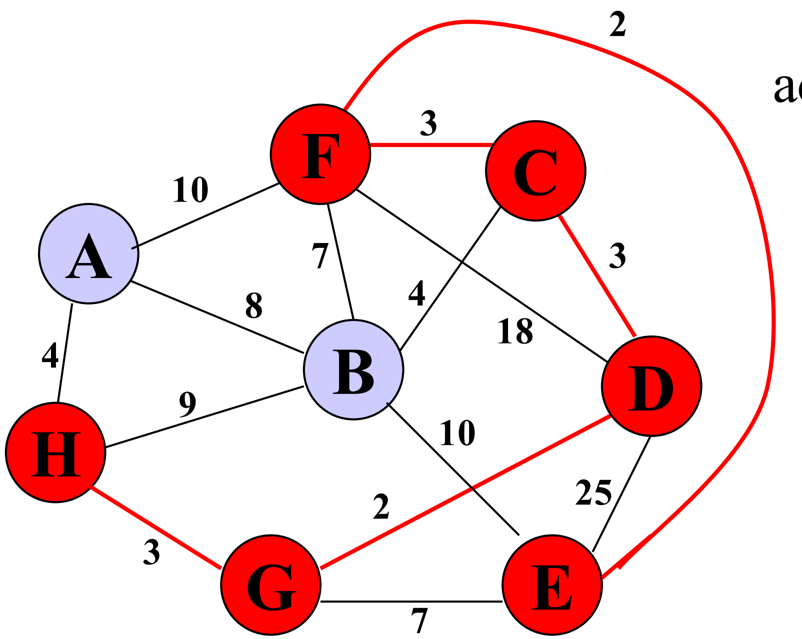
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

Table entries unchanged



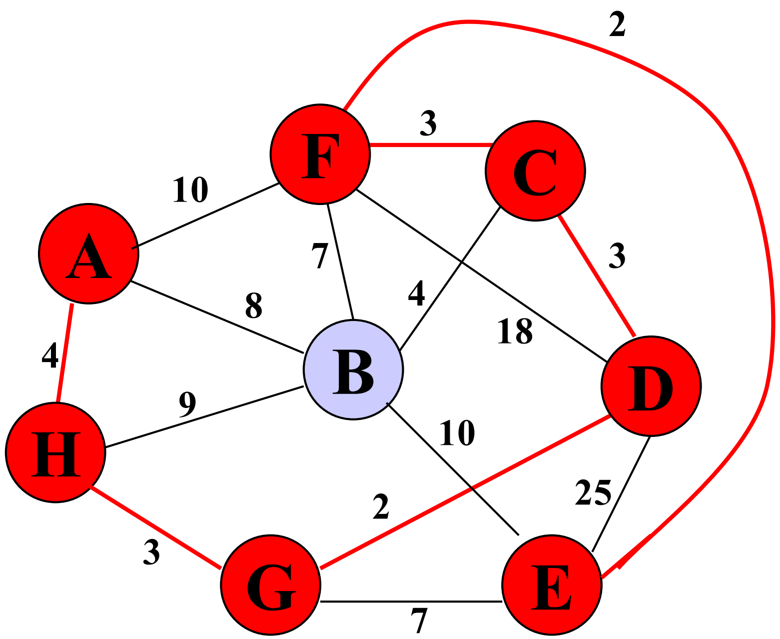
Select node with minimum distance

	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



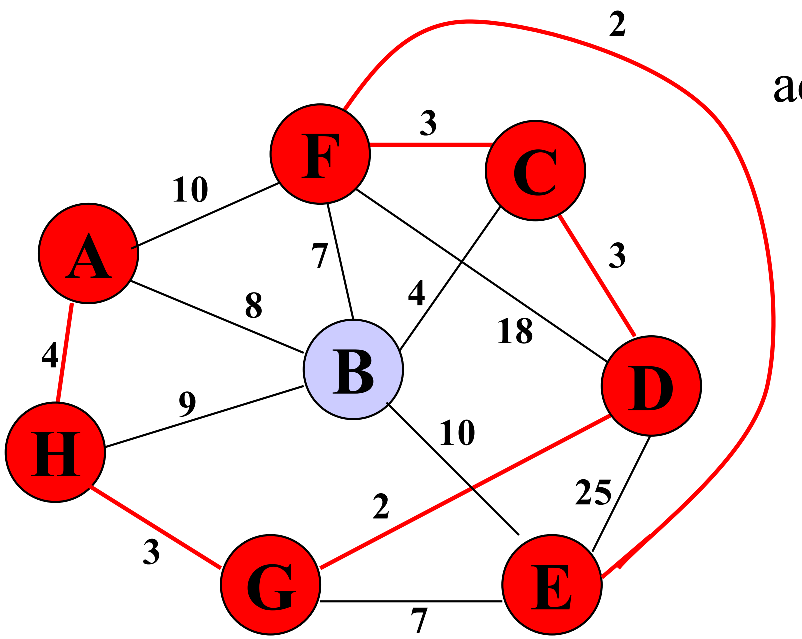
Update distances of adjacent, unselected nodes

	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



Select node with minimum distance

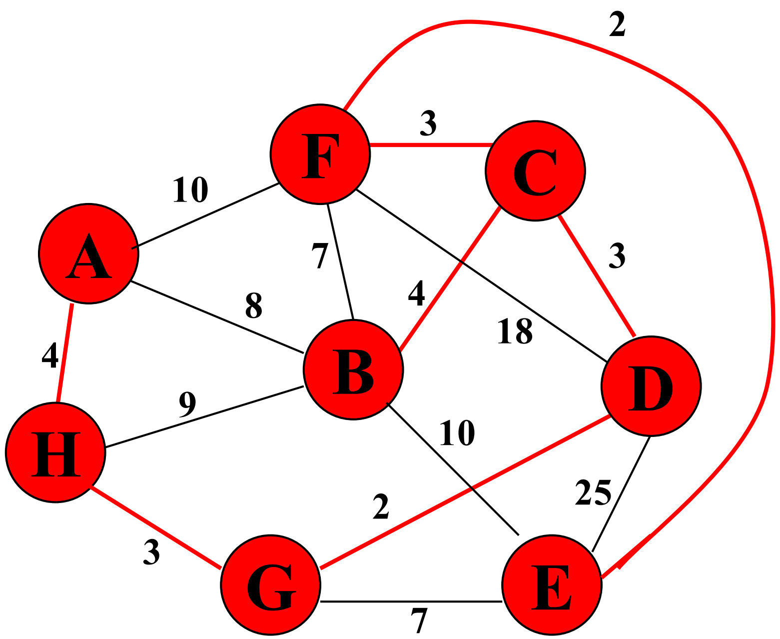
	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G



Update distances of adjacent, unselected nodes

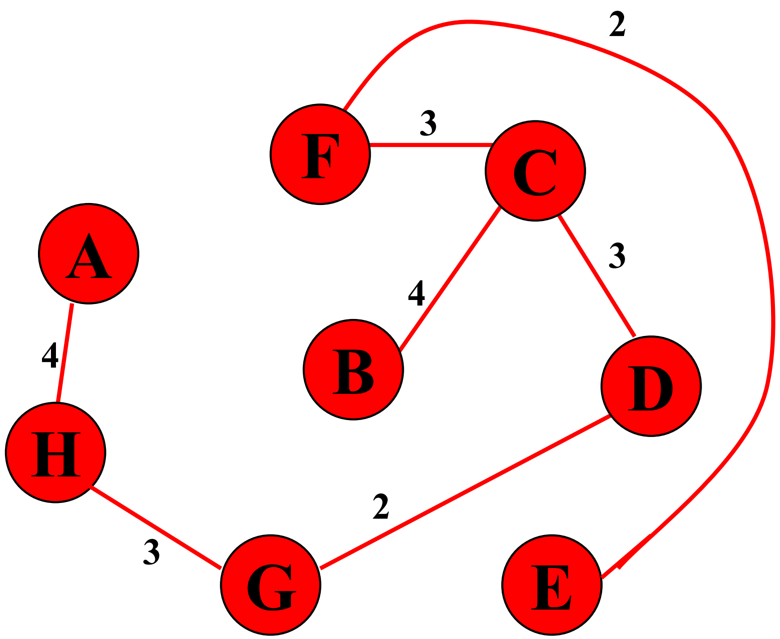
	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Table entries unchanged



Select node with minimum distance

	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 = $\Sigma\ d_v = 21$

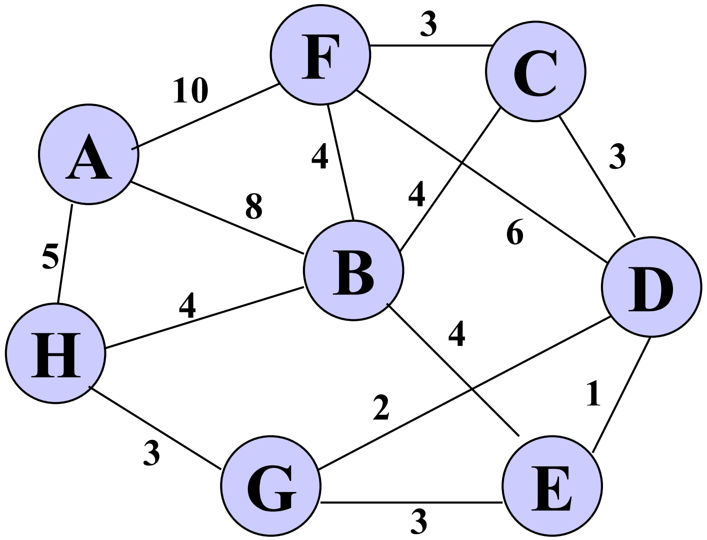
	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Done

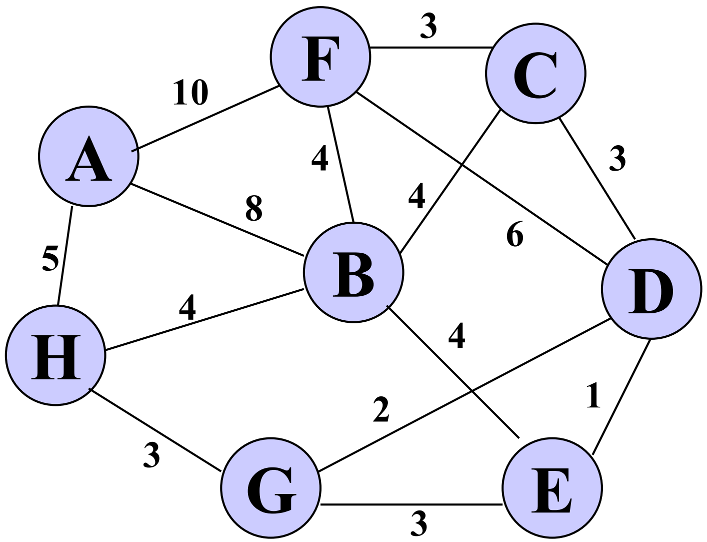
Kruskal’s Algorithm

- 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

Walk-Through



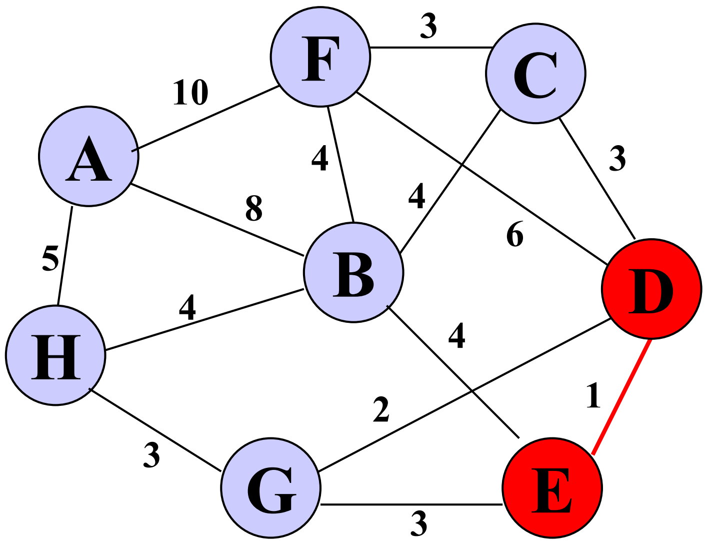
Consider an undirected, weight graph



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	

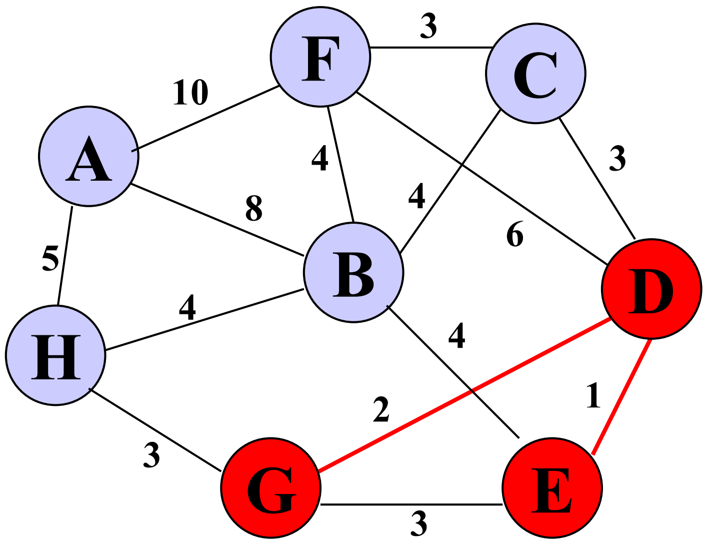
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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	

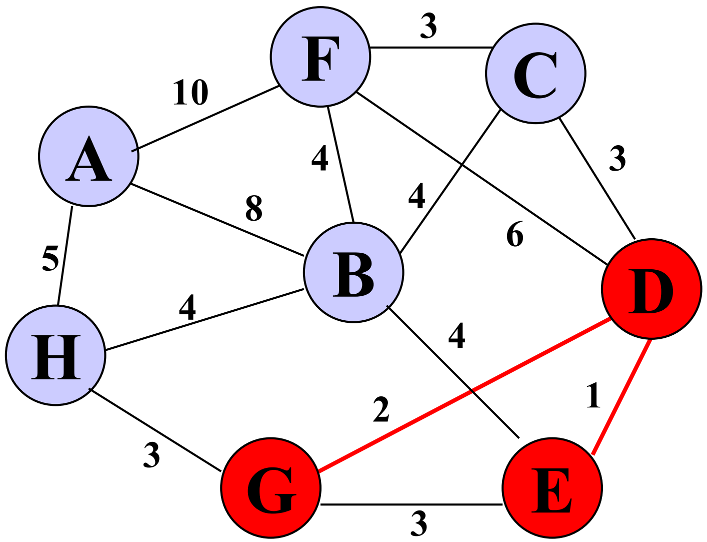
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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	

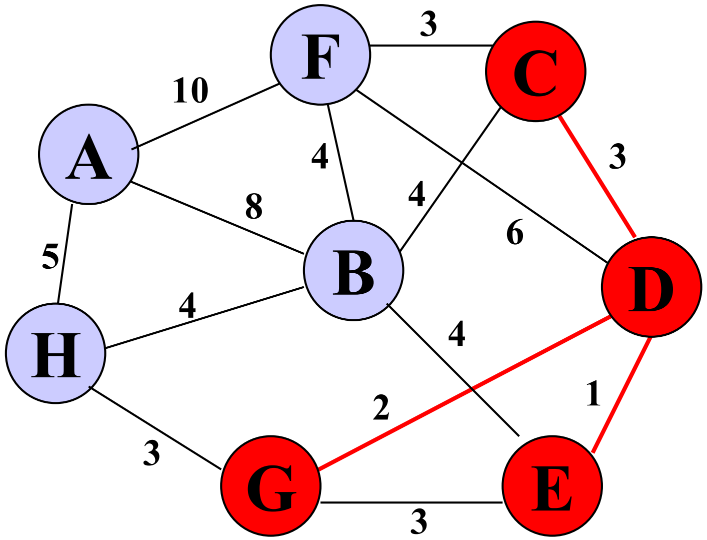


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(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	

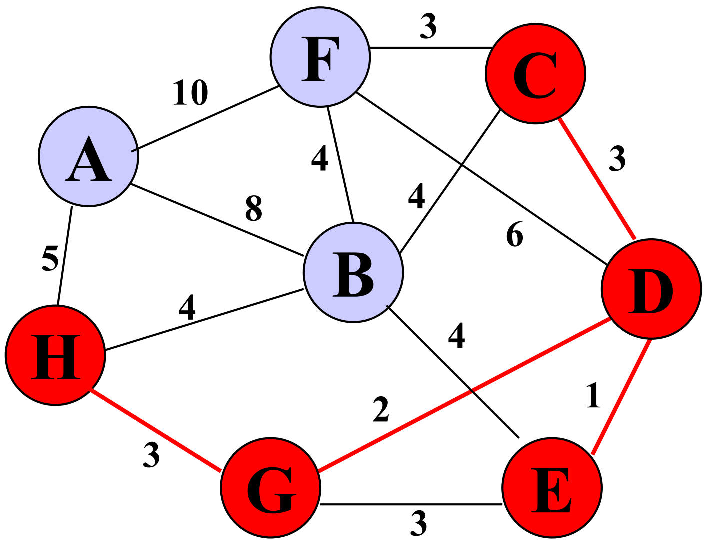
Accepting edge (E,G) would create a cycle



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

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

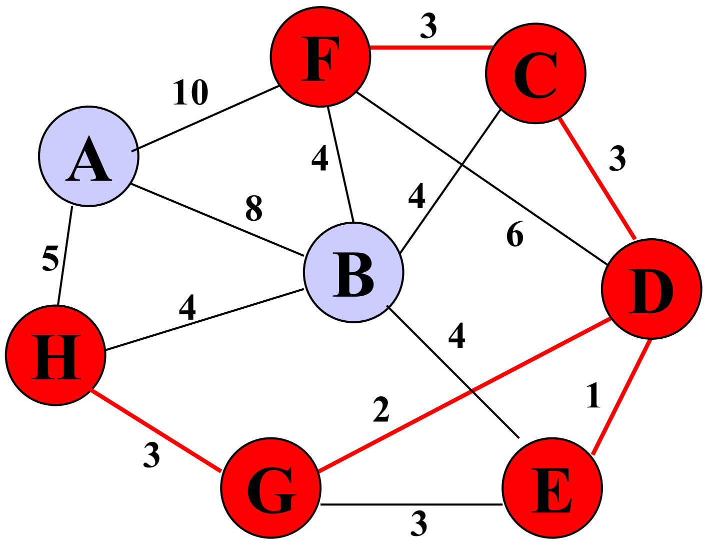
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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

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

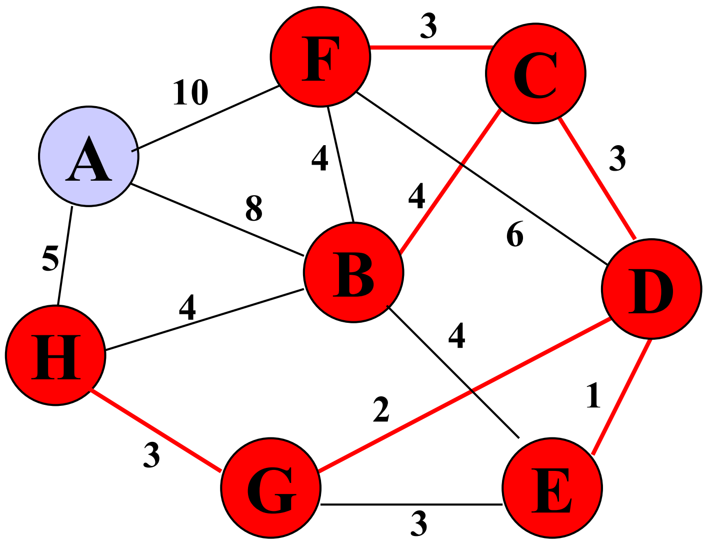
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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

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

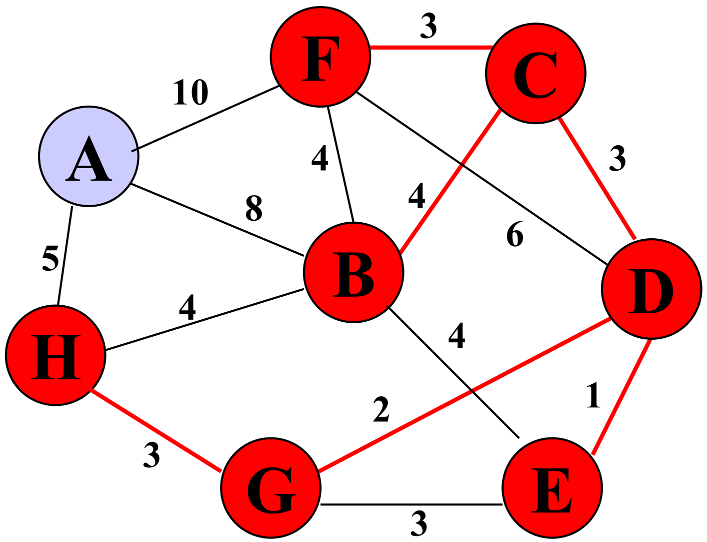
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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

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

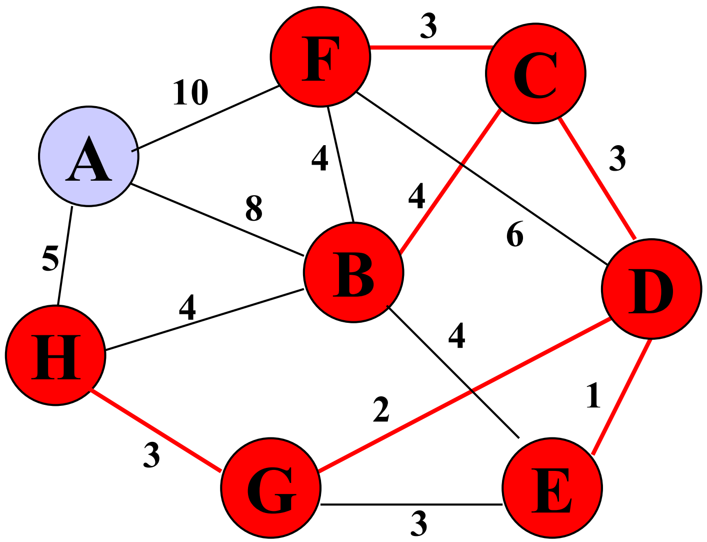
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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

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

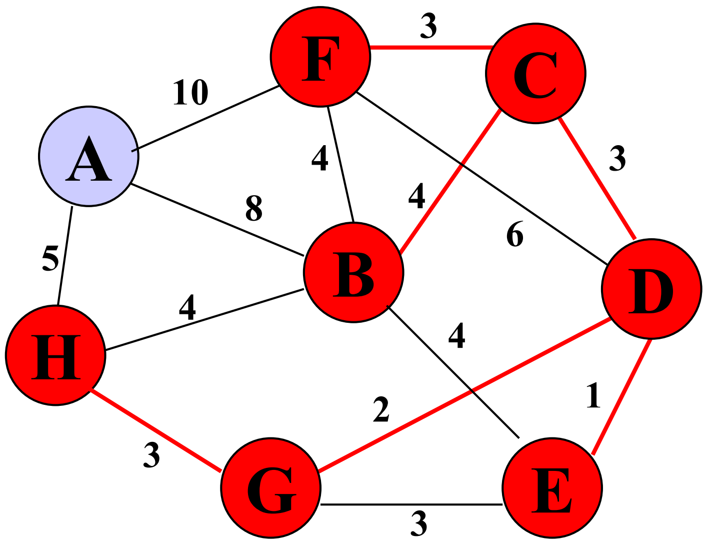
edge	d_v	
(B,E)	4	χ
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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

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

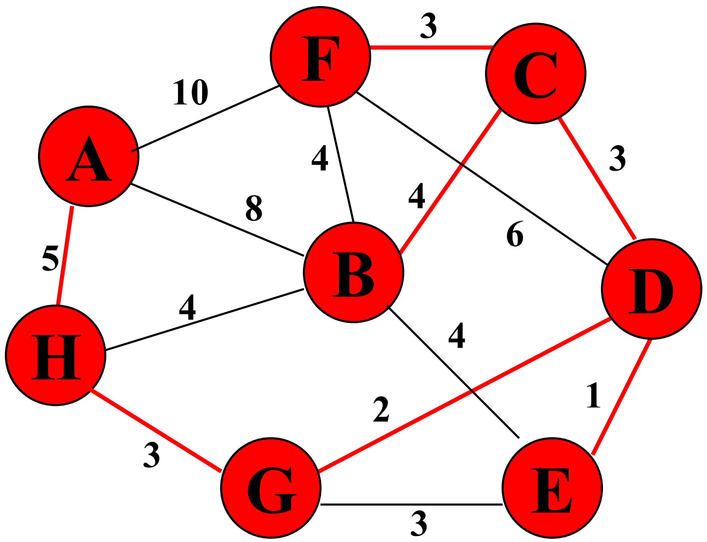
edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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

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

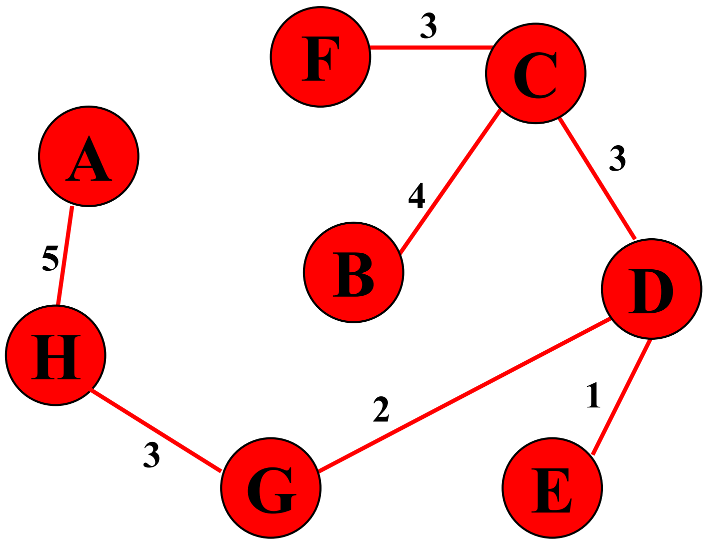
edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



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

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

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

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

not considered

Done

Total Cost $= \Sigma\, d_v = 21$

•Minimum Connector Algorithms

• Kruskal's algorithm

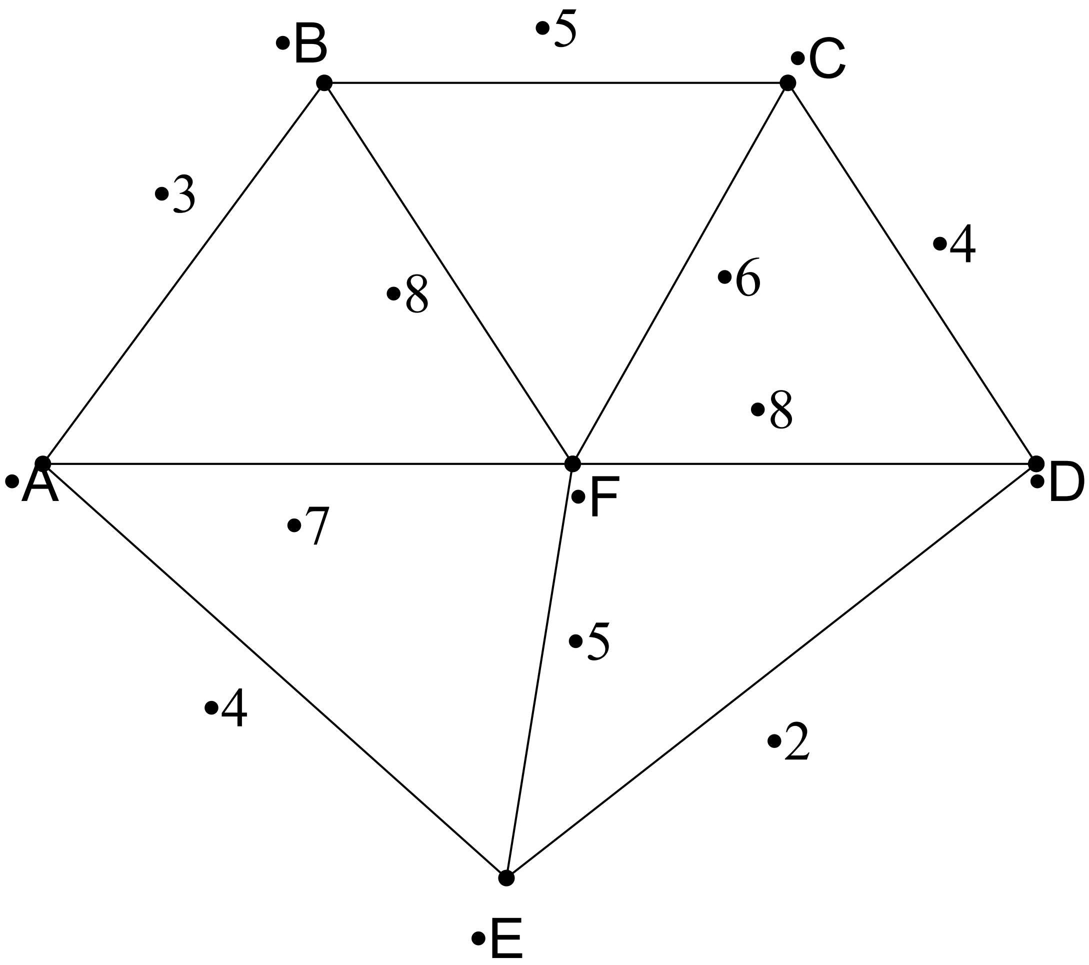
1. Select the shortest edge in a network
2. Select the next shortest edge which does not create a cycle
3. Repeat step 2 until all vertices have been connected

• Prim's algorithm

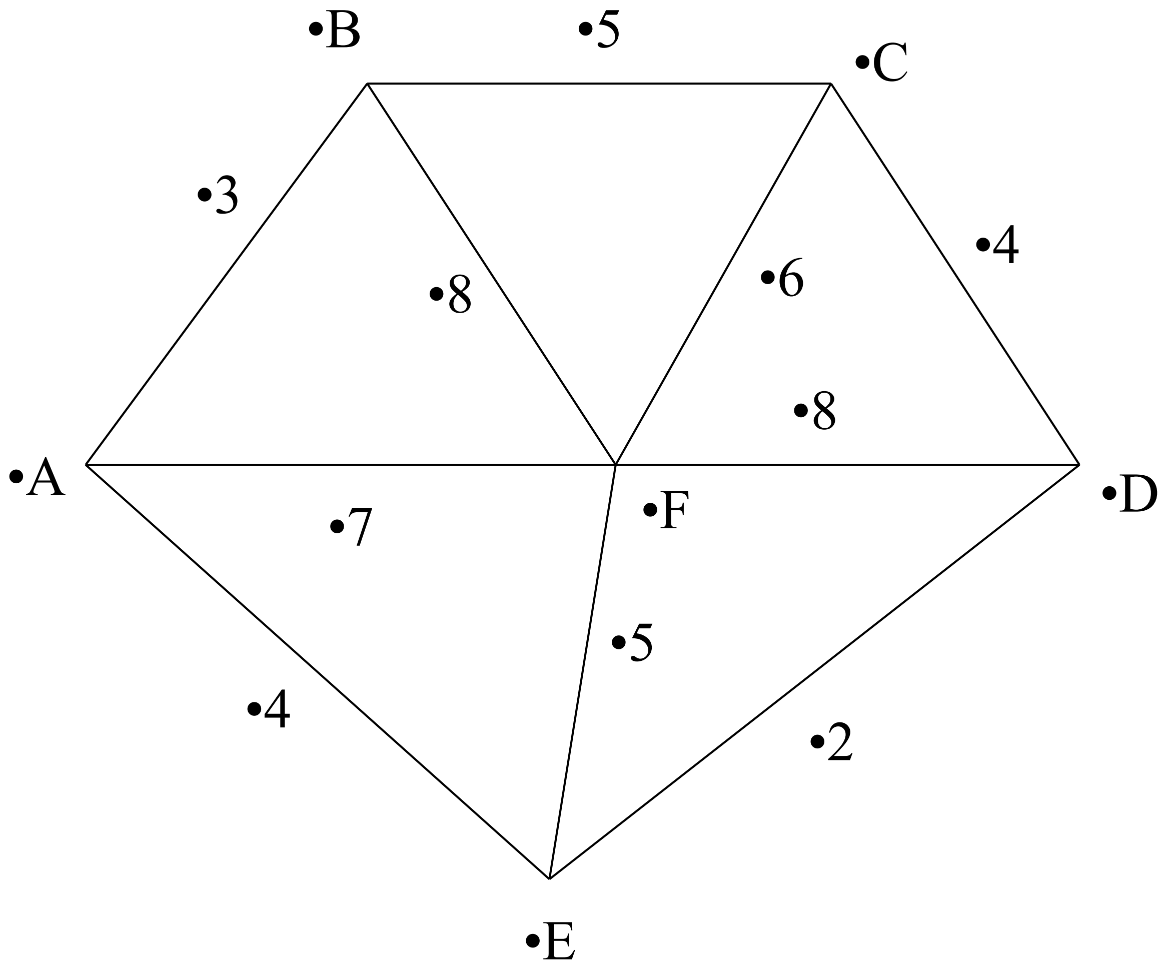
1. Select any vertex
2. Select the shortest edge connected to that vertex
3. Select the shortest edge connected to any vertex already connected
4. Repeat step 3 until all vertices have been connected

•Minimum Connector Algorithms

Prims Algorithm	Kruskal Algorithm
It start to build the MST from any of the Node.	It start to build the MST from Minimum weighted vertex in the graph.
Adjencary Matrix , Binary Heap or Fibonacci Heap is used in Prims algorithm	Disjoint Set is used in Kruskal Algorithm.
Prims Algorithm run faster in dense graphs	Kruskal Algorithm run faster in sparse graphs
Time Complexity is O(E log V) with binay heap and O(E+V log V) with fibonacci heap.	Time Complexity is **O(E log V)**
The next Node included must be connected with the node we traverse	The next edge include may or may not be connected but should not form the cycle.
It traverses the node several times in order to get the minimum distance	It travese the edge only once and based on cycle it will either reject it or accept it,
Greedy Algorithm	Greedy Algorithm


B
5
C
3
8
6
4
8
A
F
D
7
5
4
2
E

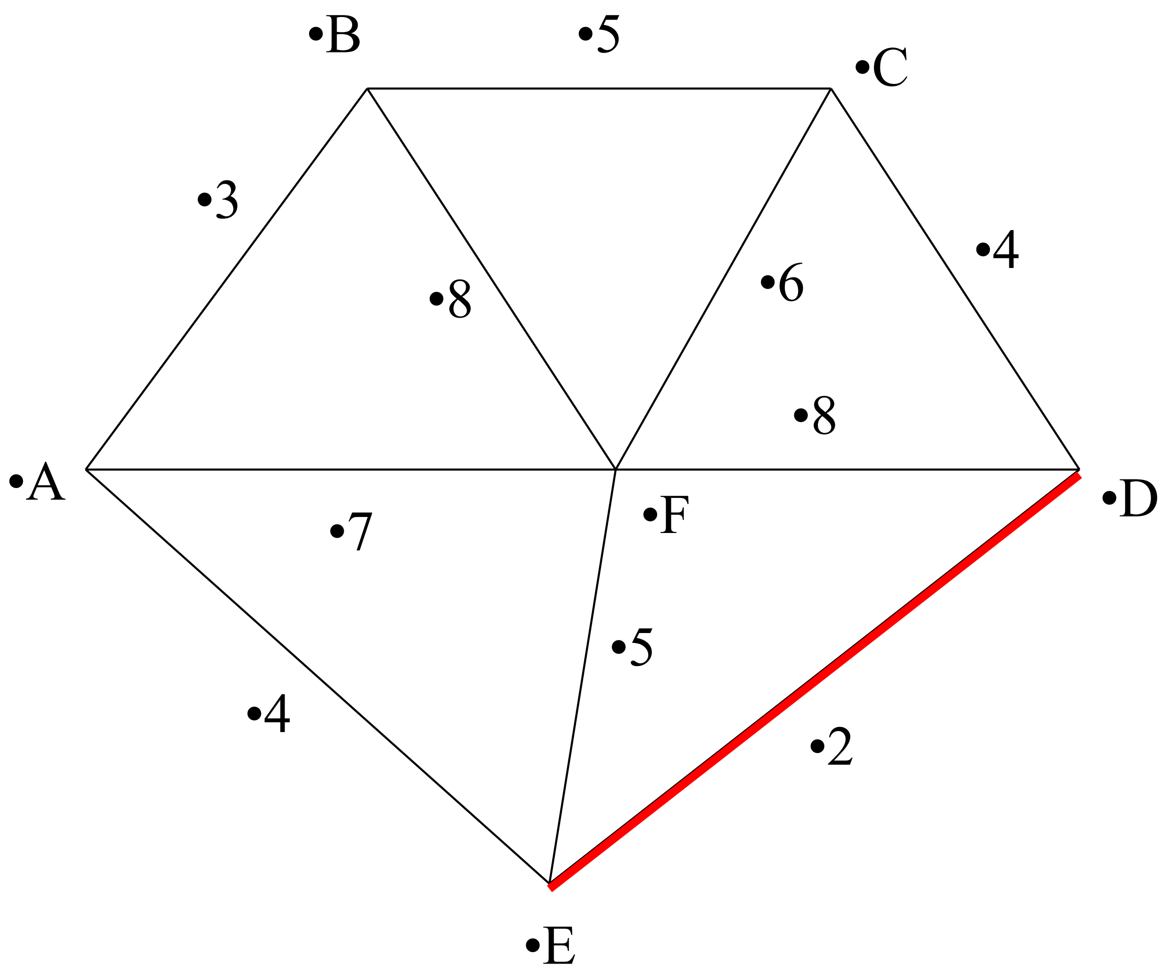
•Kruskal's Algorithm



•List the edges in order of size:

- •ED 2
- •AB 3
- •AE 4
- •CD 4
- •BC 5
- •EF 5
- •CF 6
- •AF 7
- •BF 8
- •CF 8

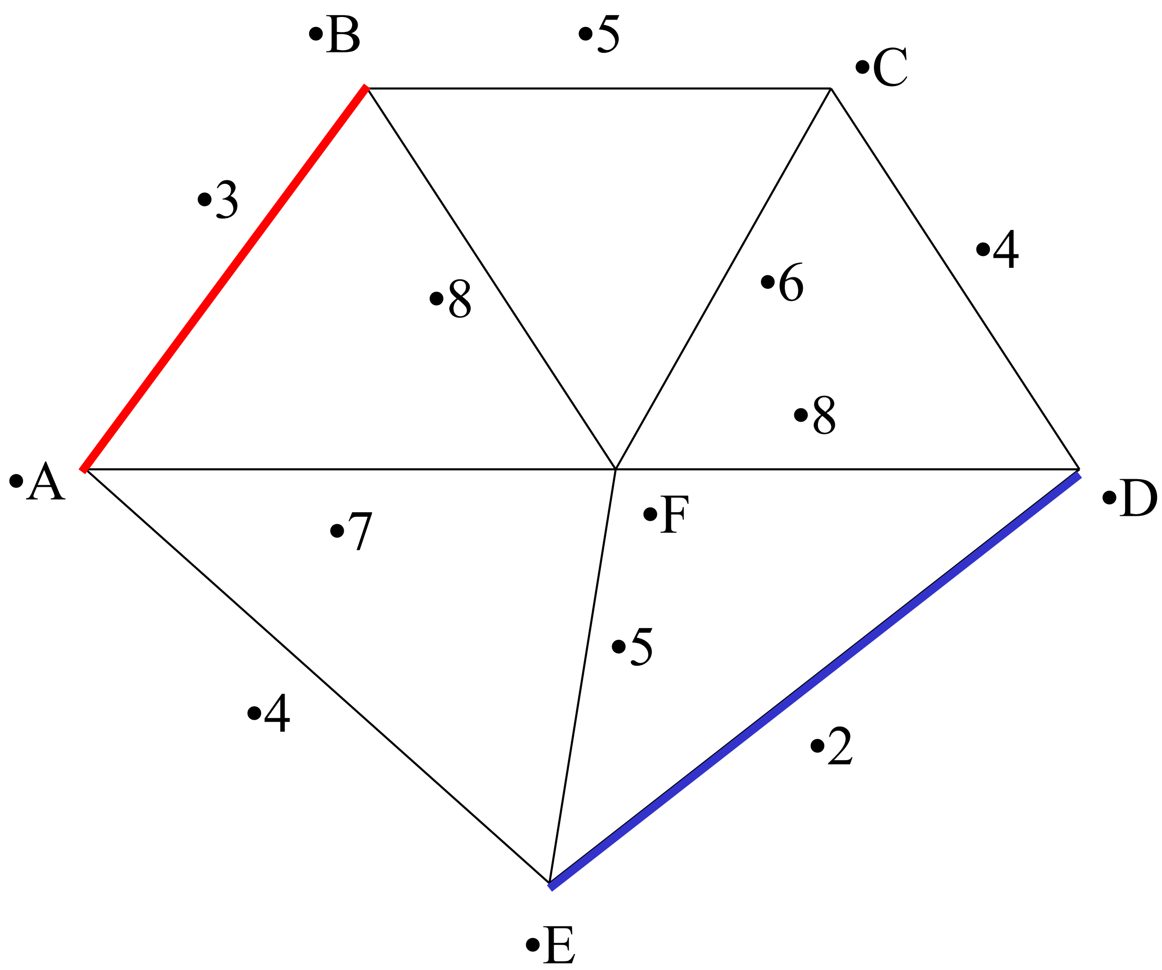
•Kruskal's Algorithm



Select the shortest edge in the network

- **ED 2**

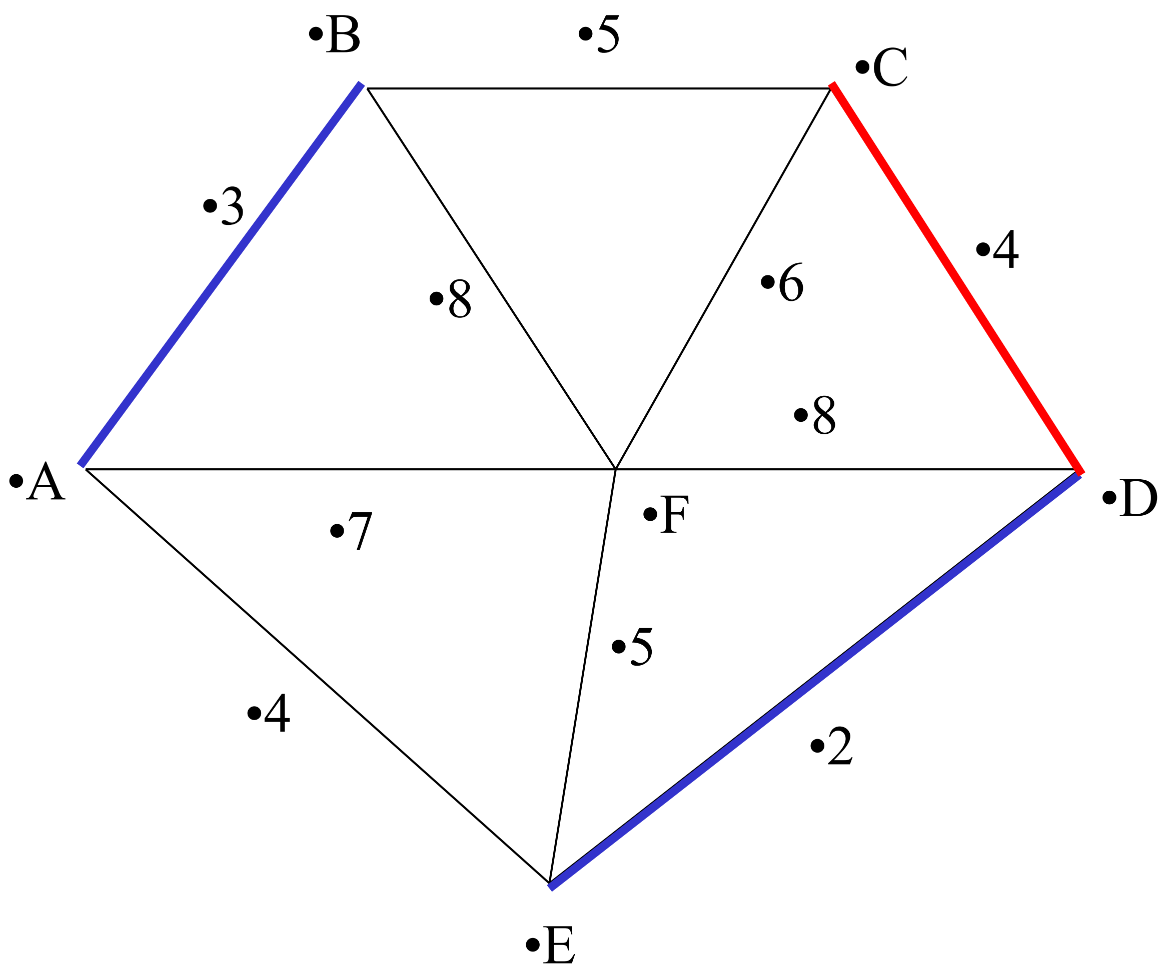
•Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

- **ED 2**
- **AB 3**

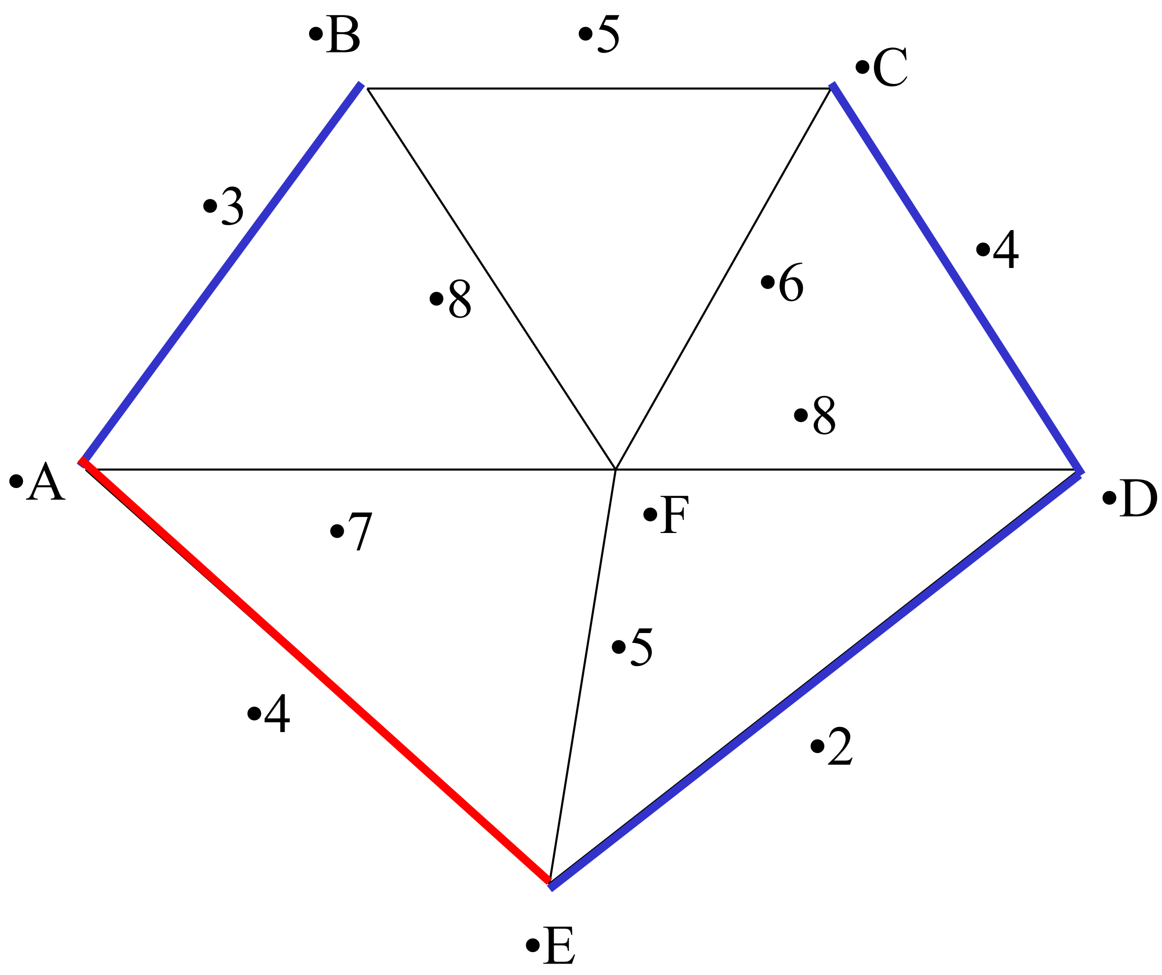
•Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

- **ED 2**
- **AB 3**
- **CD 4 (or AE 4)**

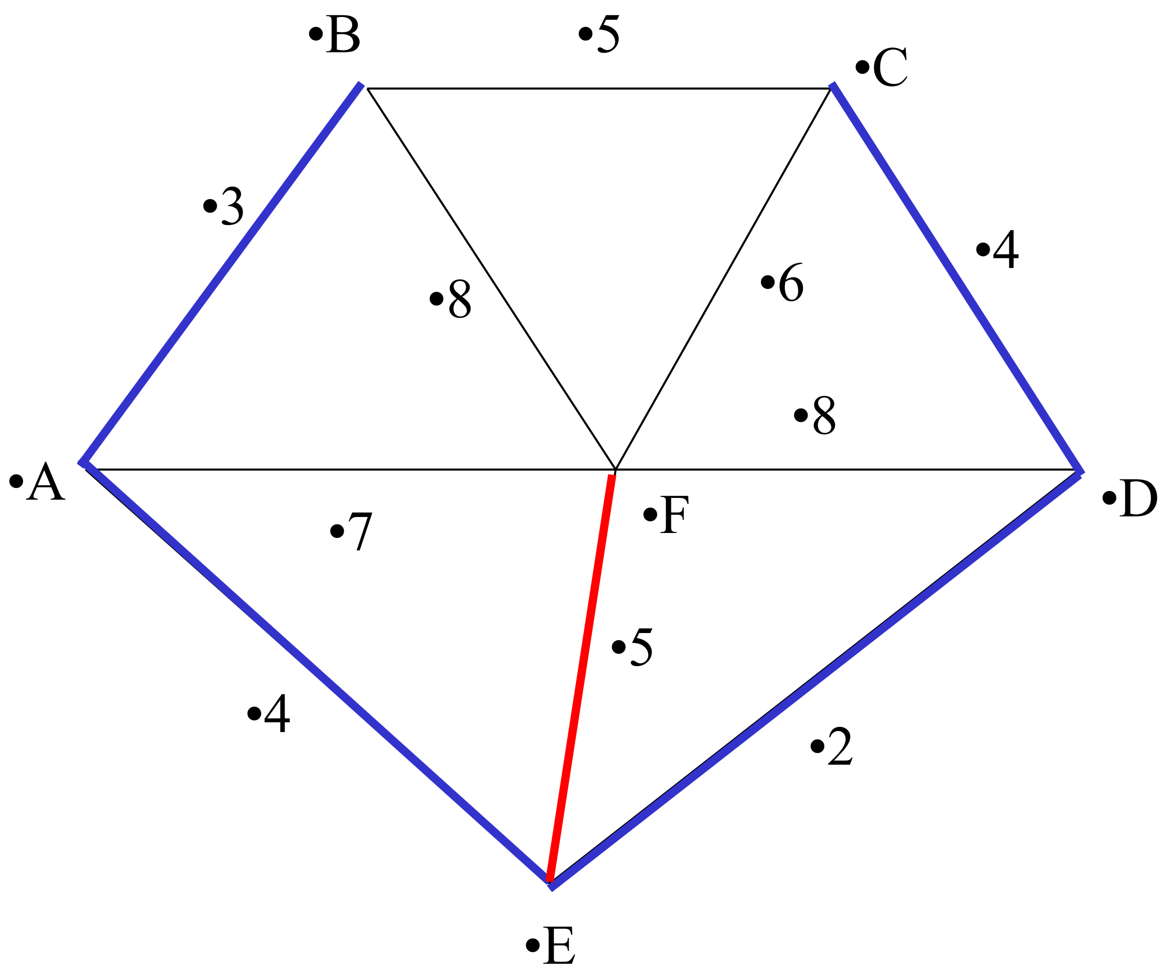
•Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

- **ED 2**
- **AB 3**
- **CD 4**
- **AE 4**

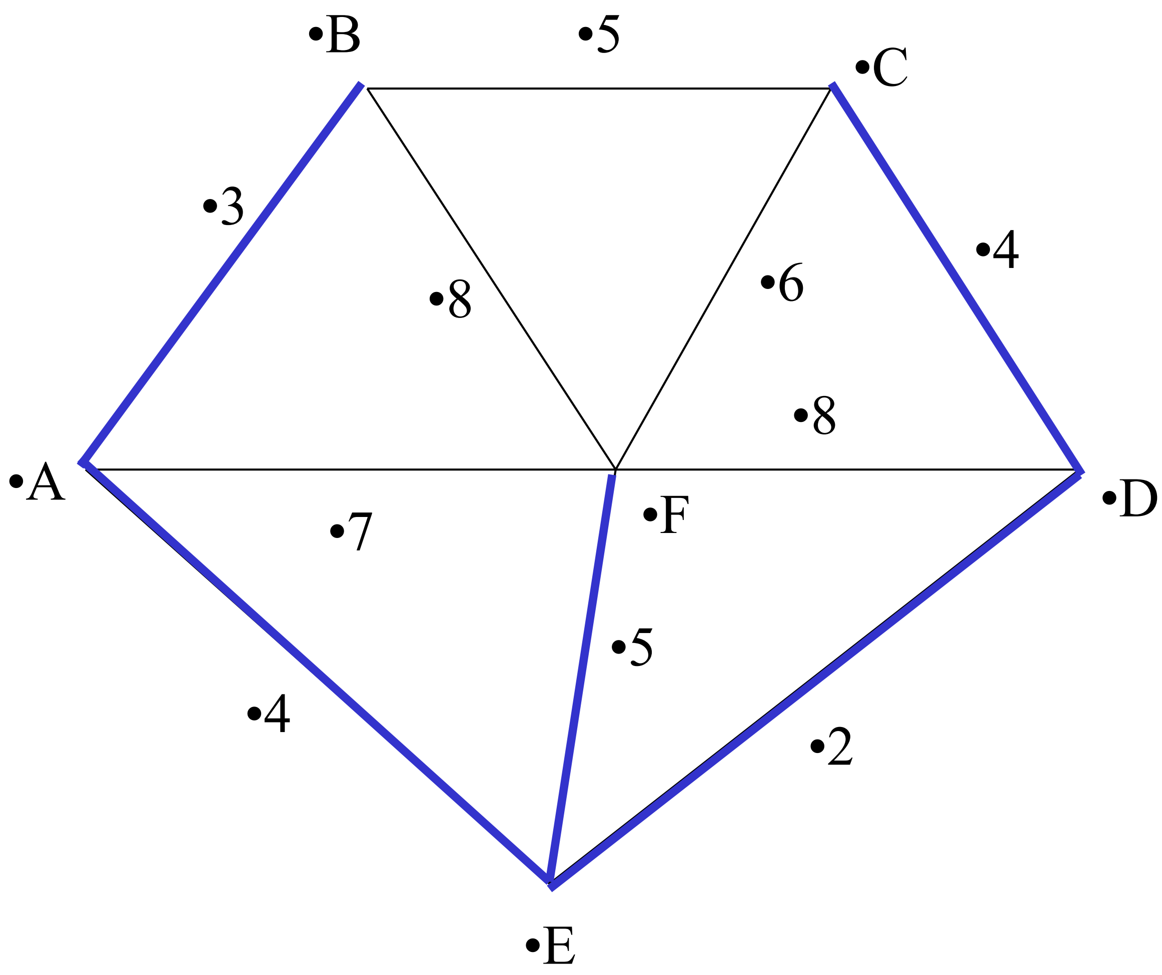
•Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

- **ED 2**
- **AB 3**
- **CD 4**
- **AE 4**
- **BC 5 – forms a cycle**
- **EF 5**

•Kruskal's Algorithm



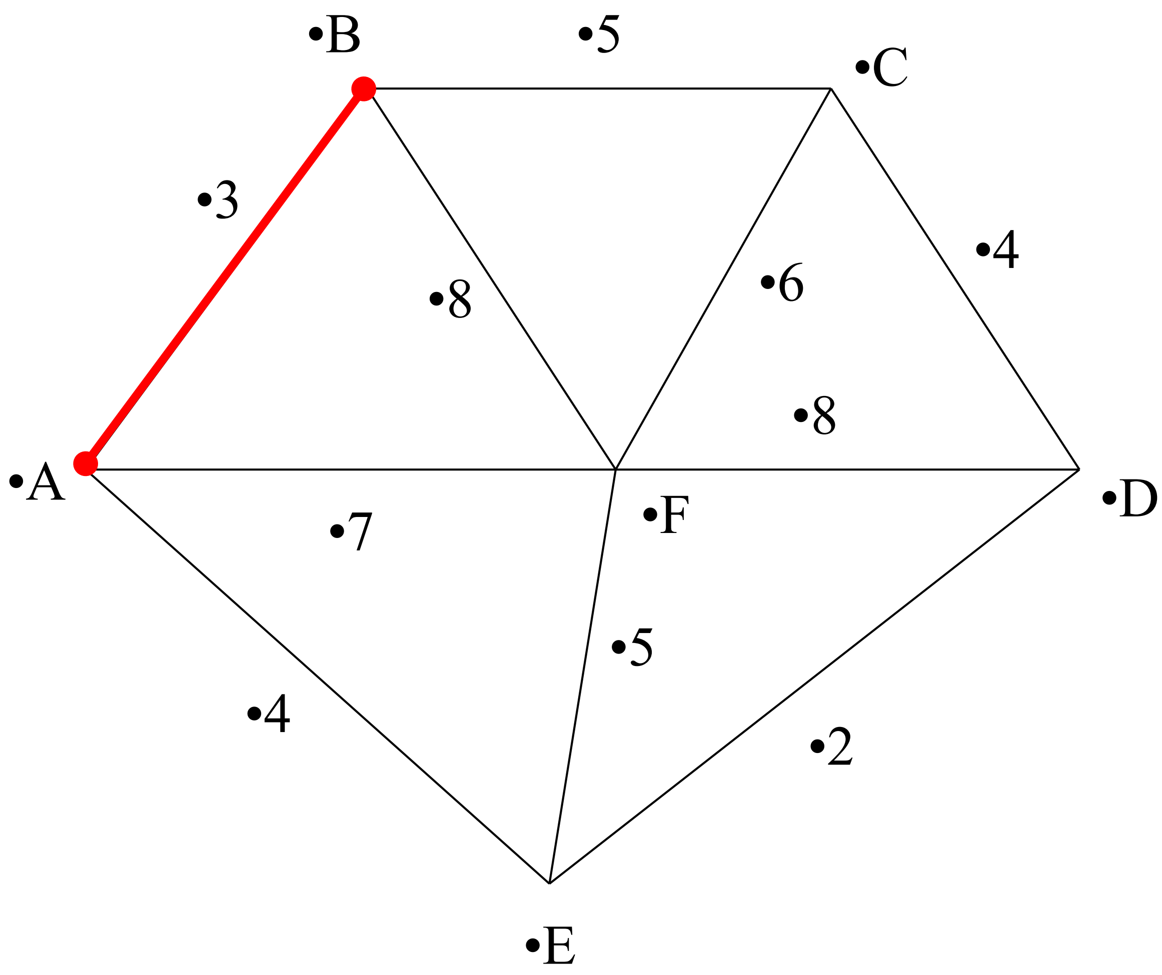
All vertices have been connected.

The solution is

- **ED 2**
- **AB 3**
- **CD 4**
- **AE 4**
- **EF 5**

- Total weight of tree: 18

•Prim's Algorithm



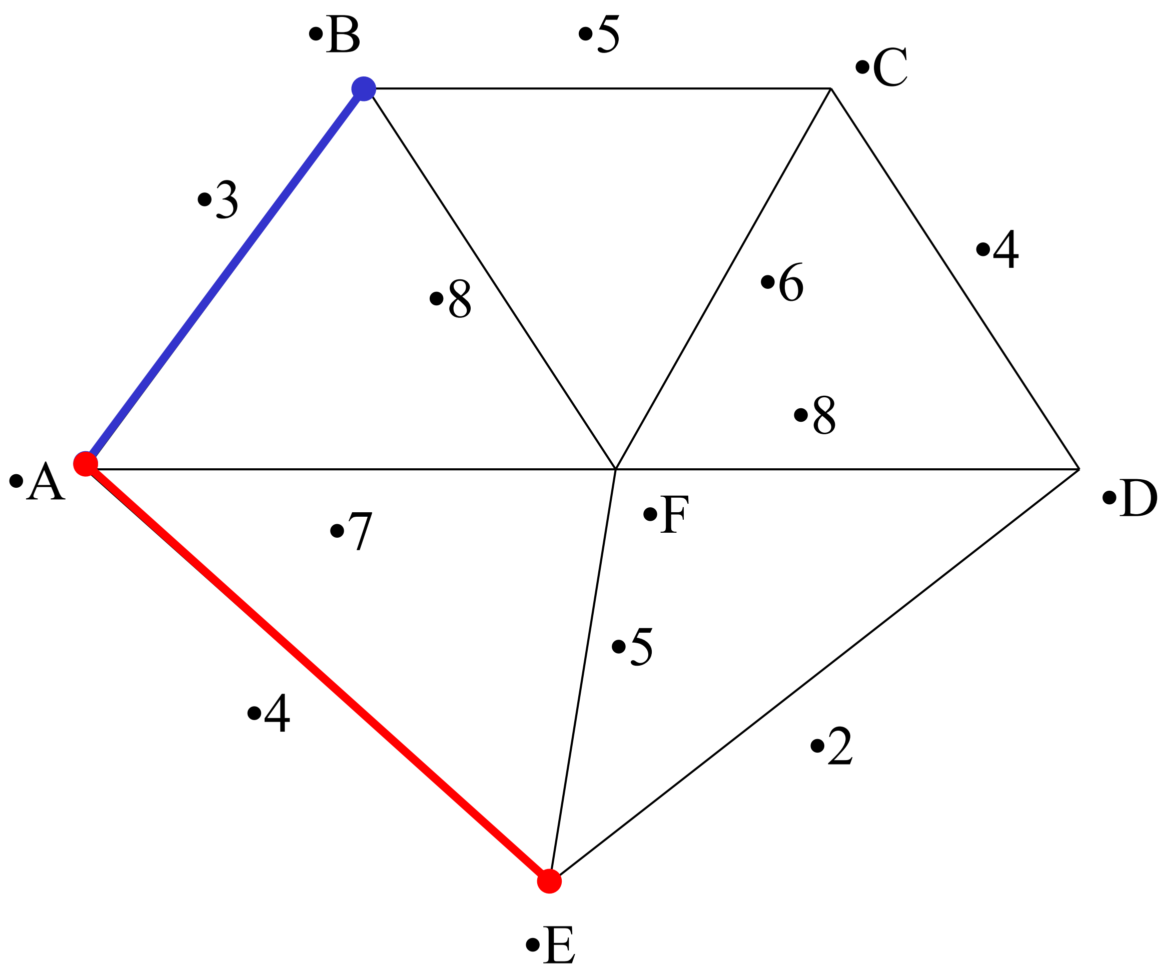
•Select any vertex

•A

•Select the shortest edge connected to that vertex

•AB 3

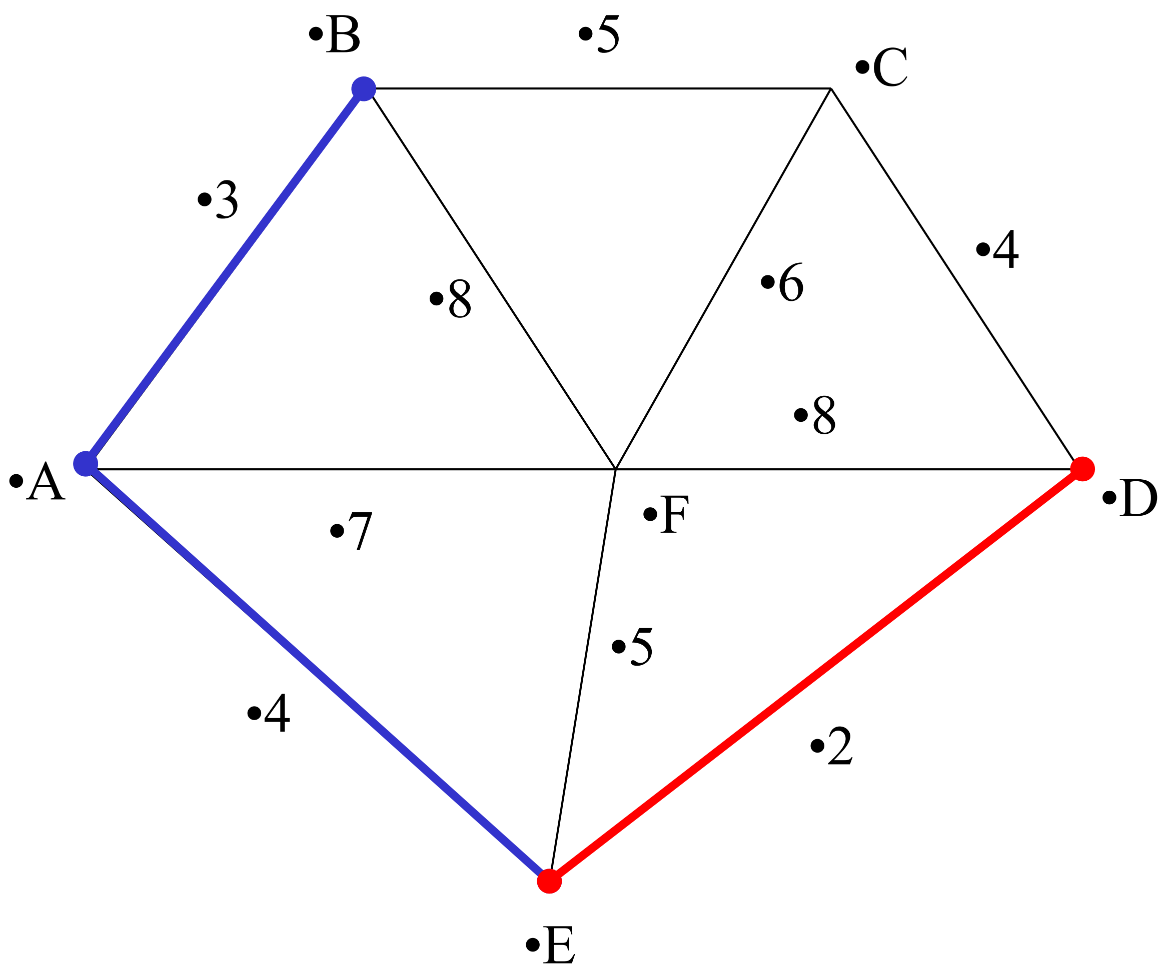
•Prim’s Algorithm



Select the shortest edge connected to any vertex already connected.

AE 4

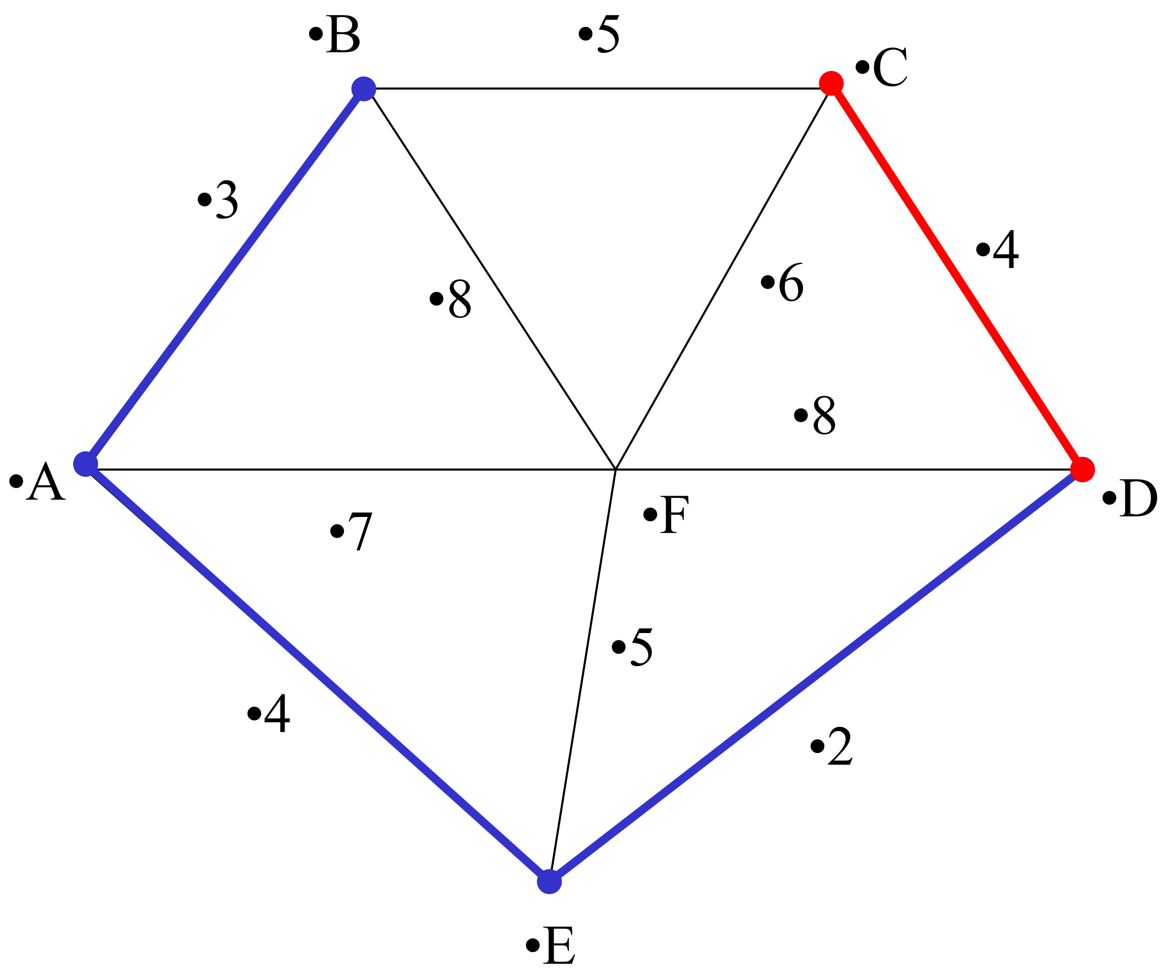
•Prim’s Algorithm



Select the shortest edge connected to any vertex already connected.

ED 2

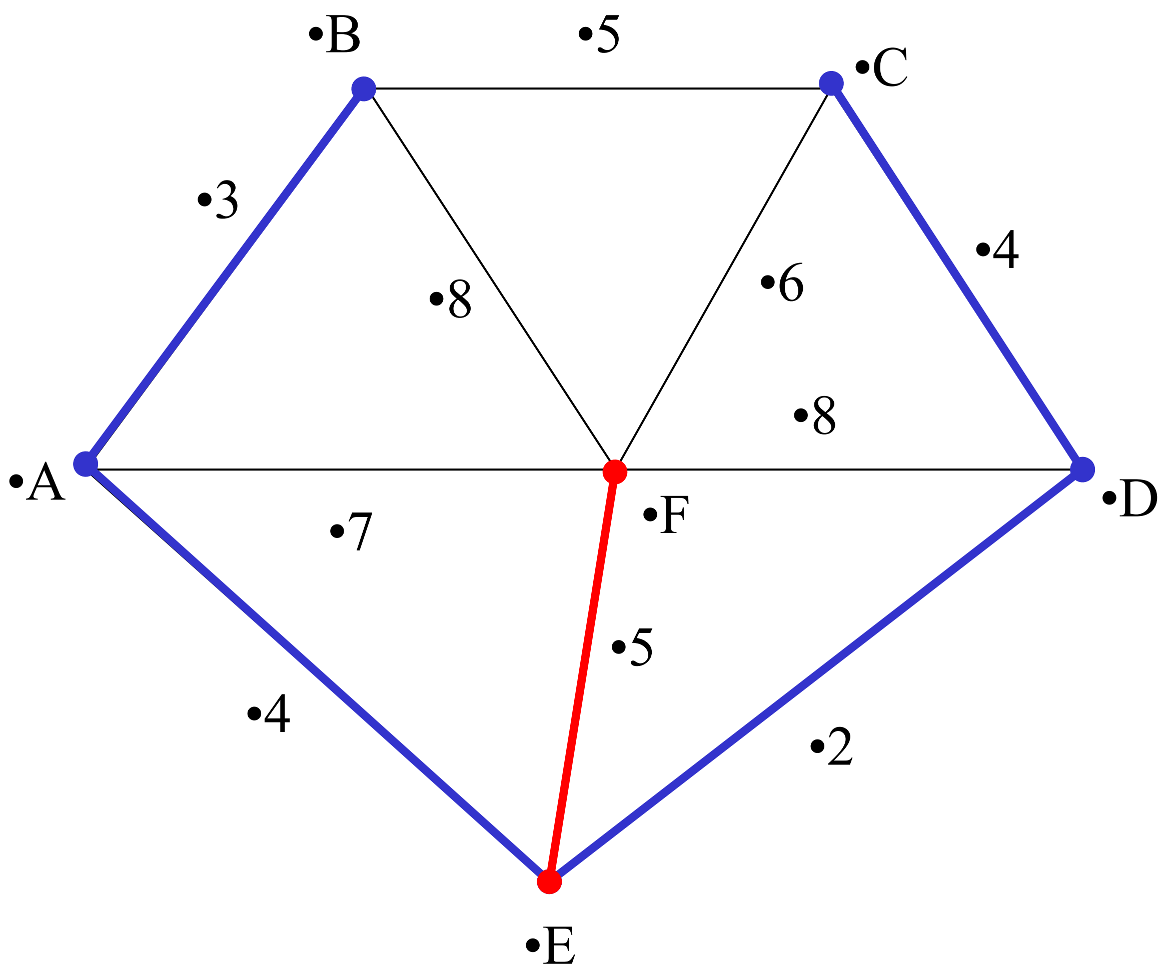
•Prim's Algorithm



Select the shortest edge connected to any vertex already connected.

DC 4

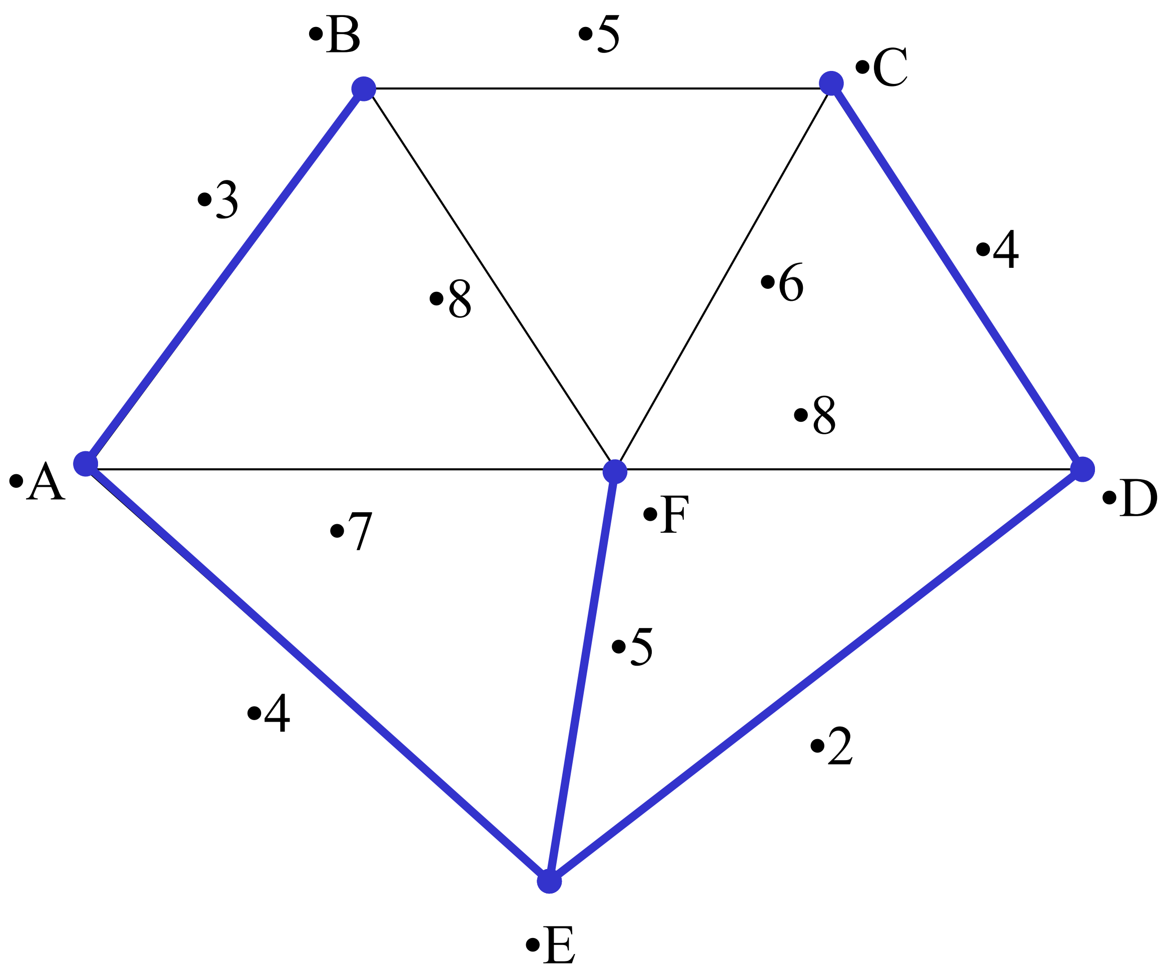
•Prim's Algorithm



Select the shortest edge connected to any vertex already connected.

EF 5

•Prim's Algorithm



All vertices have been connected.

The solution is

- **AB 3**
- **AE 4**
- **ED 2**
- **DC 4**
- **EF 5**

- Total weight of tree: 18

Some points to note

- Both algorithms will always give solutions with the same length.

- They will usually select edges in a different order – you must show this in your workings.

- Occasionally they will use different edges – this may happen when you have to choose between edges with the same length. In this case there is more than one minimum connector for the network.