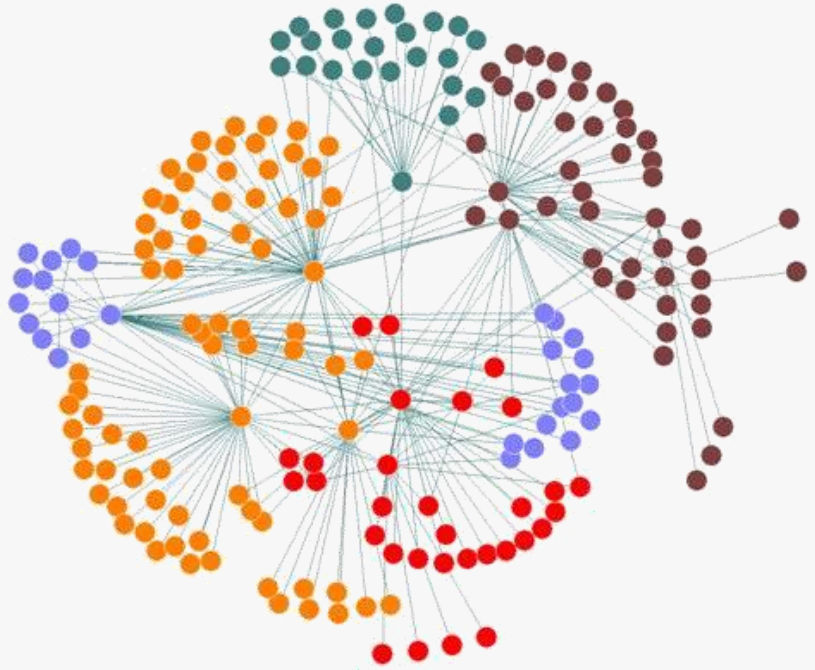
Social Network Analysis

By
Dr. Hussein Hazimeh



Lebanese University
Faculty of Information 1

Data Science Departement

3rd year – Social Network Analysis

Spring – 2021 – Chapter 2

Agenda

- Basic Concepts of Graph Theory
- Graph Representations
- Graph Terminologies
- Different Type of Graphs
- Minimum spanning trees

1

Graph Theory

Introduction

Why Graph Theory ?

- Graphs used to model pair wise relations between objects
- Generally a network can be represented by a graph
- Many practical problems can be easily represented in terms of graph theory

Definition: Graphs in Graph Theory

- A graph is a collection of nodes and edges
- Denoted by $G = (V, E)$.

V = **nodes** (vertices, points).
E = **edges** (links, arcs) between pairs of nodes.
Graph size parameters: $n = |V|, m = |E|$.

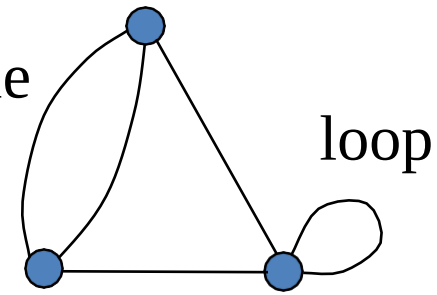
- Model relationships between pairs of objects

Vertex & Edge

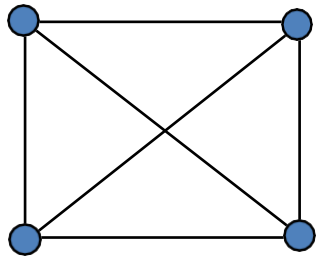
- Vertex /Node
  - Basic Element
  - Drawn as a node or a dot.
  - V**ertex set** of G is usually denoted by V(G), or V or V_G
- Edge /Arcs
  - A set of two elements
  - Drawn as a line connecting two vertices, called end vertices, or endpoints.
  - The edge set of G is usually denoted by E(G), or E or E_G
- Neighborhood
  - For any node v, the set of nodes it is connected to via an edge is called its neighborhood and is represented as N(v)

Simple Graph

Simple graph : A graph has no loops or multiple edges



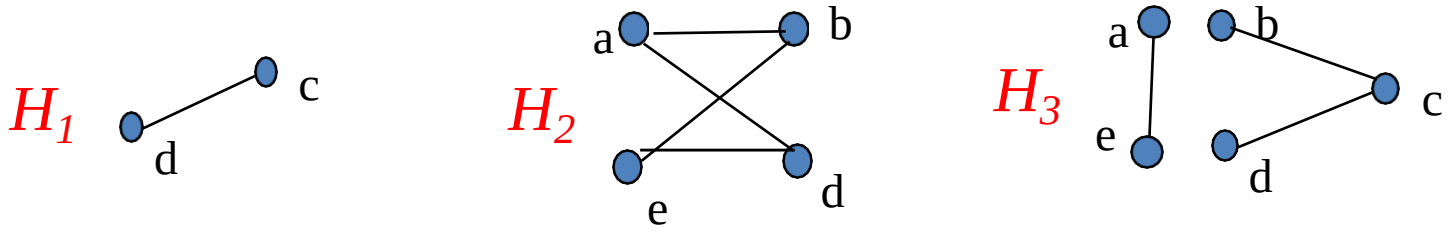
It is not simple.



It is a simple graph.

Connected & Disconnected Graphs

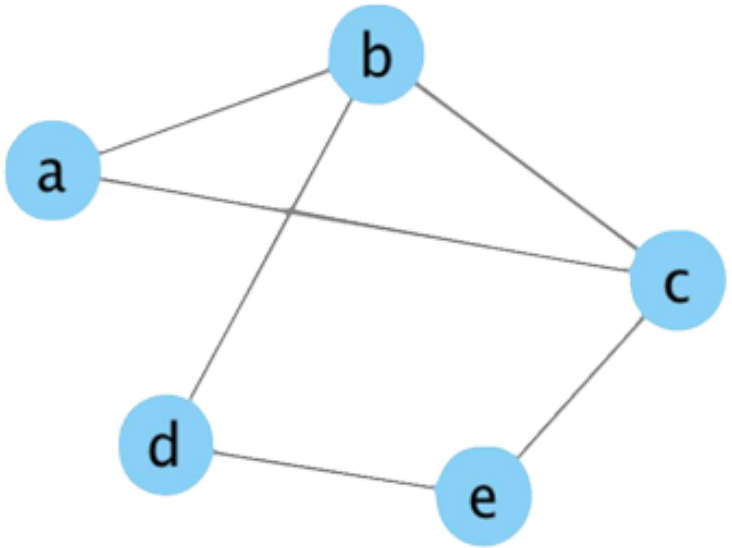
- ***Connected*** : There exists at least one path between two vertices
- ***Disconnected*** : Otherwise
- Example:
  - H_1 and H_2 are connected
  - H_3 is disconnected



Edge types

- **Undirected**;
  - E.g., distance between two cities, friendships…
- **Directed**; ordered pairs of nodes.
  - E.g ,…
  - Directed edges have a **source** (head, origin) and **target** (tail, destination) vertices
- **Weighted ;** usually weight is associated .

Undirected Graphs



V = {a, b, c, d, e}

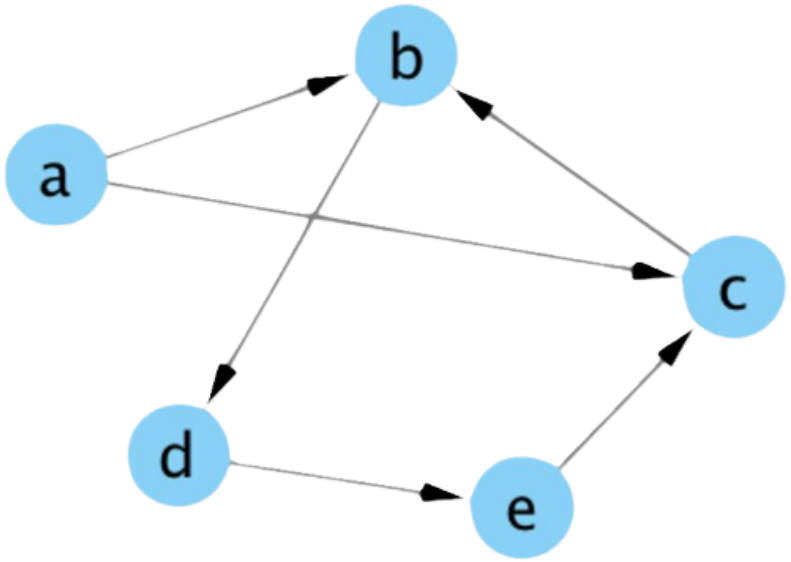
|V| = 5

E = { {a, b}, {a, c}, {b, c}, {b, d}, {d, e}, {b, e} }

|E| = 6

N(4) := Neighborhood (b) ={a,d,c}

Directed Graphs



Can we traverse this graph from **a** to **e**?

Can we traverse this graph from **e** to **a**?

V = {a, b, c, d, e}

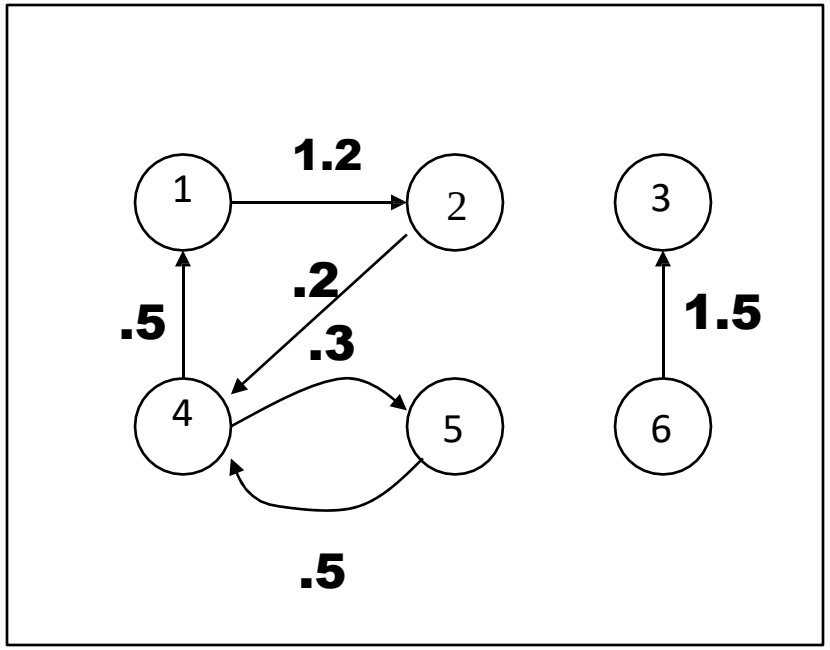
E = { (a, b), (a, c), (c, b), (b, d), (d, e), (e, c) }

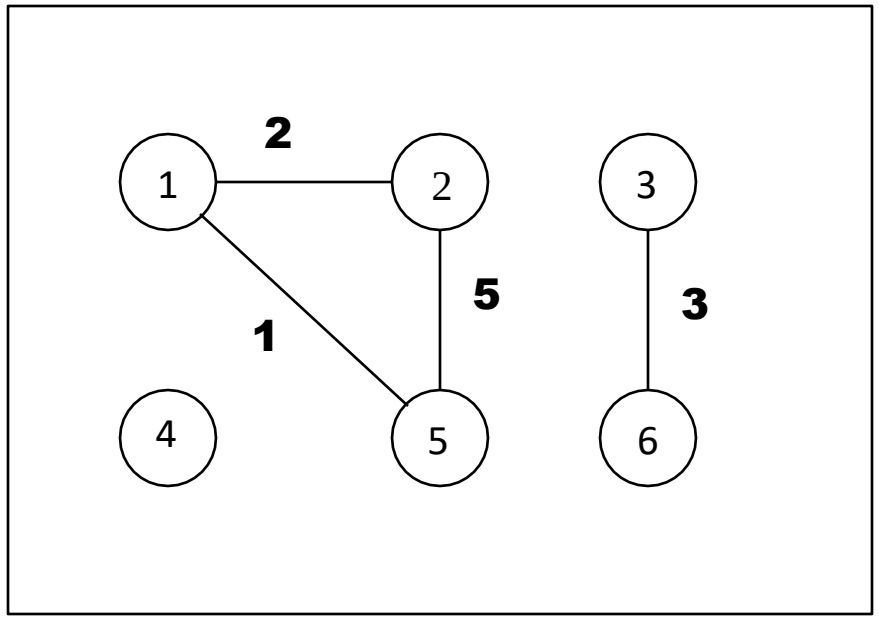
|V| = 5

|E| = 6

Weighted graph

Weighted graph is a graph for which each edge has an associated ***weight***, usually given by a ***weight function*** $w: E \rightarrow \mathbf{R}$.

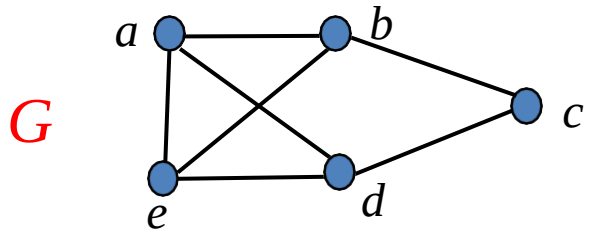


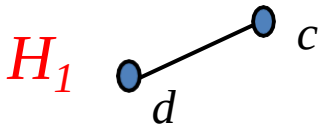


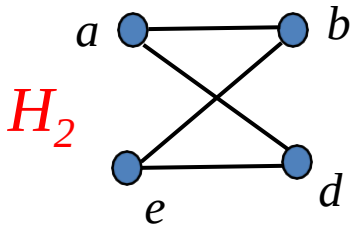
Subgraphs

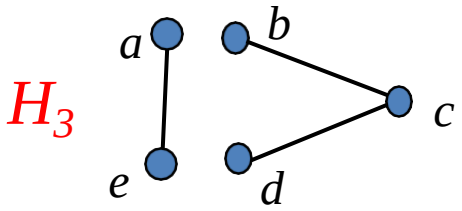
- G' = (V', E') is a subgraph of a graph G = (V, E) if
- V' ⊆ V and E' ⊆ E

Example: H_1, H_2, and H_3 are subgraphs of G



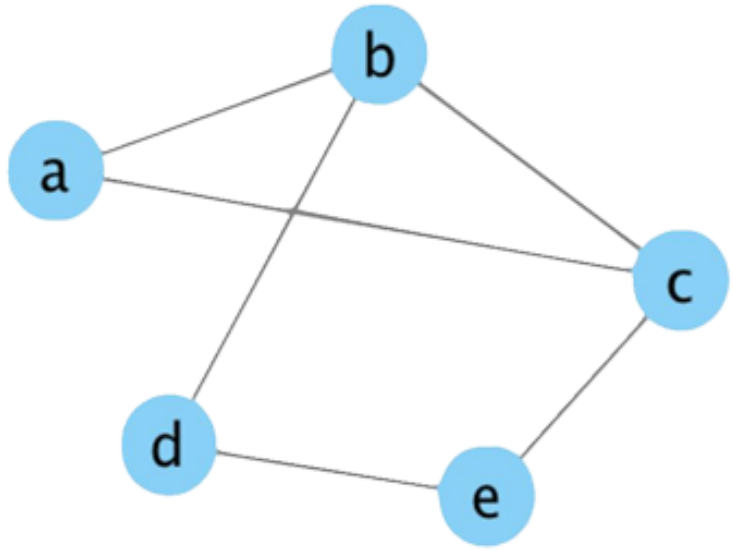






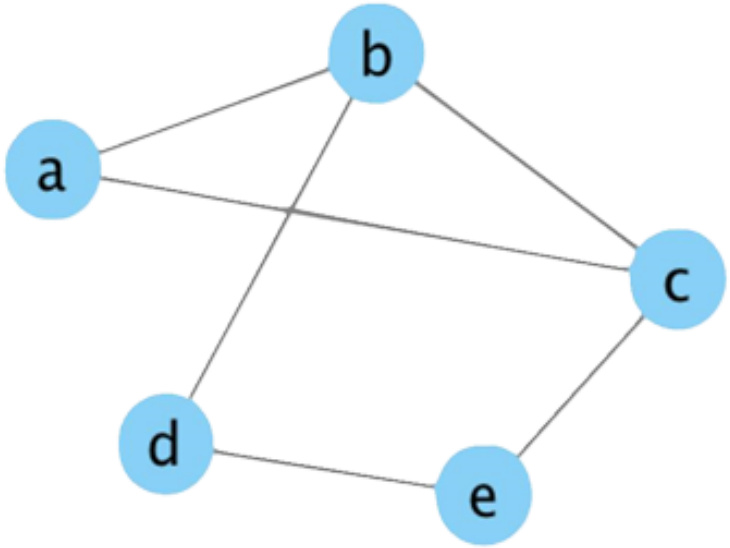
Walk

- From v_0 to v_l
- Ex. < a, b, d, b, c >
- Ca repeat a vertex or edge



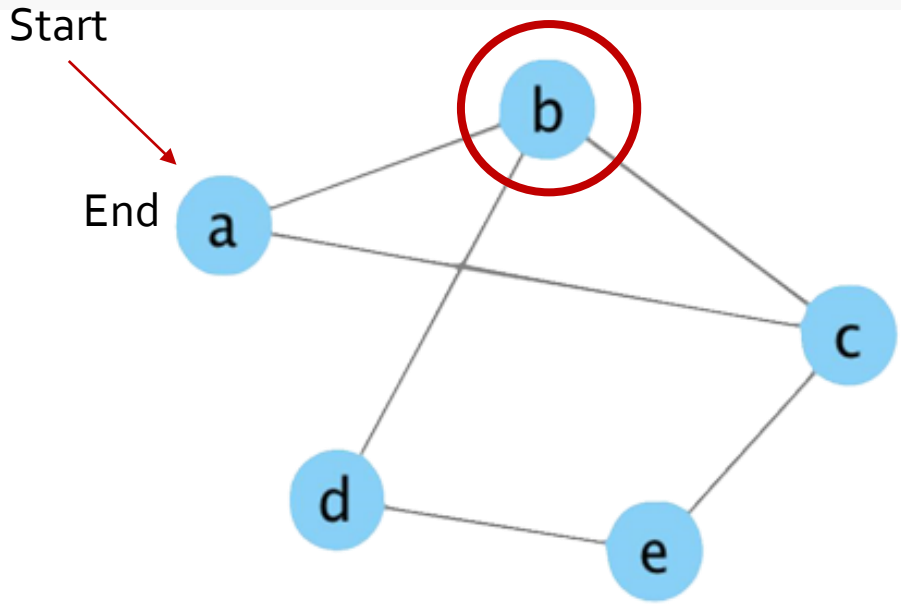
Path

- From v_0 to v_l
- Ex. <a, b, c>
- No vertex is repeated



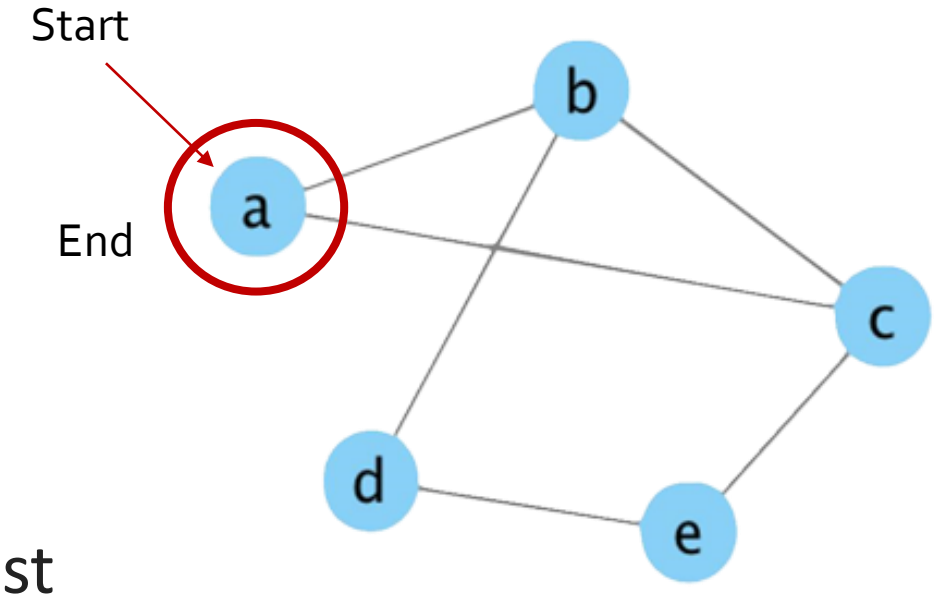
Circut

- From v_0 to v_0
- Ex. < a, b, d, e, c, b, a >
- Can repeat a vertex or edge



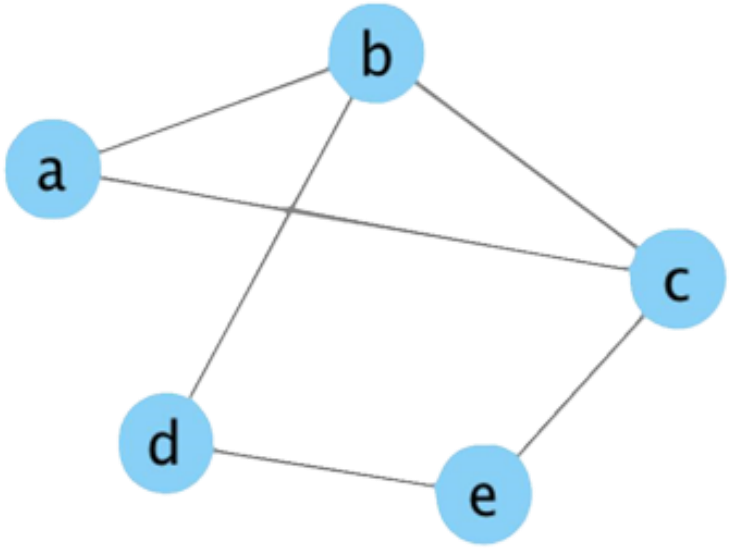
Cycle

- From v_0 to v_0
- Ex. <a, b, c, a>
- Length: at least three
- No vertex is repeated except the first and last



[Question] Walk / Path / Circuit / Cycle?

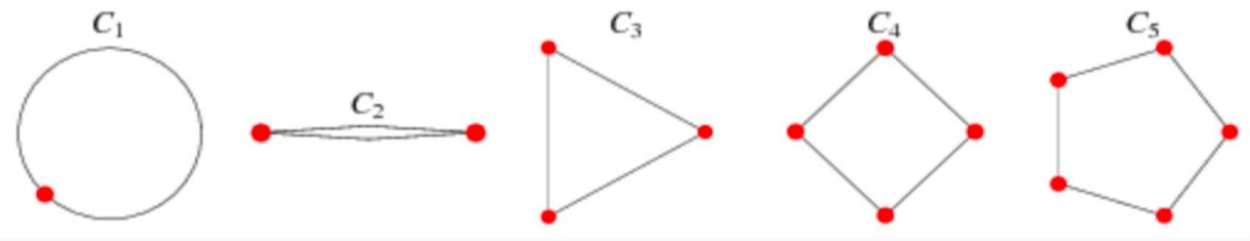
- 1) <a, b, a>
- 2) <a, b, d, e, c>
- 3) <a, b, d, e, c, b>
- 4) <a, b, d, e, c, a>



2
Graph Classes

Cycle Graphs (Cn)

- Single cycle through all vertices

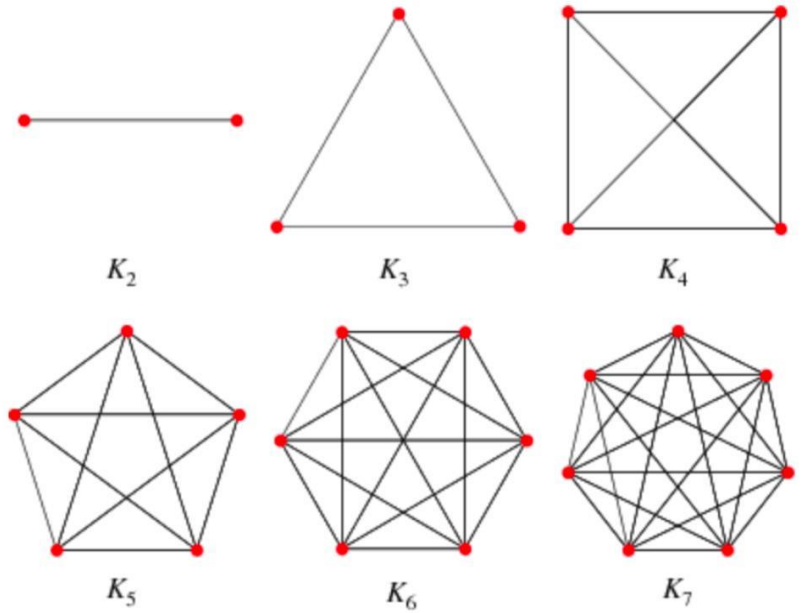


E = **n**
V = **n**

Complete Graphs (Kn)

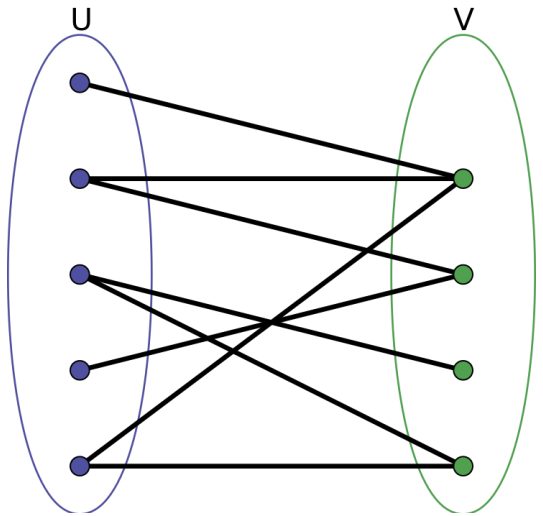
- Each vertex pair is connected by an edge.
- [Question] How many edges does K_n have?

$$E = \frac{n(n-1)}{2}$$

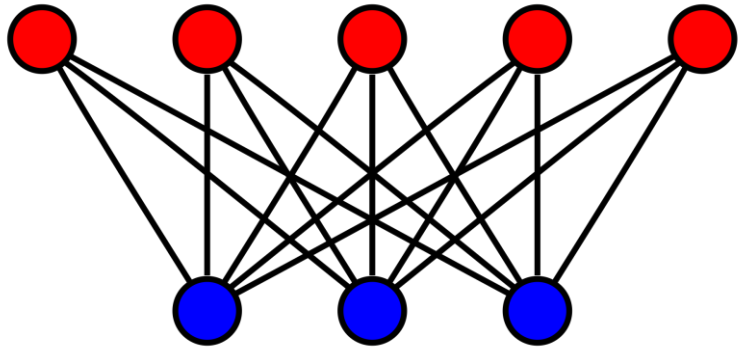


Bipartite Graphs

- Vertices: two disjoint sets.
- No two vertices within the same set are connected.



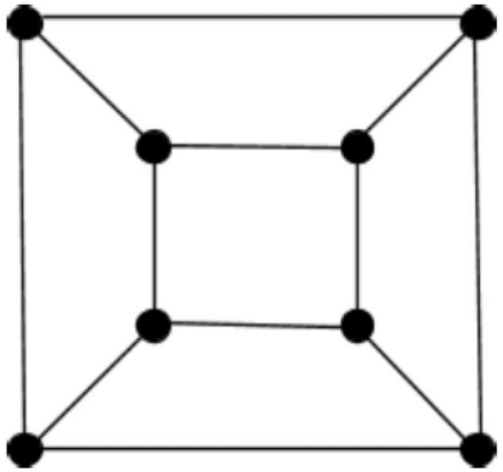
$$G = (U, V, E)$$



A complete bipartite graph with m = 5 and n = 3

Bipartite Graphs

Is this a bipartite graph?



3

Graph Representation

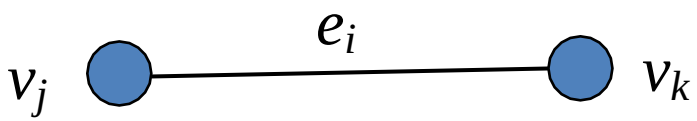
Adjacency Matrix

Incidence Matrix

Adjacency List

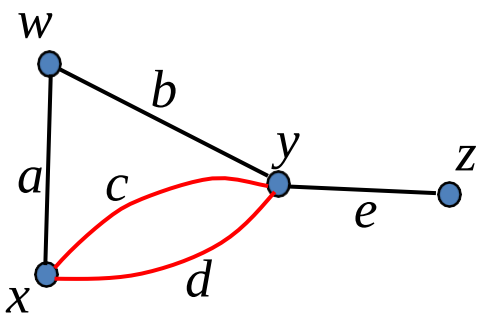
Adjacency, Incidence, and Degree

- Assume e_i is an edge whose endpoints are (v_j, v_k)
- The vertices v_j and v_k are said to be ***adjacent***
- The edge e_i is said to be ***incident upon*** v_j
- ***Degree*** of a vertex v_k is the number of edges incident upon v_k. It is denoted as $d(v_k)$



Adjacency Matrix

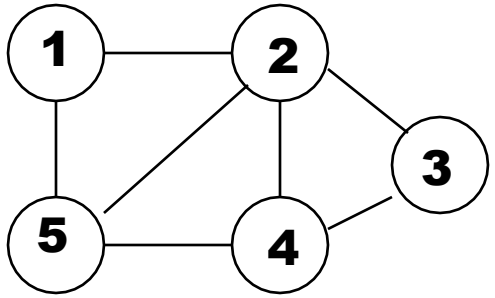
- Let $G = (V, E)$, $|V| = n$ and $|E| = m$
- The ***adjacency matrix*** of G written $A(G)$, is the $|V| \times |V|$ matrix in which entry $a_{i,j}$ is the number of edges in G with endpoints $\{v_i, v_j\}$.



$$\begin{array}{c} \\ w \\ x \\ y \\ z \end{array} \begin{array}{c} \begin{array}{cccc} w & x & y & z \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 2 & 0 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Adjacency Matrix

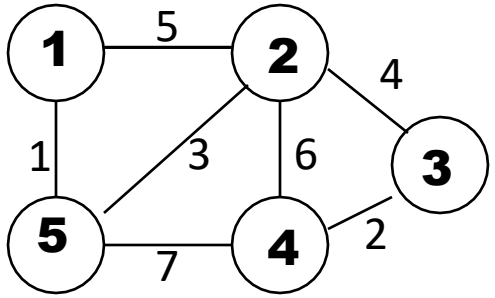
- Let $G = (V, E)$, $|V| = n$ and $|E| = m$
- The ***adjacency matrix*** of G written $A(G)$, is the $|V|$ x $|V|$ matrix in which entry $a_{i,j}$ is 1 if an edge exists otherwise it is 0



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacency Matrix (Weighted Graph)

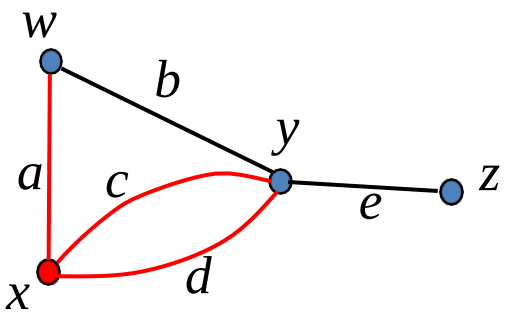
- Let $G = (V, E)$, $|V| = n$ and $|E| = m$
- The ***adjacency matrix*** of G written $A(G)$, is the $|V|$ x $|V|$ matrix in which entry $a_{i,j}$ is weight of the edge if it exists otherwise it is 0



	1	2	3	4	5
1	0	5	0	0	1
2	5	0	4	6	3
3	0	4	0	2	0
4	0	6	2	0	7
5	1	3	0	7	0

Incidence Matrix

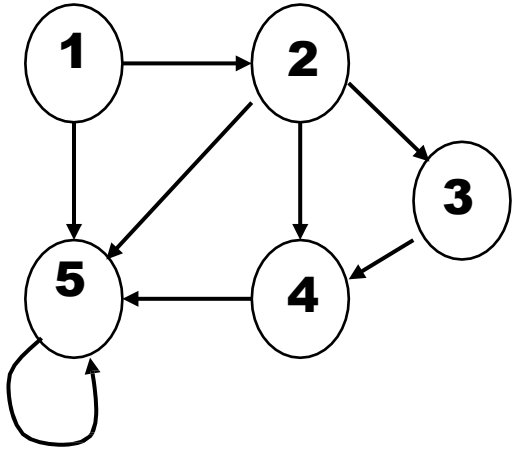
- Let $G = (V, E)$, $|V| = n$ and $|E| = m$
- The ***incidence matrix*** $M(G)$ is the $|V|$ x $|E|$ matrix in which entry $m_{i,j}$ is 1 if v_i is an endpoint of e_i and otherwise is 0.

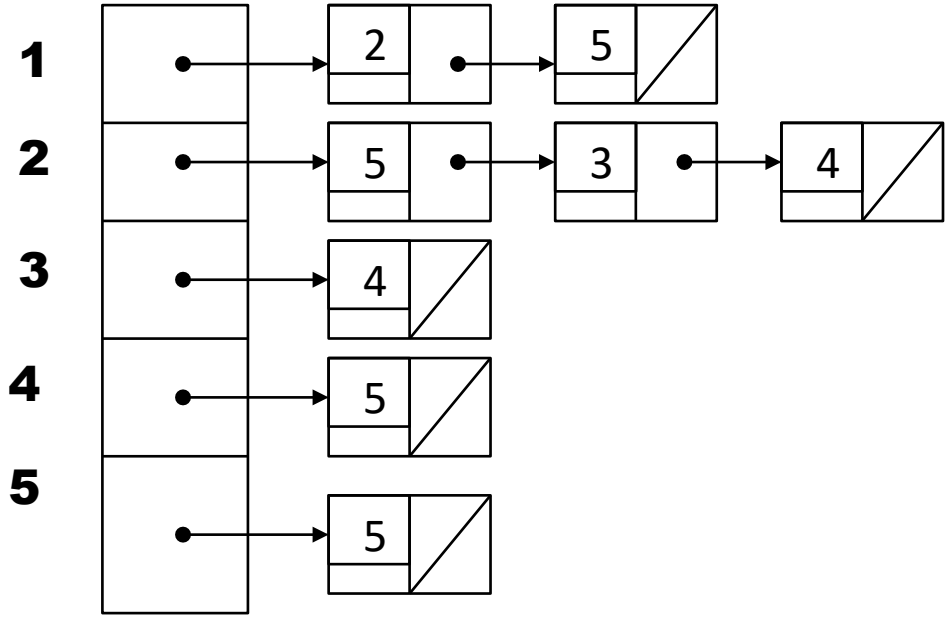


$$
\begin{array}{c} \\ w \\ x \\ y \\ z \end{array}
\begin{array}{c}
\begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{pmatrix}
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\end{array}
$$

Adjacency List Representation

- Adjacency-list representation
  - an array of $|V|$ elements, one for each vertex in V
  - For each $u \in V$, $ADJ[u]$ points to all its adjacent vertices.





4

Minimum Spanning Tree

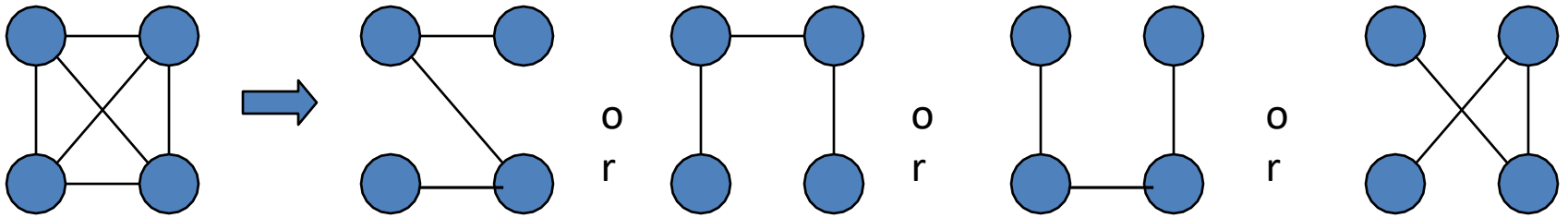
Spanning Trees

A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.

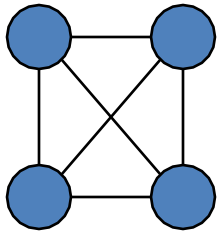
A graph may have many spanning trees.

Graph A

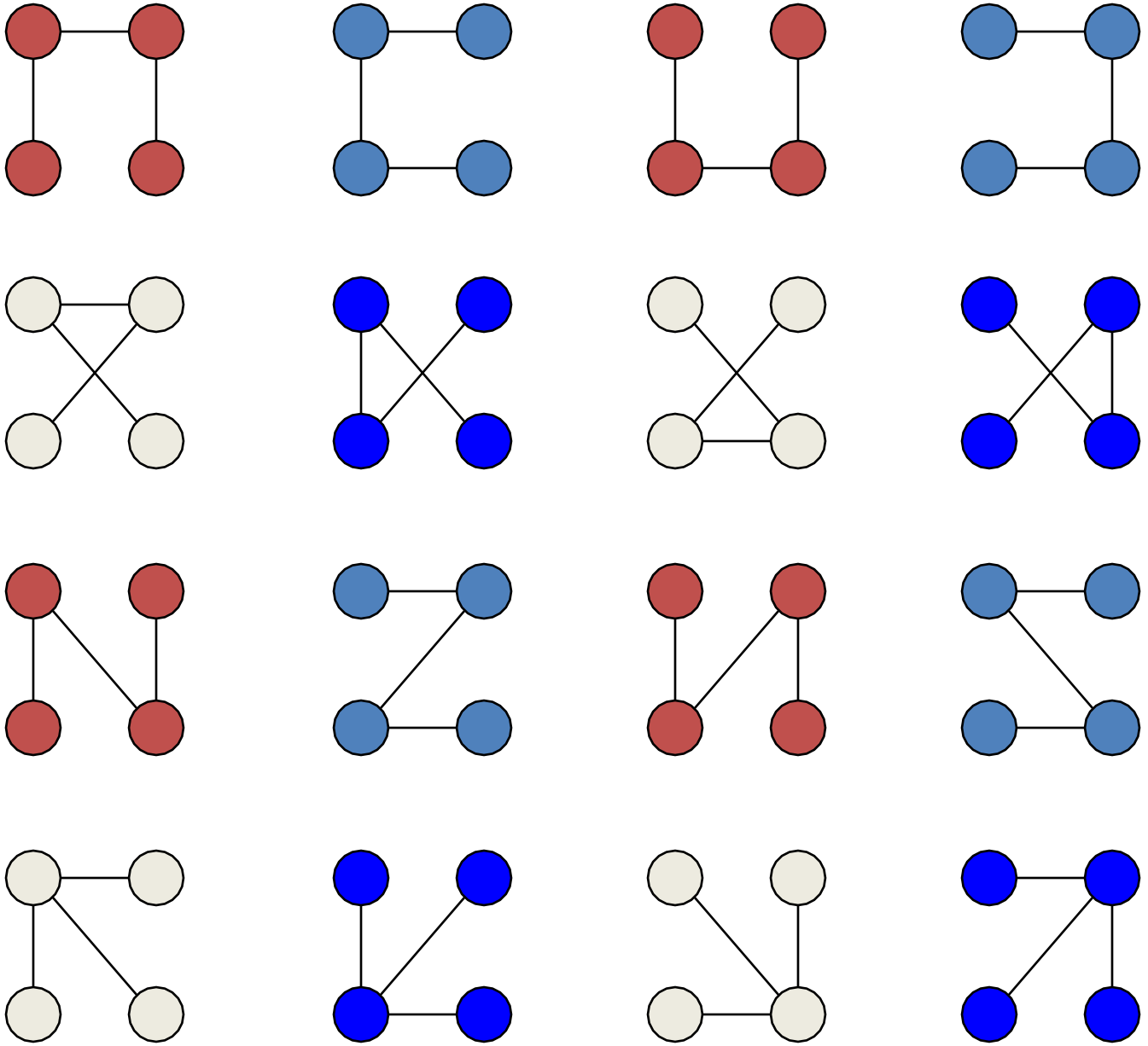
Some Spanning Trees from Graph A



Complete Graph



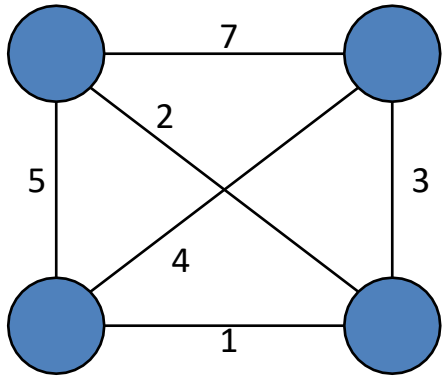
All 16 of its Spanning Trees



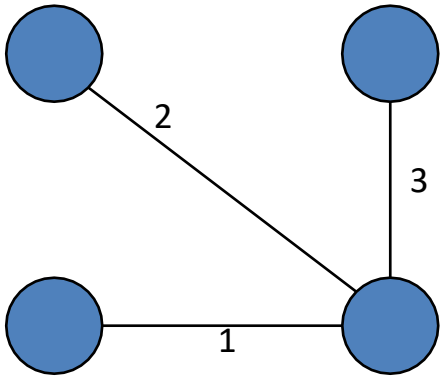
Minimum Spanning Tree

The Minimum Spanning Tree for a given graph is the Spanning Tree of minimum cost for that graph.

Complete Graph



Minimum Spanning Tree



Kruskal's Algorithm

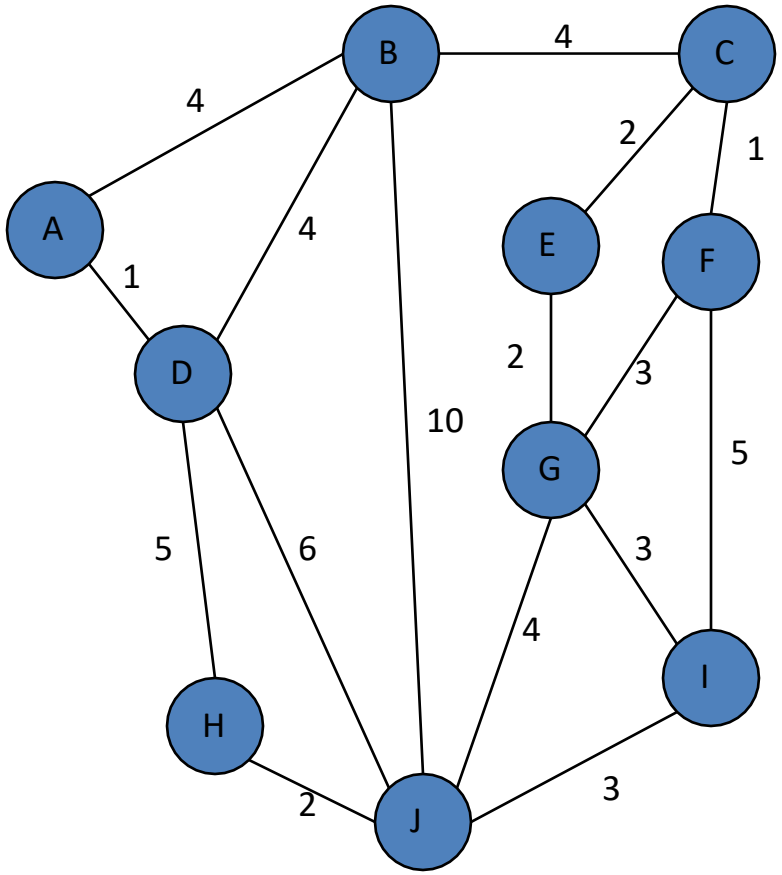
This algorithm creates a forest of trees. Initially the forest consists of n single node trees (and no edges). At each step, we add one edge (the cheapest one) so that it joins two trees together. If it were to form a cycle, it would simply link two nodes that were already part of a single connected tree, so that this edge would not be needed.

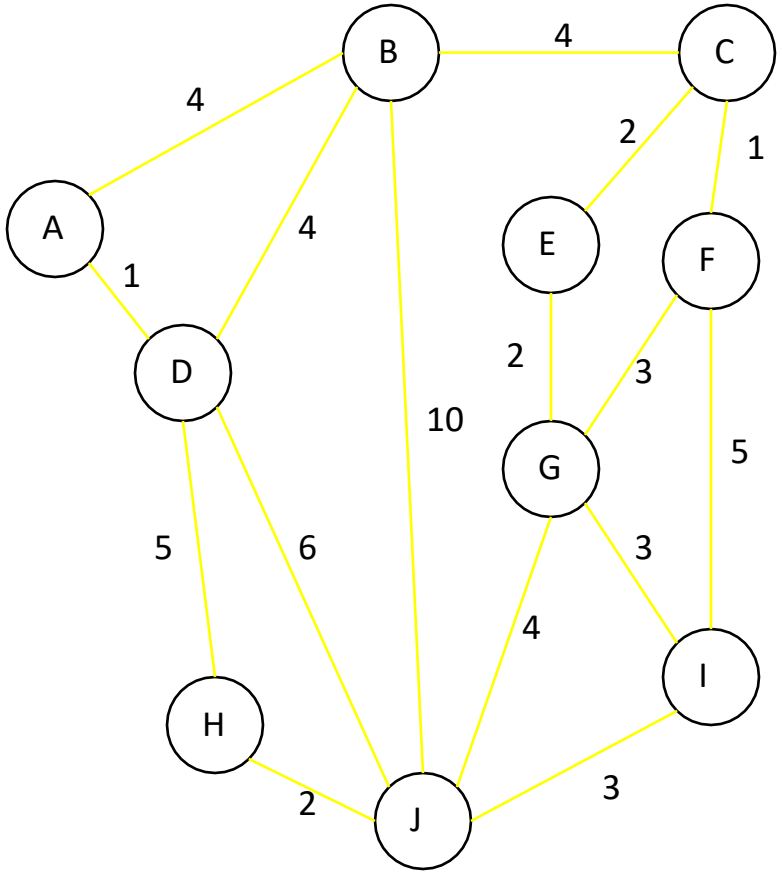
The steps are:

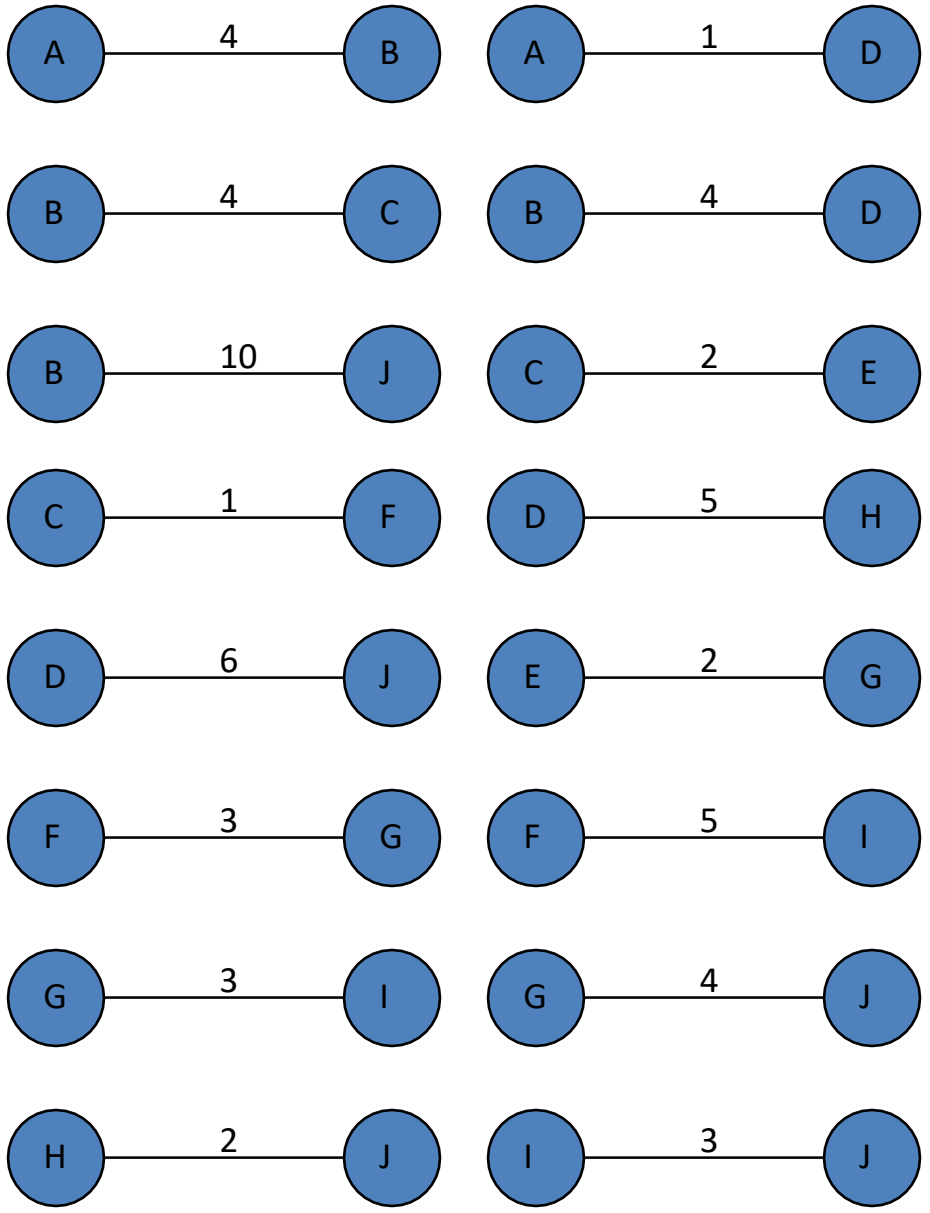
1. The forest is constructed - with each node in a separate tree.
2. The edges are placed in a priority queue.
3. Until we've added n-1 edges,
    1. Extract the cheapest edge from the queue,
    2. If it forms a cycle, reject it,
    3. Else add it to the forest. Adding it to the forest will join two trees together.

Every step will have joined two trees in the forest together, so that at the end,
there will only be one tree in T.

Complete Graph

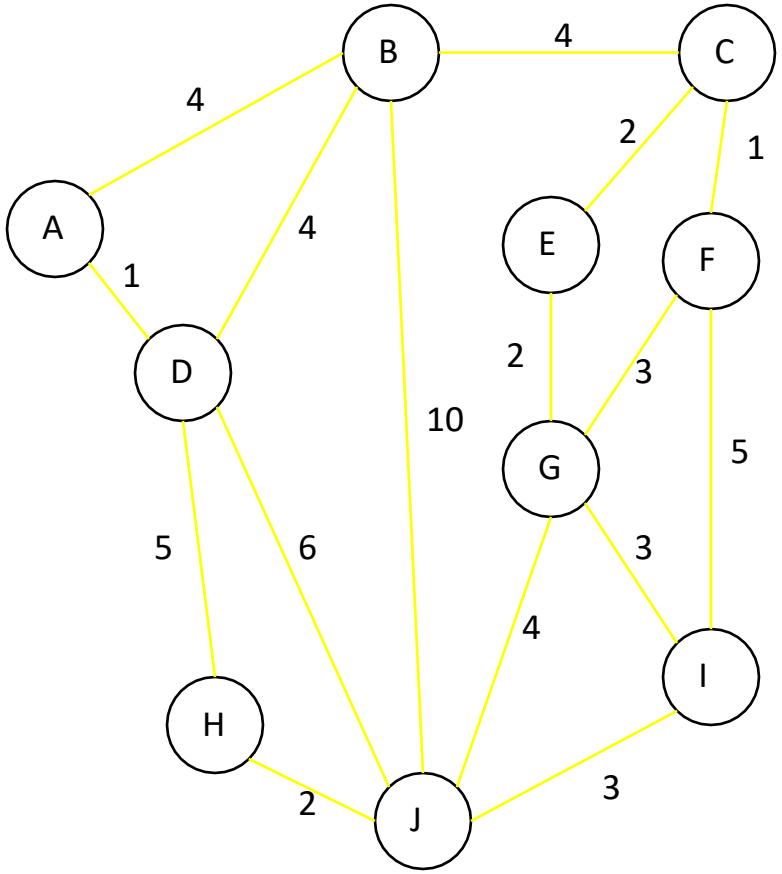


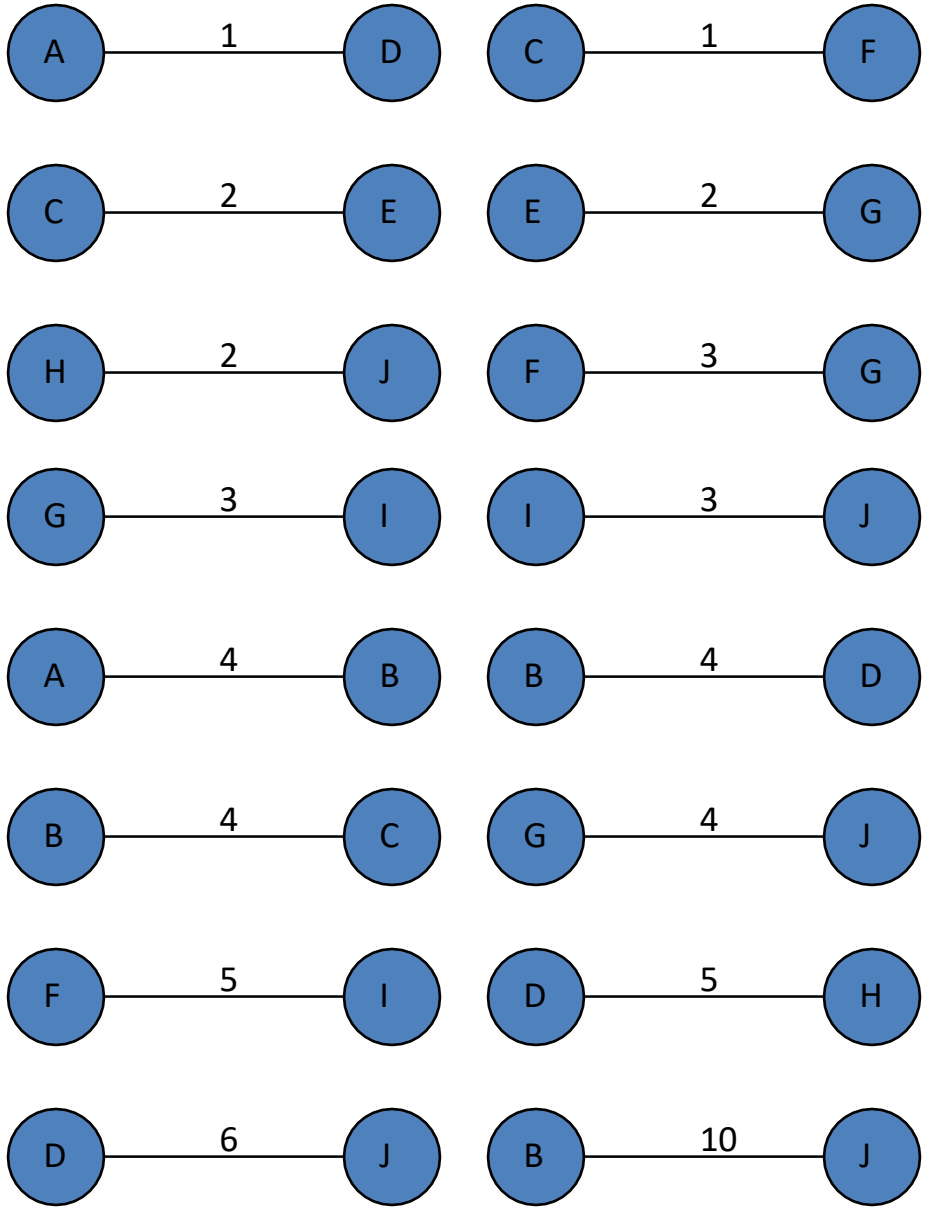

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3


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

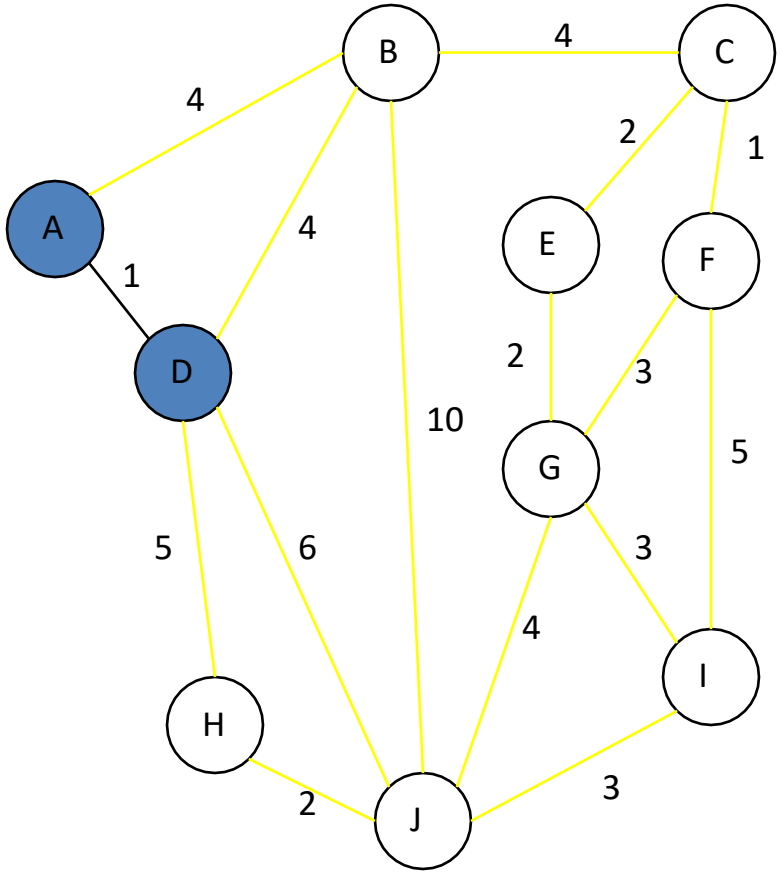
Sort Edges

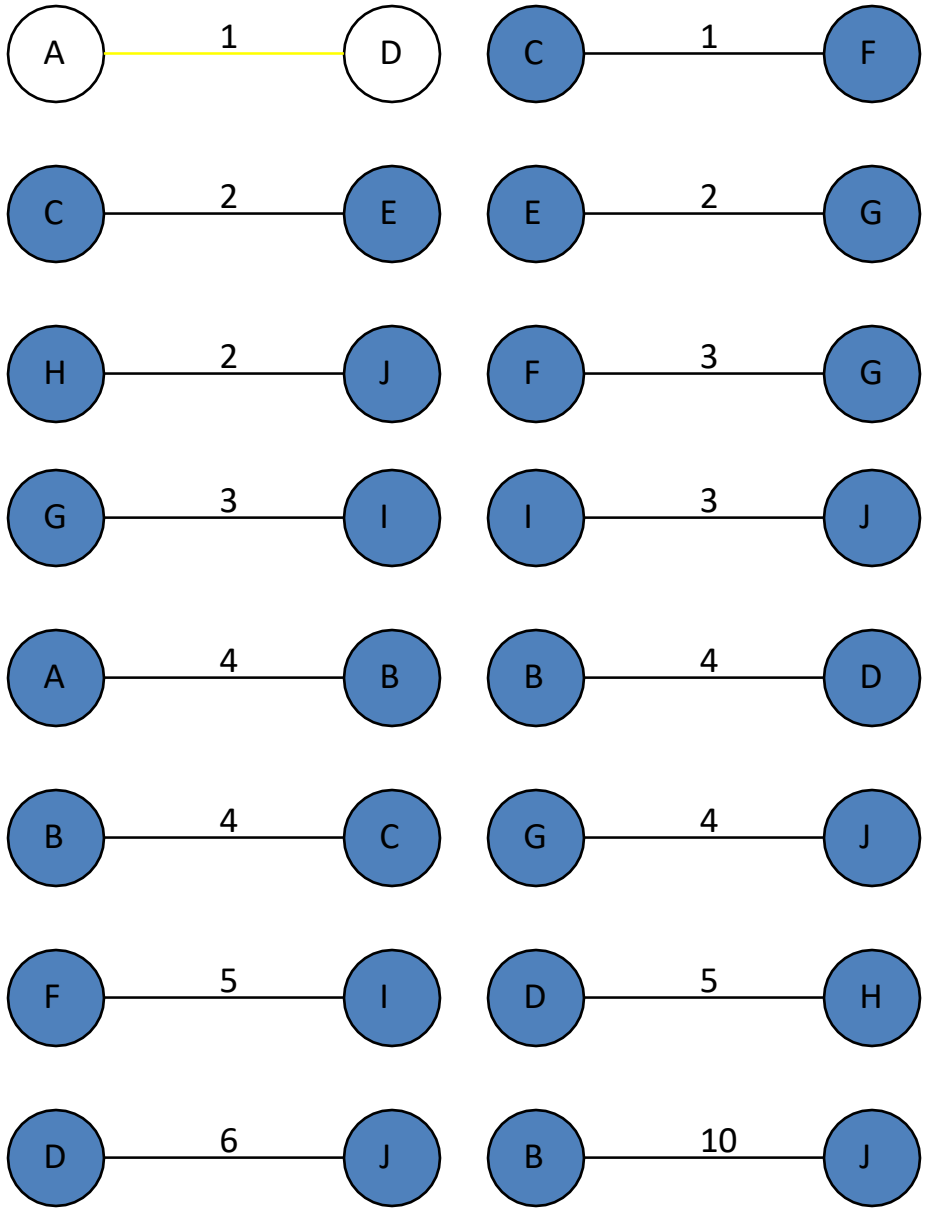
(in reality they are placed in a priority queue - not sorted - but sorting them makes the algorithm easier to visualize)



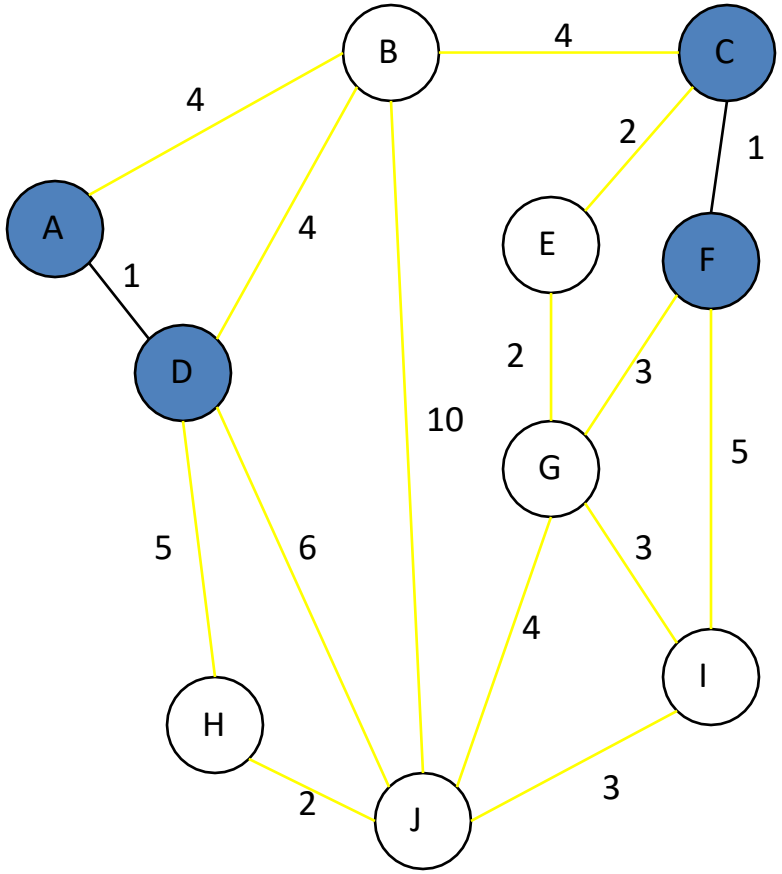


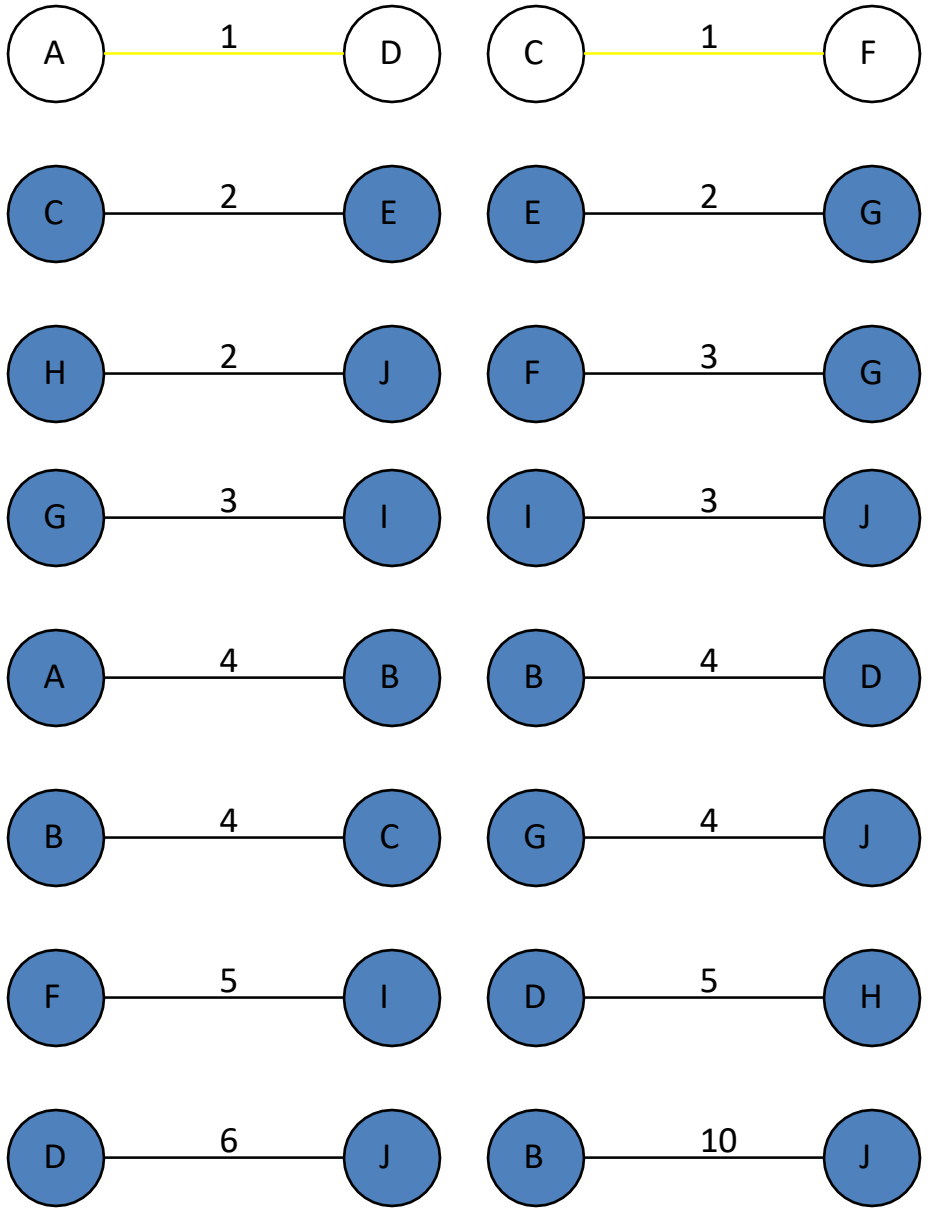
Add Edge


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

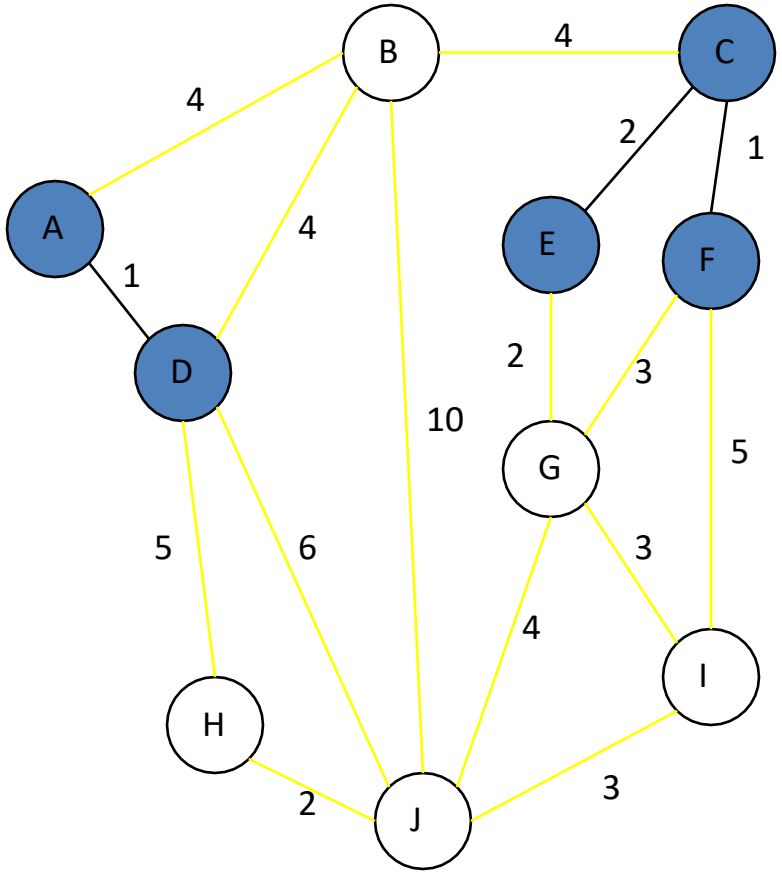

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

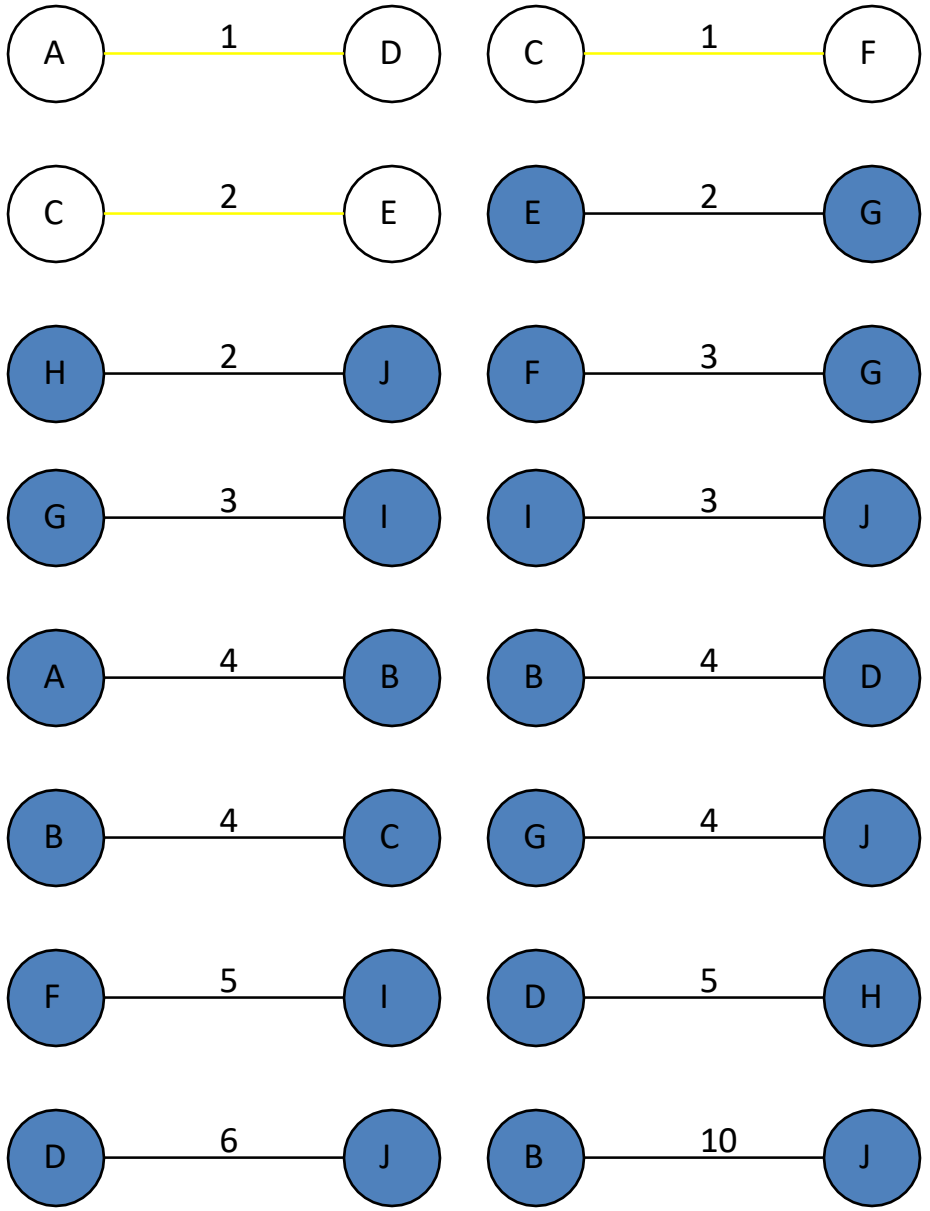
Add Edge


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

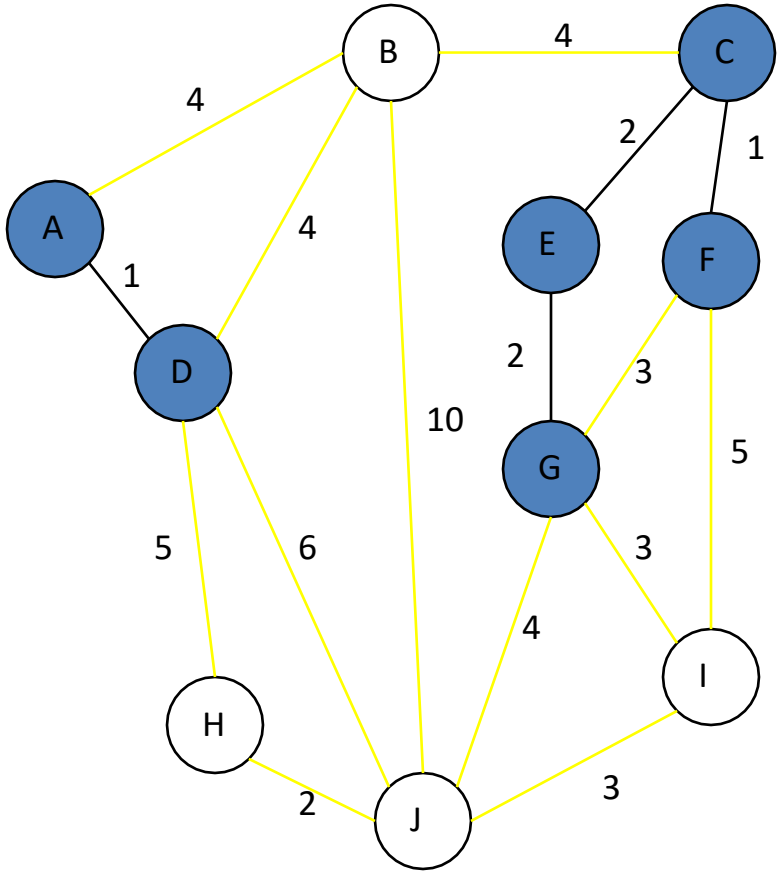

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

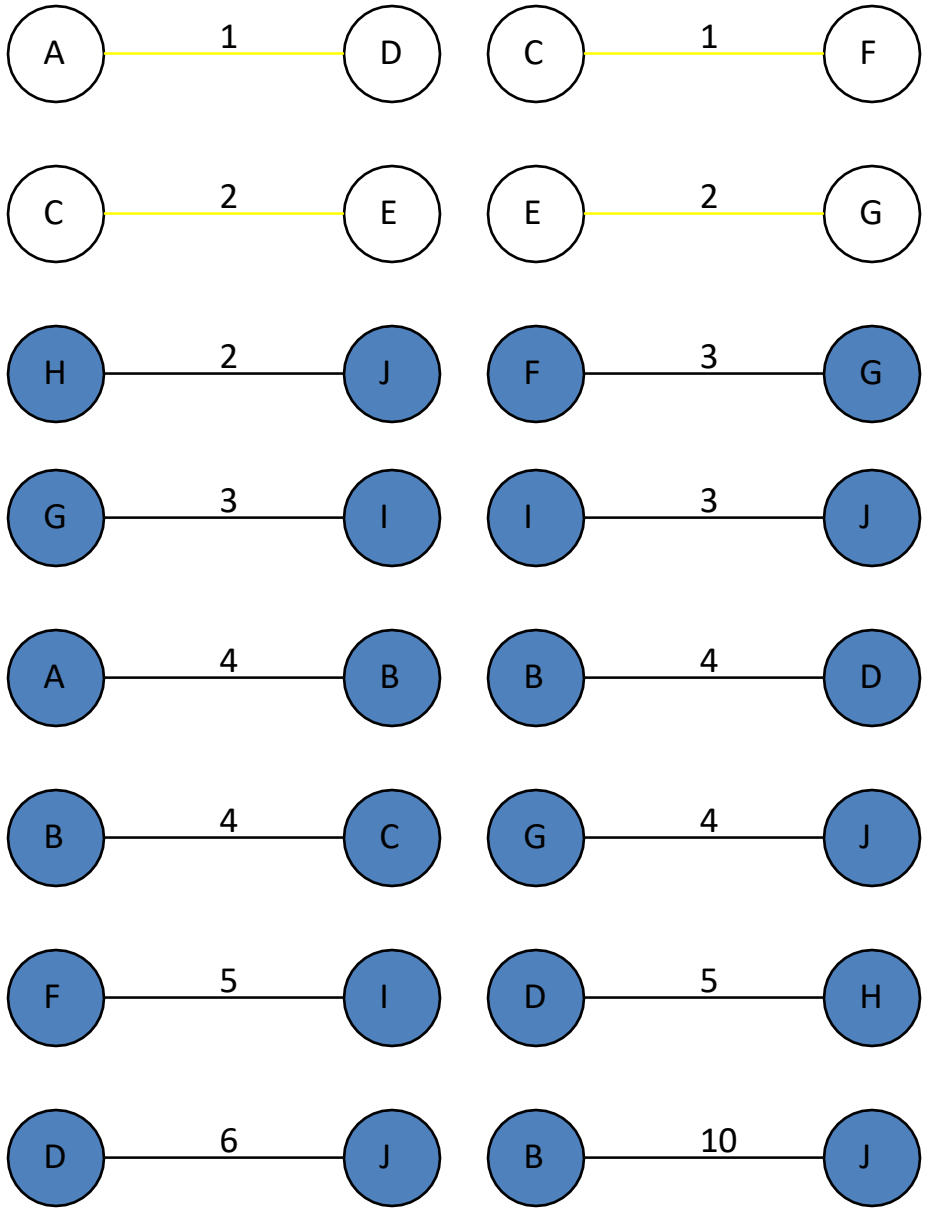
Add Edge


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

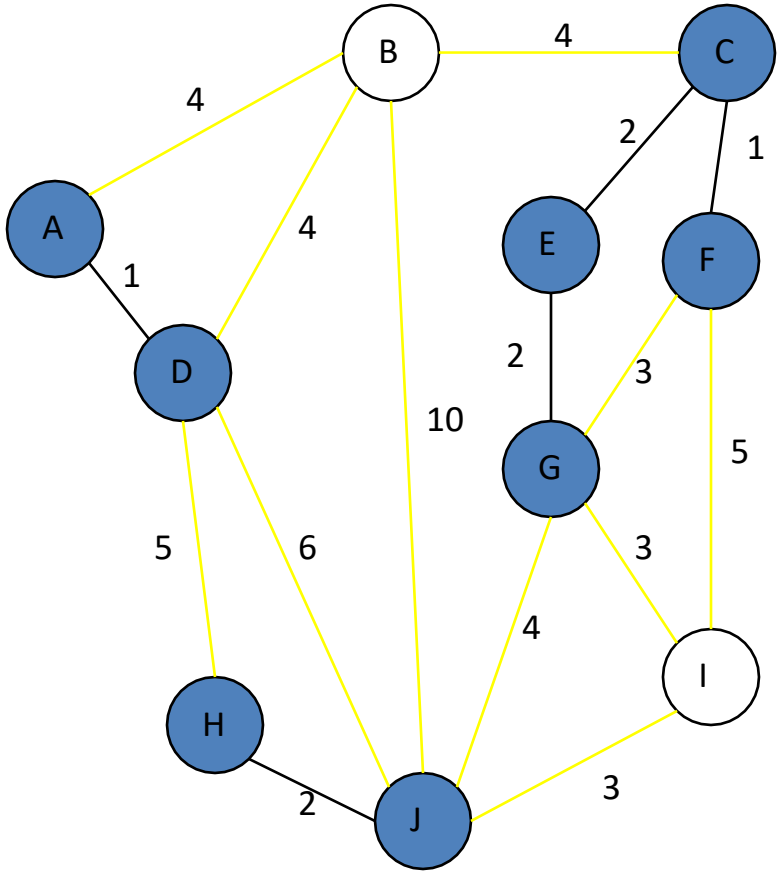

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

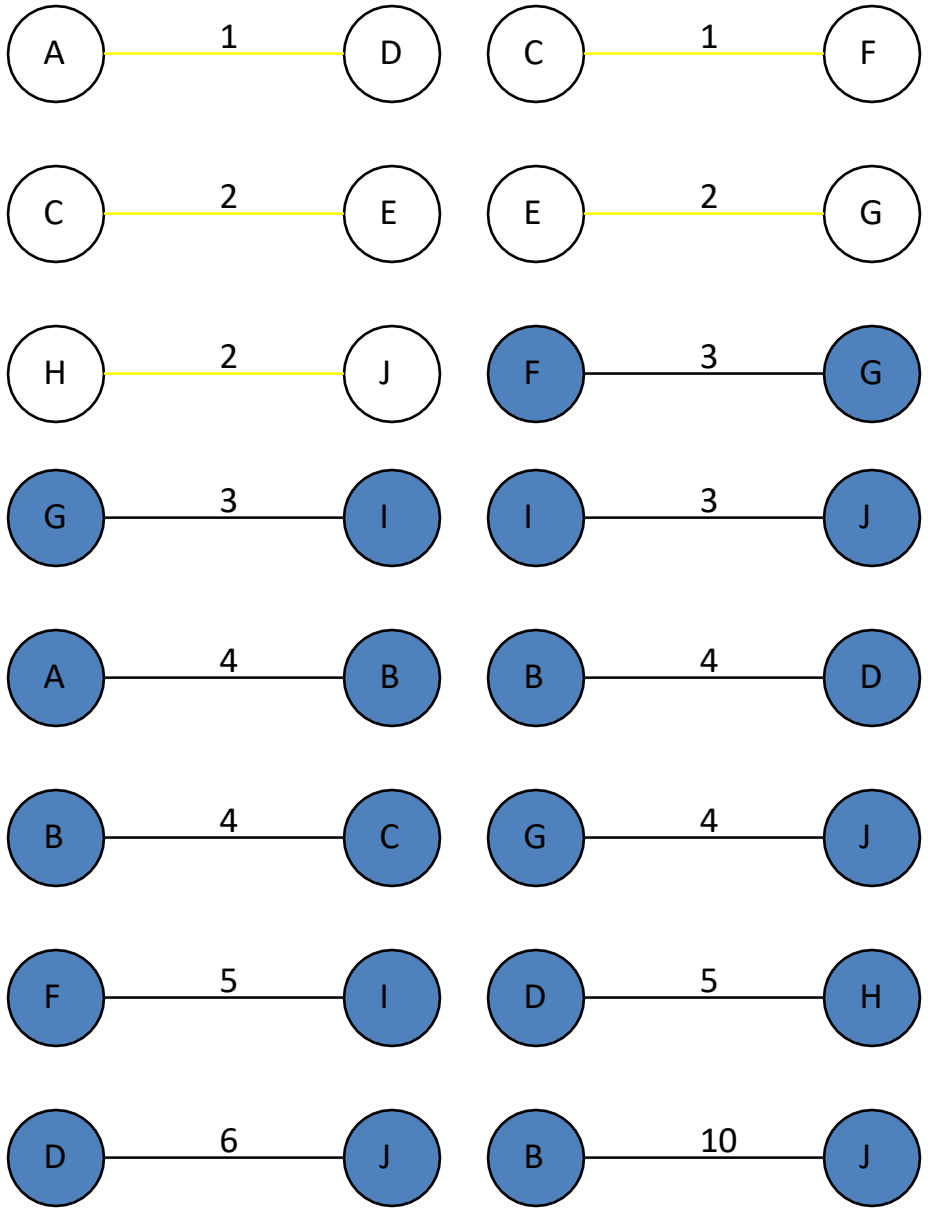
Add Edge


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

Add Edge

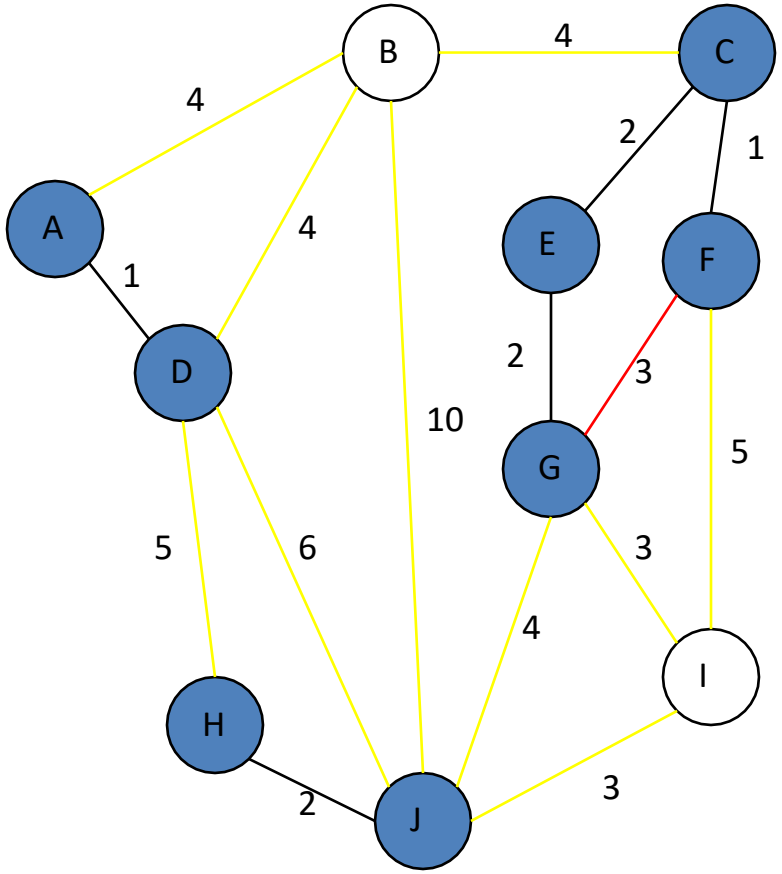

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
10
3
5
5
6
3
4
3
2
3

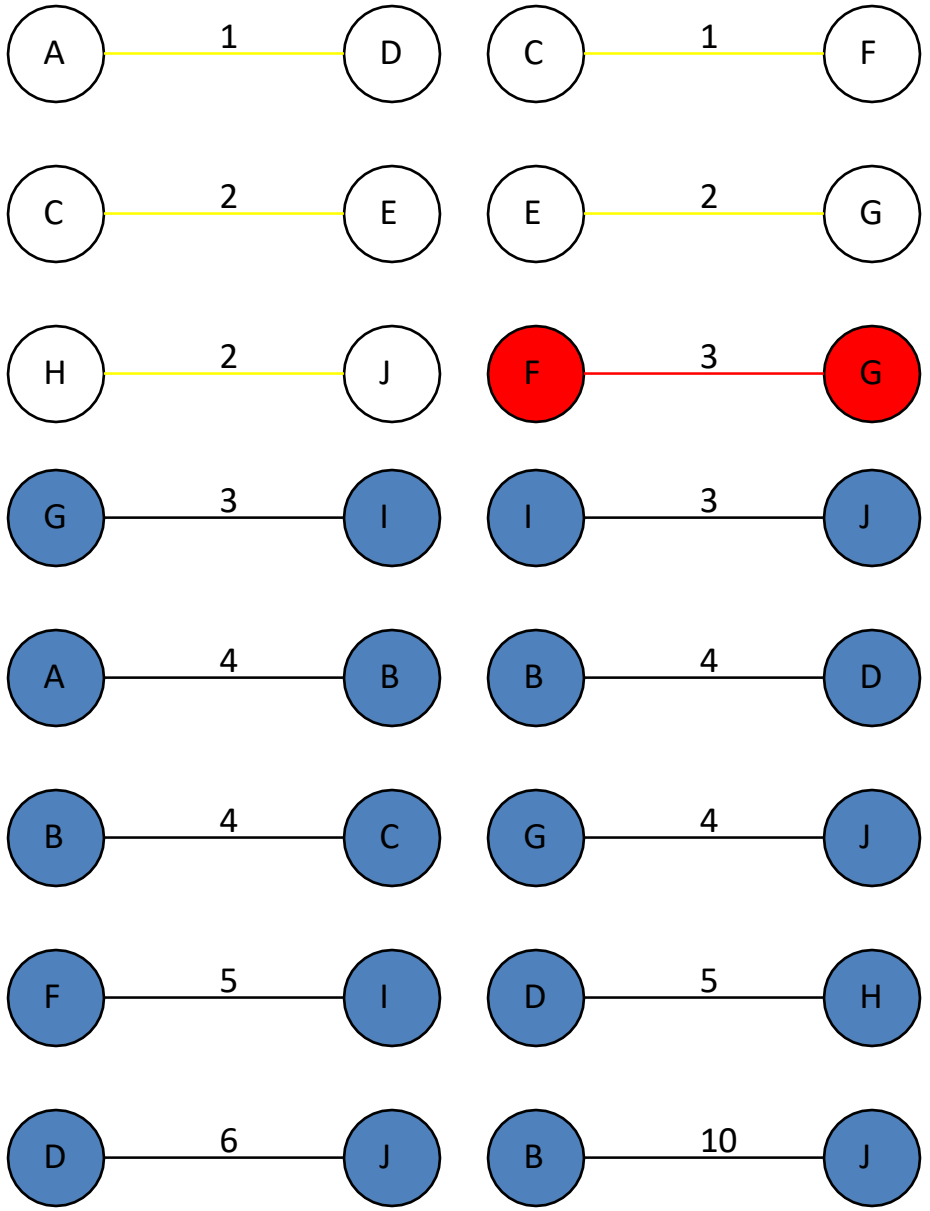

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Cycle

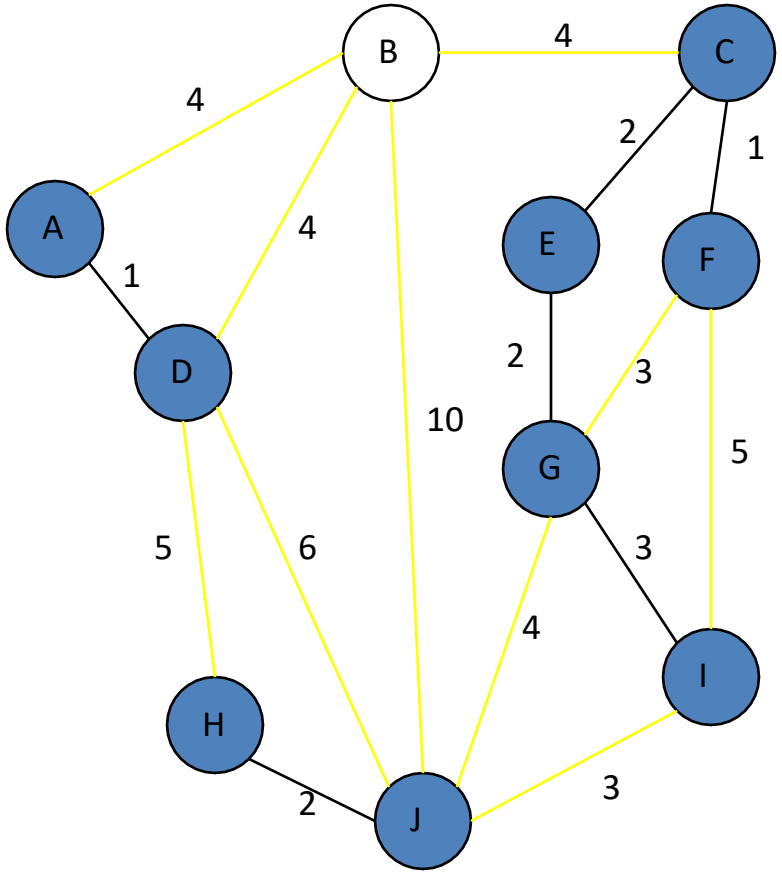
Don't Add

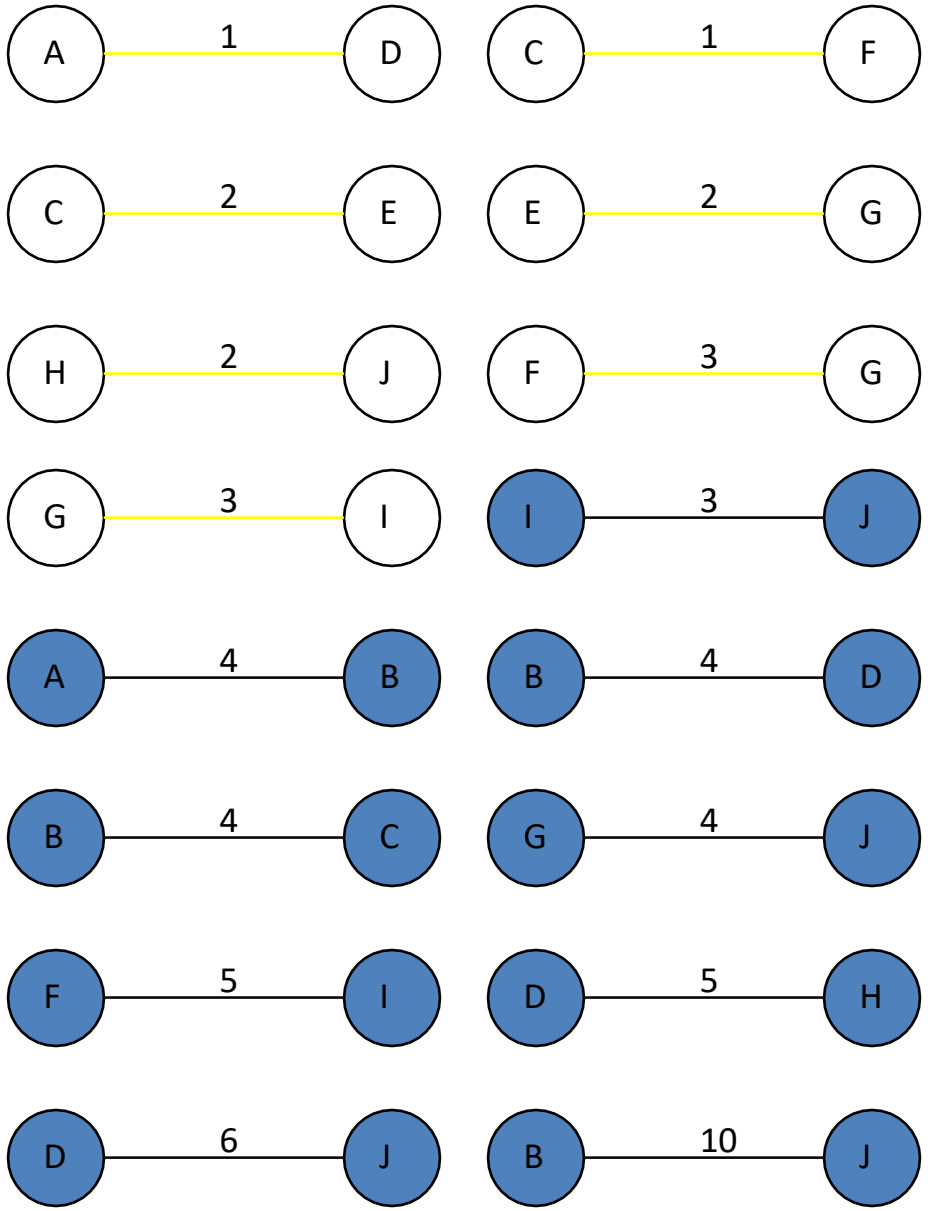
Edge



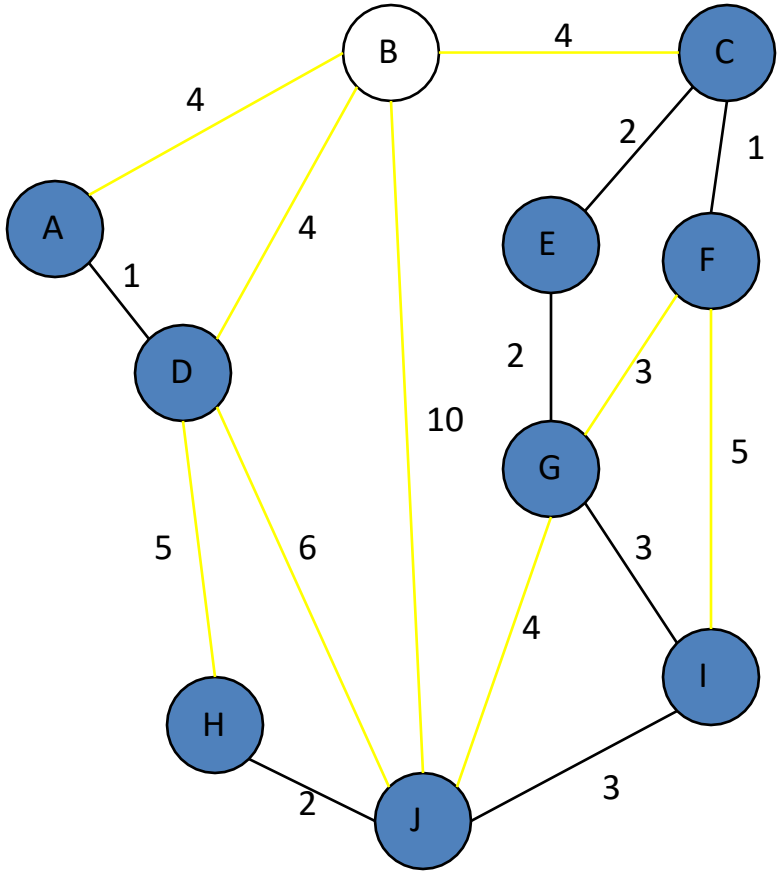


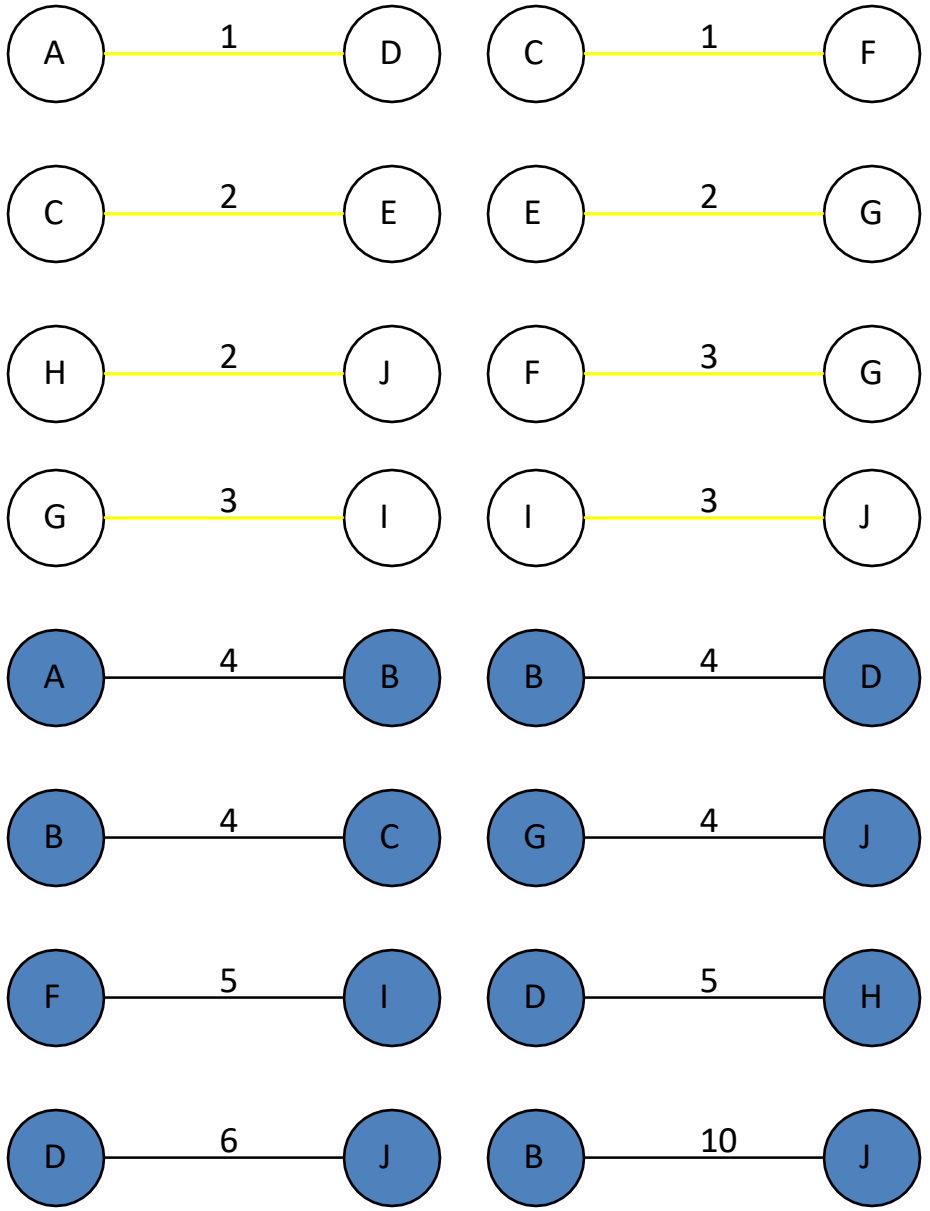
Add Edge


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

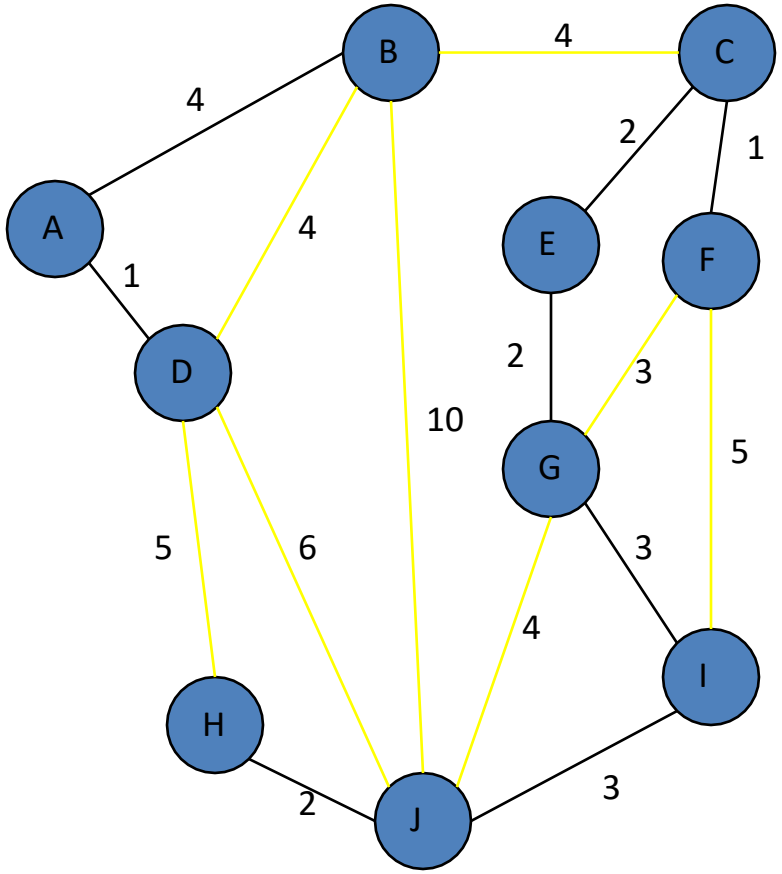

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

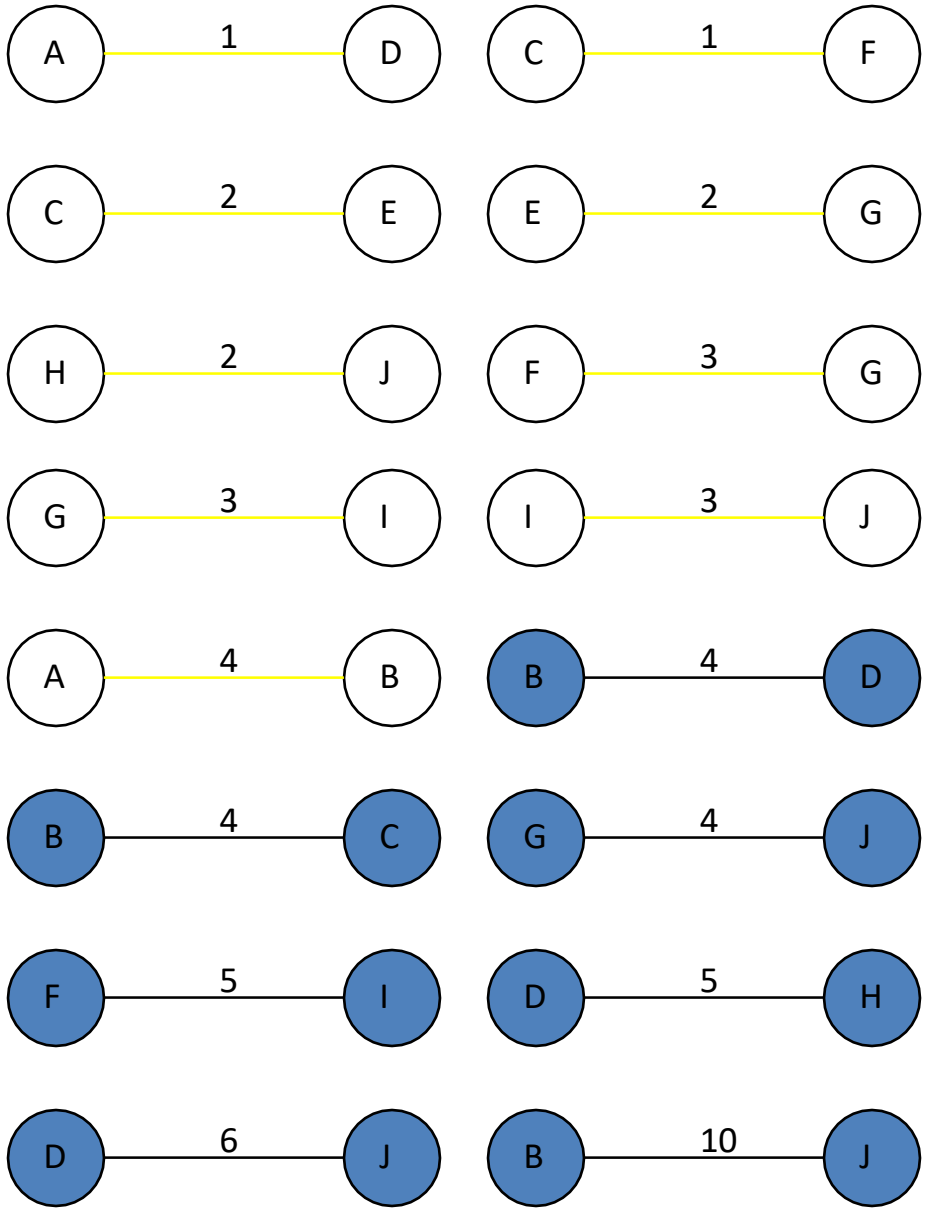
Add Edge


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

Add Edge

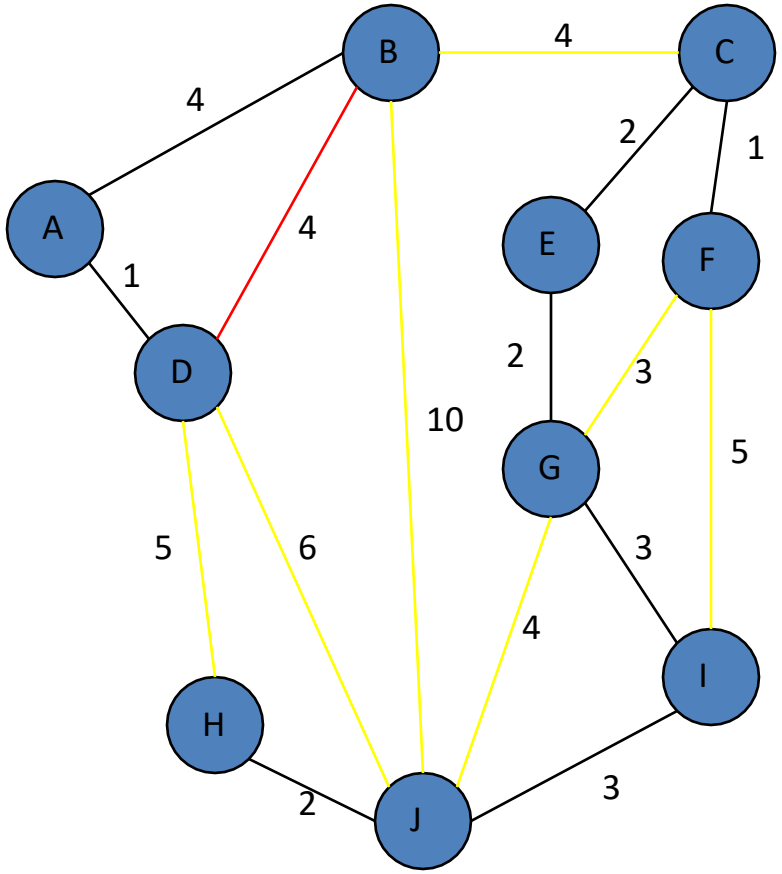

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3

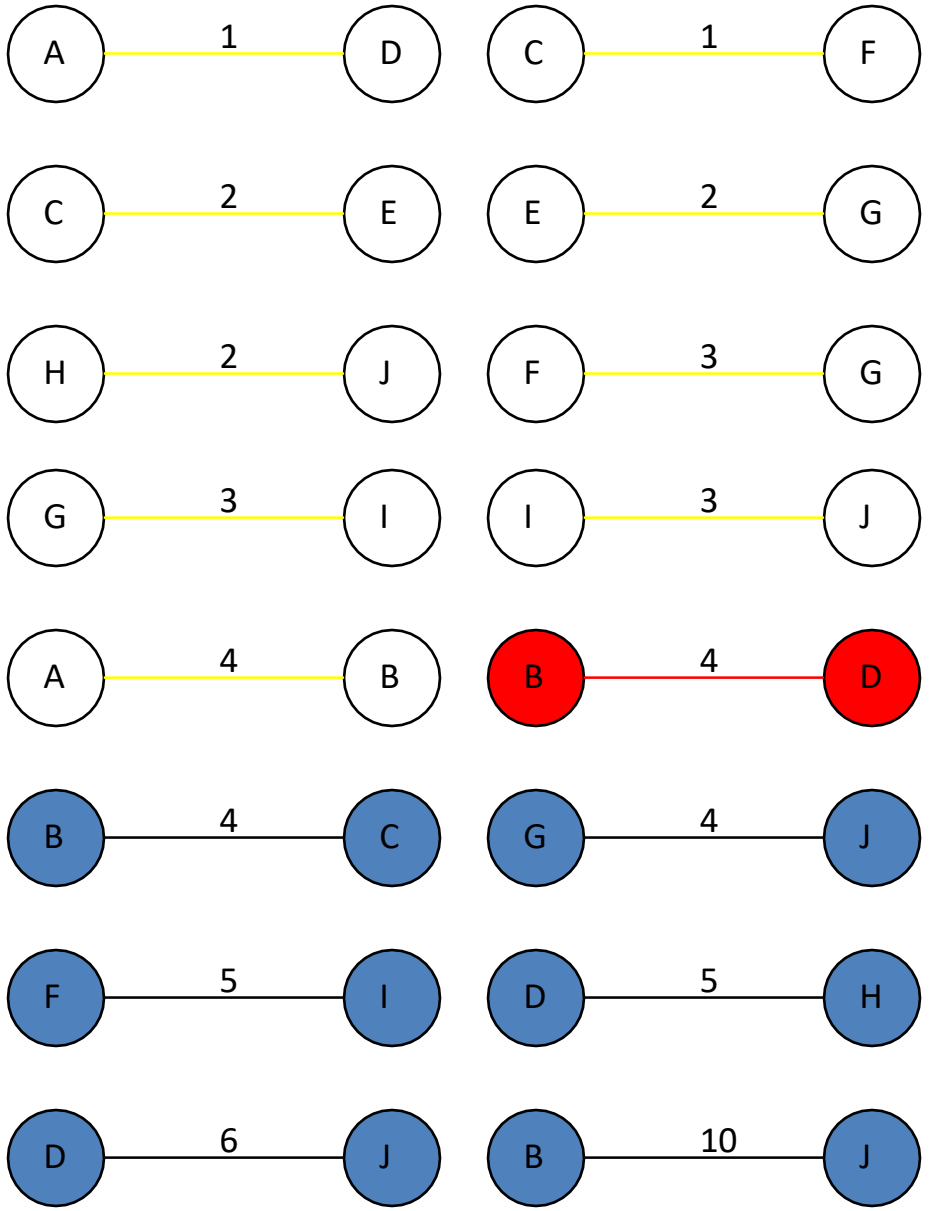

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Cycle

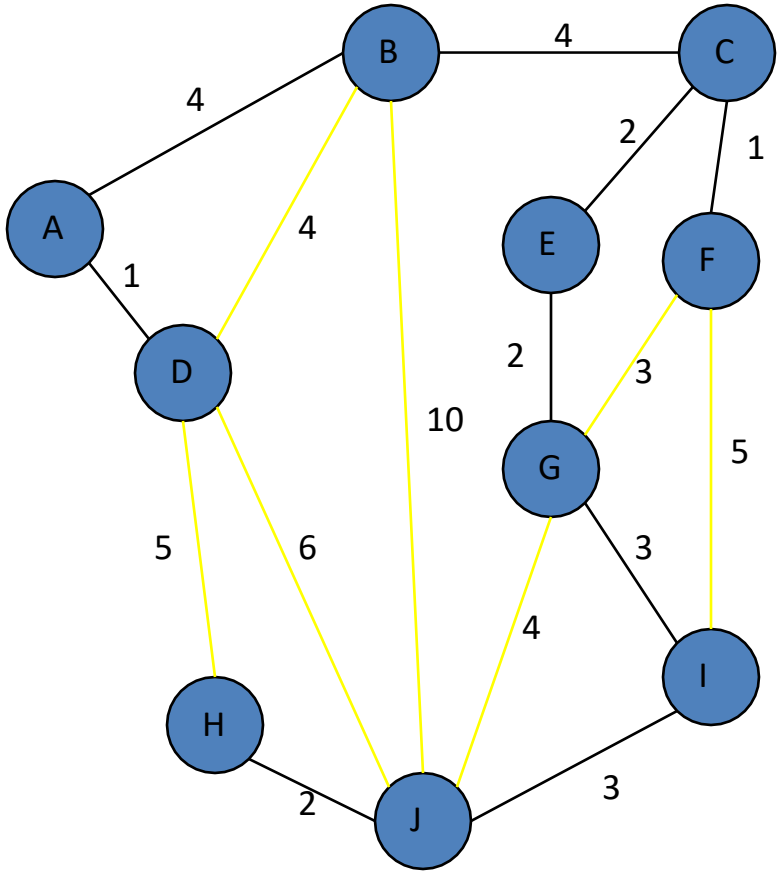
Don't Add

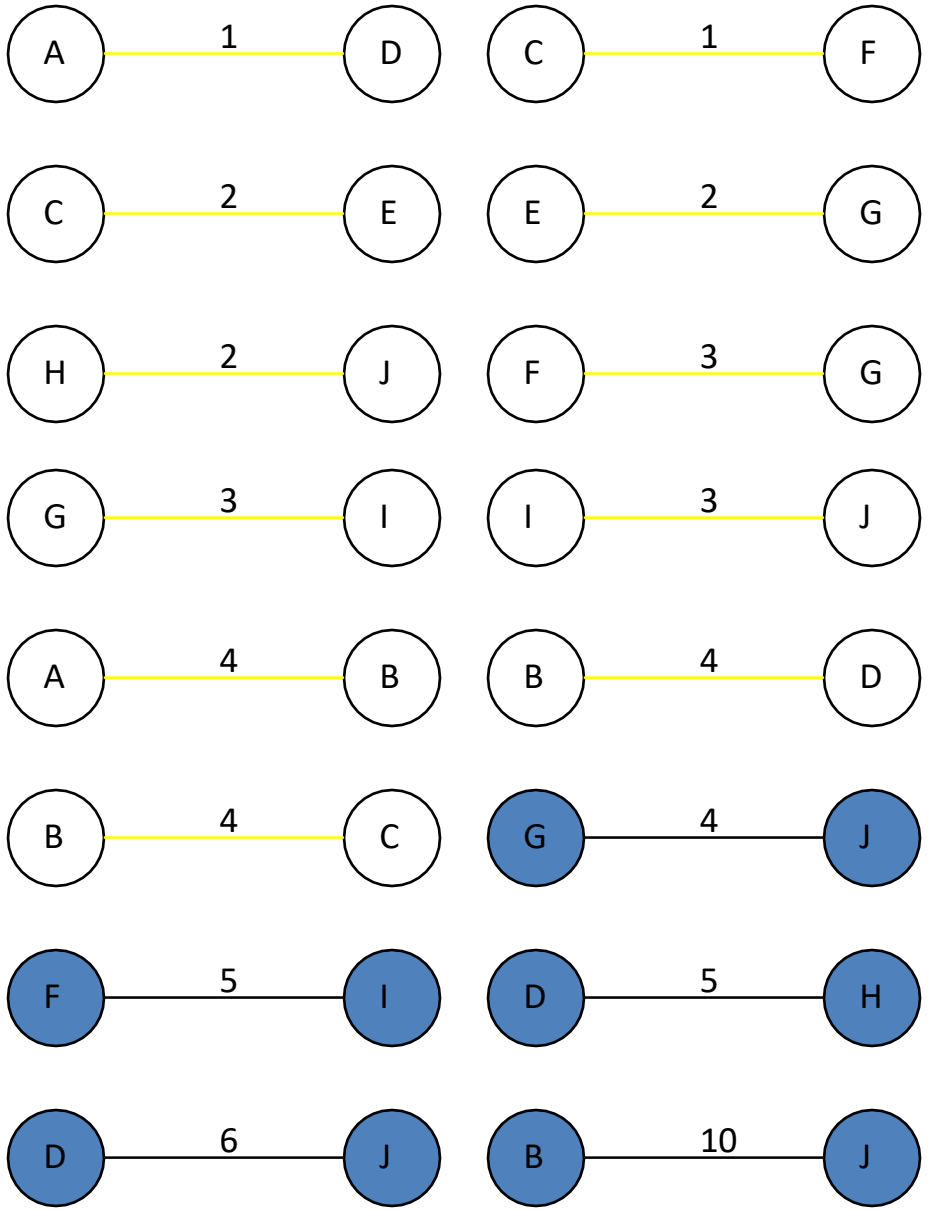
Edge



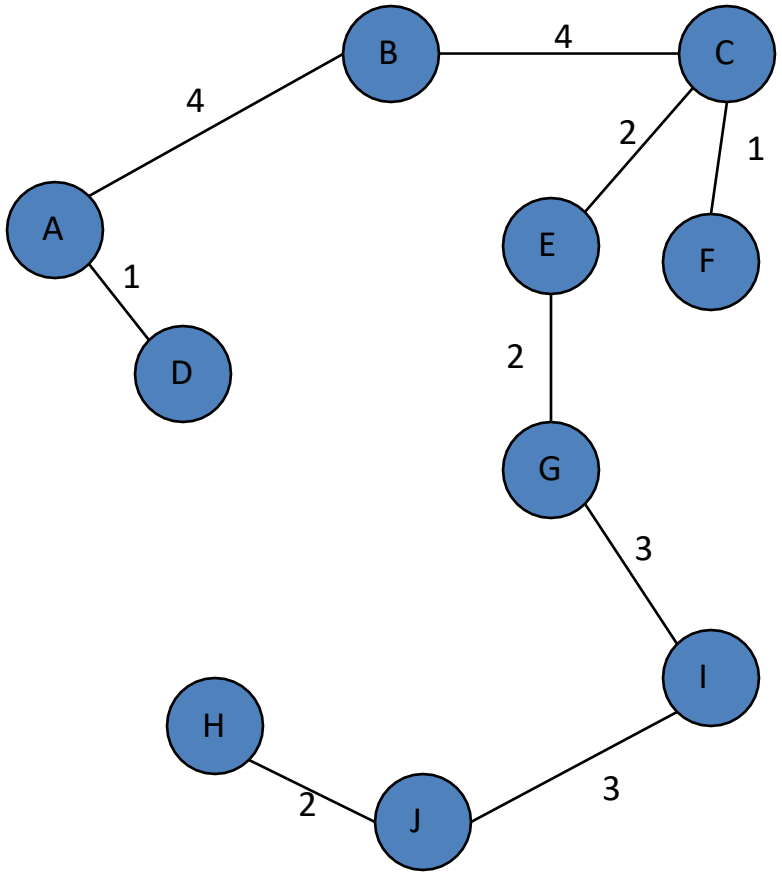


Add Edge


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