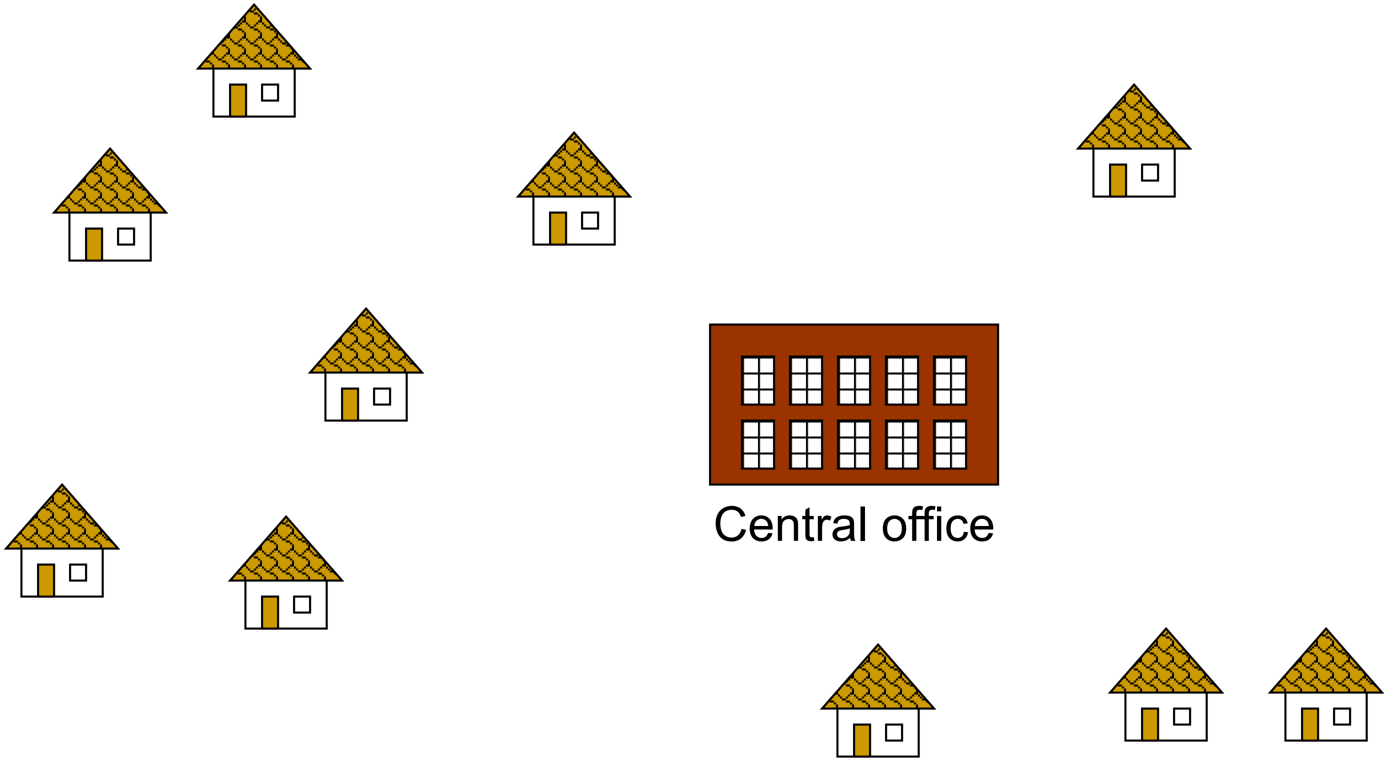
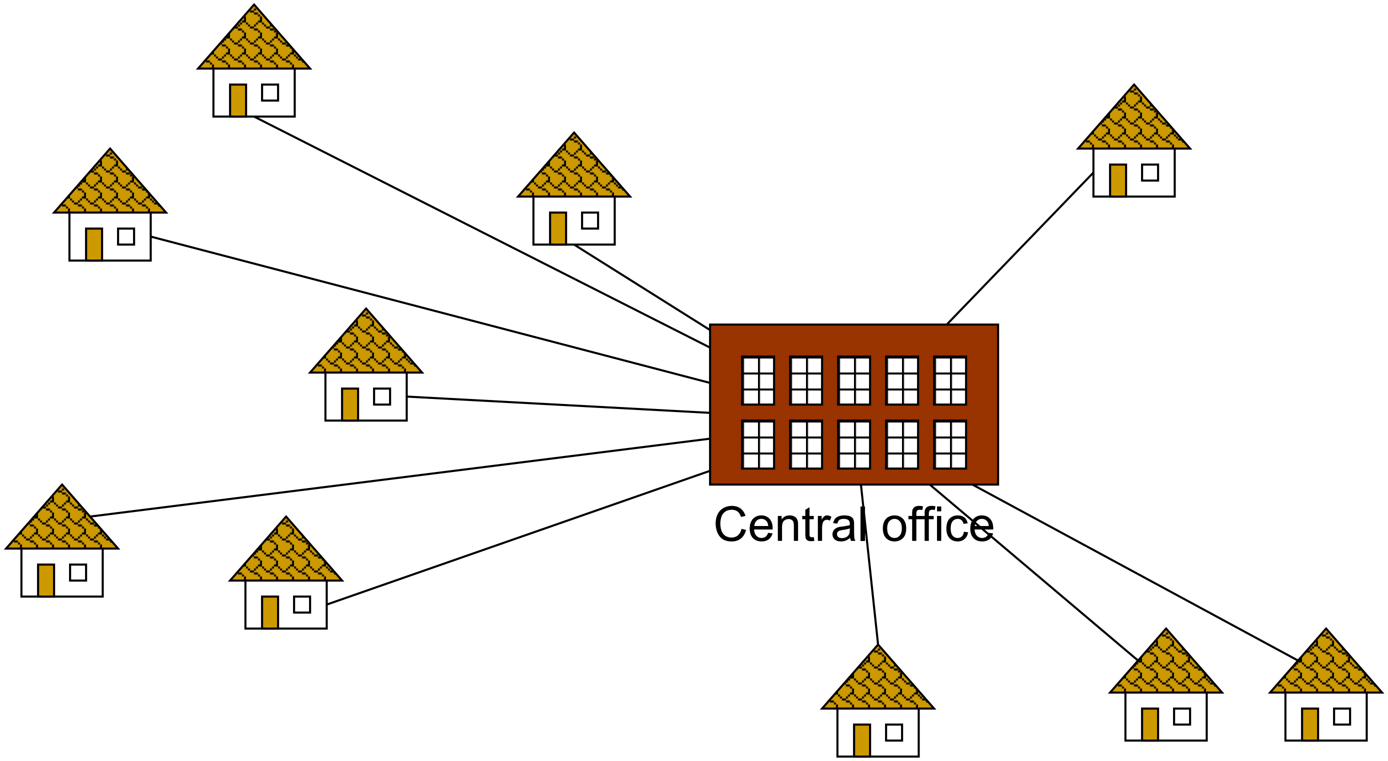
Minimum Spanning Tree in Graph

Slides by Si Dong, M.T. Goodrich and R. Tamassia

Problem: Laying Telephone Wire

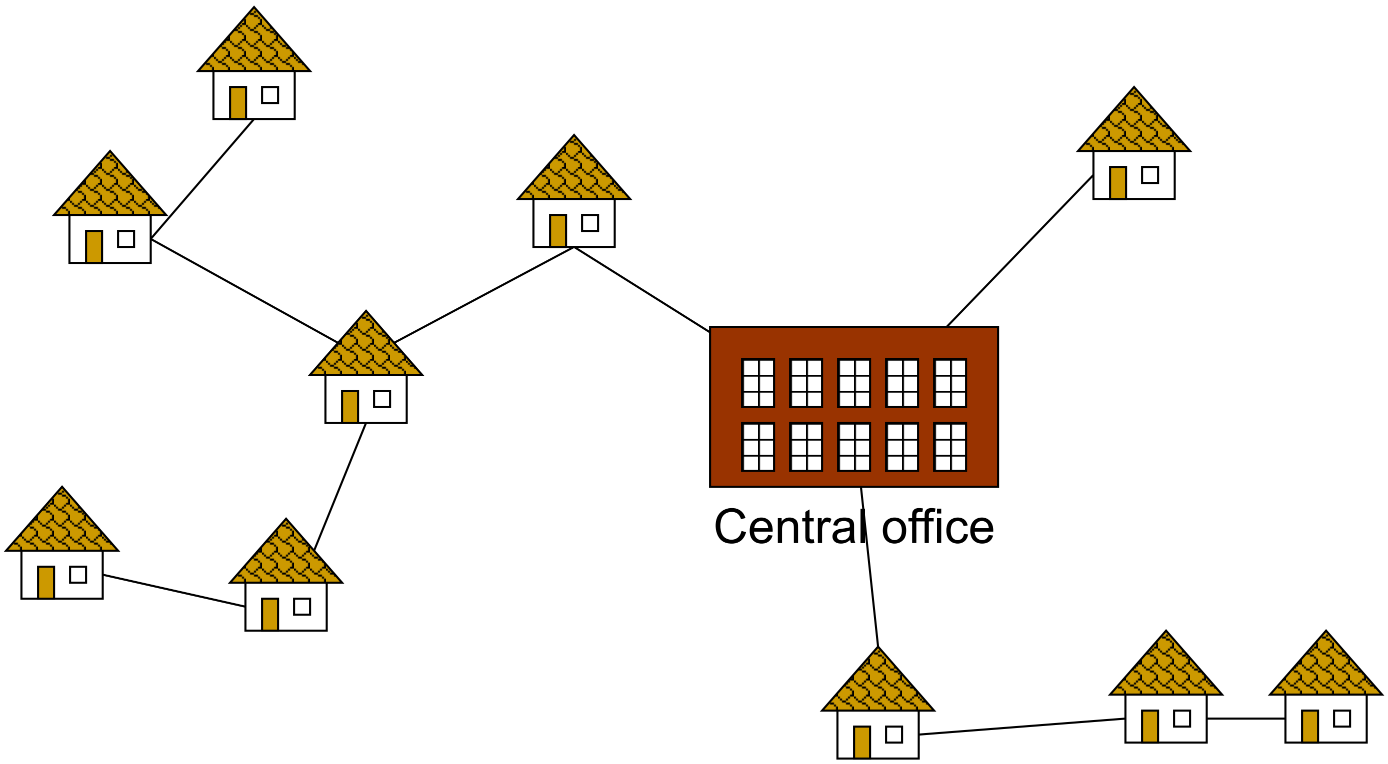


Wiring: Naive Approach



Expensive!

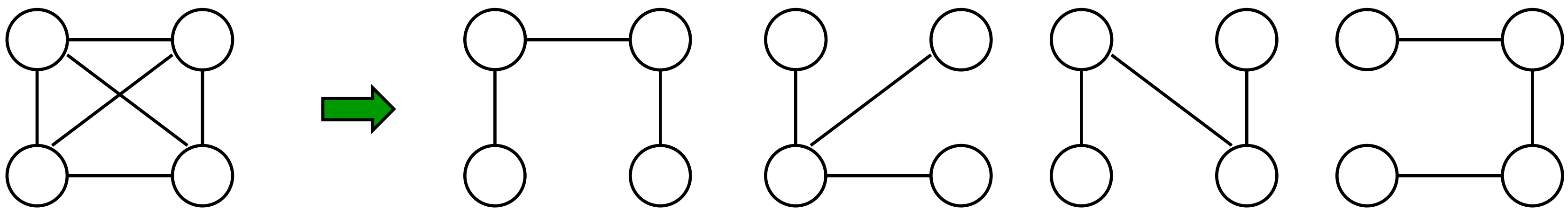
Wiring: Better Approach



Minimize the total length of wire connecting **ALL** customers

Spanning trees

- Suppose you have a **connected undirected** graph:
  - Connected: every node is reachable from every other node
  - Undirected: edges do not have an associated direction
- ...then a spanning tree of the graph is a connected subgraph which contains all the vertices and has no cycles.



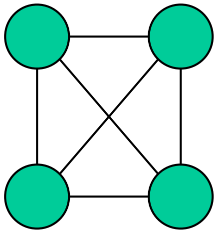
A connected, undirected graph

Four of the spanning trees of the graph

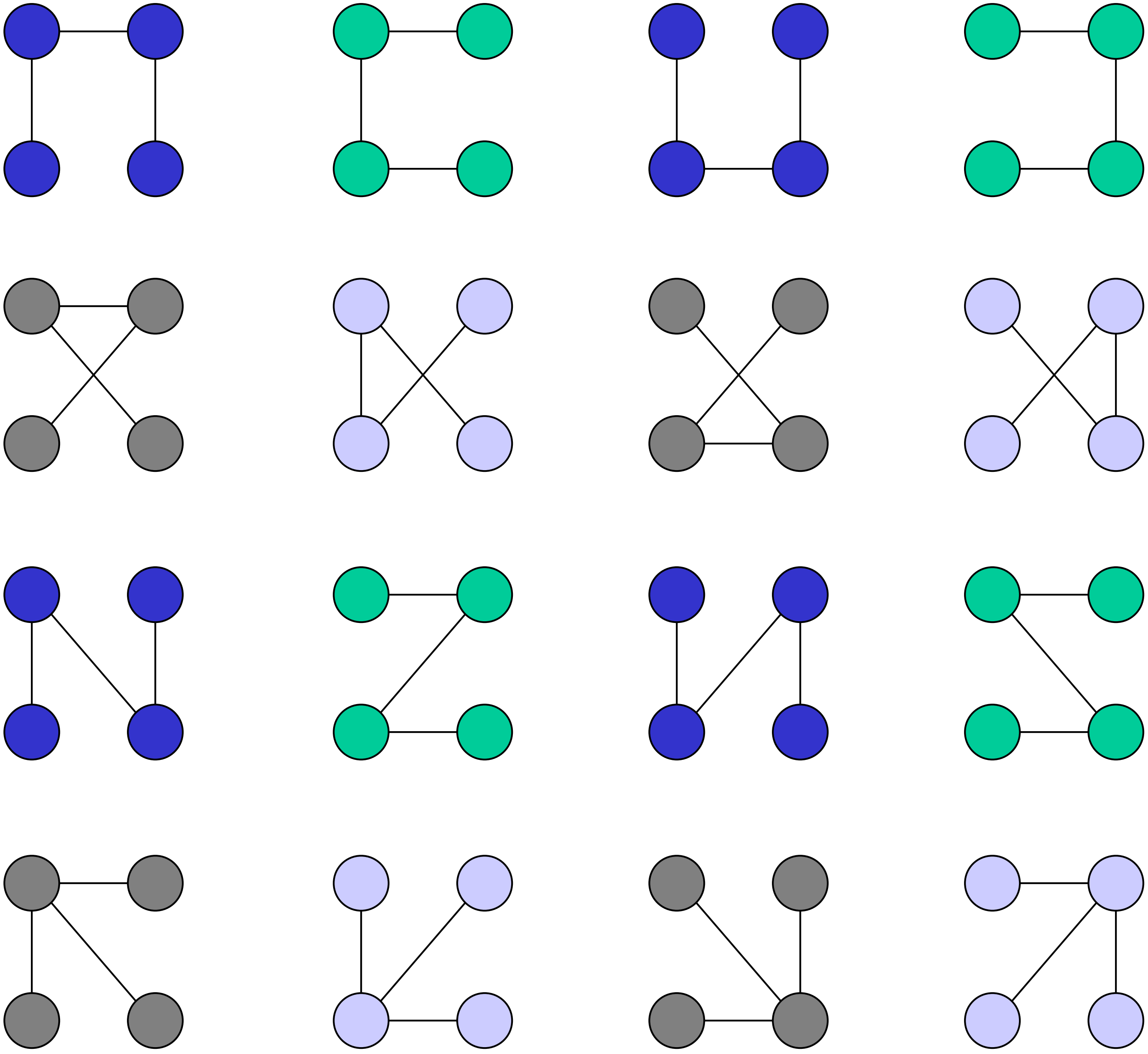
Spanning trees

- Every spanning tree has n-1 edges.
- Can be shown by induction: use the fact that every tree has a vertex with degree 1.

Complete Graph

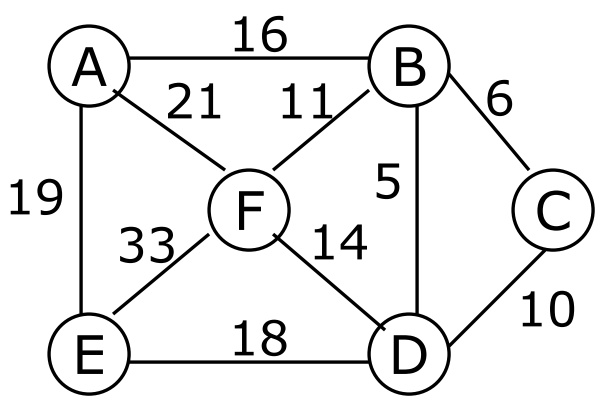


All 16 of its Spanning Trees

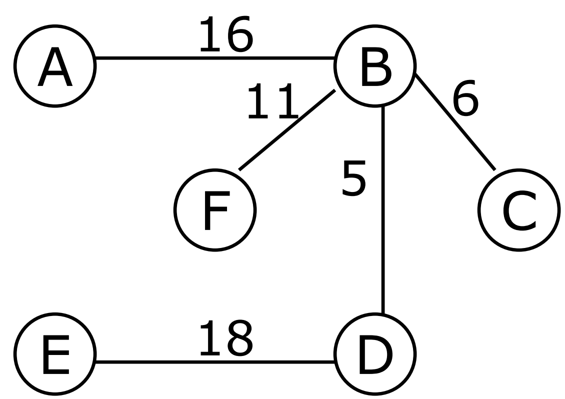


Minimum-cost spanning trees

- Suppose you have a connected undirected graph with a weight (or cost) associated with each edge.
- The cost of a spanning tree would be the sum of the costs of its edges.
- A minimum-cost spanning tree is a spanning tree that has the lowest cost.



A connected, undirected graph



A minimum-cost spanning tree

Minimum Spanning Tree (MST)

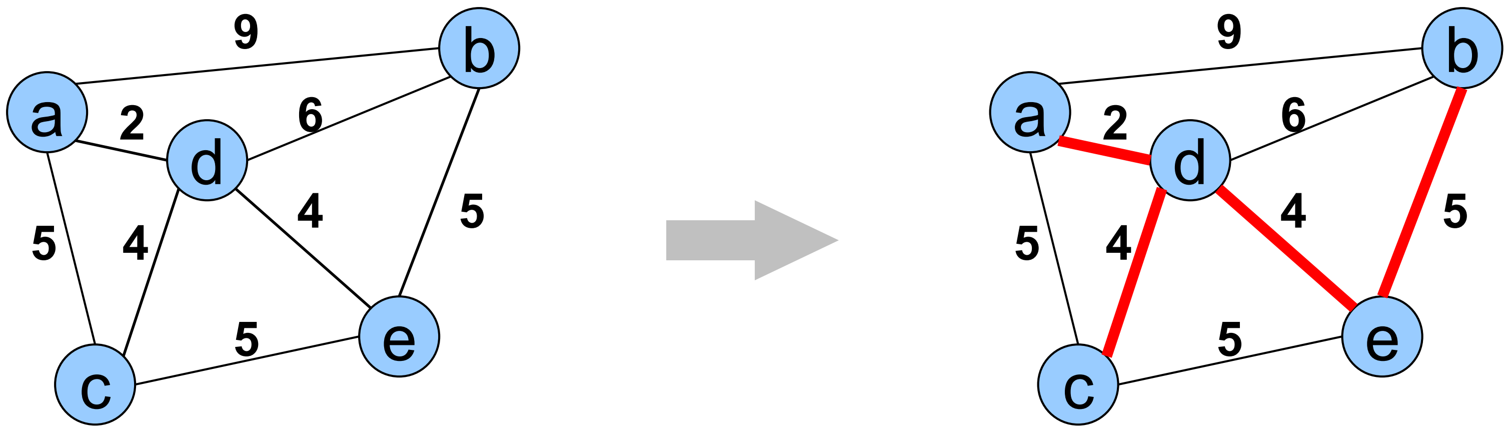
A **minimum spanning tree** is a subgraph of an undirected weighted graph ***G***, such that

- it is a tree (i.e., it is acyclic)

Tree = connected graph without cycles

- it covers all the vertices ***V***
  - contains $|V| - 1$ edges
- the total cost associated with tree edges is the minimum among all possible spanning trees
- not necessarily unique.

How Can We Generate a MST?



Finding minimum spanning trees

- **Kruskal's algorithm**

- **Idea:** consider edges in the order of cheapest edge first.

- Choose an edge unless it forms a cycle with the previous chosen edges.

Kruskal's algorithm

1. Sort edges in increasing order of cost:

$$e_1, e_2, \dots, e_m$$

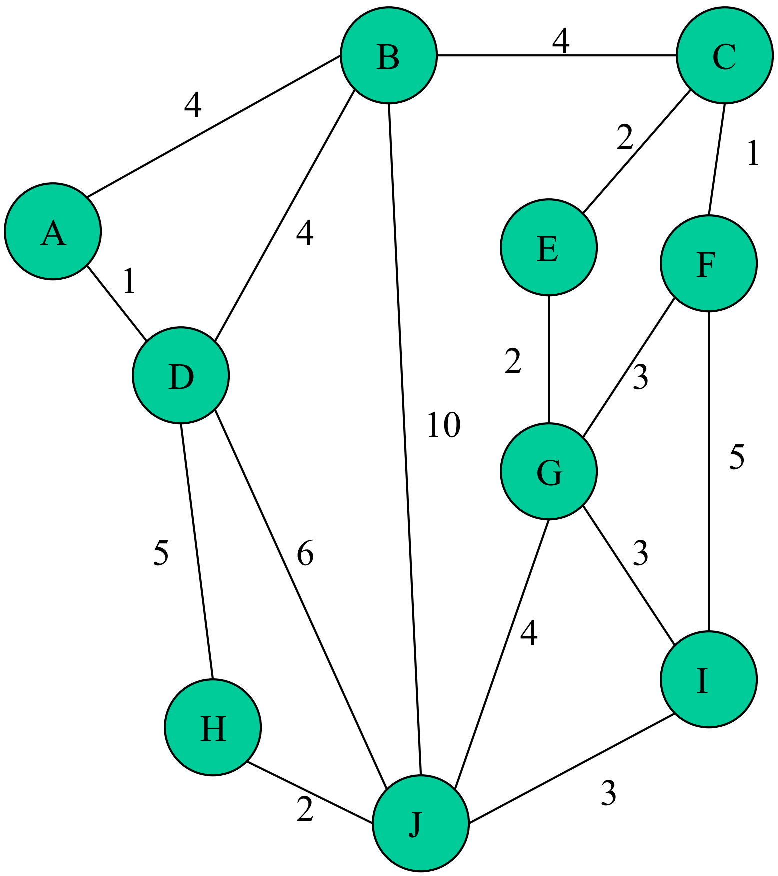
2. Maintain a forest F initialized to $\{v_1, \dots, v_n\}$ with no edges.

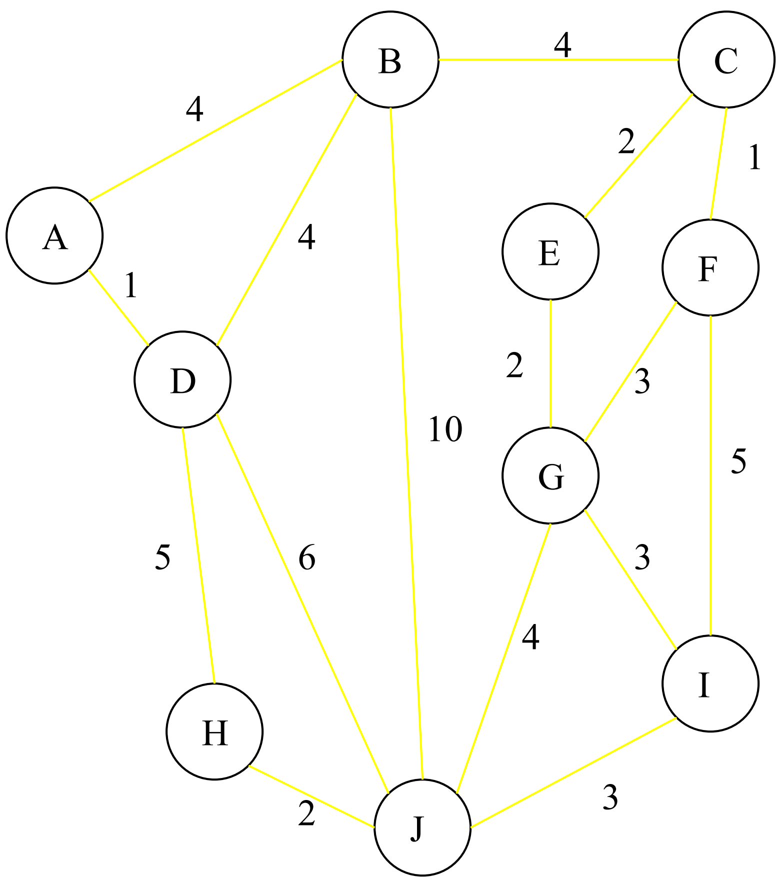
3. For i=1…m

    if adding e_i to F does not create a cycle

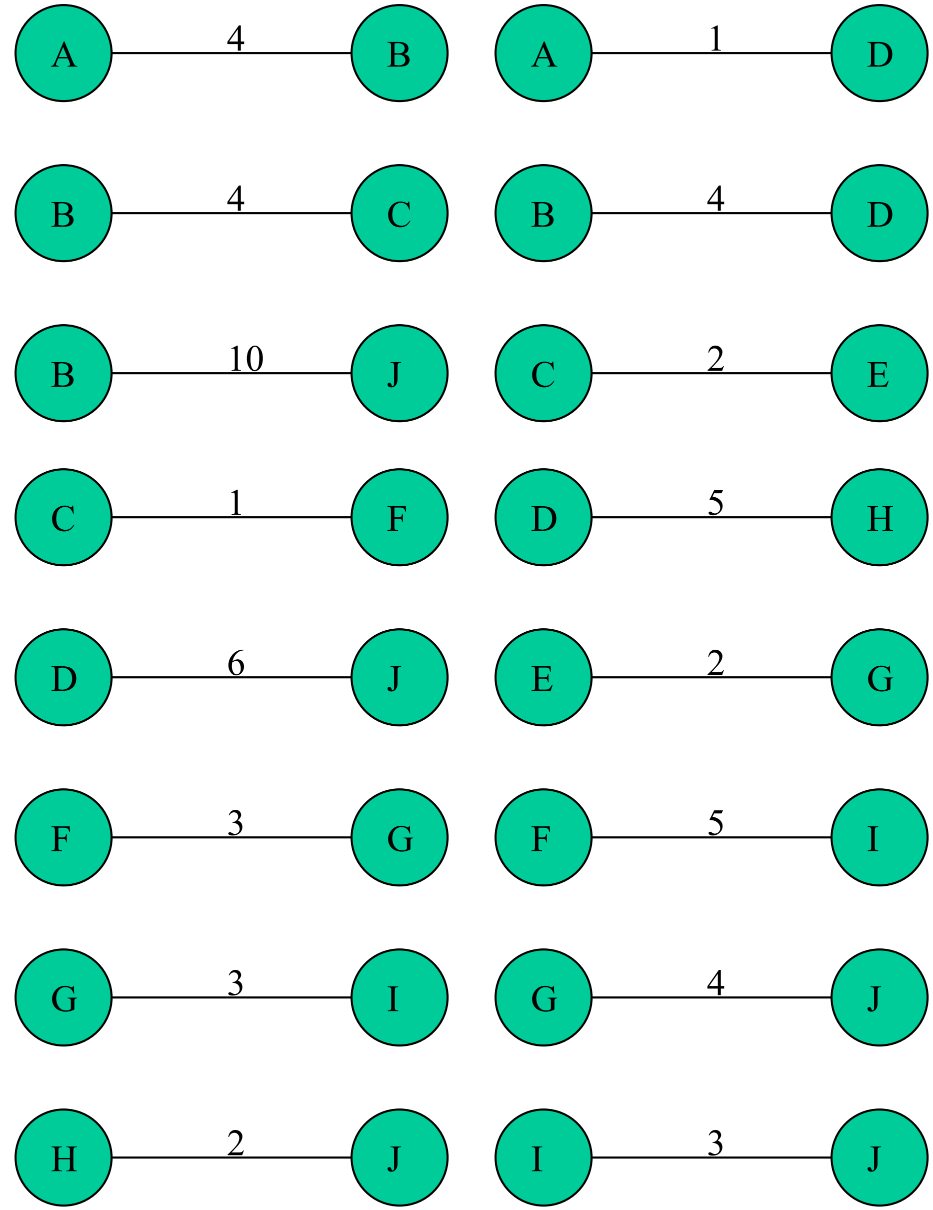
    $$F \leftarrow F \cup \{e_i\}$$

Complete Graph



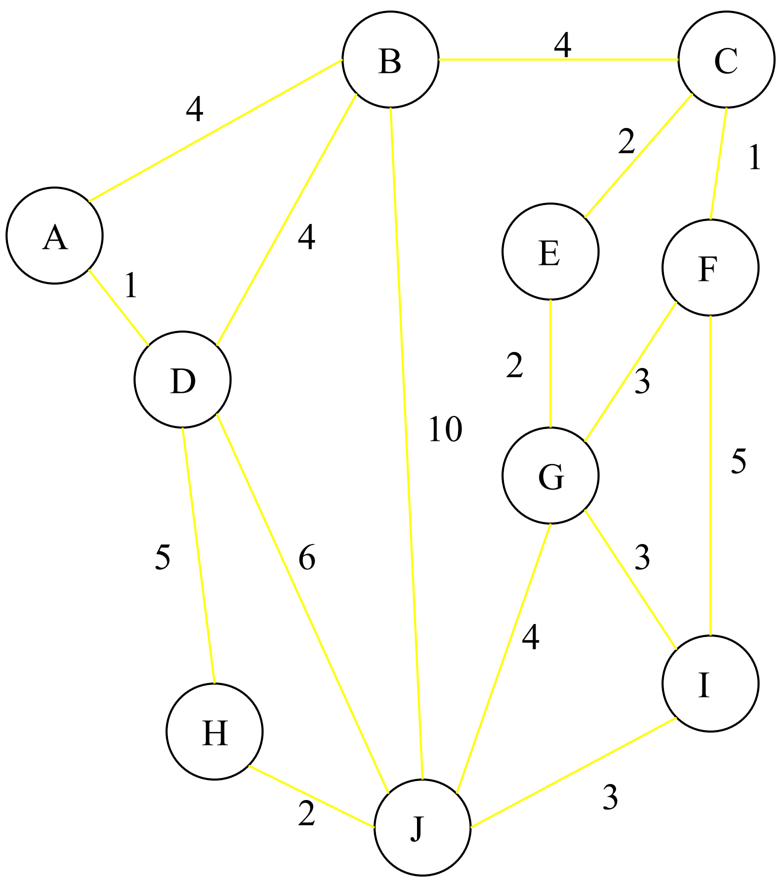


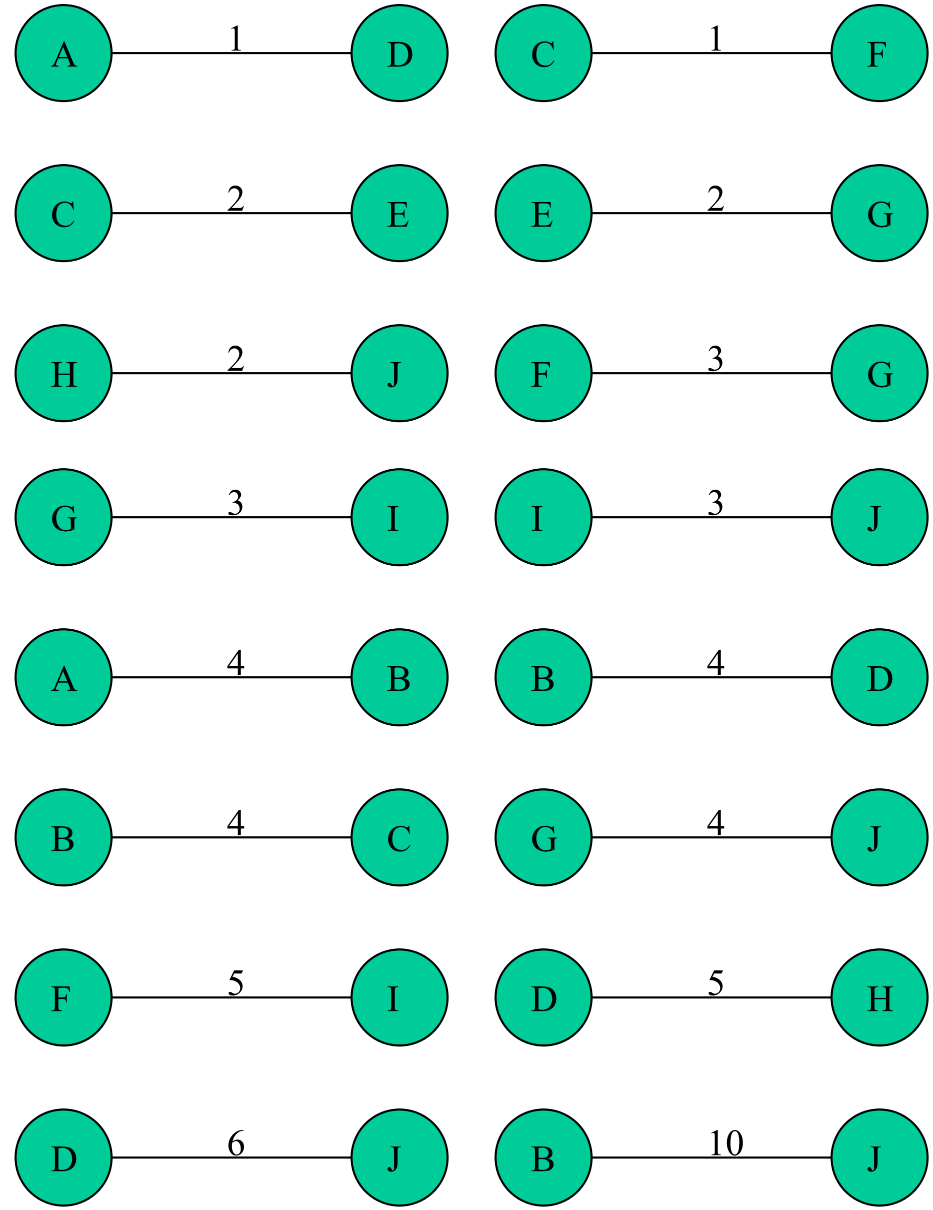
B
4
C
4
2
1
A
4
E
F
1
D
2
3
10
G
5
5
6
3
4
I
H
J
2
3



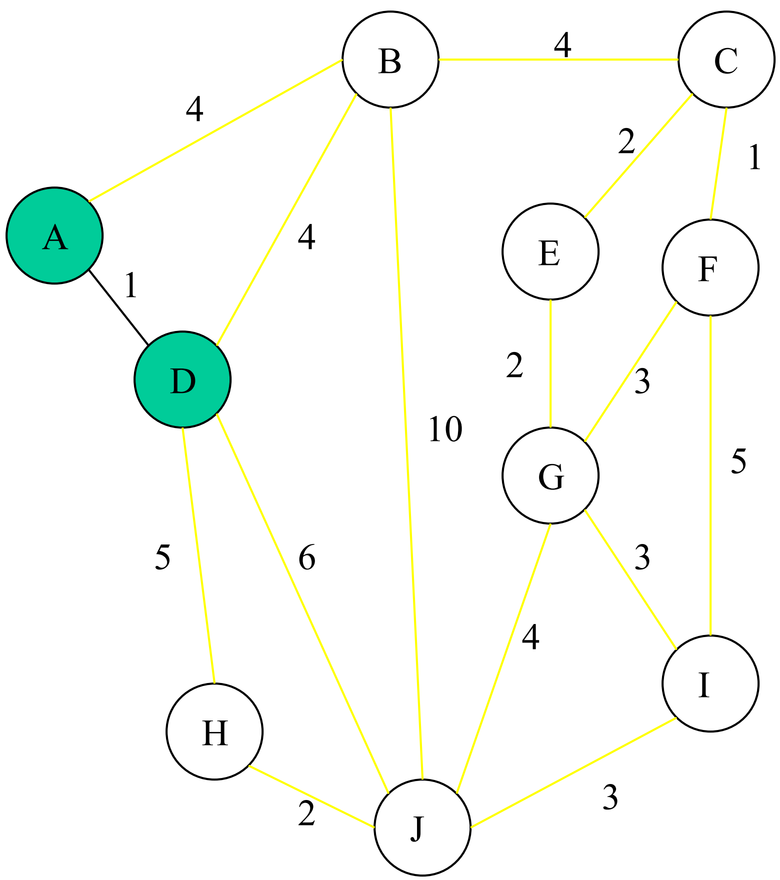
A 4 B
A 1 D
B 4 C
B 4 D
B 10 J
C 2 E
C 1 F
D 5 H
D 6 J
E 2 G
F 3 G
F 5 I
G 3 I
G 4 J
H 2 J
I 3 J

Sort Edges

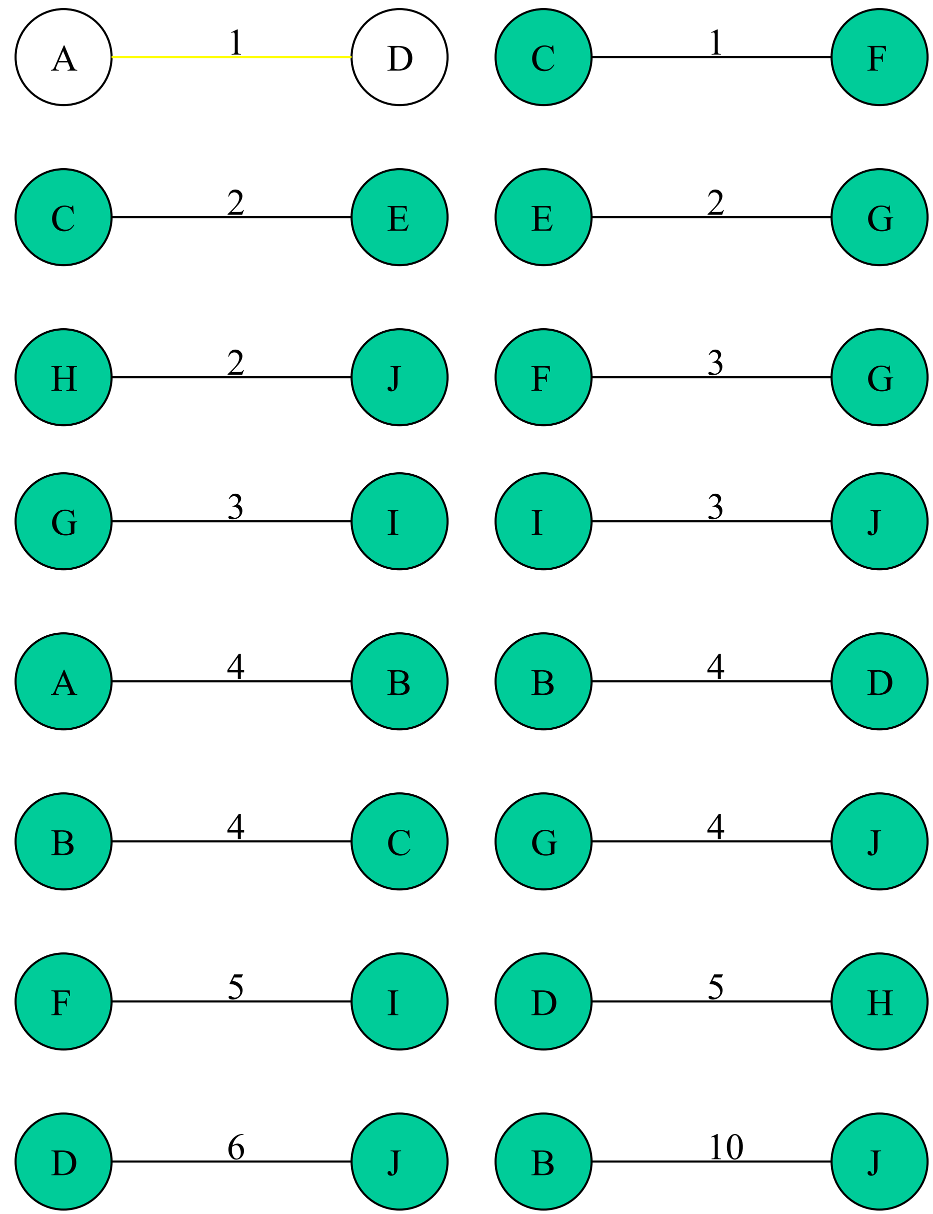

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
10
2
3
5
5
6
4
3
2
3


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

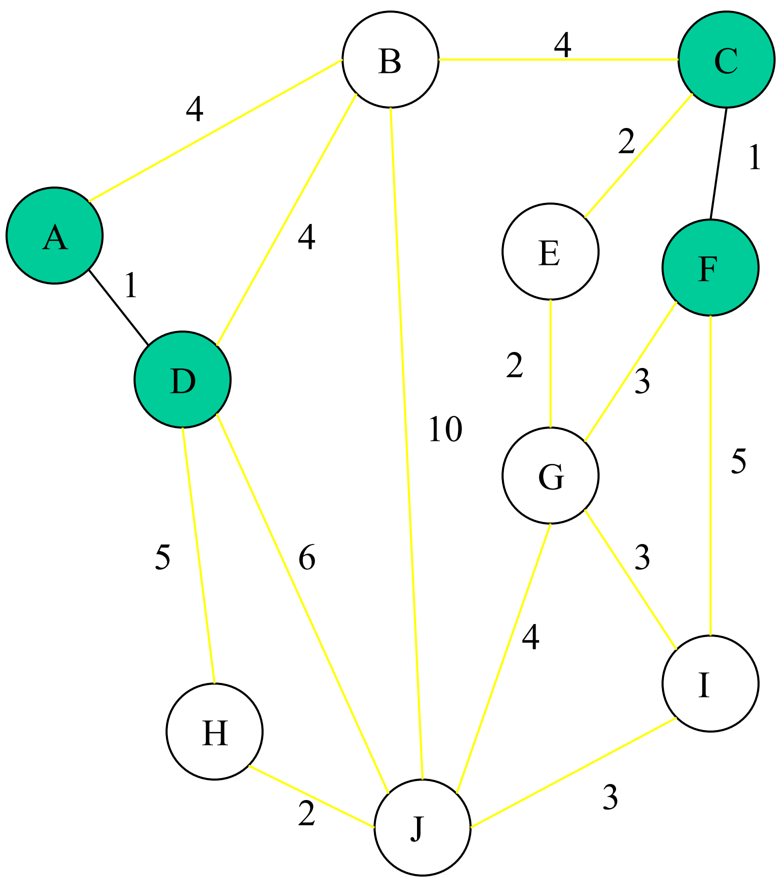


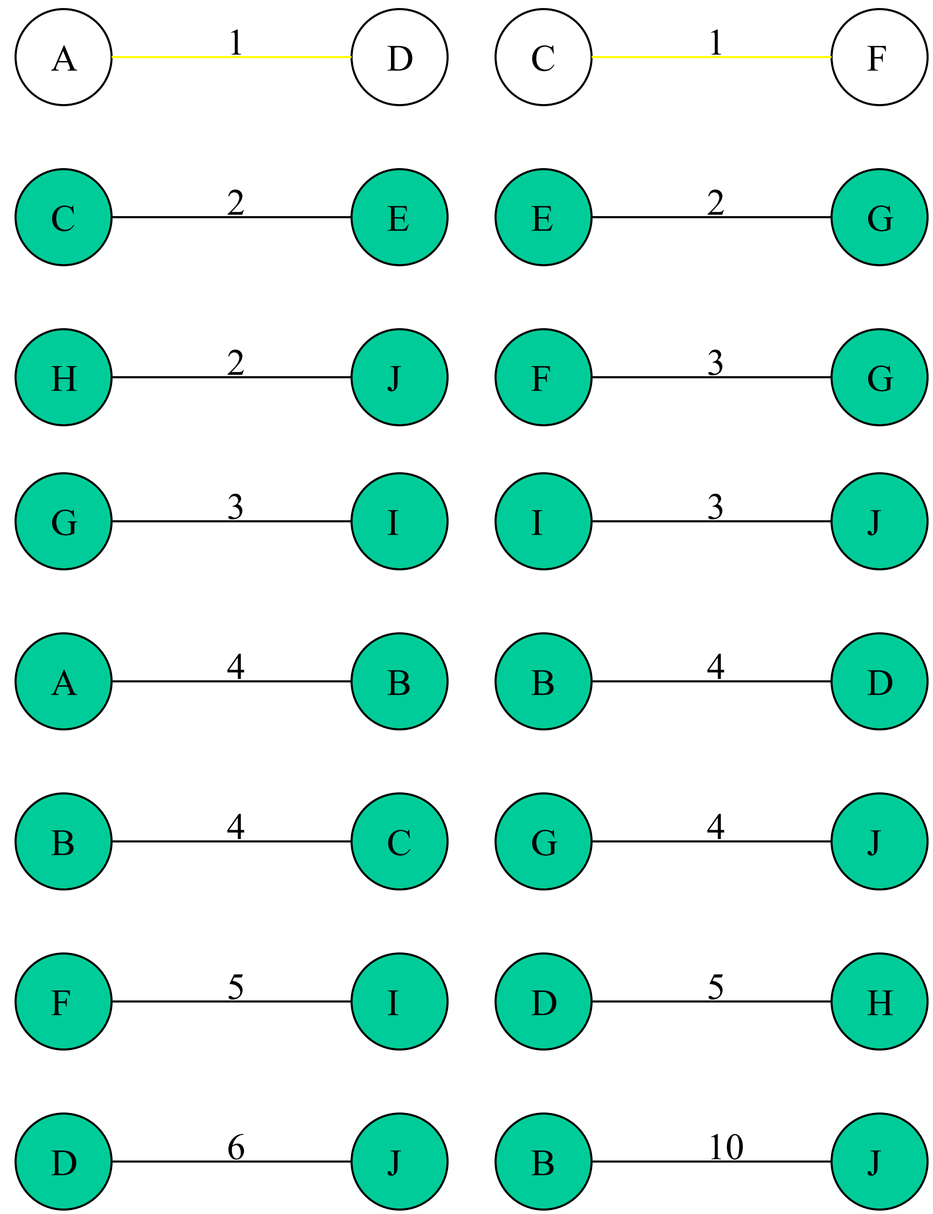
A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
10
2
3
5
5
6
4
3
2
3



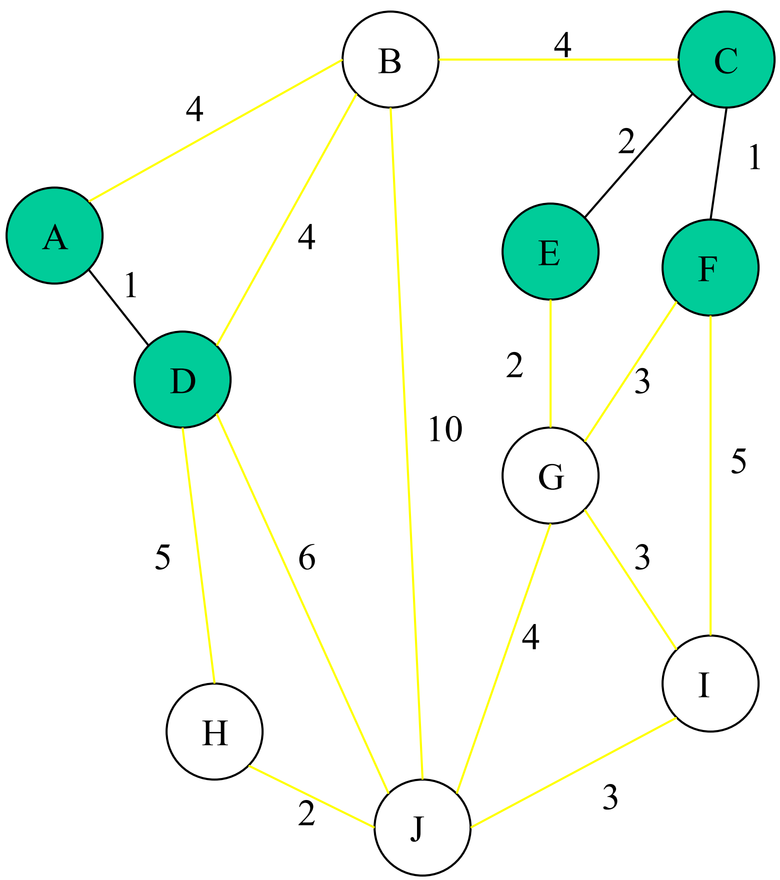
A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge

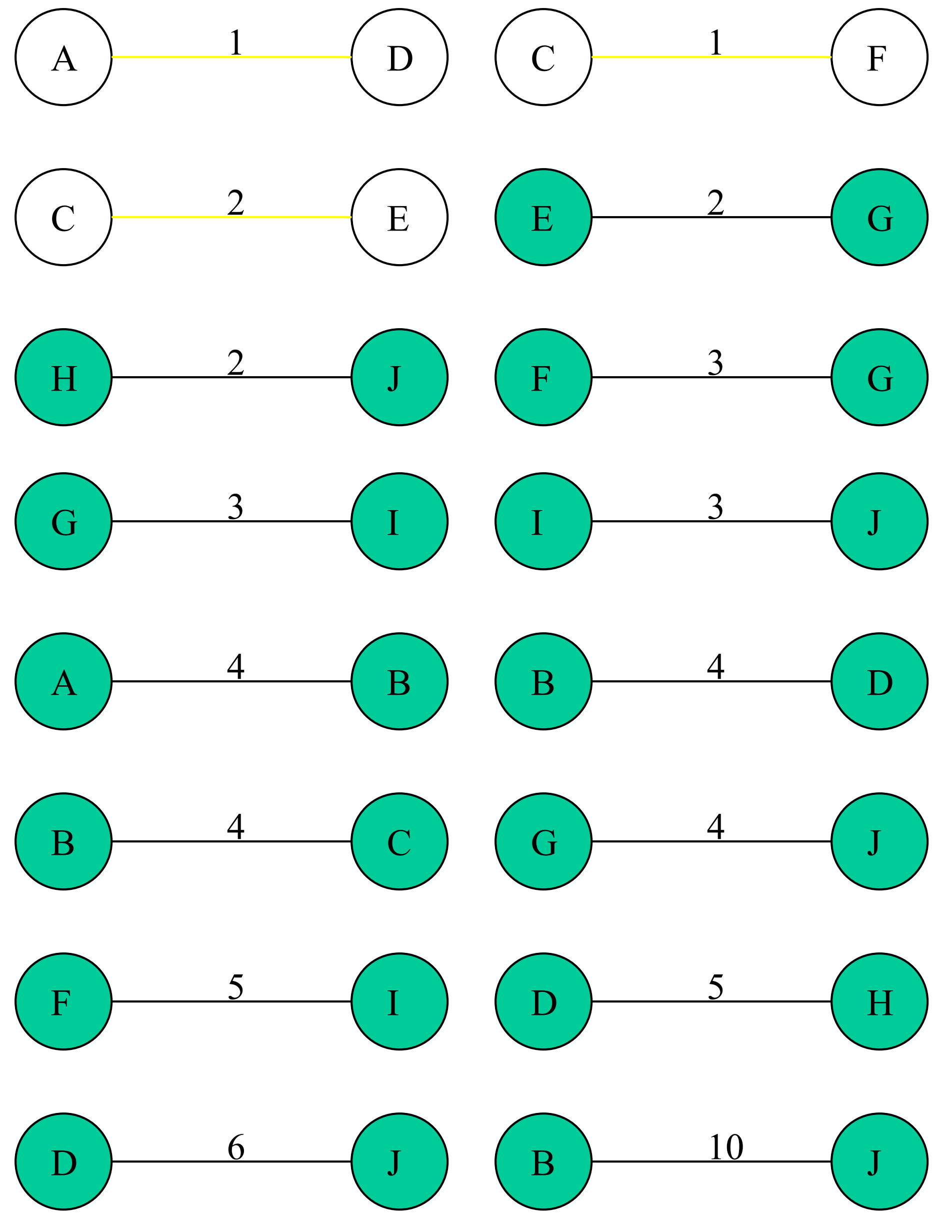

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
6
5
3
4
2
3


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

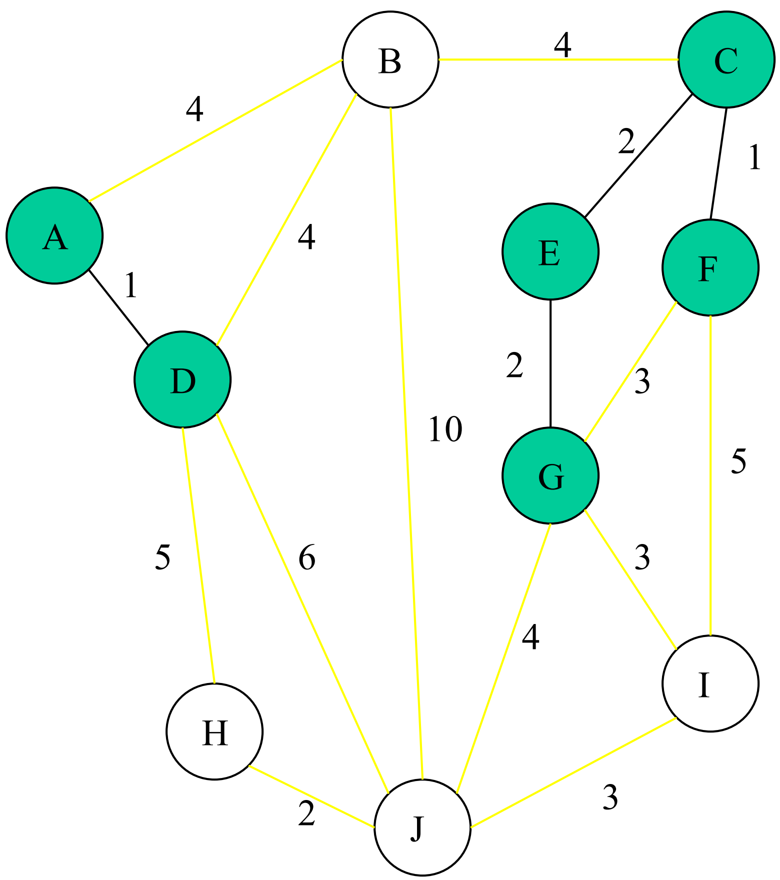


A
B
C
D
E
F
G
H
I
J
4
4
4
1
2
1
2
3
10
5
6
5
4
3
3
2
3

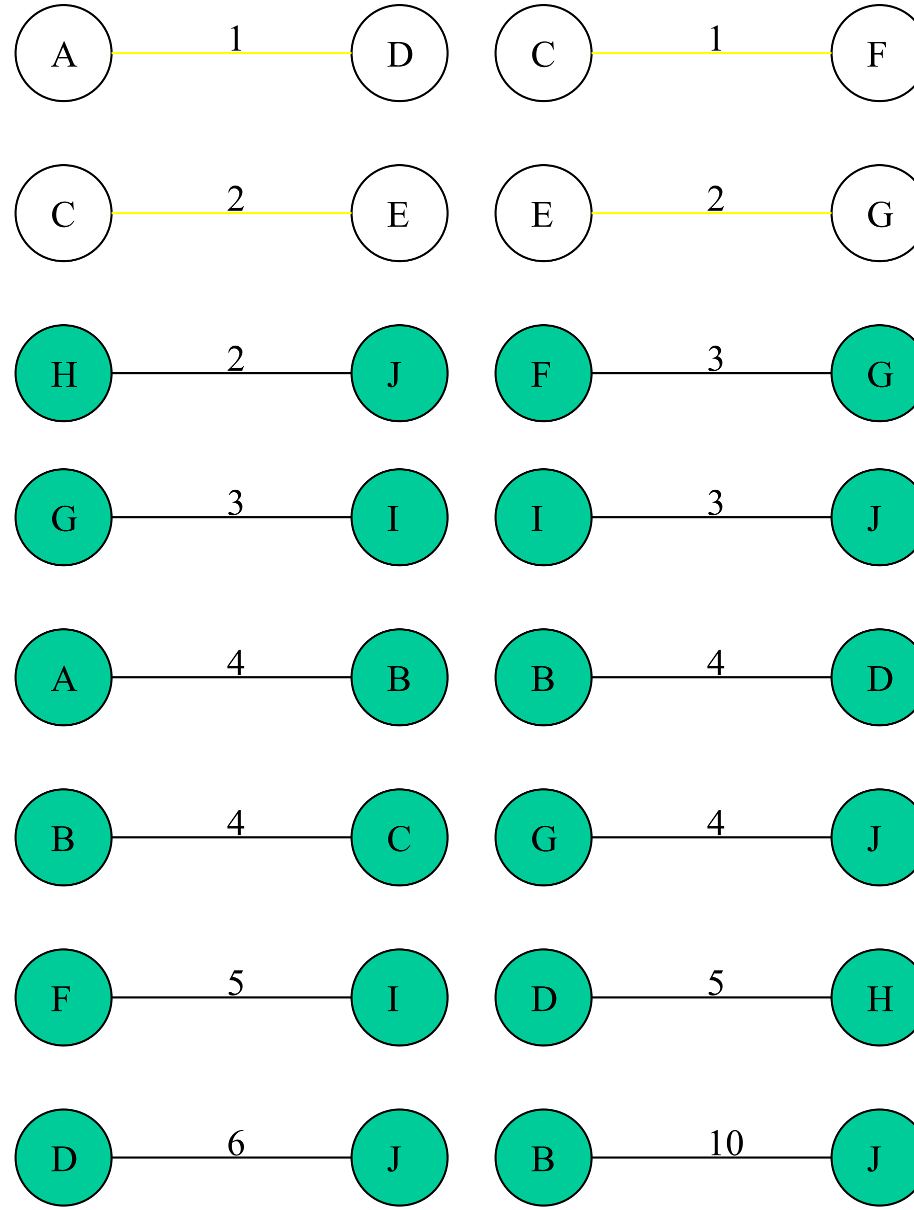


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge

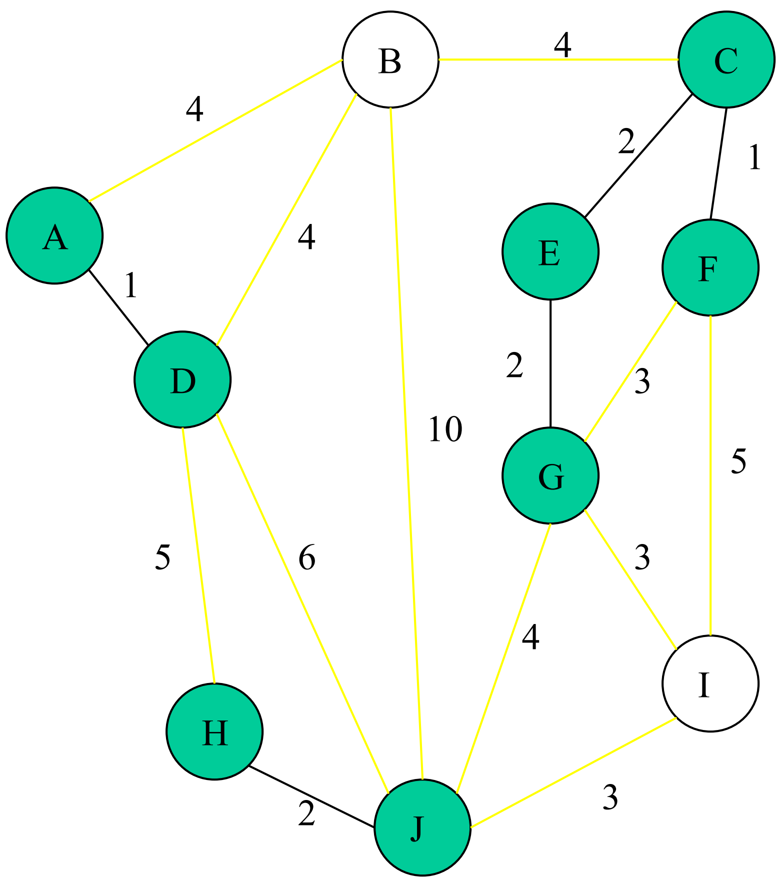


B
4
C
4
2
1
A
4
E
F
1
D
2
3
10
5
G
5
6
3
4
I
H
2
J
3

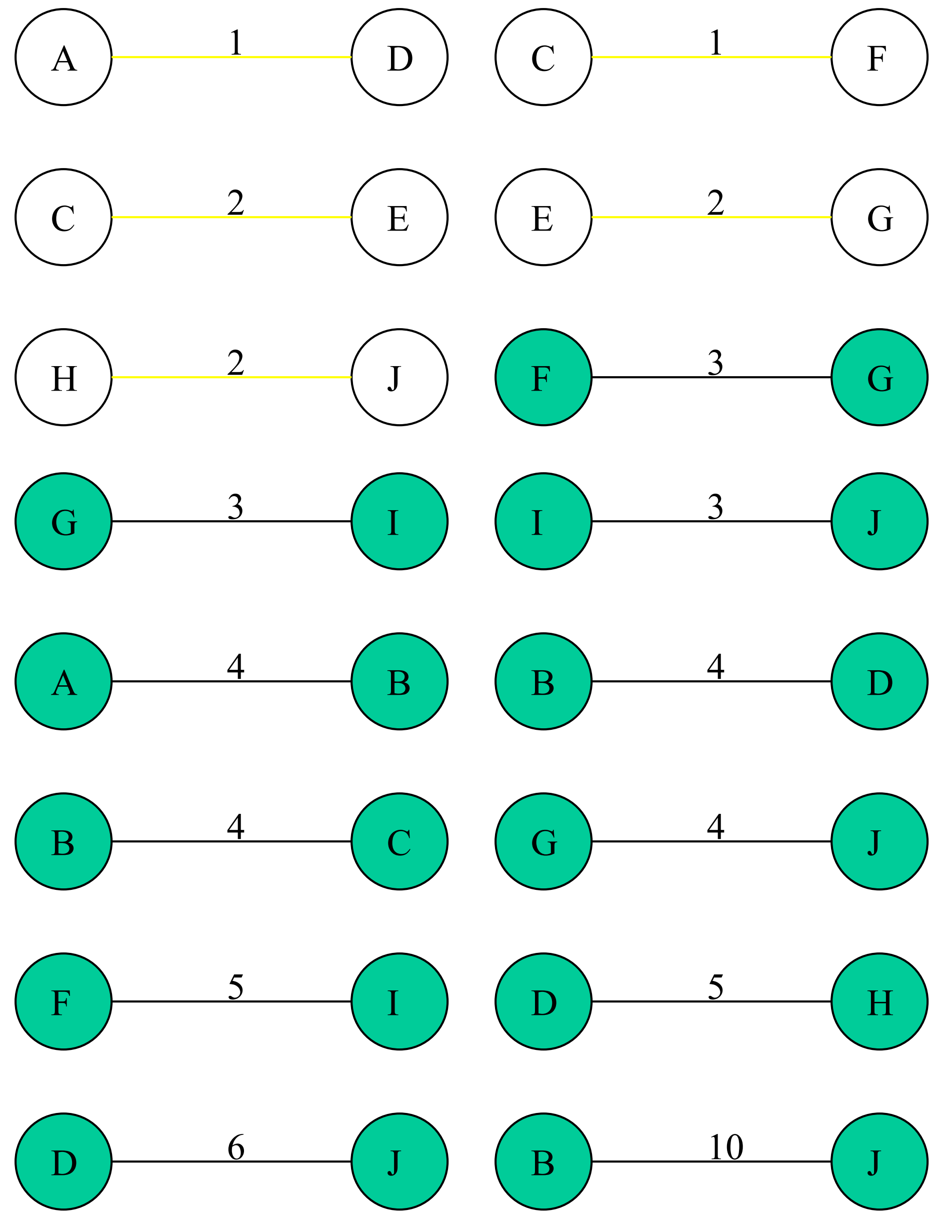


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge



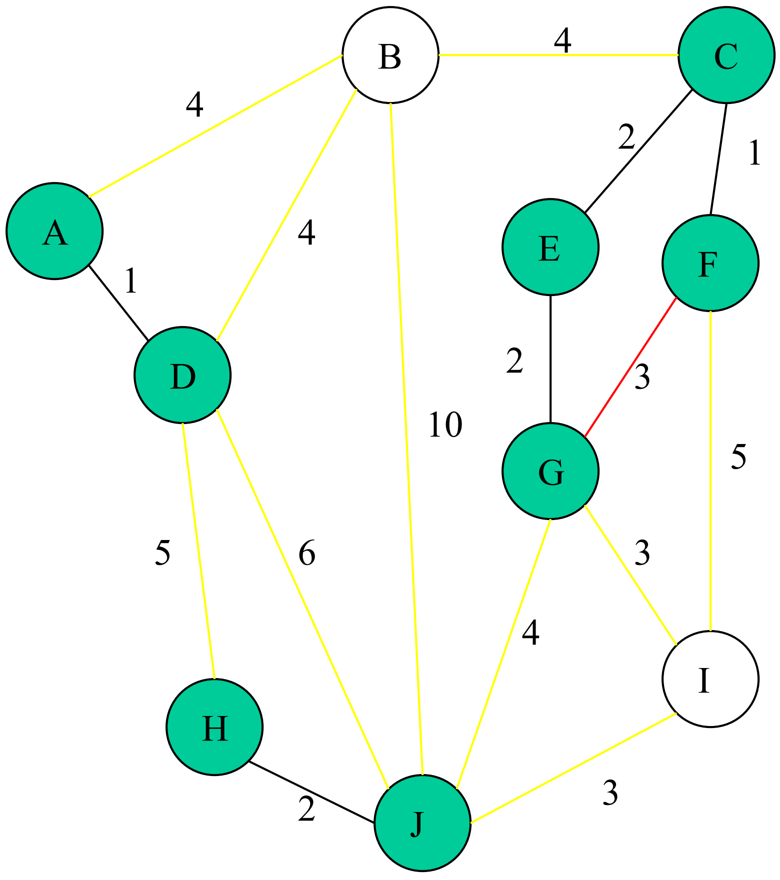
B
C
A
E
F
D
G
H
I
J
4
4
2
1
1
4
10
2
3
5
5
6
3
4
2
3

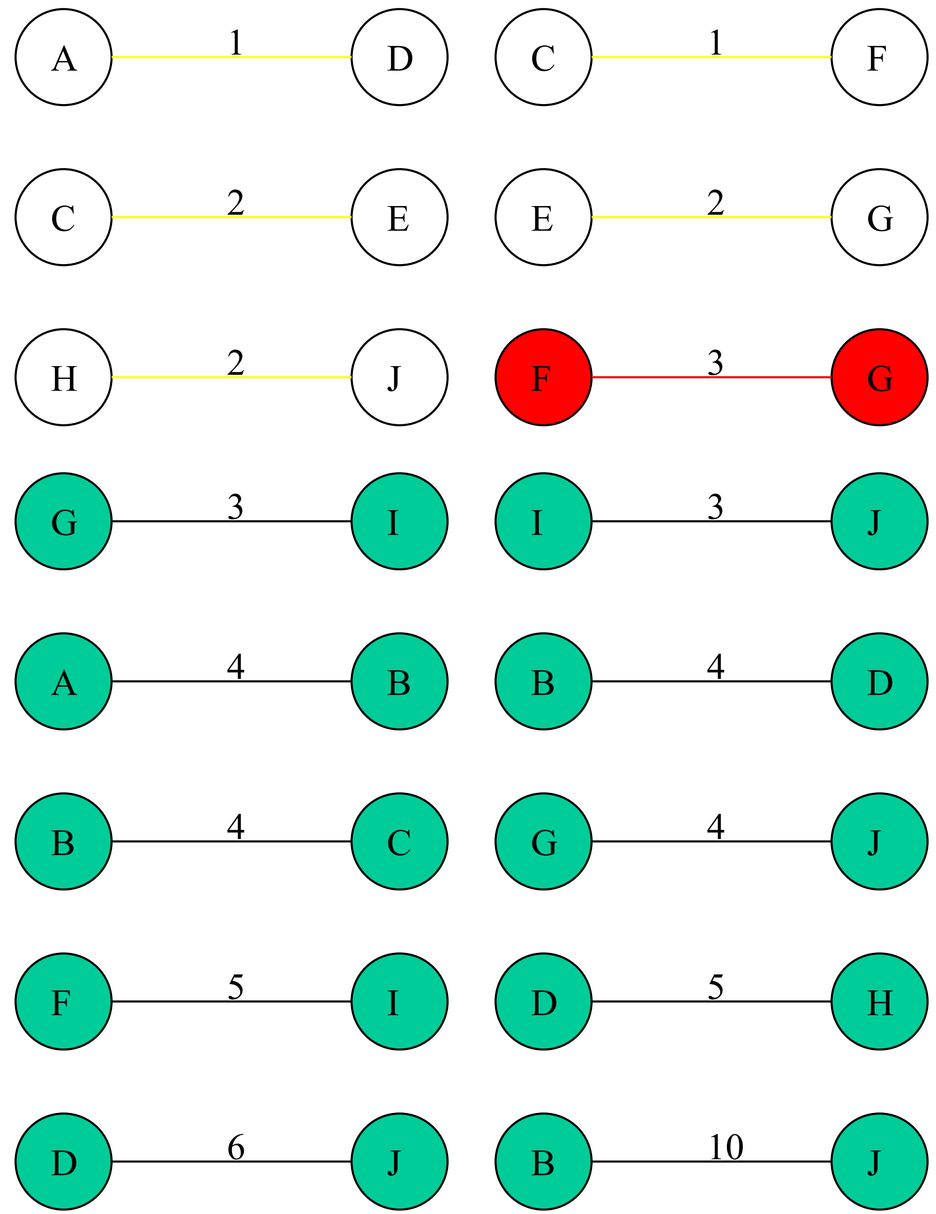


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

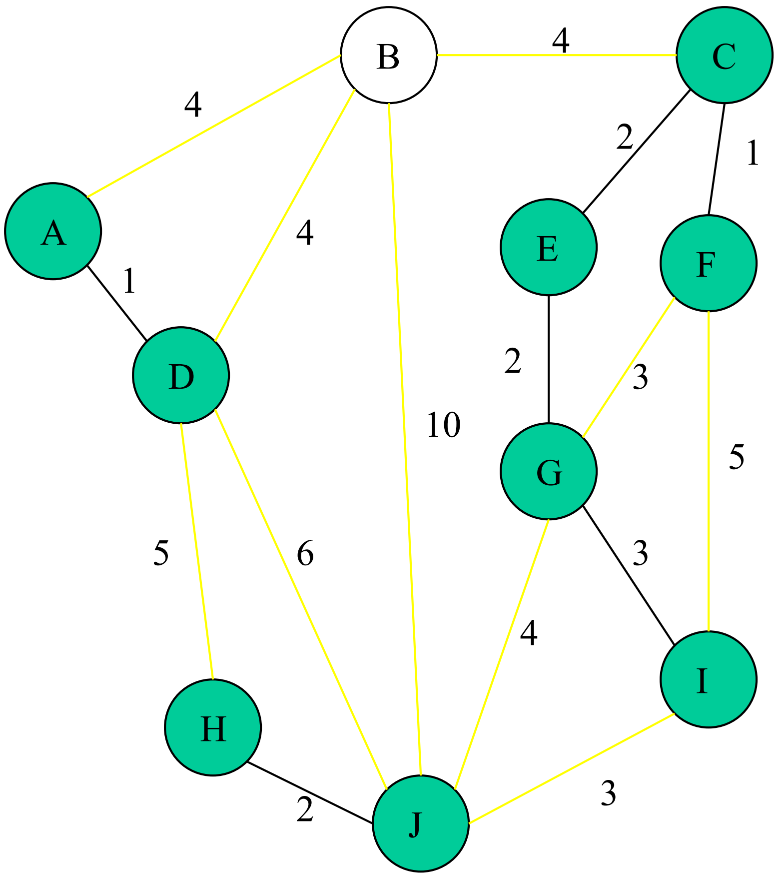
Cycle

Don’t Add Edge

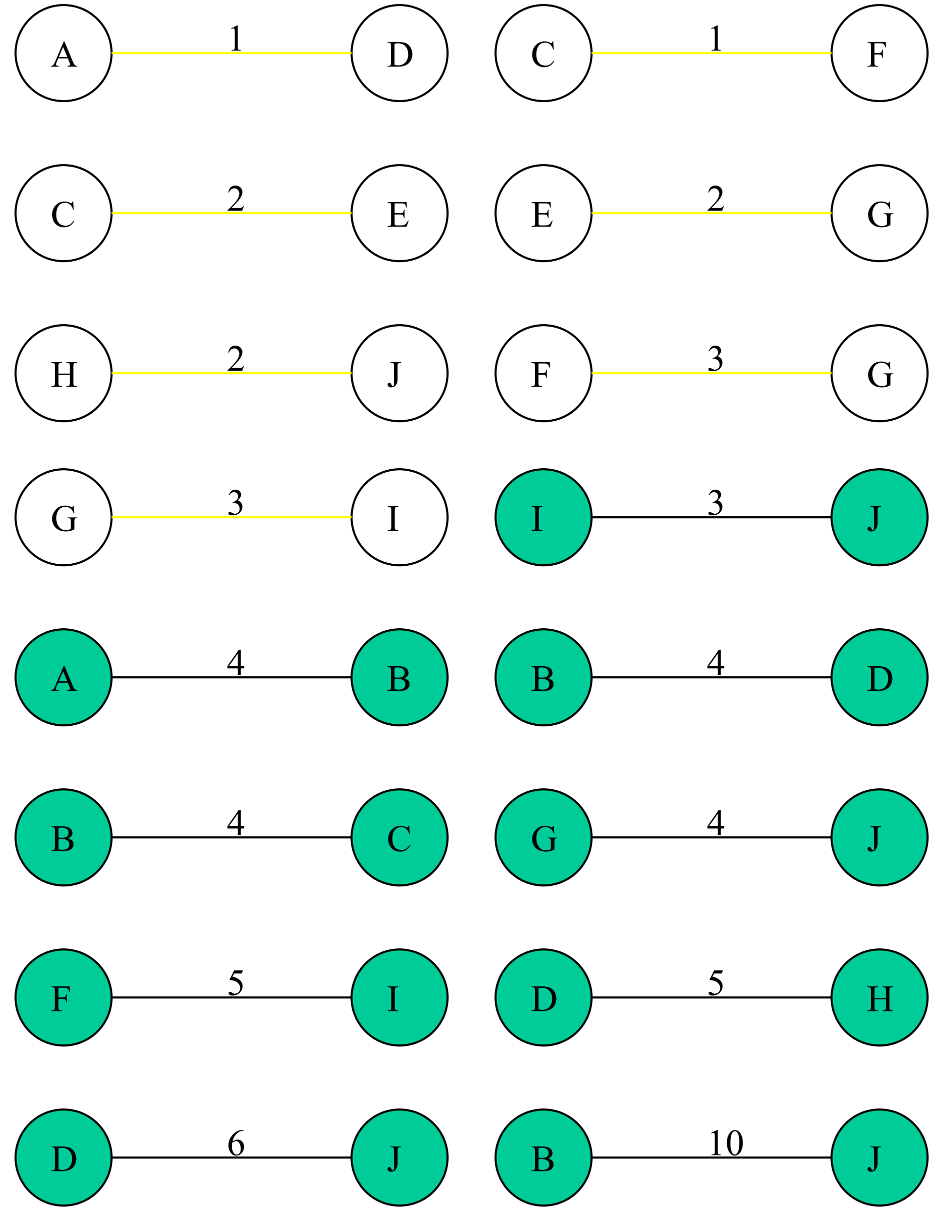




Add Edge

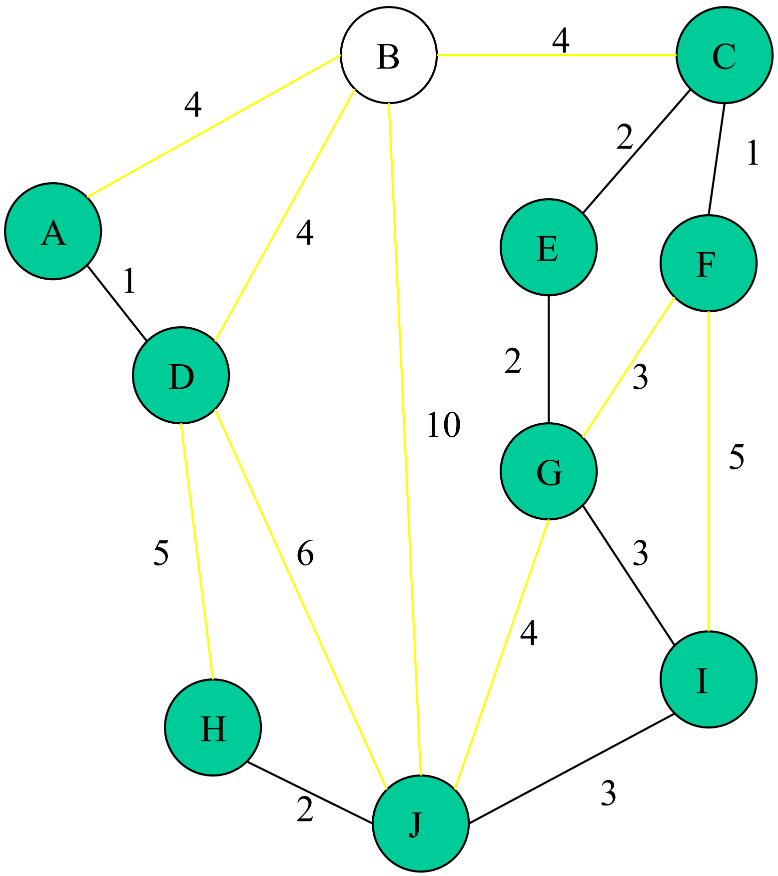


A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
4
10
2
3
5
5
6
3
4
2
3

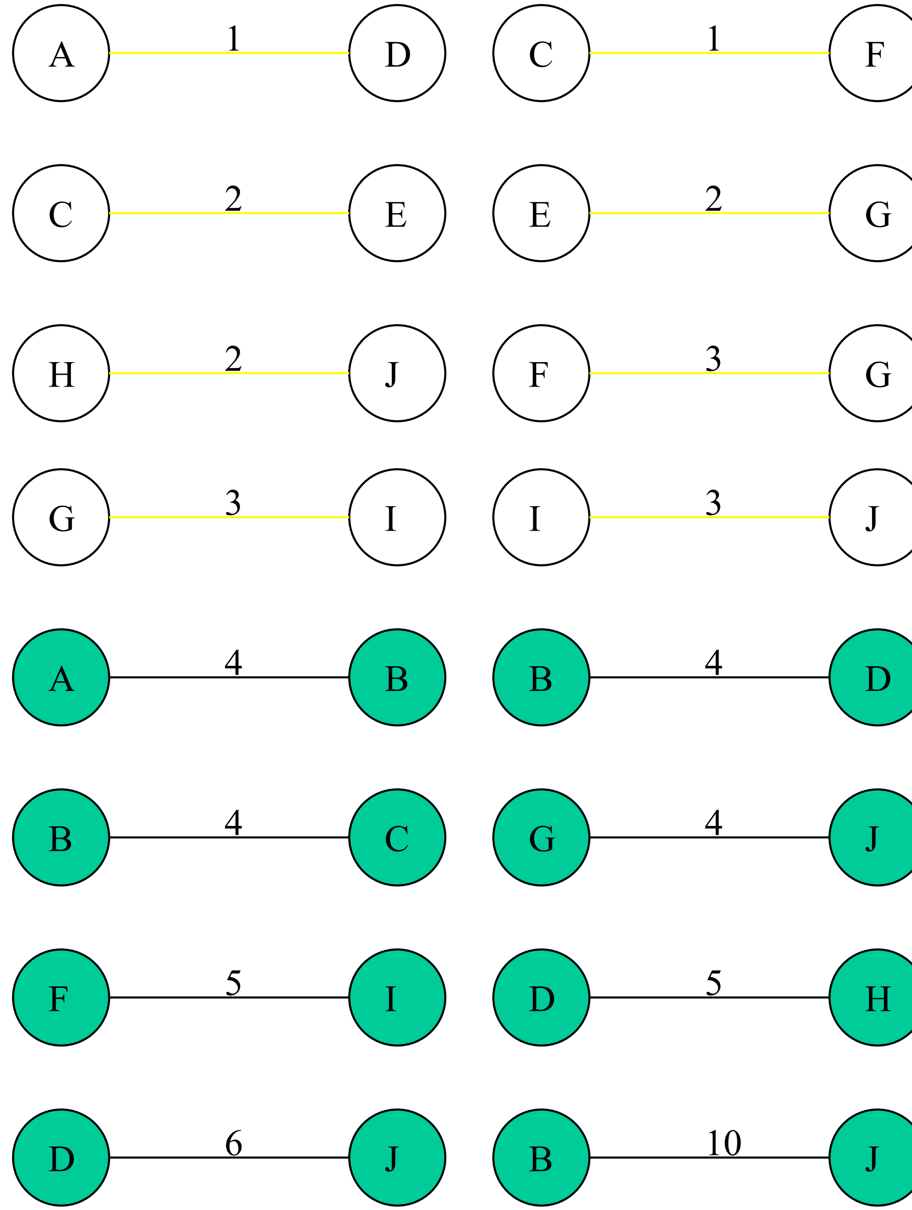


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge

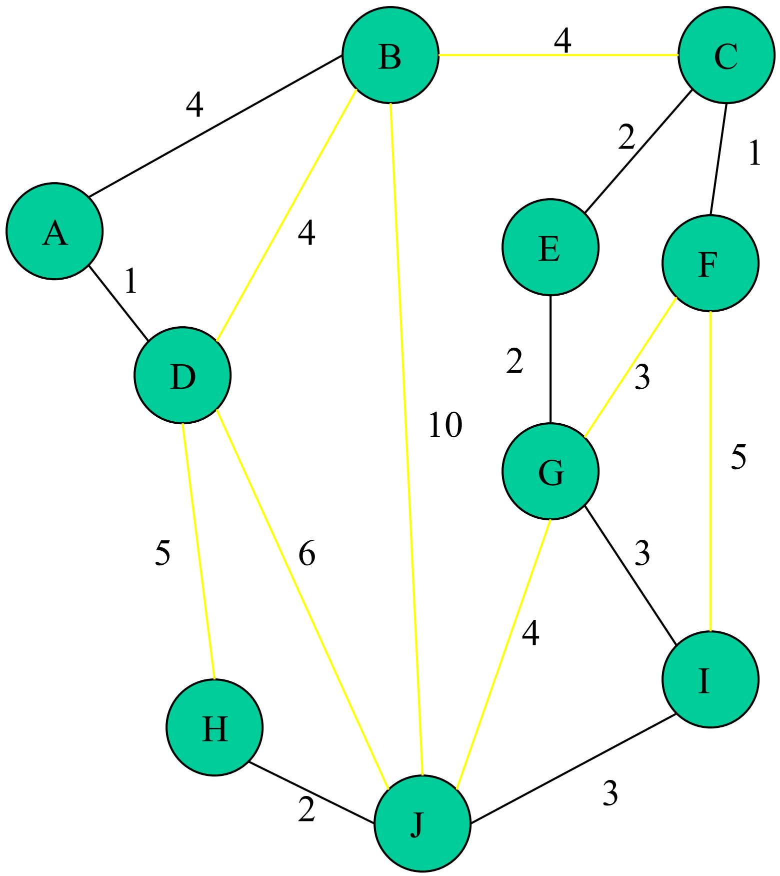


A
B
C
D
E
F
G
H
I
J
4
4
4
2
1
1
2
3
10
5
5
6
3
4
2
3

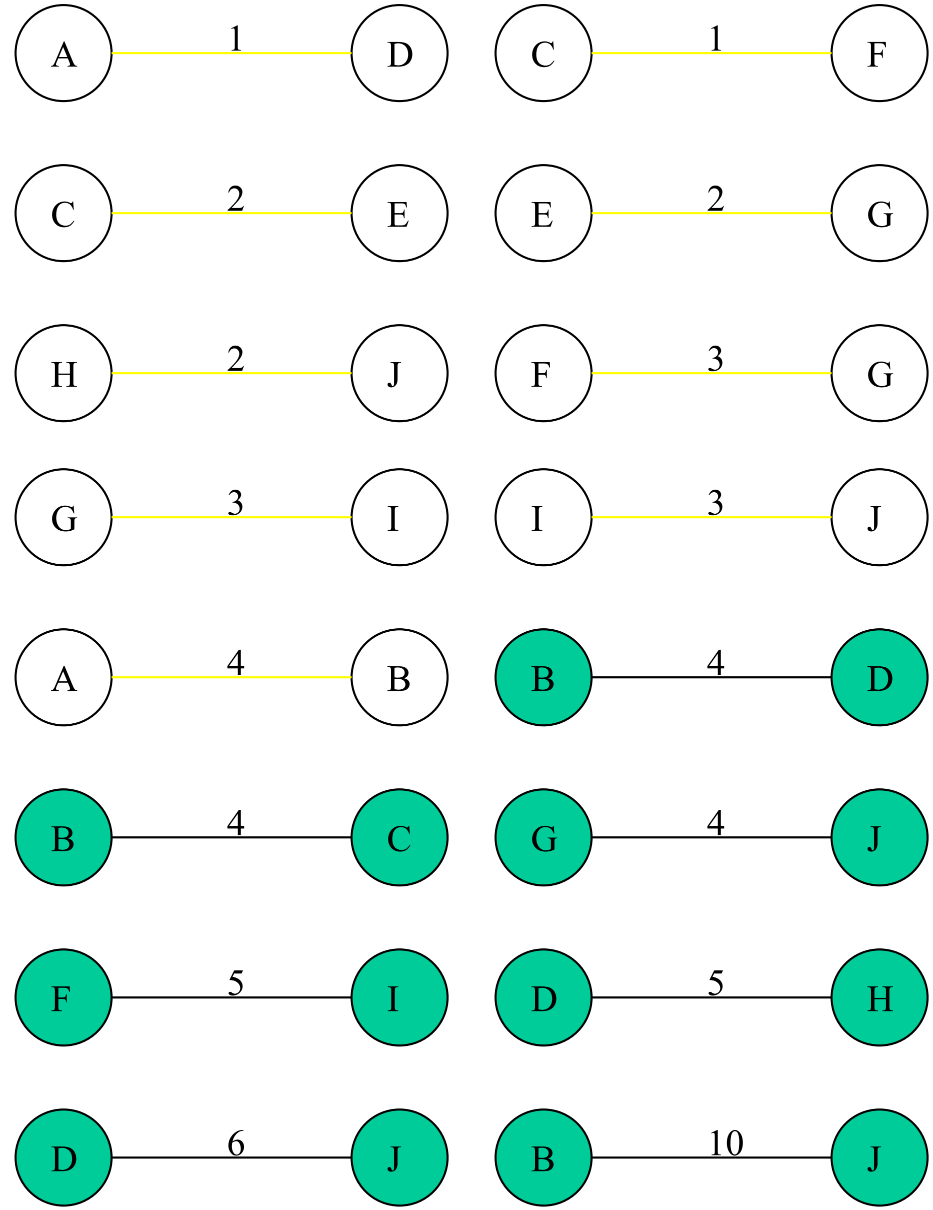


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge



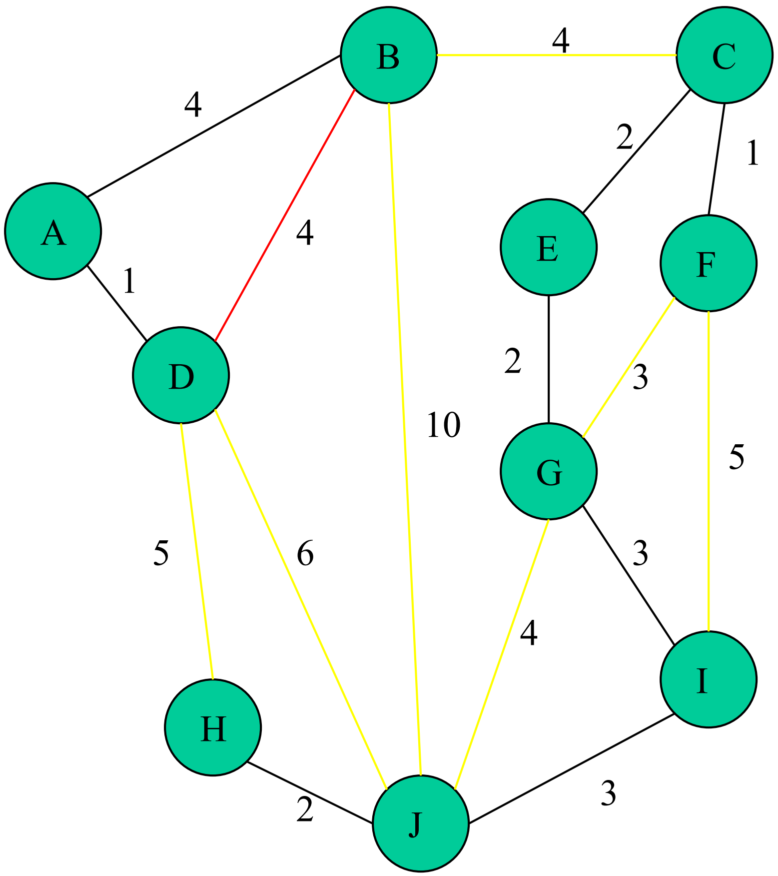
A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
2
3

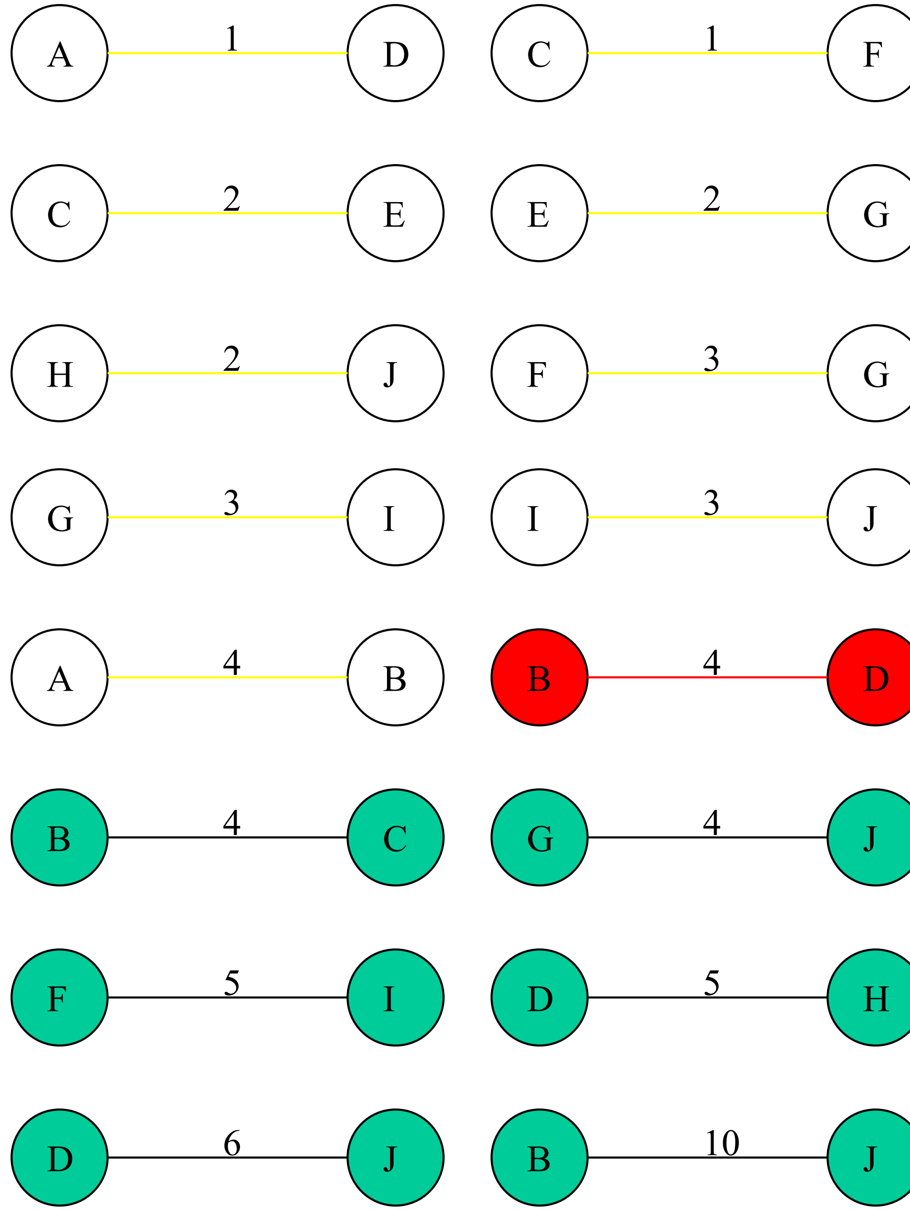


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

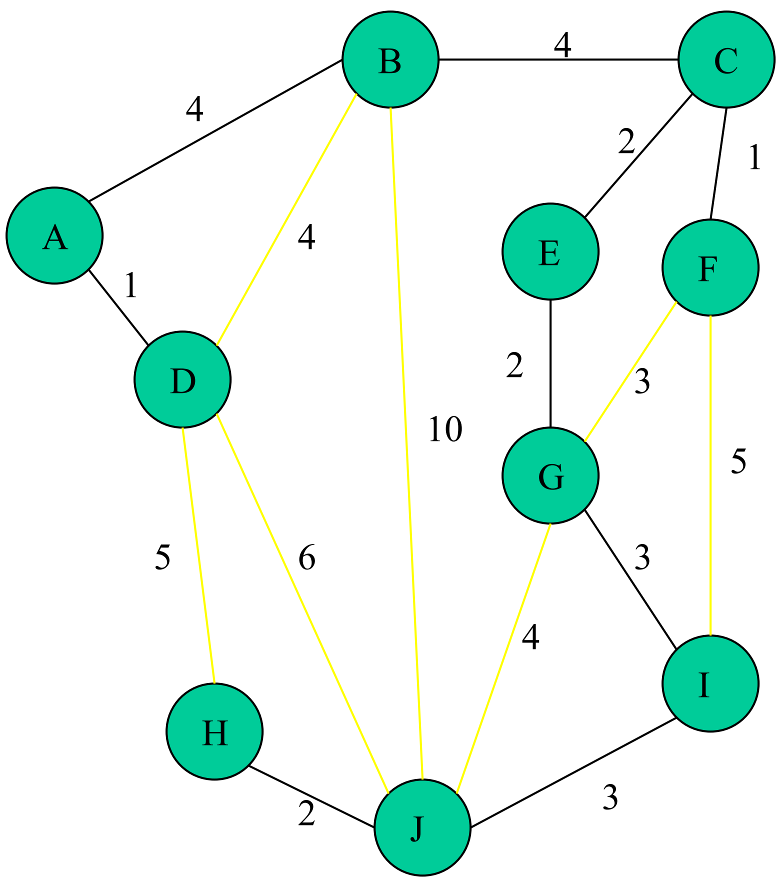
Cycle

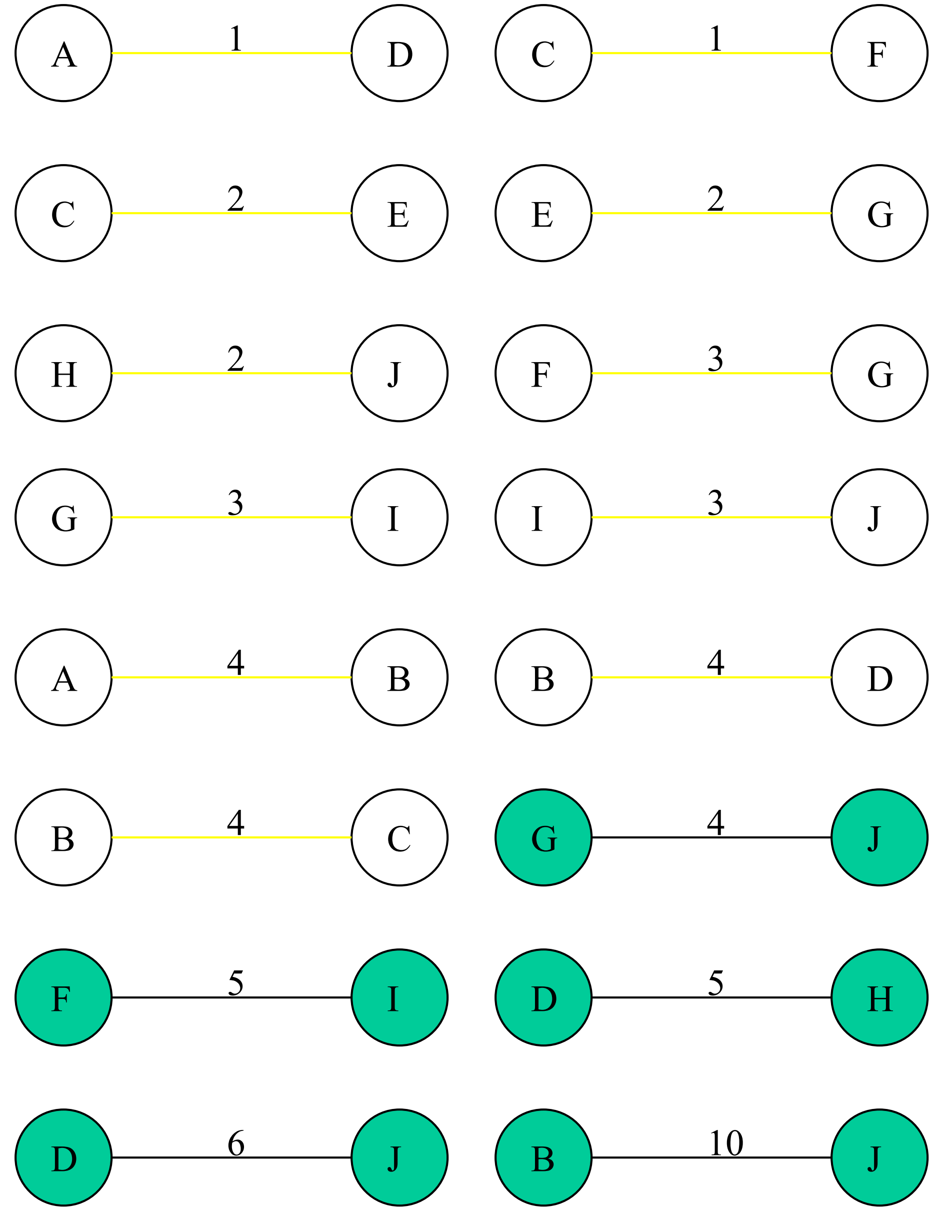
Don’t Add Edge



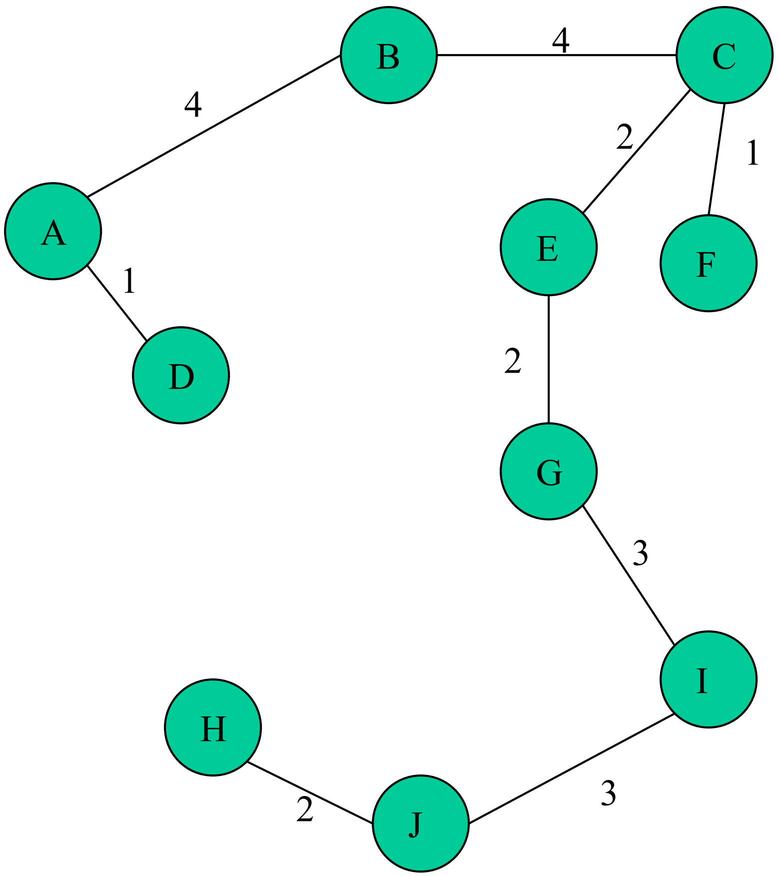


Add Edge

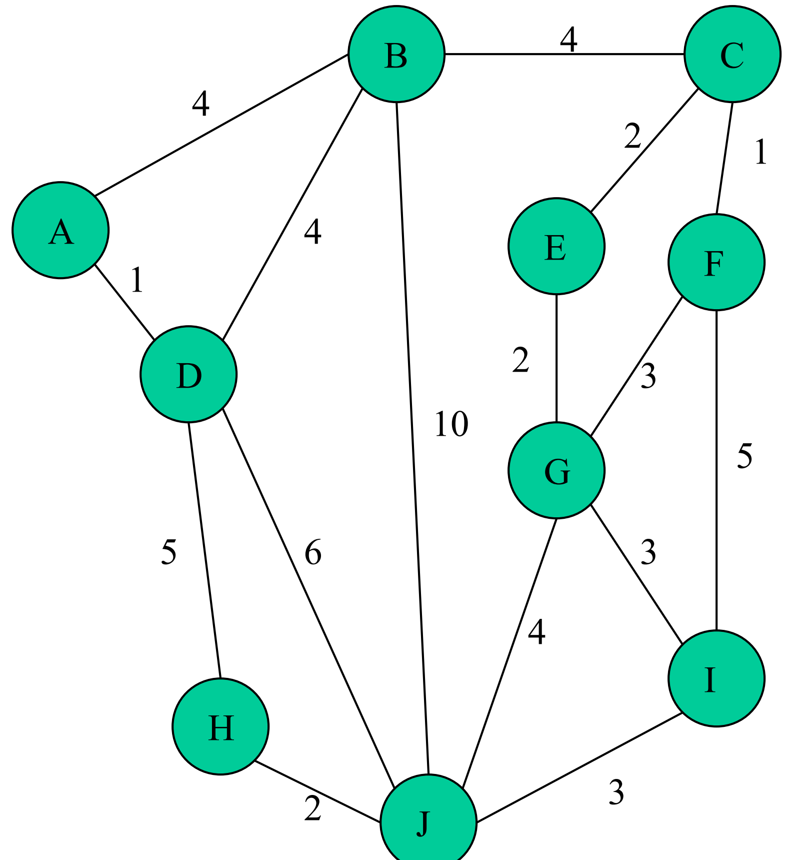

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
2
3


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

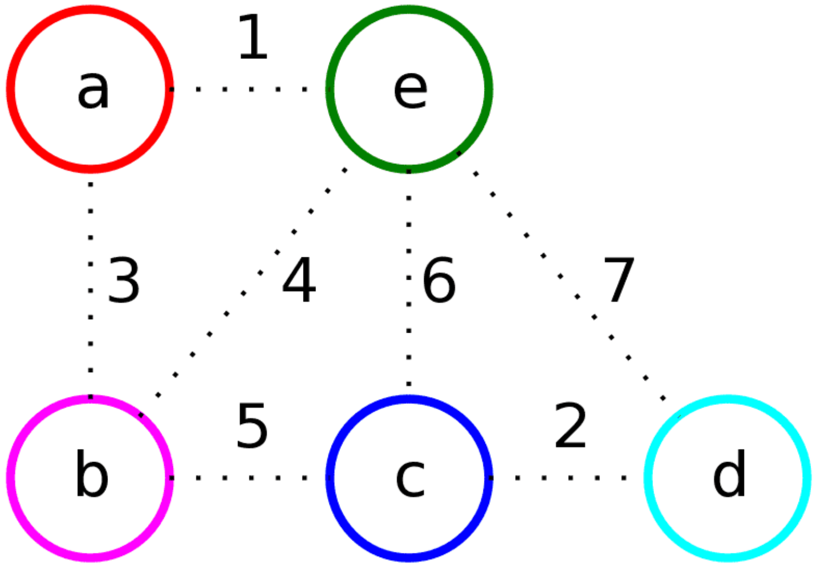


Complete Graph



Visualization of Kruskal's algorithm

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



"repeatedly add the cheapest **edge** that does not create a cycle"

Time complexity of Kruskal's Algorithm

Naïve Implementation: Maintain F as a graph. Check if an edge creates a cycle can be done by BFS/DFS (O(n) time).

Overall: O(mn) time.

A better “data-structure” :

each connected component is a set of vertices (maintain disjoint sets).

Need to operations:

(i) Given two vertices, do they belong to the same set

(ii) Replace two sets by their union.

UNION-FIND Datastructure: each operation takes O(log n) time.
Oveall O(m log n) time.

Proof of Kruskal's Algorithm

Assume all edge lengths are distinct.

Suppose the edges picked by the algorithm (in the order of picking) are

$$f_1, f_2, \dots, f_{n-1}$$

Consider an optimal solution T^*, and consider its edges in increasing cost be

$$g_1, g_2, \dots, g_n$$

Let r be the first index where they differ, i.e.,

$f_1 = g_1, \dots, f_{r-1} = g_{r-1}$, but $f_r \neq g_r$.

Proof of Kruskal's Algorithm

Add f_r to T^* : creates a cycle C.

There must be an edge in this cycle which is more expensive than f_r.