# MATH-810 Mathematical Methods for Artificial Intelligence

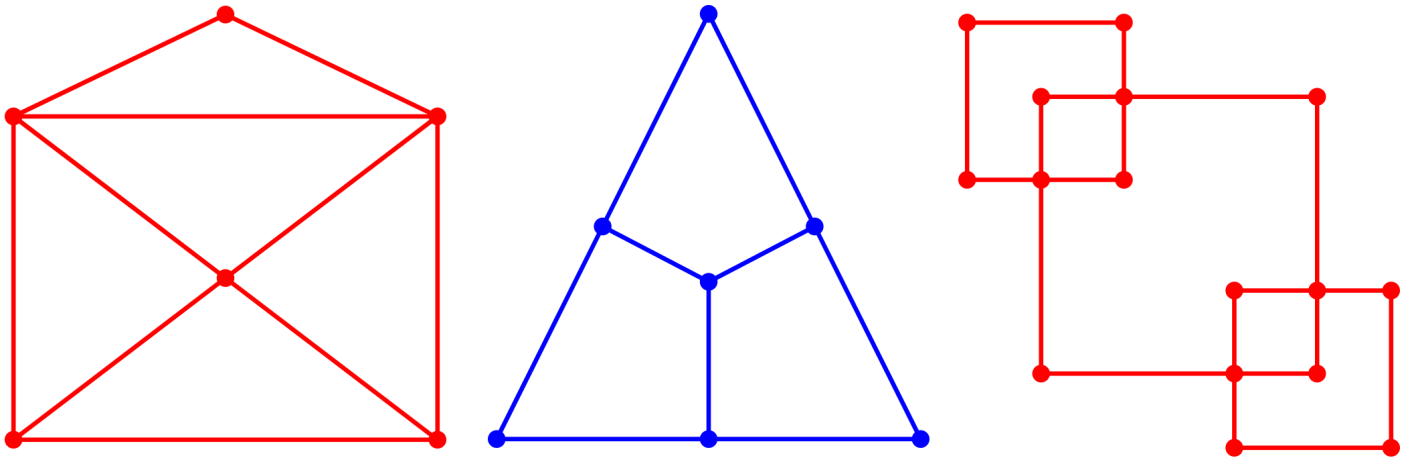
## Introduction to Graph Theory

Dr. Yasir Ali

DBS&H, CEME
National University of Sciences and Technology
Islamabad, Pakistan.

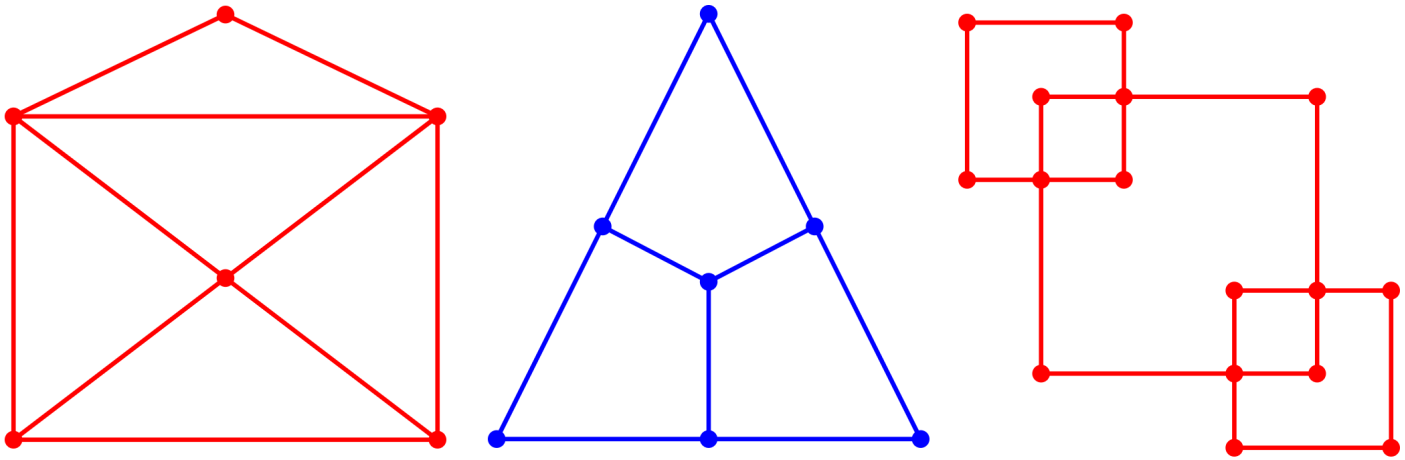
September 24, 2025

# It's Puzzle Time!



- Which of these can you draw without lifting your pencil, drawing each line only once?

# It's Puzzle Time!



- Which of these can you draw without lifting your pencil, drawing each line only once?

- Can you start and end at the same point?

# Eulerian and Hamiltonian Circuit

## Eulerian Path, Circuit, and Graph

**Eulerian Path** An Euler path is $G$ is a path that includes each edge of $G$ exactly once

**Eulerian Circuit** An Eulerian circuit in $G$ is an Eulerian path in $G$ whose end points are identical

**Eulerian Graph** A graph $G$ is said to be Eulerian if it has an Eulerian circuit

## Hamiltonian Path, Circuit, and Graph

**Hamiltonian Path** A path in a graph $G$ is called a Hamiltonian path if it contains every vertex of $G$ exactly once

**Hamiltonian Circuit** A cycle in $G$ is said to be a Hamiltonian cycle if it contains every vertex of $G$

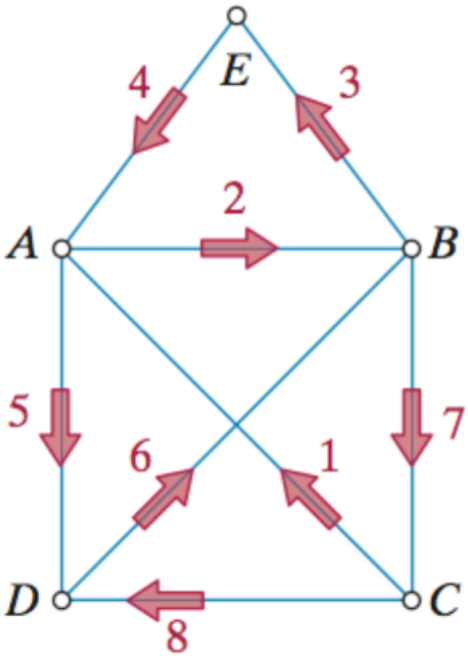
**Hamiltonian Graph** A graph is said to be a Hamiltonian graph if it contains Hamiltonian cycle

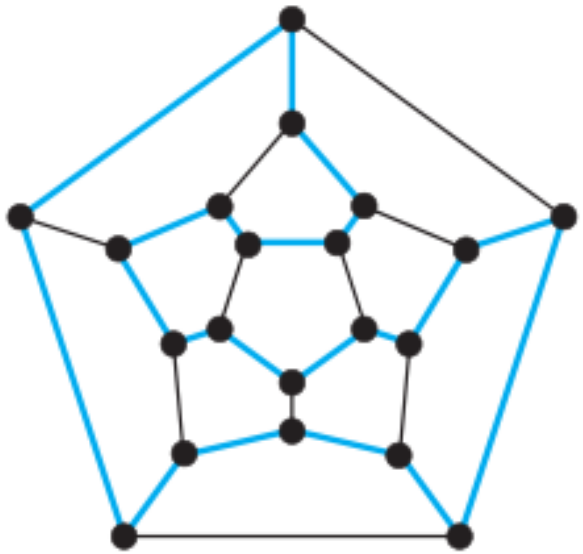
## Eulerian Path:

- This path visits every **edge** of the graph exactly once.
- It does not necessarily start and end at the same vertex.

## Hamiltonian Path:

- A path that visits each **vertex** exactly once.
- It does not necessarily start and end at the same vertex.



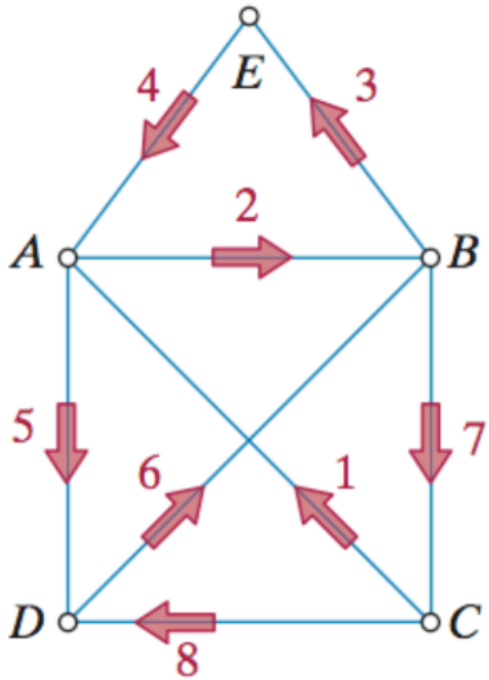
# Properties of Eulerian Graph

**The following statements are equivalent for a connected a graph $G$.**

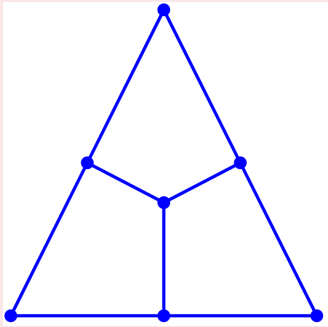
1. $G$ is Eulerian.
2. Every point of G has even degree.
3. The set of edges of G can be partitioned into cycles.

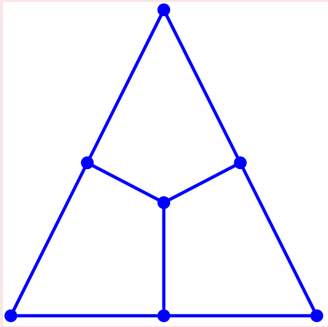
(i) If $G$ is connected graph and has exactly two vertices of odd degree, there is an Euler path in $G$.

(ii) If a graph $G$ has more than two vertices of odd degree, then there can be no Euler path in $G$.

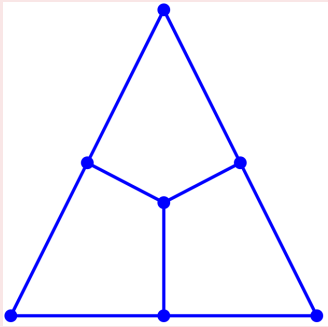



Any Euler path in graph G must begin at vertex of odd degree and end at the other.

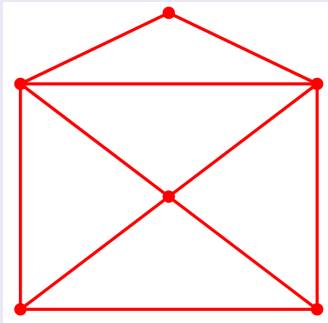


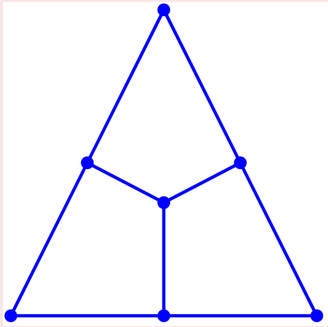


- More than two odd degree vertices
- One can not visit every edge exactly once.

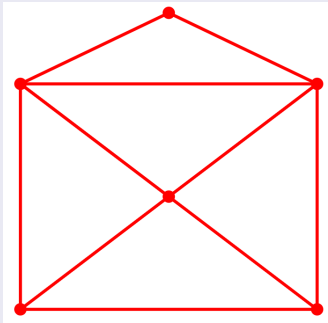


- More than two odd degree vertices
- One can not visit every edge exactly once.

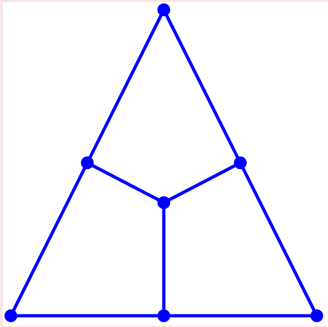




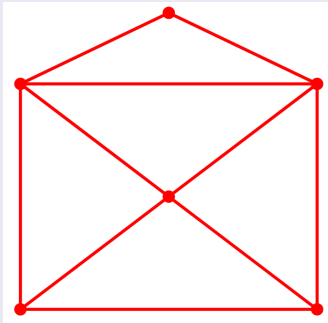
- More than two odd degree vertices
- One can not visit every edge exactly once.



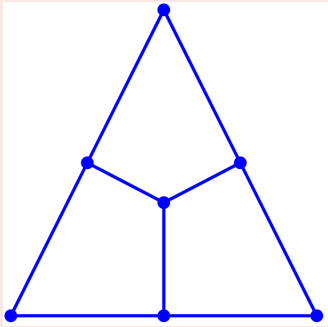
- Two odd degree vertices



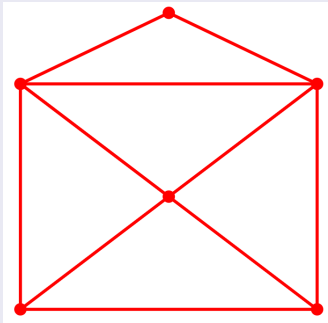
- More than two odd degree vertices
- One can not visit every edge exactly once.



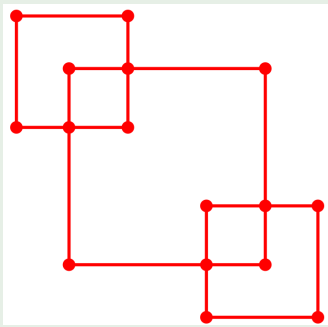
- Two odd degree vertices
- One can visit every edge exactly once.
- Starting and ending vertices will not be the same.



- More than two odd degree vertices
- One can not visit every edge exactly once.



- Two odd degree vertices
- One can visit every edge exactly once.
- Starting and ending vertices will not be the same.



- All vertices have even degree
- One can visit every edge exactly once
- Starting and ending vertices will be the same

## Algorithm to find Euler Circuit in $G$

1. Pick any vertex $v$ of $G$ at which to start.
2. Pick any sequence of adjacent vertices and edges, starting and ending at $v$(never repeating an edge). Call the resulting circuit $C_1$.
3. Check whether $C_1$ contains every edge and vertex of $G$. If so, $C_1$ is an Euler circuit, and we are finished. If not, proceed as follows:

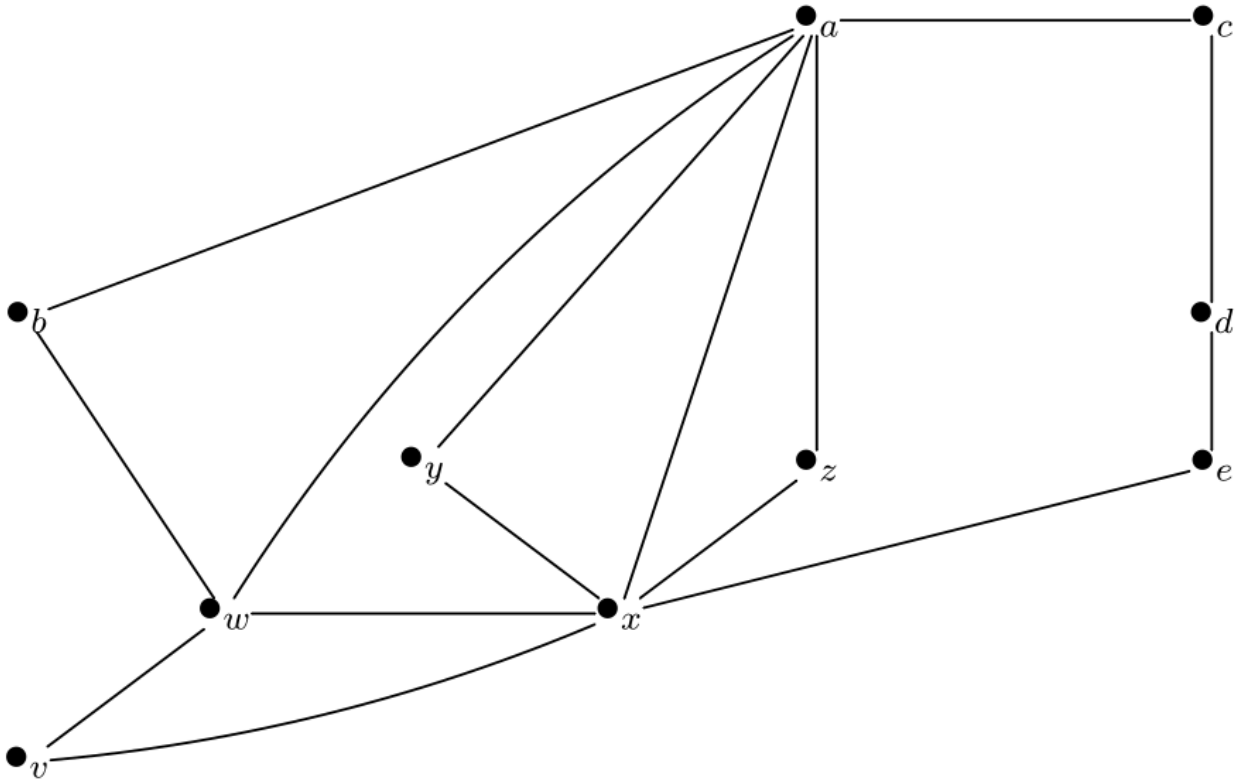
(i) Remove $C_1$ from $G$ the resulting subgraph is $G_1$

(ii) Pick any vertex $w$ common to both $C_1$ and $G_1$.

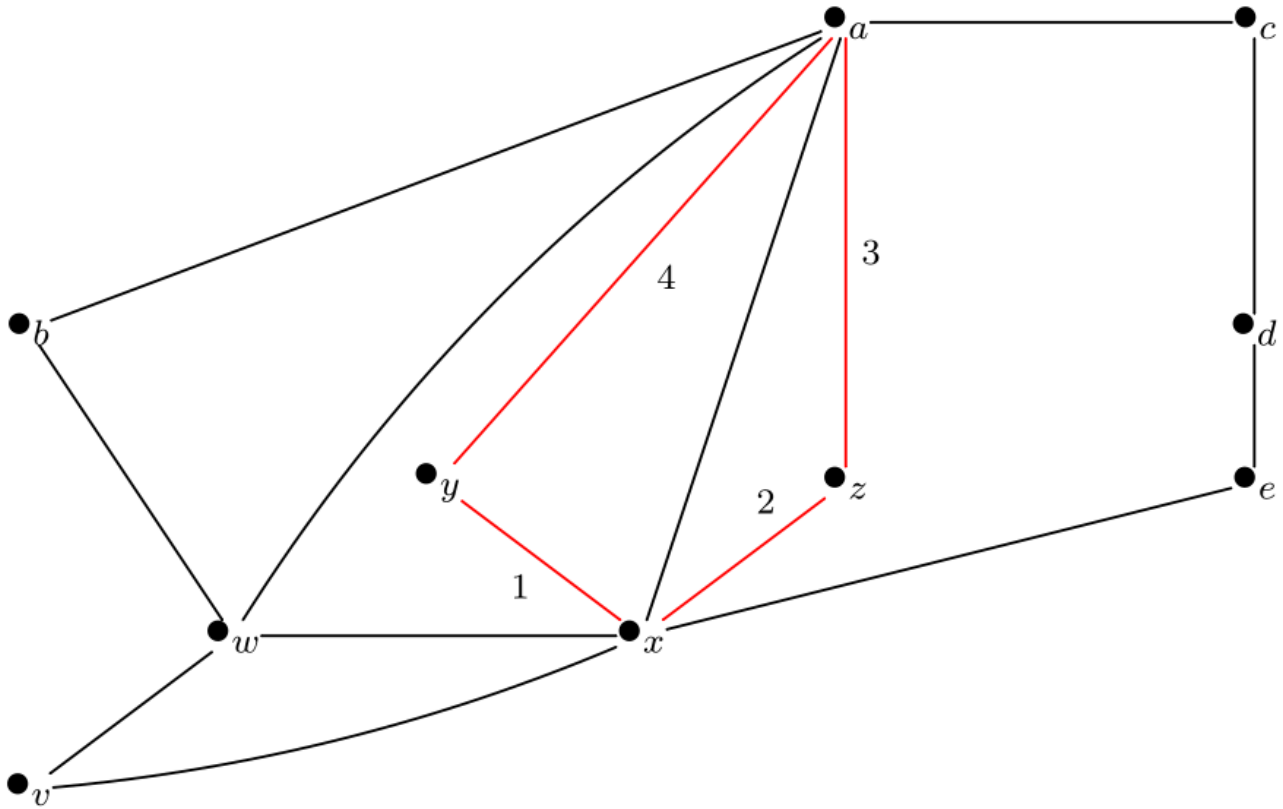
(iii) Pick any sequence of adjacent vertices and edges of $G_1$, starting and ending at $w$(never repeating an edge). Call the resulting circuit $C_2$

(iv) Patch $C_1$ and $C_2$ together to create a new circuit $C_3$.

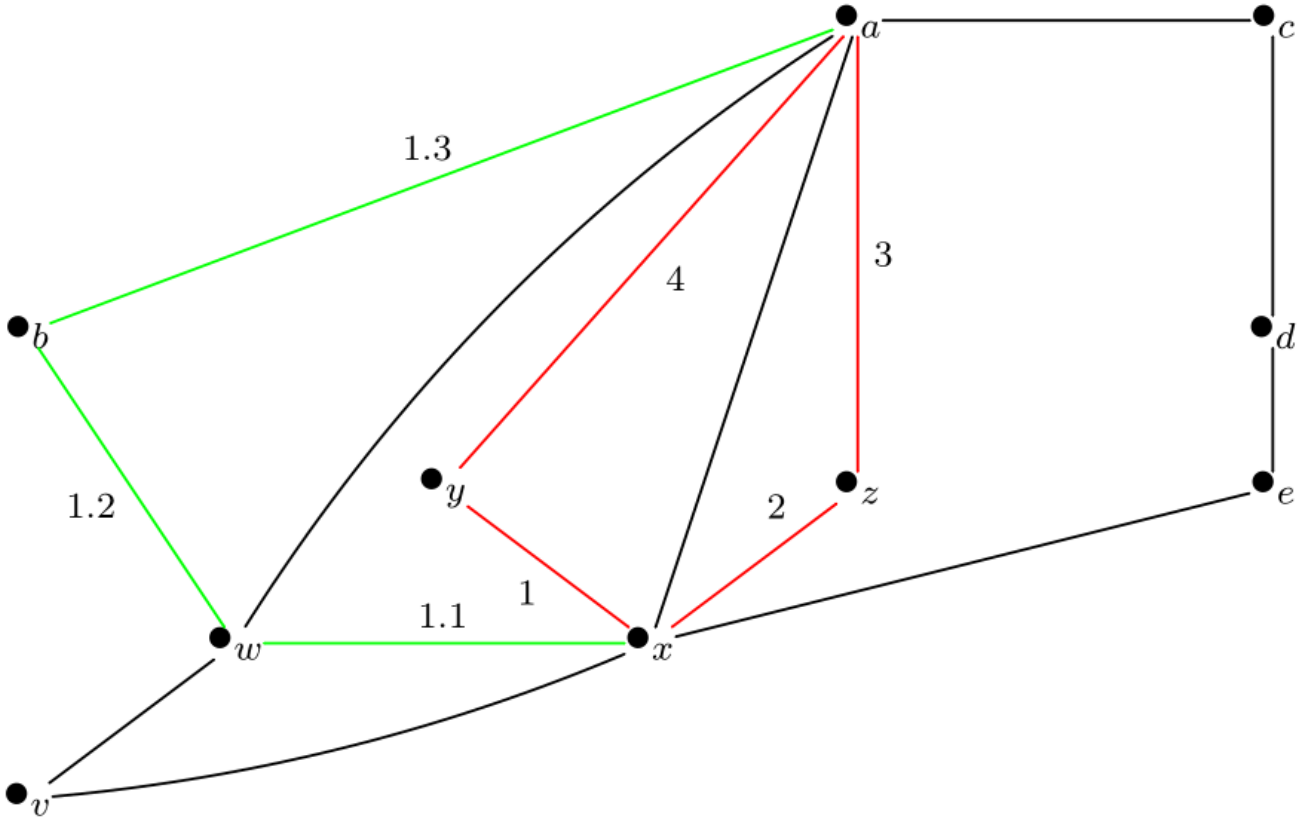
(v) Let $C = C_3$ and go back to step 3.


a
c
b
d
y
z
e
w
x
v

# First just find a cycle, starting at vertex $y$



# Now find cycle starting from $x$ so we take it and wander

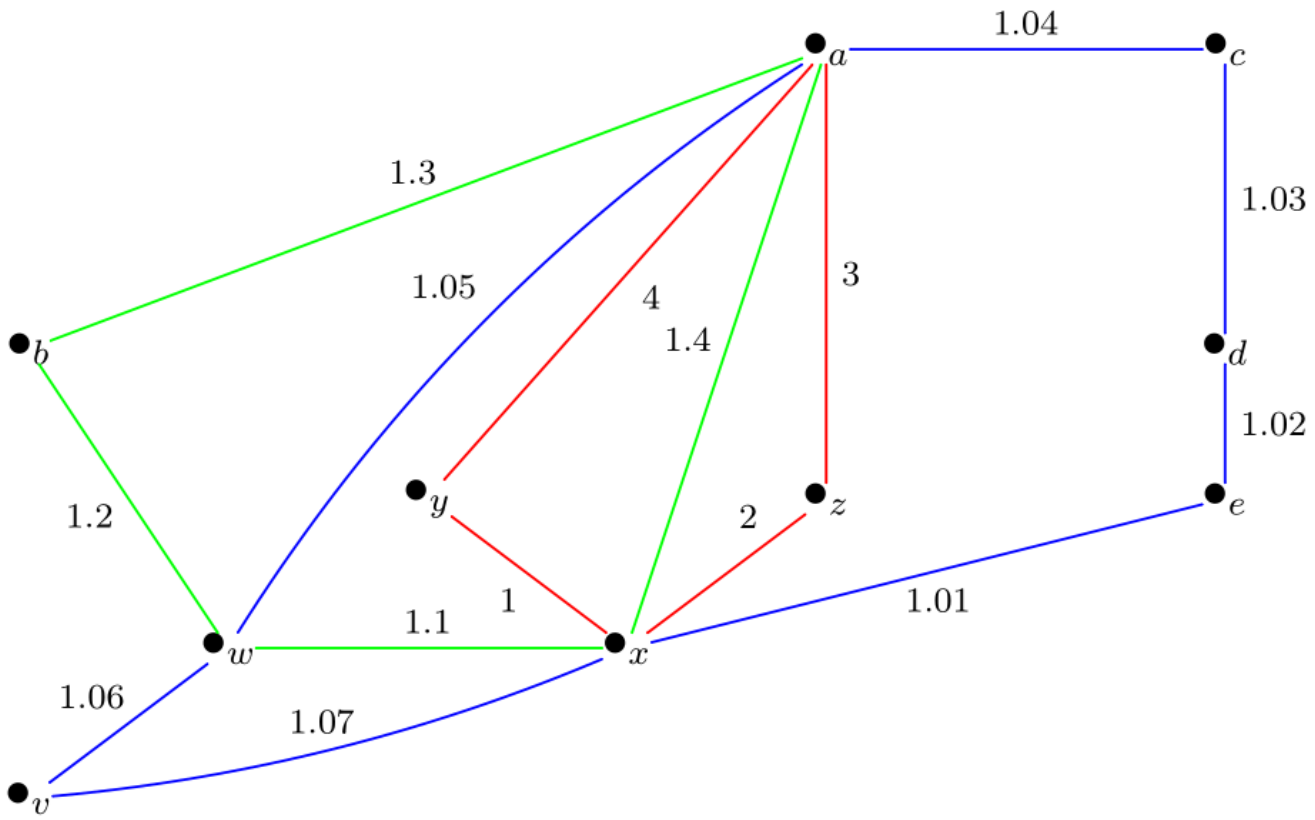



Our circuit is now described by the list of edges we have labeled

$$(1, 1.1, 1.2, 1.3, 1.4, 2, 3, 4)$$

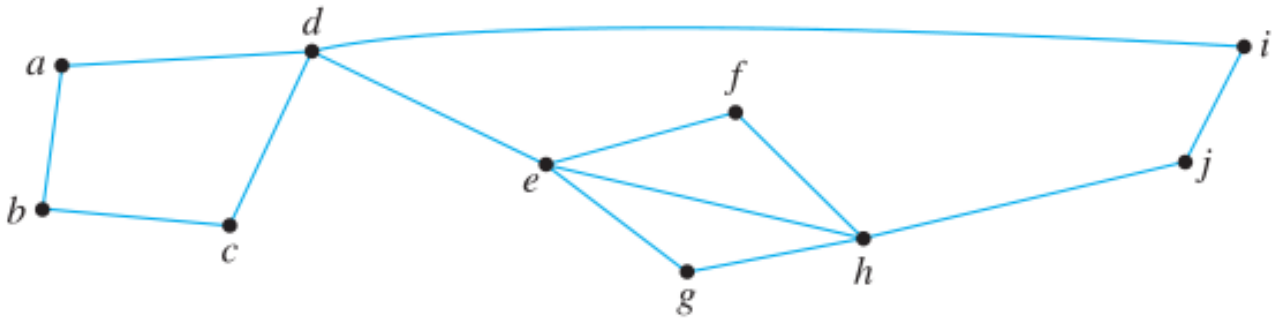
or by the list of vertices:

$$(y, x, w, b, a, x, z, a, y).$$

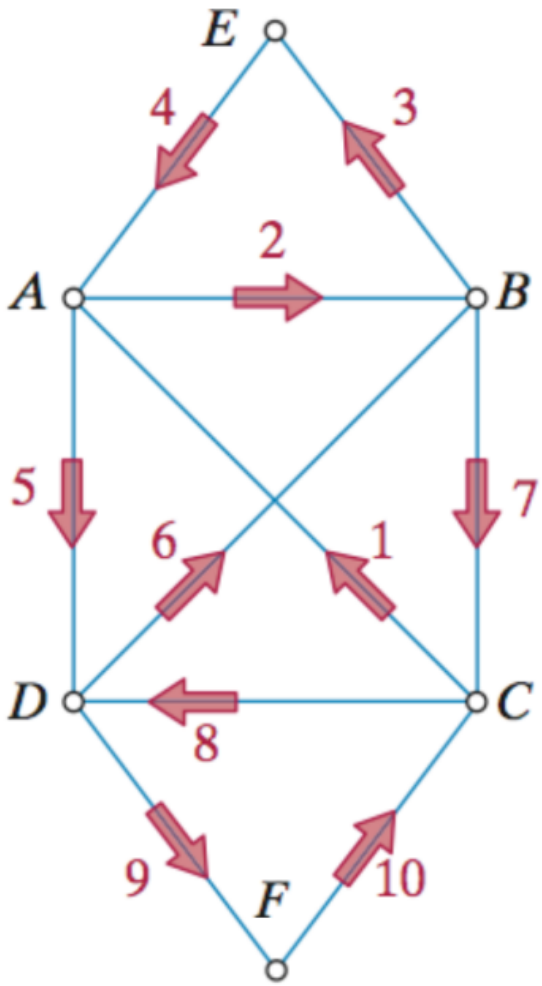



$$(y, x, e, d, c, a, w, v, x, w, b, a, x, z, a, y)$$

# Finding an Euler Circuit

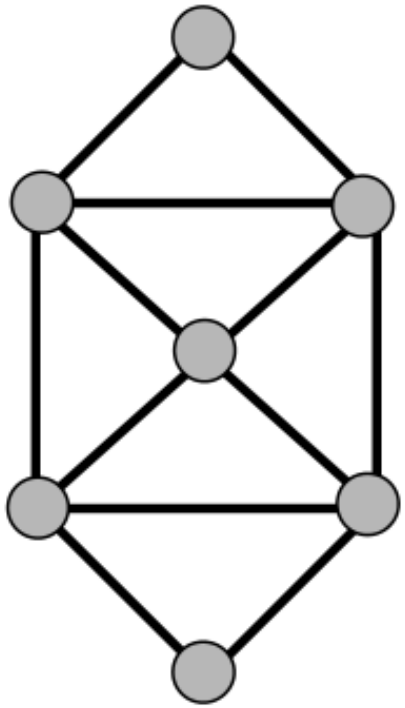


# Write $C_i$



# Finding Euler Circuits: DFS and then Splice

- Given a graph G = ($V$,$E$), find an Euler circuit in G
  - Can check if one exists in O($|V|$) time (check degrees)
- Basic Euler Circuit Algorithm:
  1. Do a depth-first search (DFS) from a vertex until you are back at this vertex
  2. Pick a vertex on this path with an unused edge and repeat 1.
  3. Splice all these paths into an Euler circuit
- Running time = O($|V| + |E|$)

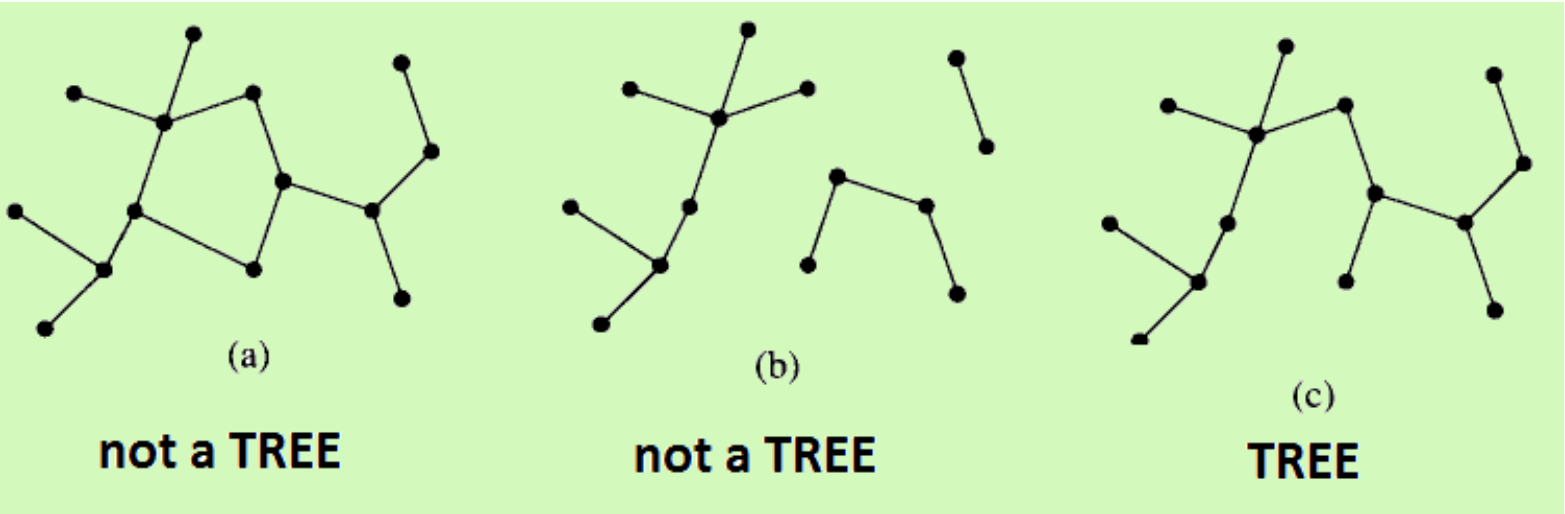


## Tree and Forest

A tree is a connected graph without any circuits.

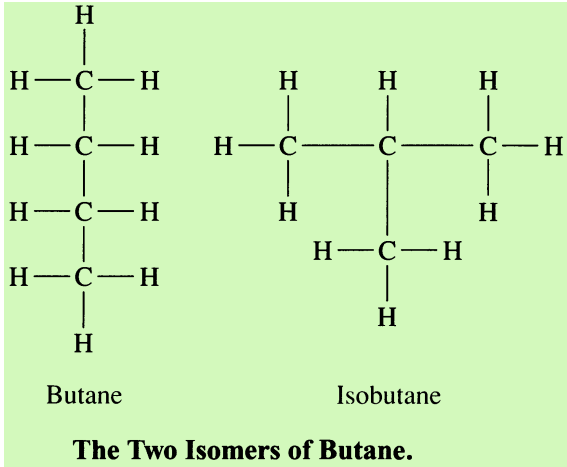
# Tree

1. Connected
2. Acyclic



## Saturated Hydrocarbons:

1. Order $= 3n + 2$
2. Size $= \frac{\text{sum of the degree of the vertices}}{2}$

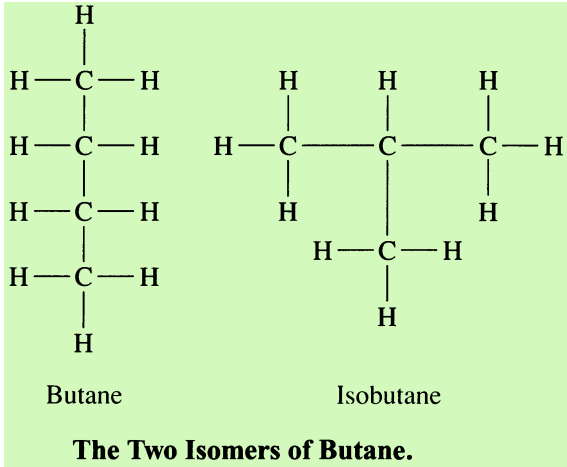


**The Two Isomers of Butane.**

**Degree of Carbon is 4 and Hydrogen is 1**

## Saturated Hydrocarbons:

1. Order $= 3n + 2$
2. Size $= \frac{\text{sum of the degree of the vertices}}{2}$



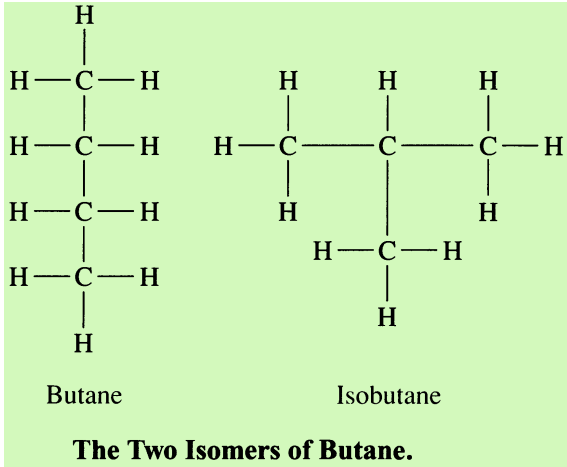
The Two Isomers of Butane.

**Degree of Carbon is 4 and Hydrogen is 1**

Total number of edges $= \frac{1}{2}\{4n + 1(2n + 2)\} = 3n + 1$

## Saturated Hydrocarbons:

1. Order $= 3n + 2$
2. Size $= \frac{\text{sum of the degree of the vertices}}{2}$



**The Two Isomers of Butane.**

**Degree of Carbon is 4 and Hydrogen is 1**

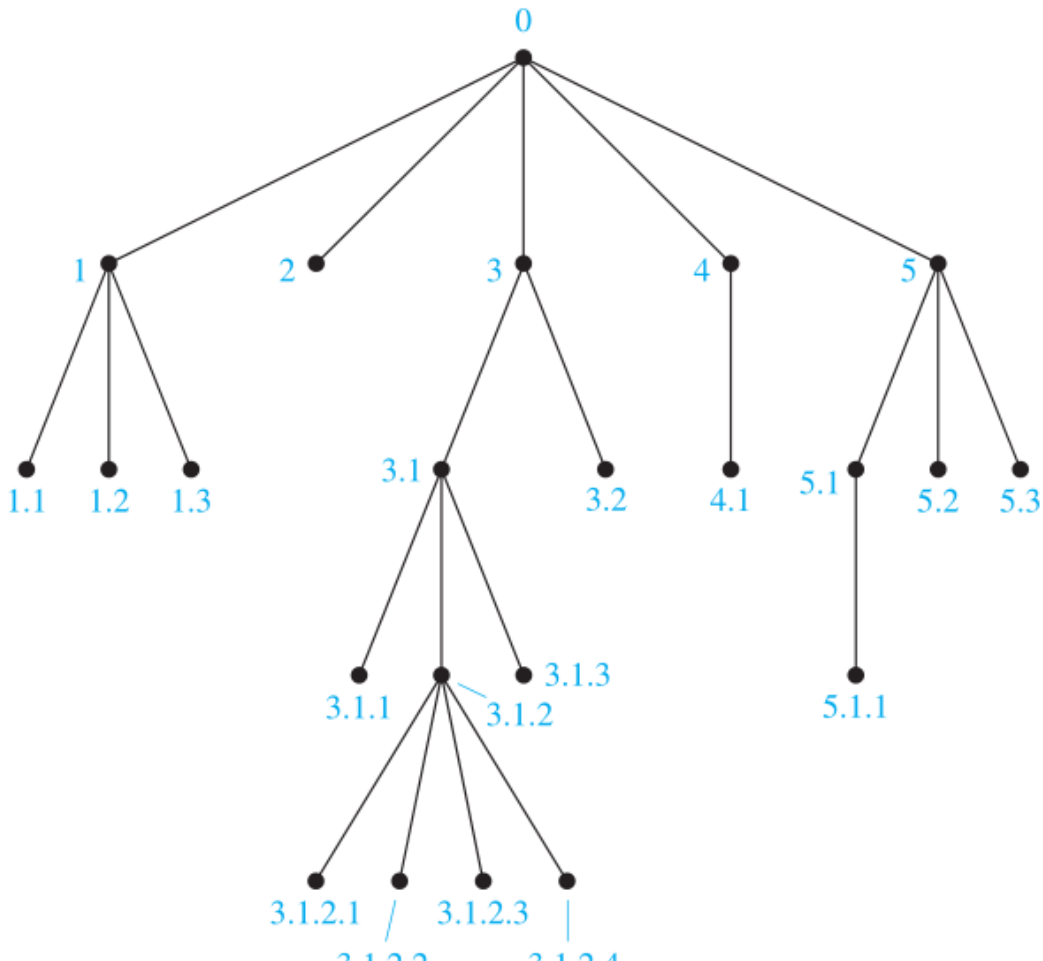
Total number of edges $= \frac{1}{2}\{4n + 1(2n + 2)\} = 3n + 1$

1. A tree $T$ with $n$ vertices has exactly $(n - 1)$ edges.
2. Every non-trivial tree has at-least 2 vertices of degree 1.

## Rooted Tree

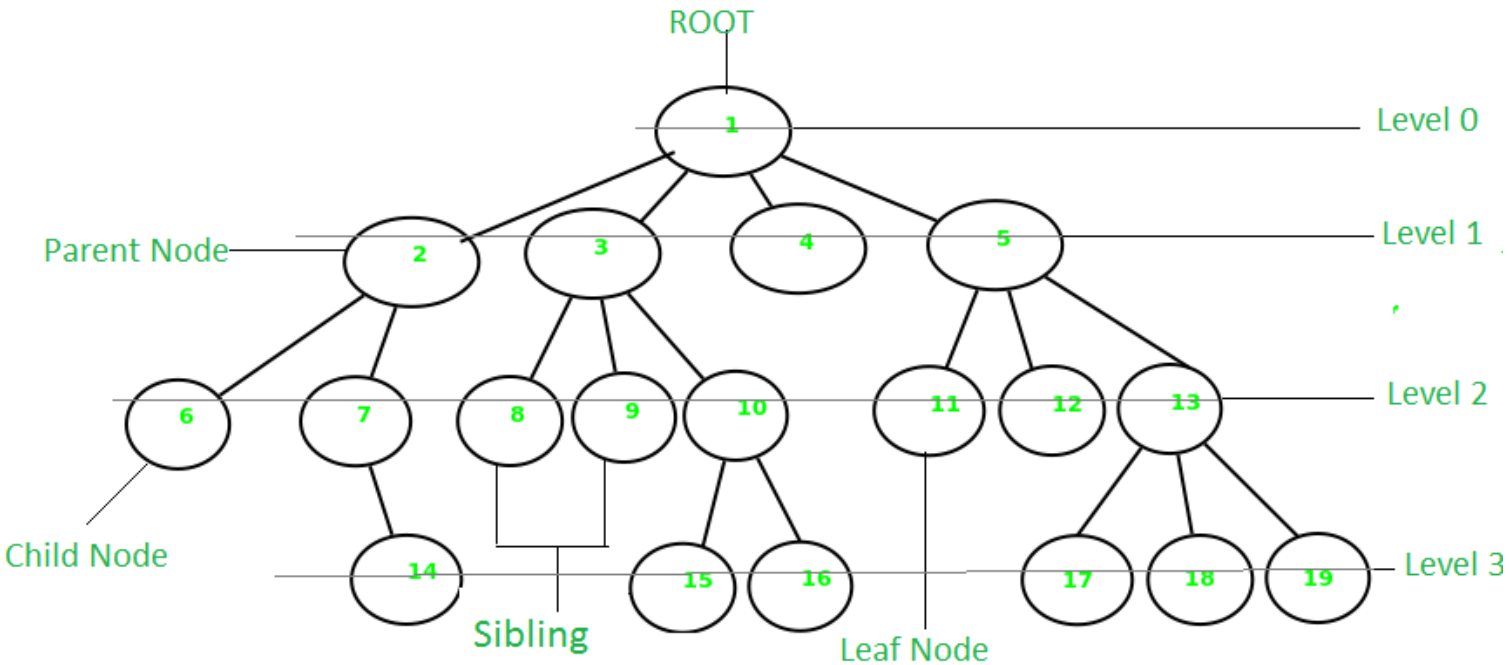
A rooted tree is a tree with a designated vertex called the root of the tree.

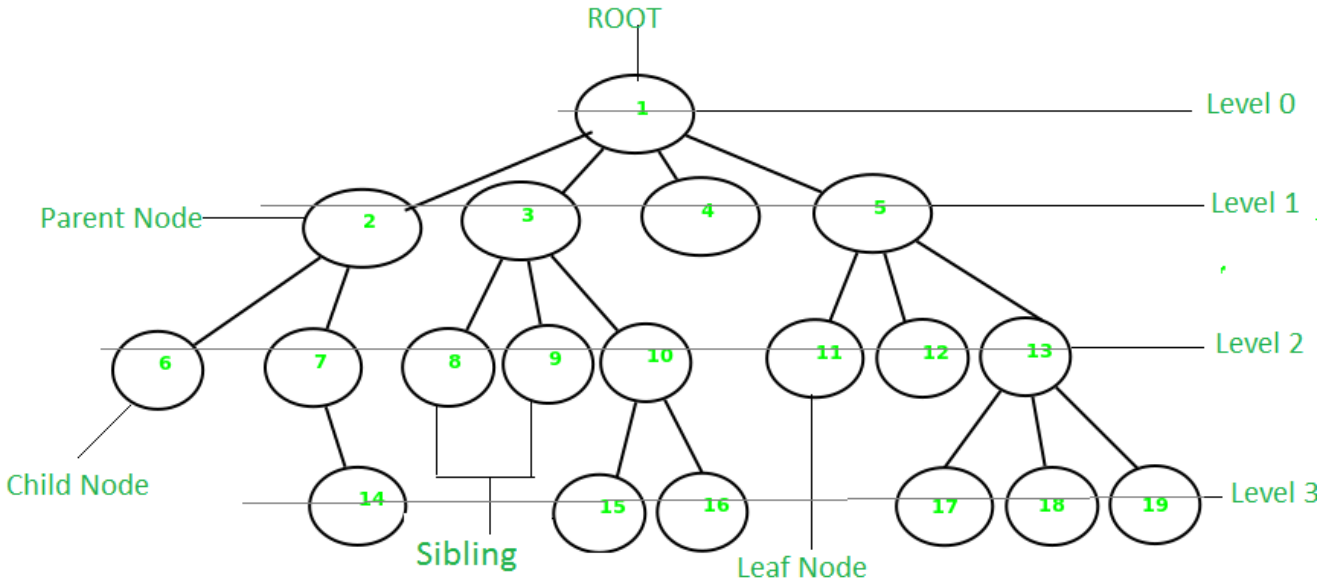
- The **level** of a vertex in a rooted tree is the length of the path (number of edges) to $v$ from the root $v_0$.



If $T = (V, E)$ is a rooted tree with designated root $v_0 \in V$ and $(v_0, v_1, v_2, \ldots, v_{n-1}, v_n)$ is a simple path in $T$, then:

- $v_{n-1}$ is called the **parent** of $v_n$
- $\{v_0, v_1, v_2, \ldots, v_{n-1}\}$ are called the **ancestors** of $v_n$
- $v_1$ is a **child** of $v_0$, $v_2$ is a **child** of $v_1$, ..., $v_n$ is a **child** of $v_{n-1}$

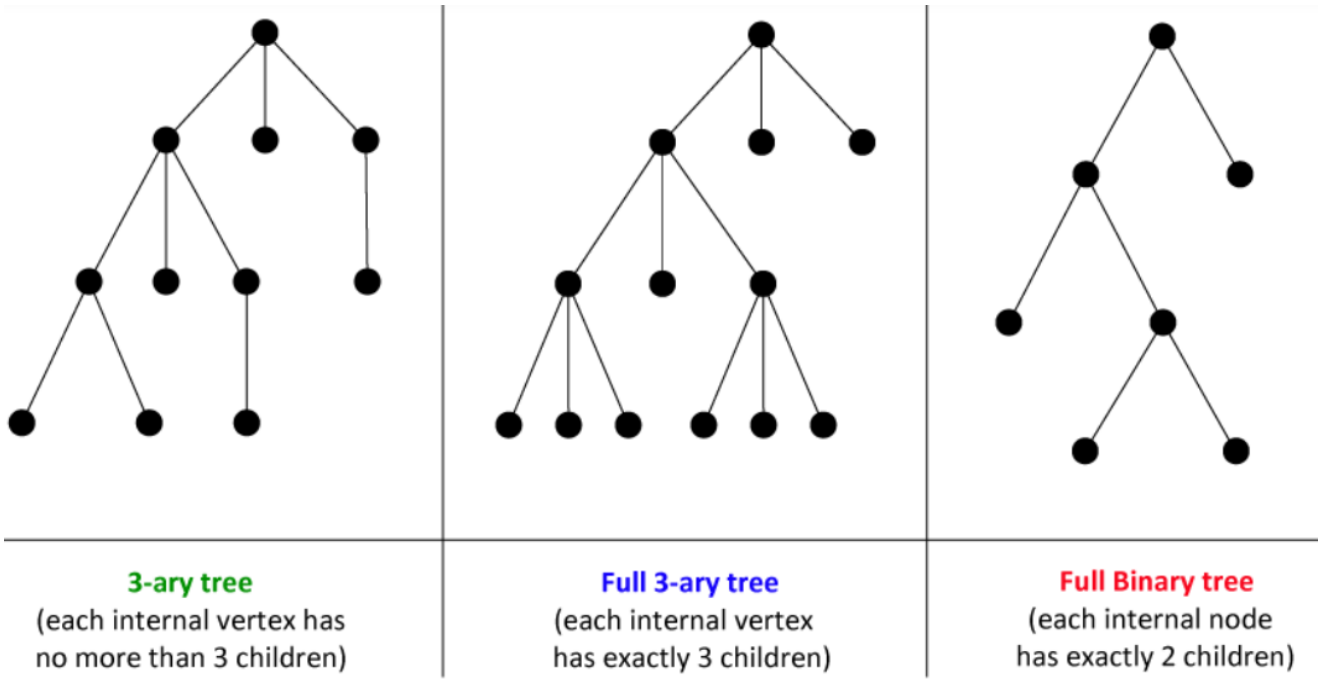





| | |
|---|---|
| Parent node | The node adjacent to a given node on the path to the root node. *Example:* Node 2 is the parent of nodes 6 and 7. |
| Child node | A node adjacent to a node (its parent) which is closer to the root node. *Example:* Nodes 6 and 7 are the children of node 2. |
| Siblings | Nodes sharing the same parent. *Nodes 6 and 7 are siblings.* |
| Internal Node | A node with a child. *Node 2 is internal, as a parent.* |
| Leaf | A childless node. *Node 6 is a leaf.* |
| Depth(Node) | The length of the path from the node to the root. *Node 6 is at depth 2.* |
| Depth of tree | The maximum node depth. *This tree has depth 3.* |

## $m-$ary Tree

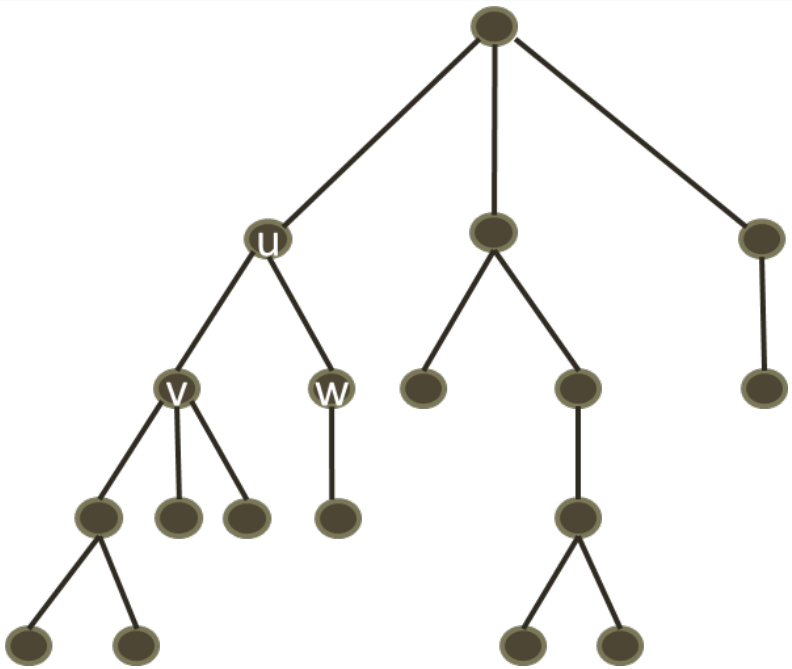
1. A rooted tree is called an $m-$ary tree if every internal node has no more than $m$ children.
2. A full $m-$ary tree occurs when every internal node has precisely $m$ children.
3. An $m$-ary tree with $m = 2$ is called a binary tree.



## Binary Tree

A **binary tree** is a rooted tree in which every parent has at most two children.

- Each child is designated as left or right.
- A full binary tree: every parent has exactly two children.
- Left/right subtrees are defined recursively.



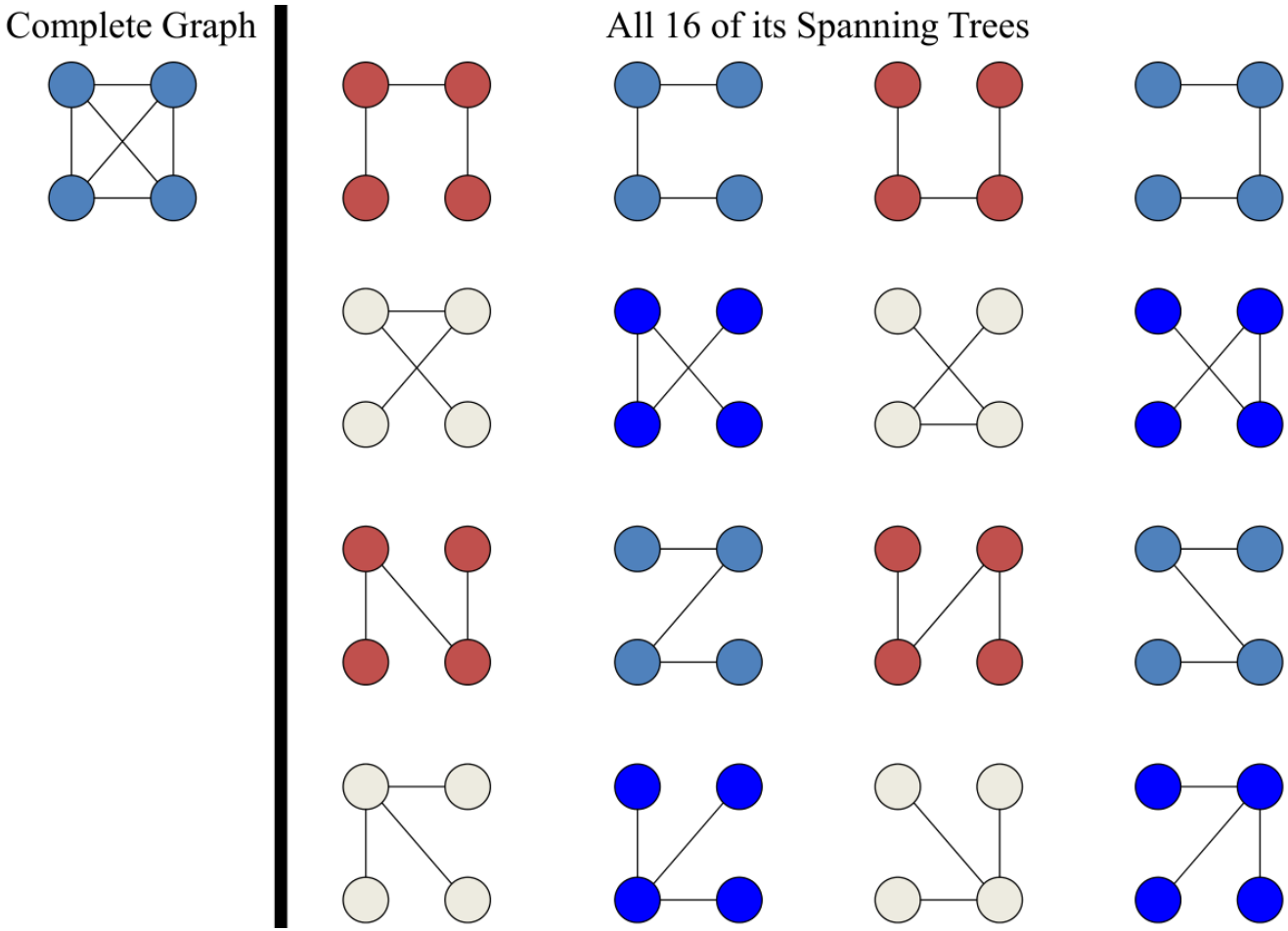
# Properties of Binary Tree

1. The number of vertices in a binary tree is odd.
2. The maximum height of a binary tree on $n$ vertices is $\frac{n-1}{2}$.
3. If $T$ is a binary tree on $n$ vertices and $p$ is the number of pendant vertices in $T$ then the number of vertices of degree 3 in $T$ is $n - p - 1$.
4. If $T$ is a binary tree on $n$ vertices then the number of pendant vertices in $T$ is $\frac{n+1}{2}$

A **spanning tree** of a graph $G$ is a subgraph of $G$ that:

- Contains all vertices of $G$.
- Is itself a tree.

Every connected graph has a spanning tree.



# Minimum Spanning Trees

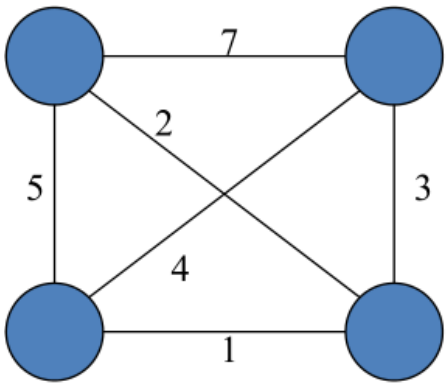
## Minimum Spanning Tree (MST)

A **minimum spanning tree** of a connected weighted graph is a spanning tree with the least possible total edge weight.

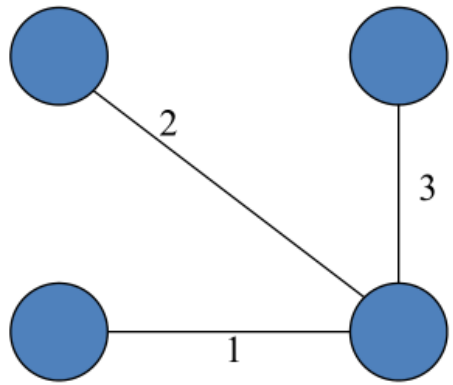
## Applications

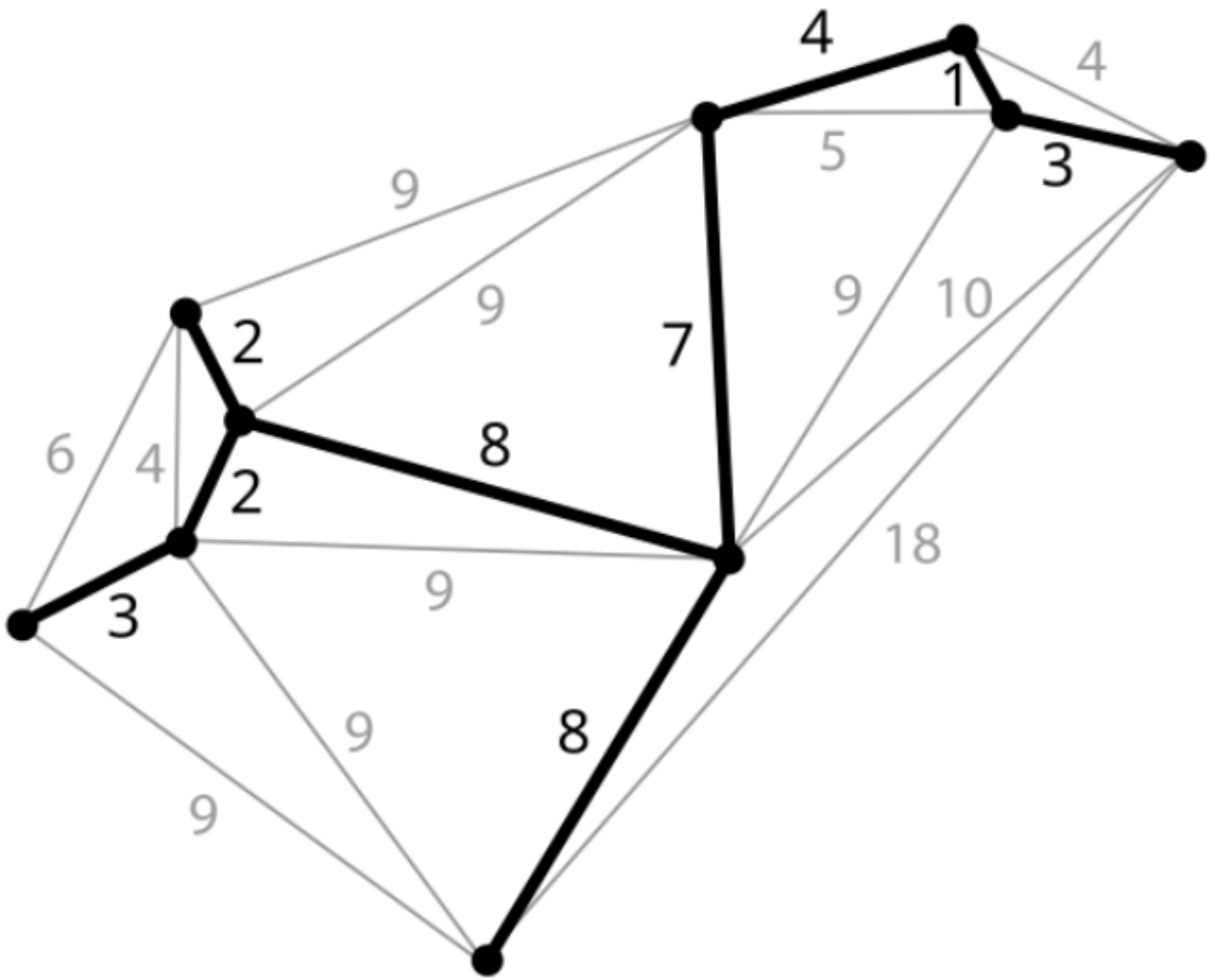
- Network design: telephone, electrical, cable, computer networks, road design.
- Example: connecting offices with minimum cost.

Complete Graph



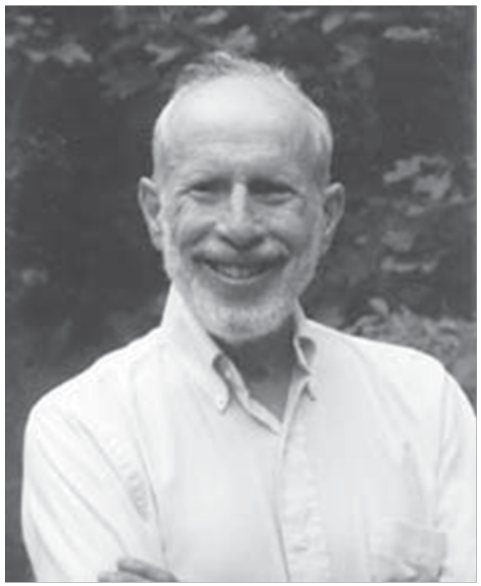

Minimum Spanning Tree




4
1
4
5
3
9
9
9
10
7
2
8
6
4
2
18
9
3
9
8
9

# Algorithms for Finding Minimum Spanning Trees

In 1956 and 1957 Joseph B. Kruskal and Robert C. Prim each described much more efficient algorithms to construct minimum spanning trees. Even for large graphs, both algorithms can be implemented so as to take relatively short computing times



*Joseph Kruskal (born 1928)*

*Robert Prim (born 1921)*

# Kruskal's Algorithm

## Steps

1. Start with each node in a separate tree.
2. Place edges in a priority queue sorted by weight.
3. Repeat until $n - 1$ edges added:
   - Extract cheapest edge.
   - If it forms a cycle, reject it.
   - Else add it to the forest (joining two trees).
4. End with a single tree containing all vertices.

# Kruskal's Minimum Spanning Tree Algorithm

**Input:** A connected weighted graph $G = (V, E)$ with $n$ vertices
**Output:** Minimum spanning tree $T$ of $G$

**Step 1:** Arrange edges in order of increasing weights

- Sort all edges: $E_{\text{sorted}} = \text{sort}(E, \text{key} = w)$
- Select edge with minimum weight: $e_{\min} = E_{\text{sorted}}[1]$
- Initialize: $T \leftarrow \emptyset$, $counter \leftarrow 0$

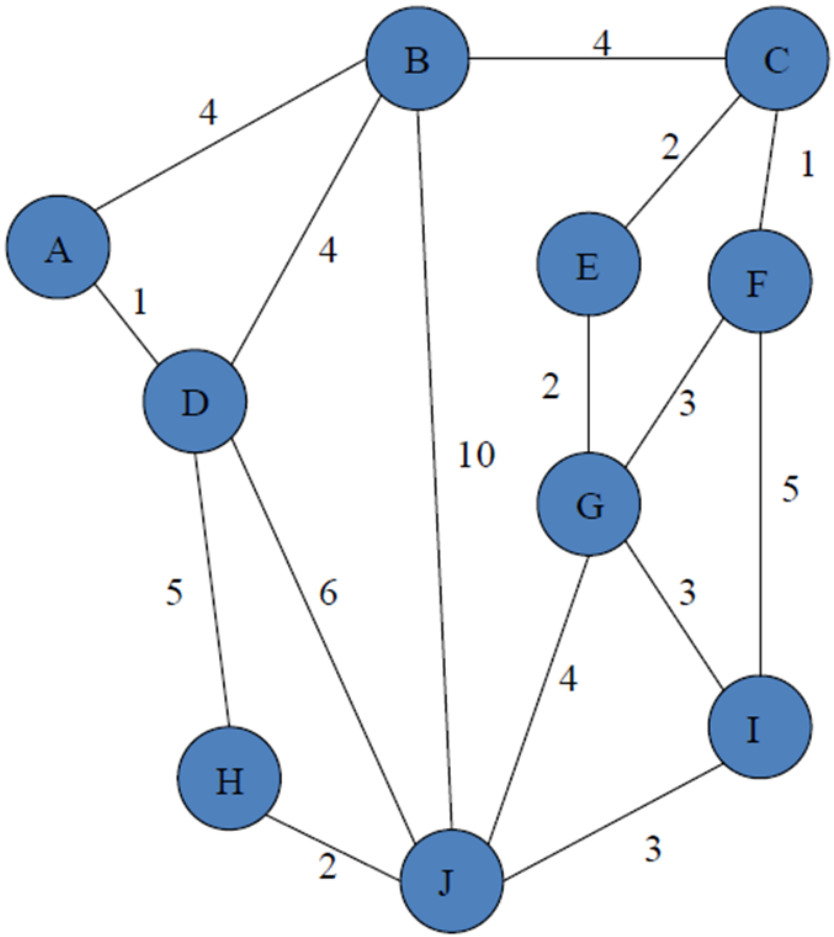
**Step 2:** Construct spanning tree iteratively

- For each edge $e$ in $E_{\text{sorted}}$ (in order):
  - If $counter = n - 1$: BREAK (MST complete)
  - If $T \cup \{e\}$ is acyclic:
    - $T \leftarrow T \cup \{e\}$
    - $counter \leftarrow counter + 1$

**Step 3:** Terminate and return result

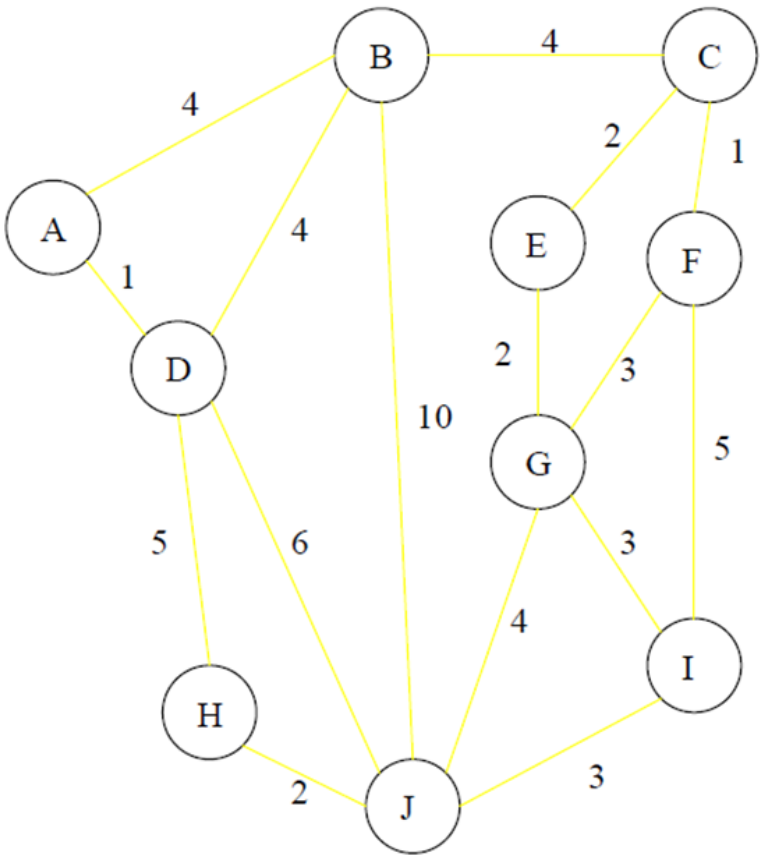
- Return the minimum spanning tree $T$
- $T$ contains exactly $n - 1$ edges
- $T$ is acyclic and connects all vertices

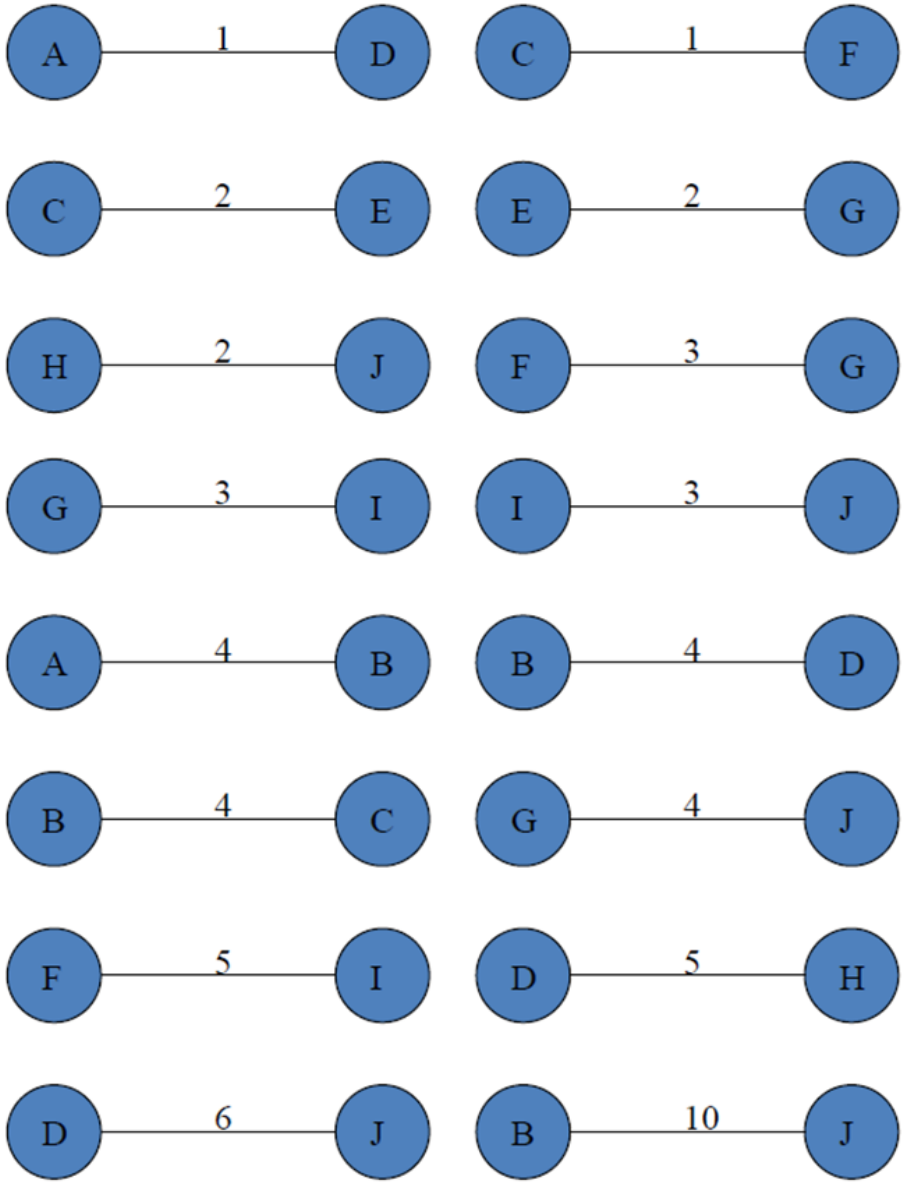
# Kruskal's Algorithm



## Sort Edges

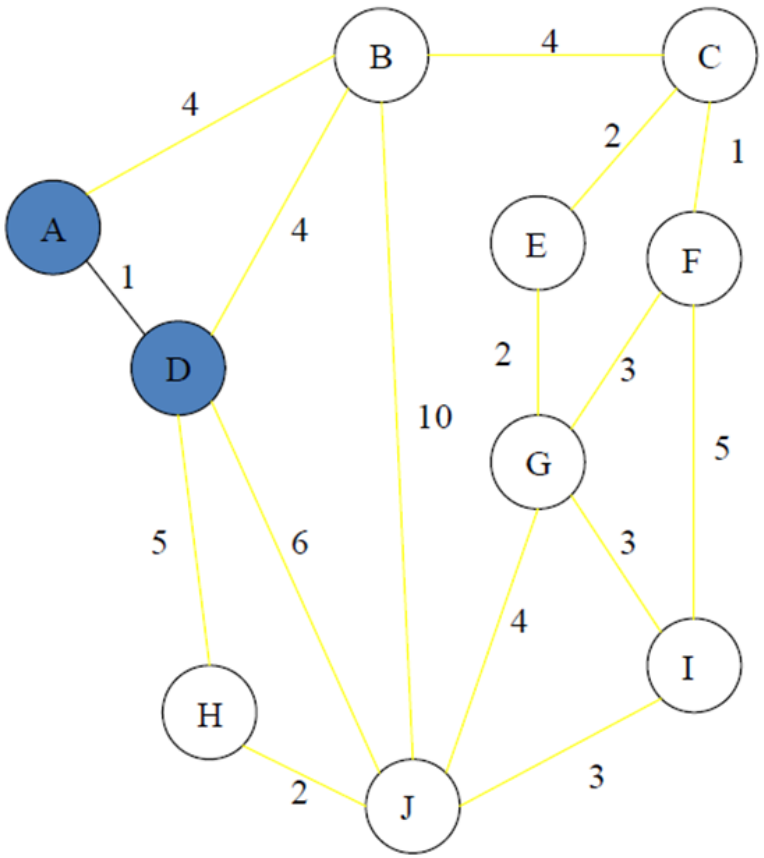
(in reality they are placed in a priority queue - not sorted - but sorting them makes the algorithm easier to visualize)

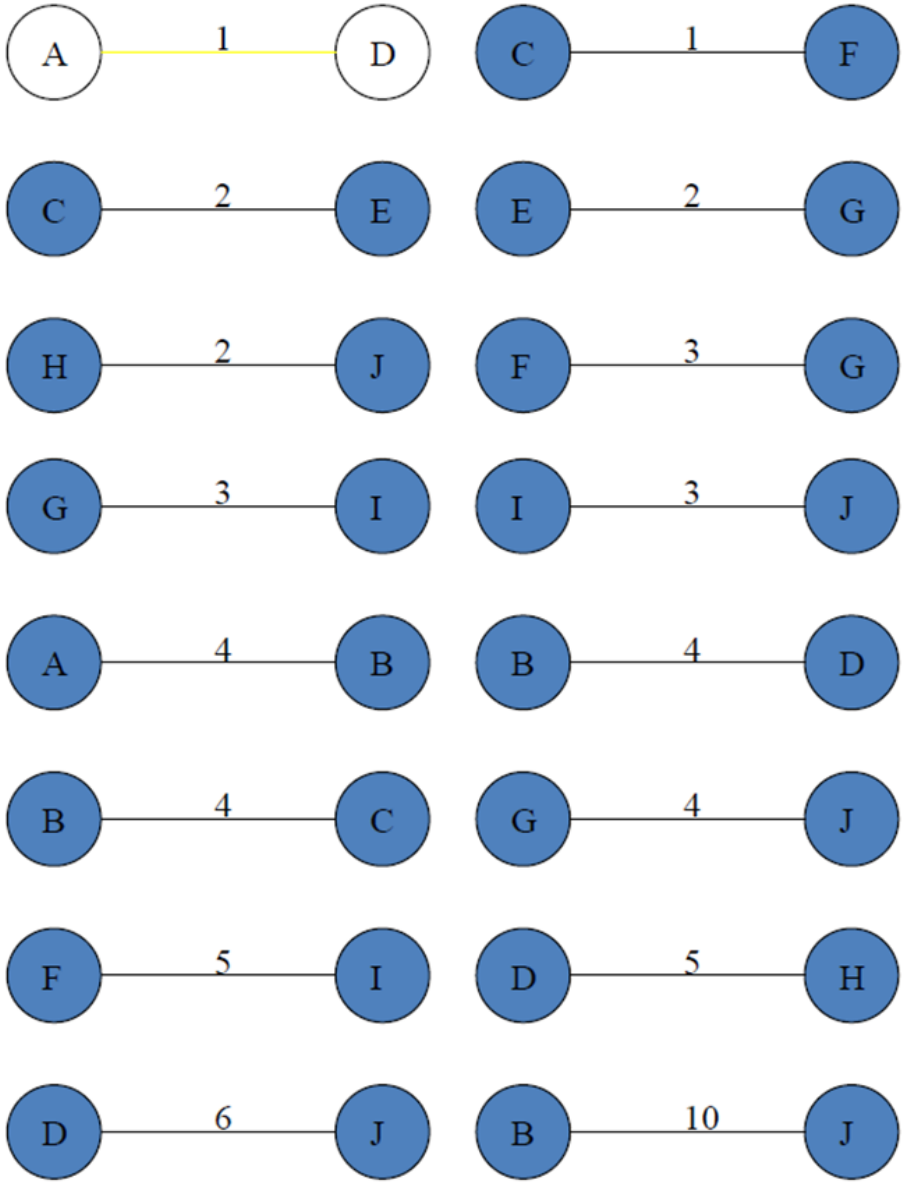


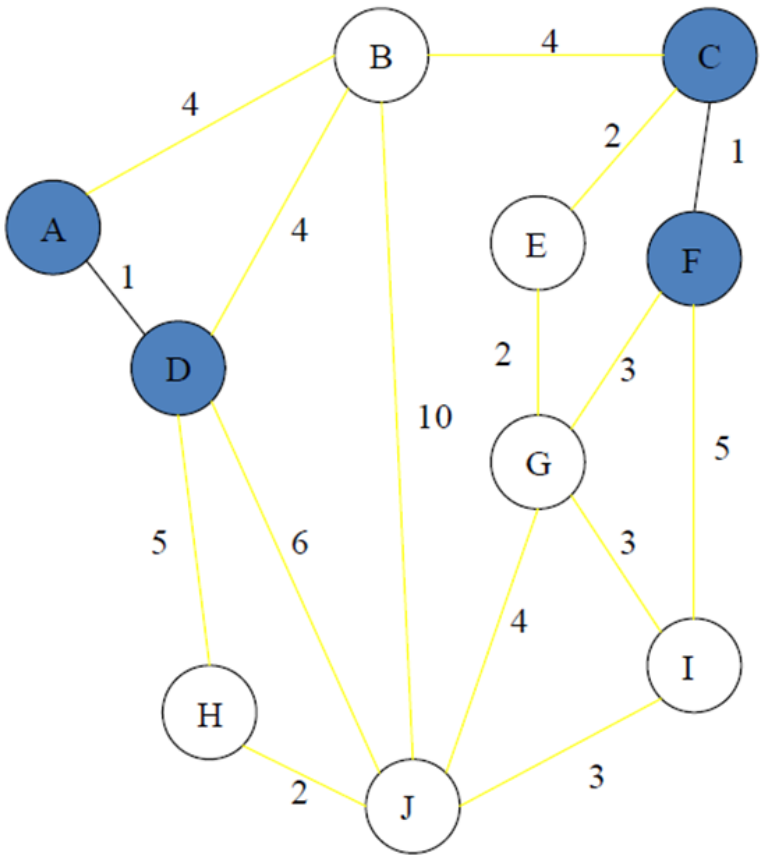
# Kruskal's Algorithm

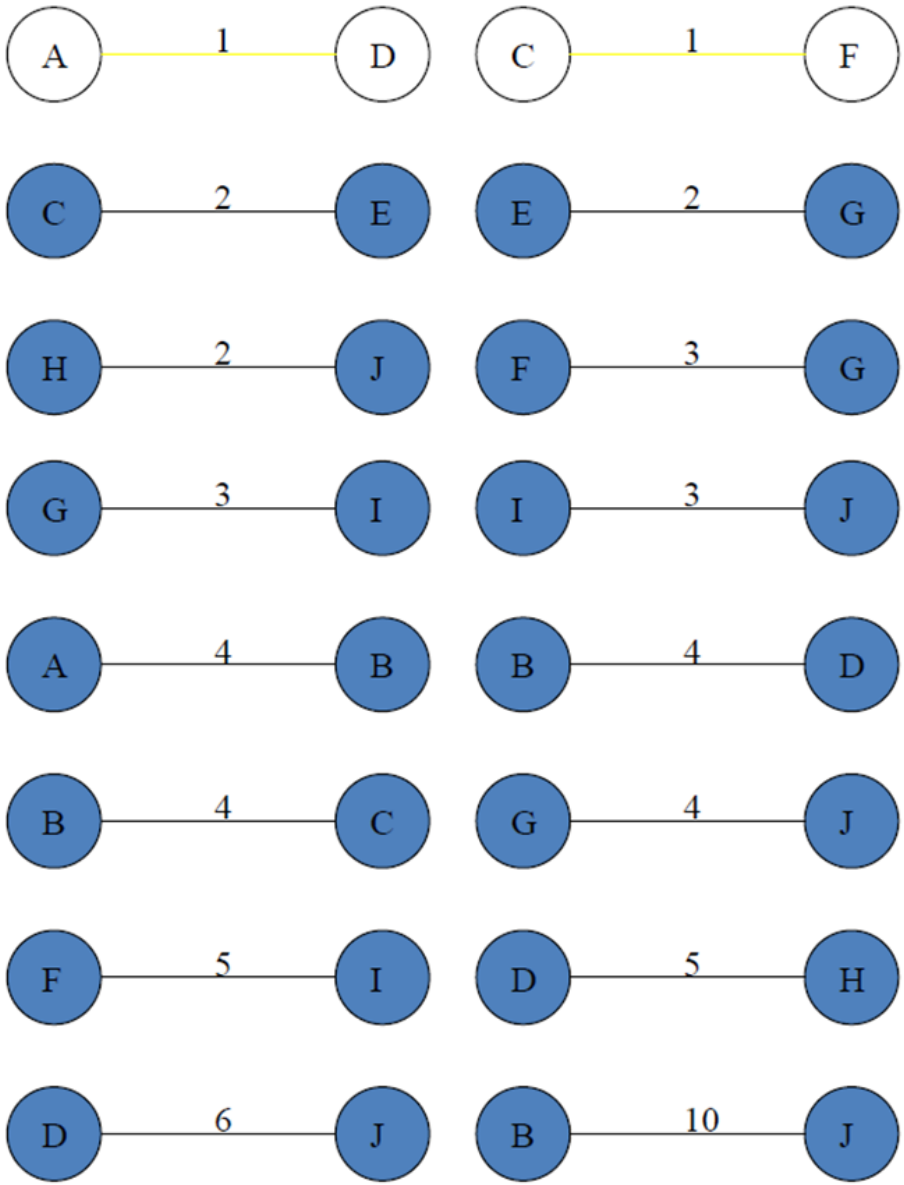
## Add Edge



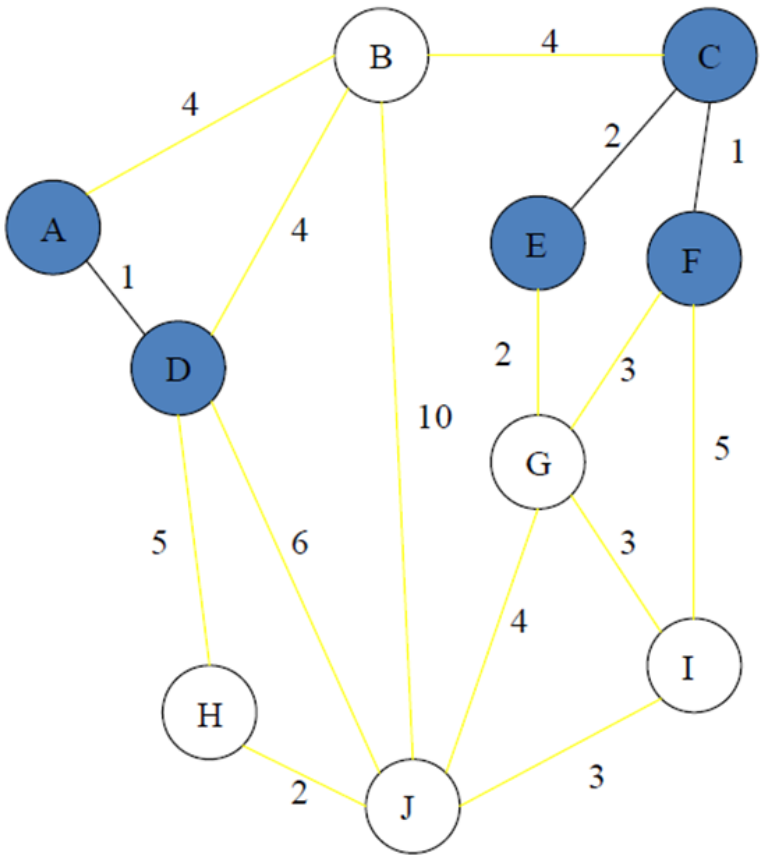
## Add Edge

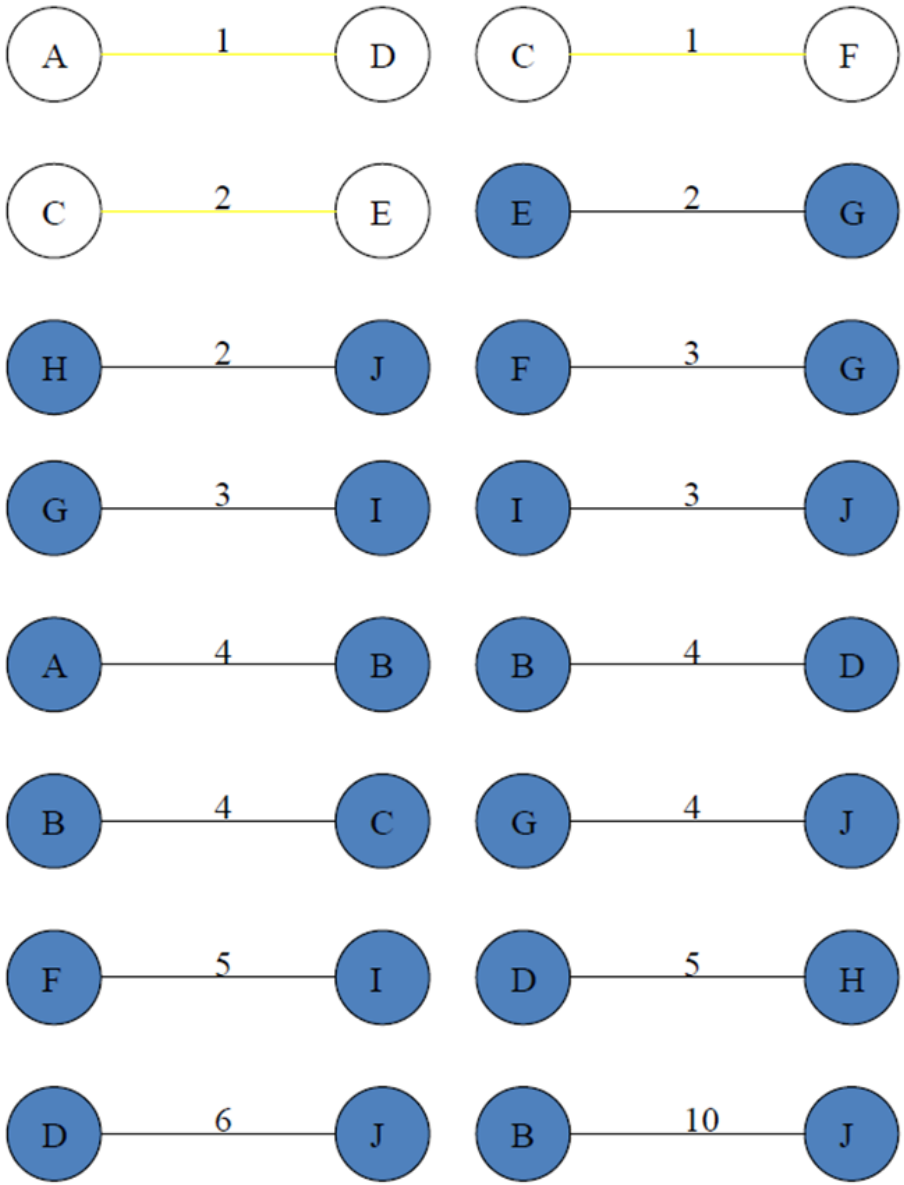

A
B
C
D
E
F
G
H
I
J
4
4
4
1
2
1
2
3
10
5
5
6
3
4
3
2
3


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

# Kruskal's Algorithm

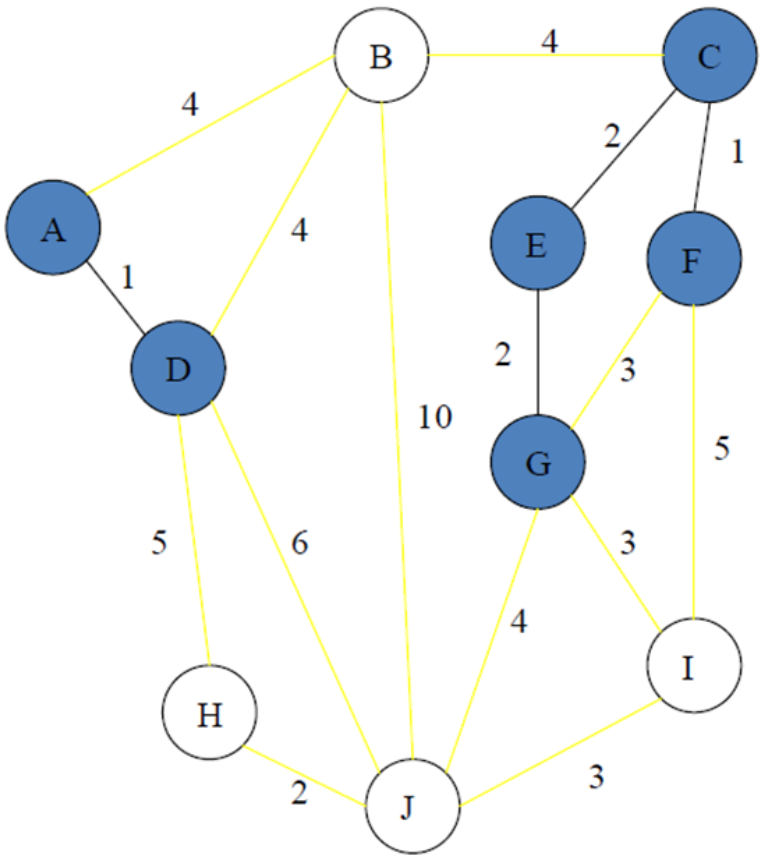
## Add Edge

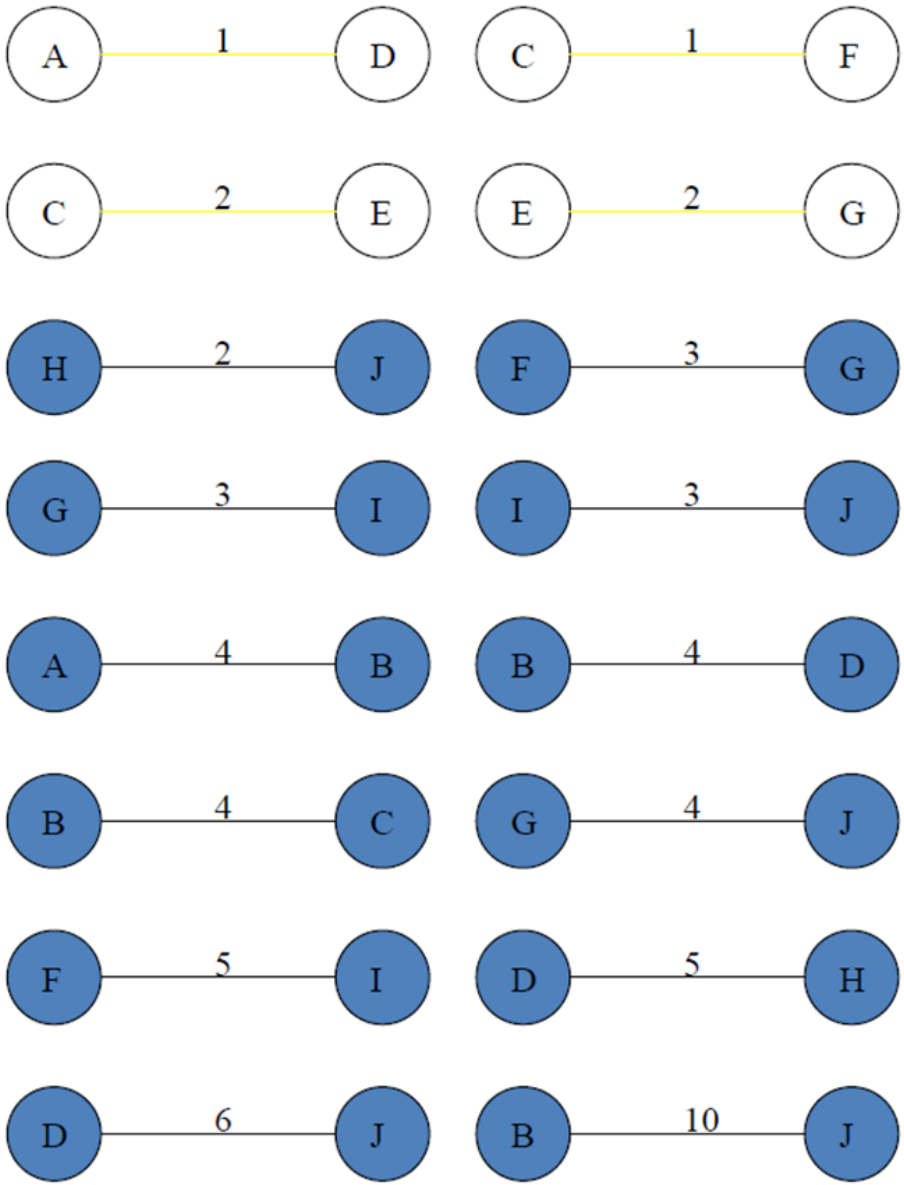


# Kruskal's Algorithm

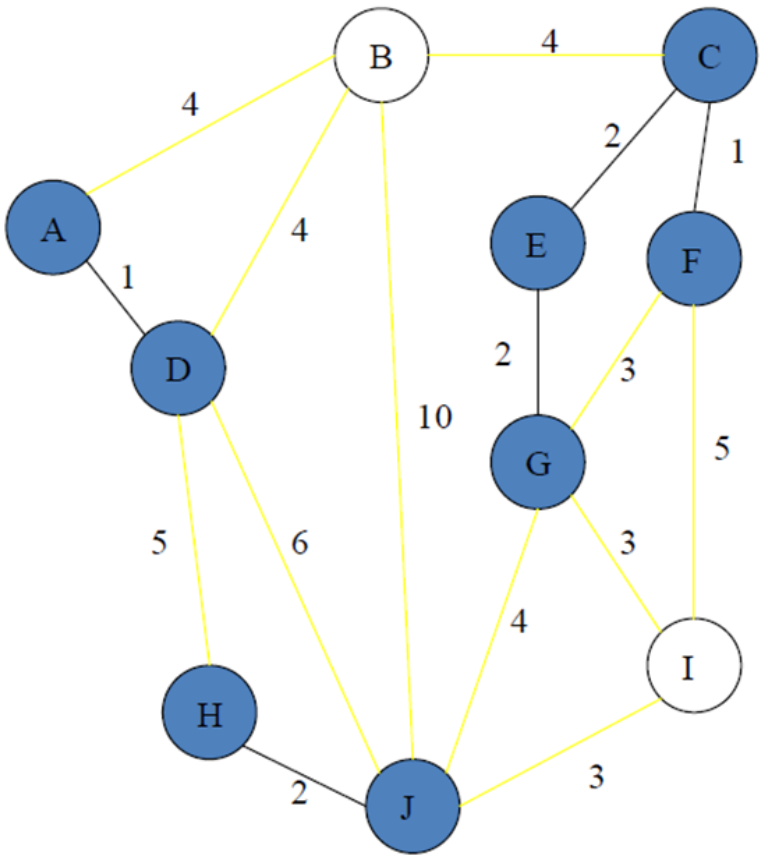
## Add Edge

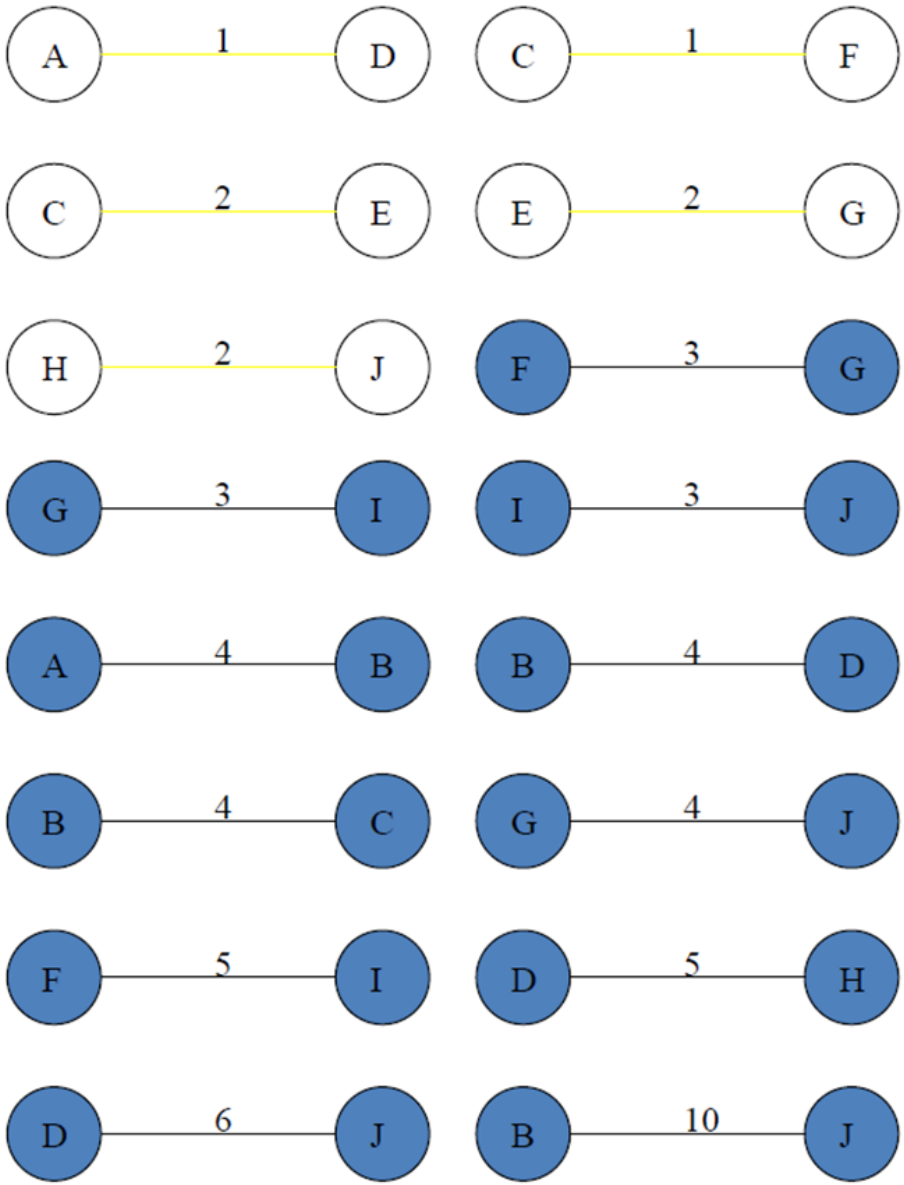


# Kruskal's Algorithm

## Add Edge

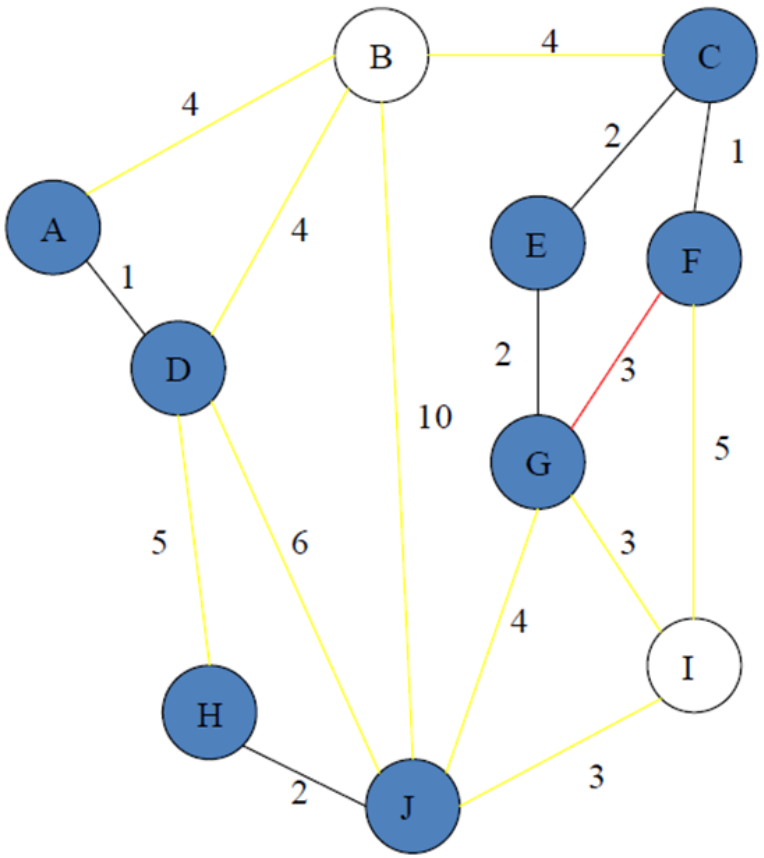


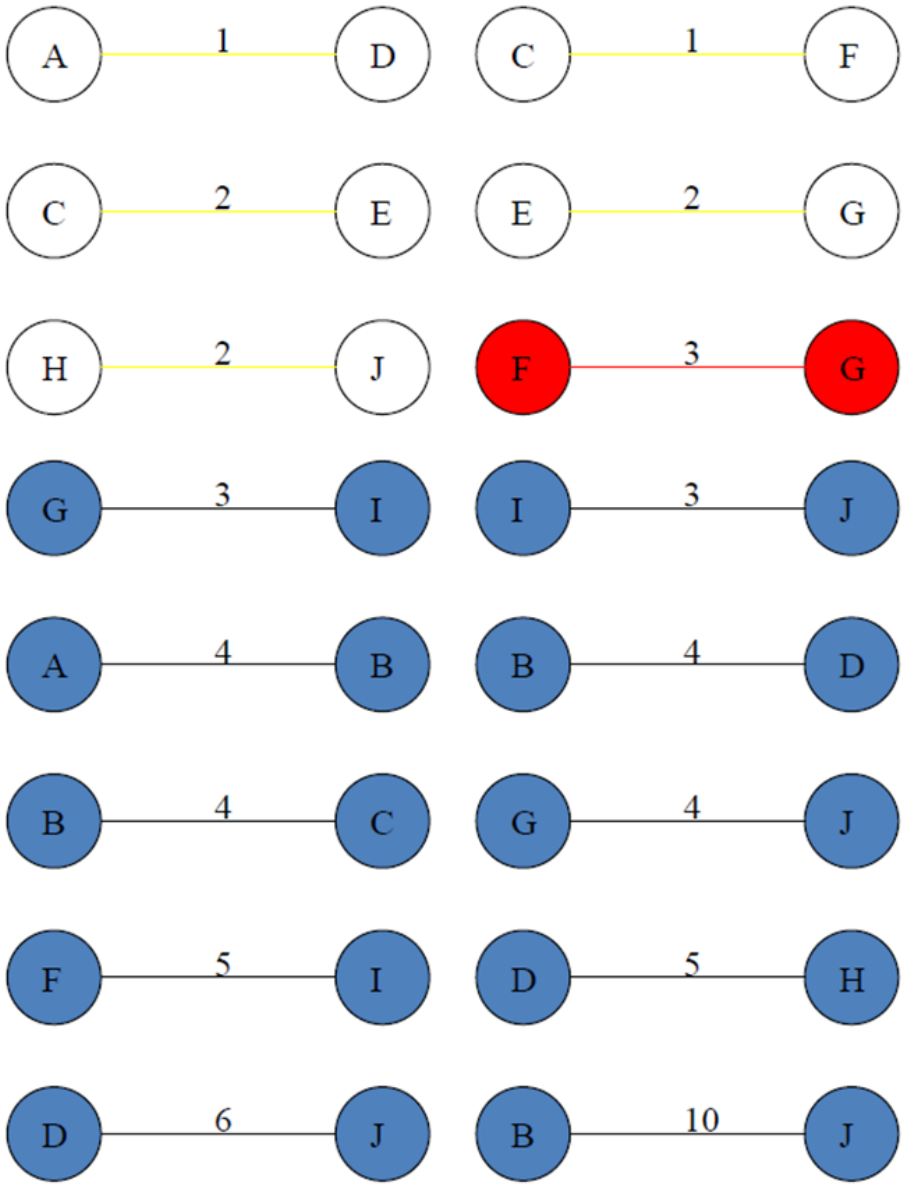



# Kruskal's Algorithm

Cycle

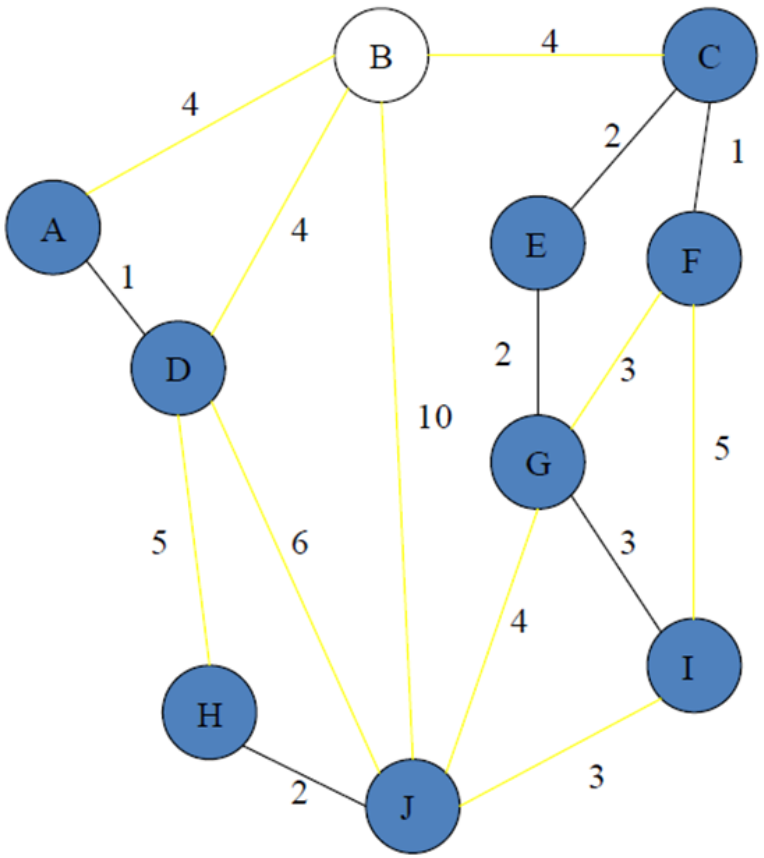
Don't Add Edge

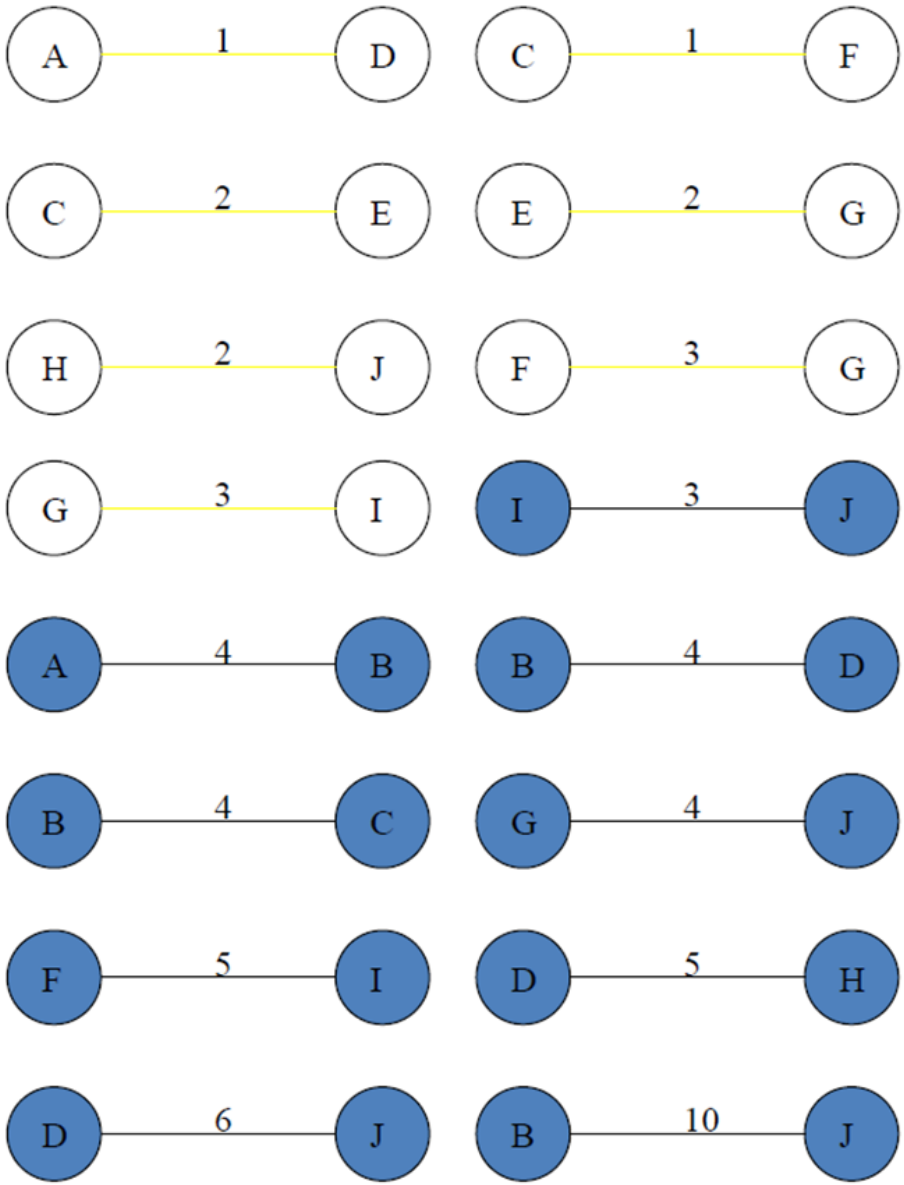


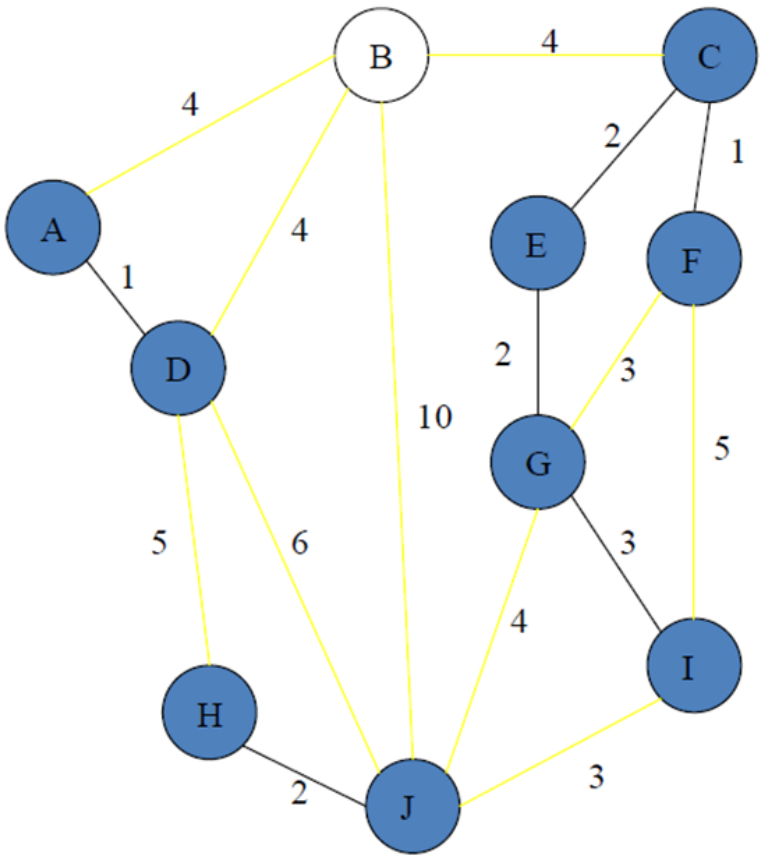
# Kruskal's Algorithm

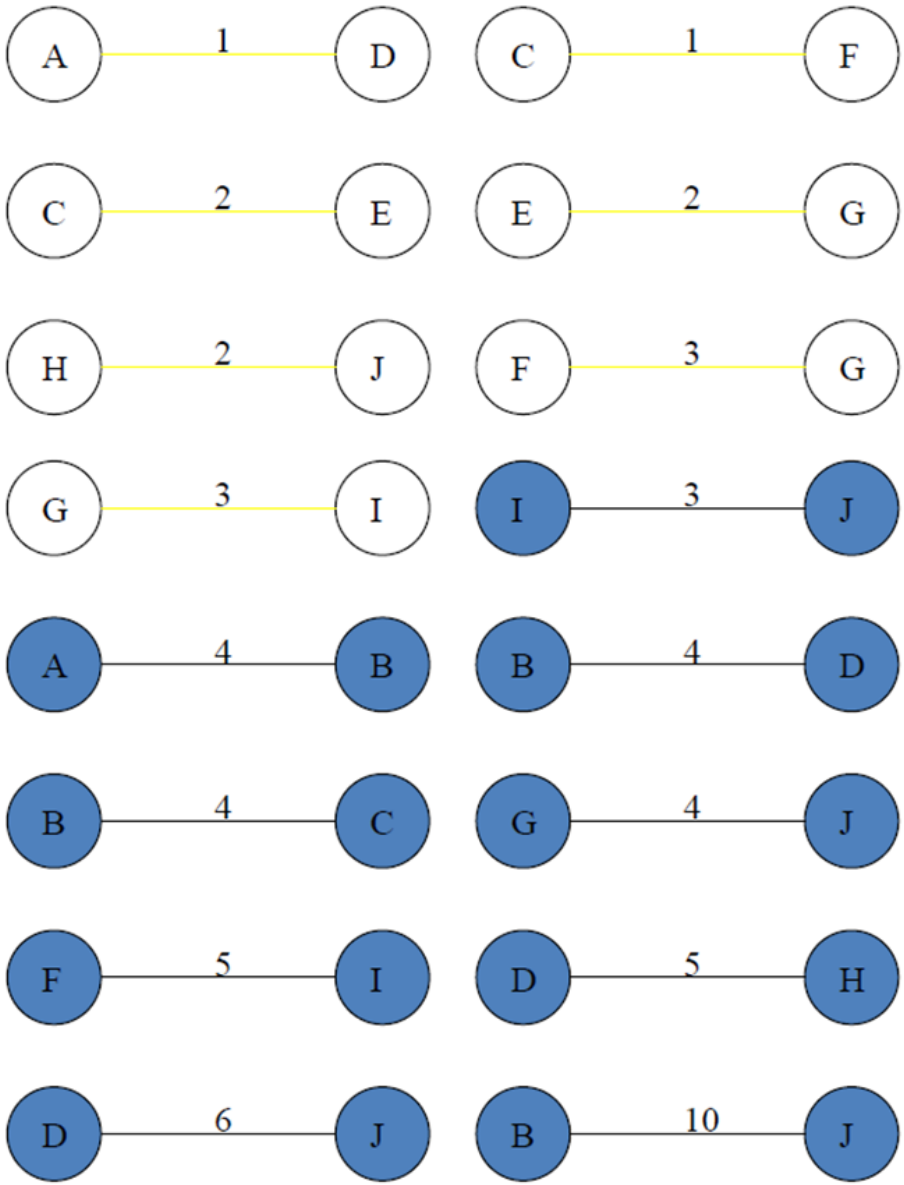
## Add Edge



Add Edge

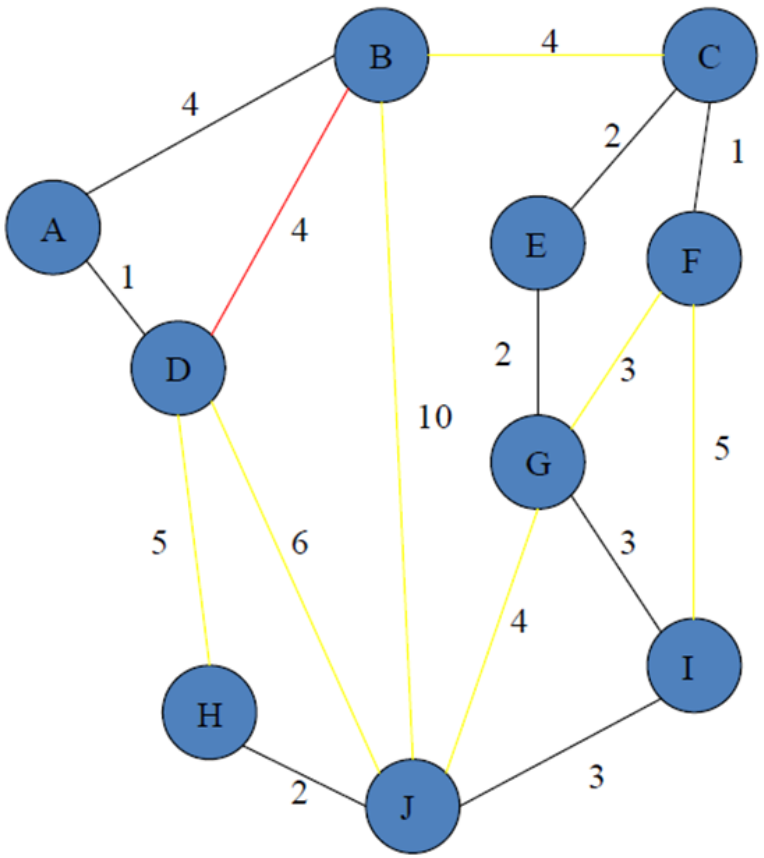

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3

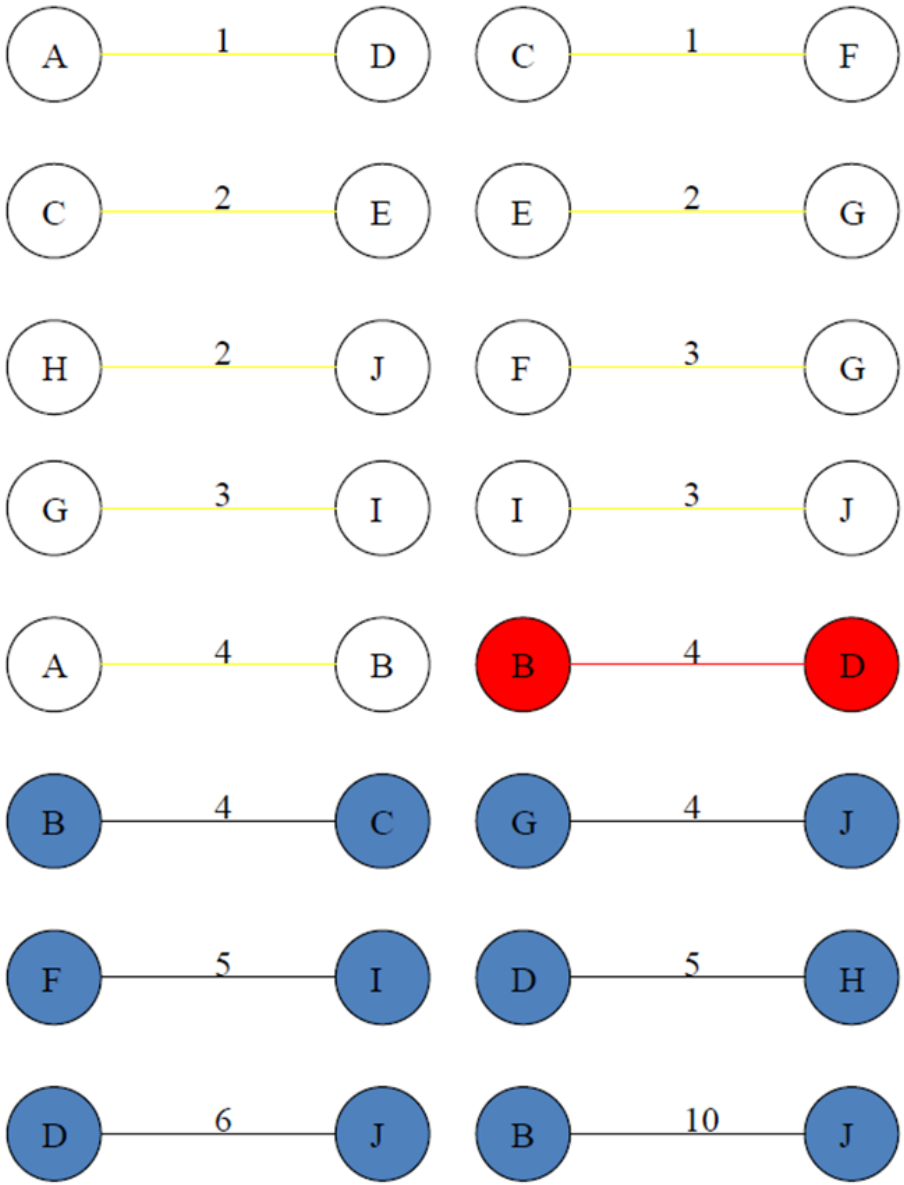

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

# Kruskal's Algorithm

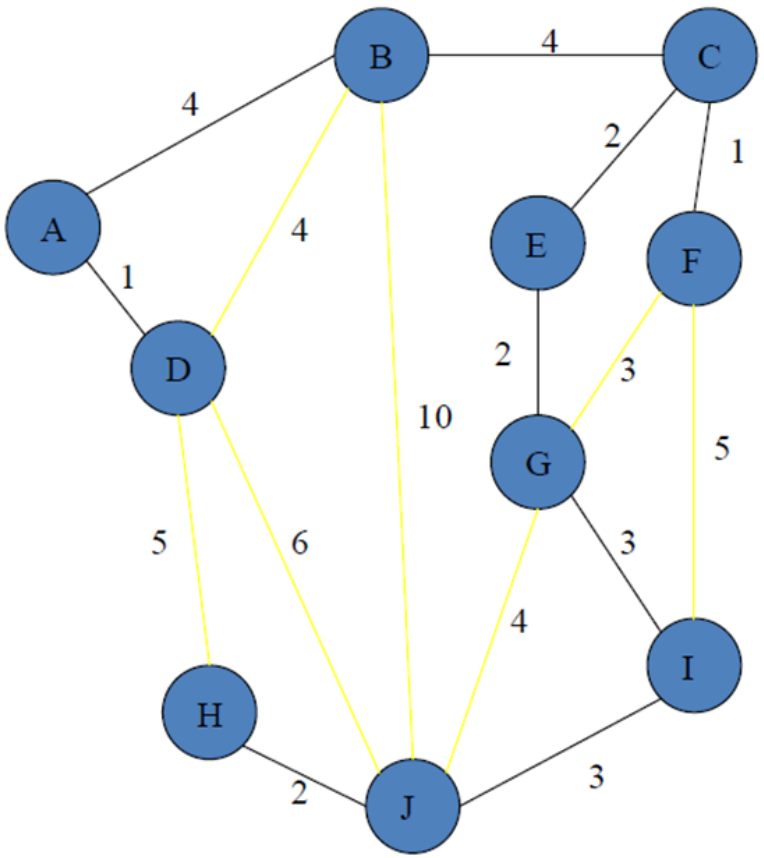
Cycle

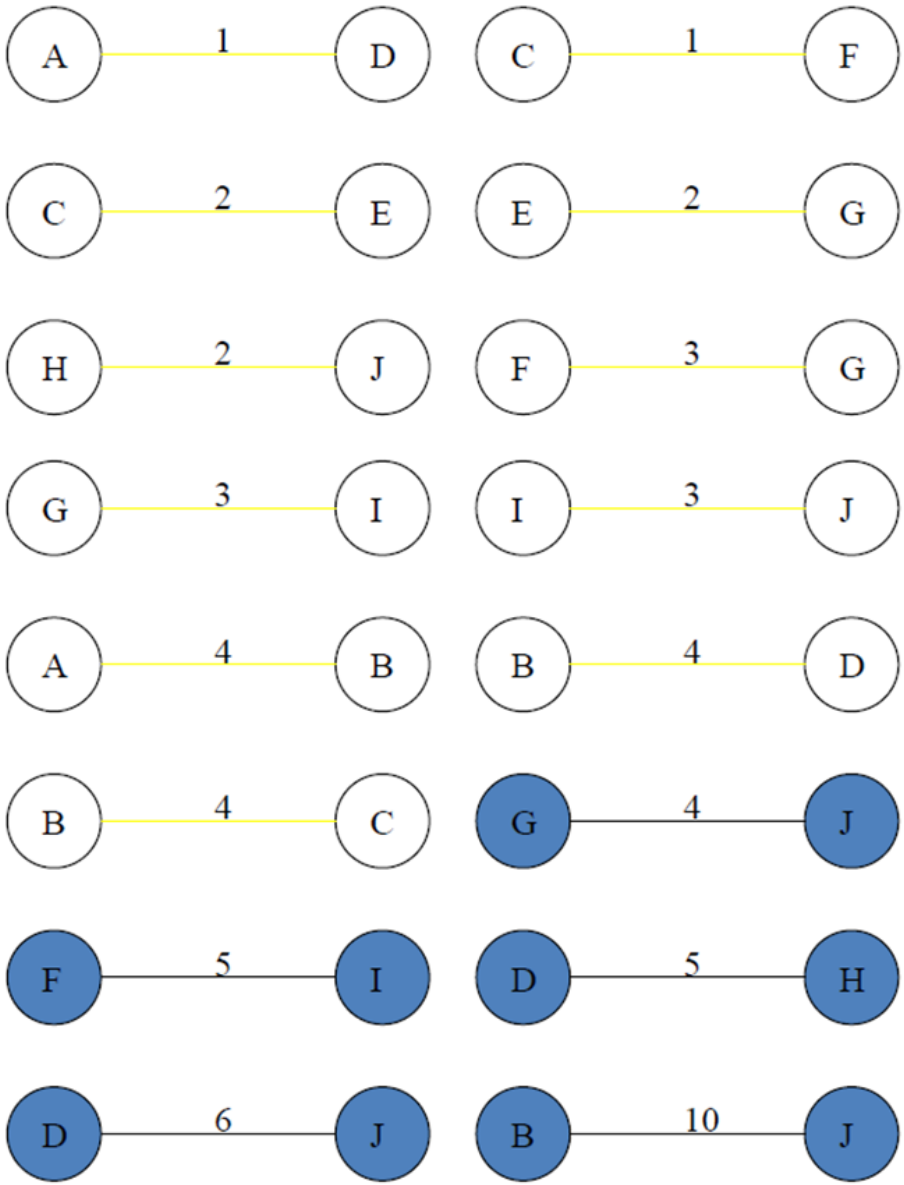
Don't Add Edge



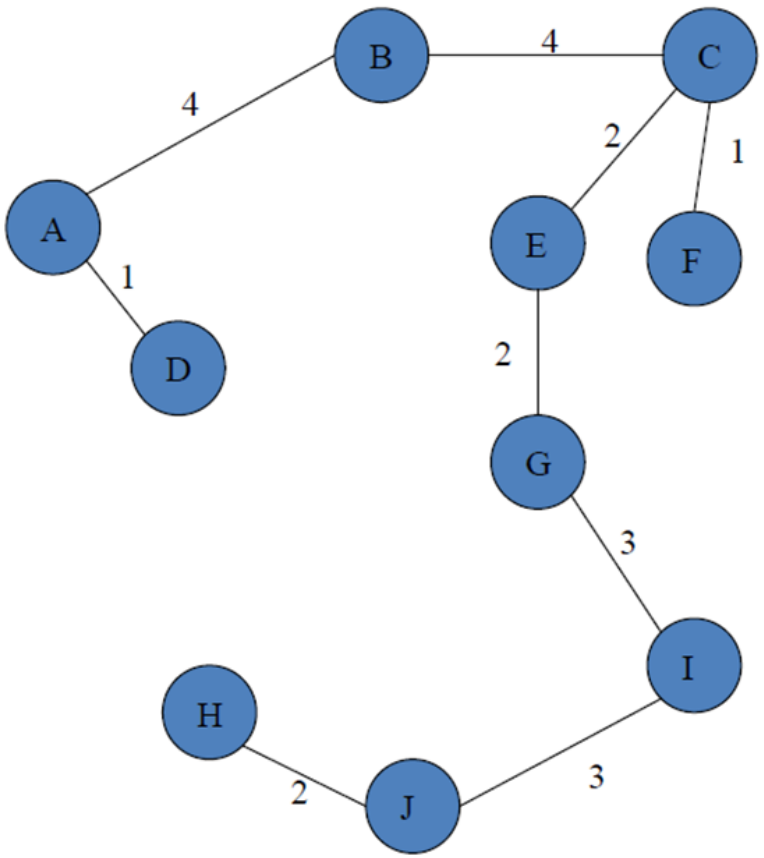



Add Edge

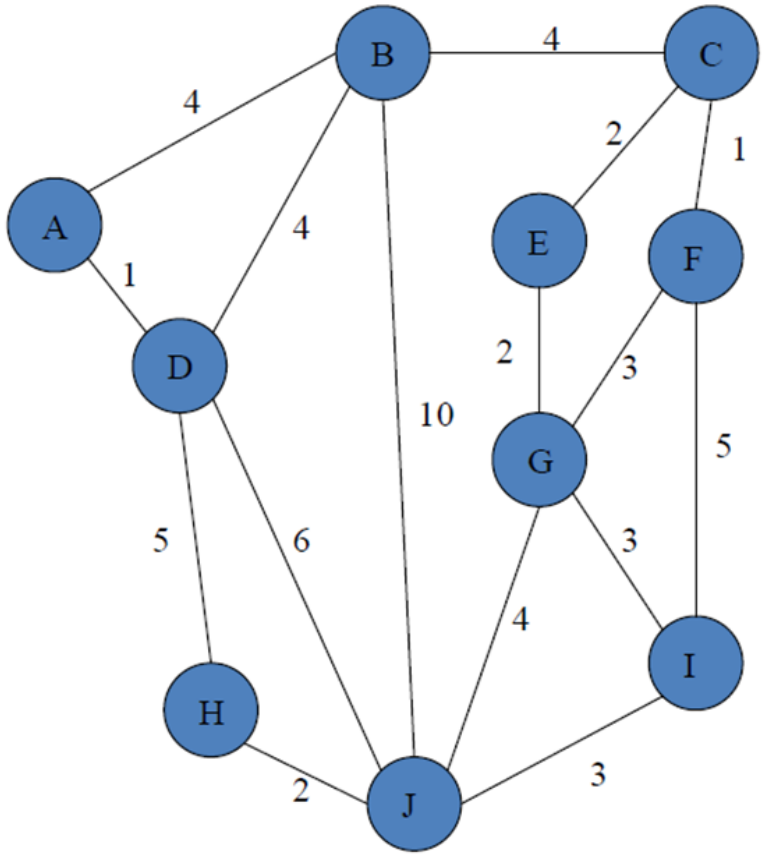

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3

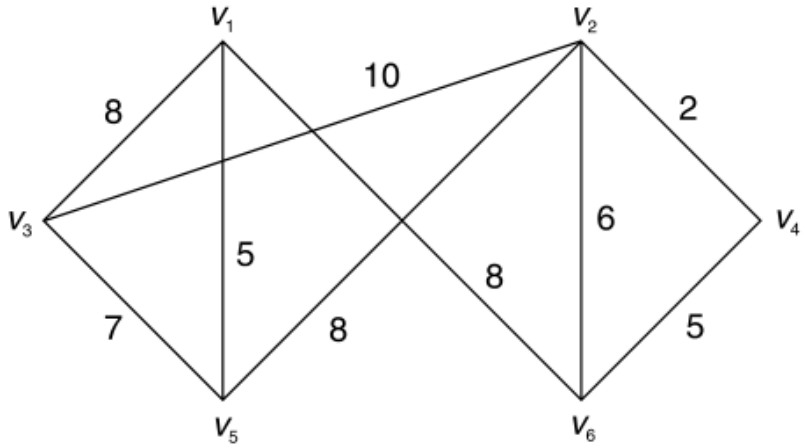

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

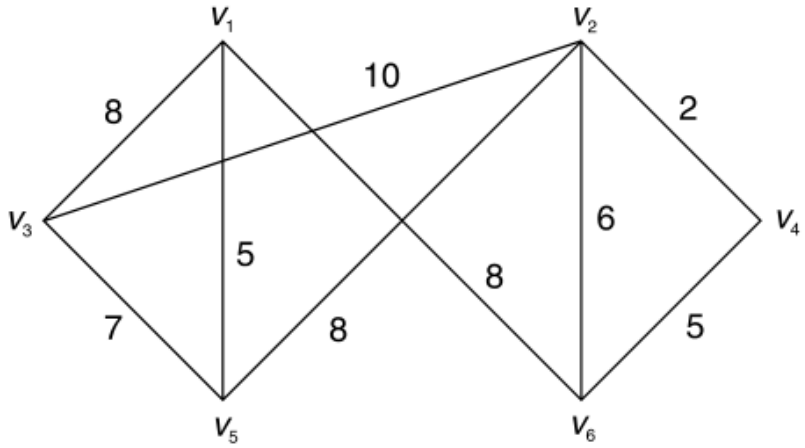
Minimum Spanning Tree



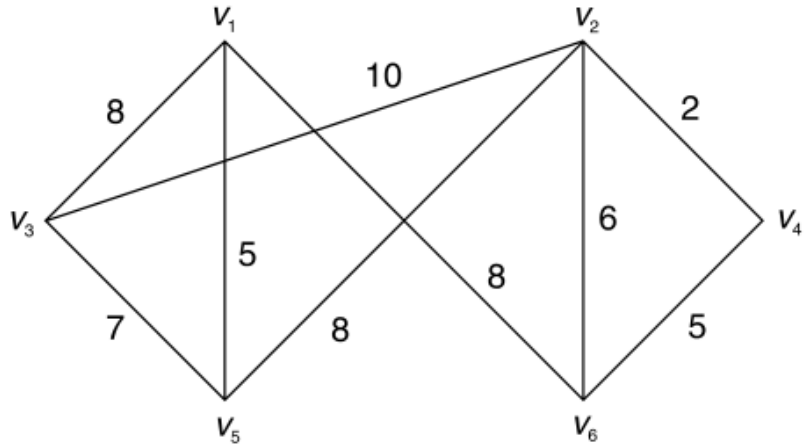

Complete Graph



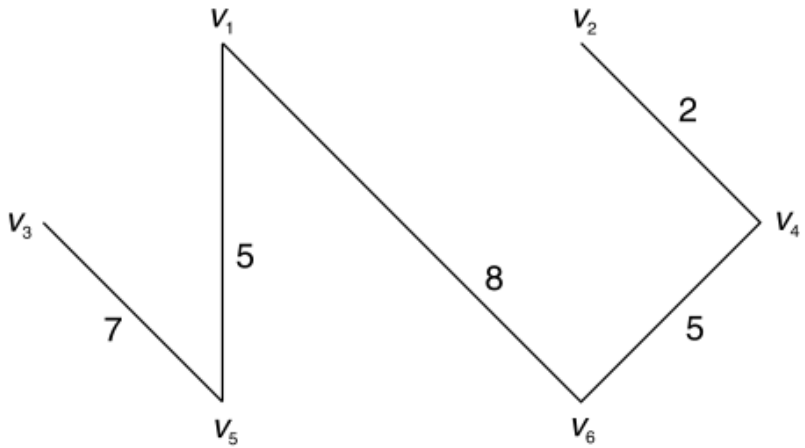

$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
8
10
2
6
5
8
7
8
5




| *Edges* | $(v_2, v_4)$ | $(v_1, v_5)$ | $(v_4, v_6)$ | $(v_2, v_6)$ | $(v_3, v_5)$ | $(v_1, v_3)$ | $(v_1, v_6)$ | $(v_2, v_5)$ | $(v_2, v_3)$ |
|---|---|---|---|---|---|---|---|---|---|
| *Weight* | 2 | 5 | 5 | 6 | 7 | 8 | 8 | 8 | 10 |
| *Add?* | Yes | Yes | Yes | No | Yes | No | Yes | No | No |




| *Edges* | $(v_2, v_4)$ | $(v_1, v_5)$ | $(v_4, v_6)$ | $(v_2, v_6)$ | $(v_3, v_5)$ | $(v_1, v_3)$ | $(v_1, v_6)$ | $(v_2, v_5)$ | $(v_2, v_3)$ |
|---|---|---|---|---|---|---|---|---|---|
| *Weight* | 2 | 5 | 5 | 6 | 7 | 8 | 8 | 8 | 10 |
| *Add?* | Yes | Yes | Yes | No | Yes | No | Yes | No | No |



# Prim's Algorithm

## Main Idea

- Start with one vertex.
- Repeatedly add the smallest weight edge that connects a new vertex to the growing tree.
- Continue until all vertices are included.

# Prim's Algorithm - Step by Step

**Input:** A connected weighted undirected graph $G$ with $n$ vertices
**Output:** A minimum spanning tree $T$ of $G$

1. **Initialization Step:**
   - Select an edge with minimum weight from the graph
   - Initialize tree $T$ with this edge
   - Let $S$ be the set of vertices currently in $T$
2. **Expansion Steps:** (Repeat $n-2$ times)
   - Find the minimum-weight edge $e$ that connects:
     - A vertex in $S$ (current tree) to
     - A vertex not in $S$ (outside the tree)
   - This edge $e$ will not create a cycle since it connects the tree to a new vertex
   - Add edge $e$ to $T$
   - Add the new vertex to set $S$
3. **Termination:**
   - After $n-1$ edges are selected, $T$ is a minimum spanning tree
   - Return $T$

# Why it Works:

- **Greedy Choice Property**: The minimum edge connecting the tree to outside vertices belongs to some MST
- **Optimal Substructure**: After adding an edge, the problem reduces to finding MST for the remaining graph
- **No Cycles**: Since we always connect to a new vertex, no cycles are created

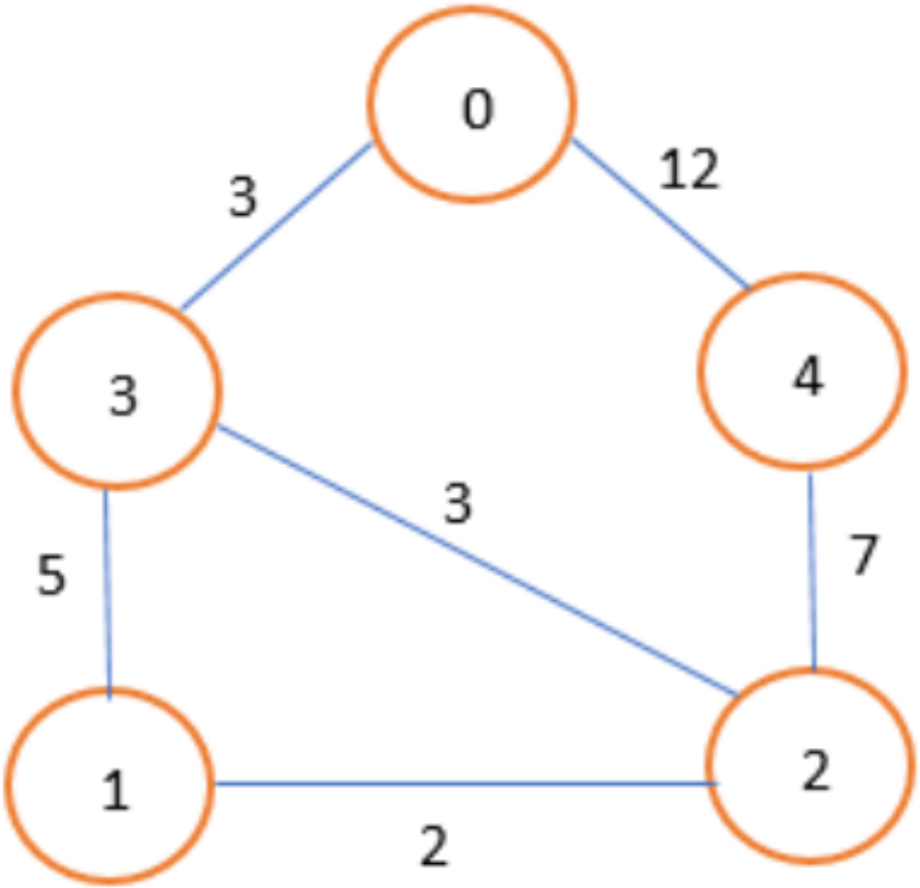
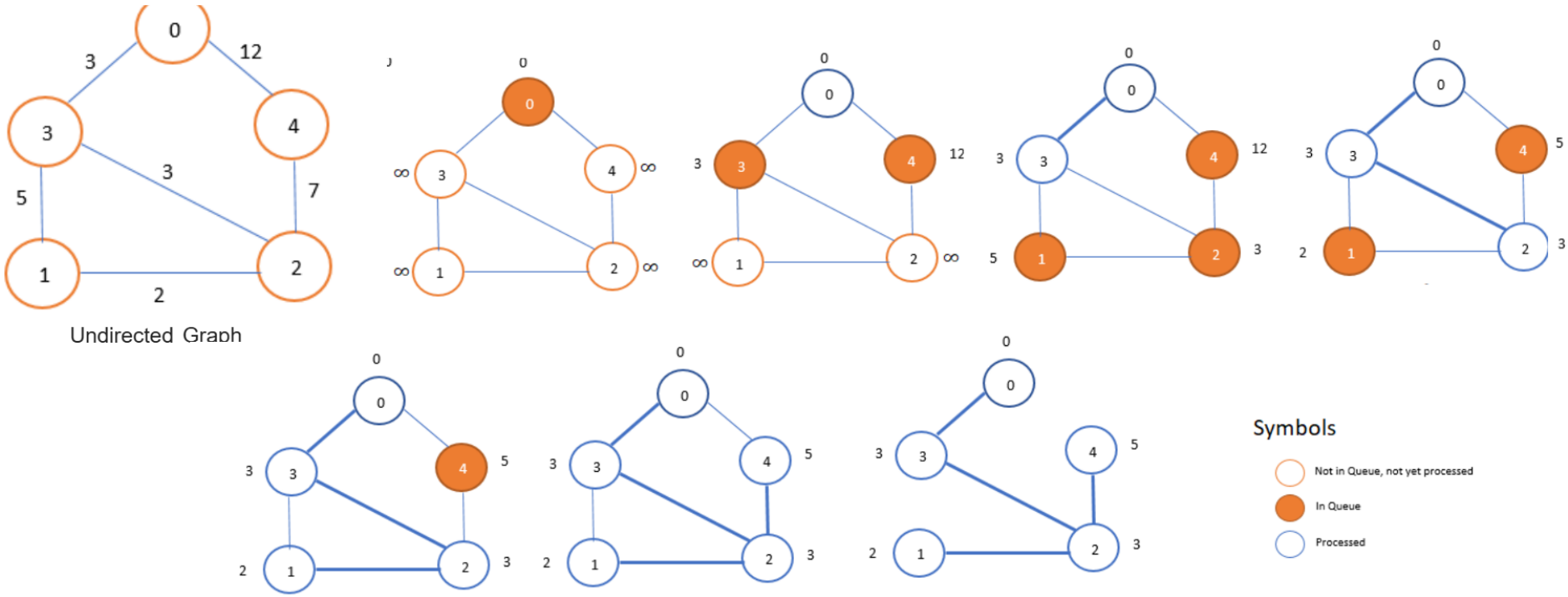
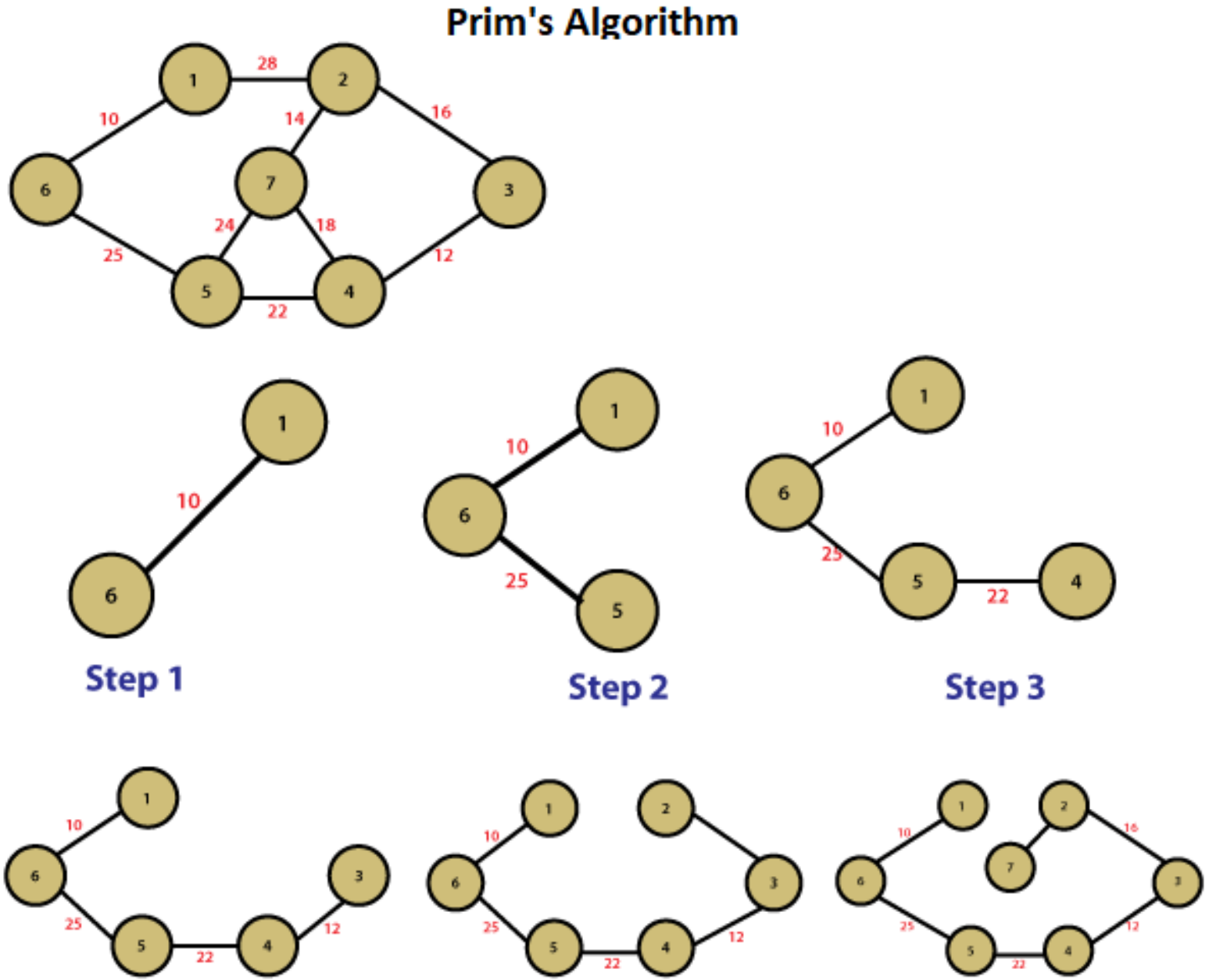



Fig 1: Undirected Graph


0
3
12
3
4
3
5
7
1
2
2
Undirected Graph
Symbols
Not in Queue, not yet processed
In Queue
Processed


Prim's Algorithm
1
2
3
4
5
6
7
28
10
14
16
24
18
25
12
22
Step 1
Step 2
Step 3

Suppose $T$ is a **Binary Tree**[1] with every vertex $N$ has the following property

[1] Each vertex has at most two children

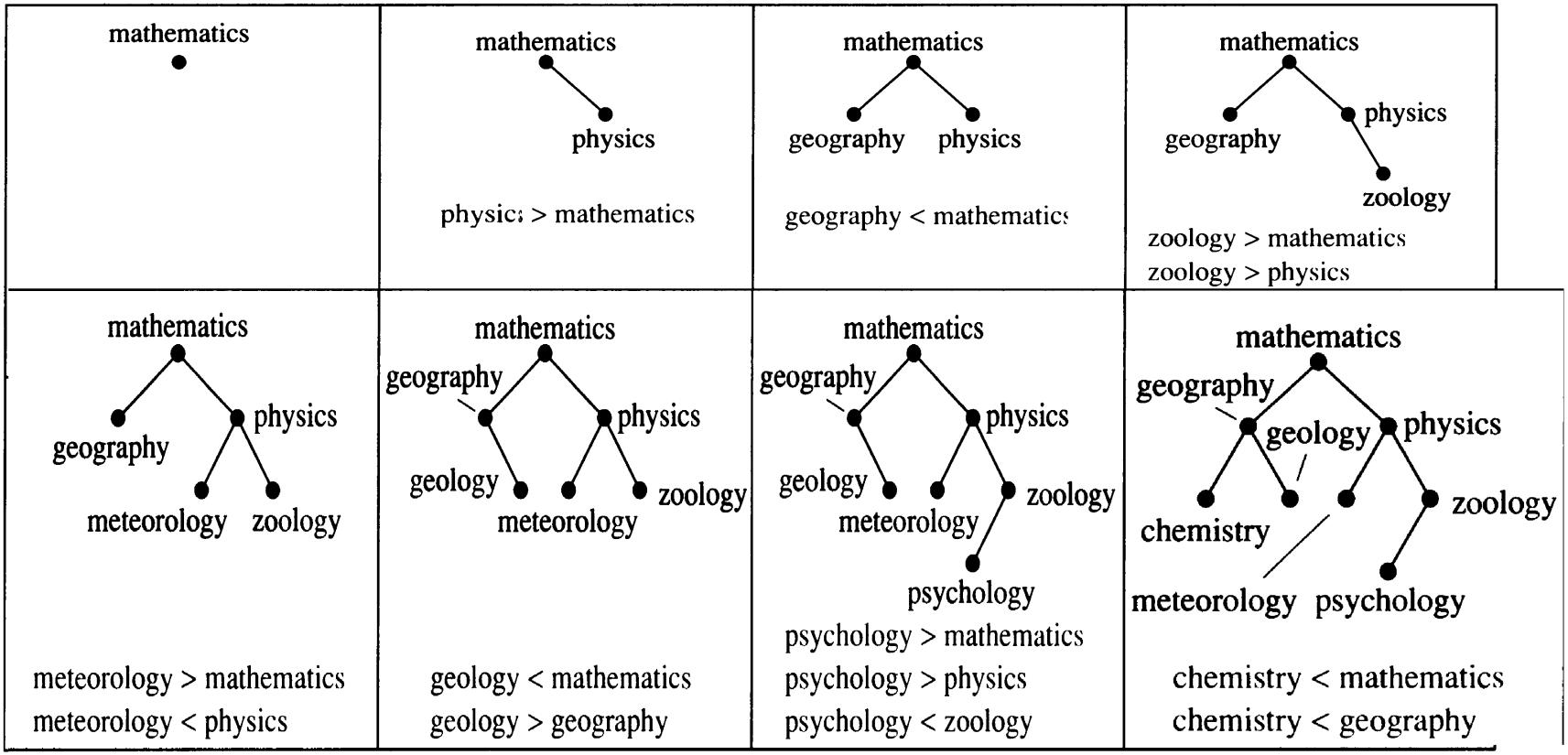
Suppose $T$ is a **Binary Tree**[1] with every vertex $N$ has the following property
$N > M$ for every vertex $M$ in a left subtree of $N$

[1] Each vertex has at most two children

Suppose $T$ is a **Binary Tree**[1] with every vertex $N$ has the following property
$N > M$ for every vertex $M$ in a left subtree of $N$
$N \leq M$ for every vertex $M$ in a right subtree of $N$

[1] Each vertex has at most two children

mathematics, physics, geography, zoology, meteorology, geology, psychology, and chemistry (using alphabetical order).



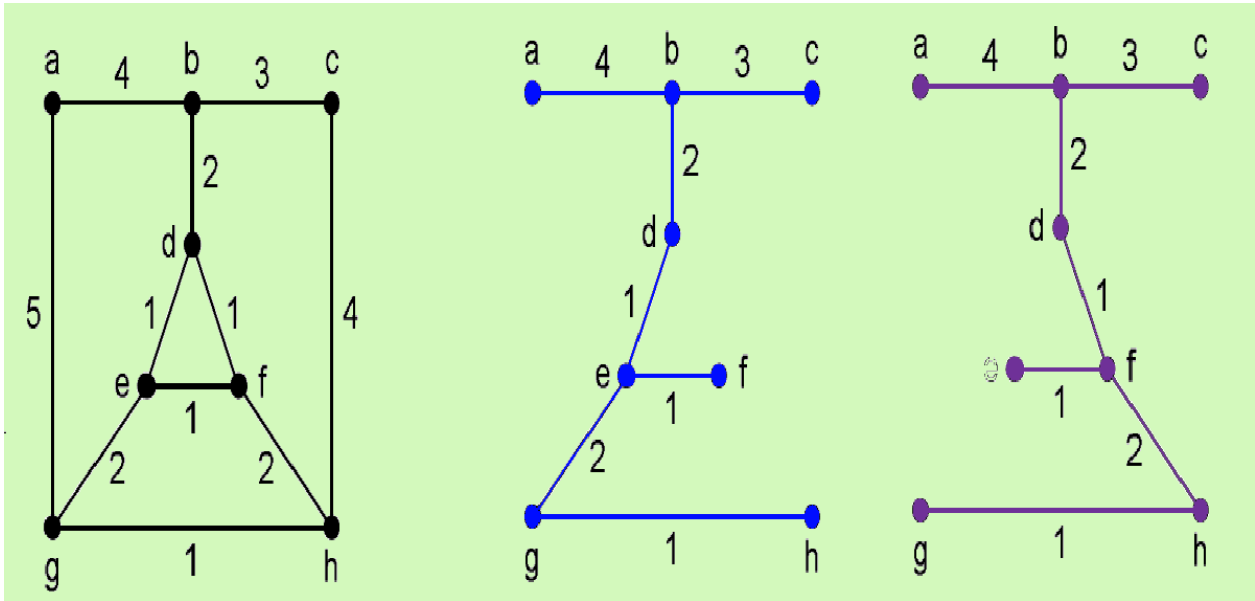
# Application of Minimum Spanning Trees

**Spanning Tree:** A spanning tree of a graph $G$ is a subgraph of $G$ that is a tree and contains all vertices of the graph.

**Minimum Spanning Tree (MST):**

(1) It spans the graph,

(2) It is a minimum.



**What is an electric network?**

**What is an electric network?**

A collection of interconnected electrical elements (or devices).

**What is an electric network?**

A collection of interconnected electrical elements (or devices).

**Behavior of Electrical network**

**What is an electric network?**

A collection of interconnected electrical elements (or devices).

**Behavior of Electrical network**

**1**- The characterization of each device.

**What is an electric network?**

A collection of interconnected electrical elements (or devices).

**Behavior of Electrical network**

**1**- The characterization of each device.

**2**- How devices or connected (their topology).

**What is an electric network?**

A collection of interconnected electrical elements (or devices).

**Behavior of Electrical network**

**1**- The characterization of each device.

**2**- How devices or connected (their topology).

**Terminology**

*Node* - vertex, *Branch* - edge, *Loop* - cycle.

**What is an electric network?**

A collection of interconnected electrical elements (or devices).

**Behavior of Electrical network**

**1**- The characterization of each device.

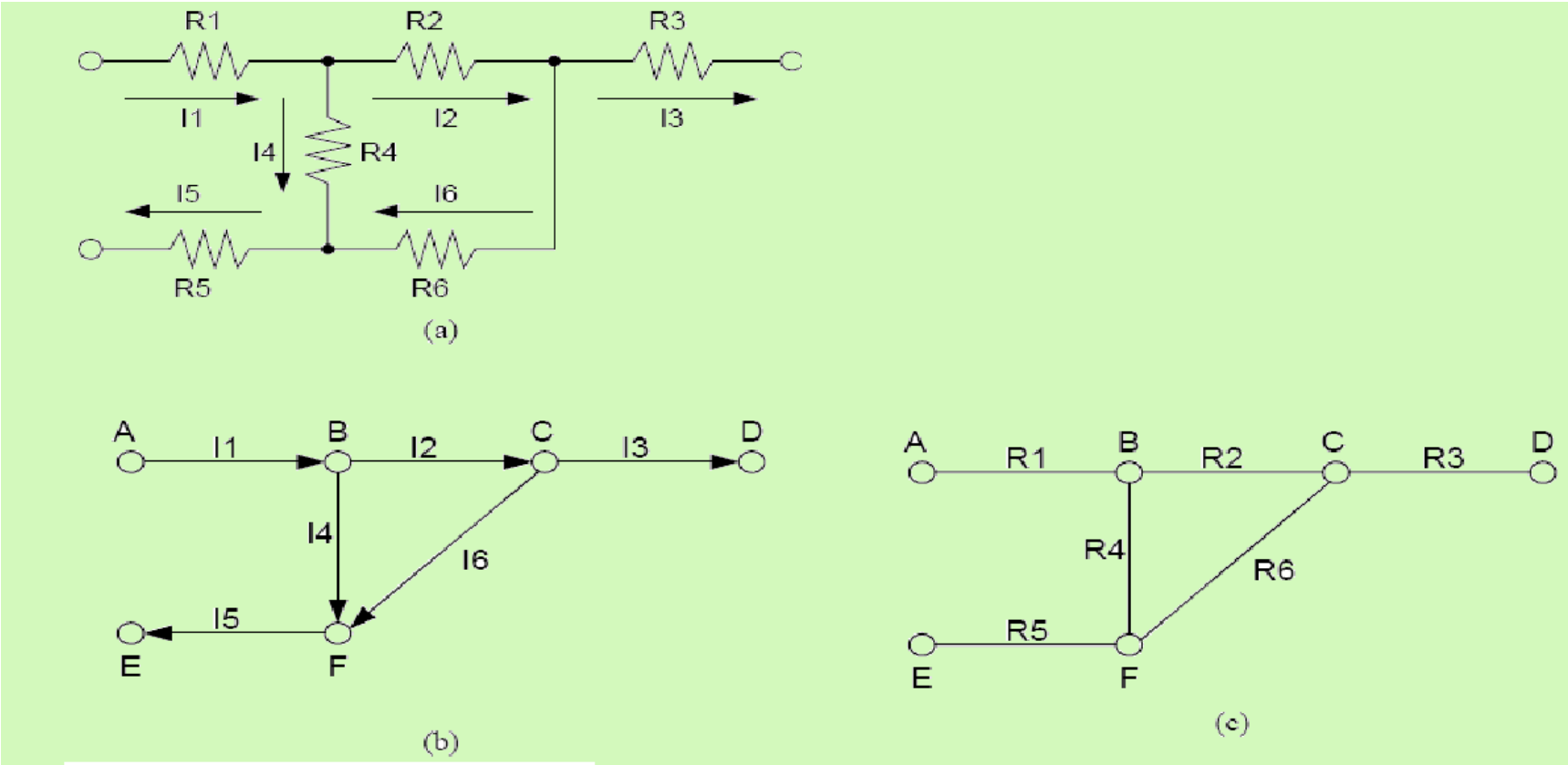
**2**- How devices or connected (their topology).

**Terminology**

*Node* - vertex, *Branch* - edge, *Loop* - cycle.

An electrical network is known as an *Electrical Circuit*.

**Drawing Graph:** *Junctions* and *devices* are represented by *nodes* and *branches*, respectively.



# References


- Deo. N., Graph theory with application to the engineering and computer science, Printce-Hall, Inc. (1974).
- Wu. B. Y. and Chao. K. M., Spanning Trees and Optimization Problems, CHAPMAN and HALL/CRC. (2004) .
- Rosen. K. H., Discrete Mathematics and Its Applications, McGraw-Hill Inc. (2007).
- Gale, D. and Shapley, L. S., College admissions and the stability of marriage. The American Mathematical Monthly, 69(1):9-15, 1962.
- Broersma, H. J., Li, X. L., Woegingerr, G. and Zhang, S., Paths and cycles in colored graphs, Australian J. Combin. 31(2005), 297-309.


Thank
YOU!