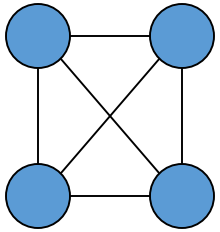
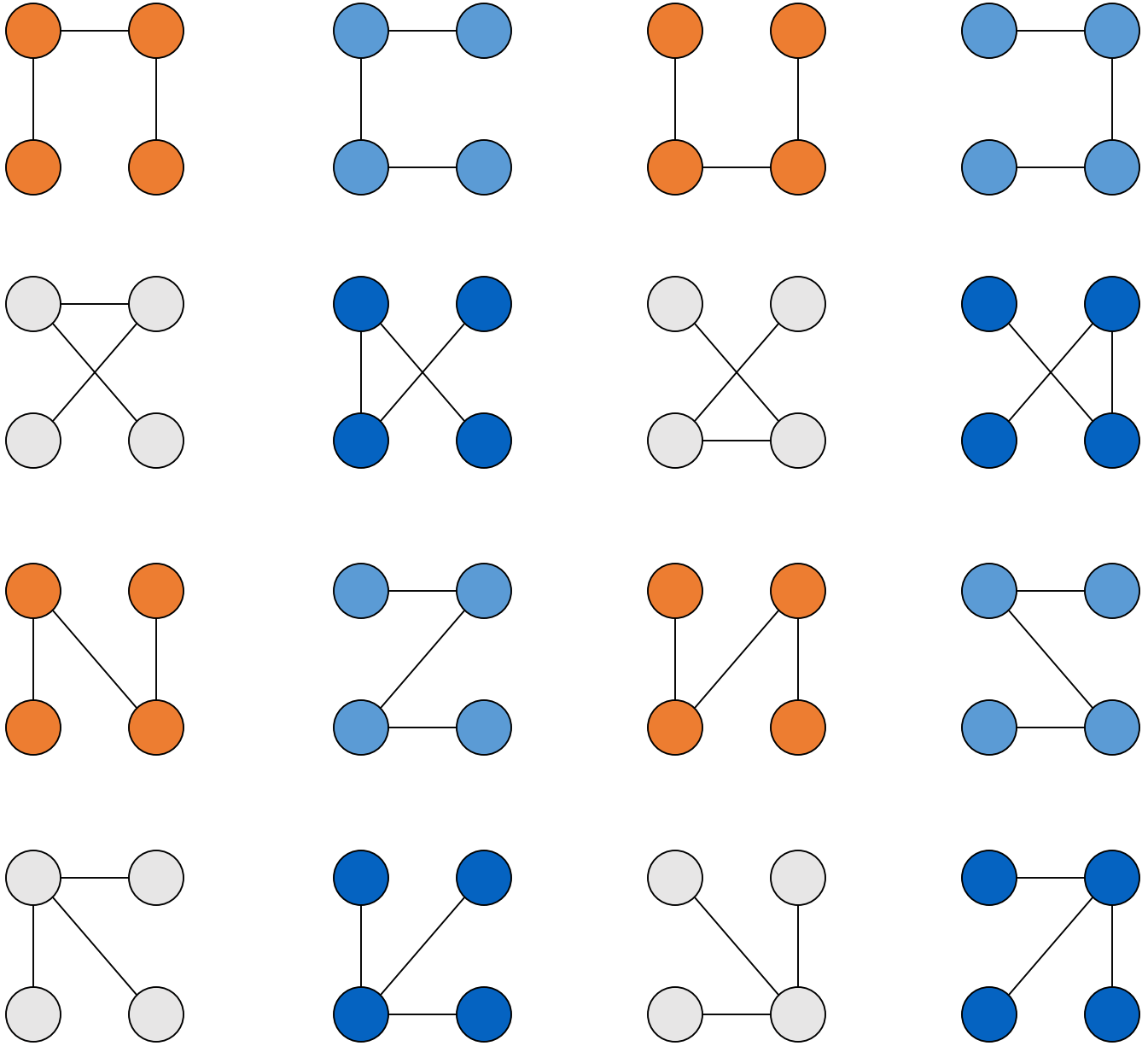
Spanning Trees

- **Tree**: a connected graph **without cycles.**
- A **spanning tree** of a connected undirected graph is a **subgraph** that spans (**contains**) **all nodes** and enough edges to connect them with **no cycles**.
- There could be **more** than one spanning tree for a graph, and the commonly used one in problem solving is the minimum spanning tree.

Complete Graph

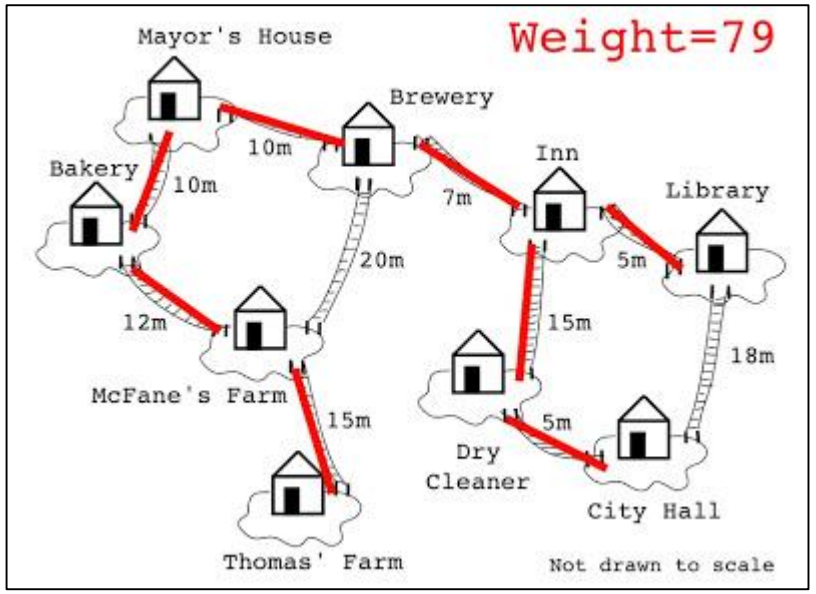


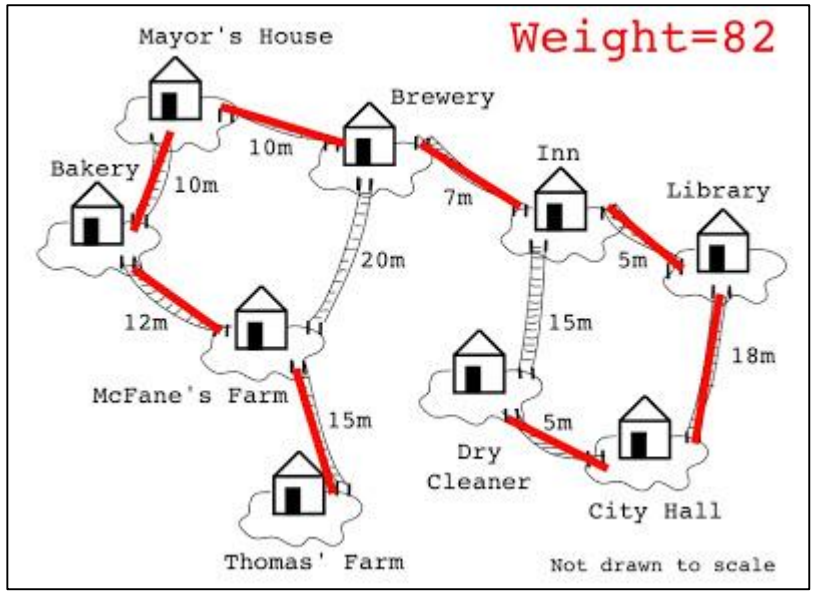
All 16 of its Spanning Trees



Minimizing costs

- Suppose you want to supply a set of houses with electric power, water, sewage lines, or telephone lines.
- To keep costs down, you could connect these houses with a spanning tree. However, the houses are not all equal distances apart.



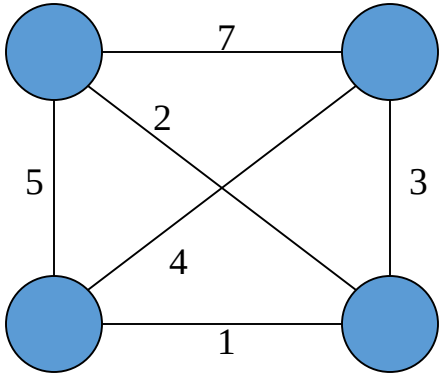


- To reduce costs even further, you could connect the houses with a ***minimum* spanning tree.**

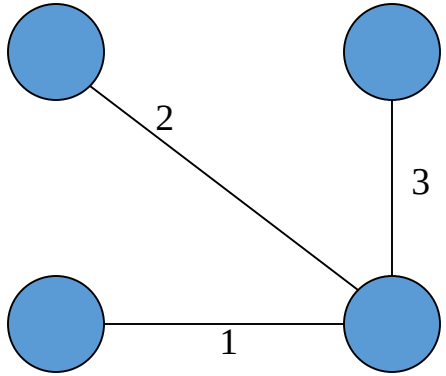
Minimum Spanning Trees

A spanning tree that has the lowest weight

Complete Graph



Minimum Spanning Tree



Finding minimum spanning trees

1. **Kruskal's algorithm**
2. **Prim's algorithm**
3. **Borůvka's algorithm**

Kruskal's Algorithm

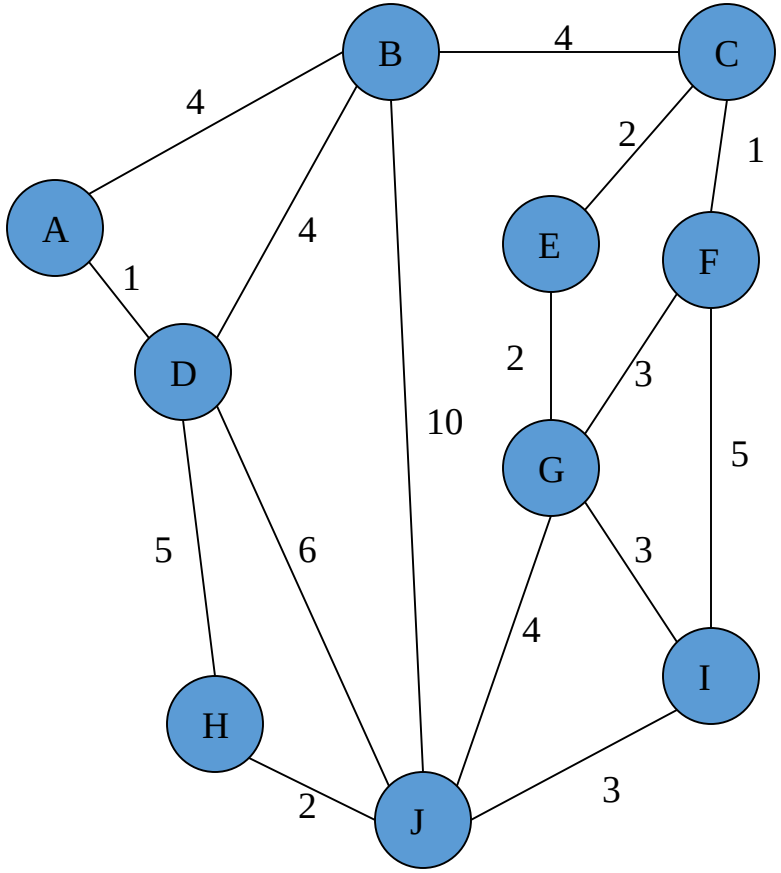
This algorithm creates a **forest** of trees. Initially the forest consists of **n single node trees** (and no edges). At each step, we **add one edge** (the **cheapest** one) so that it joins two trees together. If it were to form a **cycle**, it would simply link two nodes that were already part of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

The steps are:

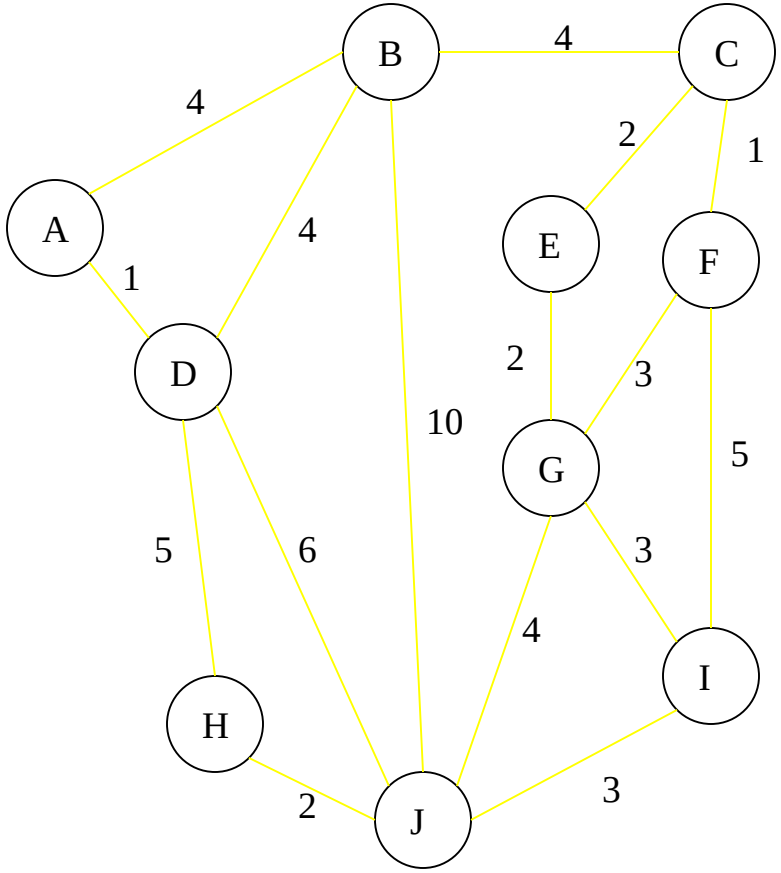
1. The **forest** is constructed - with each node in a separate tree.
2. The edges are placed in a **priority queue**.
3. Until we've added n-1 edges,
    1. Extract the **cheapest edge** from the queue,
    2. If it forms a **cycle**, **reject** it,
    3. Else **add** it to the forest. Adding it to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

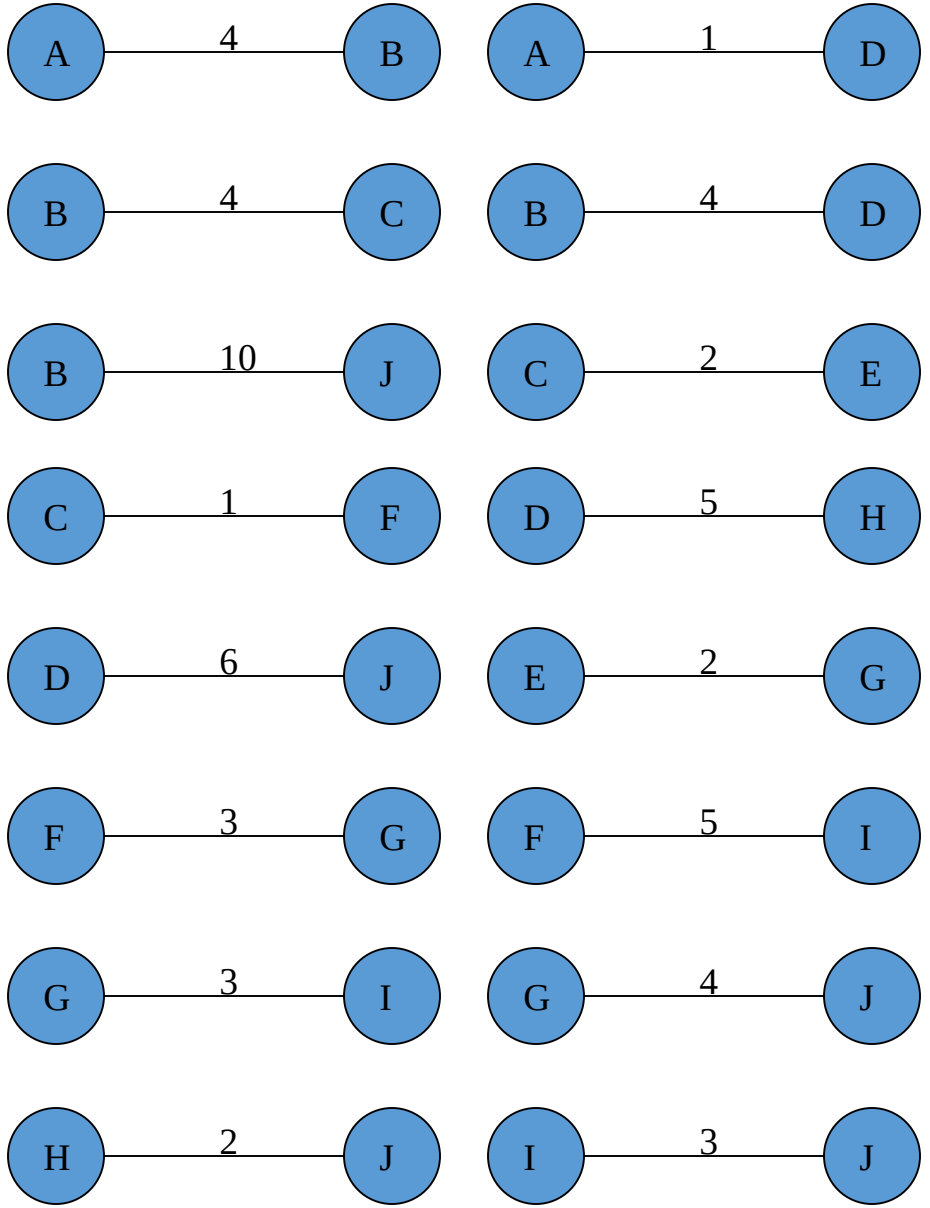
- Every step will have joined two trees in the forest together, so that at the end, there will only be one tree in T.
- Efficient testing for a cycle requires (UNION-FIND) algorithm which we don't cover in this course.

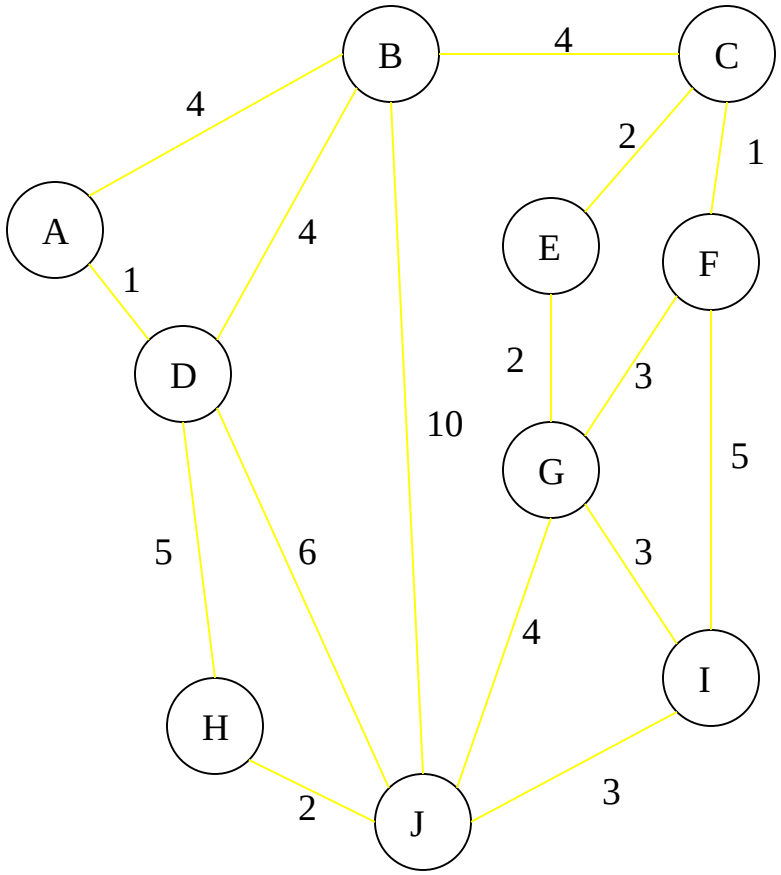
Complete Graph



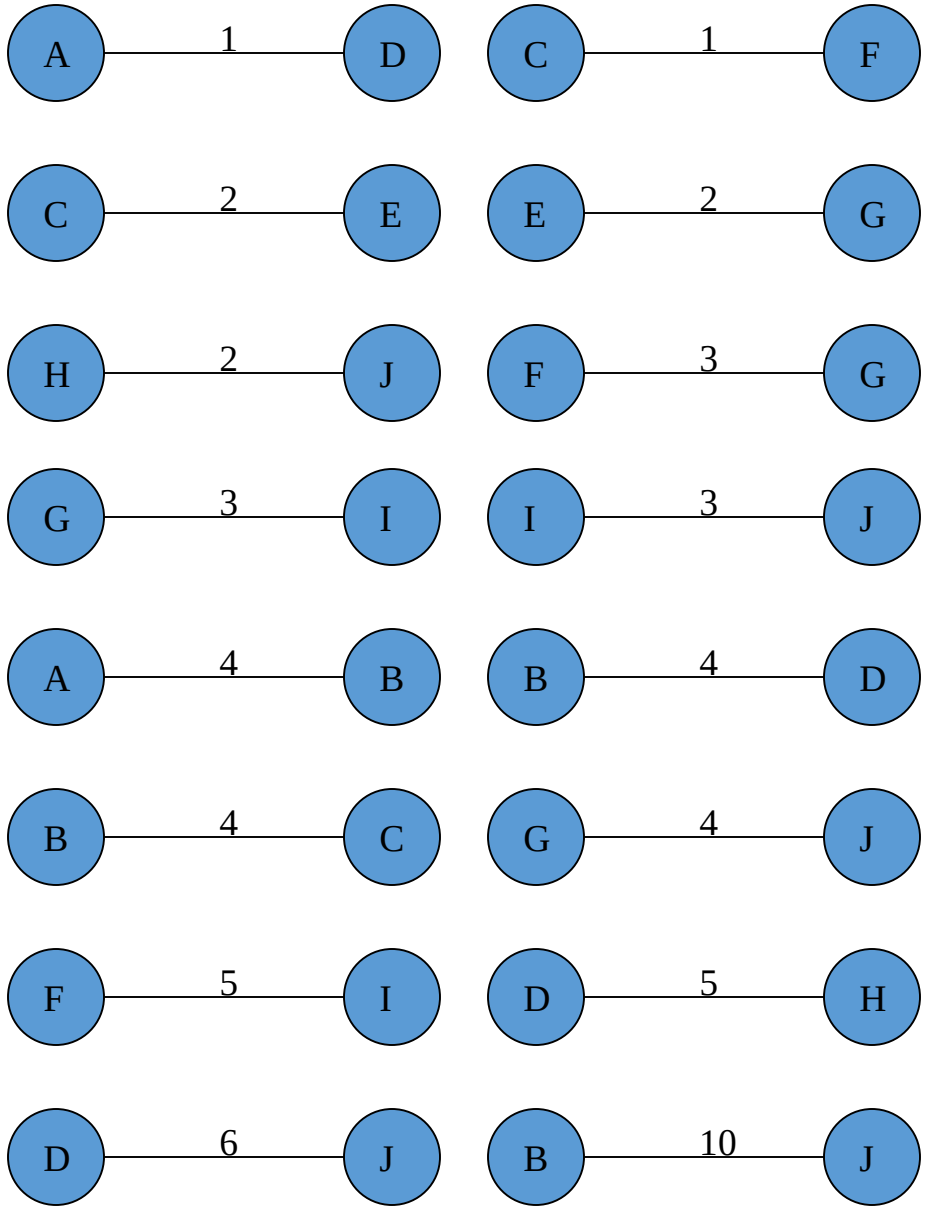
Forest


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

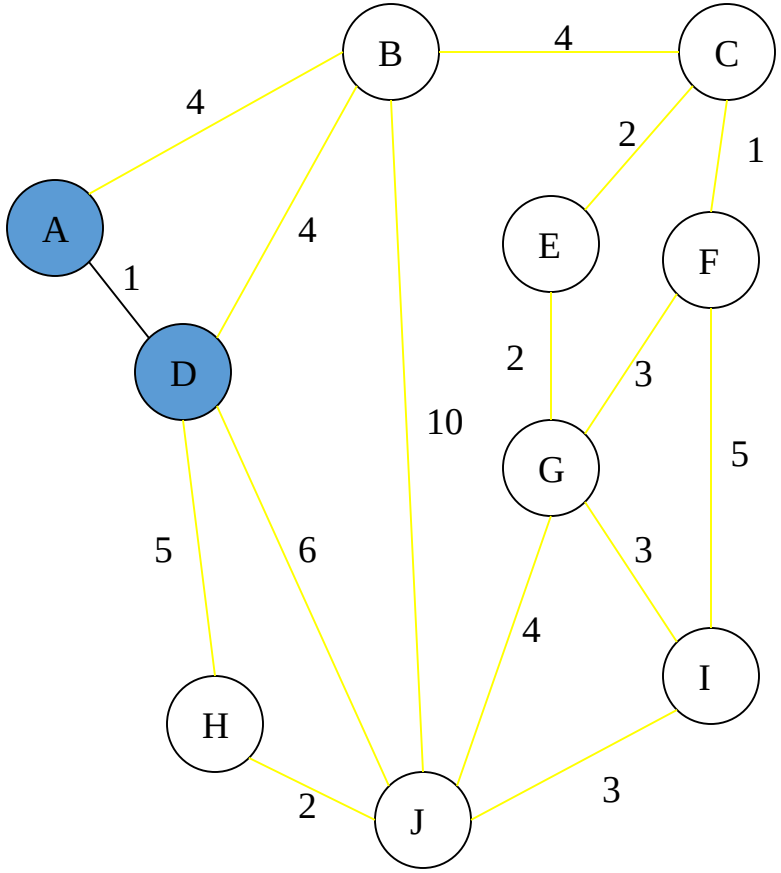

A 4 B
A 1 D
B 4 C
B 4 D
B 10 J
C 2 E
C 1 F
D 5 H
D 6 J
E 2 G
F 3 G
F 5 I
G 3 I
G 4 J
H 2 J
I 3 J

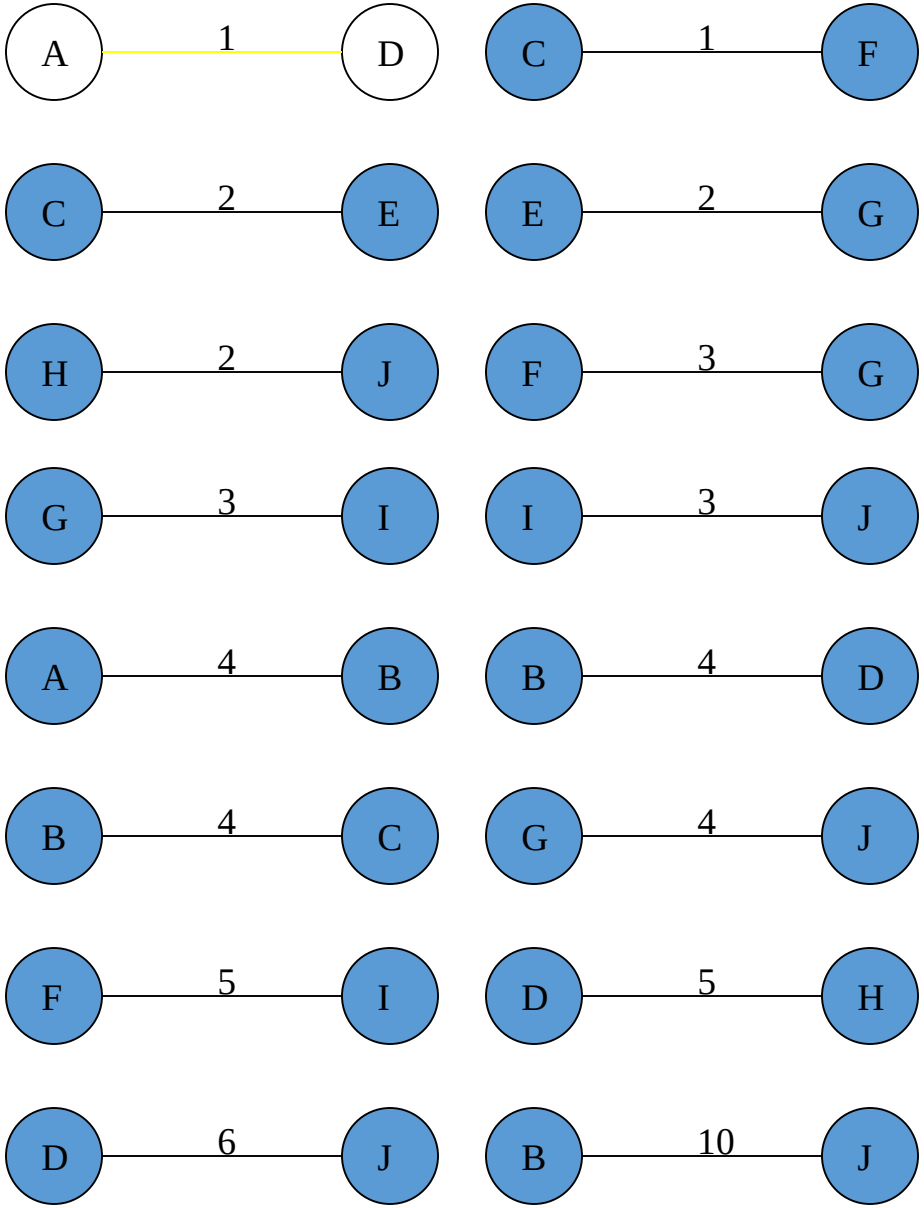


Edges are placed in a priority queue

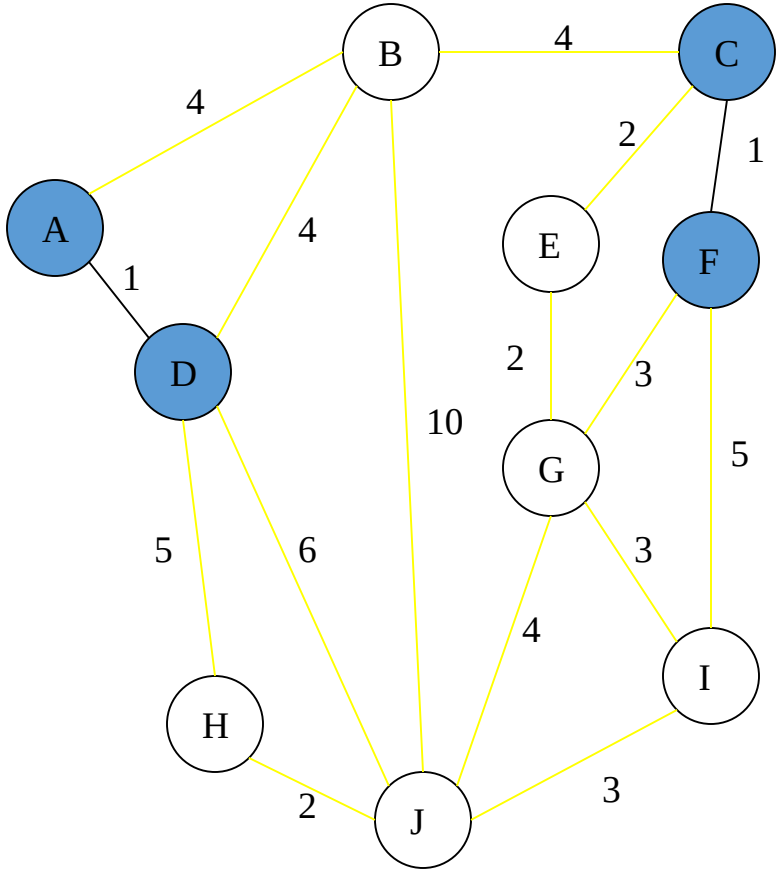


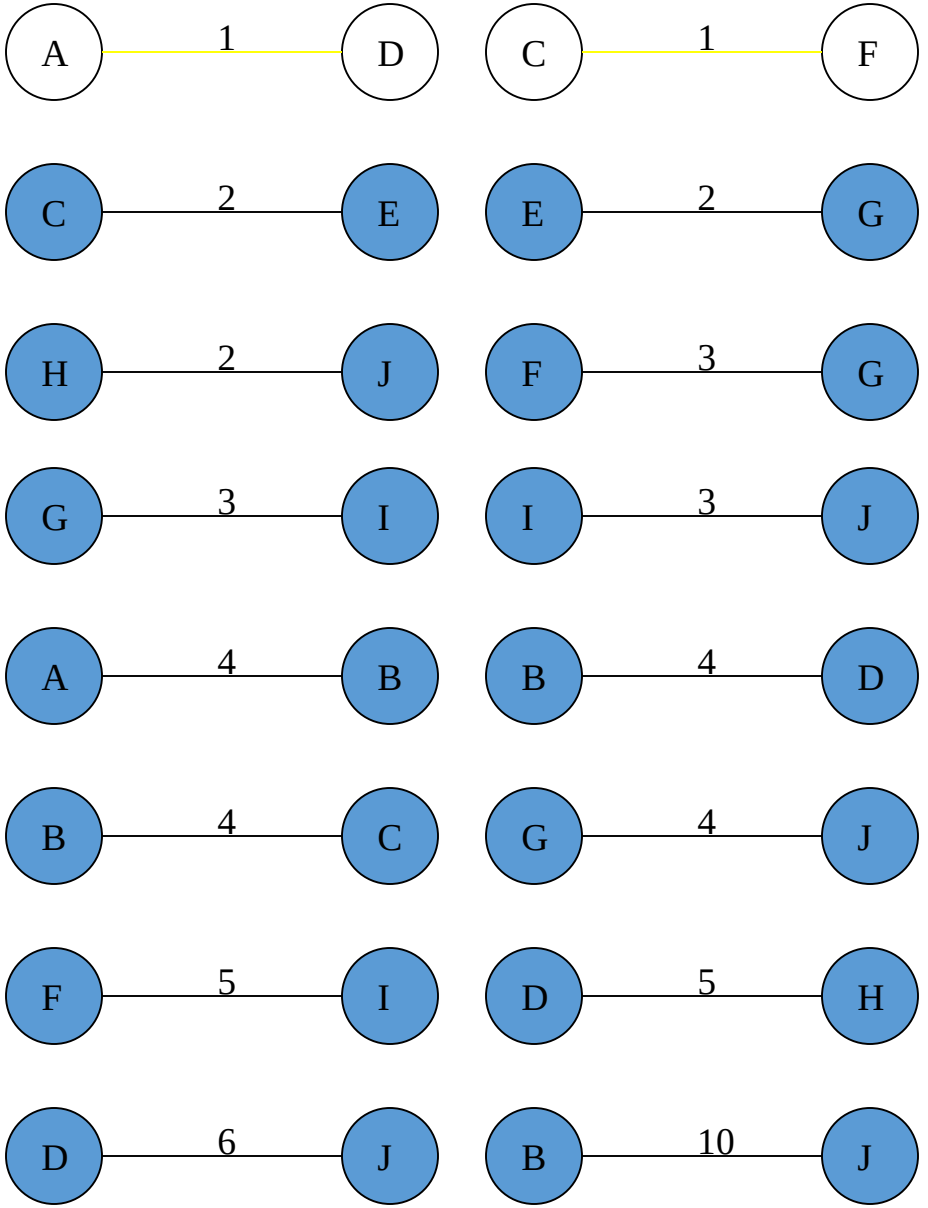
Add Edge


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

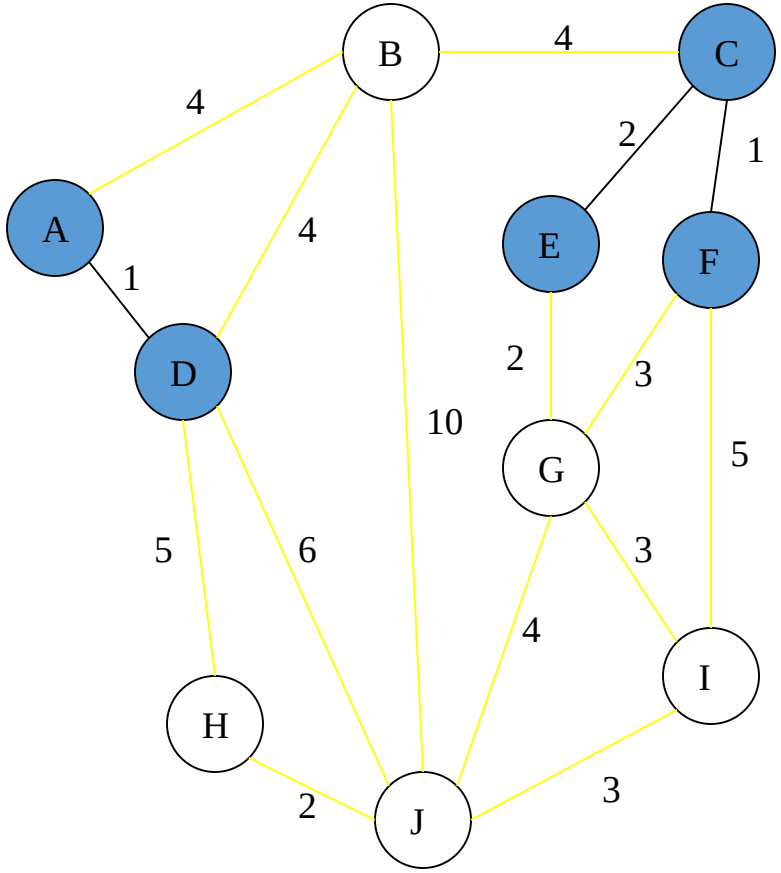

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

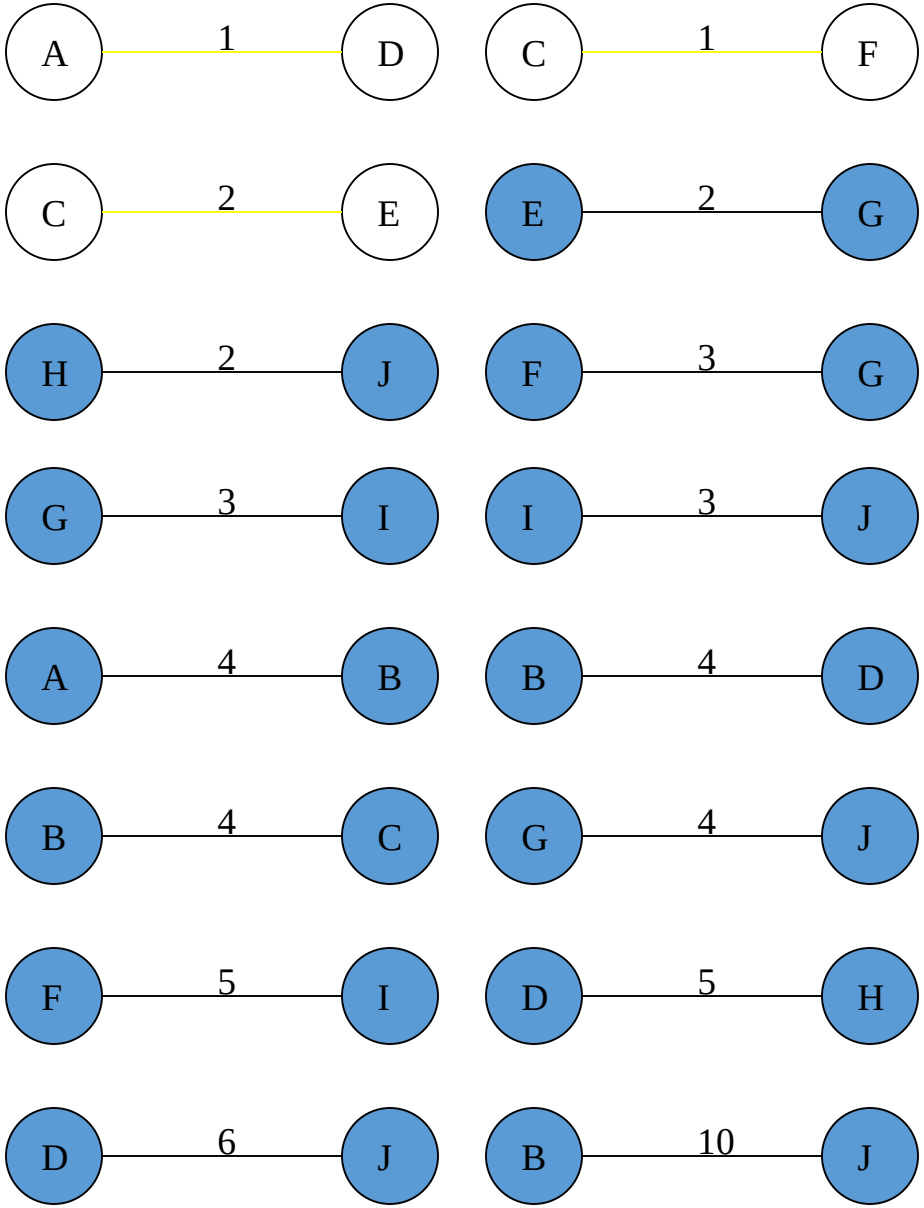
Add Edge


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

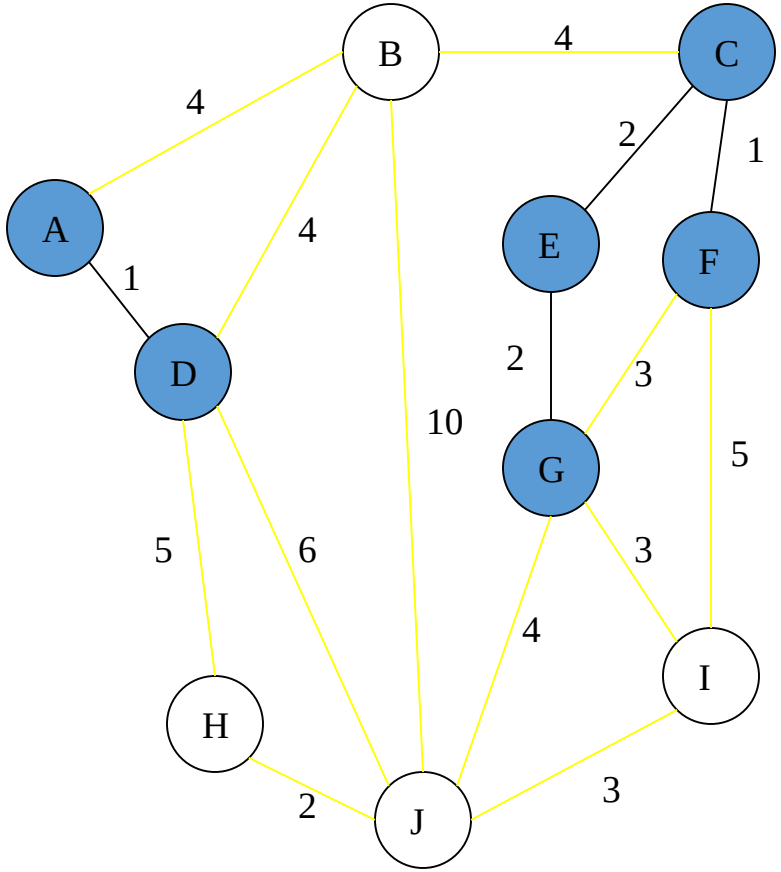

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

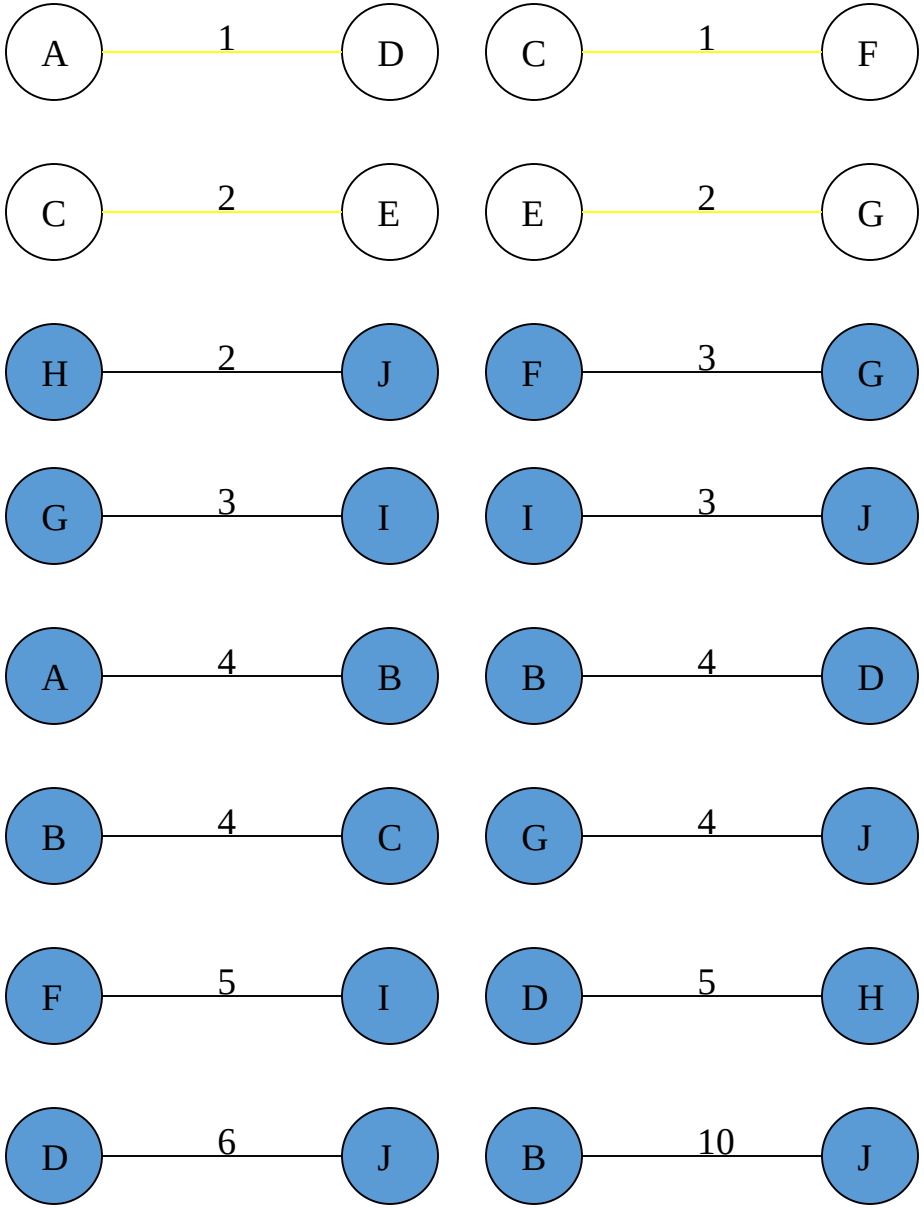
Add Edge


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

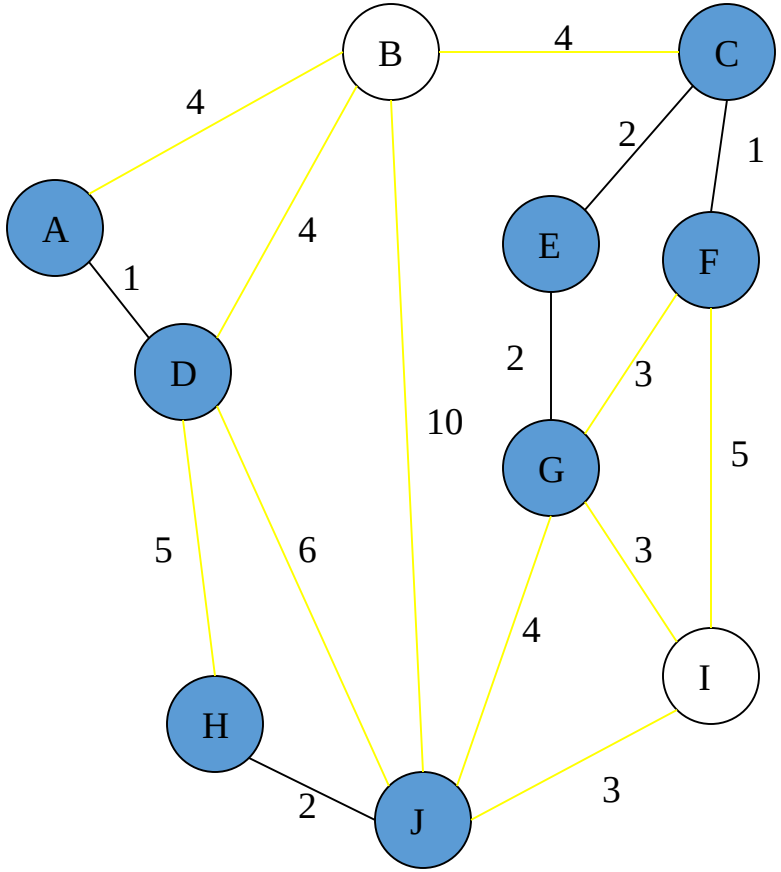

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

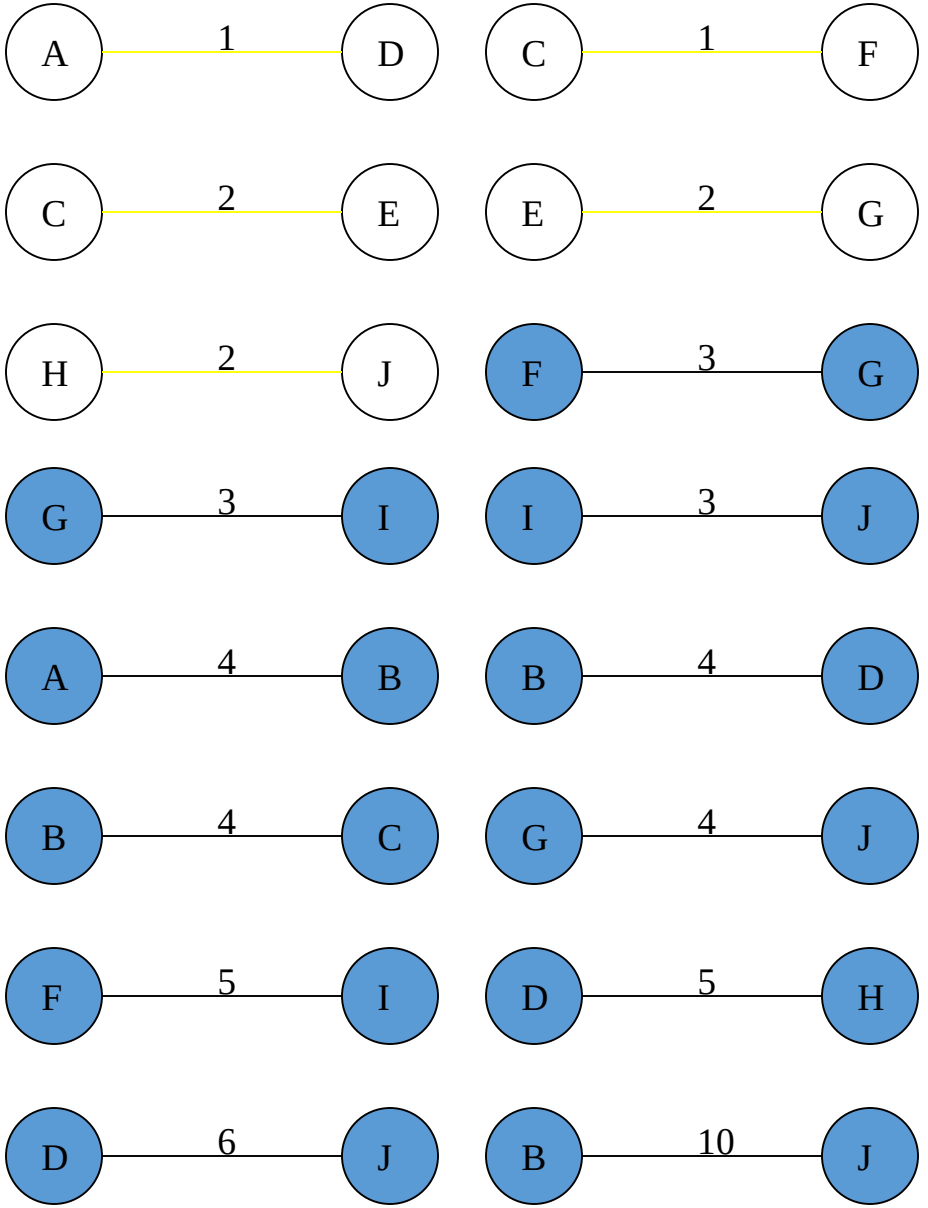
Add Edge


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

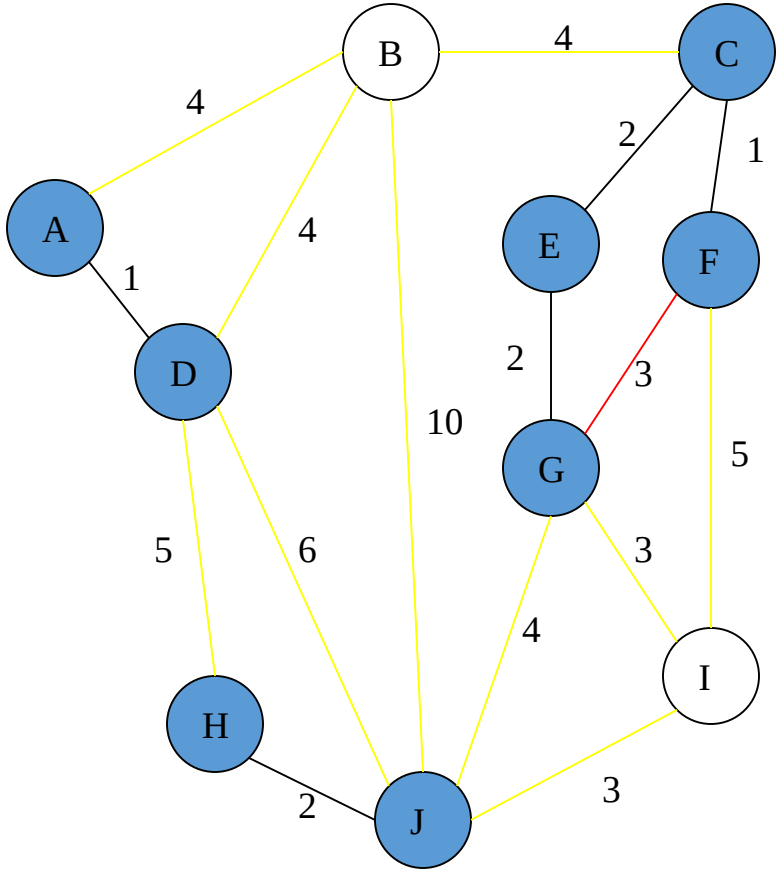
Add Edge

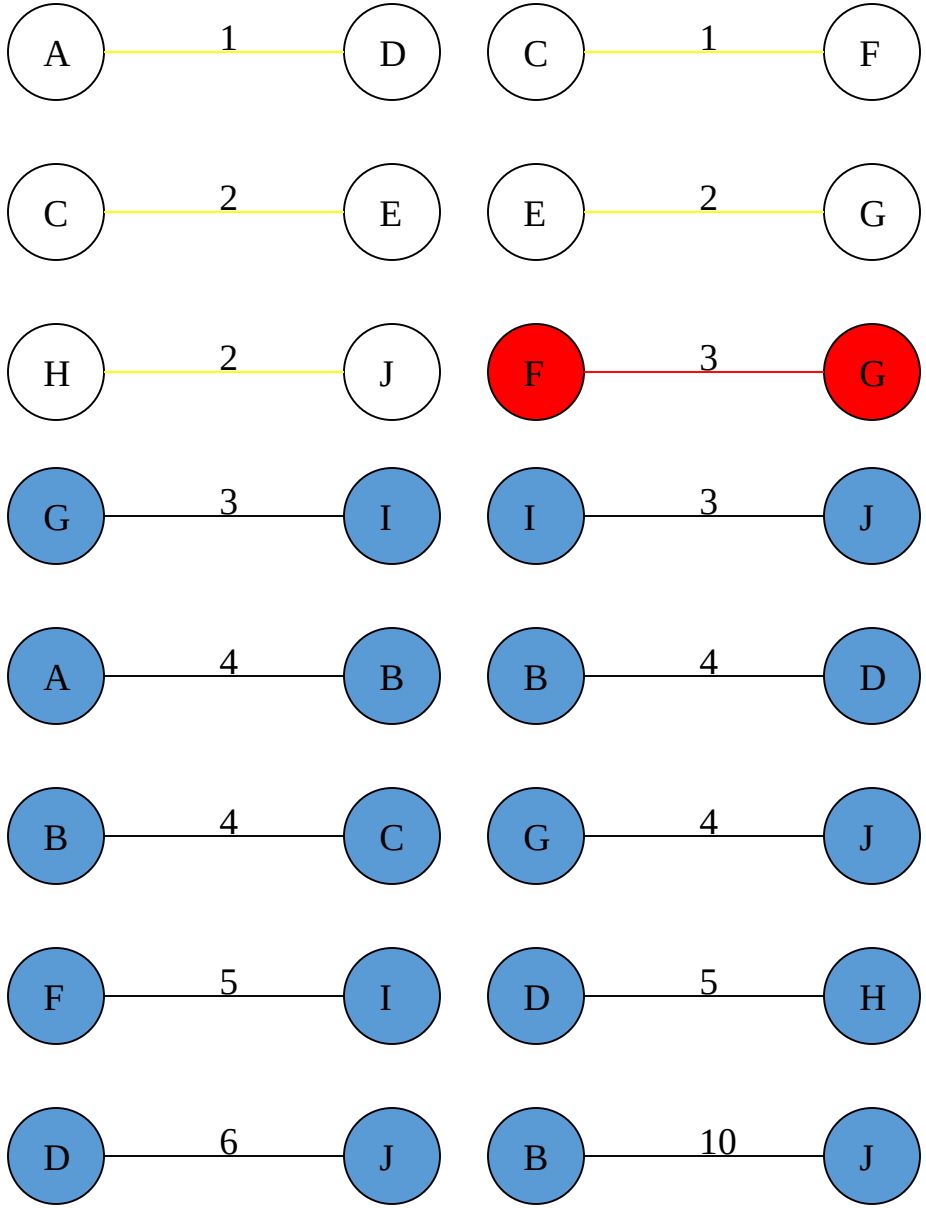

A
B
C
D
E
F
G
H
I
J
4
4
1
4
10
2
1
2
3
5
5
6
3
4
2
3


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

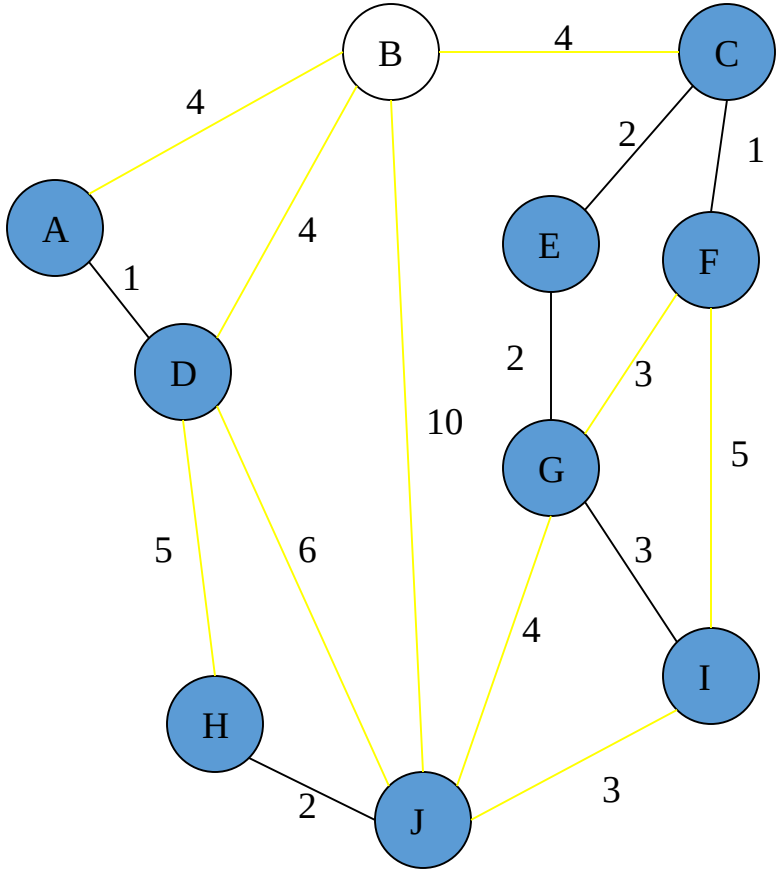
Cycle

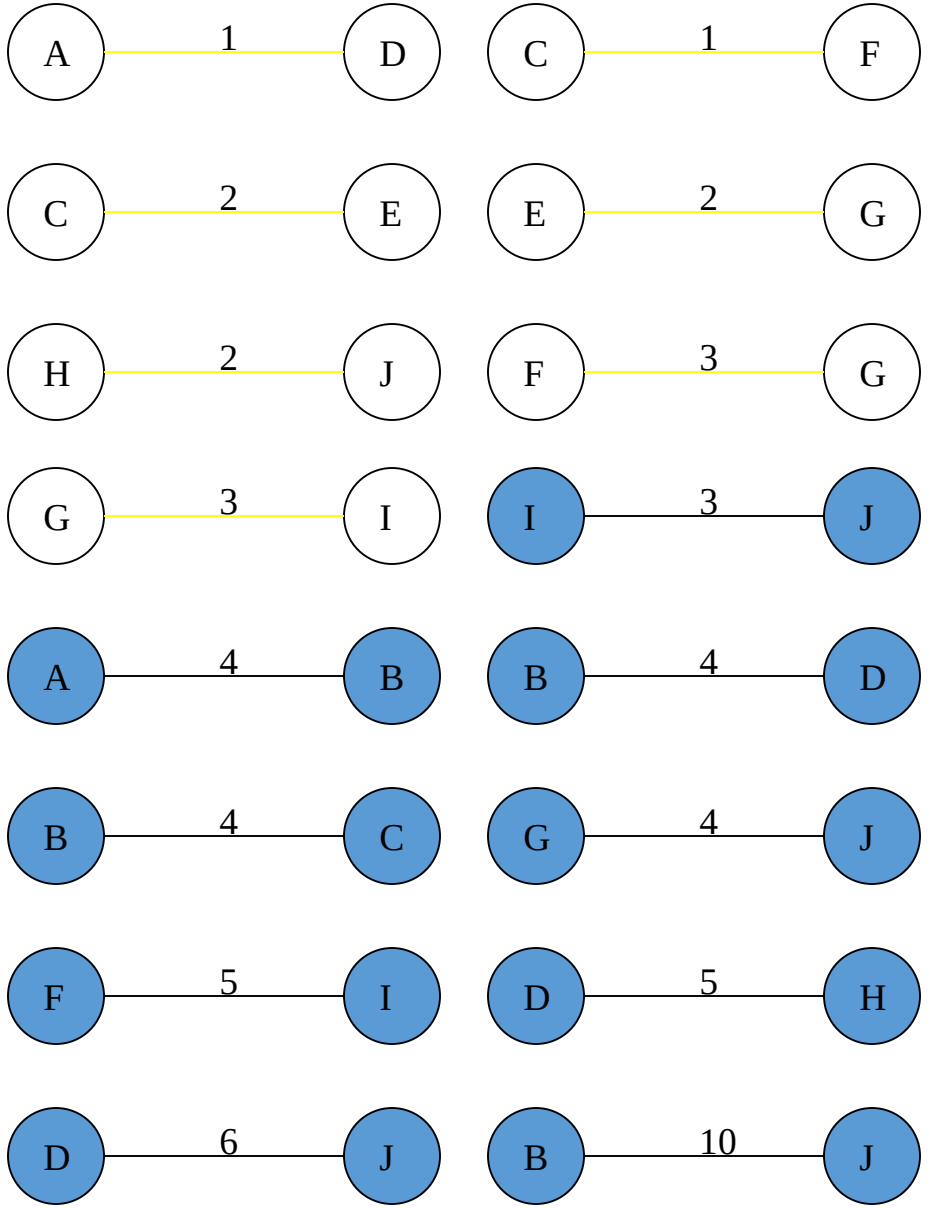
Don’t Add Edge



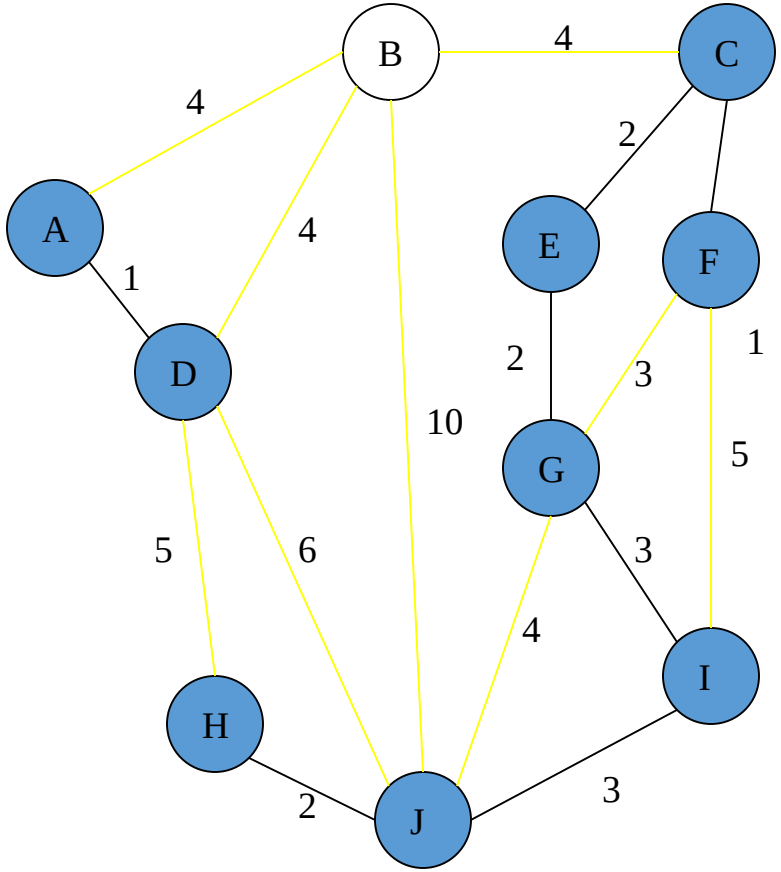


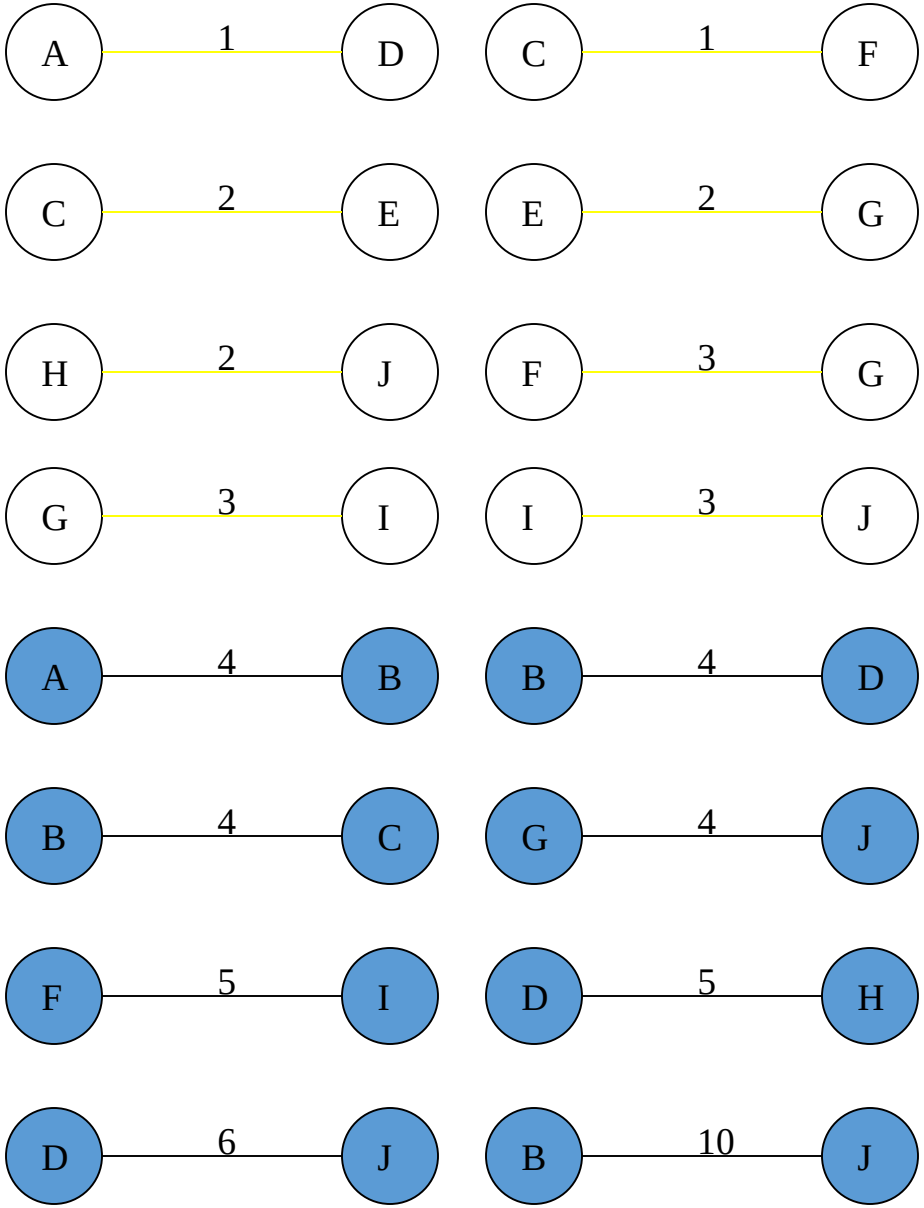
Add Edge


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

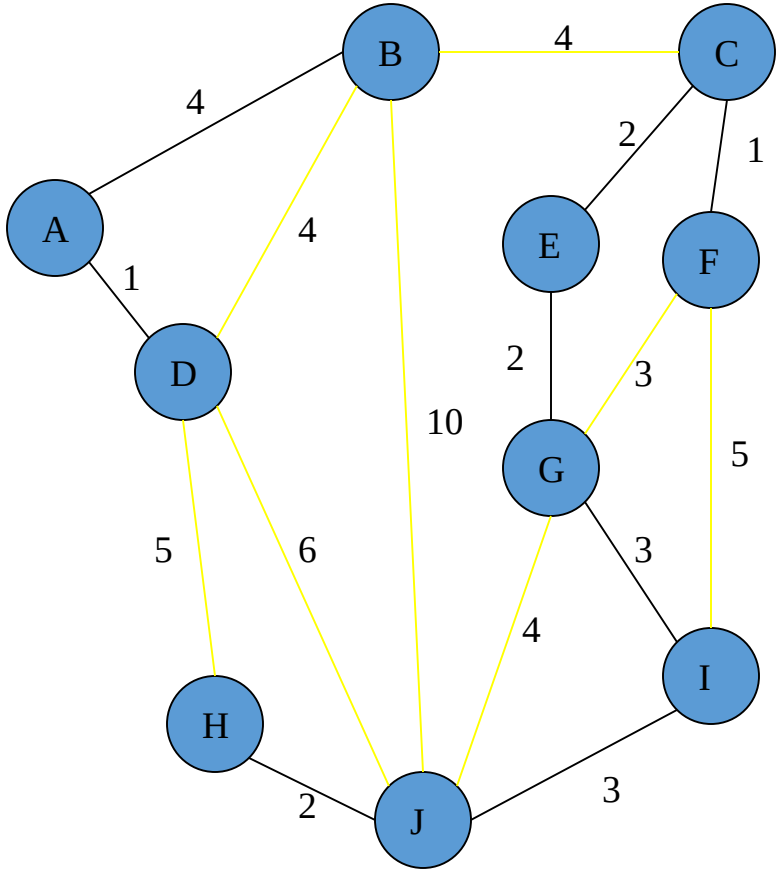

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

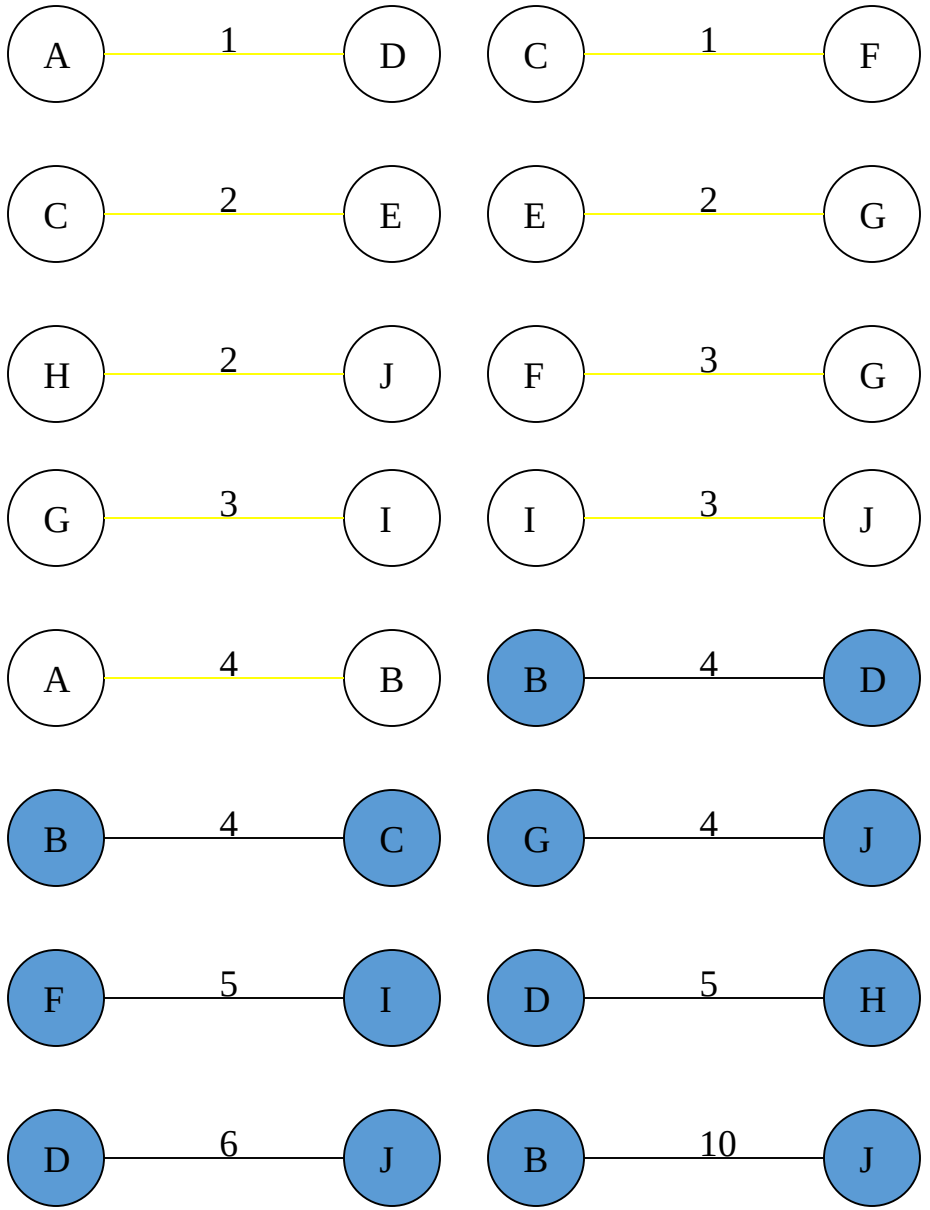
Add Edge


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

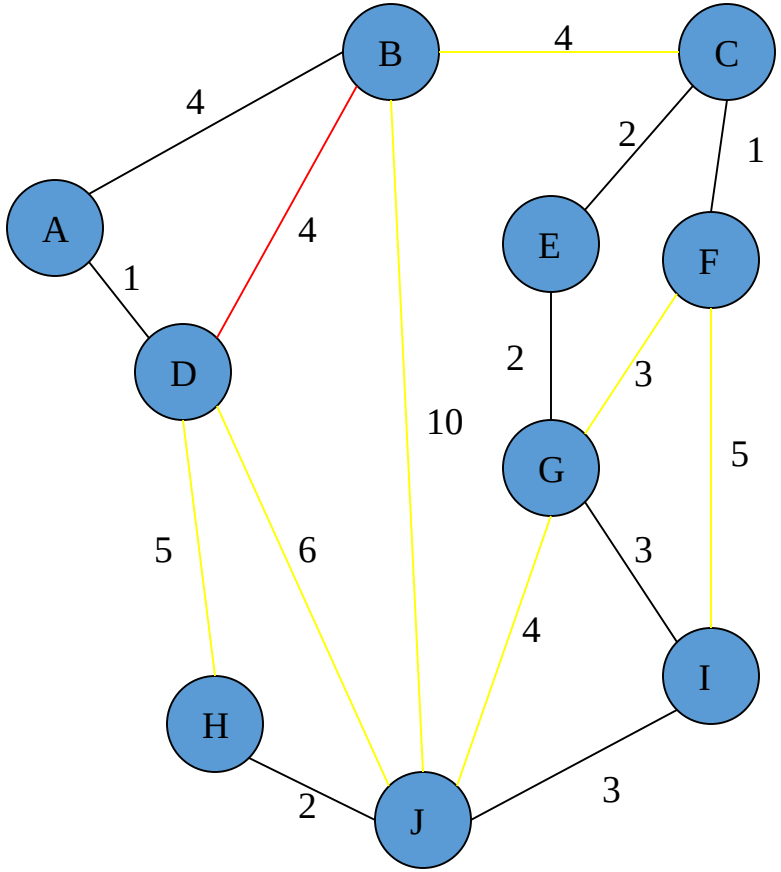
Add Edge

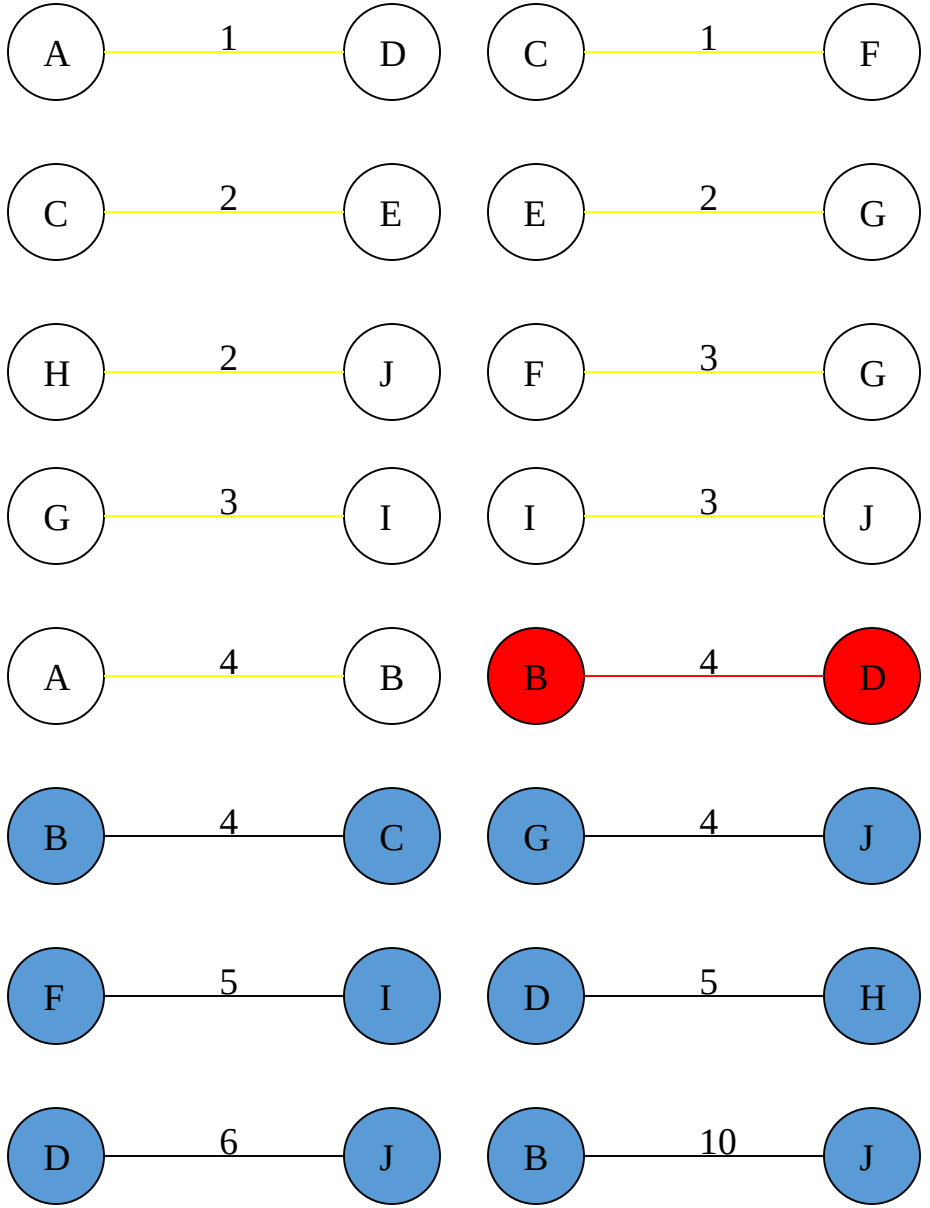

A
B
C
D
E
F
G
H
I
J
4
4
1
4
10
2
1
2
3
5
5
6
3
4
2
3


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

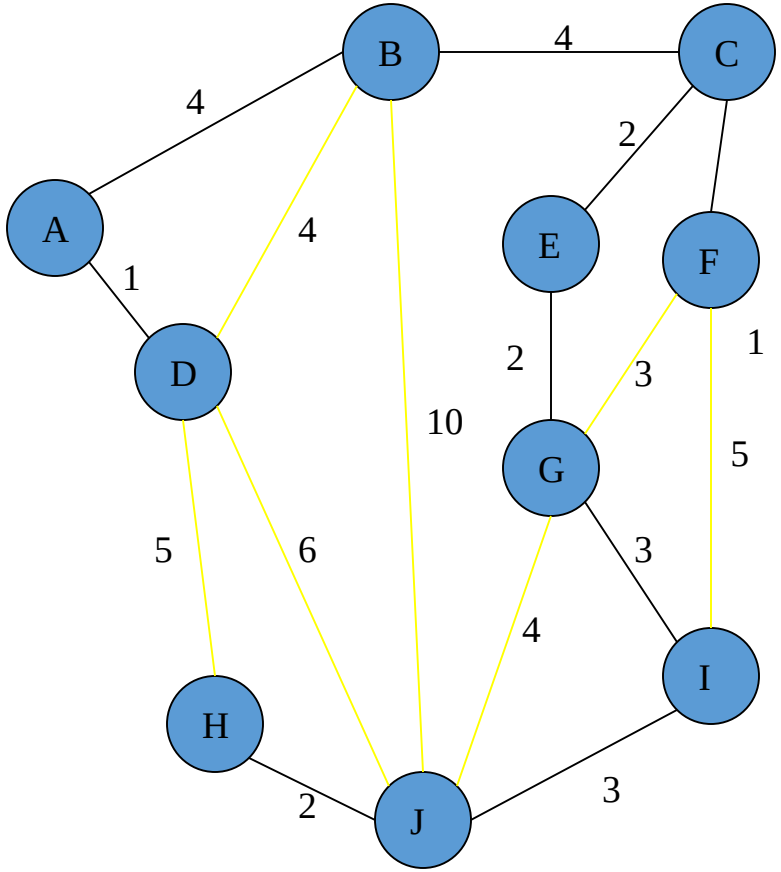
Cycle

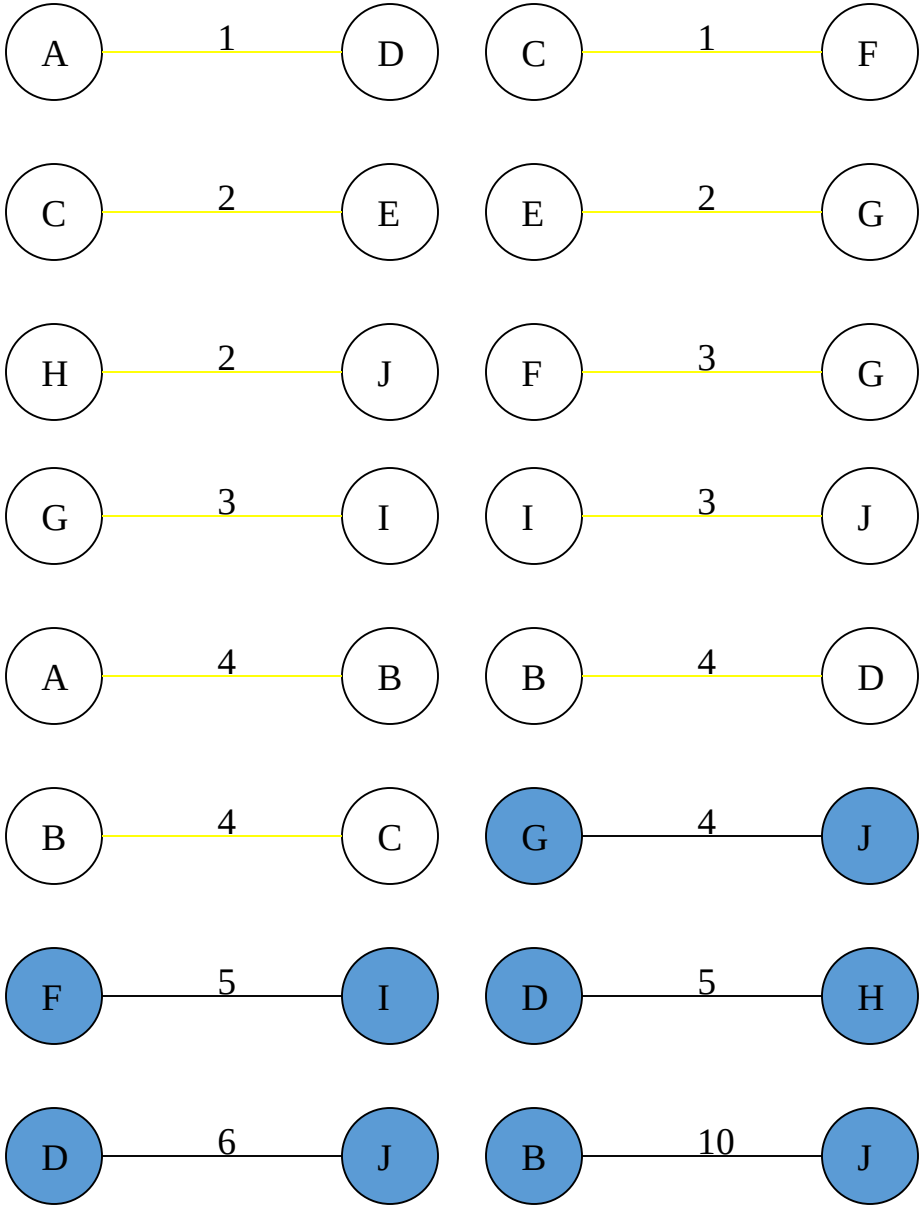
Don’t Add Edge



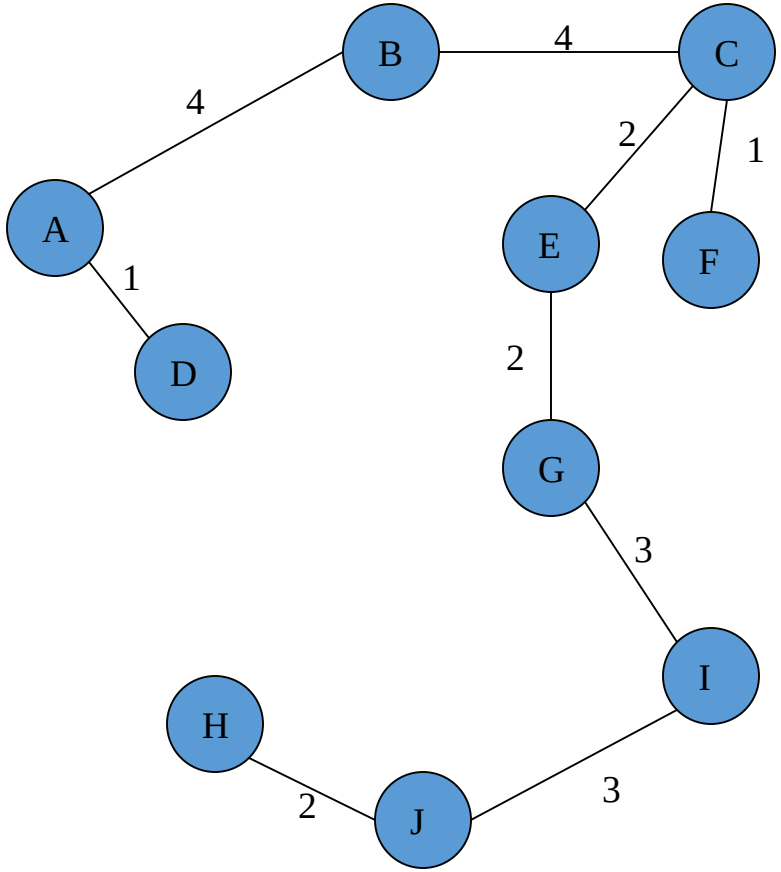


Add Edge

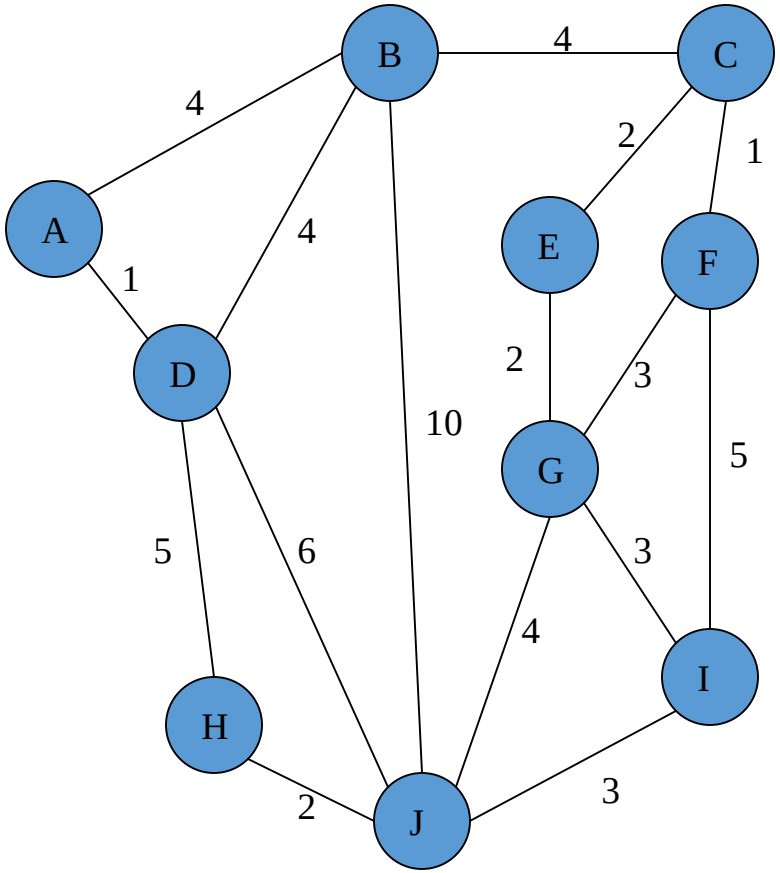

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
10
2
3
1
5
5
6
3
4
2
3


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



Prim's Algorithm

This algorithm **starts with one node**. It then, one by one, adds a node that is unconnected to the new graph. Each time it selects the node whose connecting edge has the smallest weight out of the available nodes' connecting edges.

Prim's Algorithm

primsAlgorithm(vertex v)

Mark vertex v as visited and include it in the MST

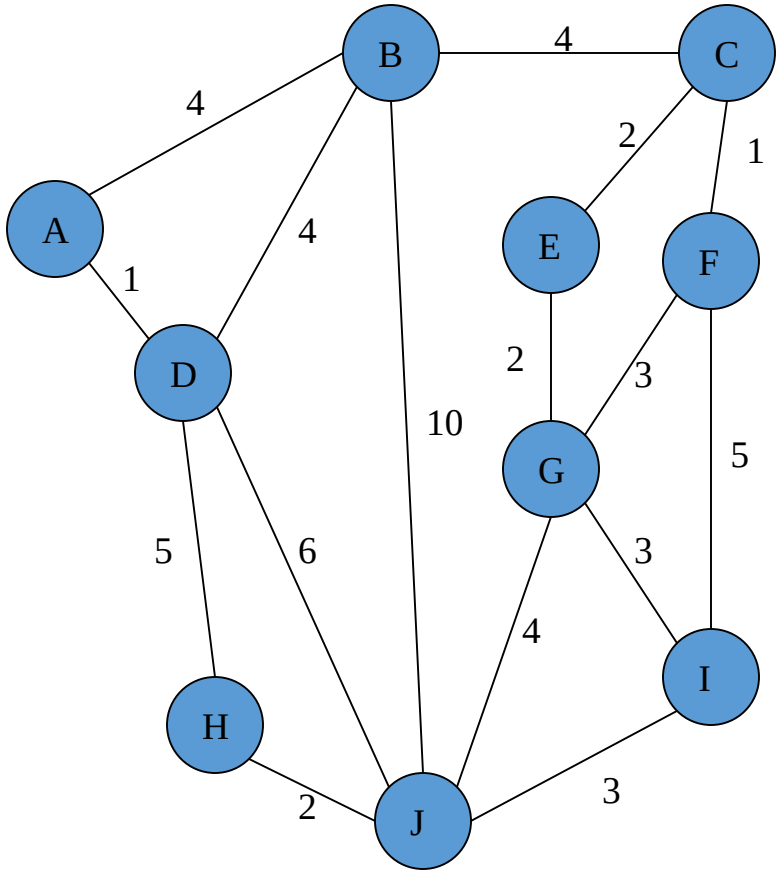
while(there are unvisited vertices)

Find the least-cost edge (v, u) from a visited vertex v to some unvisited vertex u

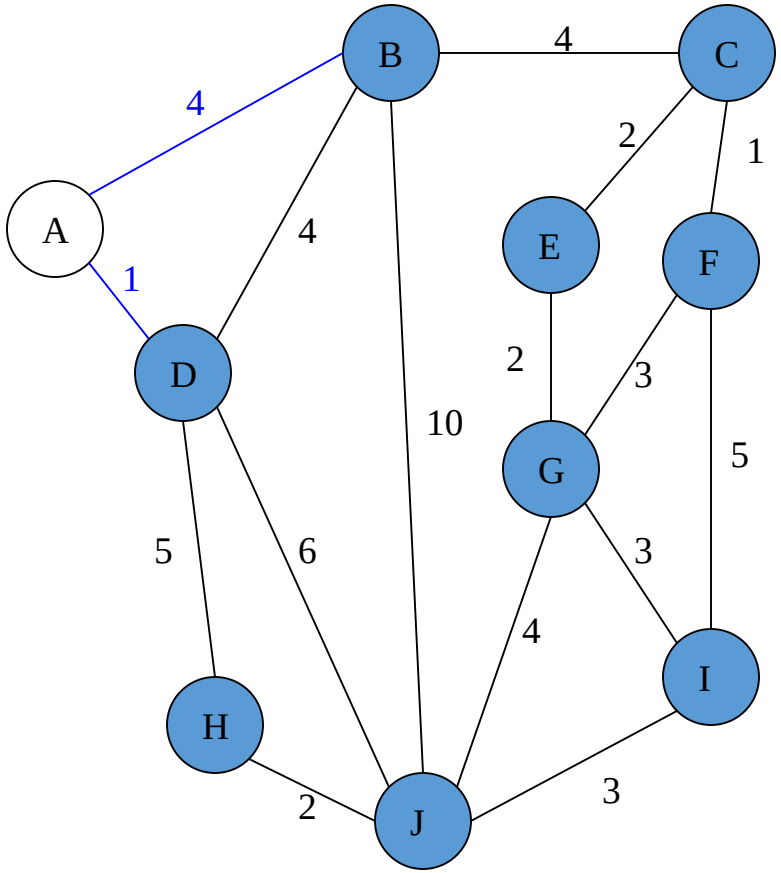
Mark u as visited

Add the vertex u and the edge (v, u) to the MST

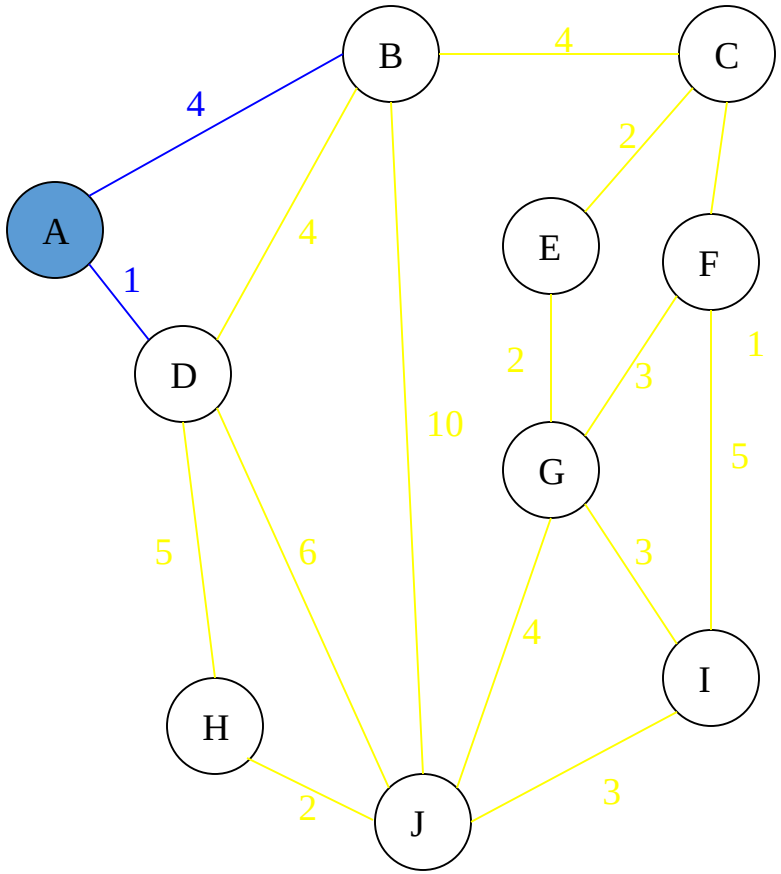
Complete Graph



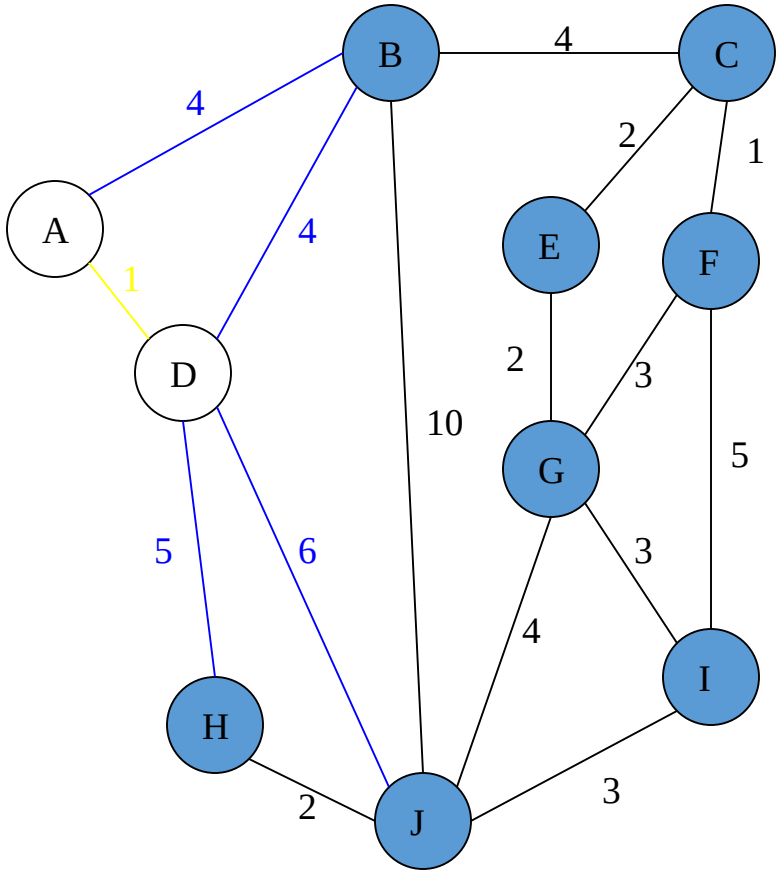
Old Graph



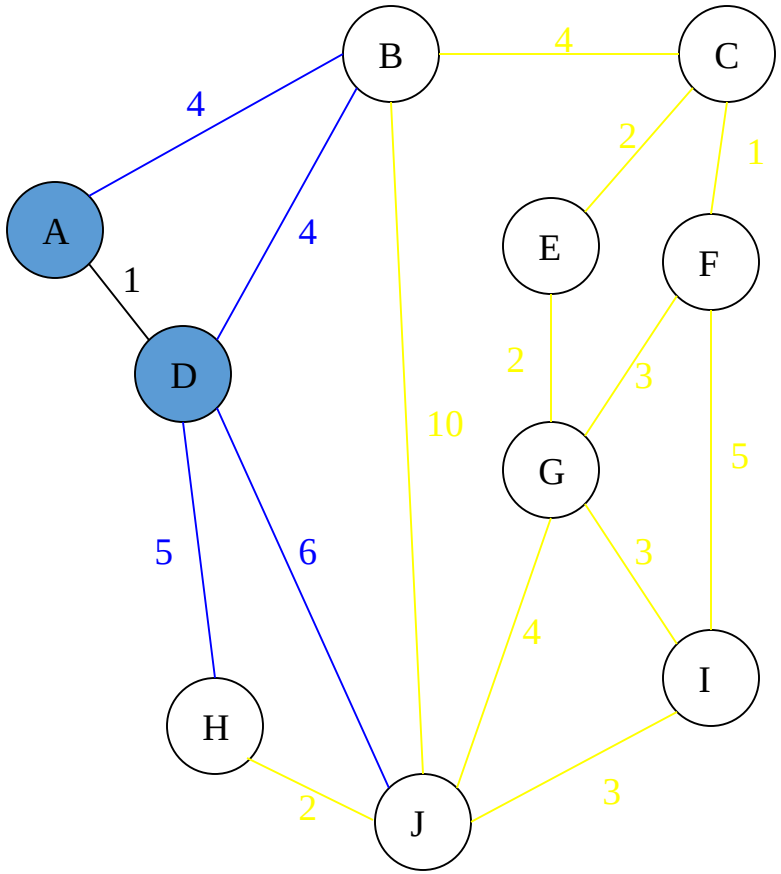
New Graph



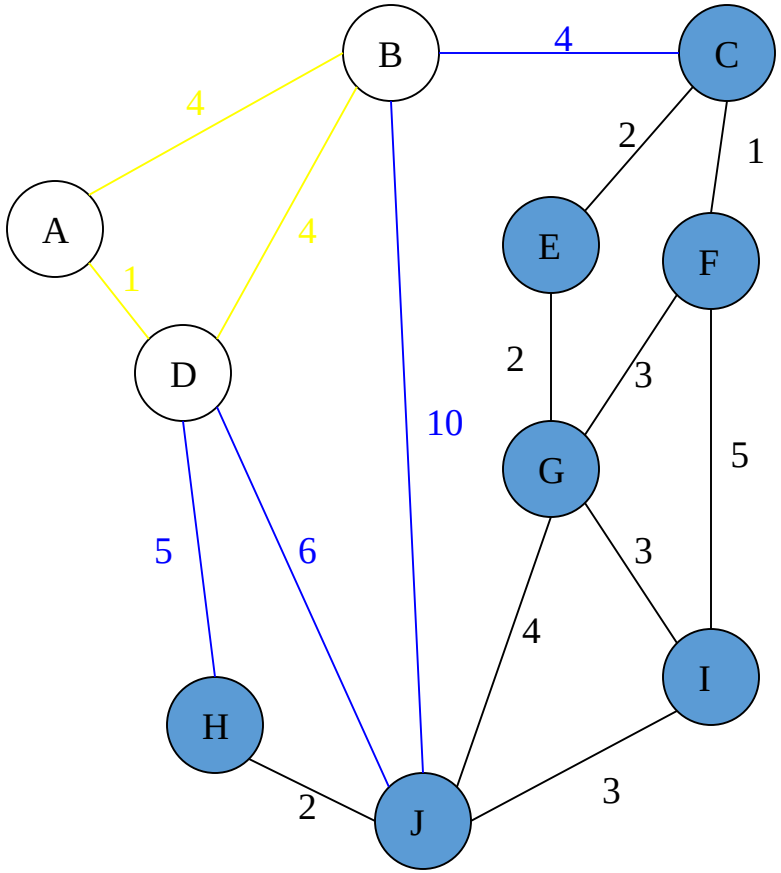
Old Graph



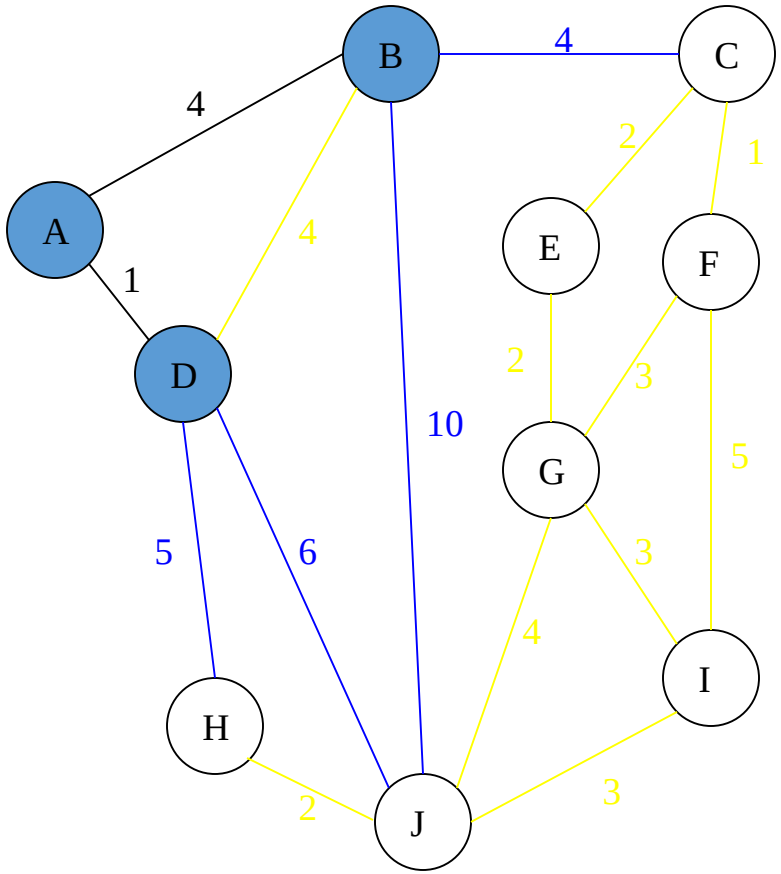
New Graph



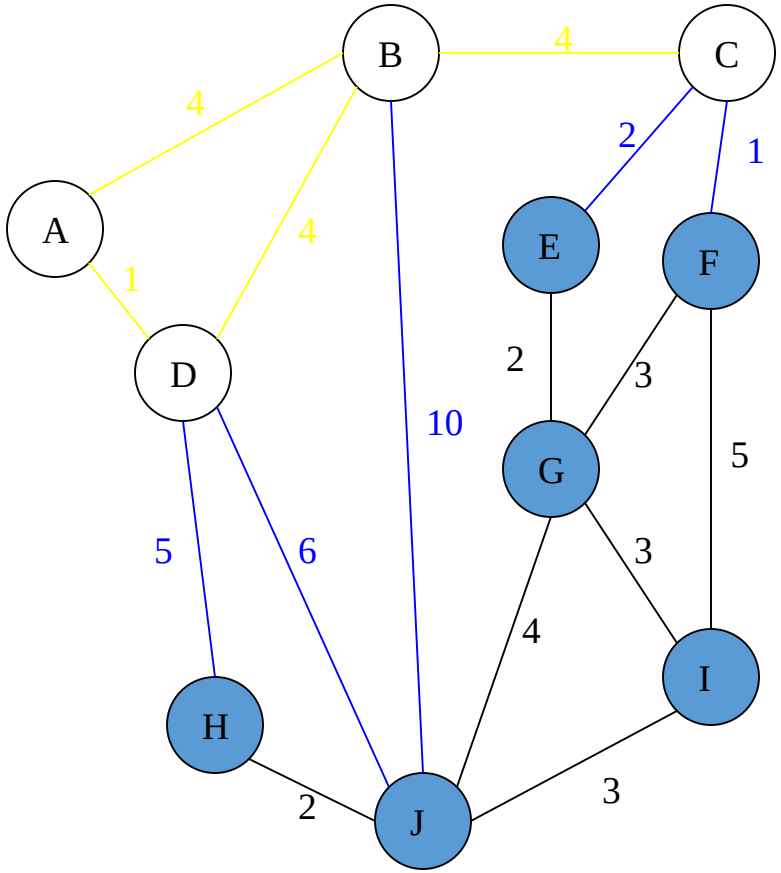
Old Graph



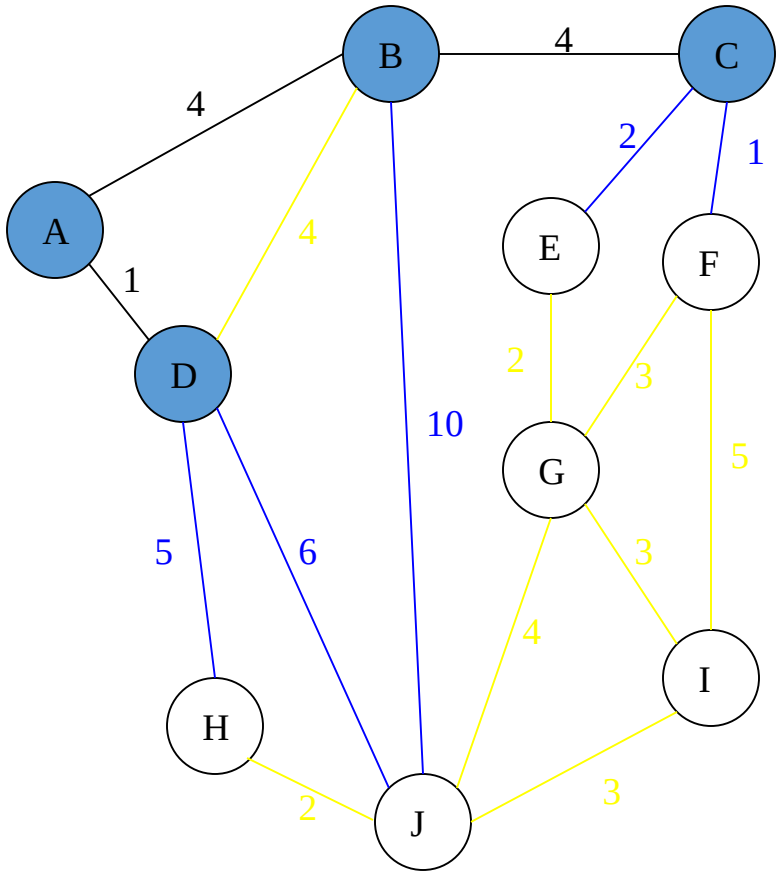
New Graph



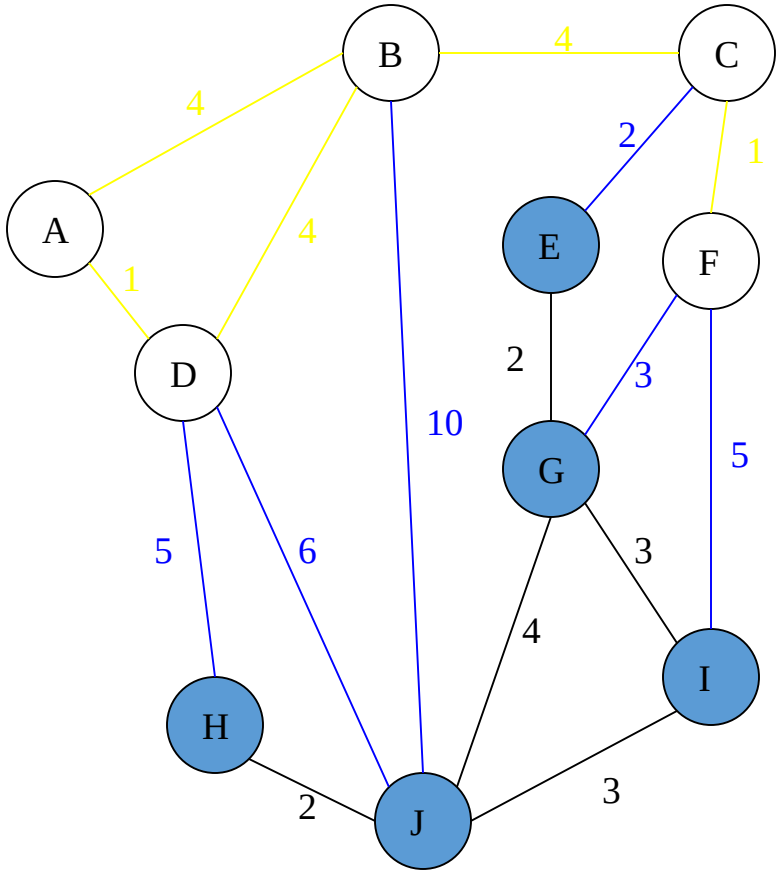
Old Graph



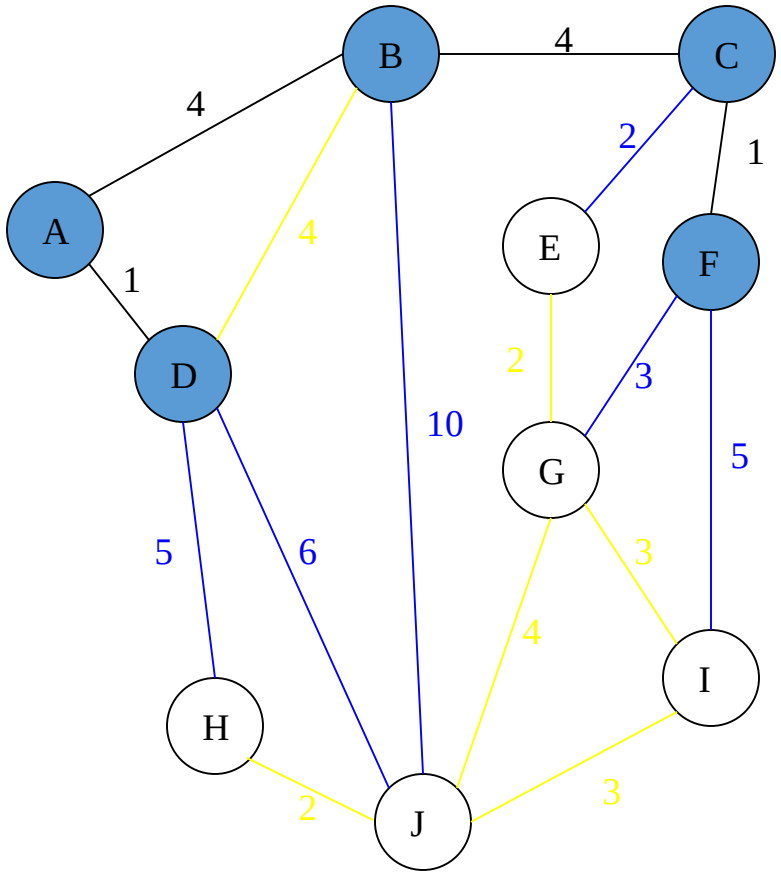
New Graph



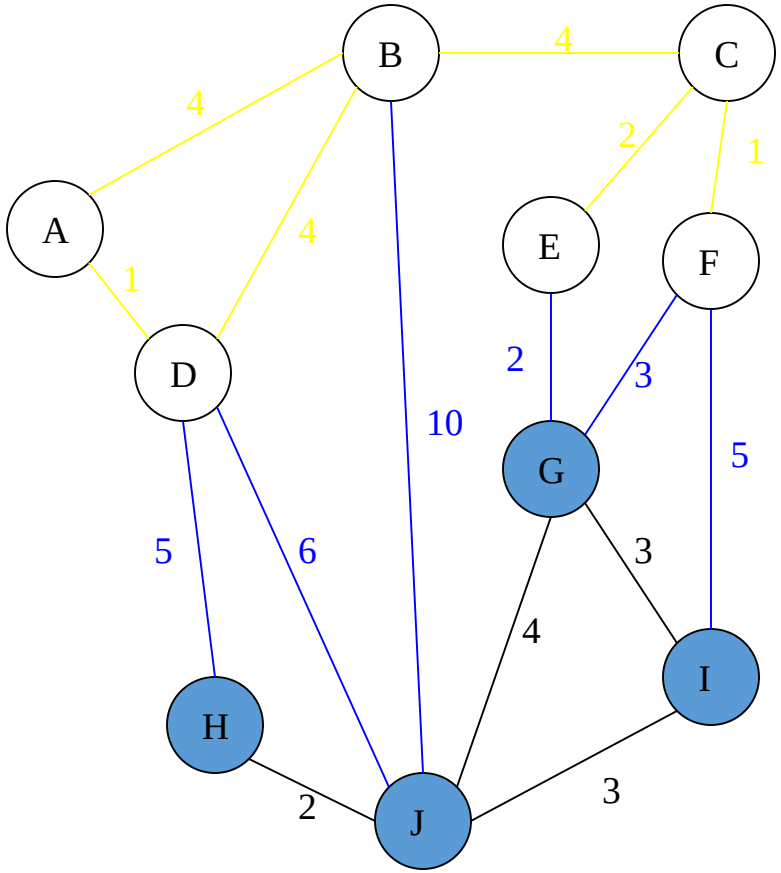
Old Graph



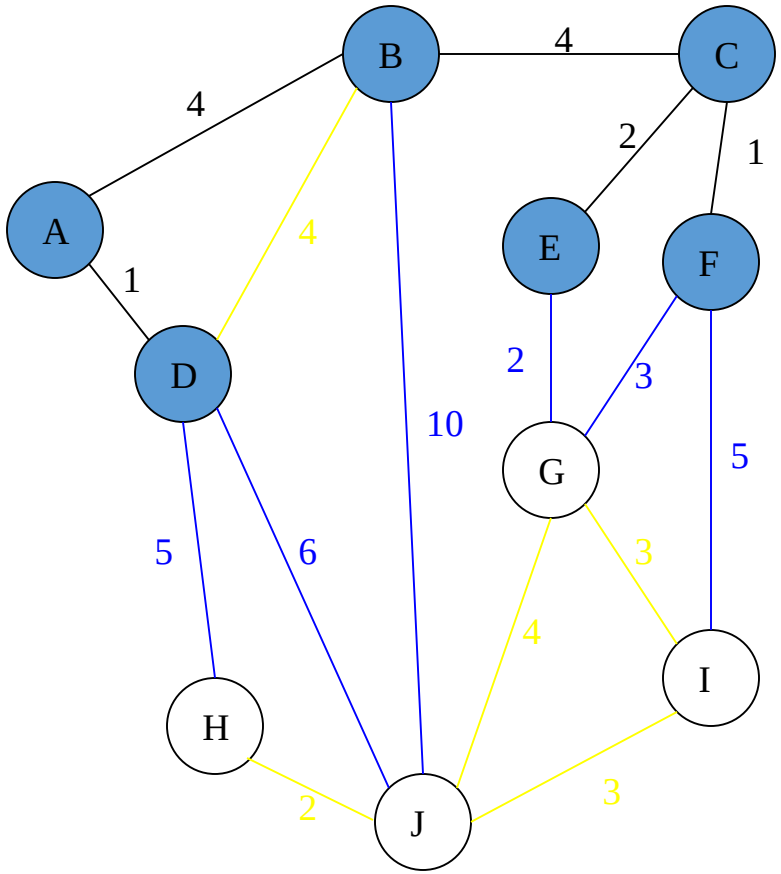
New Graph



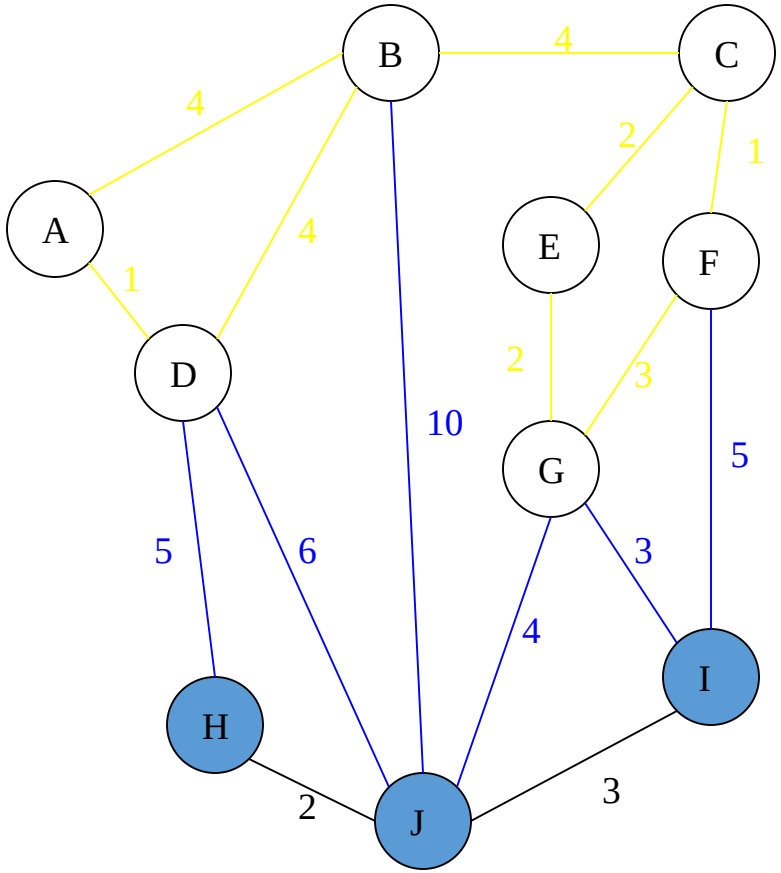
Old Graph



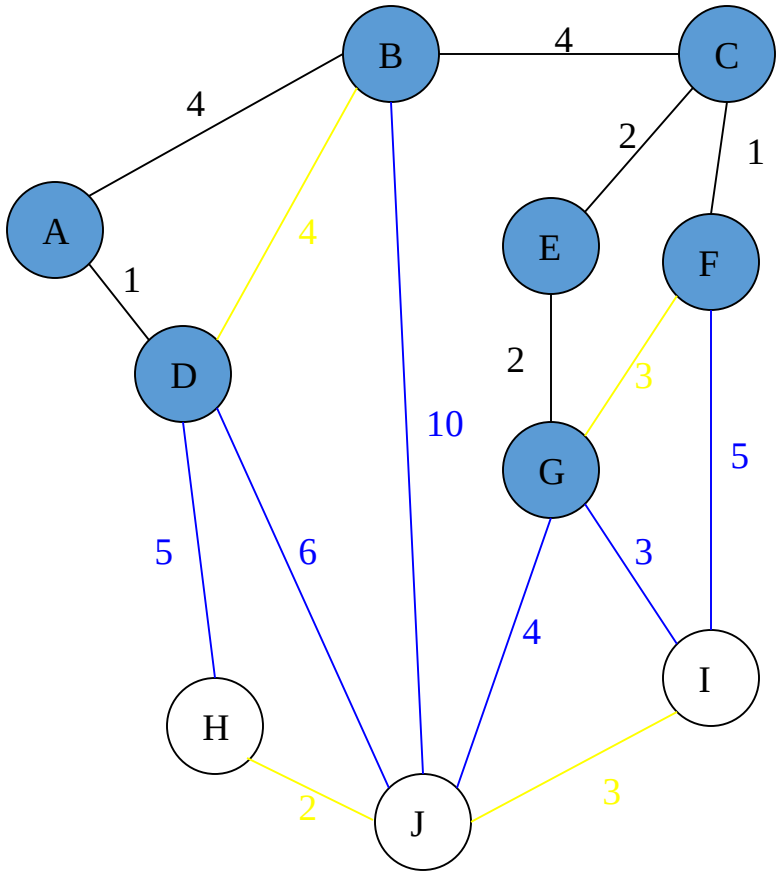
New Graph



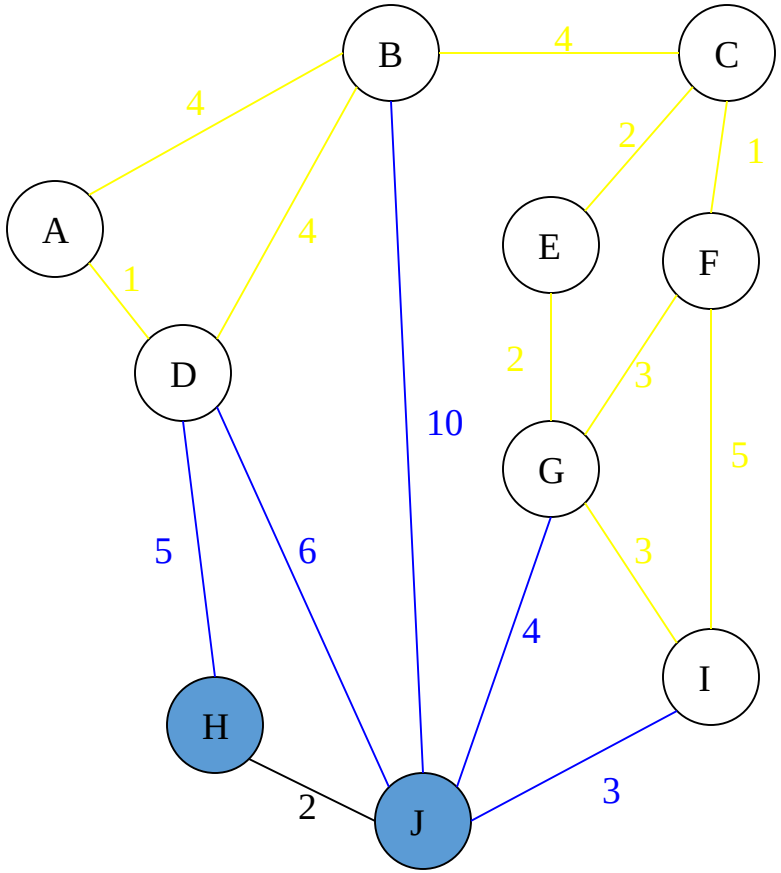
Old Graph



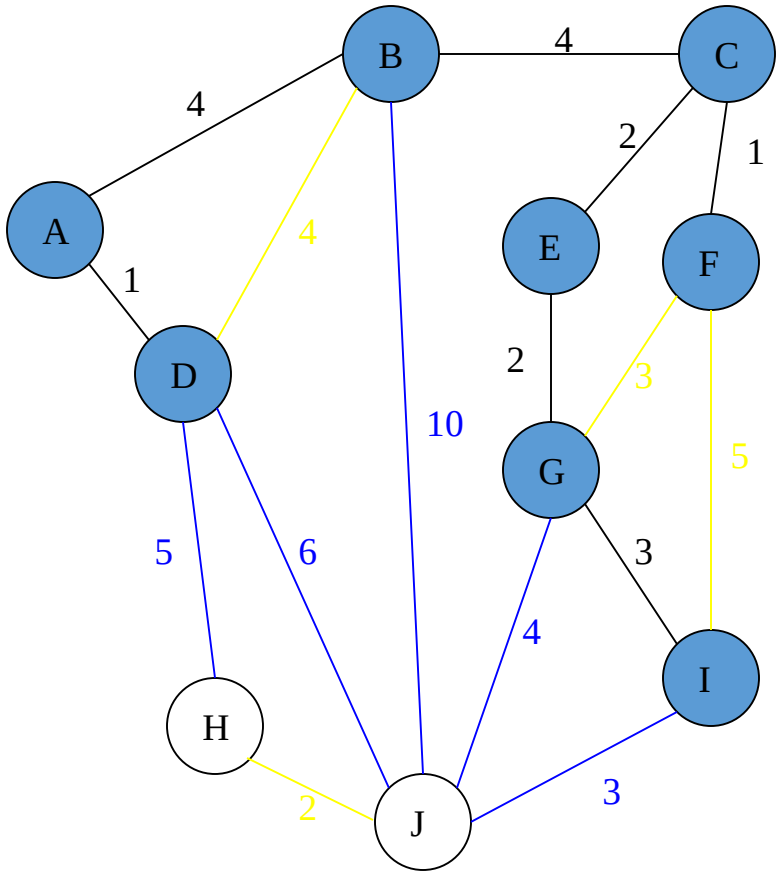
New Graph



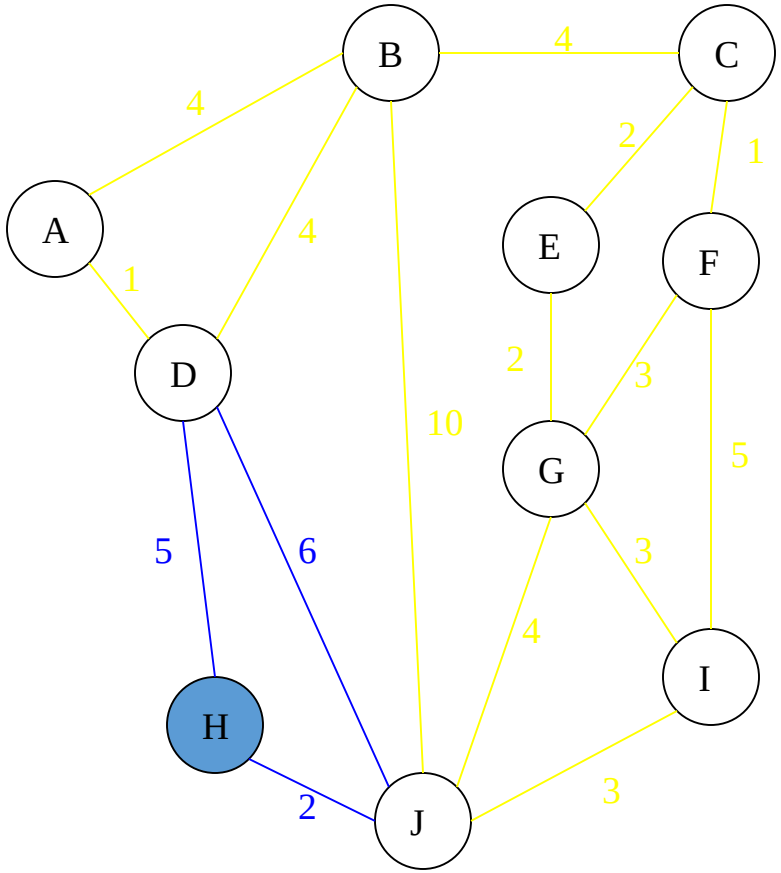
Old Graph



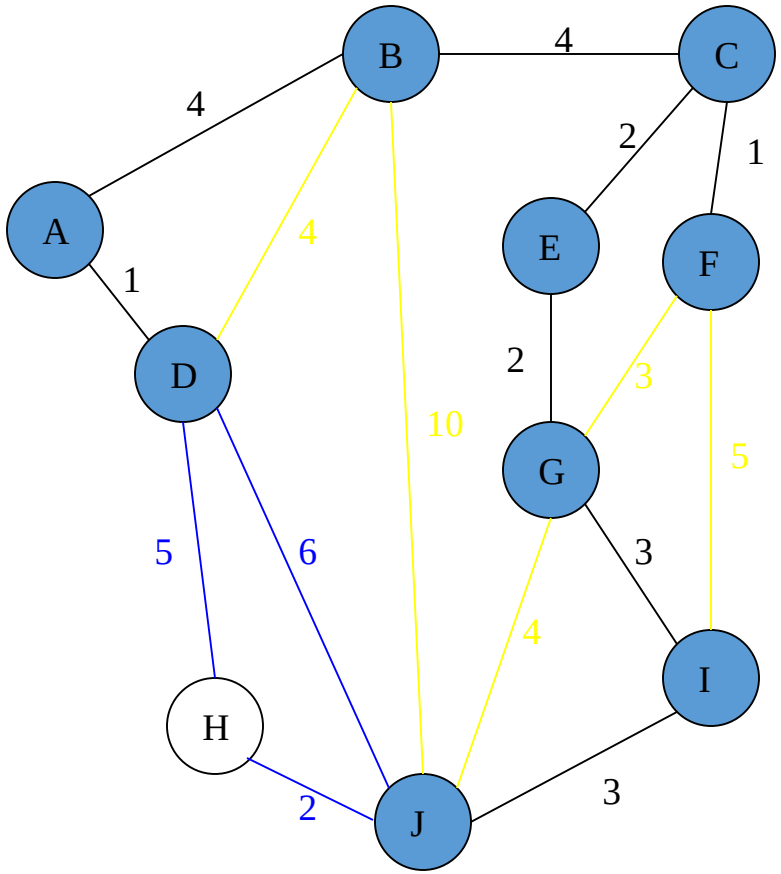
New Graph



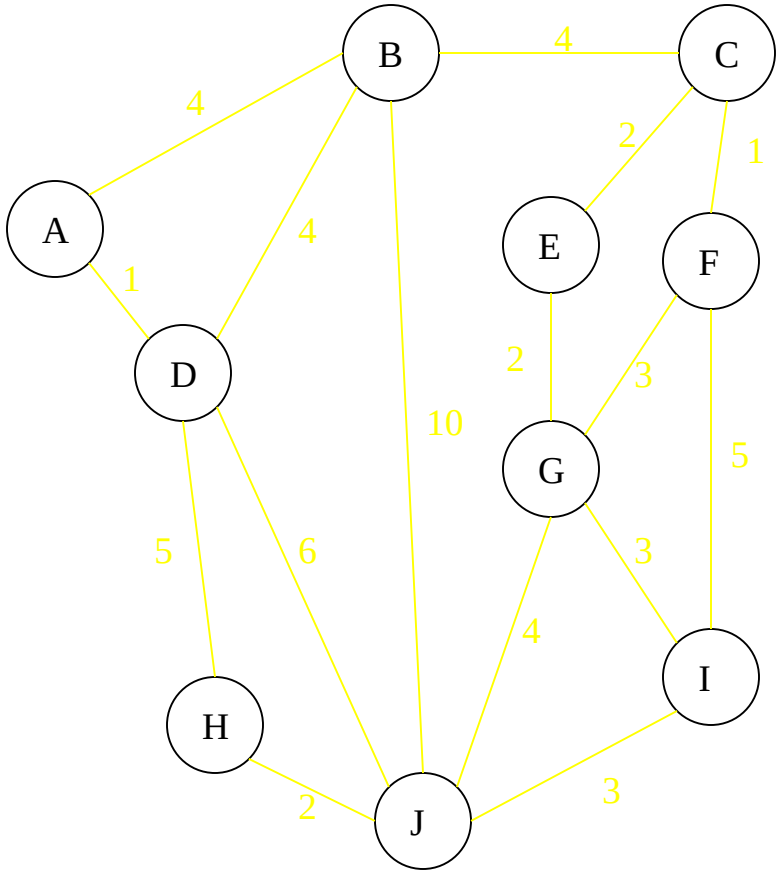
Old Graph



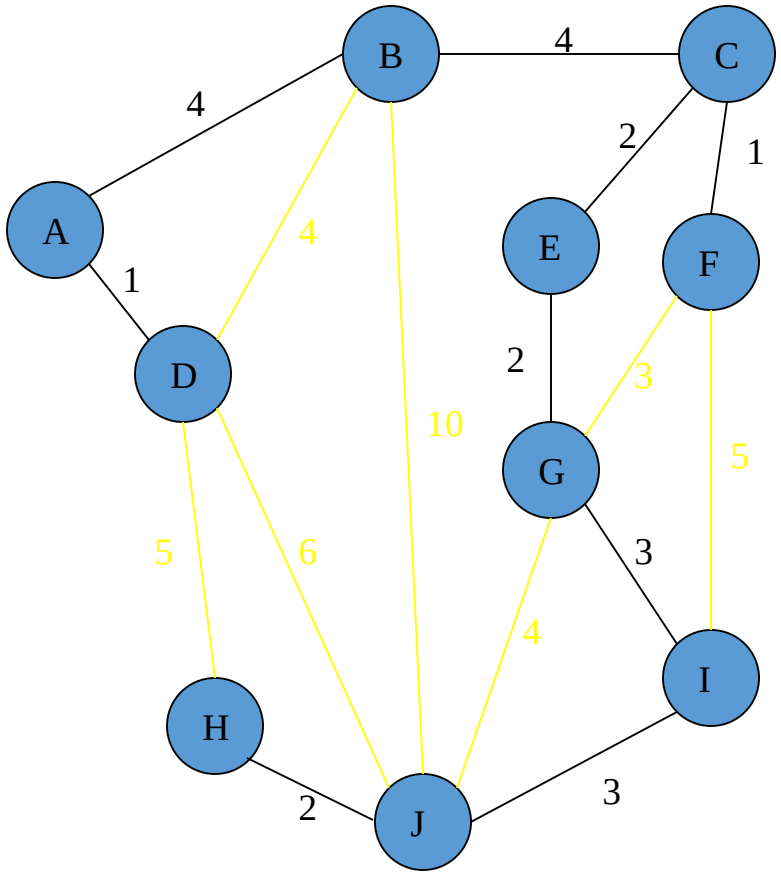
New Graph



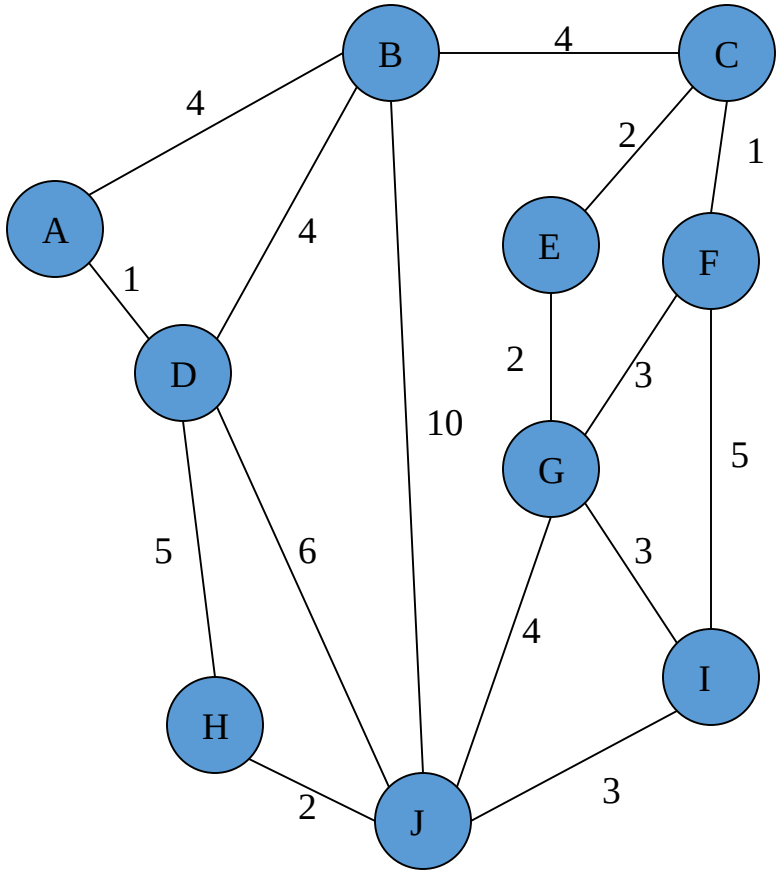
Old Graph



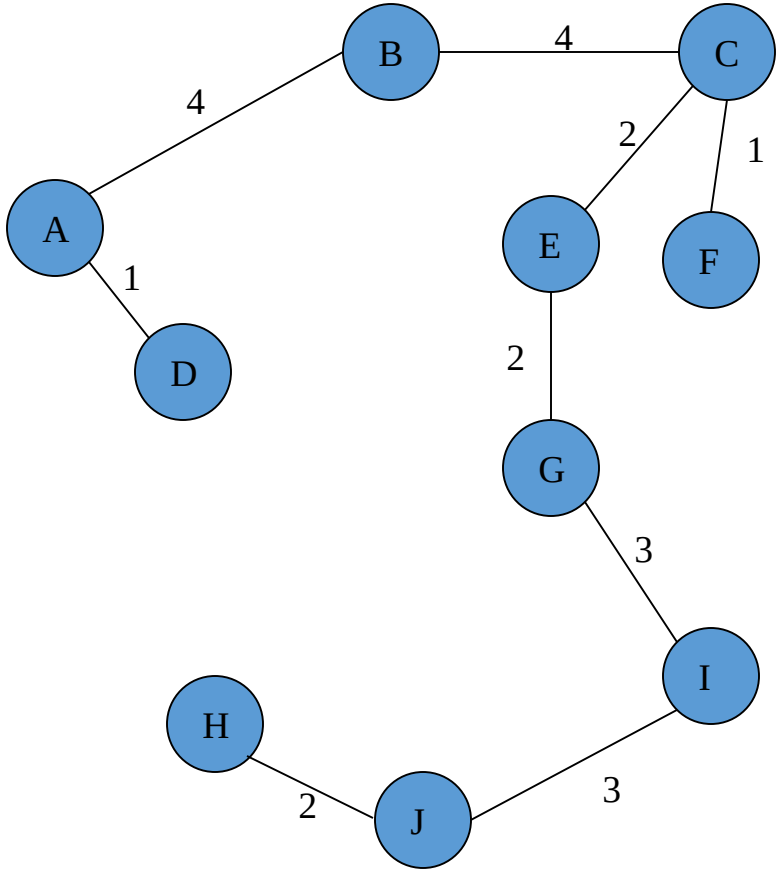
New Graph



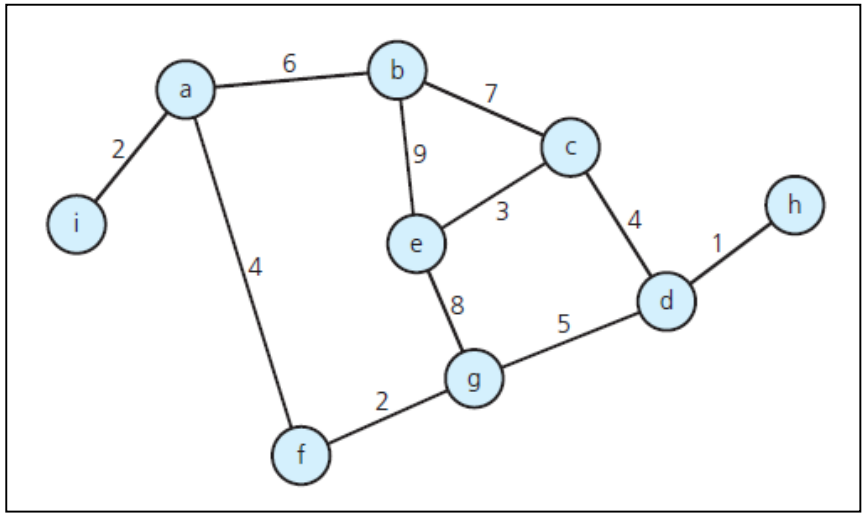
Complete Graph

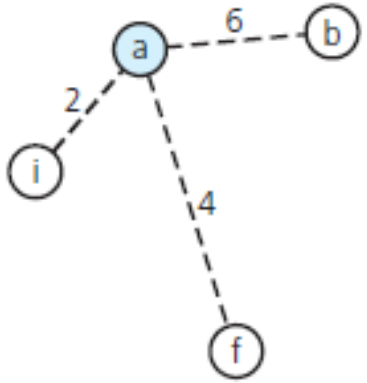


Minimum Spanning Tree

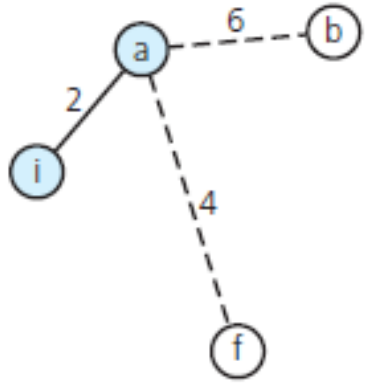


Exercise: Using Prim's algorithm find the minimum spanning tree starting from node a

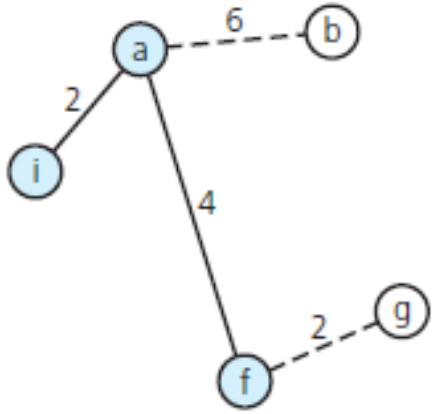




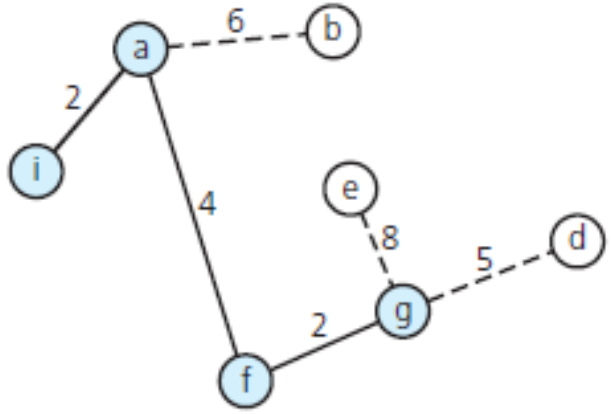
(a) Mark a, consider edges from a



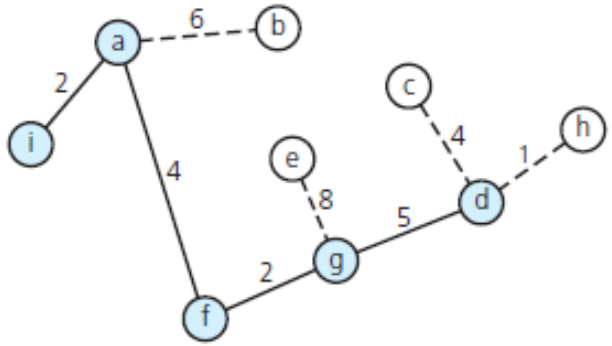
(b) Mark i, include edge (a, i)



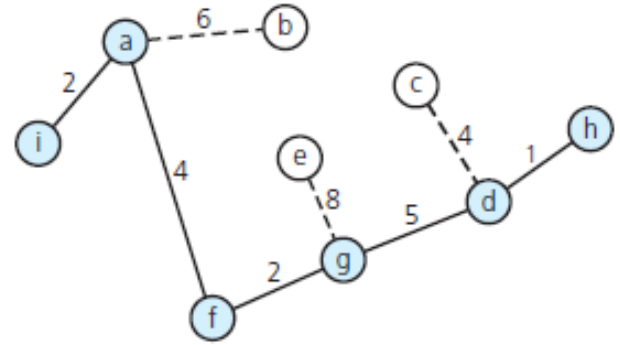
(c) Mark f, include edge (a, f)



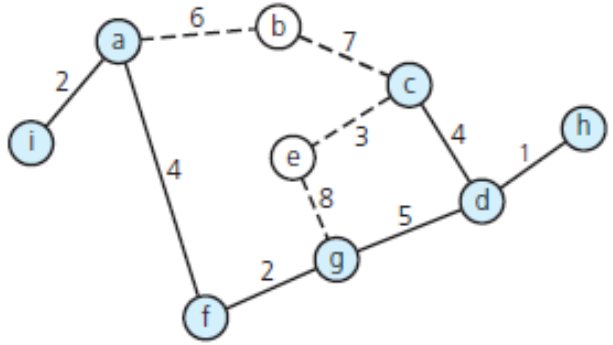
(d) Mark g, include edge (f, g)



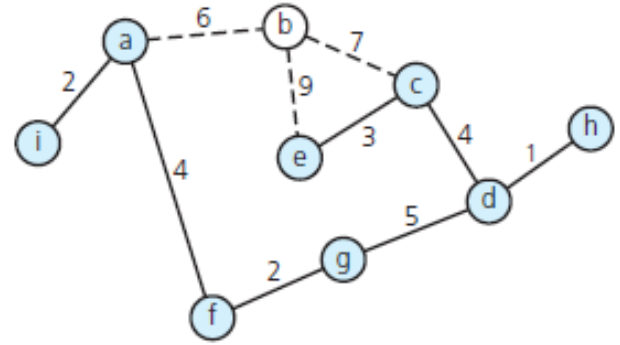
(e) Mark d, include edge (g, d)



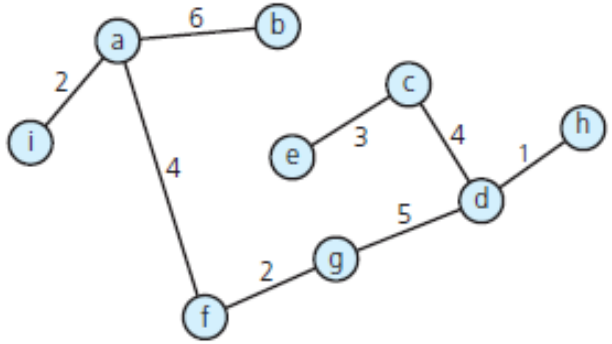
(f) Mark h, include edge (d, h)



(g) Mark c, include edge (d, c)

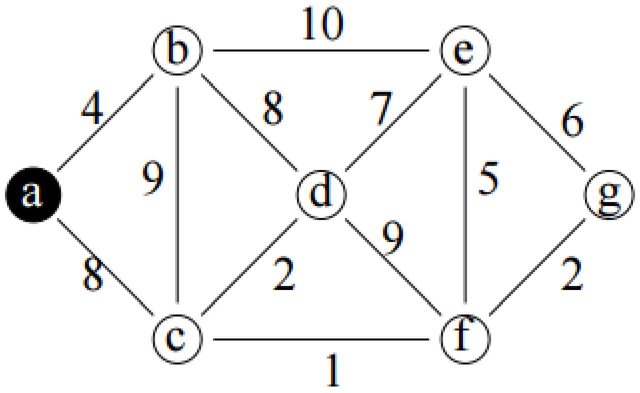


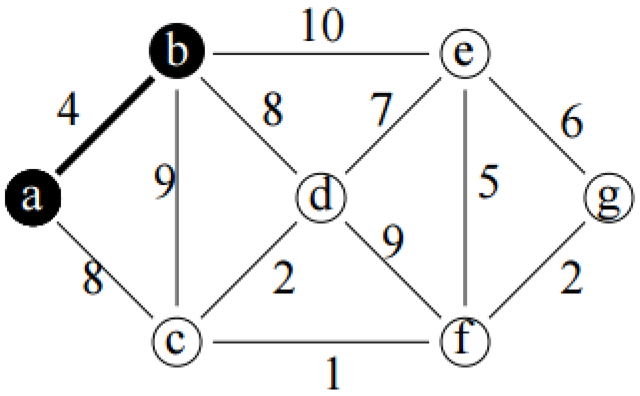
(h) Mark e, include edge (c, e)

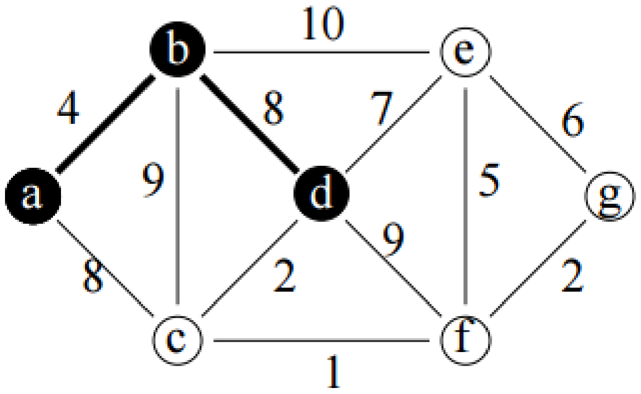


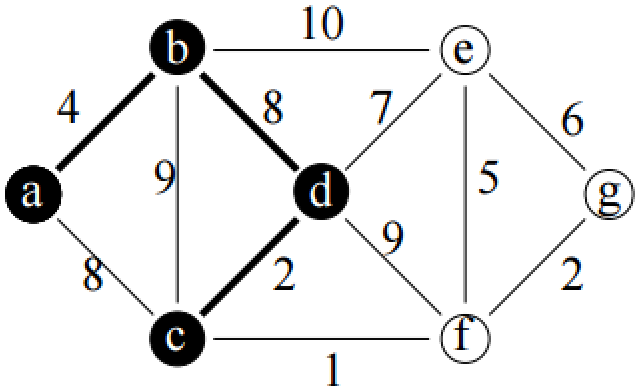
(i) Mark b, include edge (a, b)

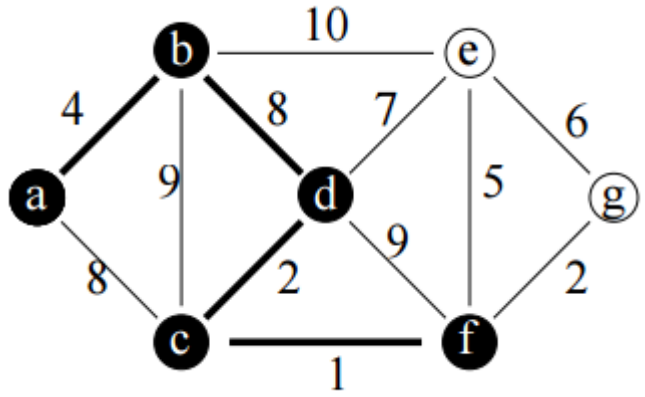
Exercise: Using Prim's algorithm find the minimum spanning tree starting from node a

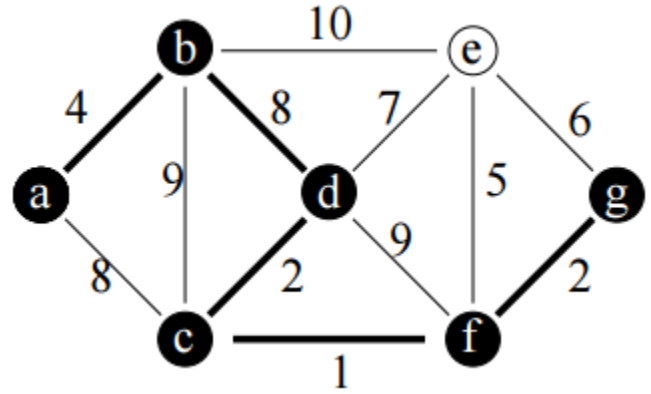


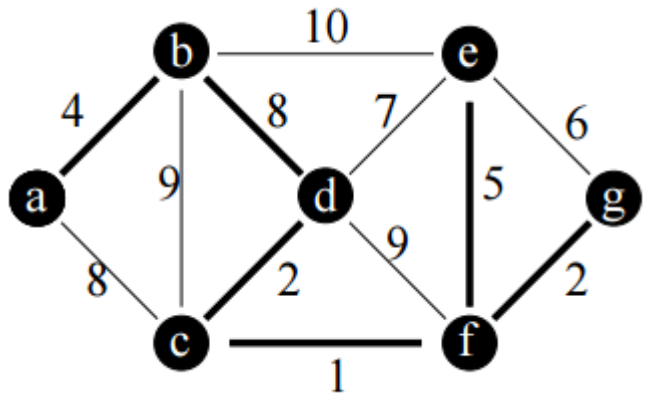







b
e
a
d
g
c
f
10
4
8
7
6
9
5
8
2
9
2
1

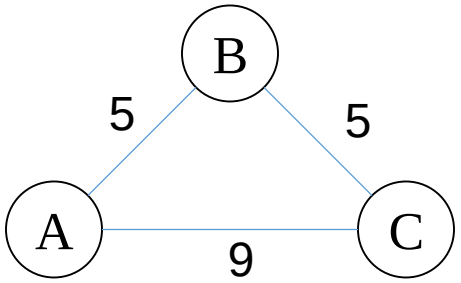

b
e
a
d
g
c
f
10
4
8
7
6
9
5
8
2
9
2
1

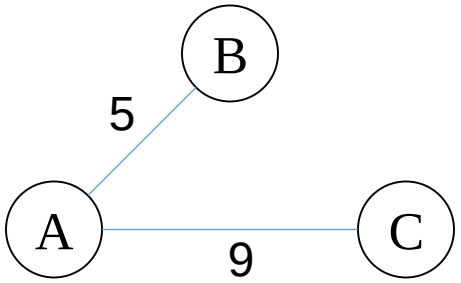

b
e
a
d
g
c
f
10
4
8
7
6
9
5
8
2
9
2
1

Shortest path vs MST

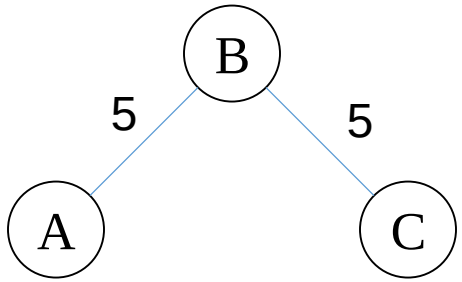
Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?

No. The MST assures that the total weight of the tree is kept at its minimum. But it doesn't mean that the distance between any two nodes involved in the MST is minimum. Also the shortest path tree is not guaranteed to be a MST. Another important difference in the type of graphs the algorithms work on. Prim's algorithm works on undirected graphs only while Dijkstra's algorithm will work fine on directed graphs.





Shortest path from node A



MST

Boruvka's (Sollin) Algorithm

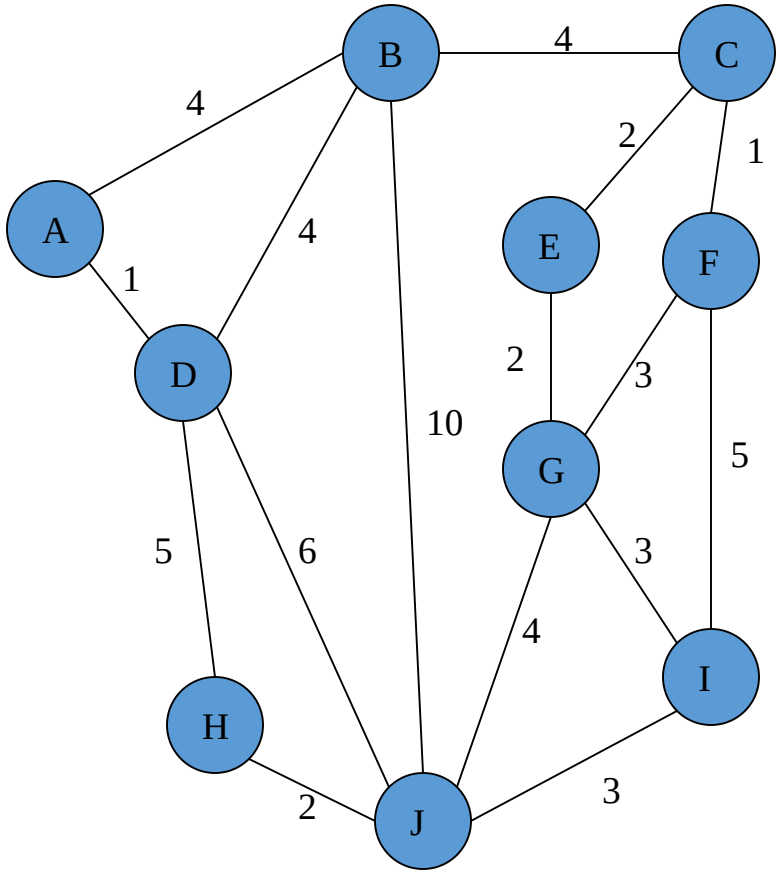
This algorithm is similar to Prim's, but nodes are added to the new graph in parallel all around the graph. It creates a **list of trees**, each containing **one node** from the original graph and proceeds to merge them along the smallest-weight connecting edges until there's only one tree, which is, of course, the MST. It works rather like a merge sort.

The steps are:

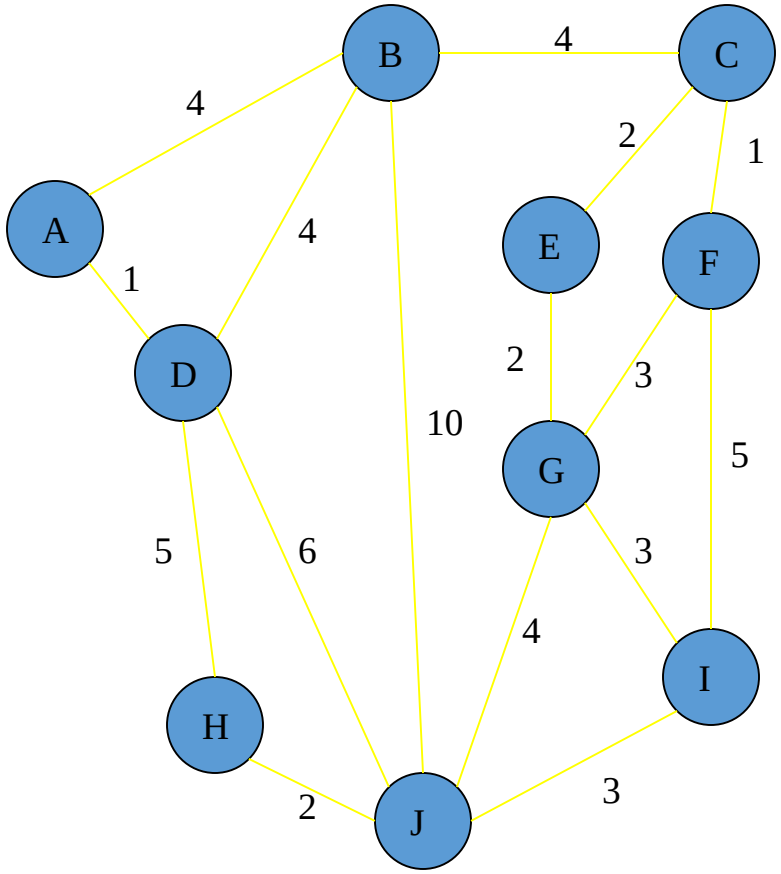
1. Make a list of **n trees**, each containing a single node
2. While list has more than one tree,
    1. For each tree in the list, find the **node not connected** to the tree with the smallest connecting edge to that tree,
    2. **Add** all the edges found to the new graph, thus creating a new set of trees

Every step will have joined groups of trees, until only one tree remains.

Complete Graph



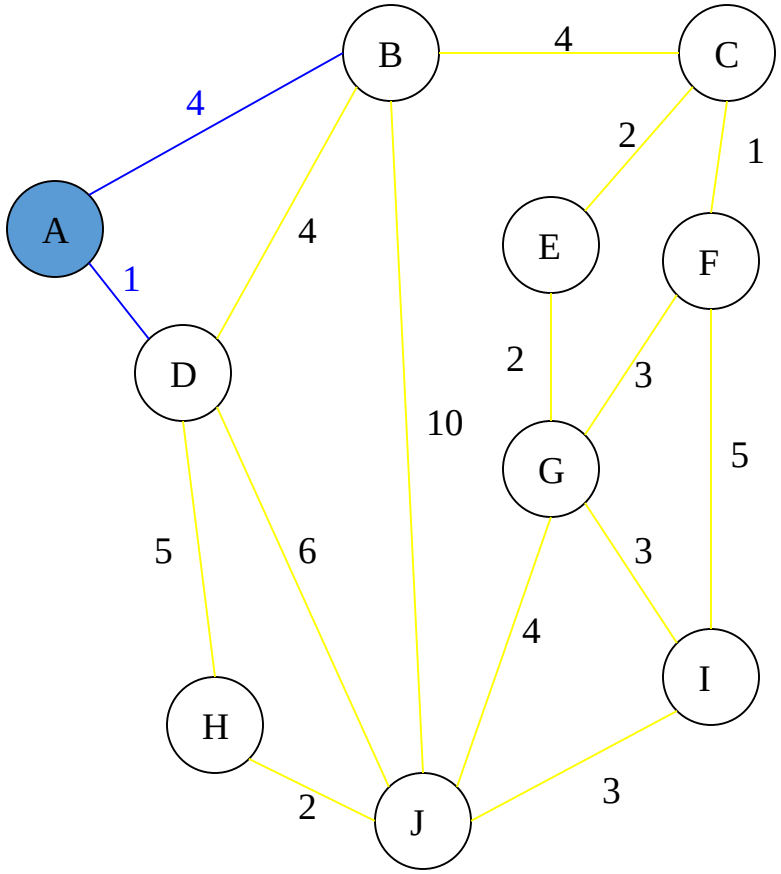
Trees of the Graph at Beginning of Round 1



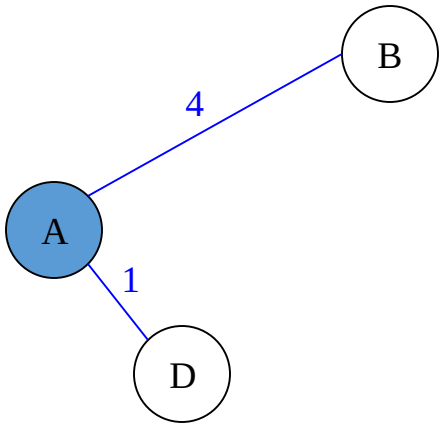
List of Trees

- A
- B
- C
- D
- E
- F
- G
- H
- I
- J

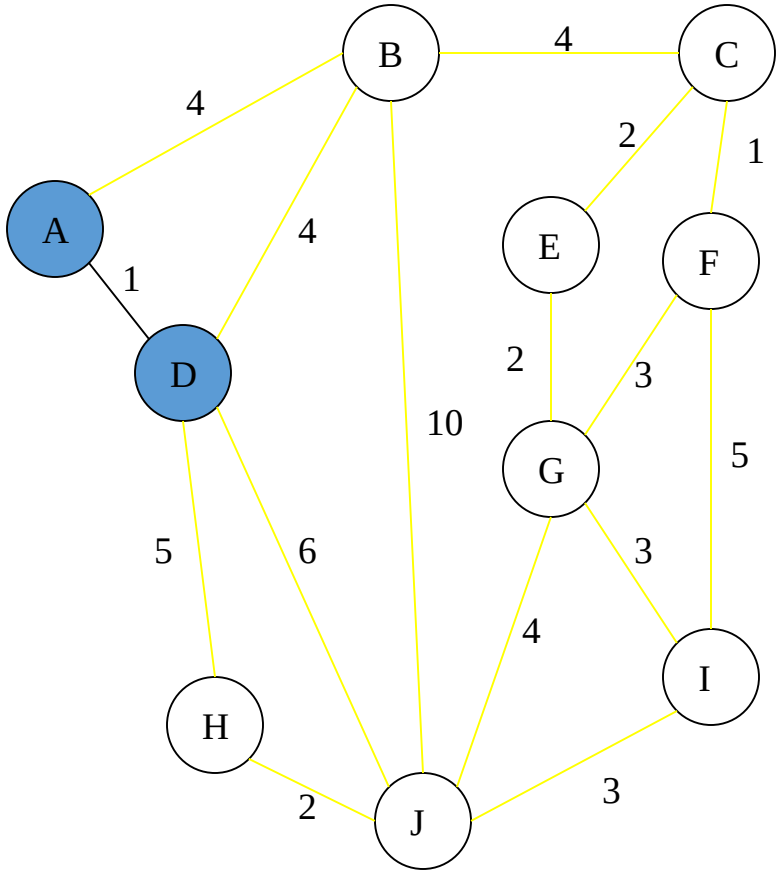
Round 1



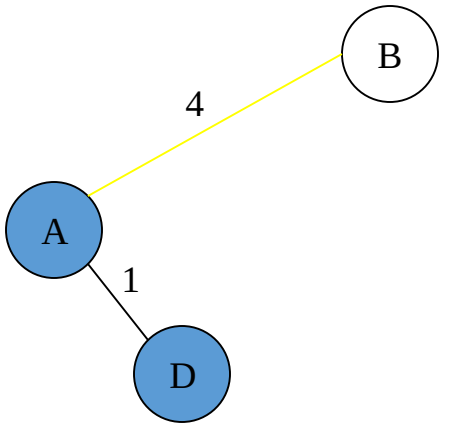
Tree A



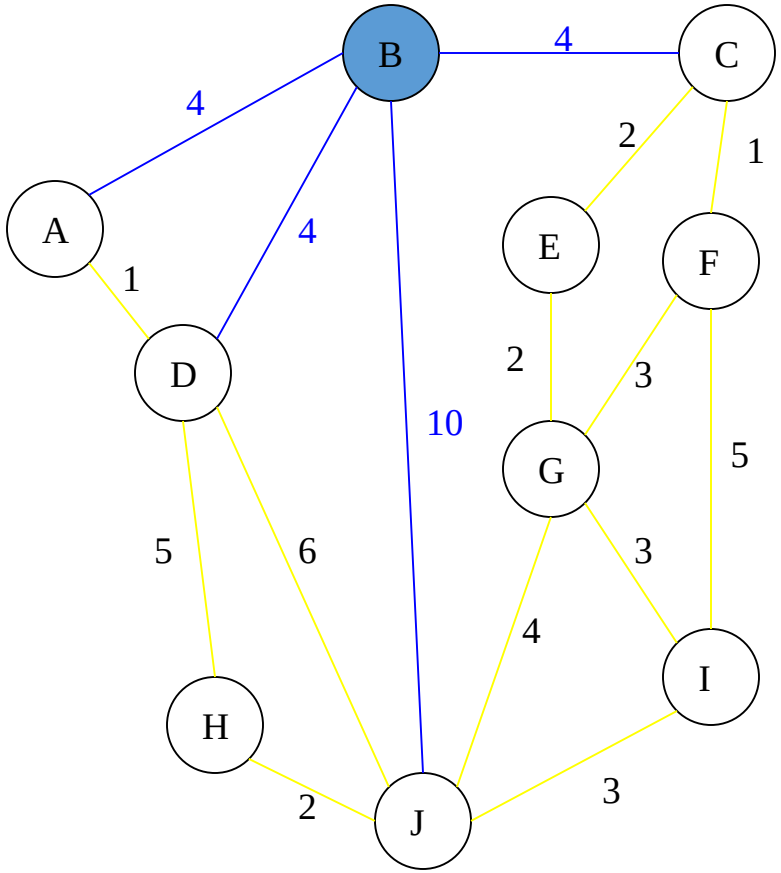
Round 1



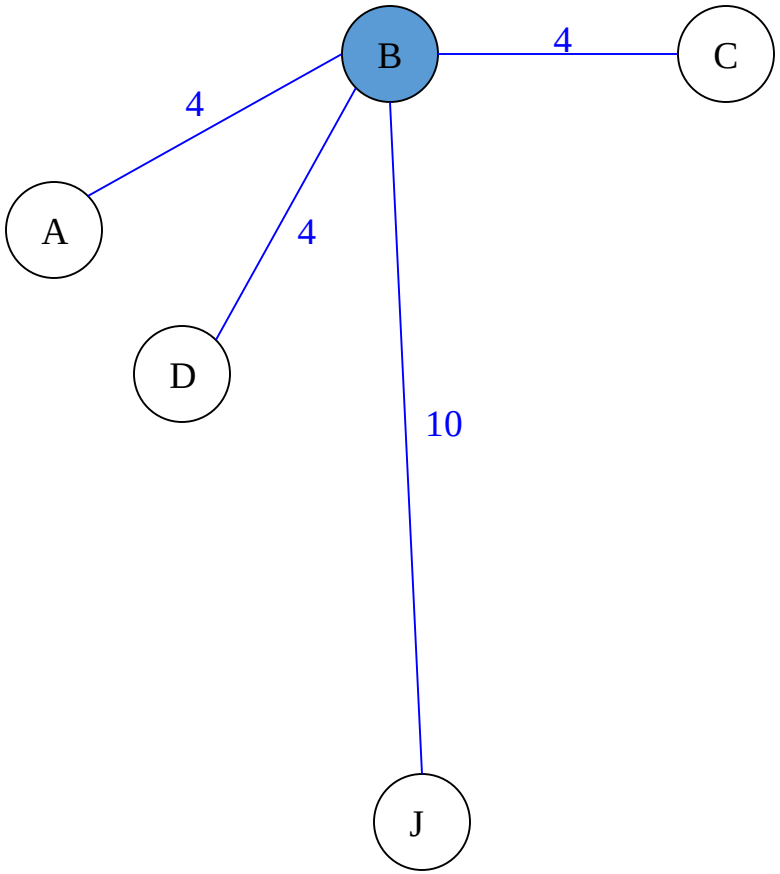
Edge A-D



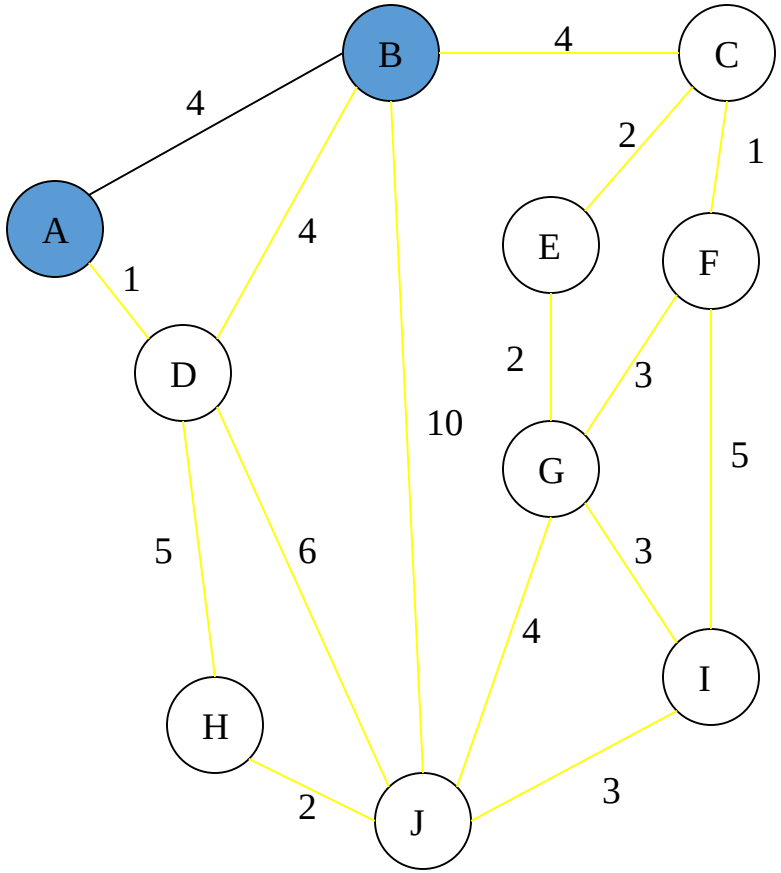
Round 1



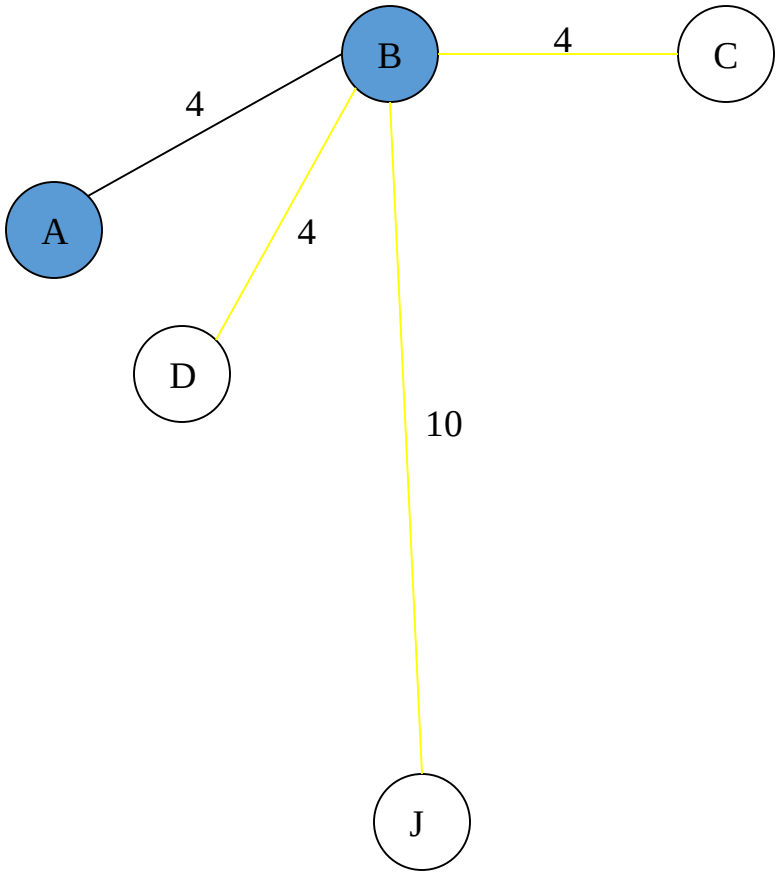
Tree B



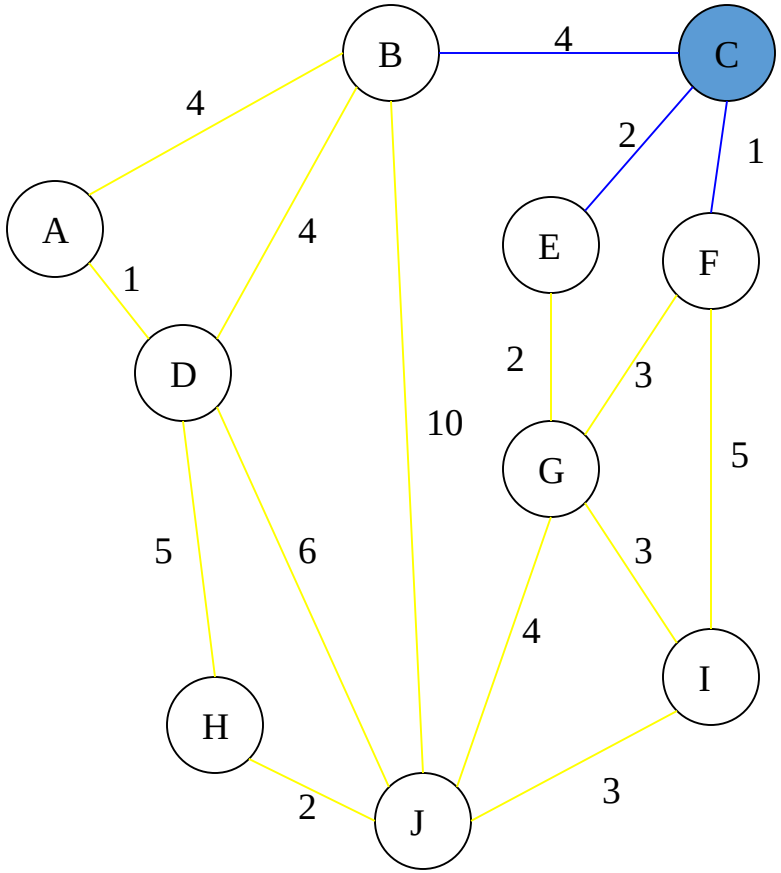
Round 1



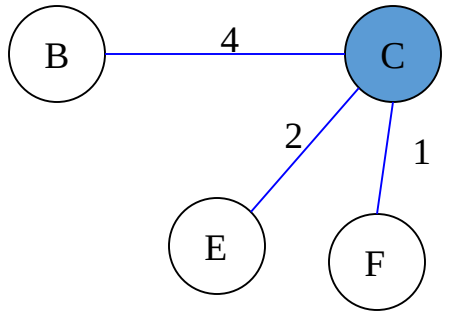
Edge B-A



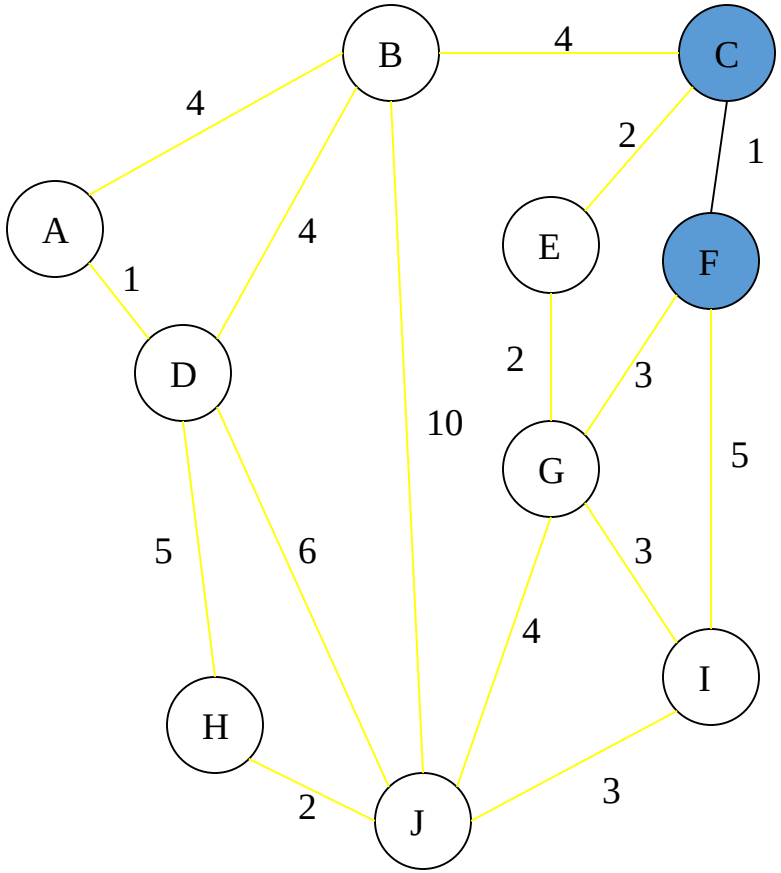
Round 1



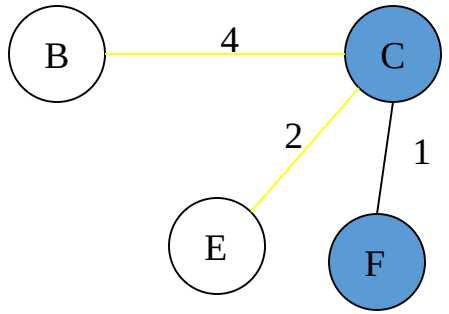
Tree C



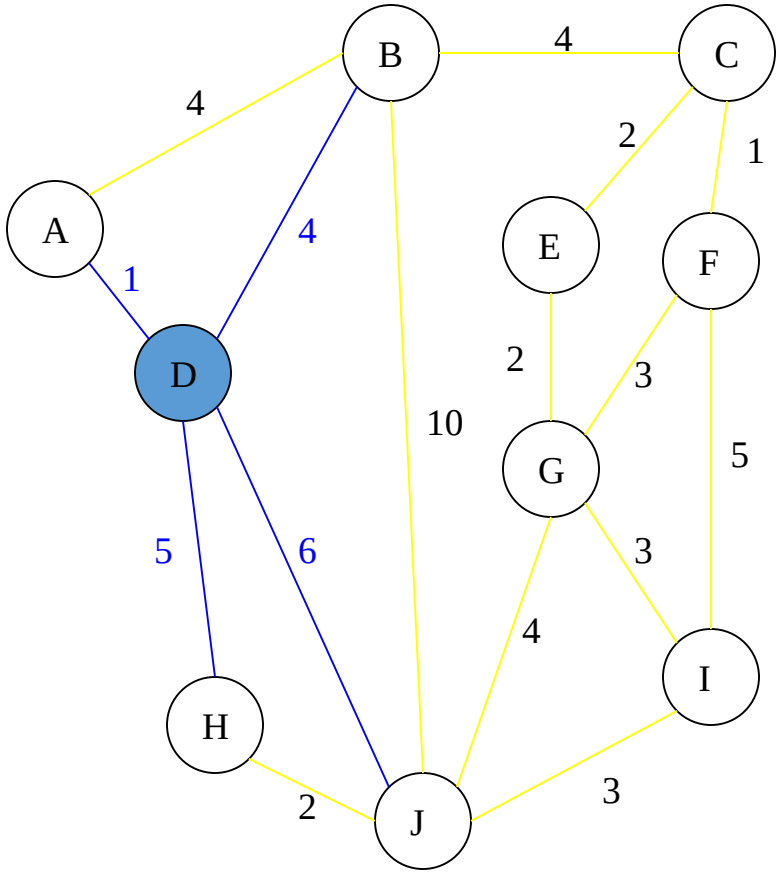
Round 1



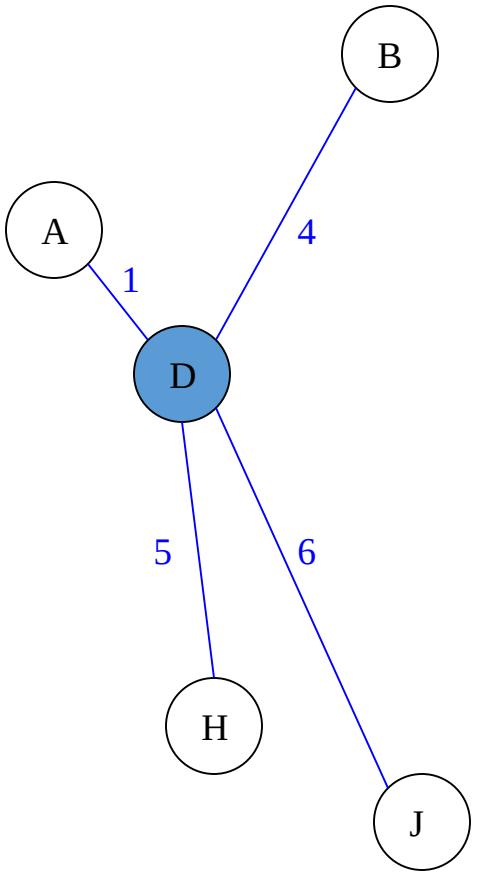
Edge C-F



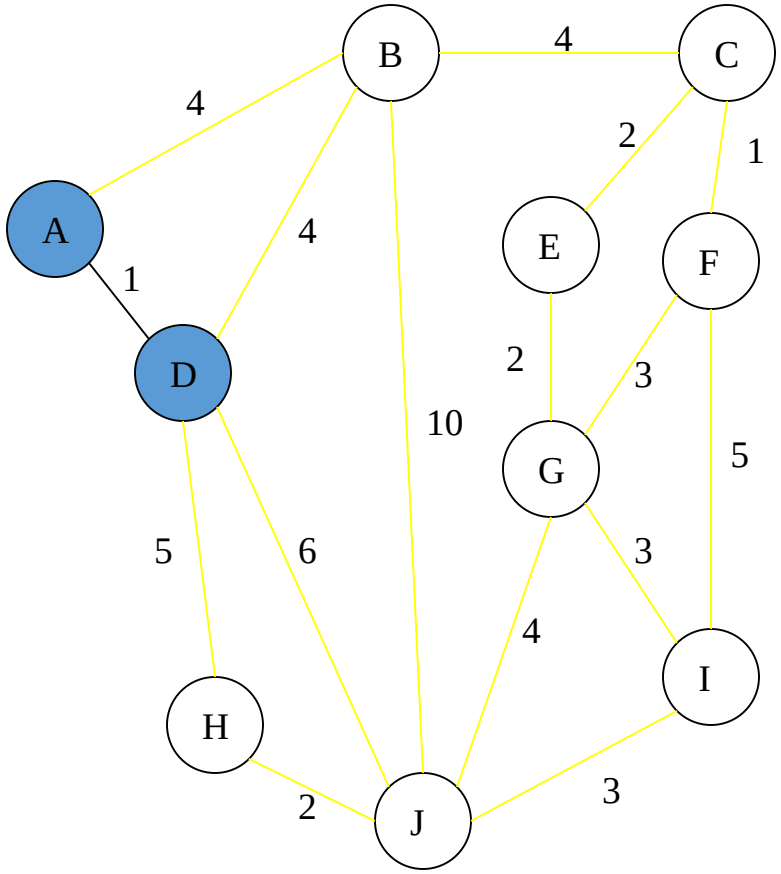
Round 1



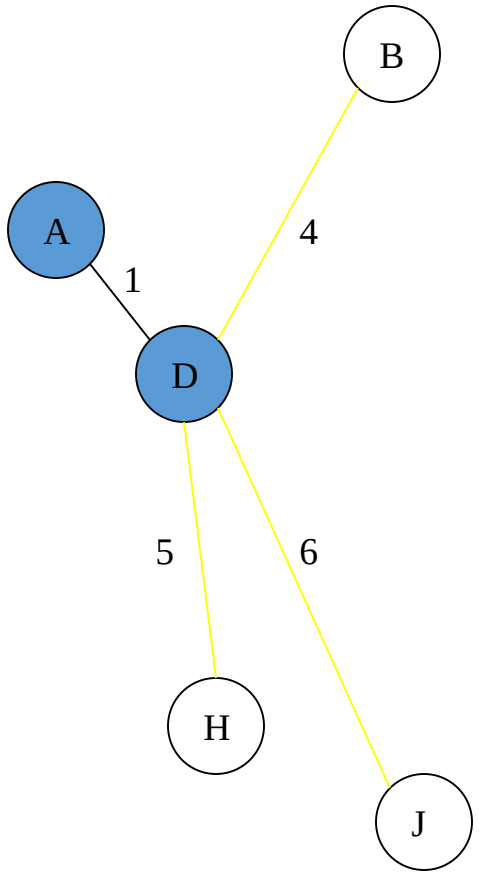
Tree D



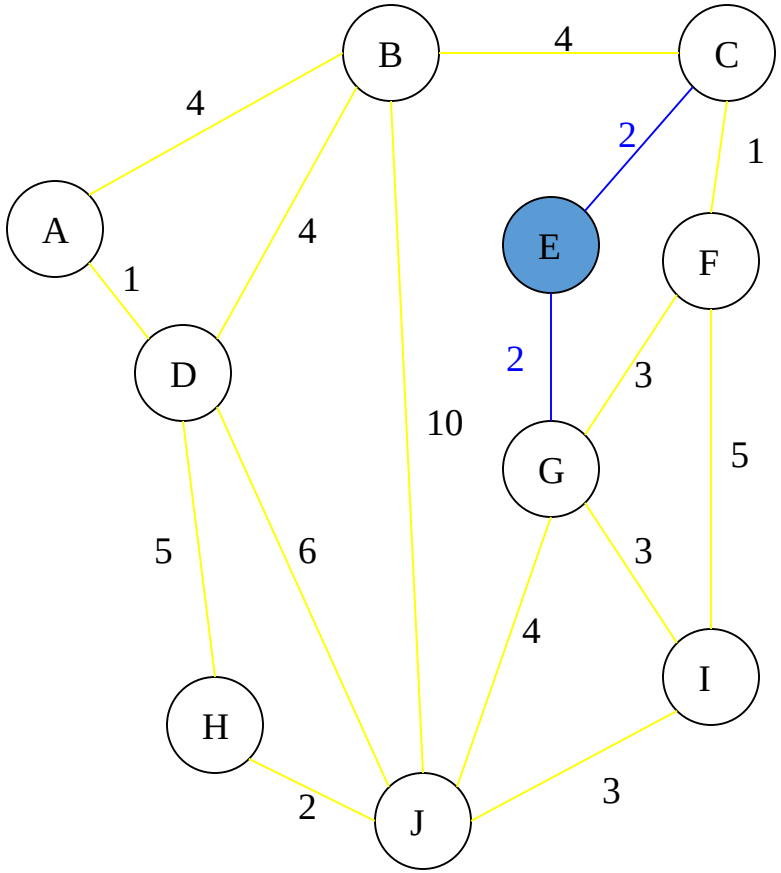
Round 1



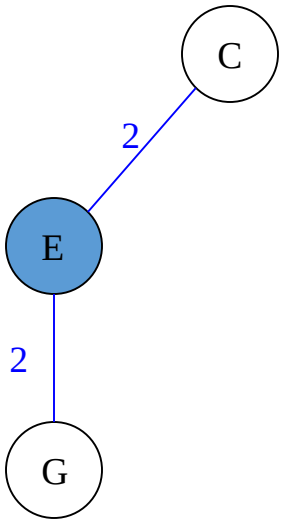
Edge D-A



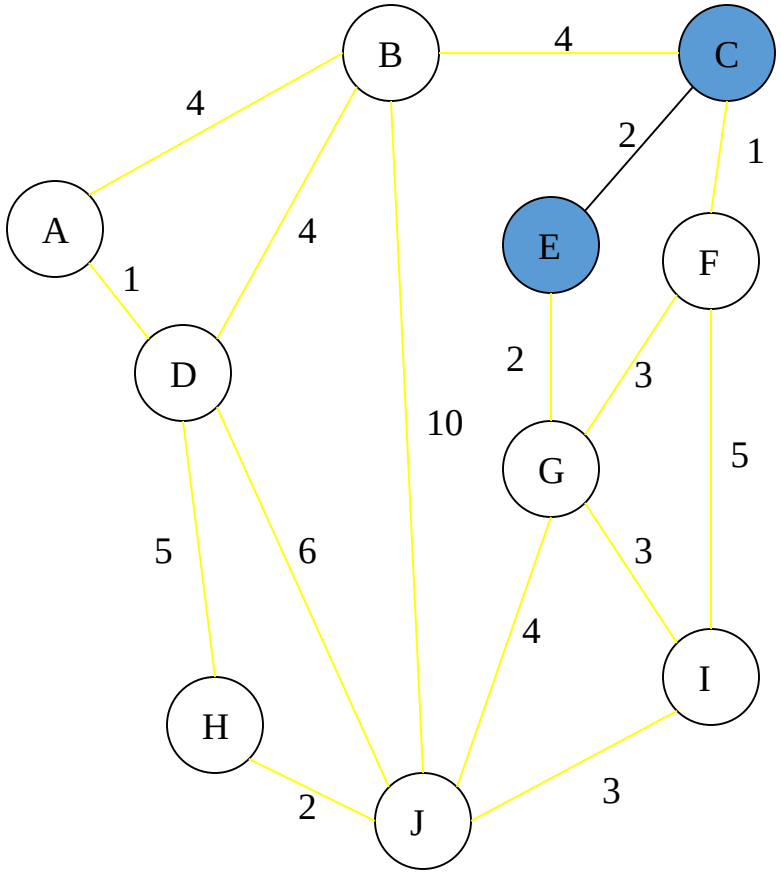
Round 1



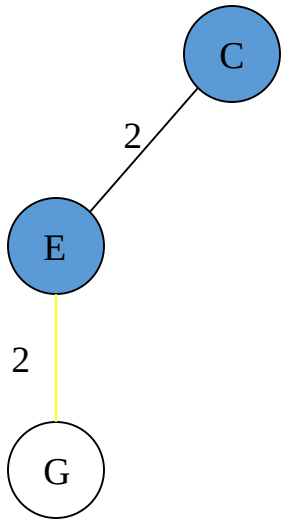
Tree E



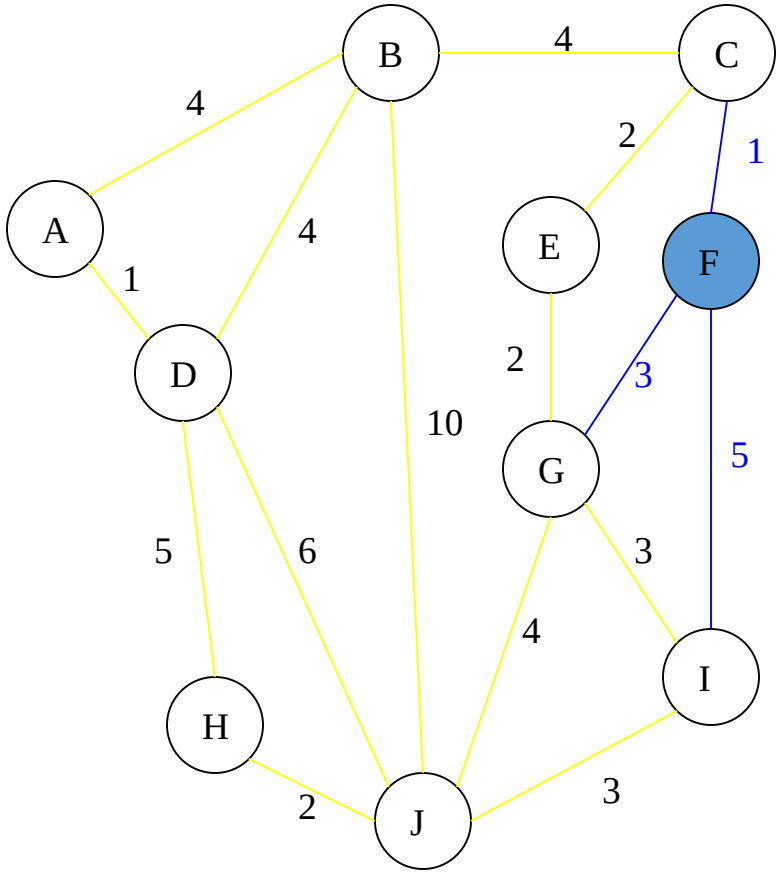
Round 1



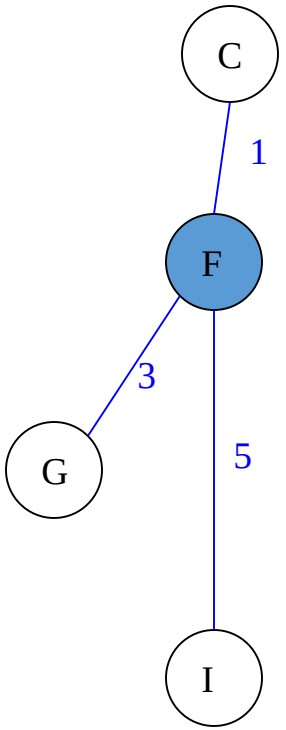
Edge E-C



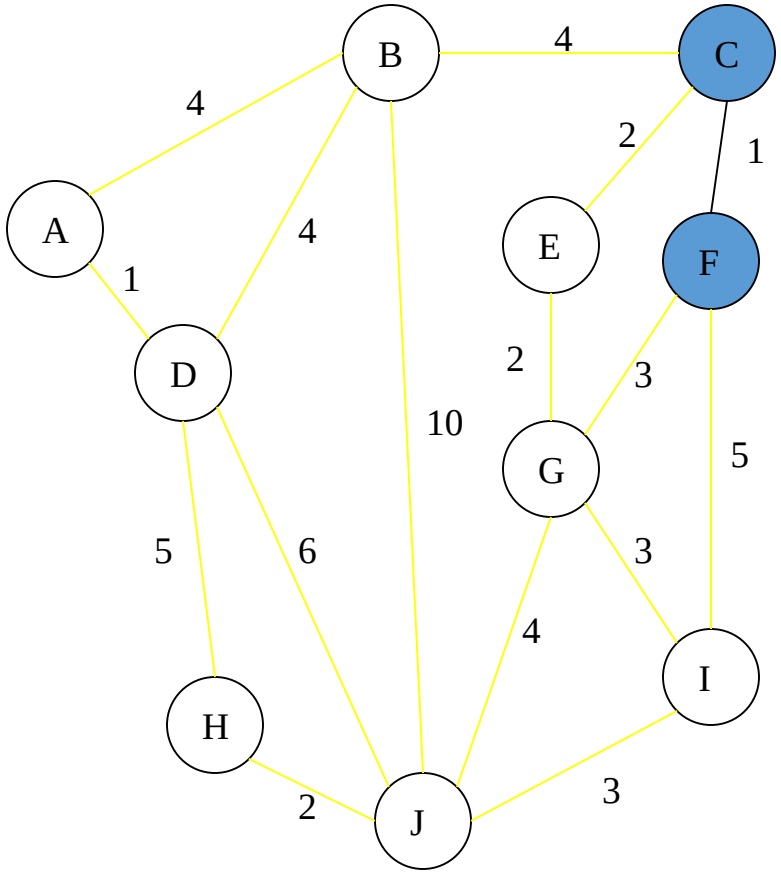
Round 1



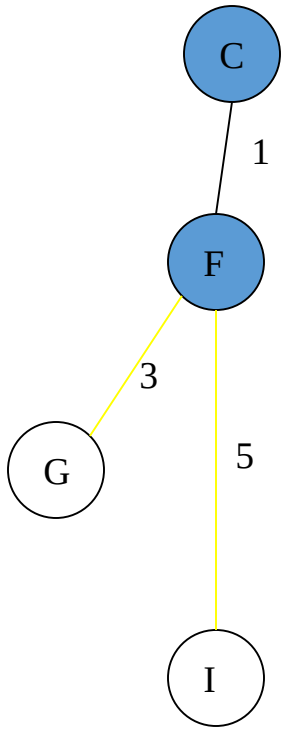
Tree F



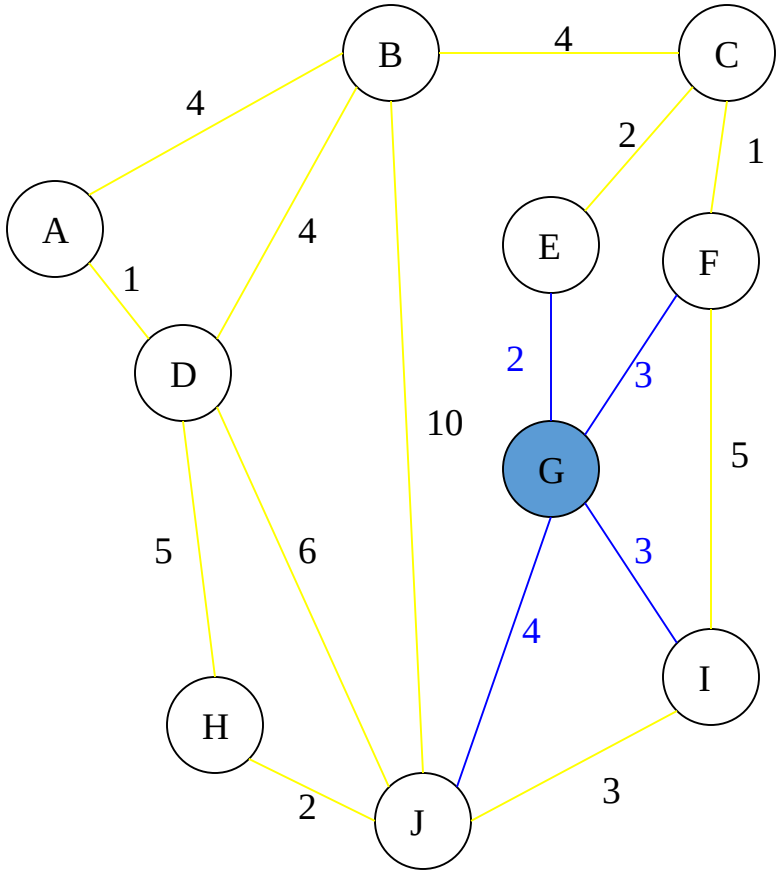
Round 1



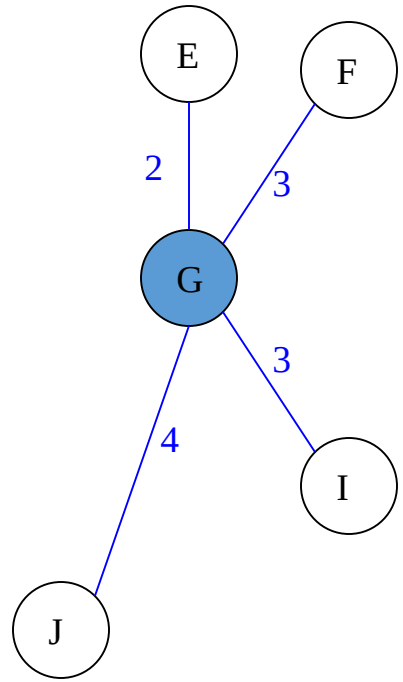
Edge F-C



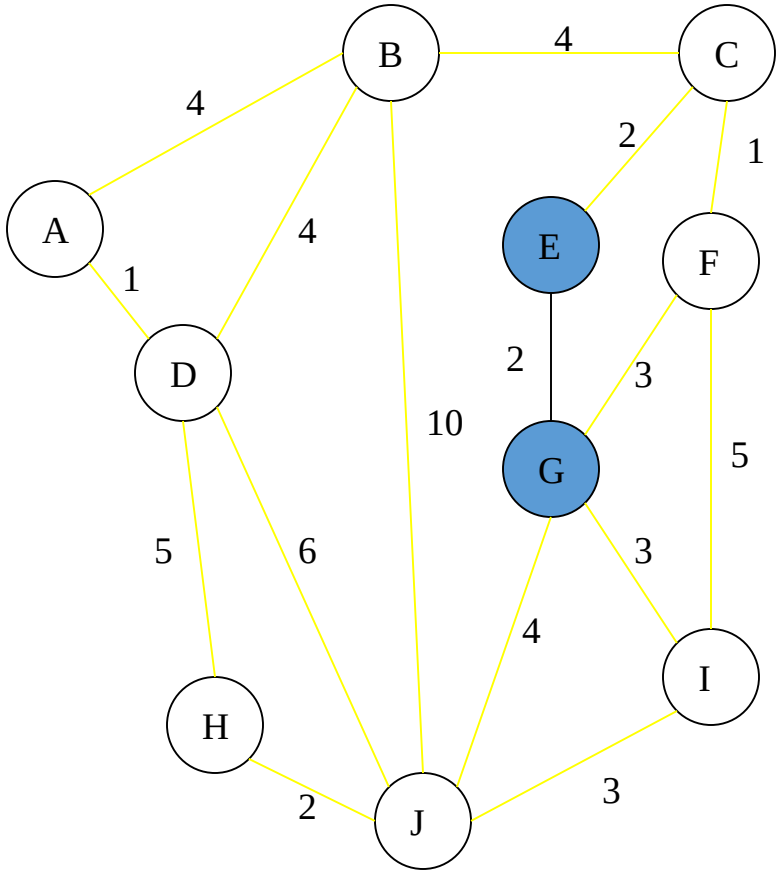
Round 1



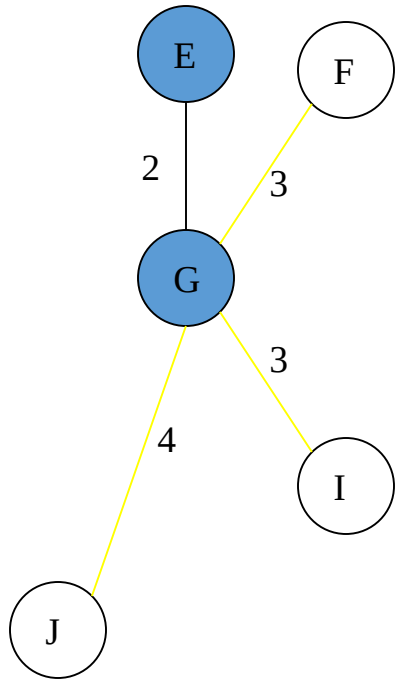
Tree G



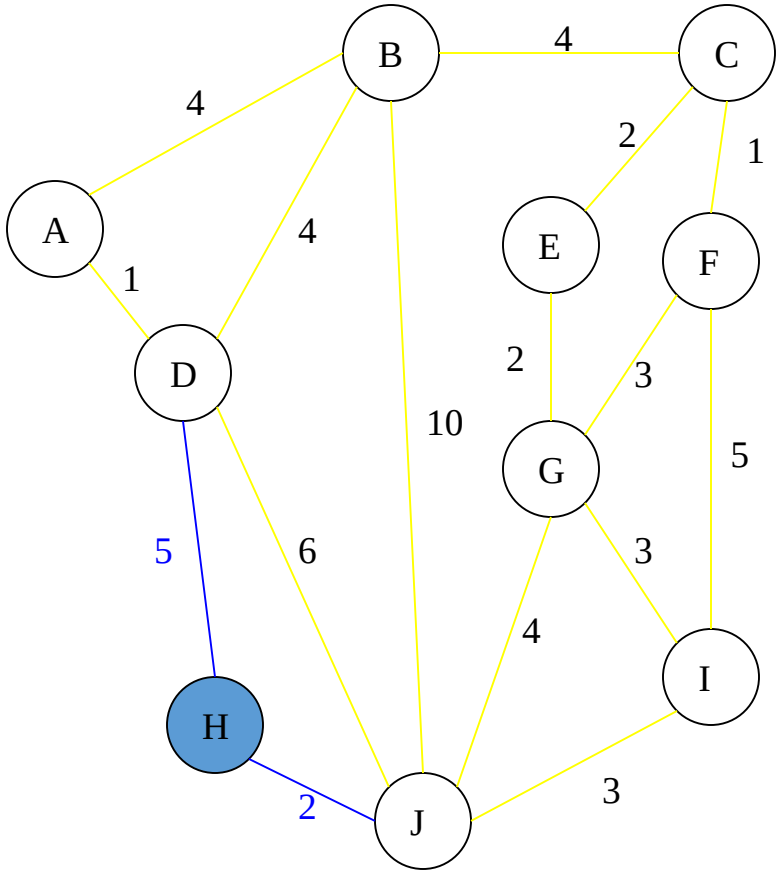
Round 1



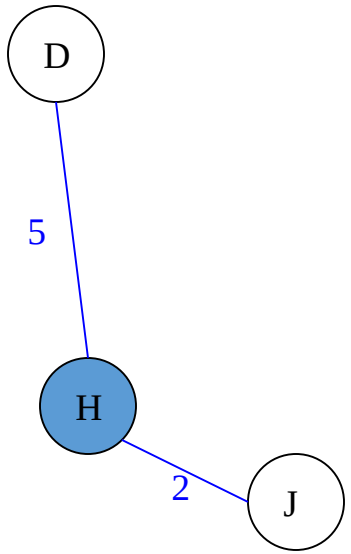
Edge G-E



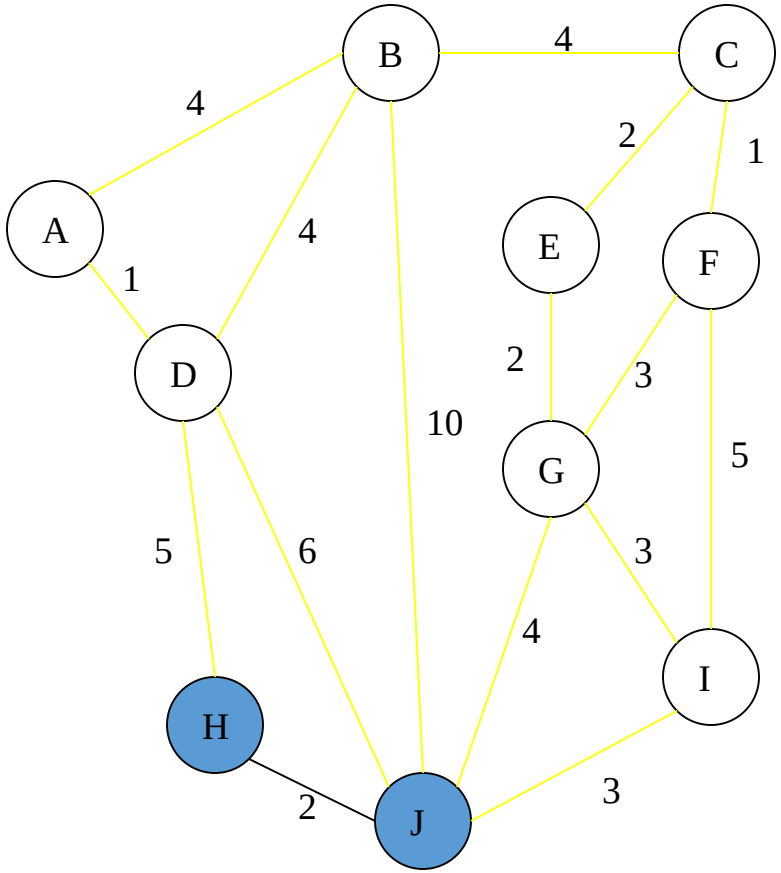
Round 1



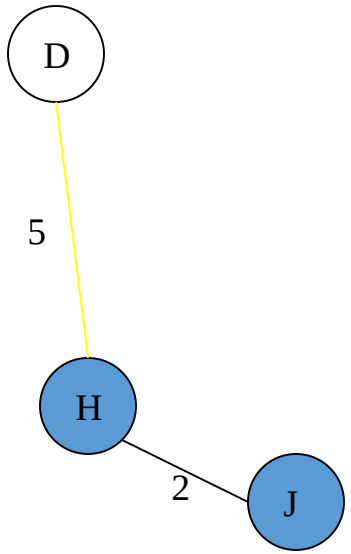
Tree H



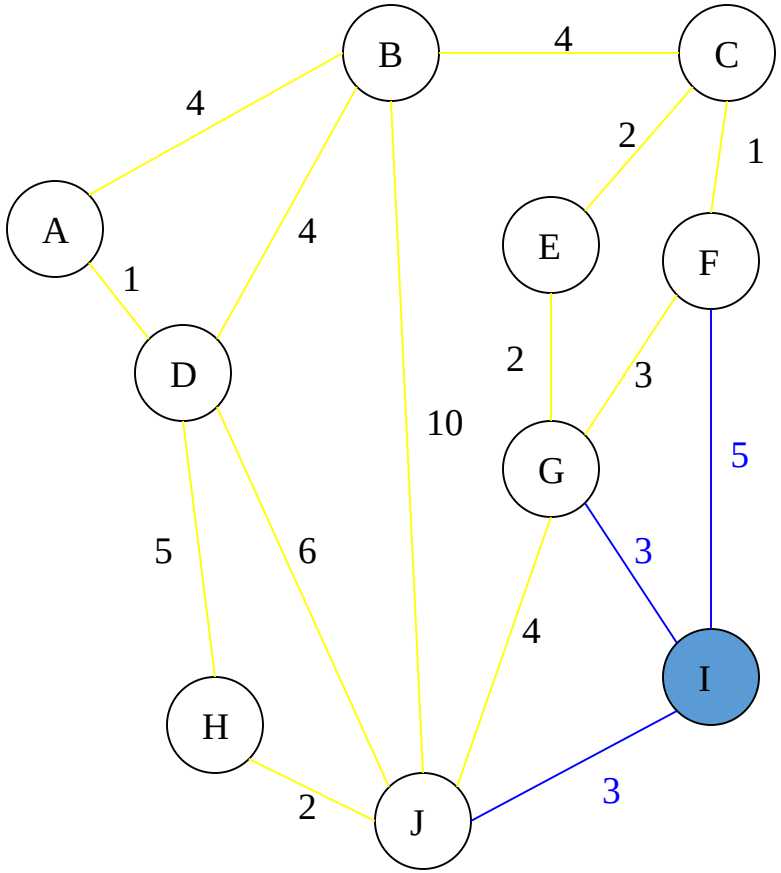
Round 1



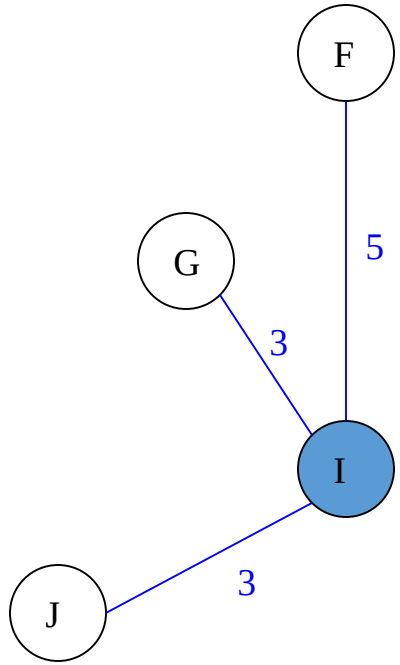
Edge H-J



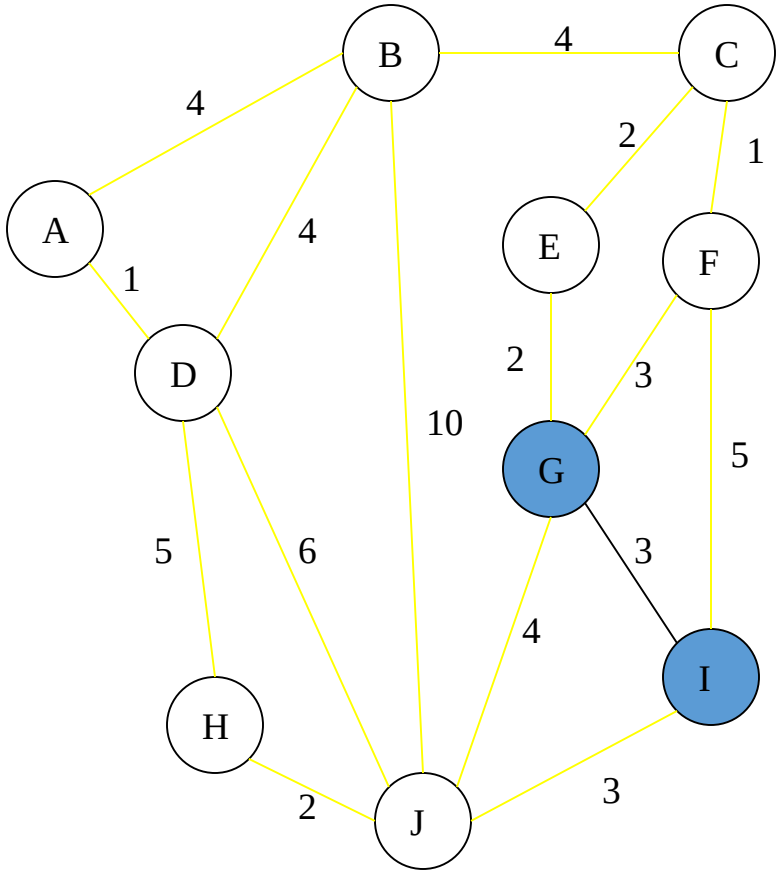
Round 1



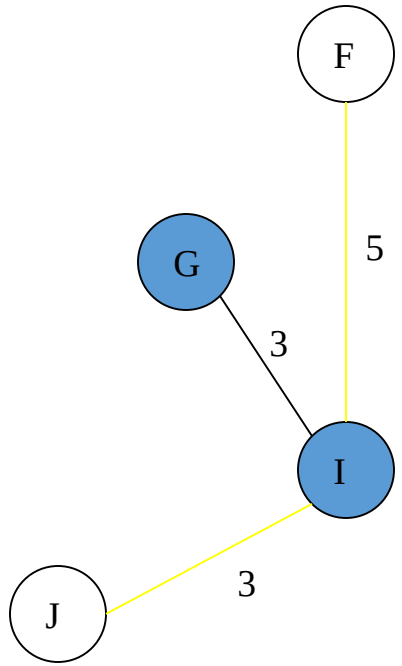
Tree I



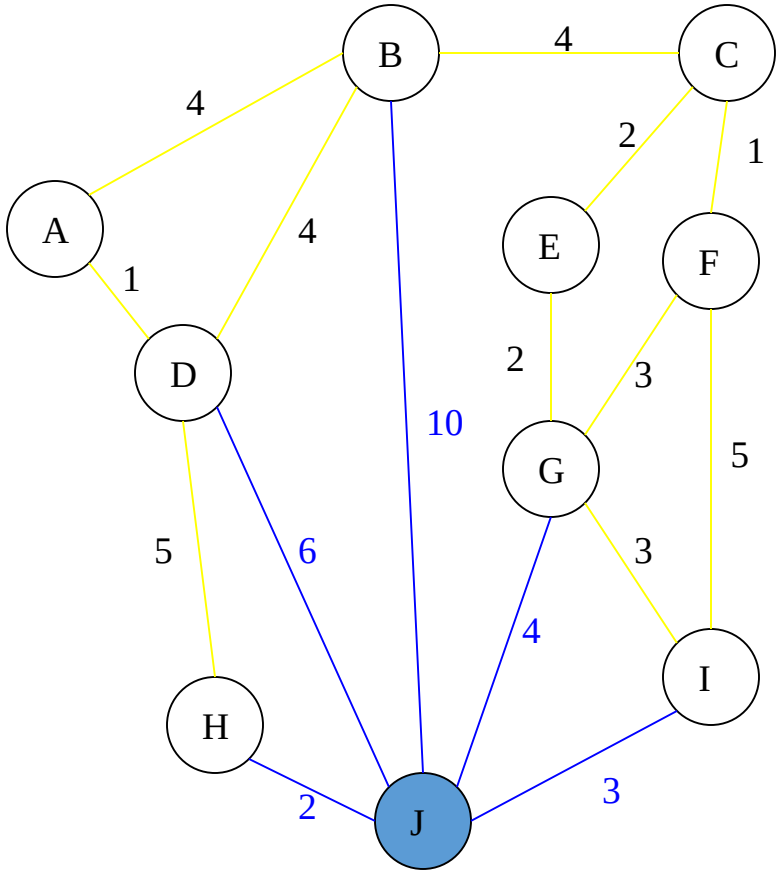
Round 1



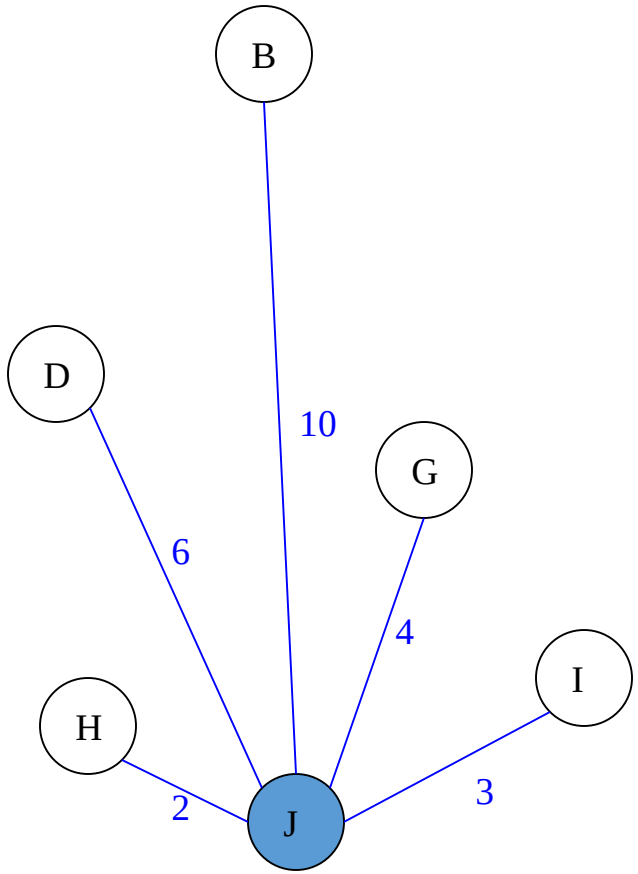
Edge I-G



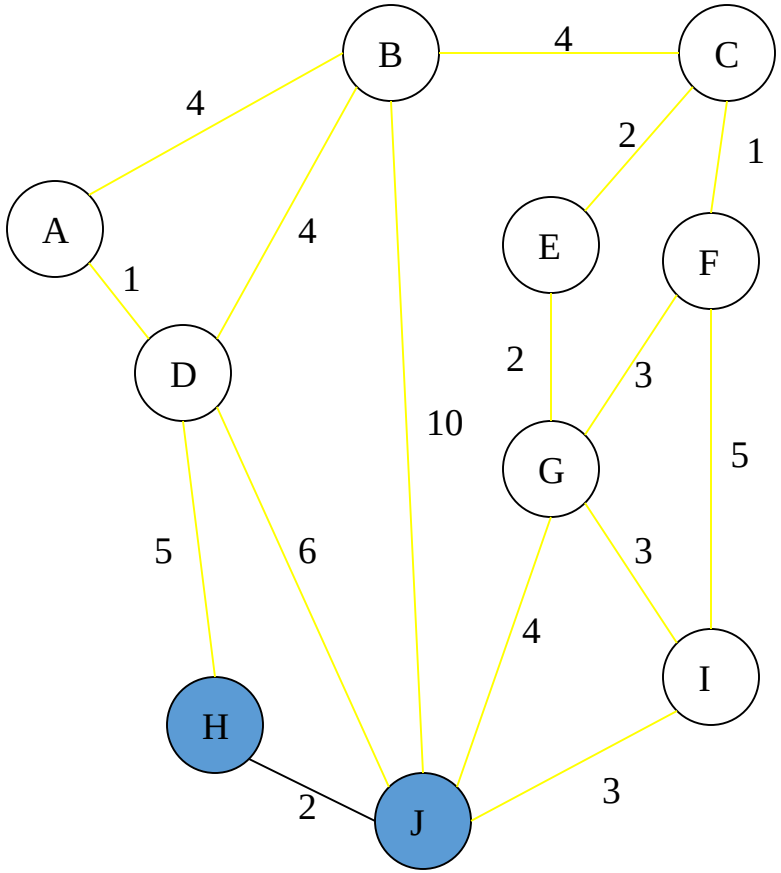
Round 1



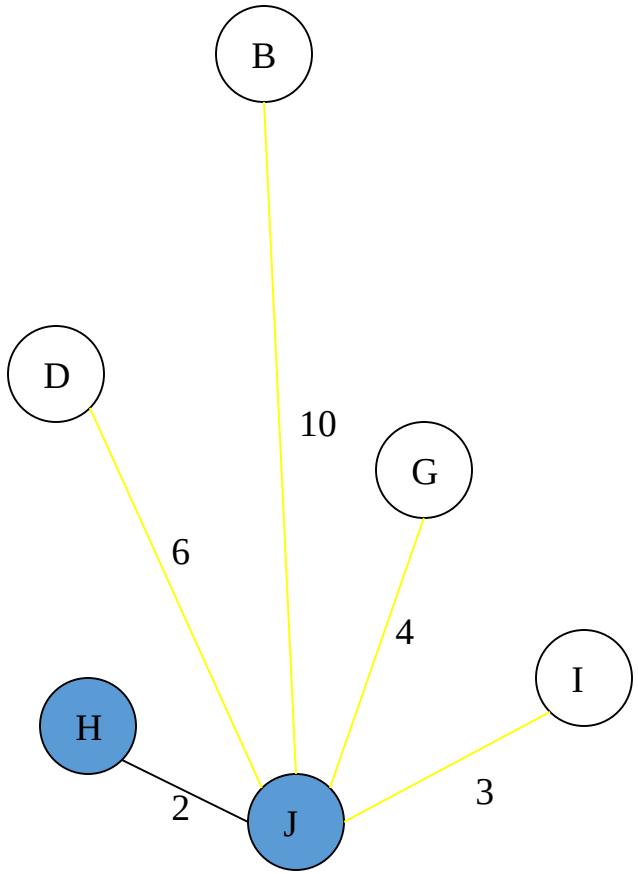
Tree J



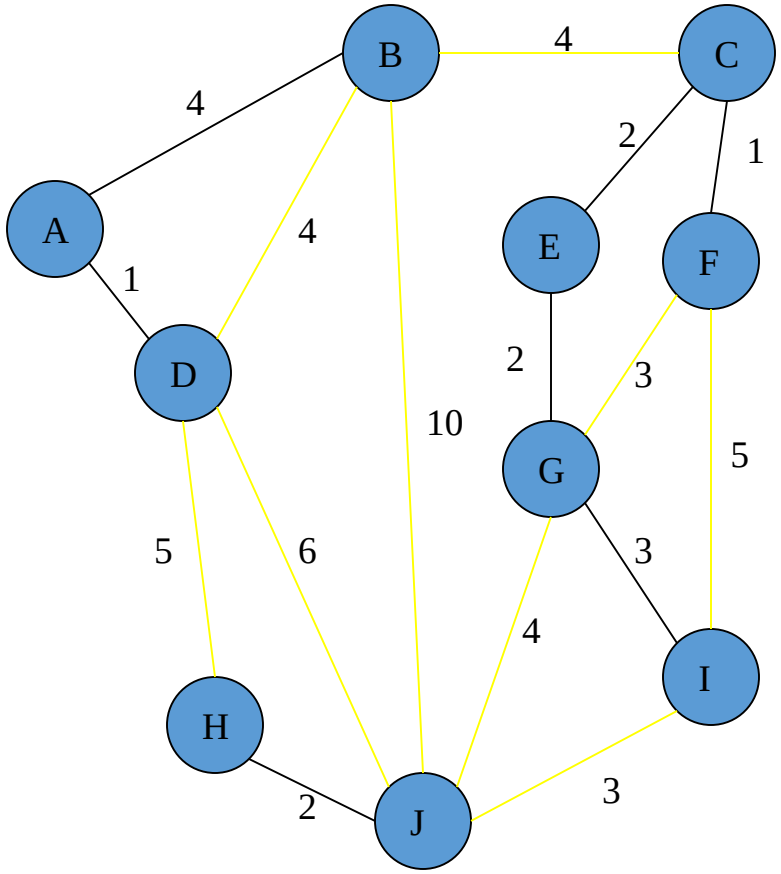
Round 1



Edge J-H



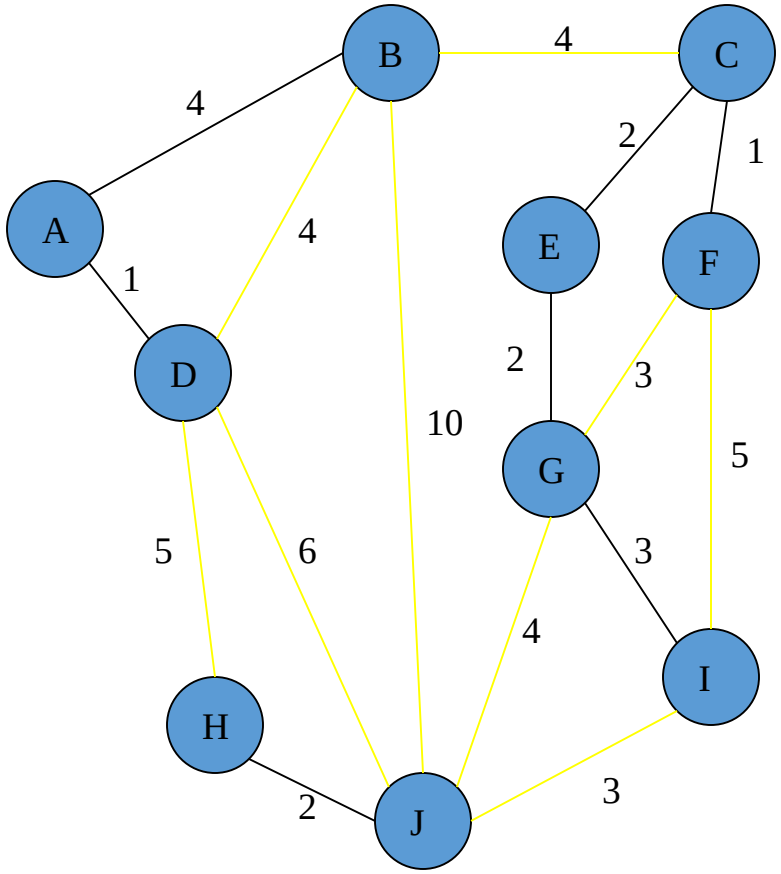
Round 1 Ends - Add Edges



List of Edges to Add

- **A-D**
- B-A
- **C-F**
- **D-A**
- E-C
- **F-C**
- G-E
- **H-J**
- I-G
- **J-H**

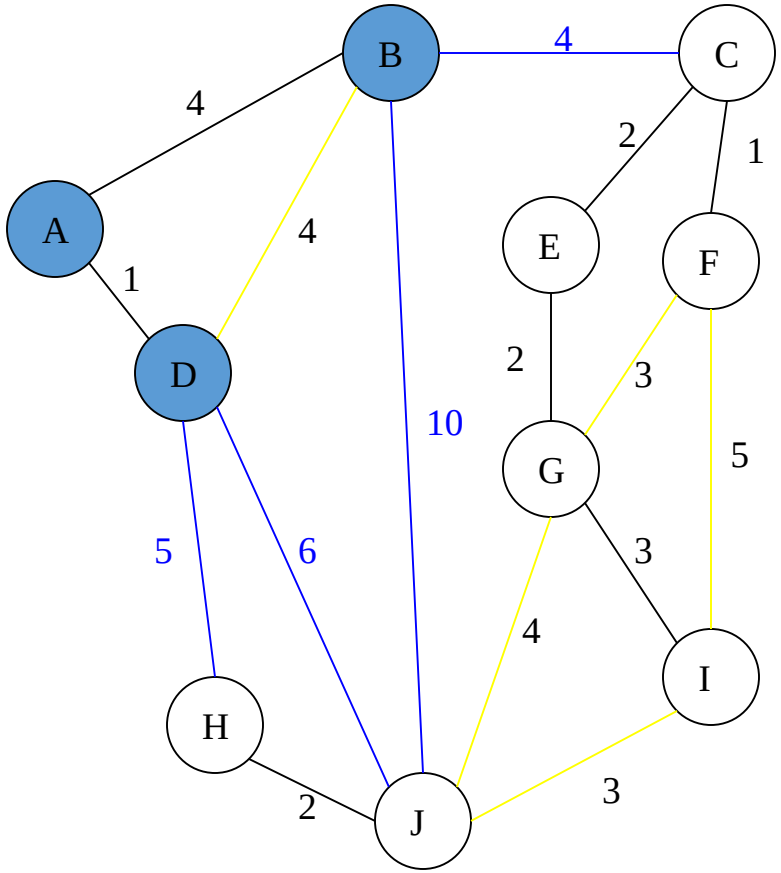
Trees of the Graph at Beginning of Round 2



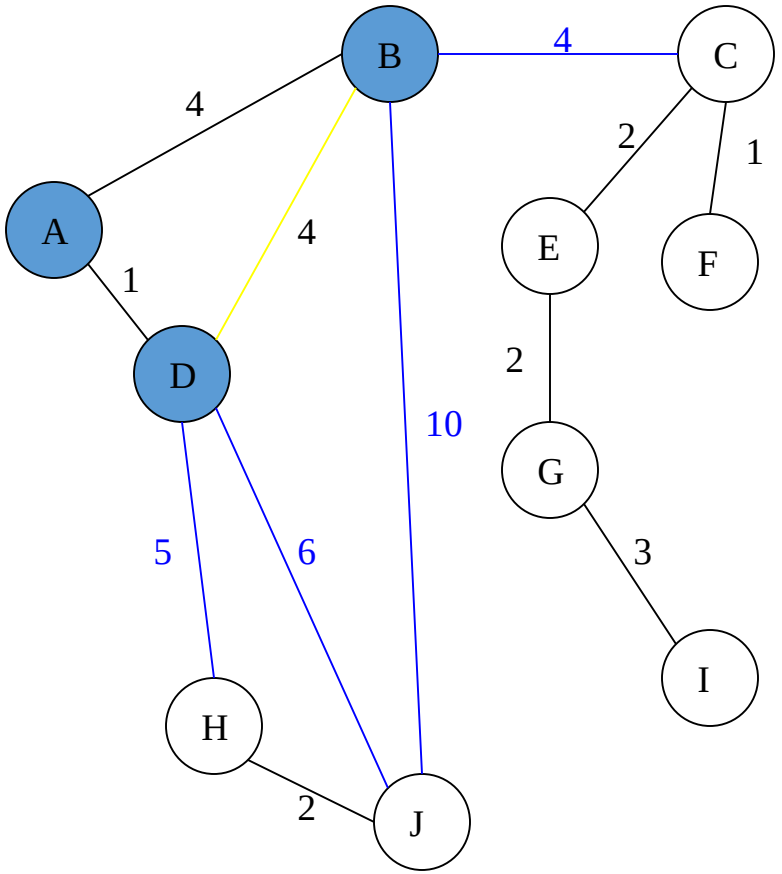
List of Trees

- D-A-B
- F-C-E-G-I
- H-J

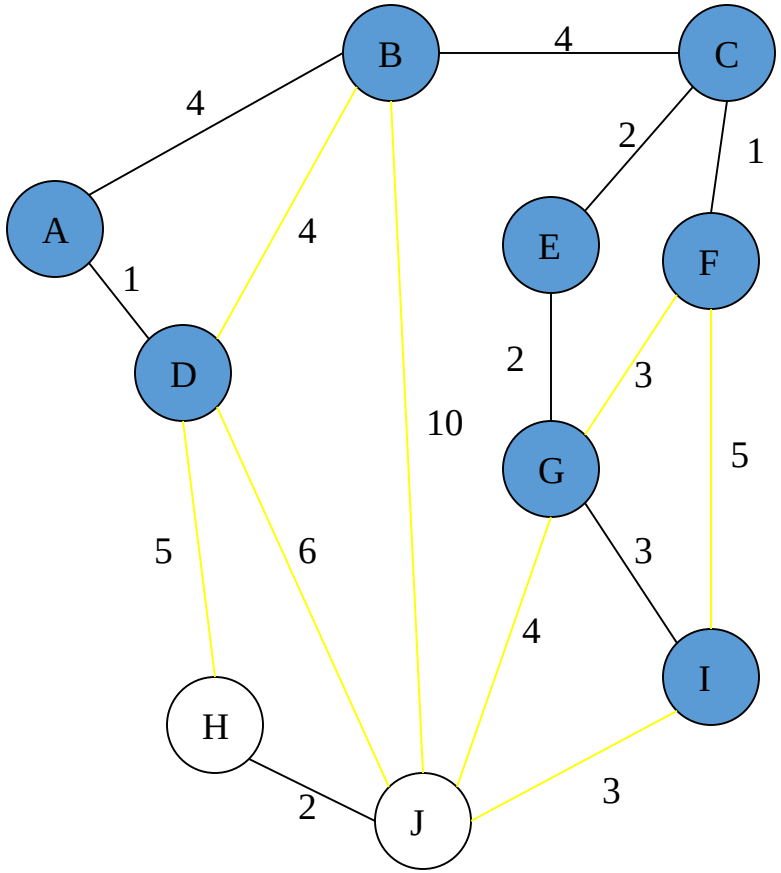
Round 2



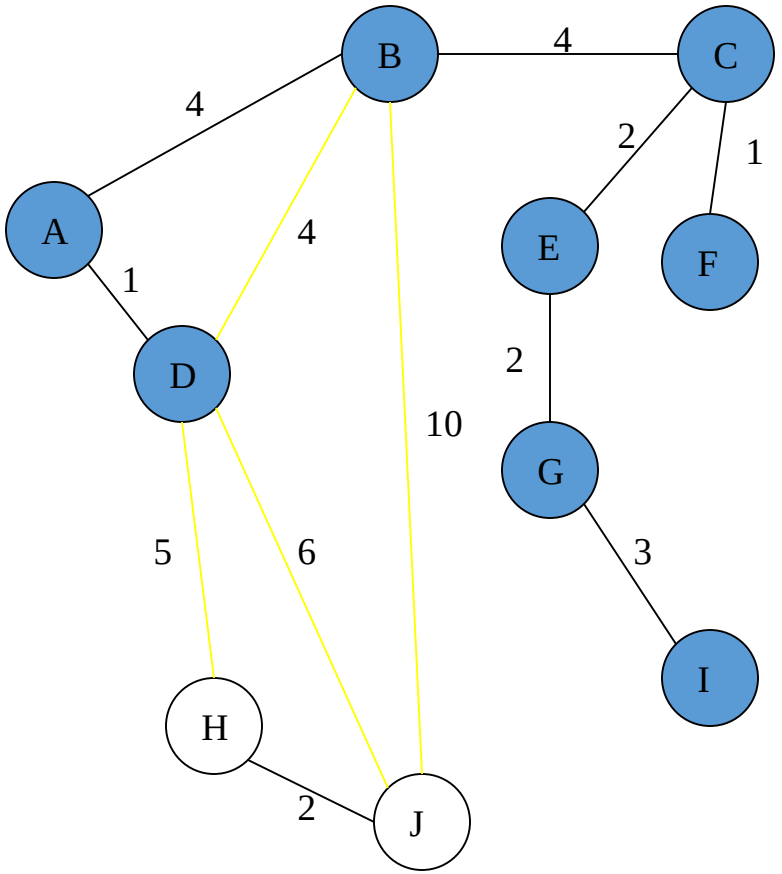
Tree D-A-B



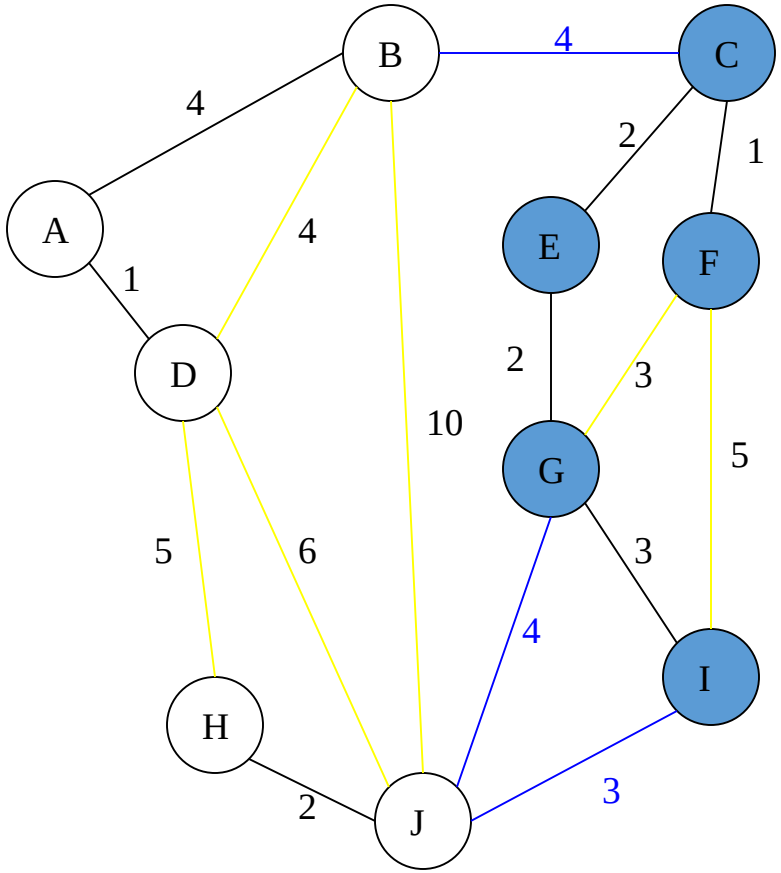
Round 2



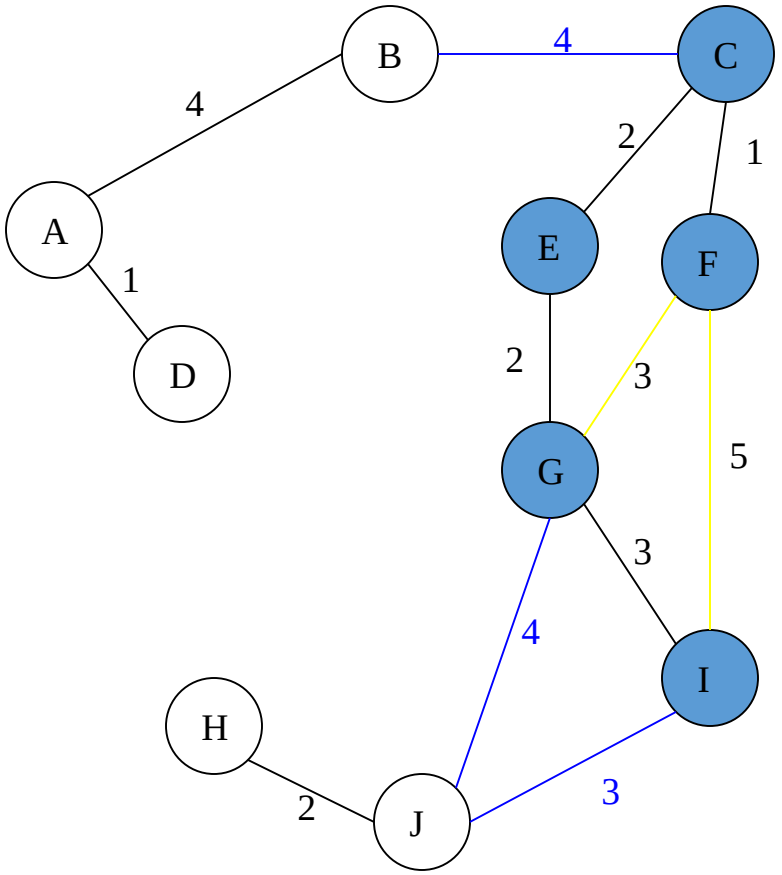
Edge B-C



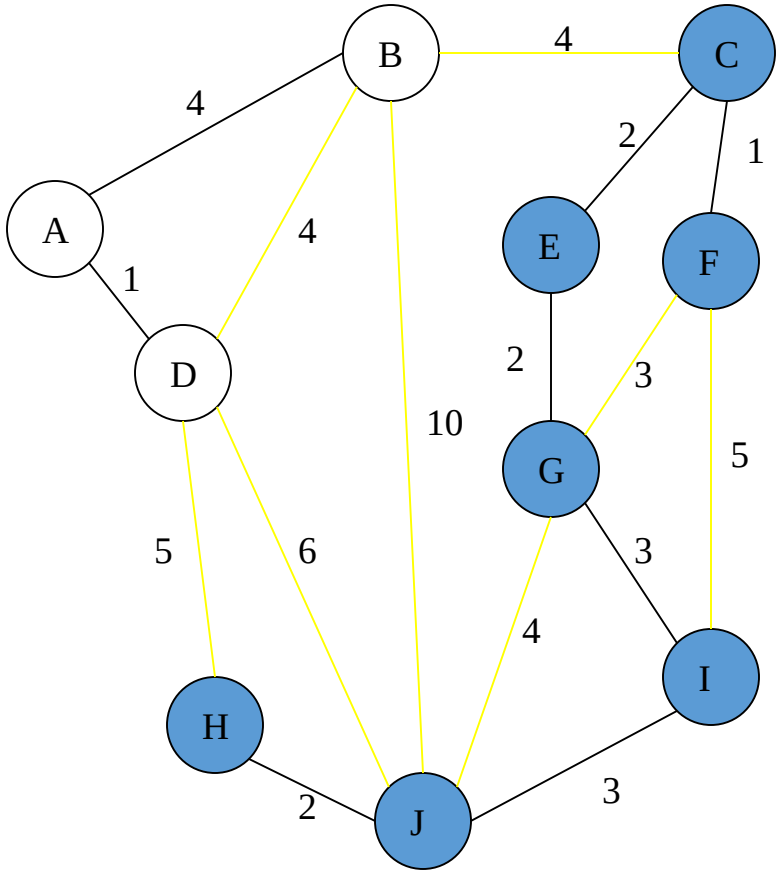
Round 2



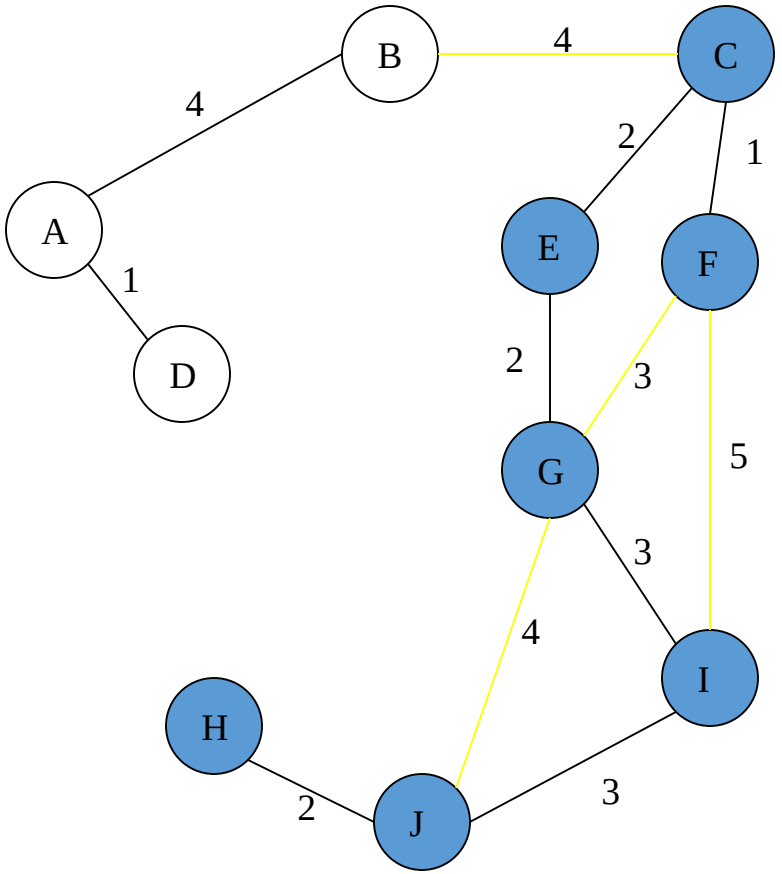
Tree F-C-E-G-I



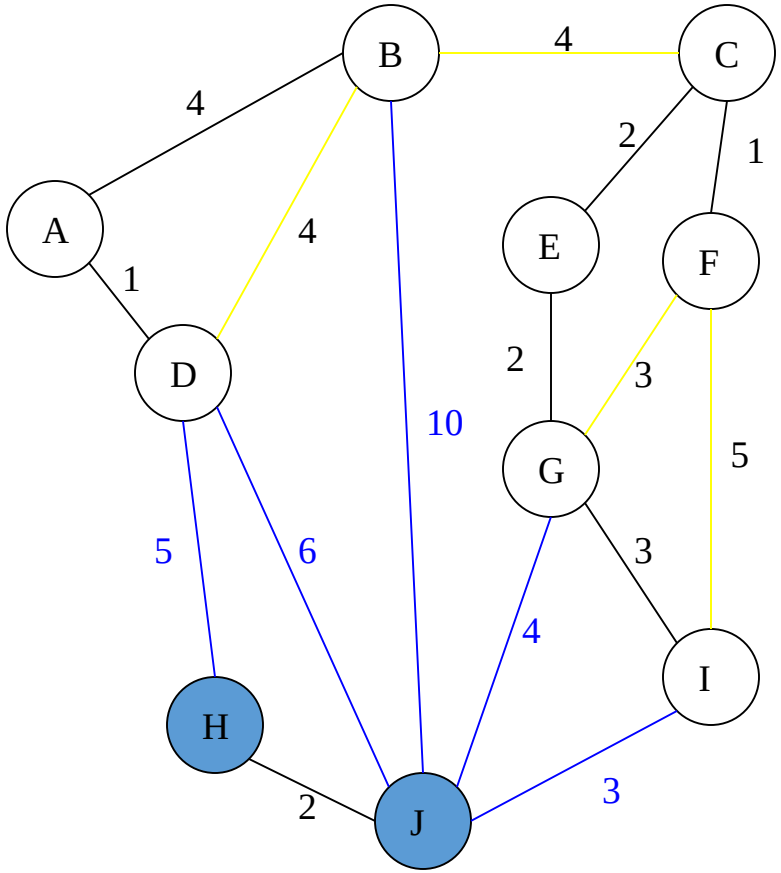
Round 2



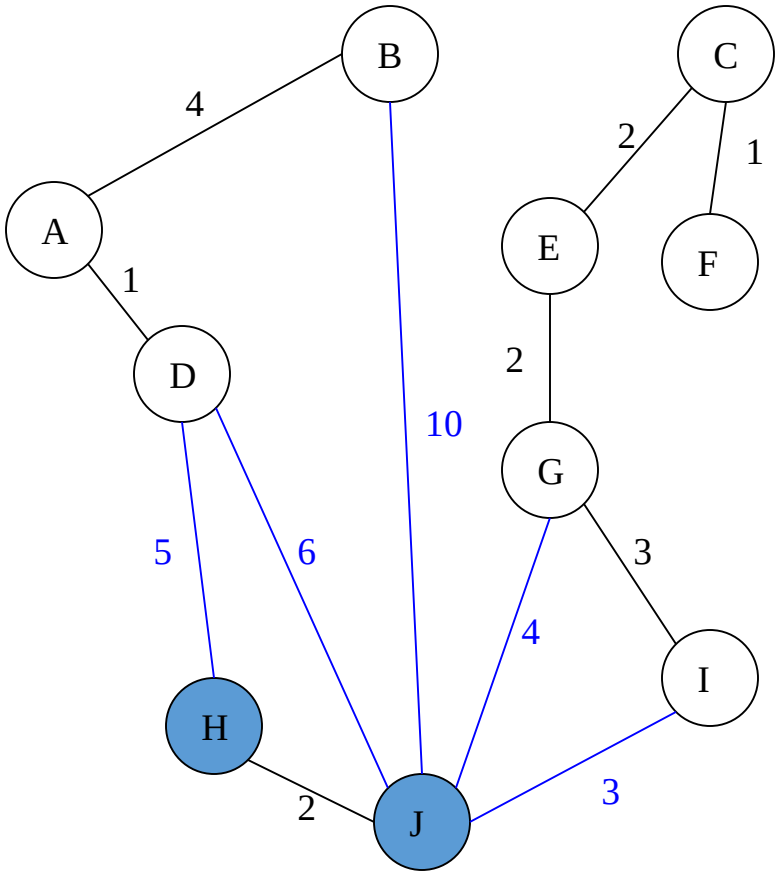
Edge I-J



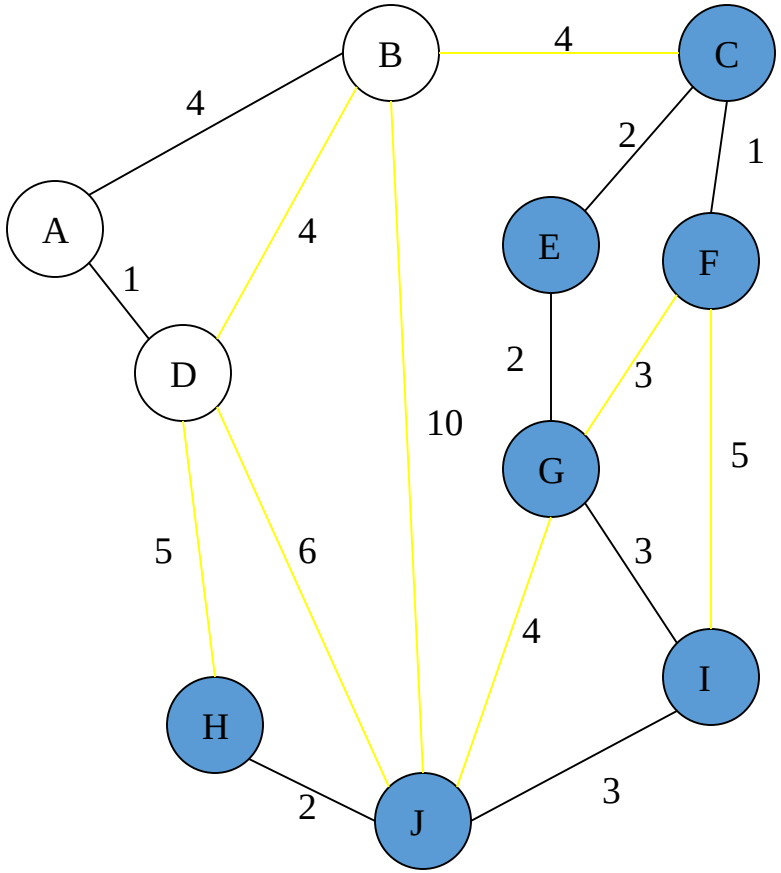
Round 2



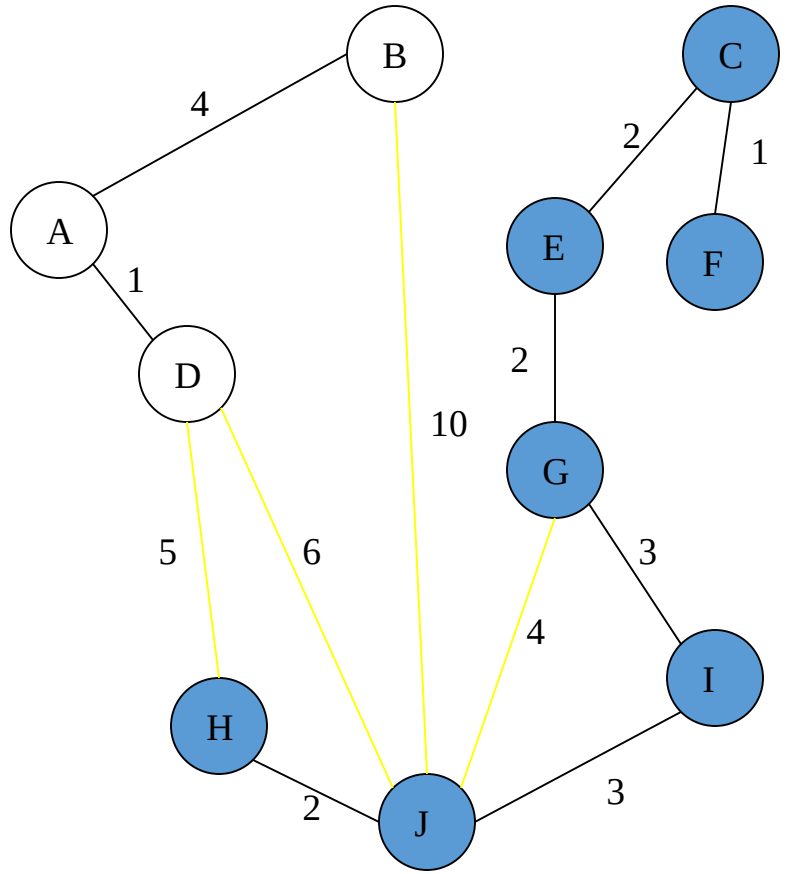
Tree H-J



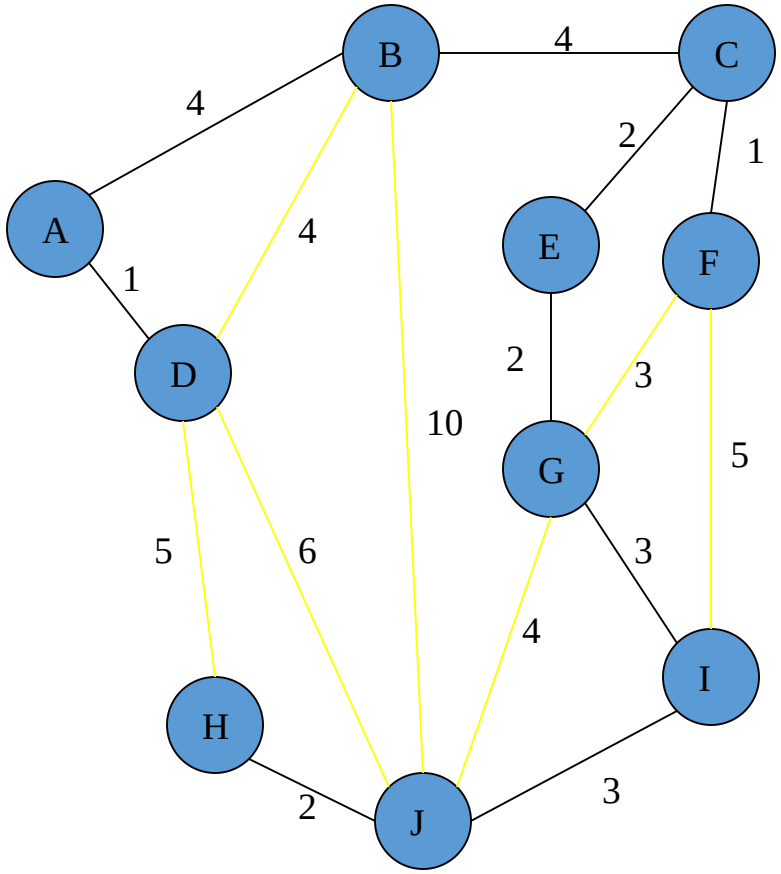
Round 2



Edge J-I



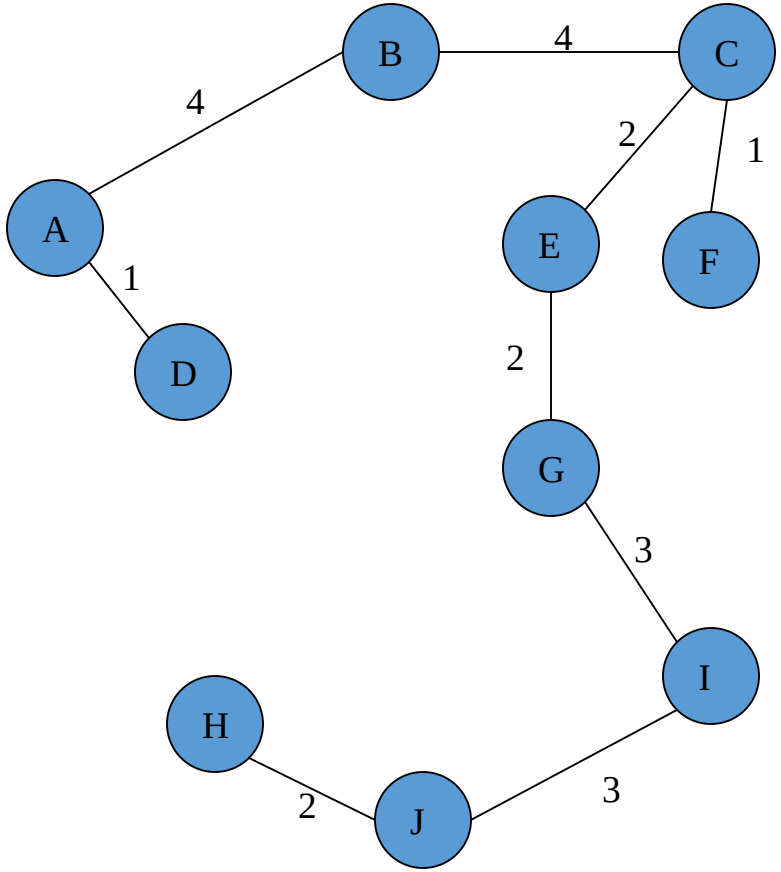
Round 2 Ends - Add Edges



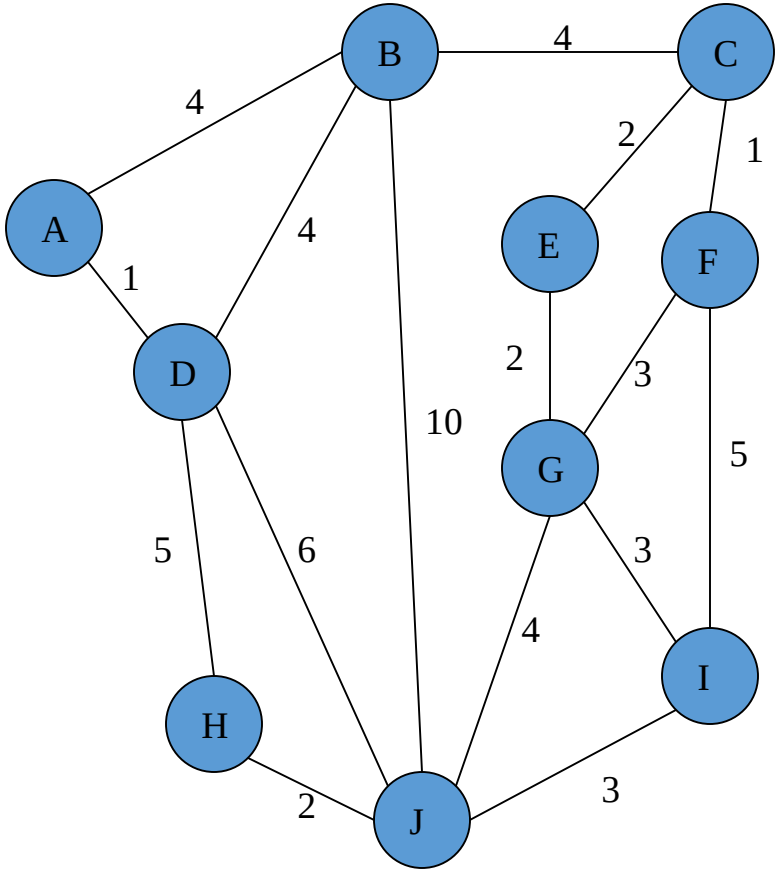
List of Edges to Add

- B-C
- I-J
- J-I

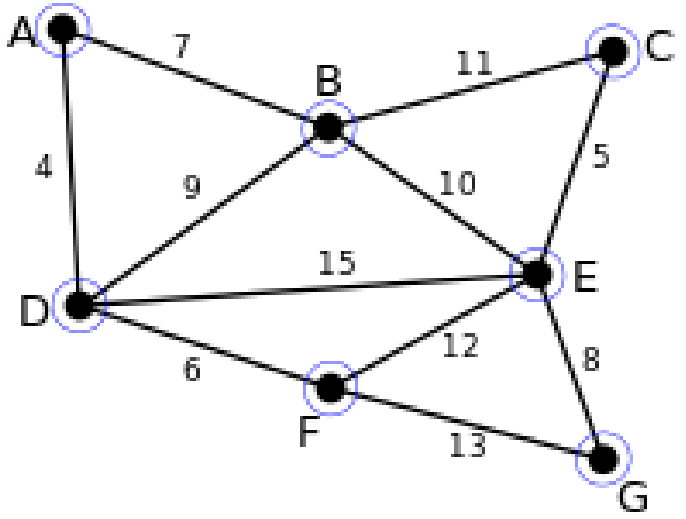
Minimum Spanning Tree

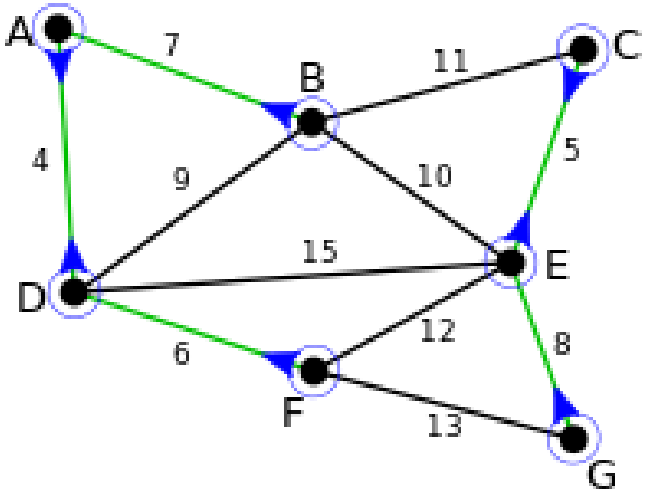


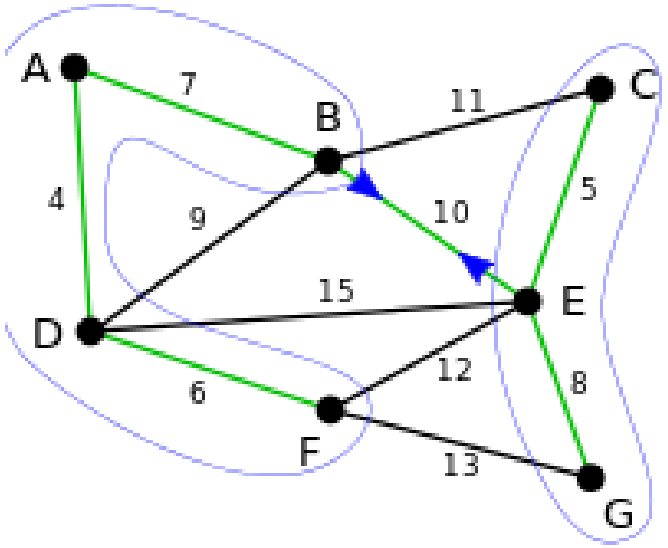
Complete Graph



Exercise: Using Boruvka's algorithm find the minimum spanning tree







Conclusion

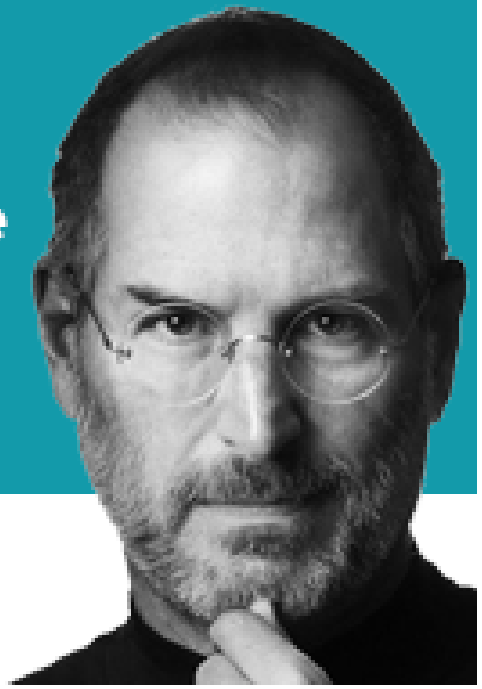
Kruskal's and Boruvka's have better running times if the number of edges is low, while Prim's has a better running time if both the number of edges and the number of nodes are low.

Boruvka's avoids the complicated data structures needed for the other two algorithms.

So, of course, the best algorithm depends on the graph and if you want to bear the cost of complex data structures.

"You can't connect the dots looking forward; you can only connect them looking backwards. So you have to trust that the dots will somehow connect in your future. You have to trust in something – your gut, destiny, life, karma, whatever. This approach has never let me down, and it has made all the difference in my life."

- STEVE JOBS



Useful links

- Dijkstra’s shortest path algorithm
  http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/
- Kruskal's algorithm example
  https://en.wikipedia.org/wiki/Kruskal%27s_algorithm#Example
- Kruskal's algorithm example
  https://www.youtube.com/watch?v=71UQH7Pr9kU
  Prim's algorithm example
  https://www.youtube.com/watch?v=cplfcGZmX7I