CS202 – Data Structures

LECTURE-21

Graphs – IV

Shortest Path and MST

Dr. Maryam Abdul Ghafoor
Assistant Professor
Department of Computer Science, SBASSE

Agenda

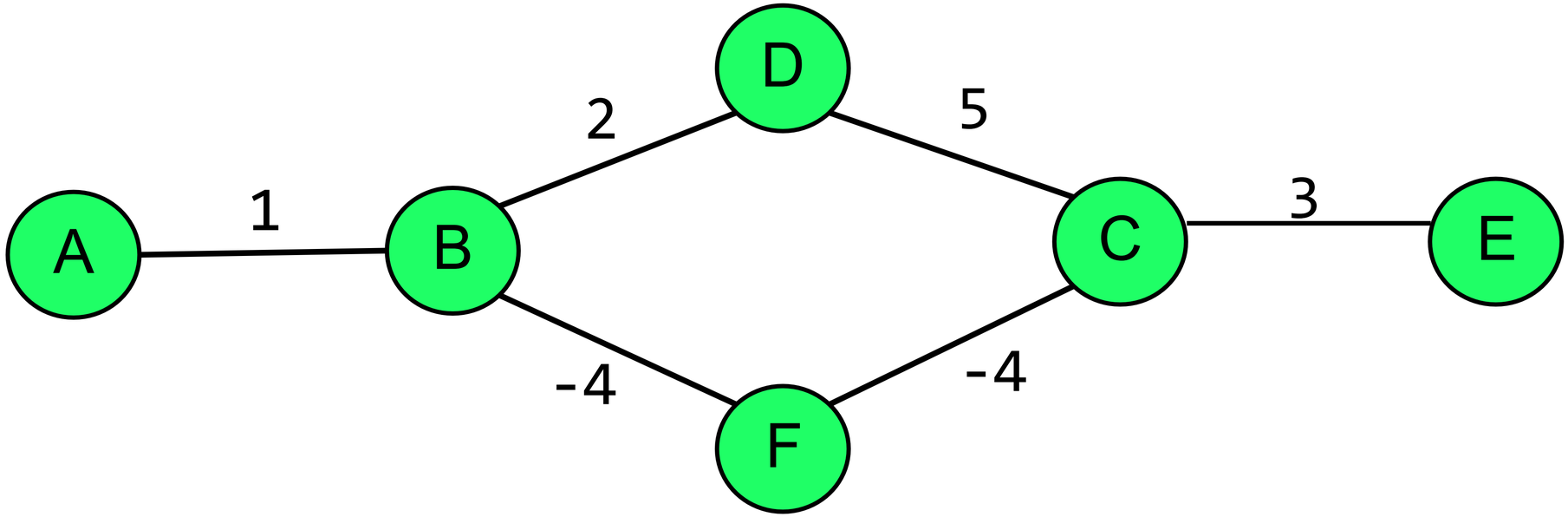
- Shortest path algorithm for negative weights
- Minimum Spanning Trees

Dijkstra's algorithm - Summary

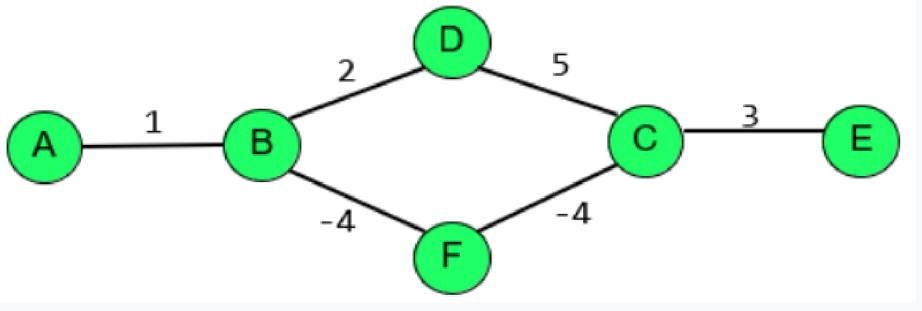
- Finds single-source shortest paths in a weighted graph (directed or undirected) with no negative-weight edges

- An example of a greedy algorithm:
  - At each step, always does what seems best at that step
    - A locally optimal step, not necessarily globally optimal
  - Once a vertex is known, it is not revisited
    - Turns out Dijkstra's algorithm is globally optimal

What is the shortest Path from A to E?



What is the weight of the shortest path from A to E in the given graph?

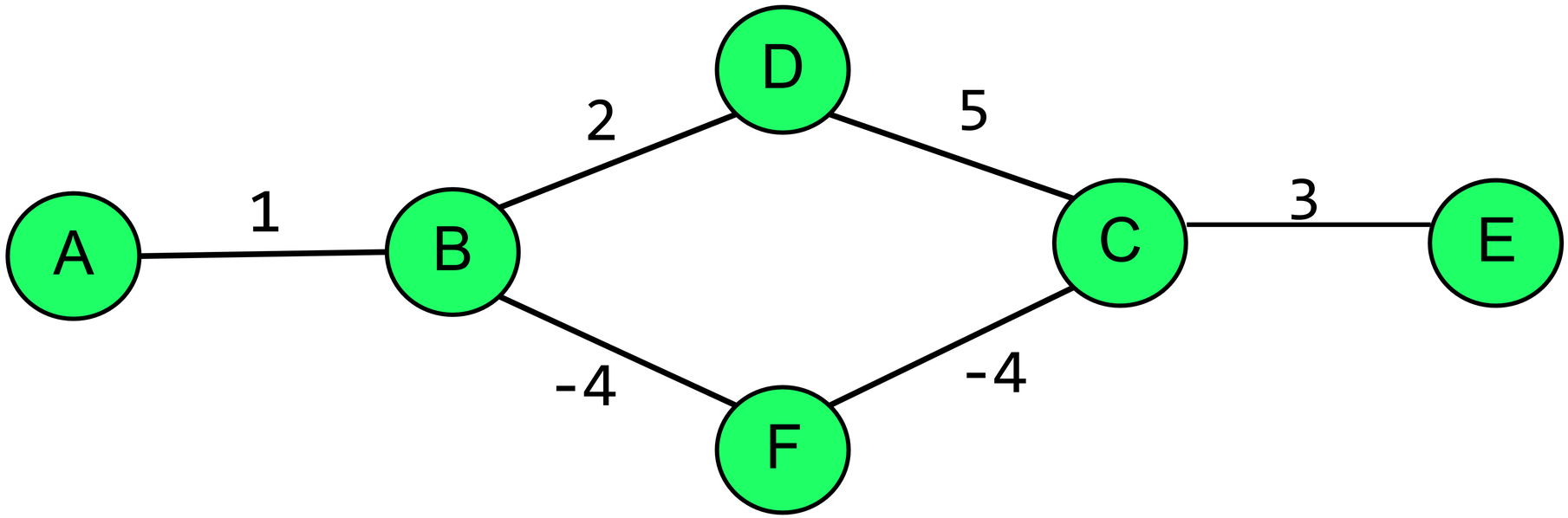


Nobody has responded yet.

Hang tight! Responses are coming in.

What is the shortest Path from A to E?

- The shortest path from A-E is not well-defined on this graph due to negative-cost cycle!



Total Cycle Weight = 2 + 5 - 4 - 4

How to deal with negative edge weights?

- Dijkstra fails with negative edge weights

- Many real-world problems, for example, currency exchange or network routing involve such edges

- Solution:
  - Bellman-Ford's Algorithm
    - Relax all edges V-1 times

Why V-1 Edge Relaxations?

- Edge Relaxation
  - If the path to v via u is shorter than the best path known so far, update it.

```
if dist[u] + weight(u, v) < dist[v]:
    dist[v] = dist[u] + weight(u, v)
```

  - Relaxing edge once will compute the shortest path for vertices that are **1 edge** away
  - Why not V iterations?

Bellman Ford's Algorithm – Key Idea

- Initialize distance to all nodes as ∞, except the source (0)

- Relax all edges V−1 times:

  If dist[u] + weight(u,v) < dist[v]

  update dist[v]

How can we detect a Negative Cycle?

Do one more pass: if any edge still relaxes, a negative cycle exists

Pseudocode & Efficiency

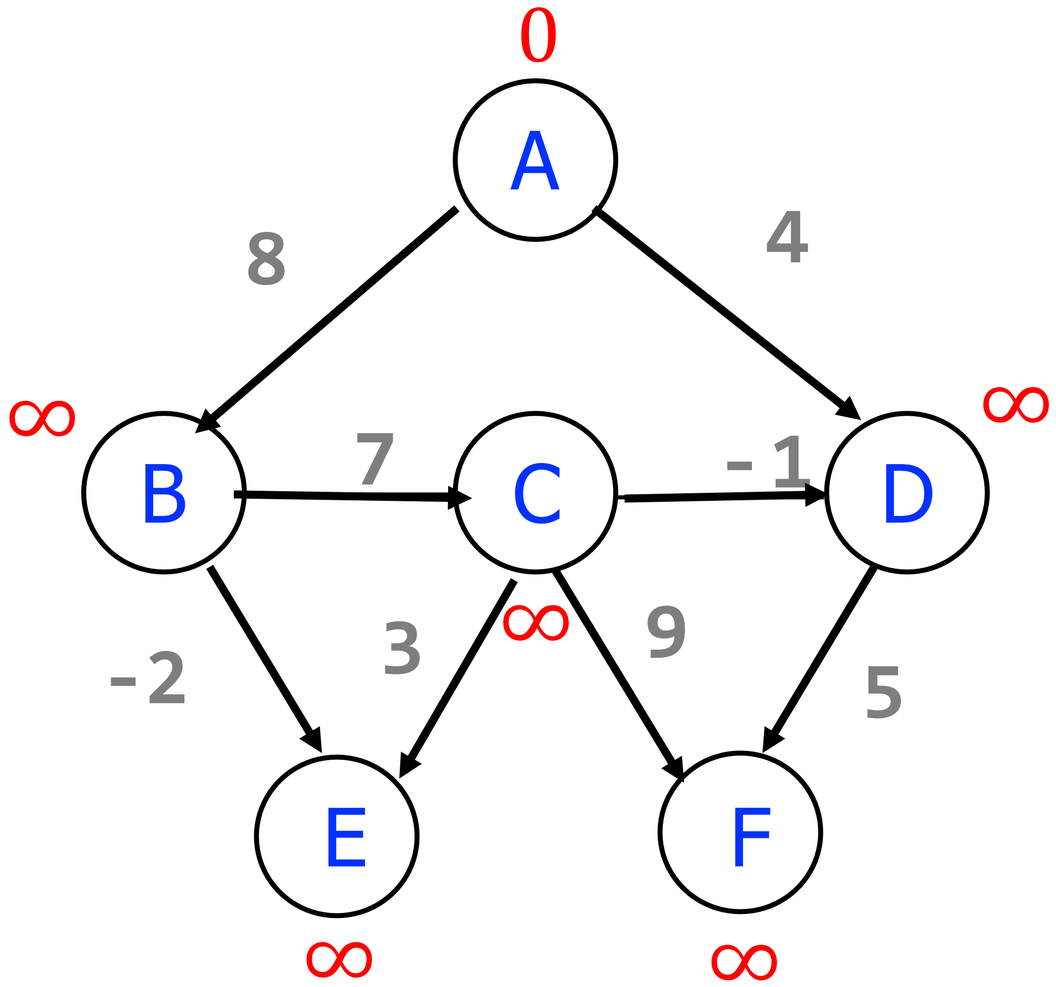
```
bellmanFord(Graph G, Node v) {
   for each vertex:
       D[u]= ∞
   D[v]=0 //source vertex

  Repeat v-1 times:

  for each edge (b,a) in G {
       if(D[b] + weight((b,a)) < D[a]){
            D[a] = D[b] + weight(b,a)
               a.path = b //for computing actual paths
       }
     }
}
```

O(|V|)

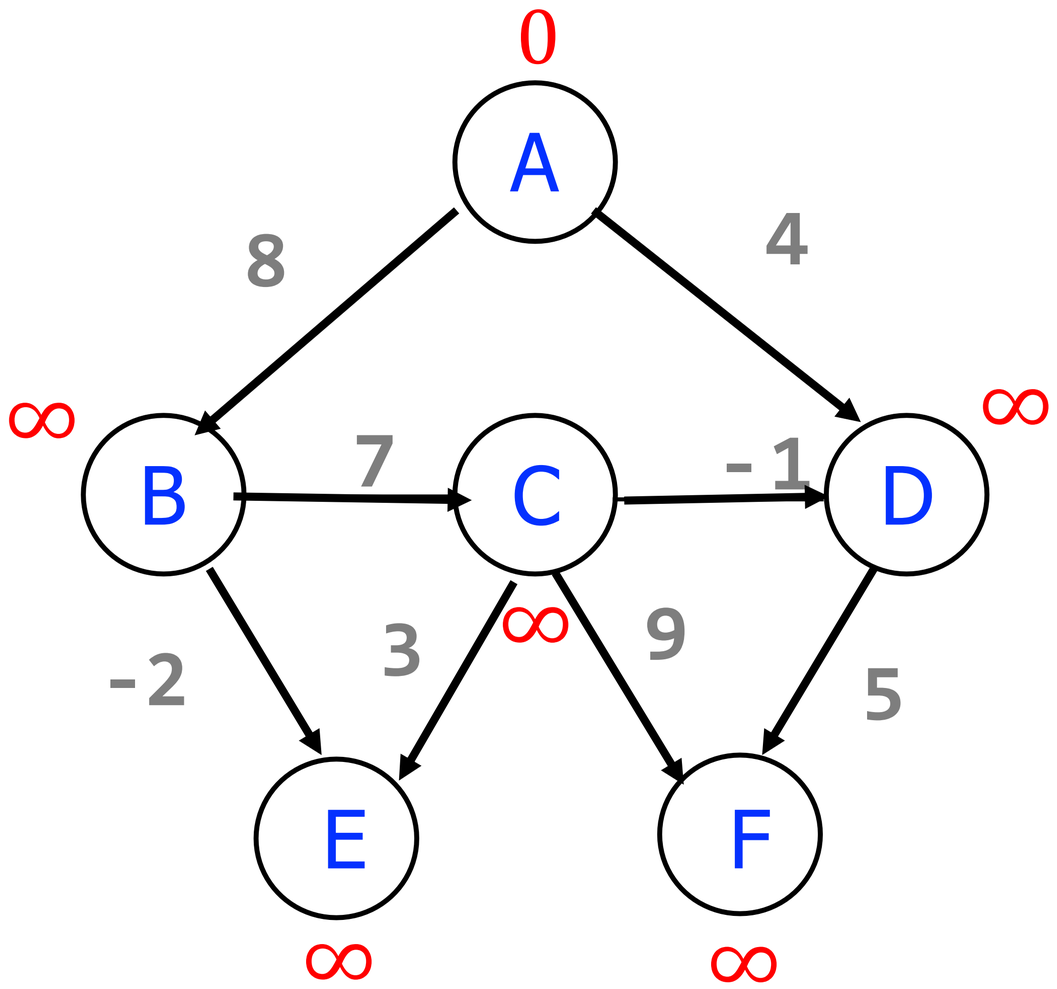
O(|V||E|)

O(|V||E|)

Bellman-Ford's Algorithm Example



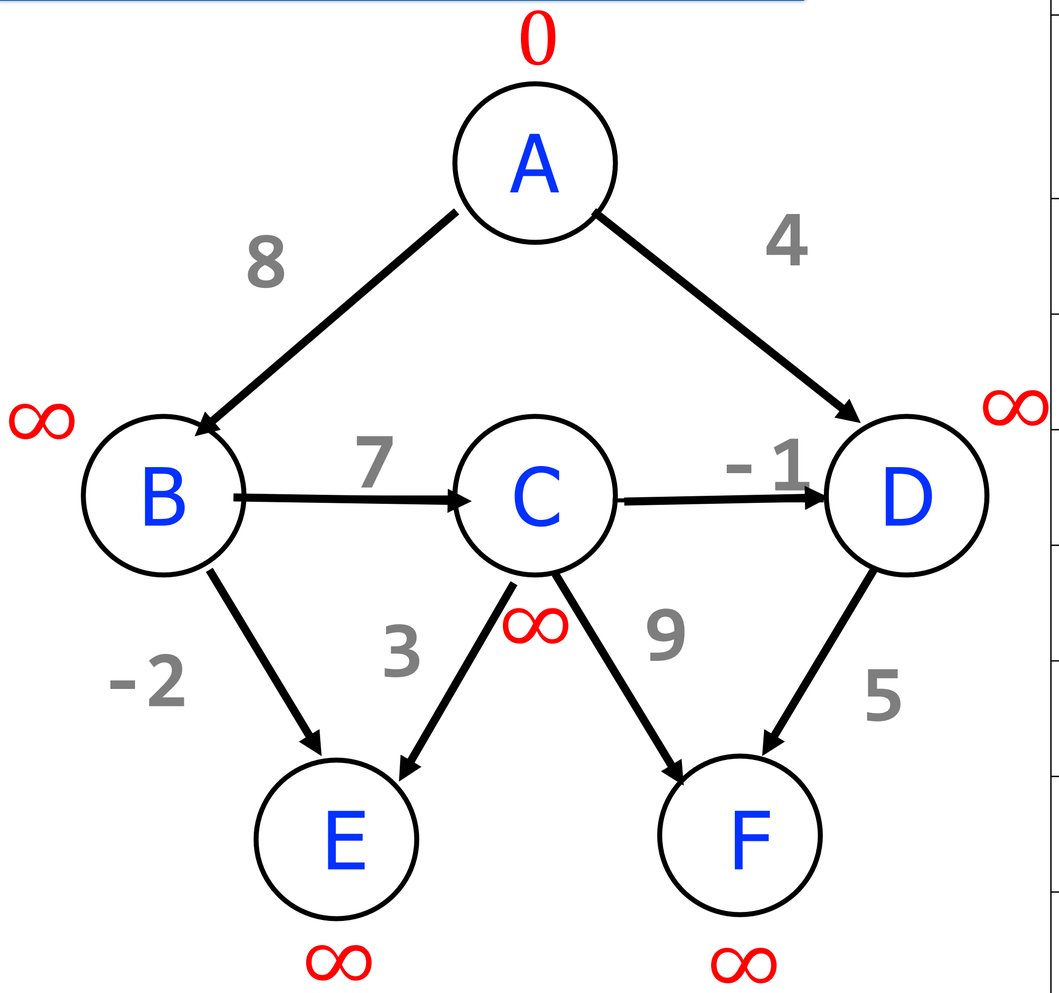
Bellman-Ford's Algorithm Example



Edge	Dist[]	parent
A→B	0	-
A→ D	∞	-
B → C	∞	-
C →D	∞	-
D→F	∞	-
C→ F	∞	-
C→E	∞	-
B→E	∞	-

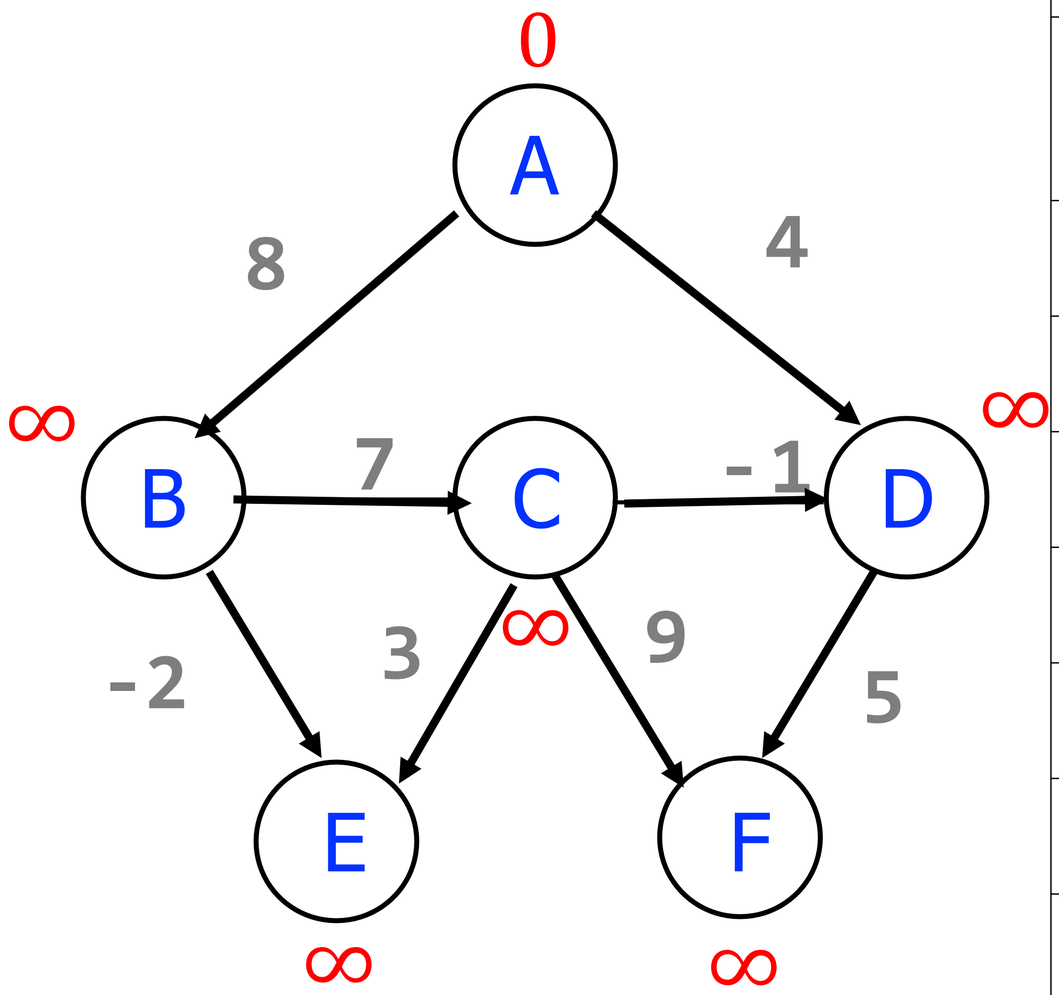
Bellman-Ford's Algorithm – 1st Iteration

Compute distances.



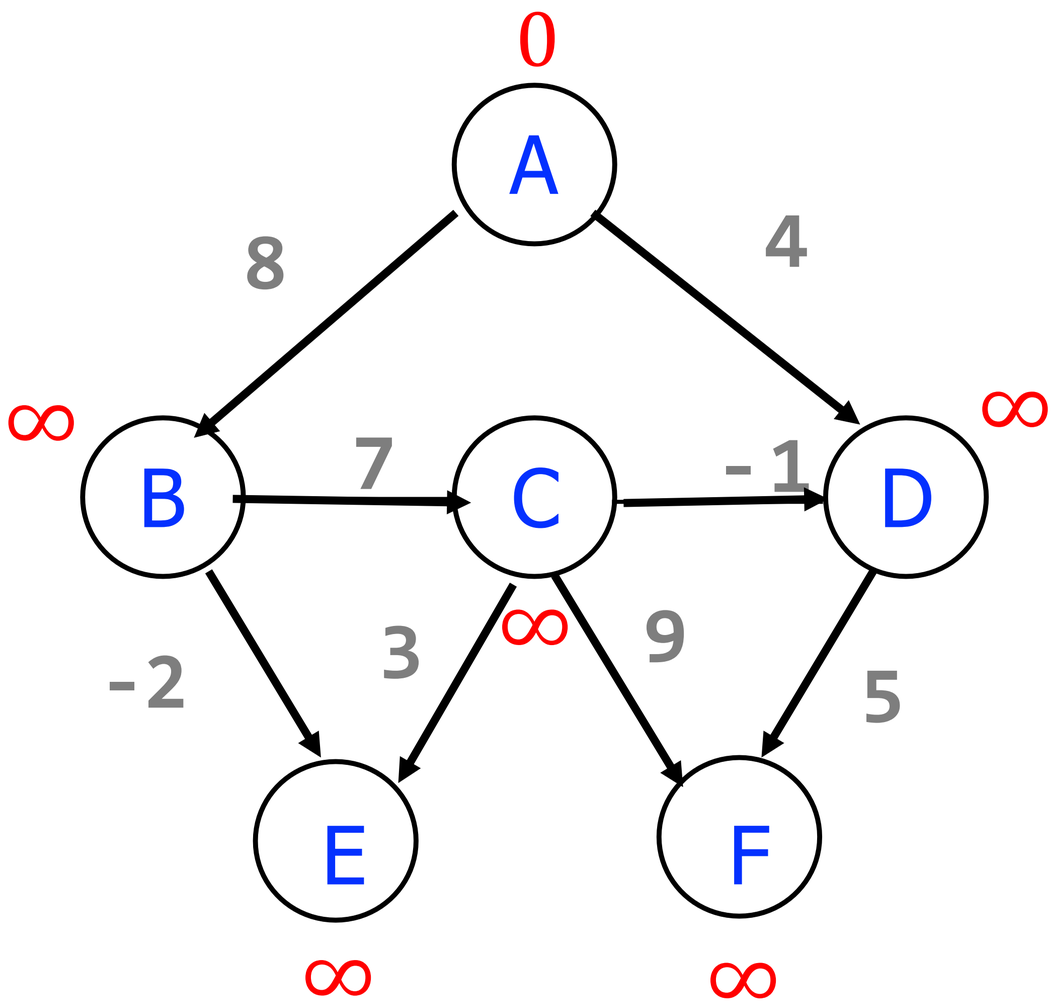
Edge	Dist[]	parent
A→B	dist[b] = min(dist[b],8+dist[a]) = 8	A
A→ D	dist[d]=min(dist[d],4+dist[a]=4	A
B → C	dist[c]= 7+dist[b] =15	B
C →D		
D→F		
C→ F		
C→E		
B→E		

Bellman-Ford's Algorithm – 1st Iteration



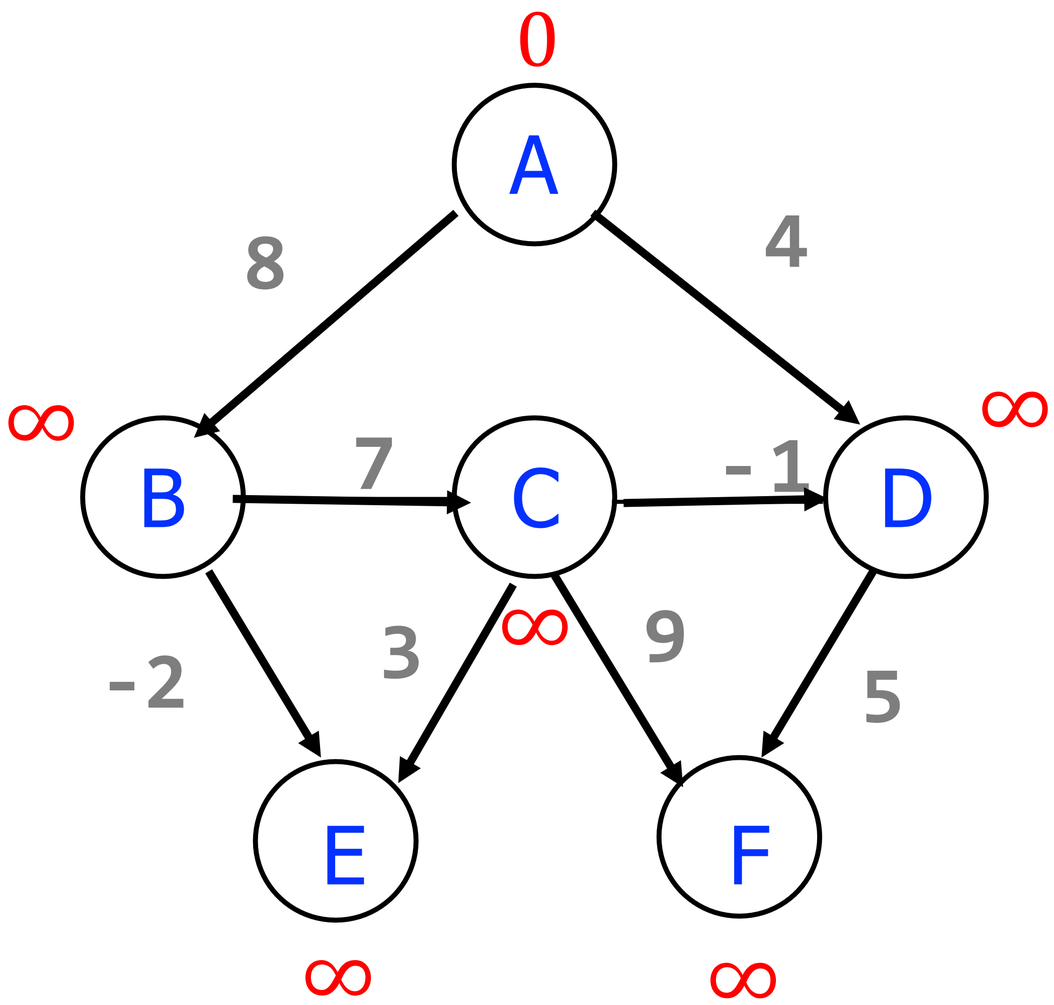
Edge	Dist[]	parent
A→B	dist[b] = min(dist[b],8+dist[a]) = 8	A
A→ D	dist[d]=min(dist[d],4+dist[a]=4	A
B → C	dist[c]= 7+dist[b] =15	B
C →D	Dist[d]=4	A
D→F	Dist[f]=9	D
C→ F	Dist[f]=9	D
C→E	Dist[e]=18	C
B→E	Dist[e]=6	B

Bellman-Ford's Algorithm- 2nd Iteration



Edge	Dist[]	Dist[]	Parent
A→B	dist[b] = 8		A
A→ D	Dist[d] = 4		A
B → C	Dist[c]=15		B
C →D	Dist[d]=4		A
D→F	Dist[f]=9		D
C→ F	Dist[f]=9		D
C→E	Dist[e]=18		C
B→E	Dist[e]=6		B

Bellman-Ford's Algorithm- 2nd Iteration



Edge	Dist[]	Dist[]	parent
A→B	dist[b] = 8	dist[b] = 8	A
A→ D	Dist[d] = 4	Dist[d] = 4	A
B → C	Dist[c]=15	Dist[c]=15	B
C →D	Dist[d]=4	Dist[d]=4	A
D→F	Dist[f]=9	Dist[f]=9	D
C→ F	Dist[f]=9	Dist[f]=9	D
C→E	Dist[e]=18	Dist[e]=18	C
B→E	Dist[e]=6	Dist[e]=6	B

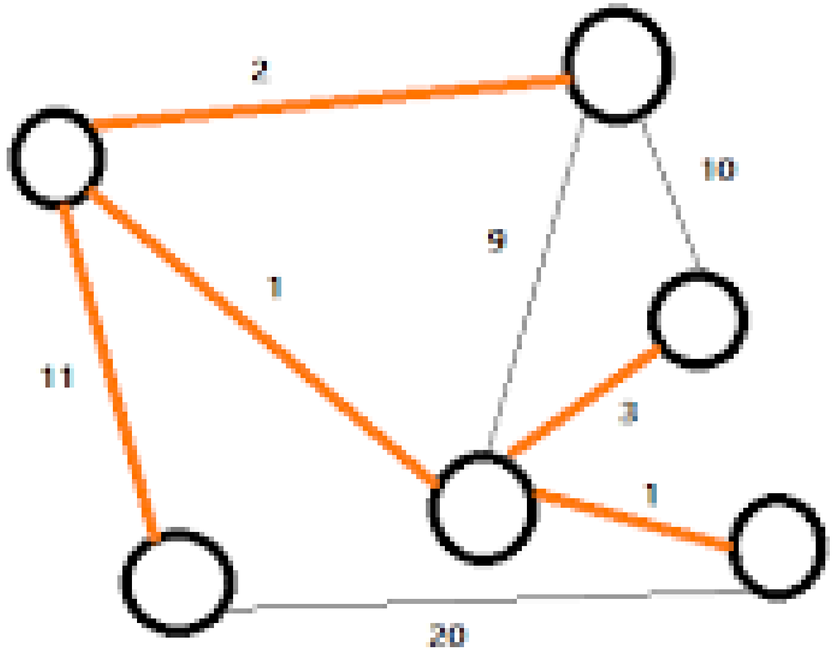
Spanning Trees

Spanning Tree

- Given an **undirected** graph, a **spanning** tree T is a subgraph of G, where T:
  - Is connected.
  - Is acyclic.

  } Tree
  - Includes all vertices.

  } Spanning

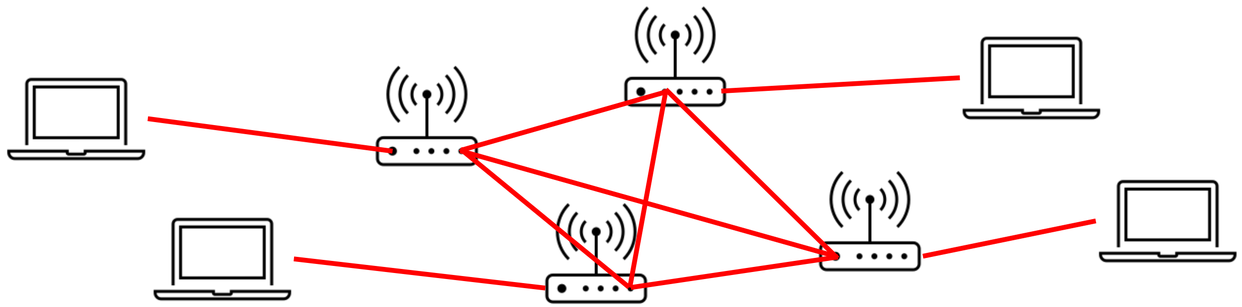


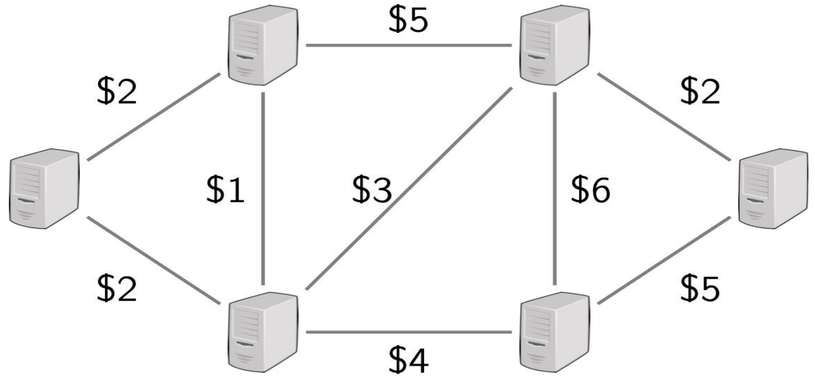
Example:

- Spanning tree is the nodes connected with orange edges

Applications

- A spanning tree connects all nodes with as few edges as possible



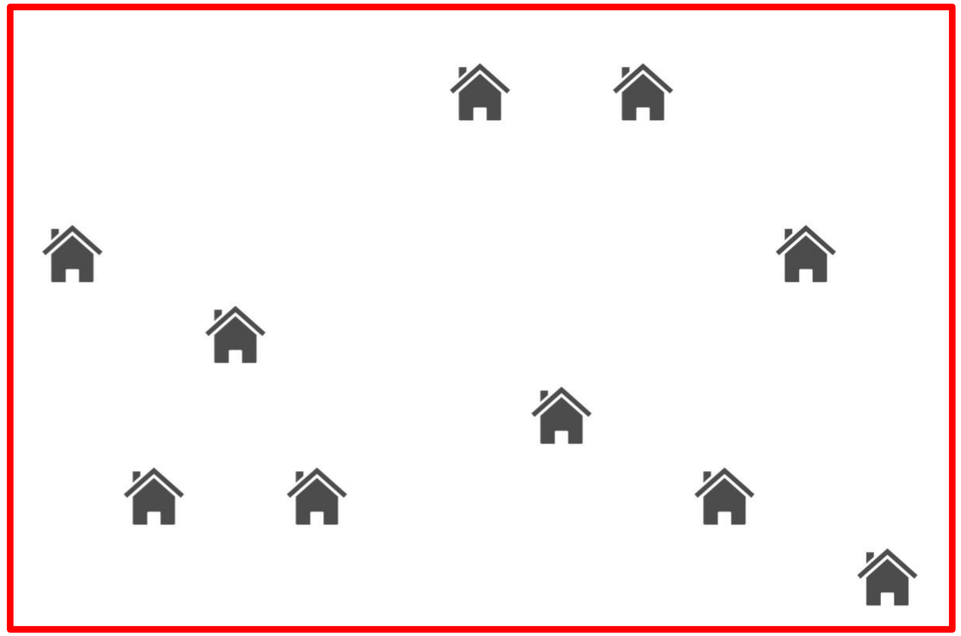


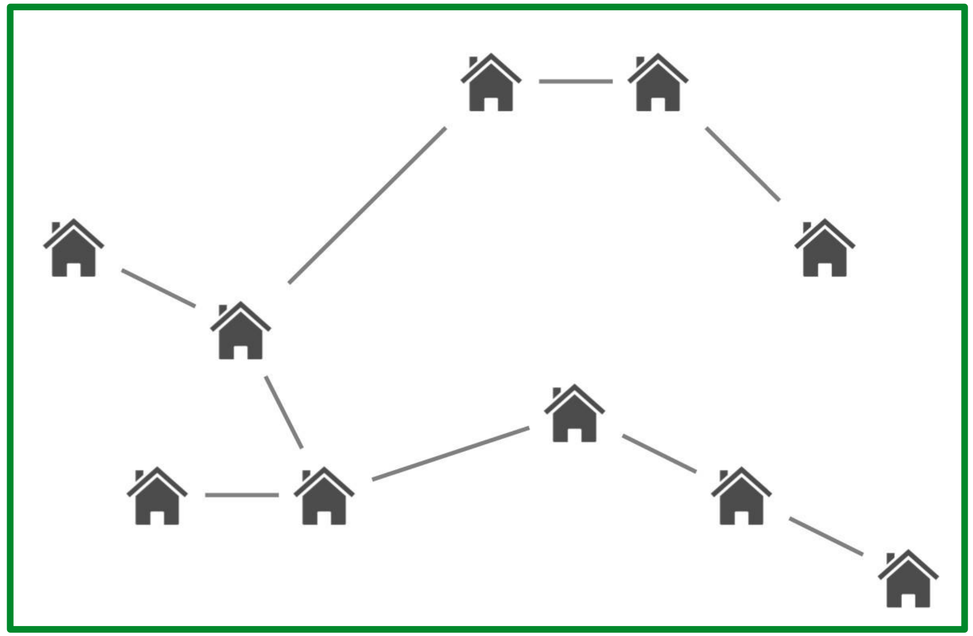
Ex-1: A local area network so everybody gets the message and no unnecessary messages get sent

Ex-2: Connecting computers with wires

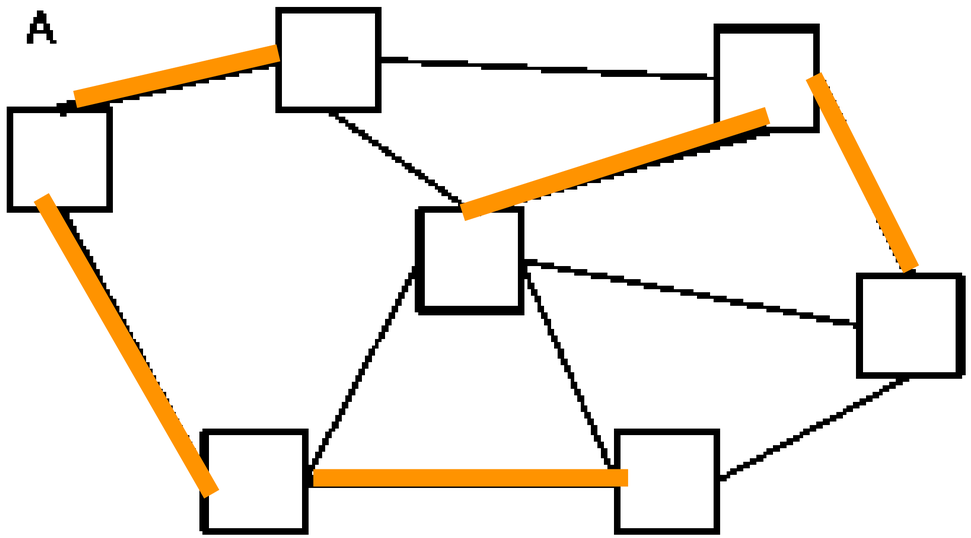
Applications (2/2)

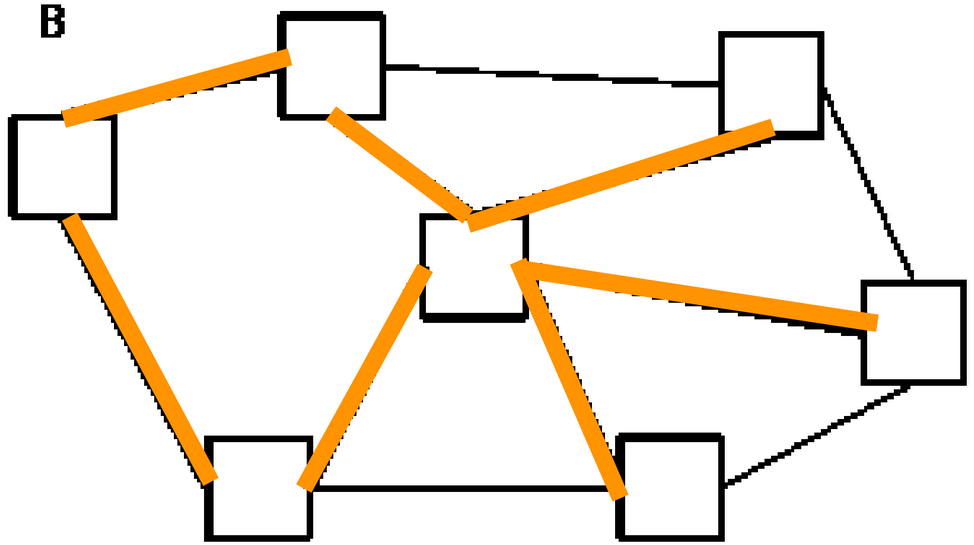
- Ex-3: Building roads
  - Other similar examples: electrical wiring for a house or wires on a chip

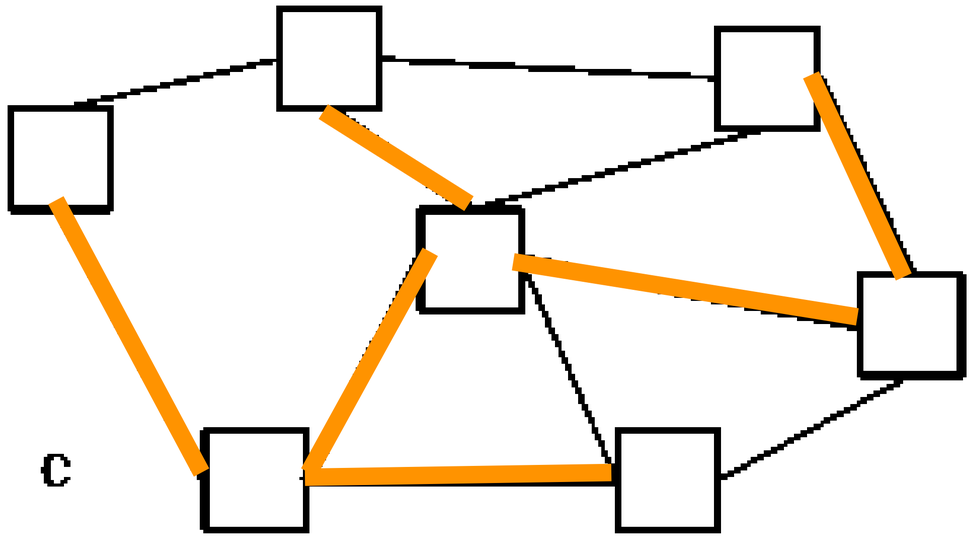




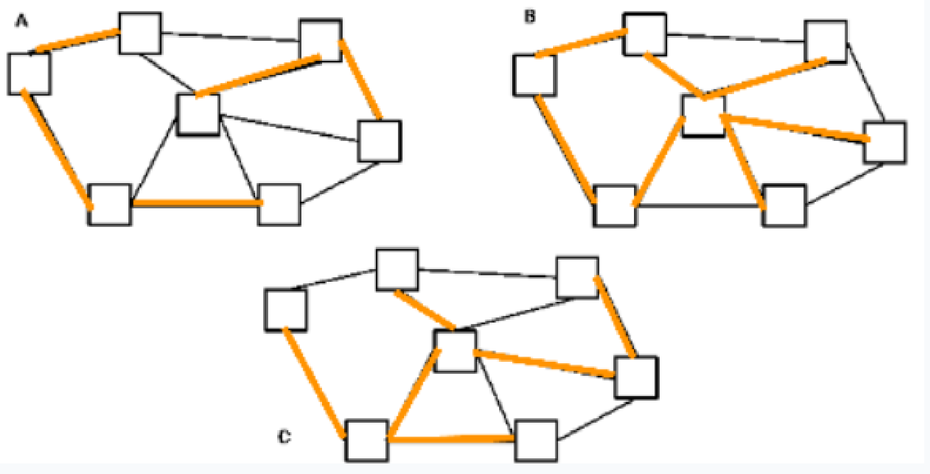
Which of the following are Spanning Trees?







Which of the following is/are not a spanning tree?



A

0%

B

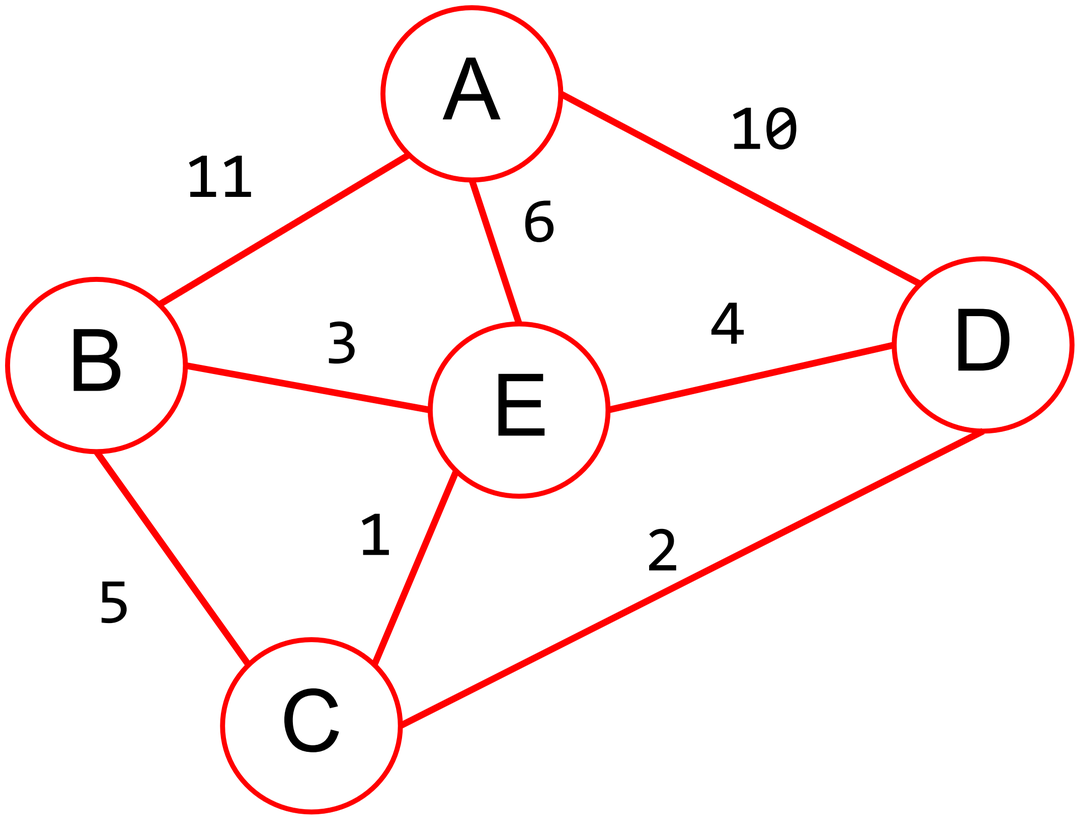
0%

C

0%

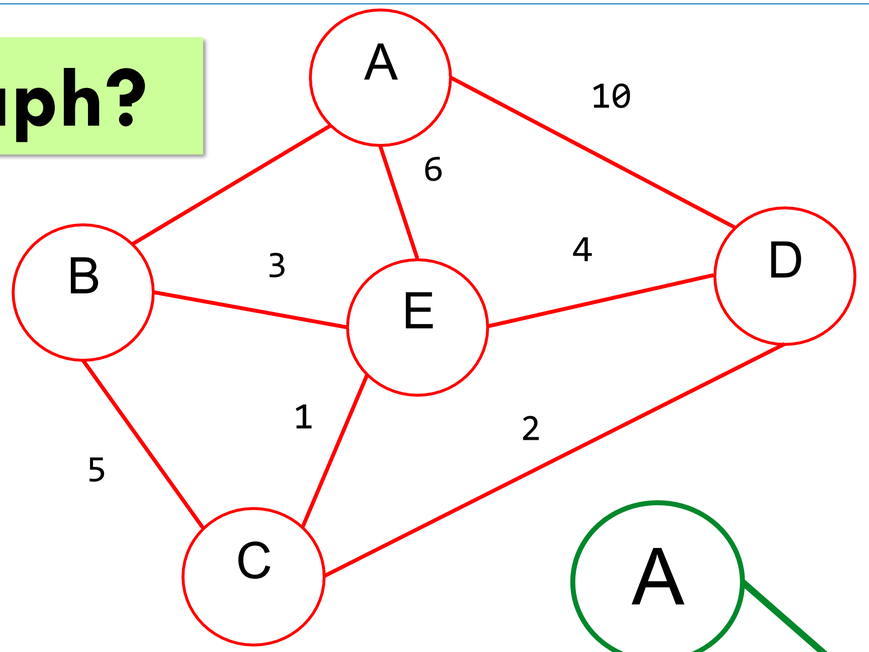
Finding Minimum Spanning Trees (MSTs)

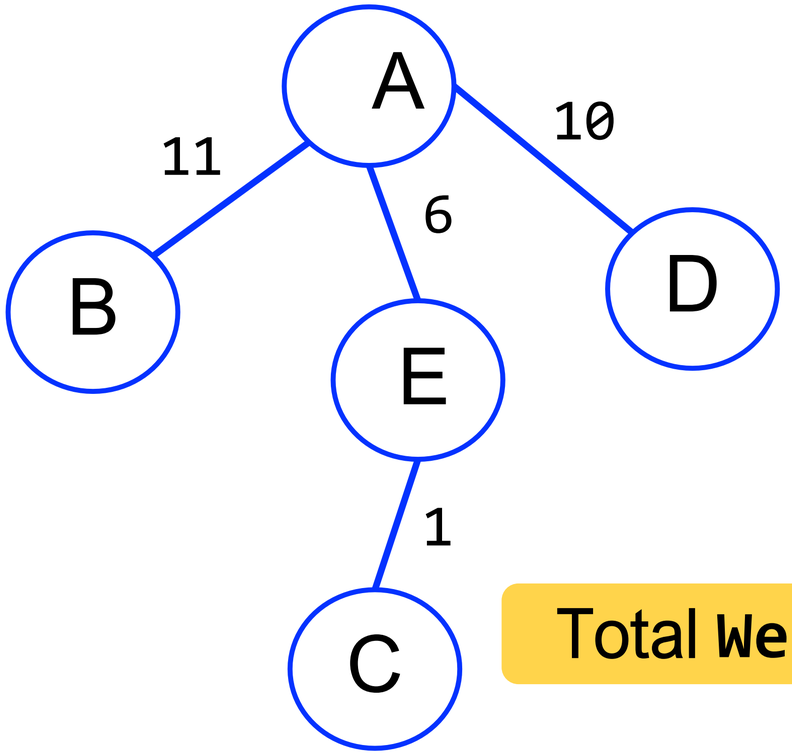
- Suppose we are given a weighted graph G=(V,E). We would like to find a tree of G with minimum total weight of edges



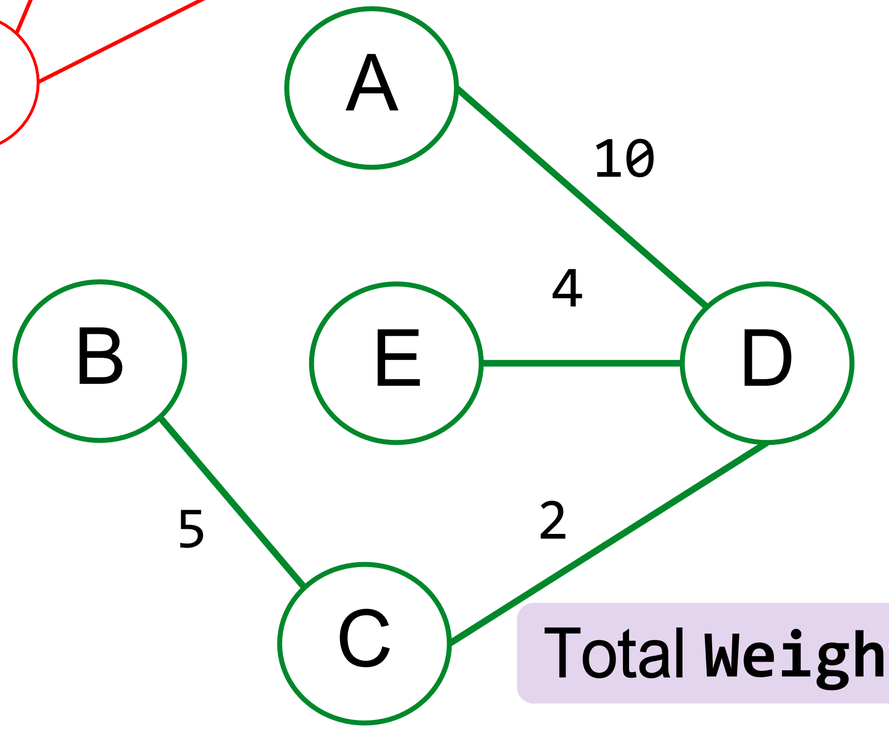
Minimum Spanning Trees (MSTs)

What is the MST for this graph?



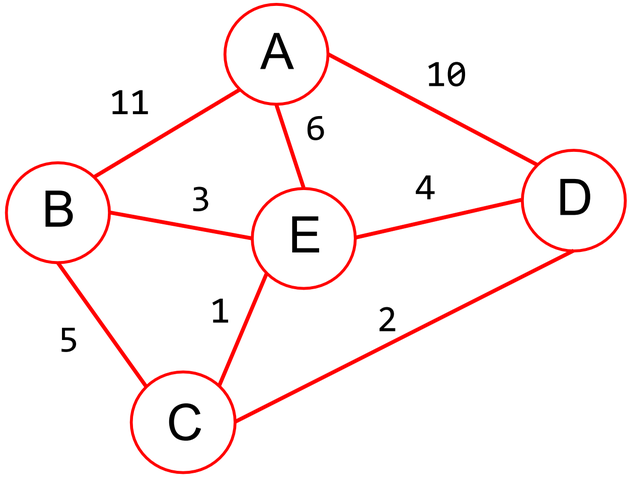


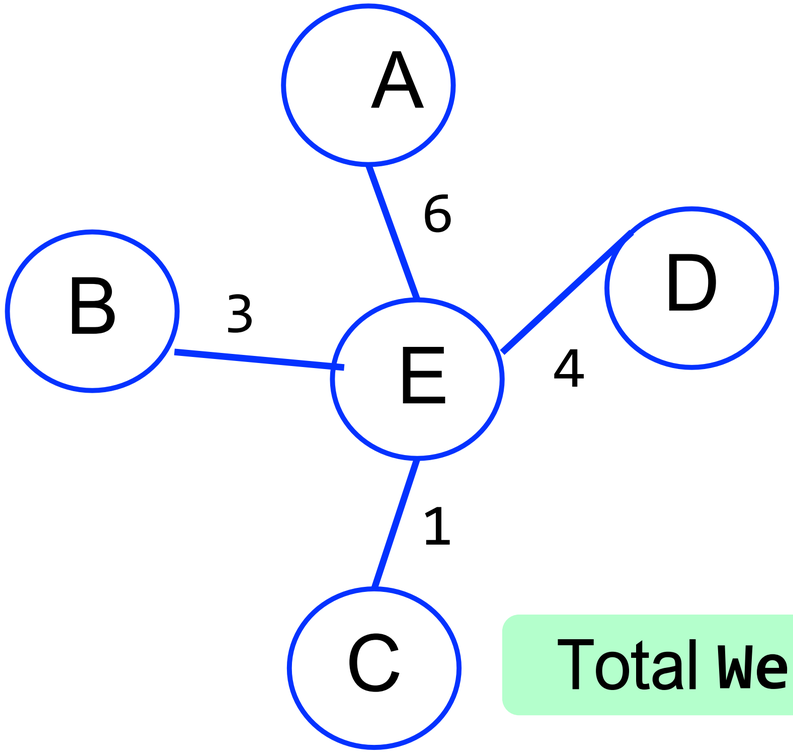
Total **Weight** = 28



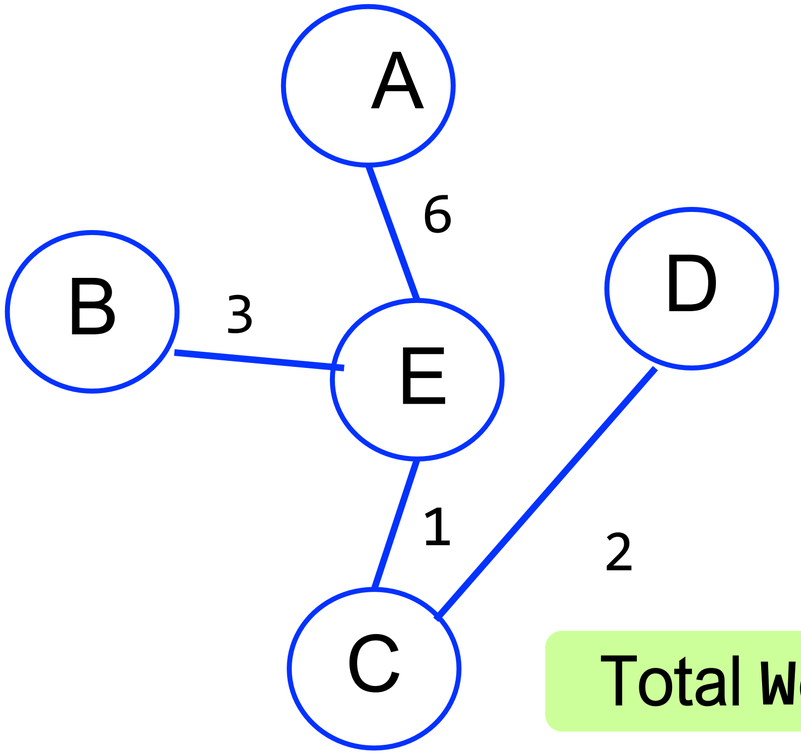
Total **Weight** = 21

Minimum Spanning Trees (MSTs)



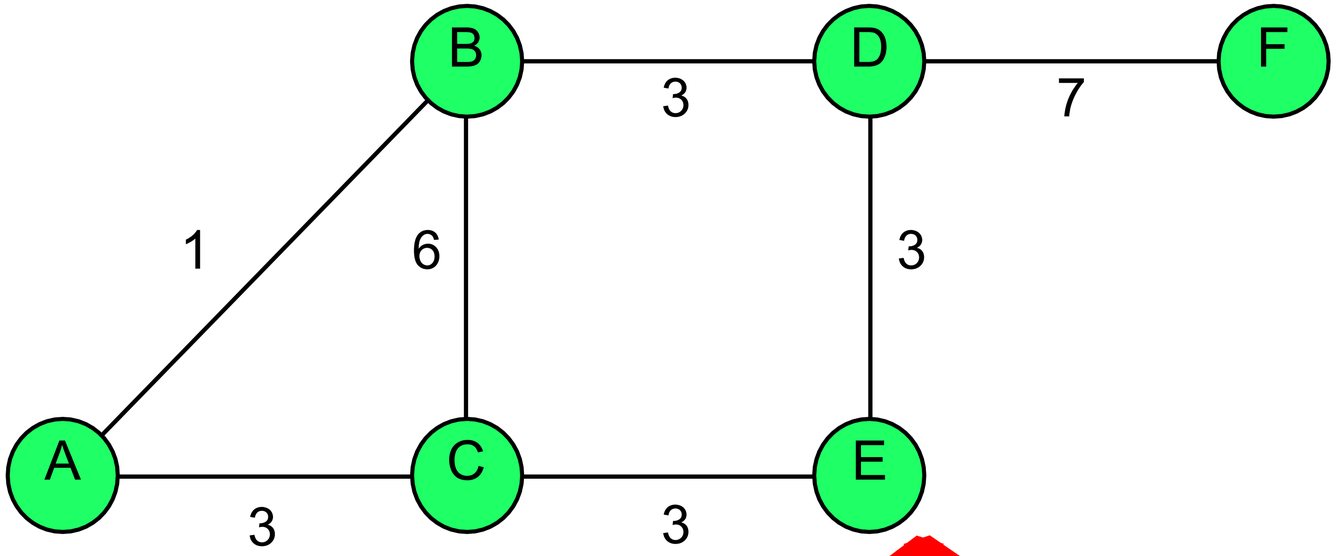


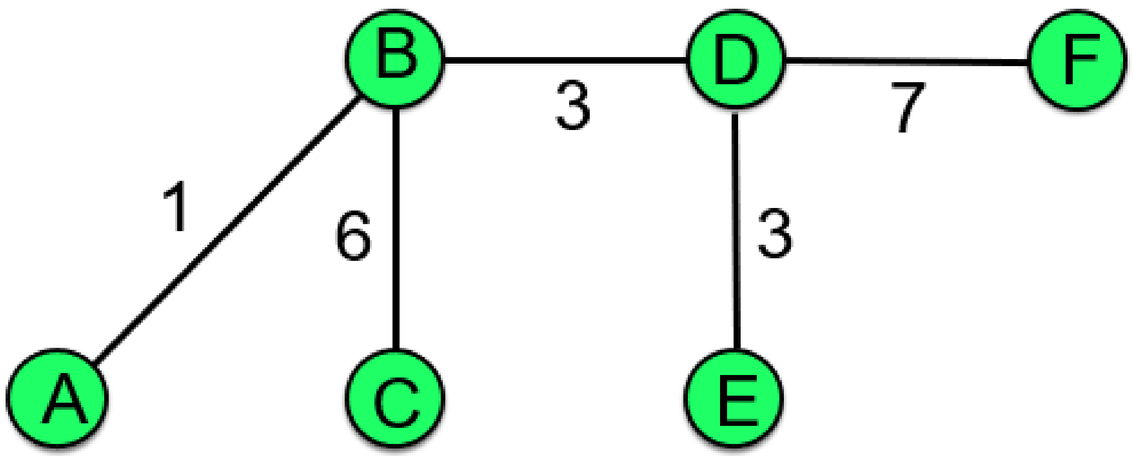
Total **Weight** = 14

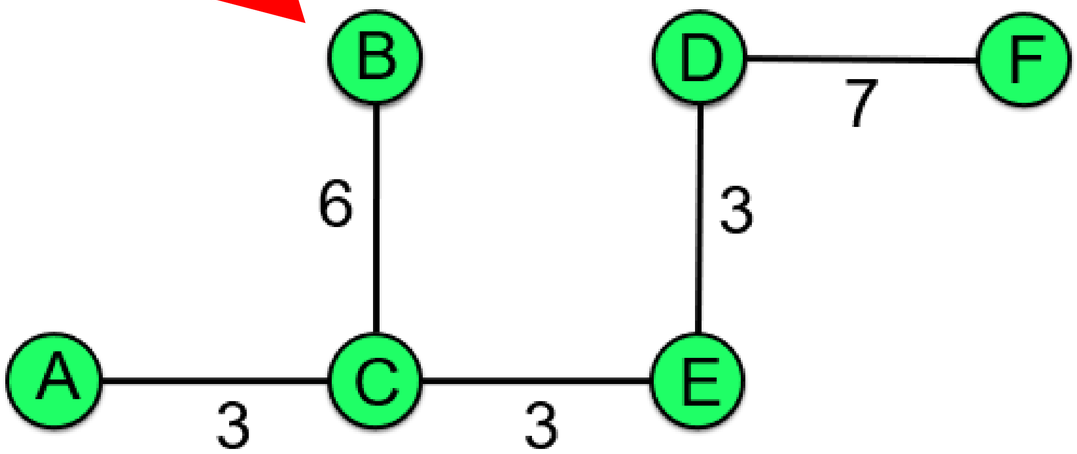


Total **Weight** = 12

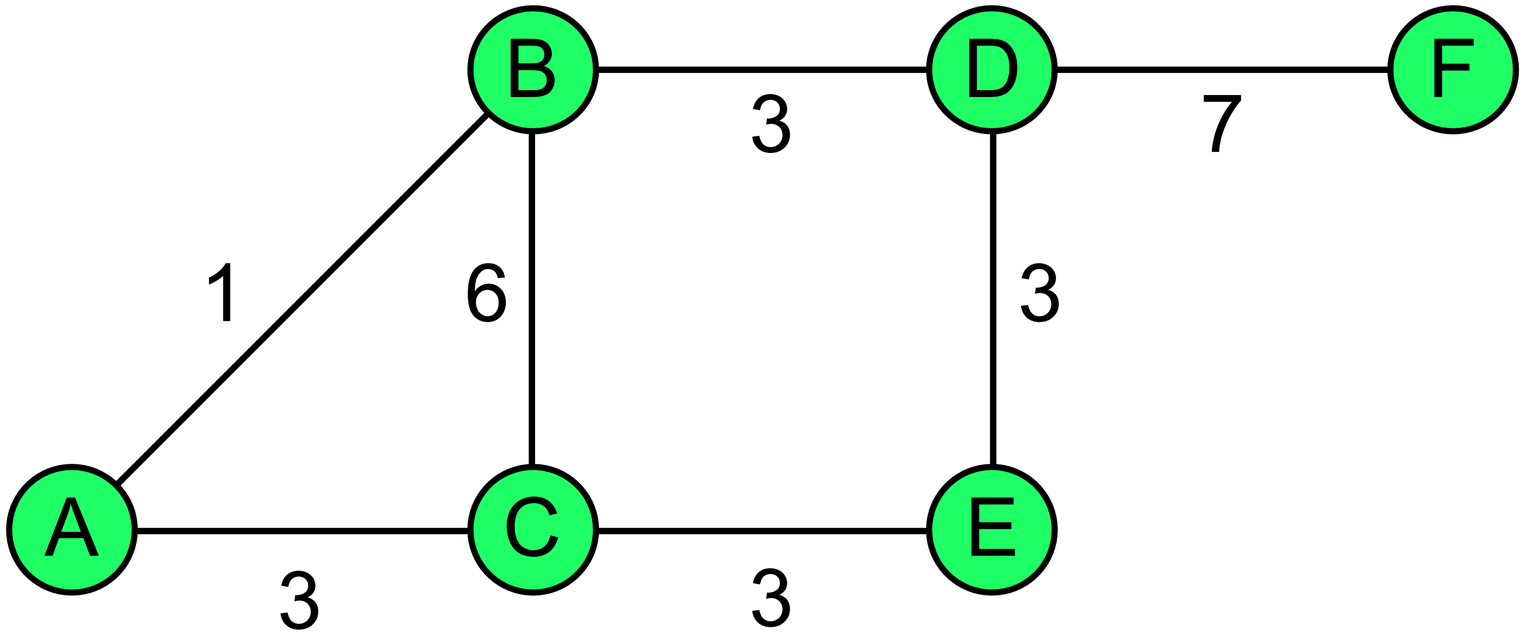
Multiple distinct spanning tree in a graph



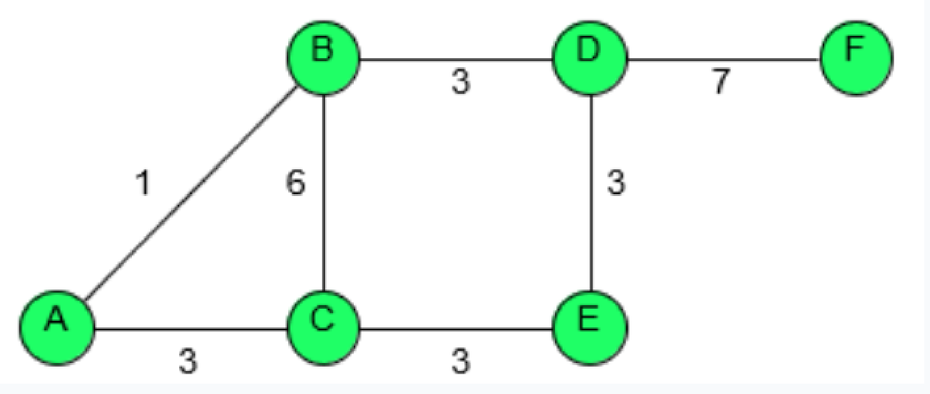




How many distinct MSTs are in this graph?



How many distinct MSTs are in the following graph?



0-1 0%

2-3 0%

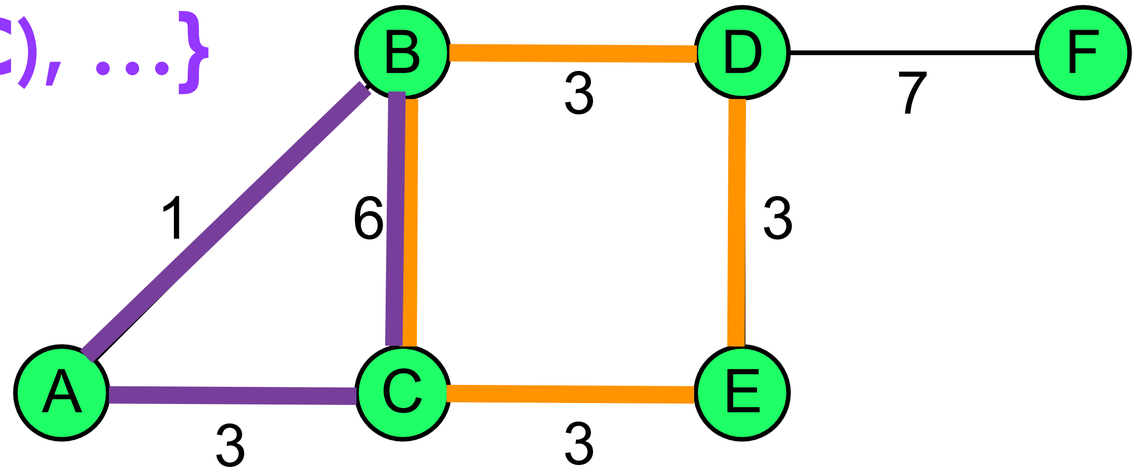
4-5 0%

> 5 0%

Let's Discuss!

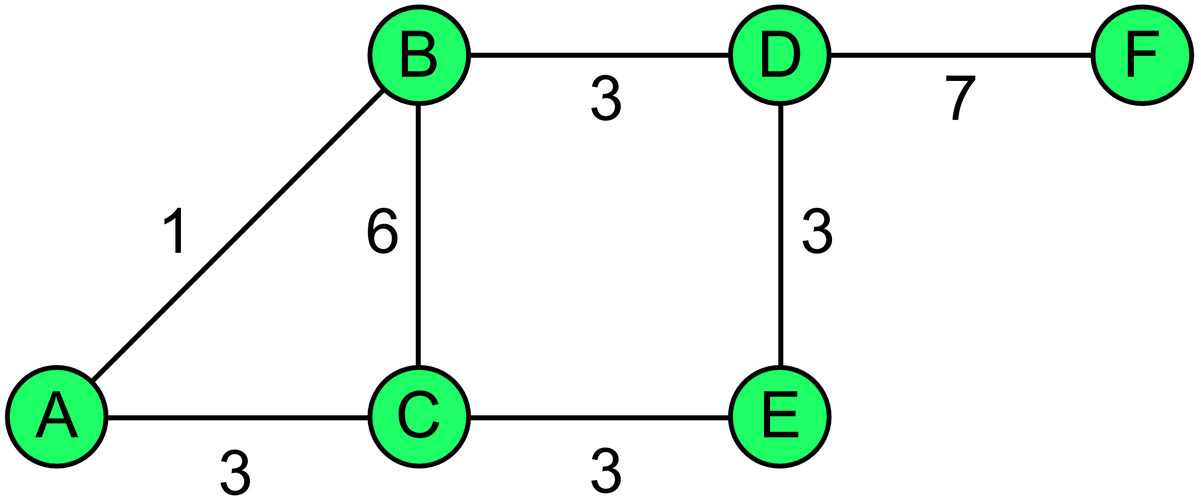
- There are three cycles in this graph:
  1. {A, B, C}
  2. {B, C, D, E}
  3. {A,B,D,E,C}

shared edges = **{(B,C), …}**



Let's Discuss

- There are edges with the same weight!
- In any MST, 6 won't be there!
- We are now left with 4 edges with weight 3, removing one will give us a MST!
- Thus, there are 4 distinct MSTs



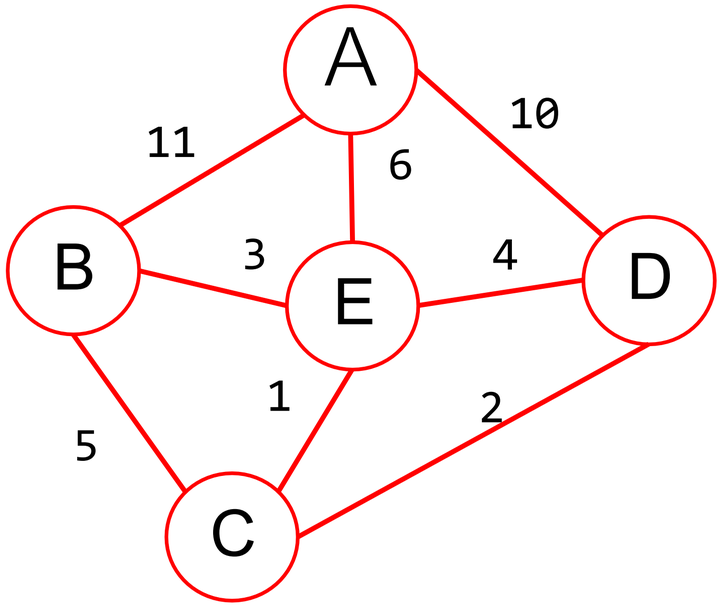
Key Observations

- Are MSTs unique for the graph?
  - … unless original graph was already a tree
- Can we find MST in disconnected graph?
- Edges in MST?
  - A tree with |V| nodes has |V|-1 edges
  - So, every solution to the spanning tree problem has |V|-1 edges

So, how can we find the MST of a graph?

Problem: How to Find MST of a Graph?

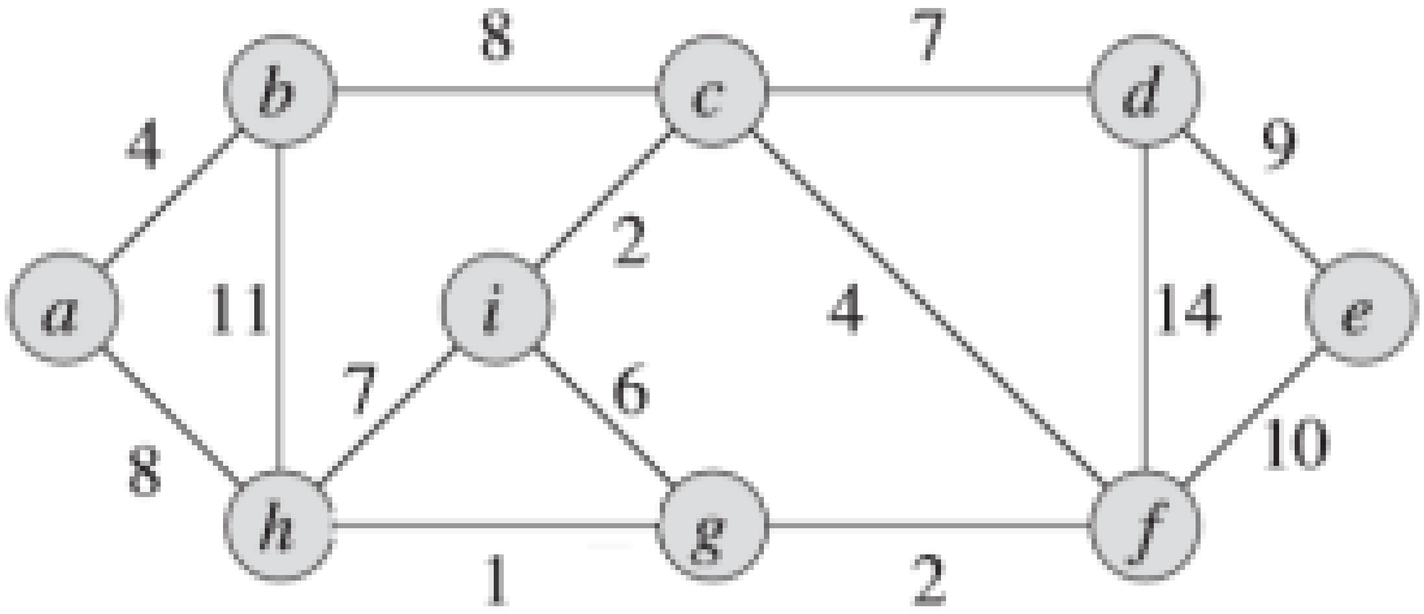
- Let G=(V,E) be an undirected graph
- T=(V,F) of G is a graph with the same vertices as G and |V|- 1 edges that form a tree (Spanning Tree)
- If G is not connected, T is a Forest (or collection of trees)



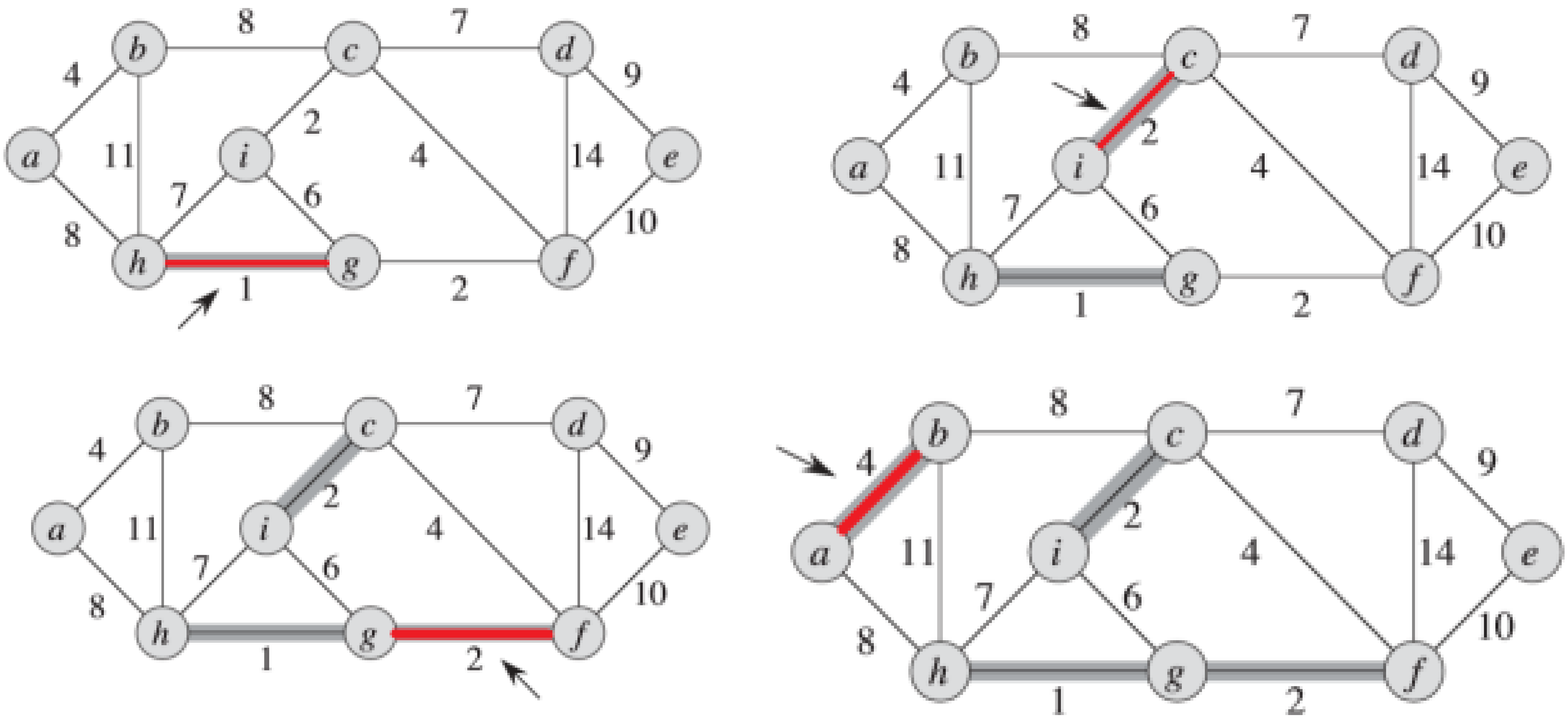
Kruskal's Algorithm

```
(1) Create a new graph T with same vertices as  G
    but no edges yet
(2) Make a list of all edges in G
(3) Sort edges by weight (lowest to highest)
(4) Loop through edges in sorted order
        For each edge (u,w)
            if u & w are not connected by a path then we
            connect them, i.e., add (u,w) to T
```

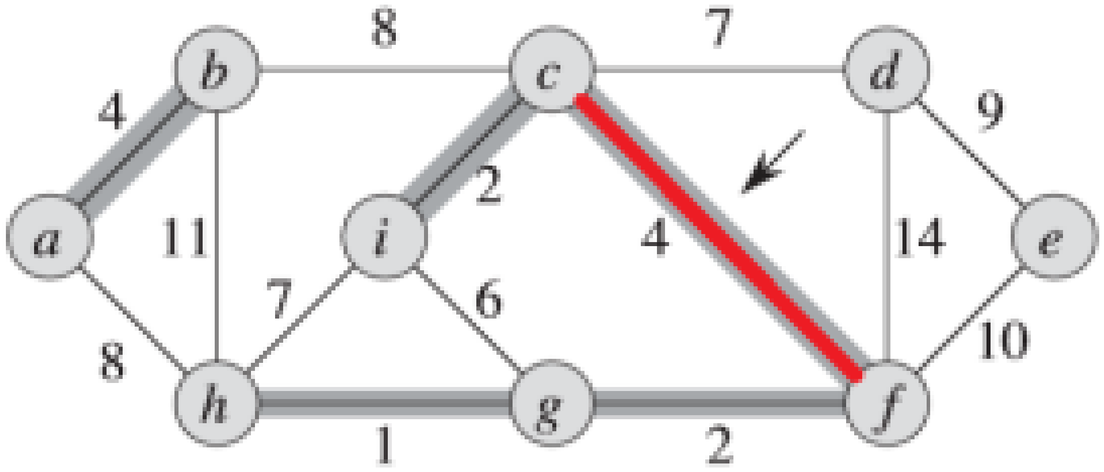
Finding MST?

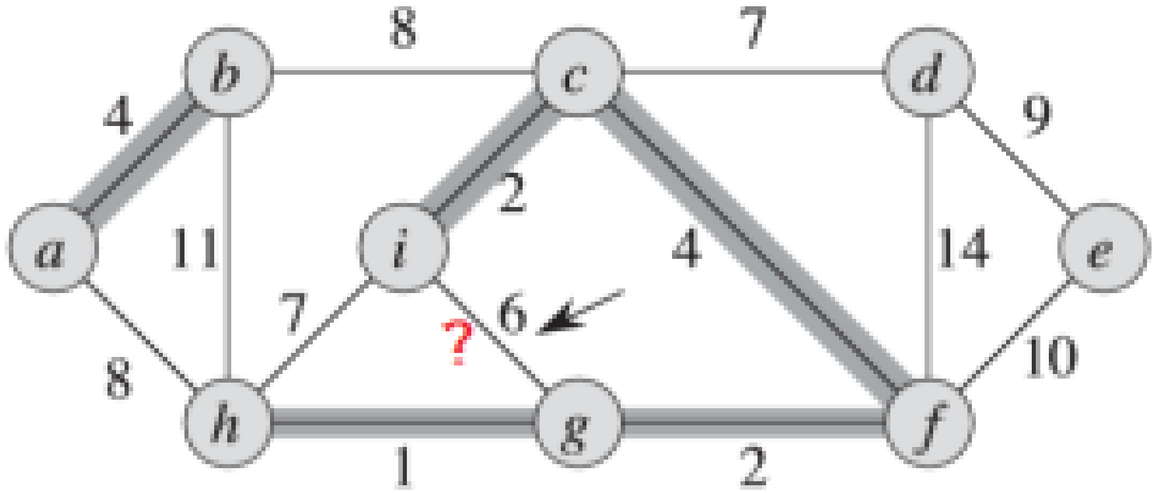


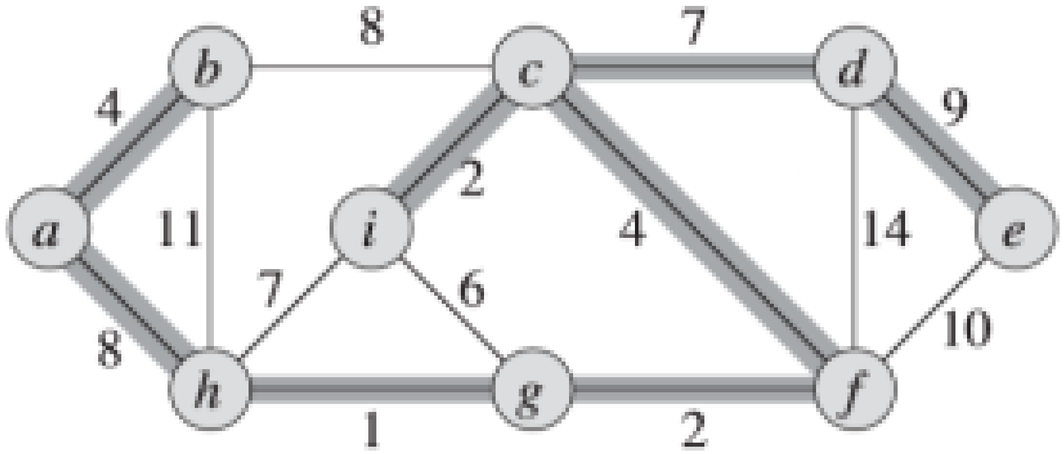
Finding MST


a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 8 10 1 2

Finding MST


a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
2
4
7
6
10
8
1
2


a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
2
4
7
?
6
10
8
1
2


a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
2
4
7
6
10
8
1
2

Time Complexity

- AL representation
  - O(|V|) // Creating a graph T with |V| vertices but not edges
  - O(|E|) // Creating a list of edges
  - O(|E|log|E|) // Sorting edges
  - O(|E|) * O(|V|) // For each edge, checking if (u,v) are already connected in T
  - Total: O(|E||V|+|E|log E)

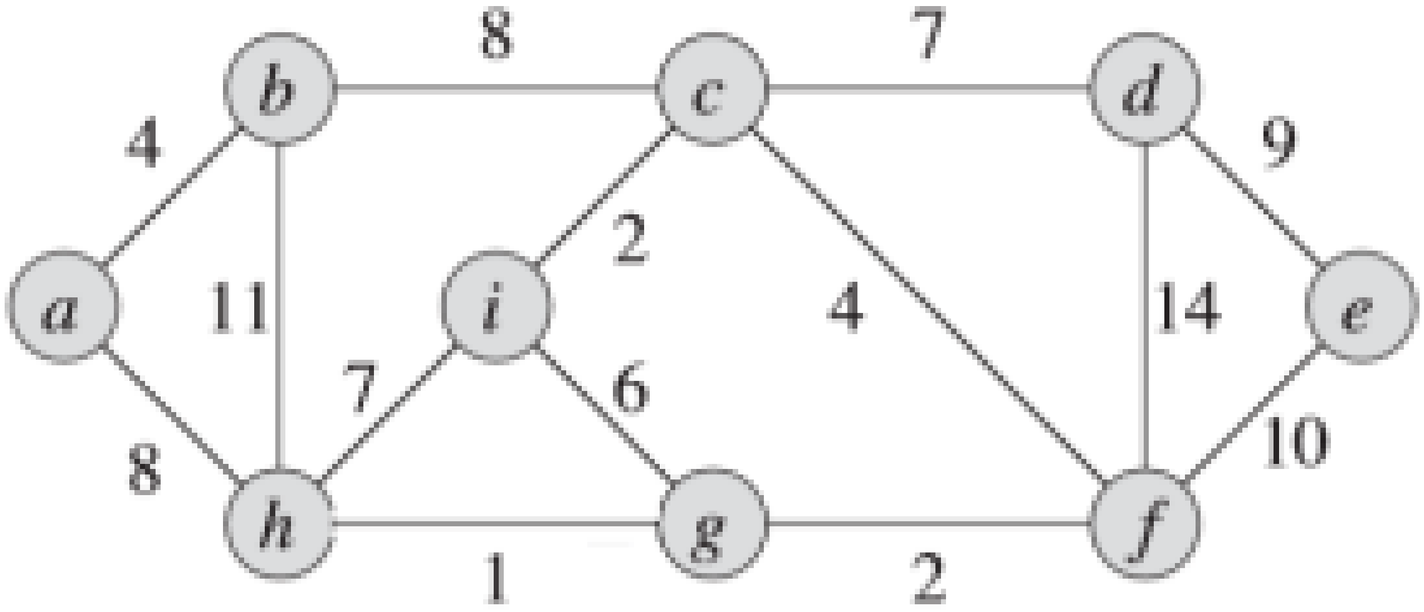
- Find the complexity with AM

Prim's Algorithm for finding MST

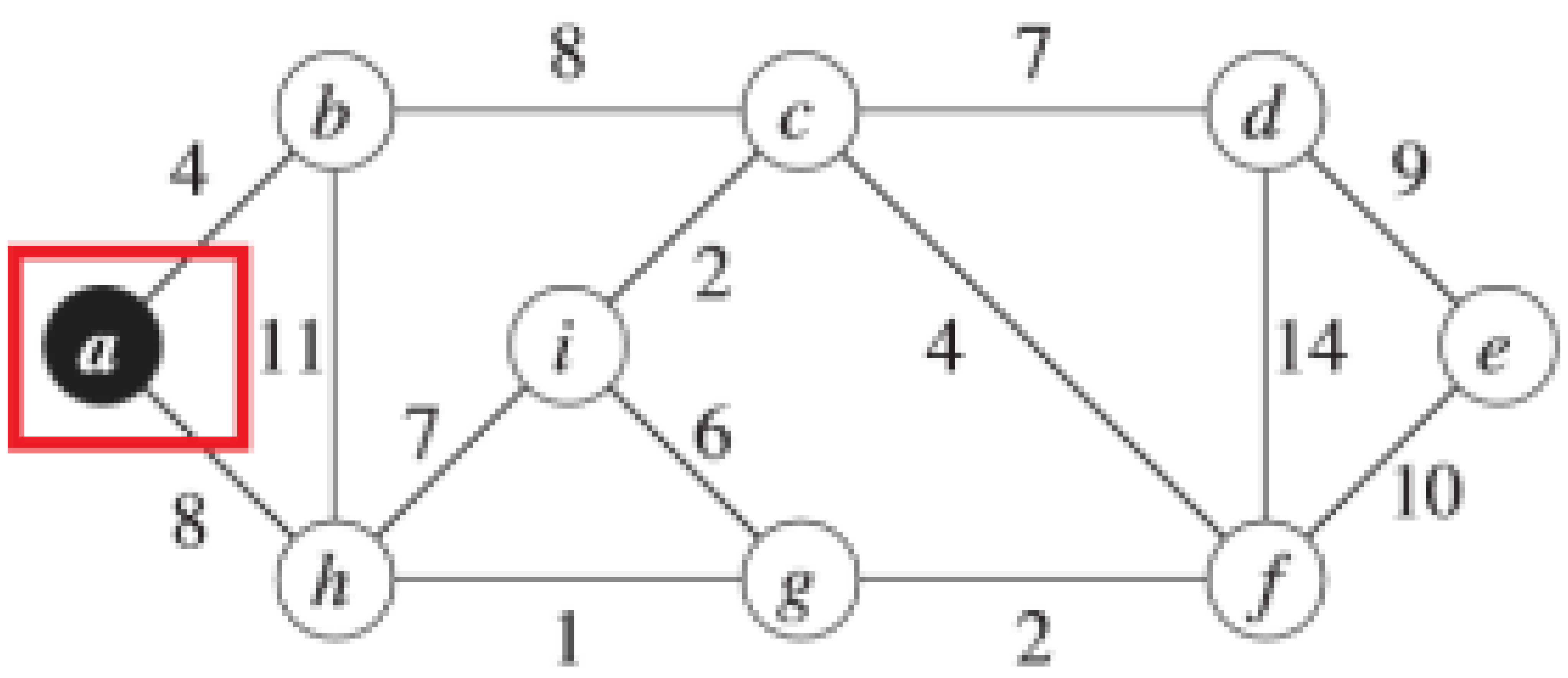
Create a new graph T with same vertices as G but no edges yet
Choose an arbitrary Starting vertex and add it to the visited set
While T does not have all vertices
    From the set of visited vertices, find the smallest weight edge that connects to an unvisited vertex
    Add that edge to T
    Add the newly reached vertex to the visited set

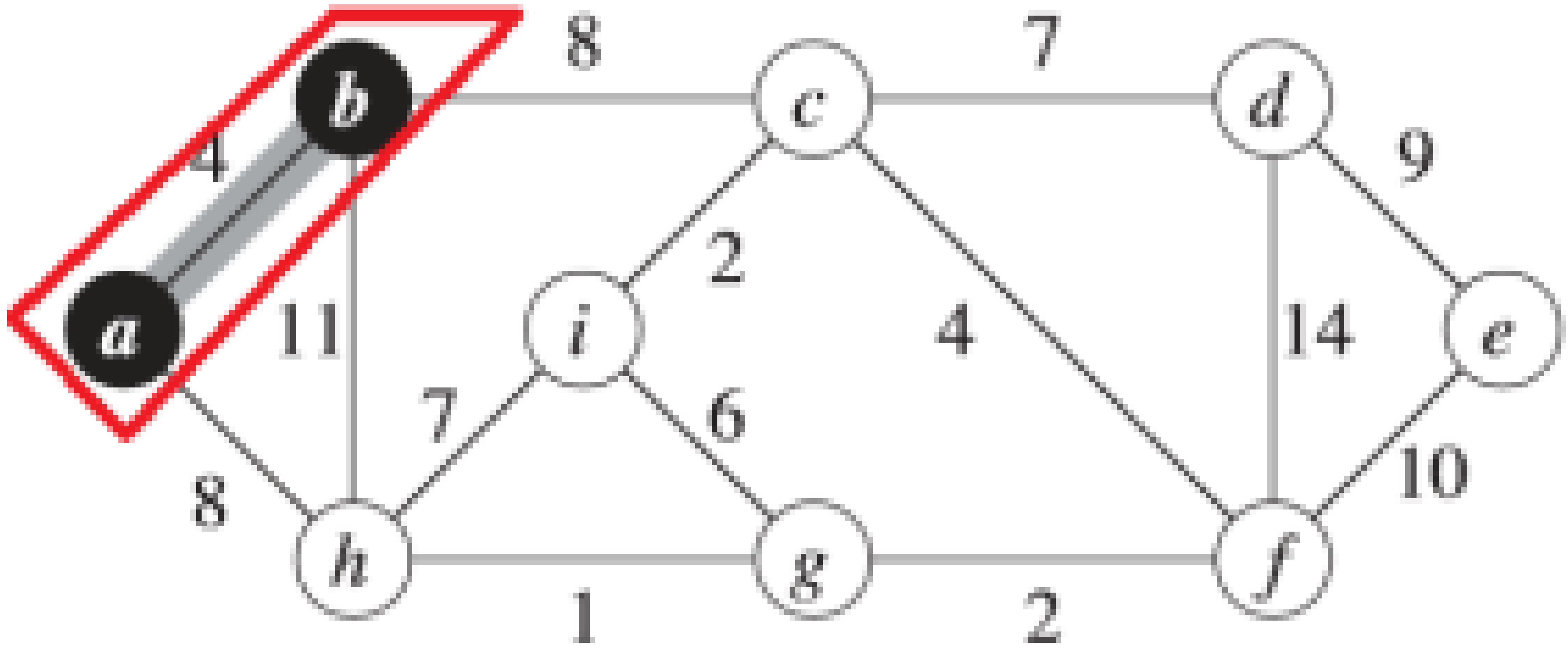
Finding MST – Approach 2

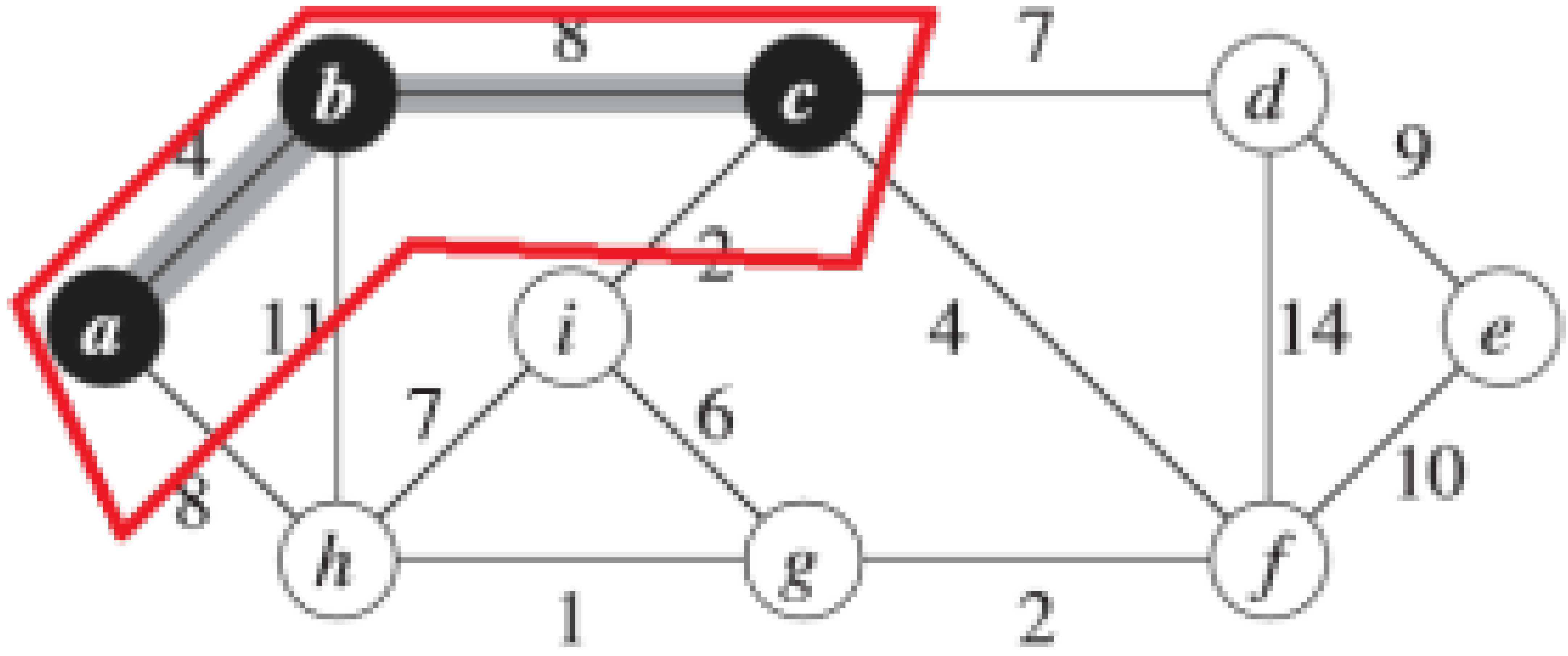
- Start from an arbitrary vertex

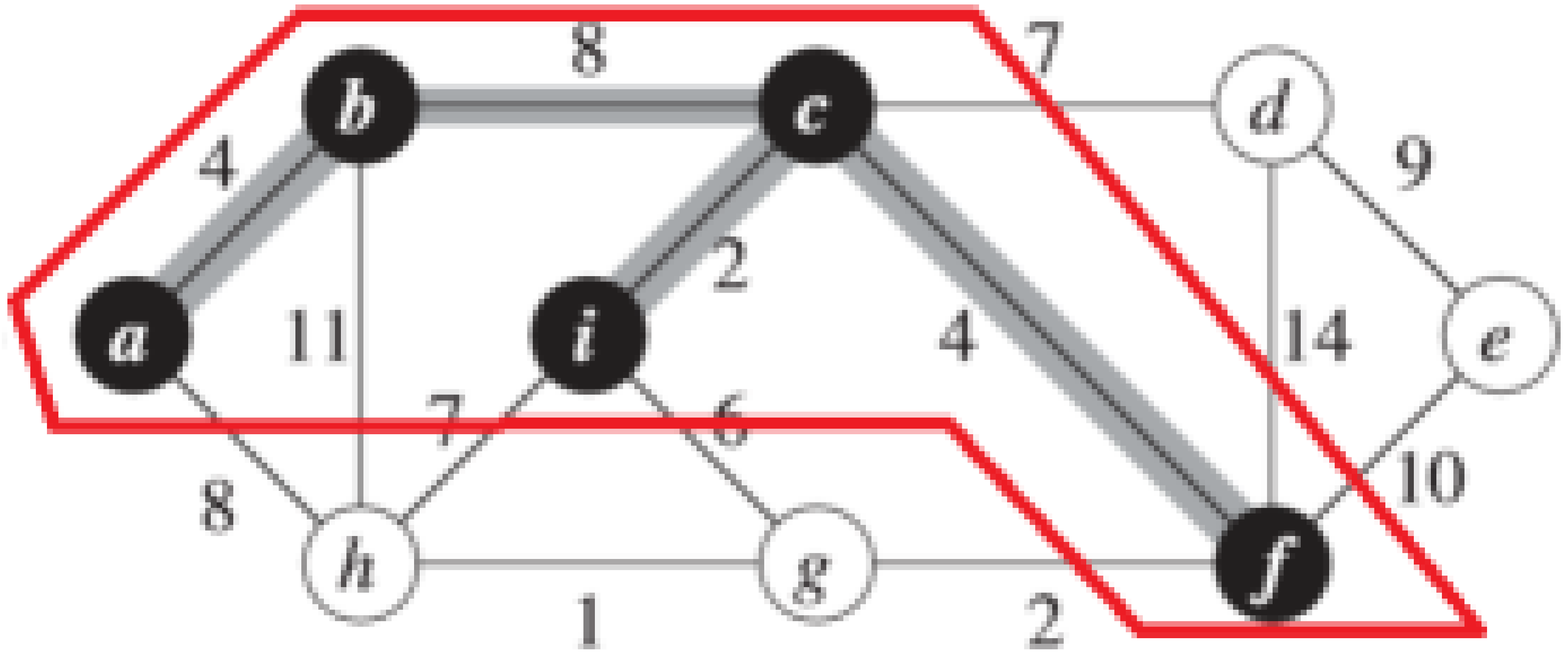


Finding MST – Approach 2

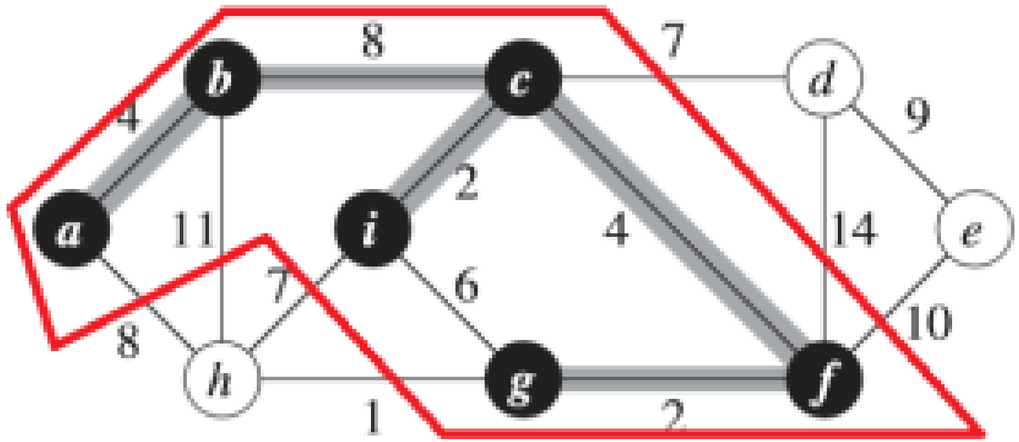


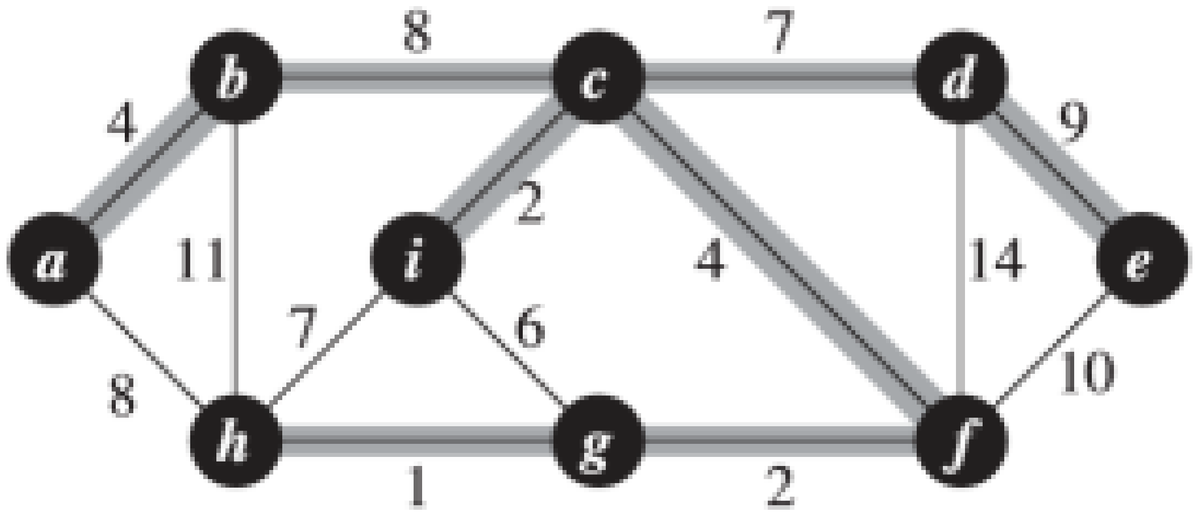






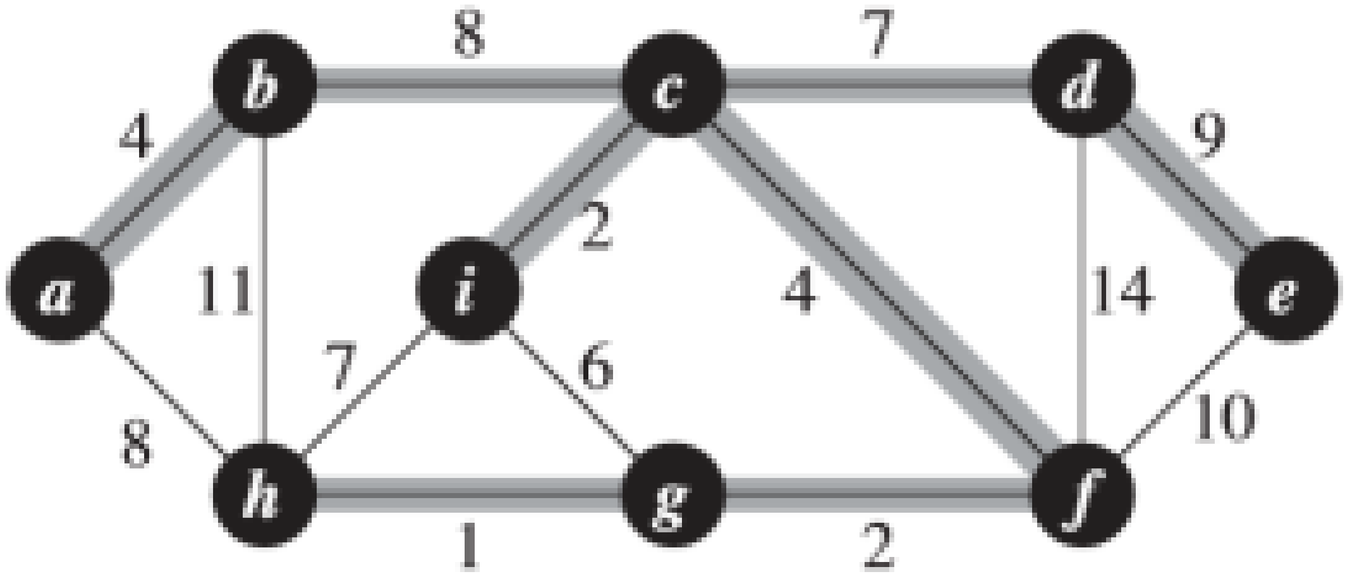
Finding MST – Approach 2





Prim's Algorithm

- Is it optimal?
- Is it the Minimum Spanning Tree (MST)?
- This is Prim's algorithm to compute MST



Prim's Algorithm – Time Complexity

- Time Complexity: O((E + V)log V)
  - Implemented using binary heap and adjacency list
    - Insertion – O(log V), total of V insertions O(V log V)
    - Remove min – O(log V), for V vertices O(E log V)
    - Edge selection – O(E)
  - What will be the time complexity, if we use adjacency matrix?

Kruskal's Algorithm

- Simple to implement
- Works well with edge lists
- Slower for dense graphs due to sorting all edges

Prim's Algorithm

- Efficient for dense graphs
- Naturally fits with adjacency list representation
- Needs a priority queue

You are running Kruskal's algorithm on a connected graph G and you come across an edge with weight 10 that connects two vertices already in the same tree. What should Kruskal's algorithm do?

Include the edge

0%

Exclude the edge

0%

Include it only if it's the lightest edge

0%

Restart from other vertex

0%

Which of the following is true about Kruskal's and Prim's algorithms?

Both always give the same spanning tree

0%

Prim’s works without a priority queue

0%

Prim’s algorithm may form cycles temporarily

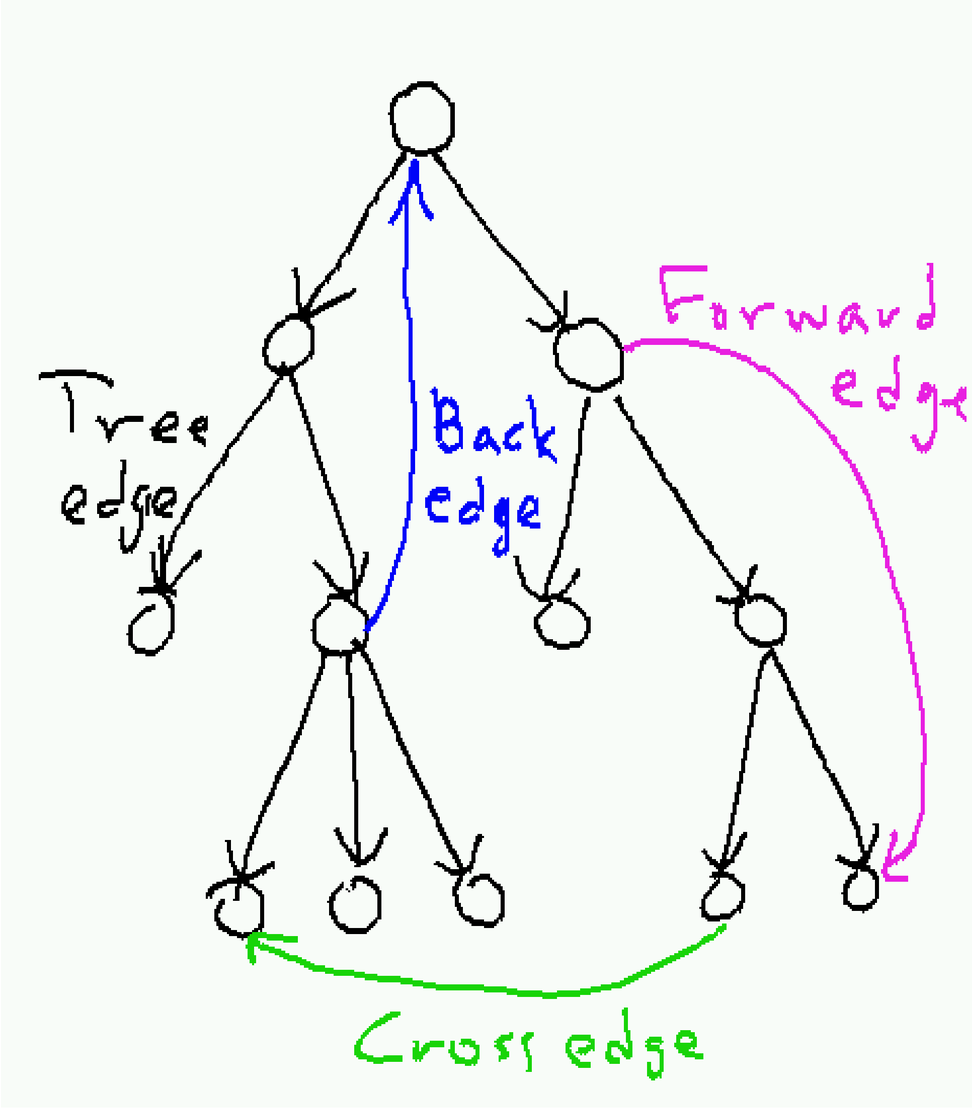
0%

Kruskal’s algorithm may work with disconnected graphs

0%

Edge Types

- Tree Edge
  - Leads to unvisited vertex, i.e., first discovery of the node
- Back Edge
  - Points to an ancestor in the graph,
- Forward Edge
  - Points to descendant already explored
- Cross Edge
  - Connect nodes in the different branches



Thank you!