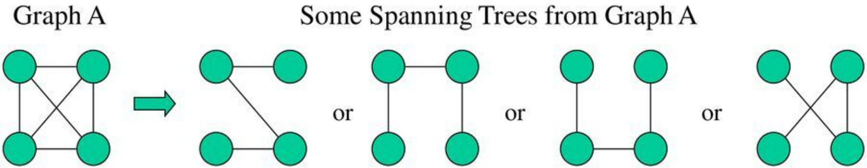
Minimum Spanning Tree

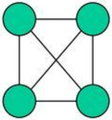
Spanning Trees

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

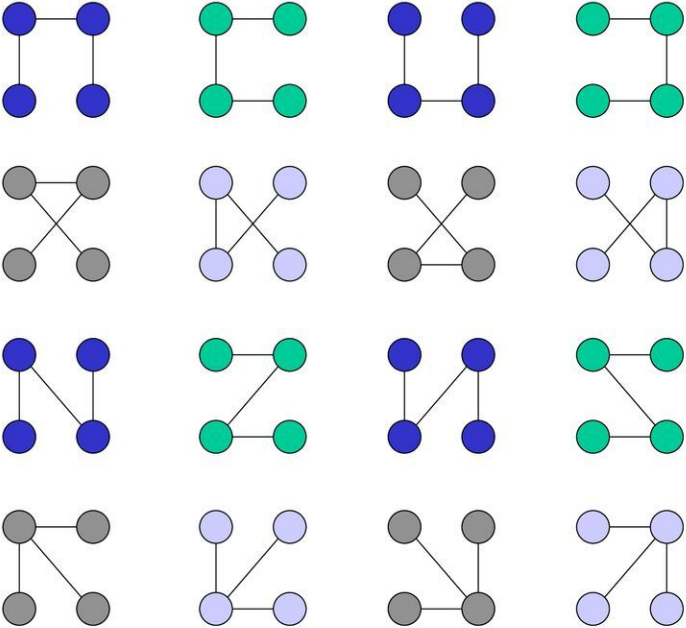
A graph may have many spanning trees.



Complete Graph



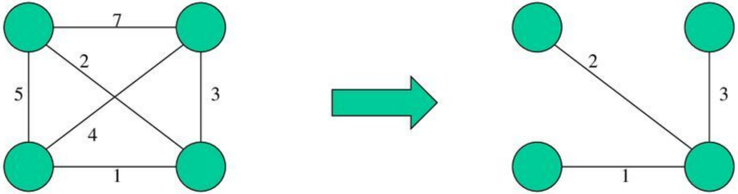
All 16 of its Spanning Trees



Minimum Spanning Trees

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

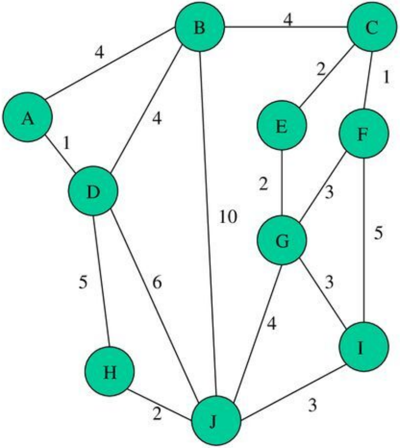
Complete Graph Minimum Spanning Tree

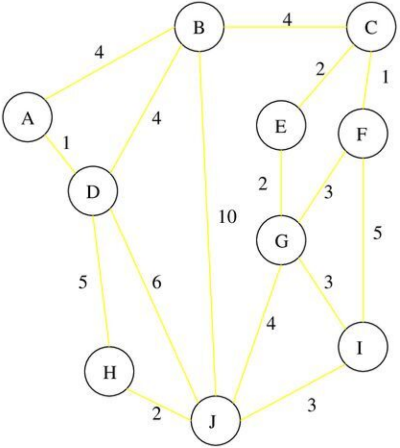


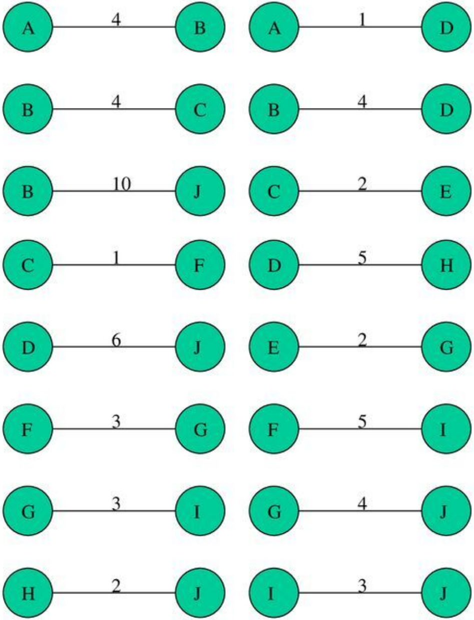
Algorithms for Obtaining the Minimum Spanning Tree

- Kruskal's Algorithm
- Prim's Algorithm
- Boruvka's Algorithm

Complete Graph

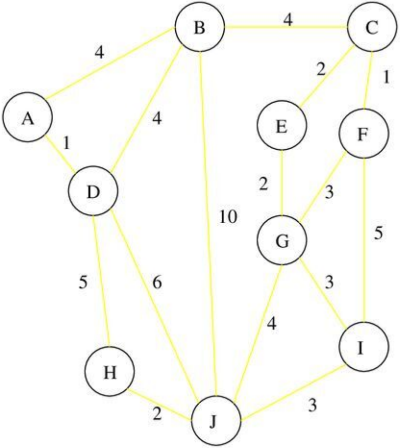


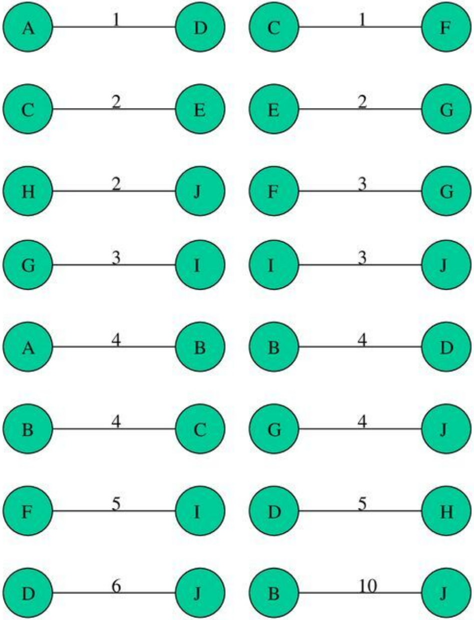

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
10
2
3
5
6
5
3
4
2
3


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

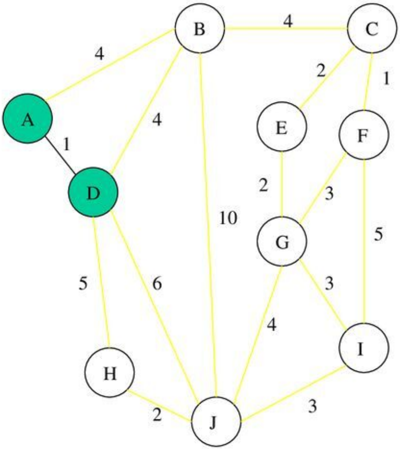
Sort Edges

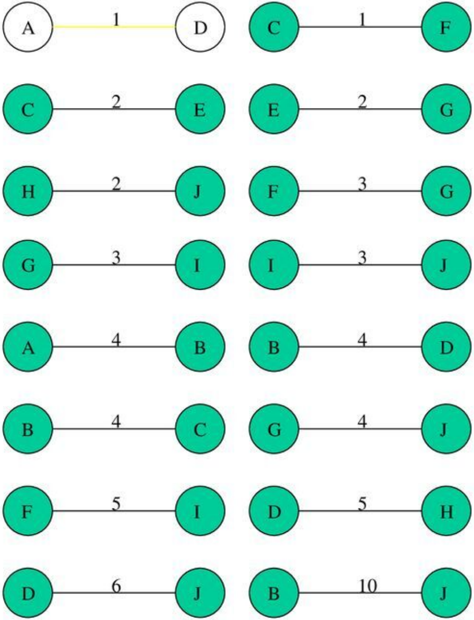
(in reality they are placed in a priority queue - not sorted - but sorting them makes the algorithm easier to visualize)



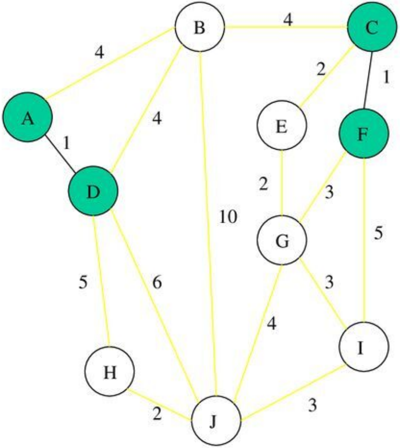


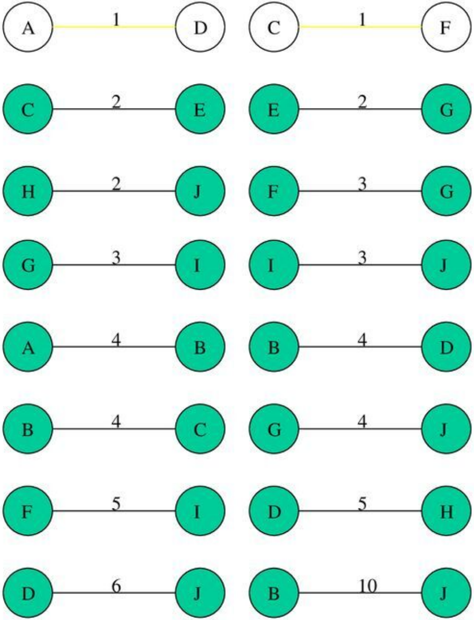
Add Edge


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

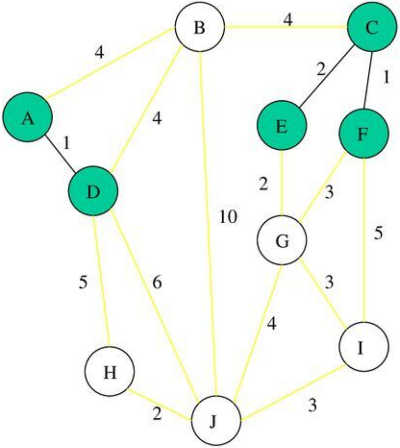

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

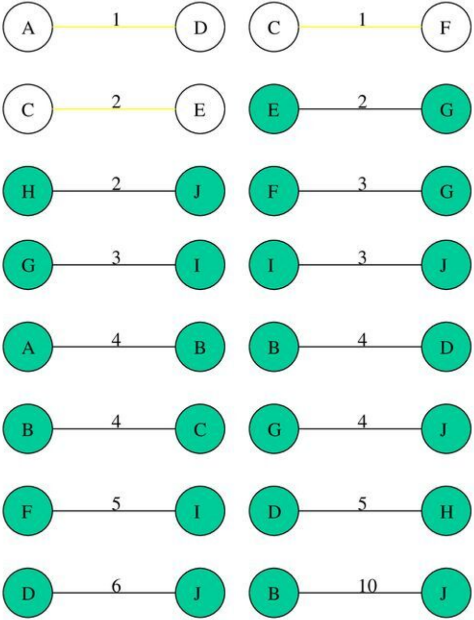
Add Edge


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

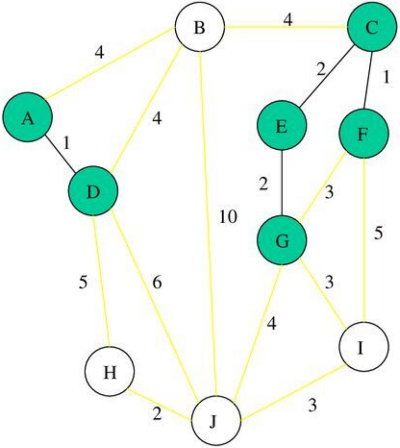

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge

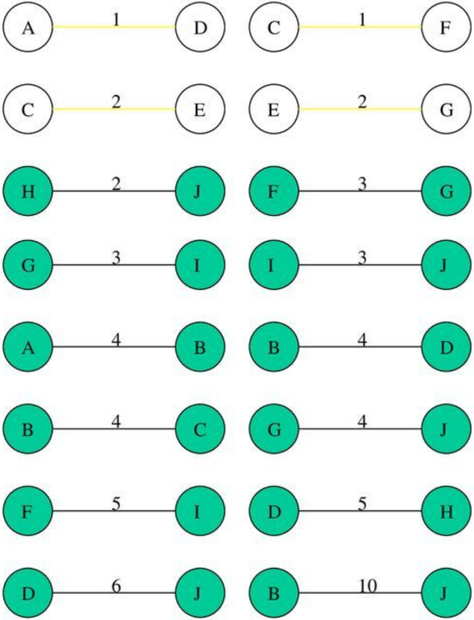

A
B
C
D
E
F
G
H
I
J
4
4
1
2
1
4
10
2
3
5
5
6
4
3
2
3


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

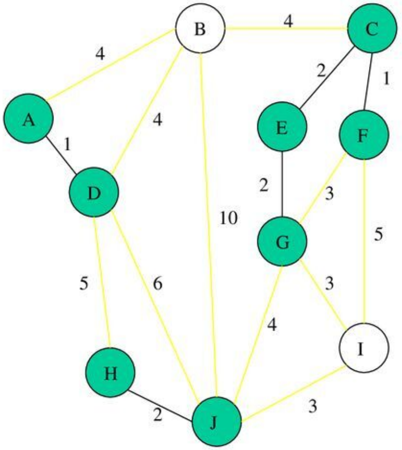


B
C
A
E
F
D
G
H
J
I
4
4
2
1
4
1
2
3
10
5
5
6
3
4
2
3

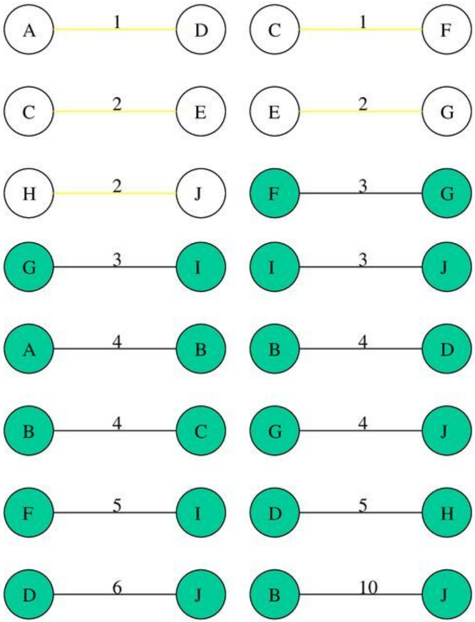


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge



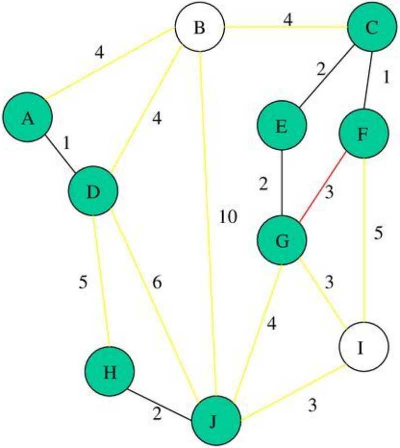
A
B
C
D
E
F
G
H
I
J
4
4
4
2
1
1
10
2
3
5
6
5
3
4
3
2

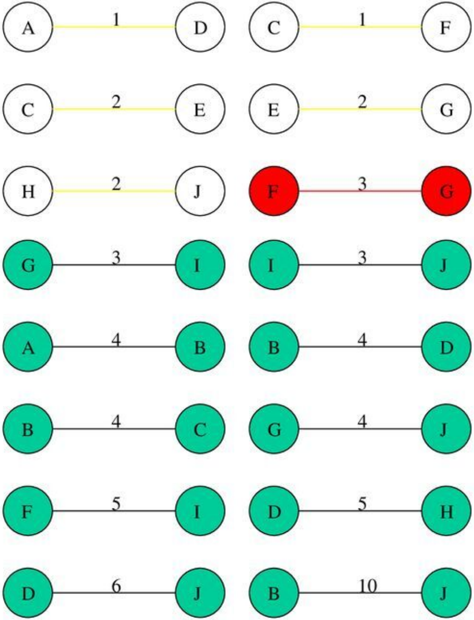


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

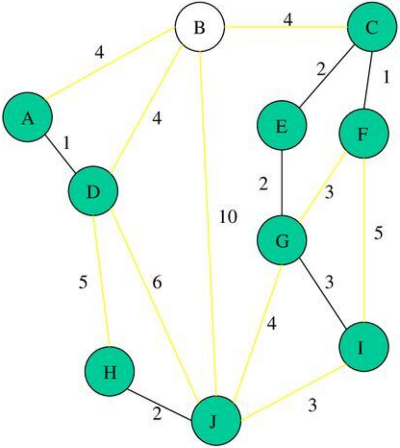
Cycle

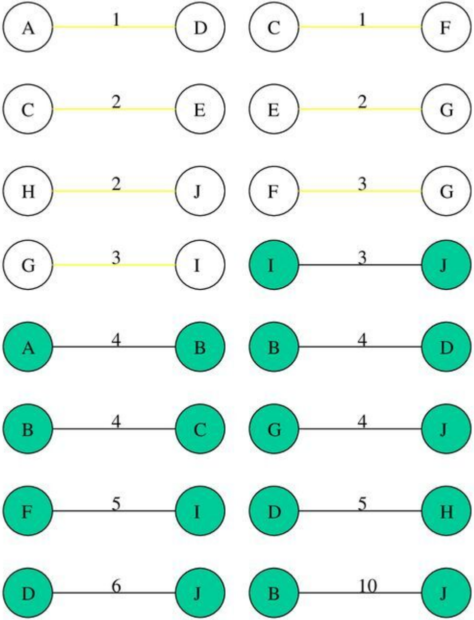
Don't Add Edge



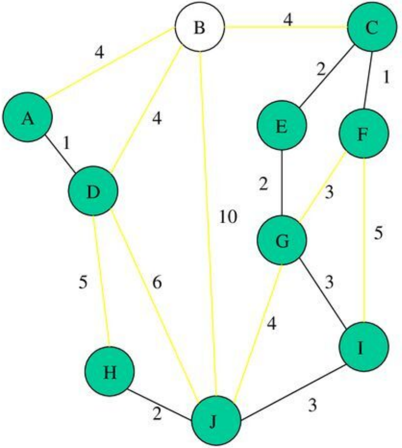


Add Edge

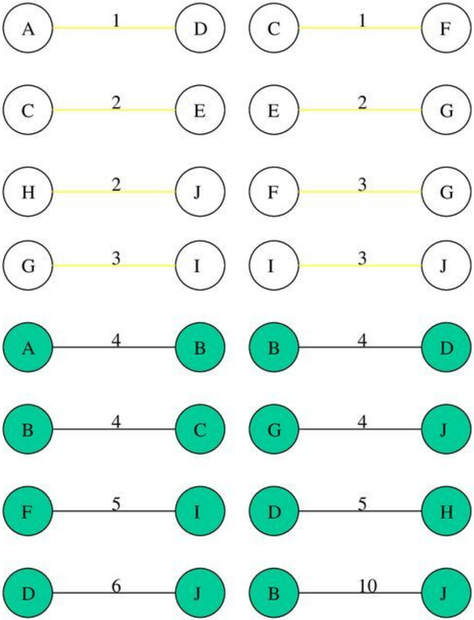

B
4
C
4
2
1
A
4
E
F
1
D
2
3
10
5
G
5
6
3
4
I
H
2
J
3


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

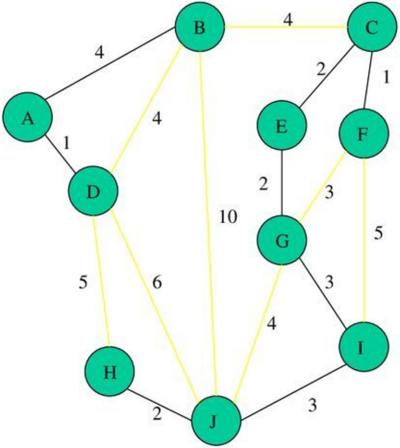


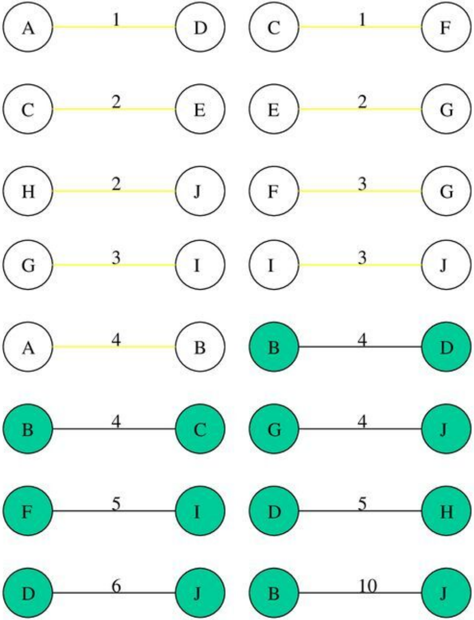
B
4
C
4
2
1
A
4
E
F
1
D
2
3
10
G
5
5
6
3
4
I
H
2
J
3



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

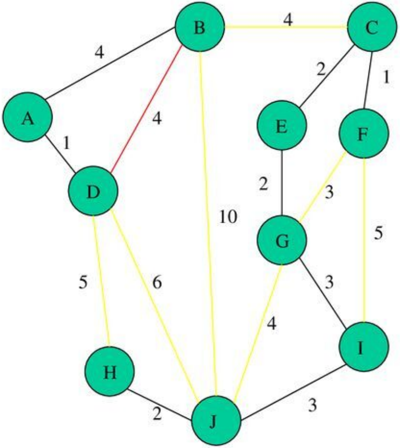
Add Edge

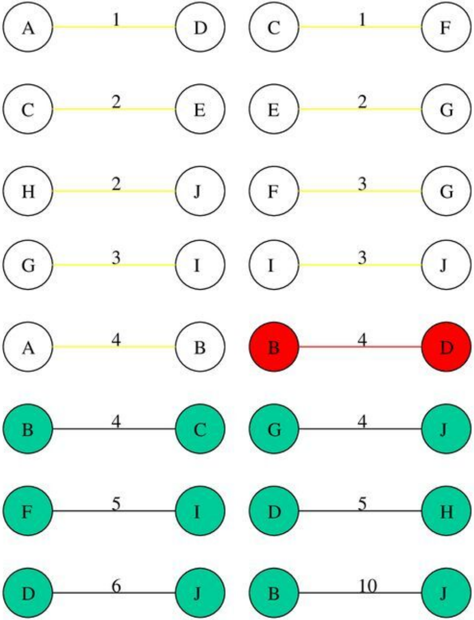

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
2
3


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

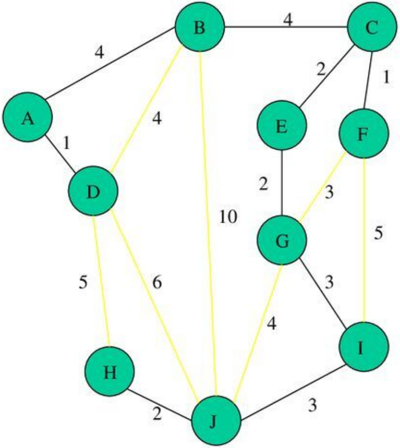
Cycle

Don’t Add Edge

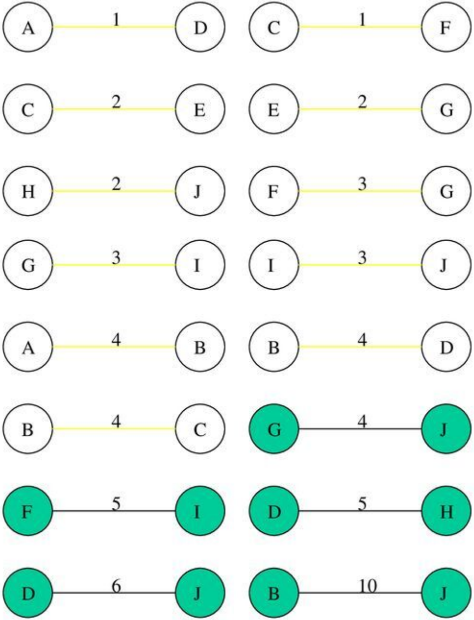




Add Edge

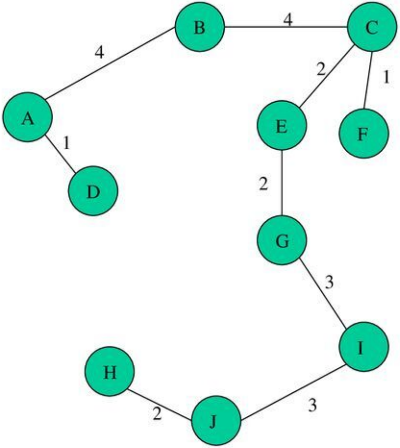


A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
10
2
3
5
5
6
3
4
2
3

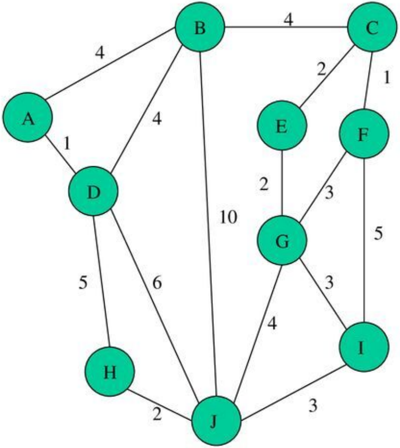


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Minimum Spanning Tree



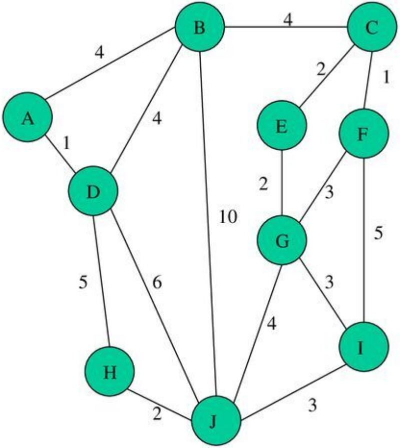
Complete Graph



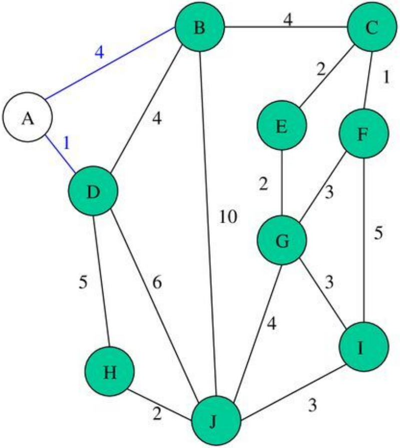
Prim's Algorithm

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

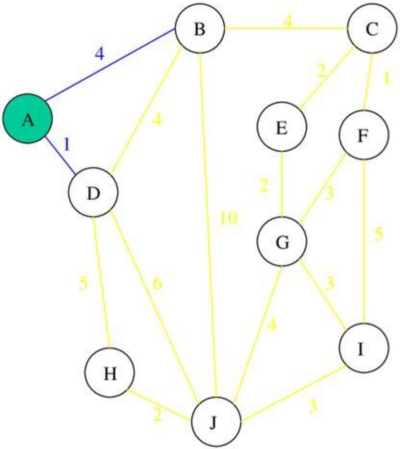
Complete Graph



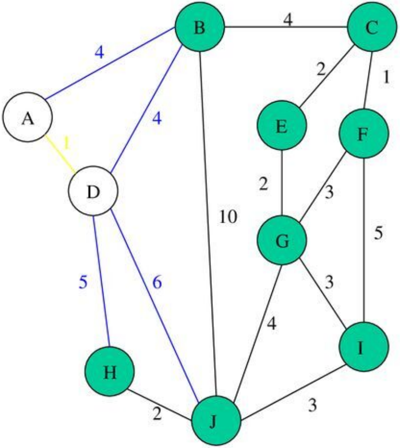
Old Graph



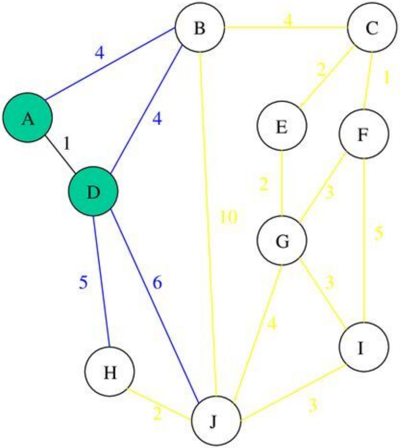
New Graph



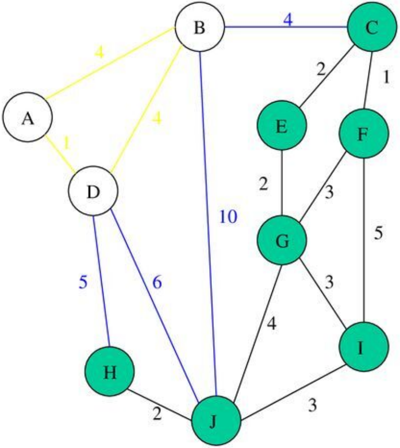
Old Graph



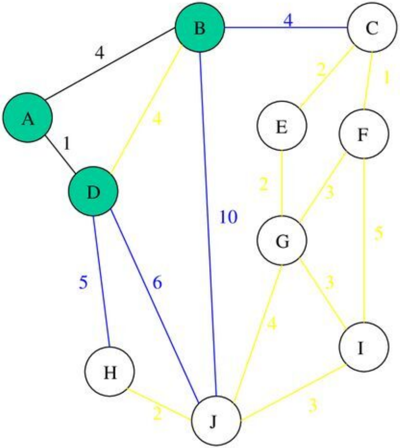
New Graph



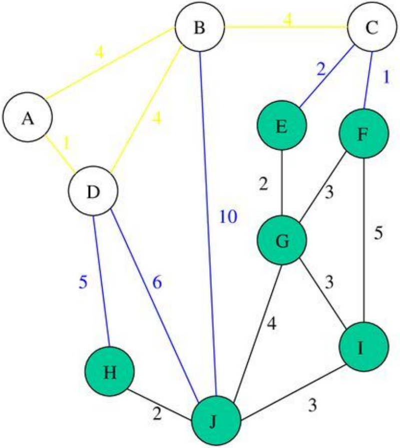
Old Graph



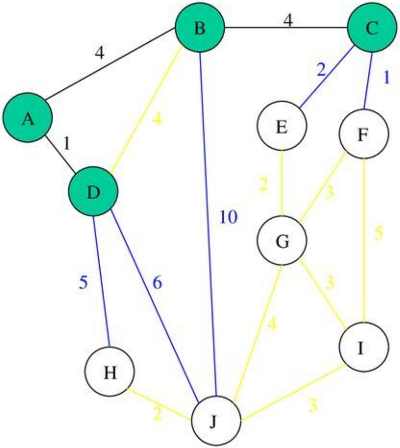
New Graph



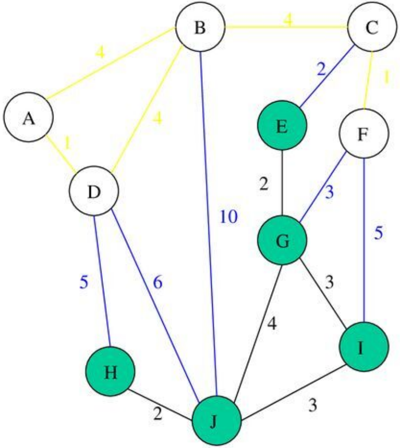
Old Graph



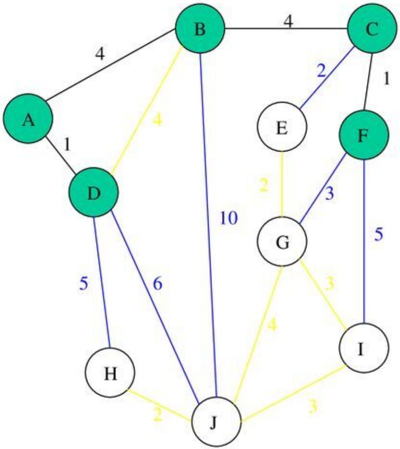
New Graph



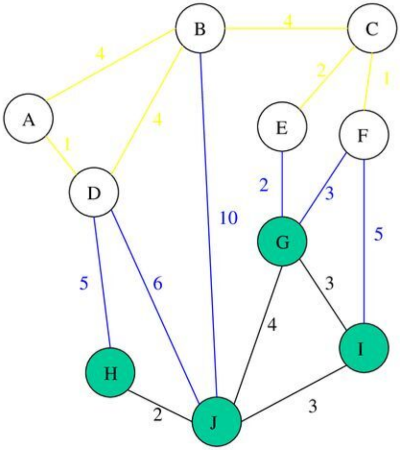
Old Graph



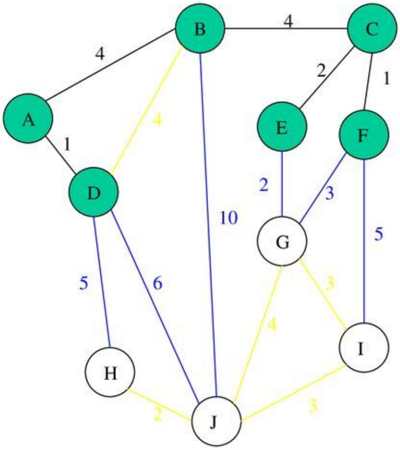
New Graph



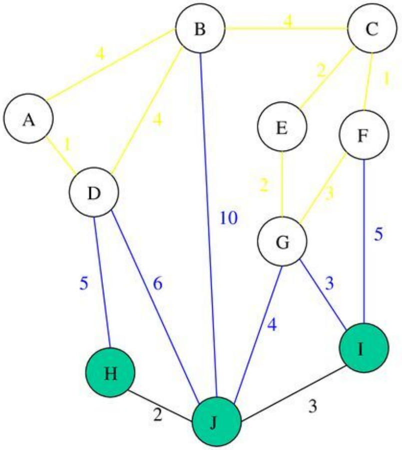
Old Graph



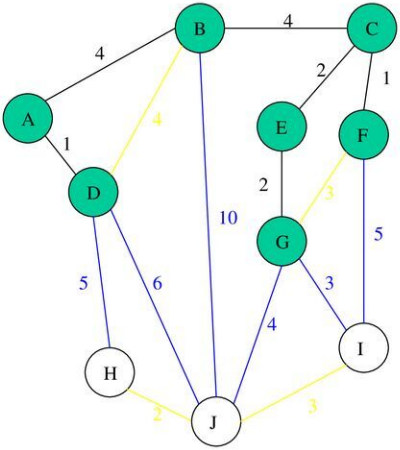
New Graph



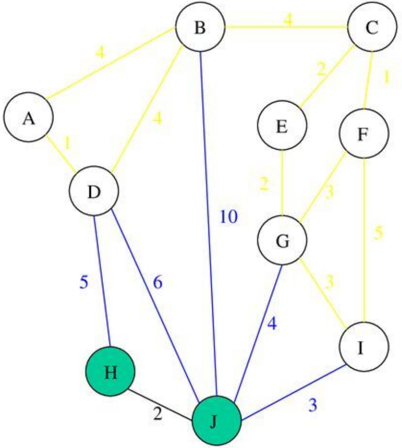
Old Graph



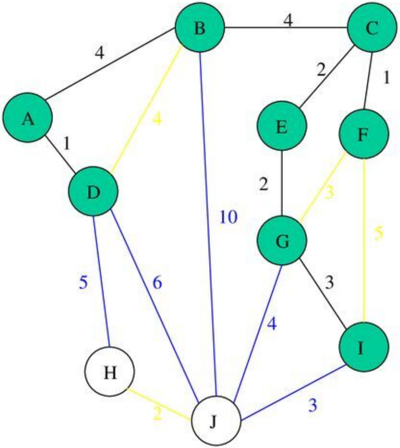
New Graph



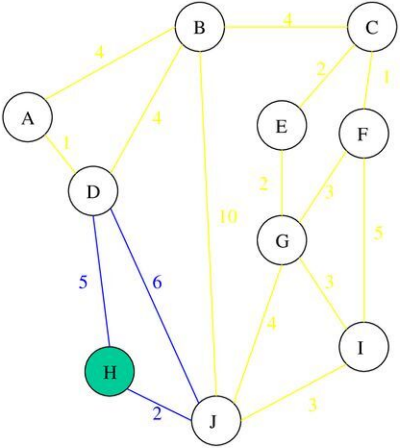
Old Graph



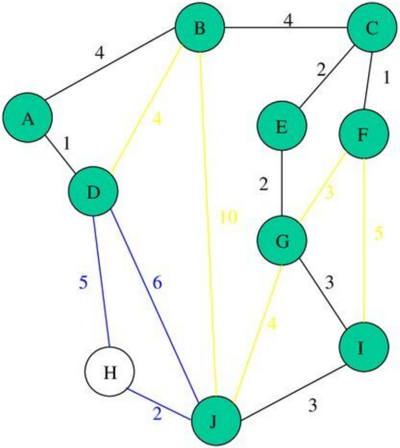
New Graph



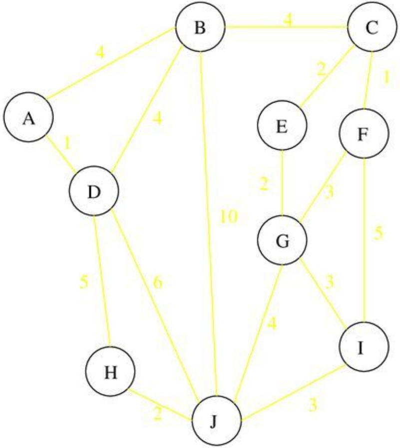
Old Graph



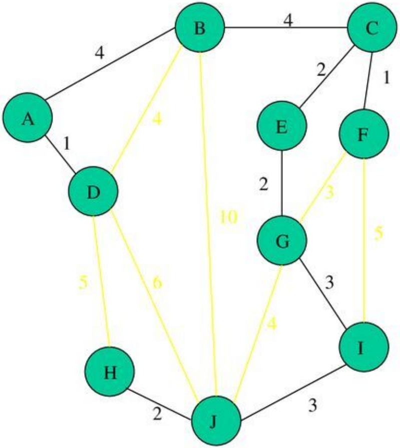
New Graph



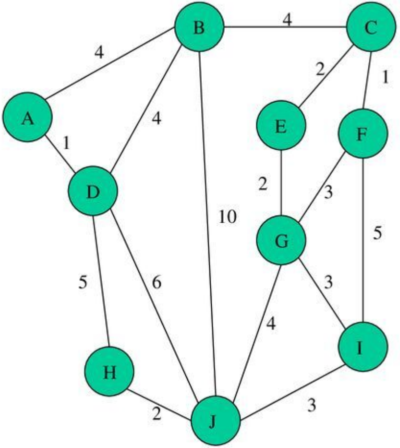
Old Graph



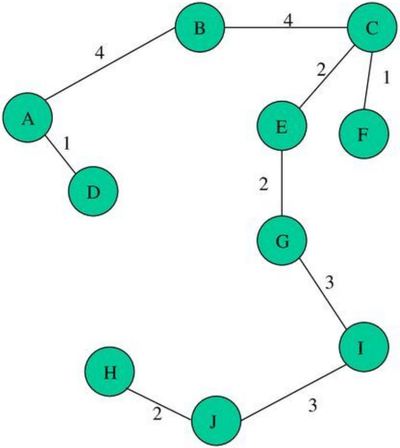
New Graph



Complete Graph



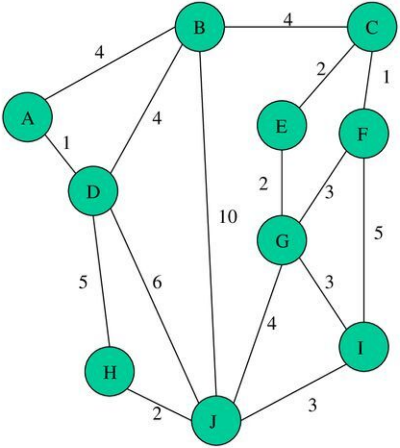
Minimum Spanning Tree



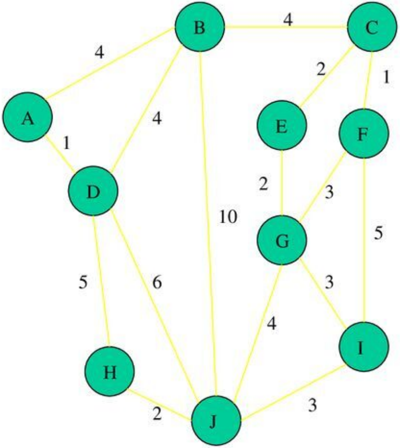
Boruvka's Algorithm

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.

Complete Graph



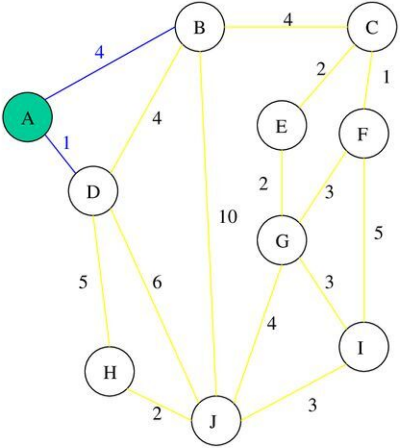
Trees of the Graph at Beginning of Round 1



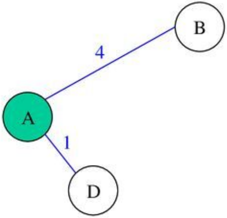
List of Trees

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

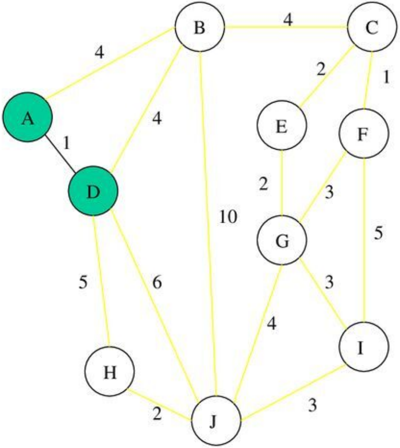
Round 1



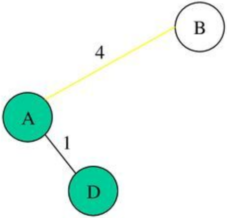
Tree A



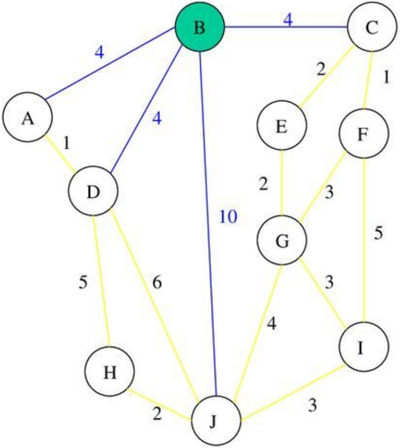
Round 1



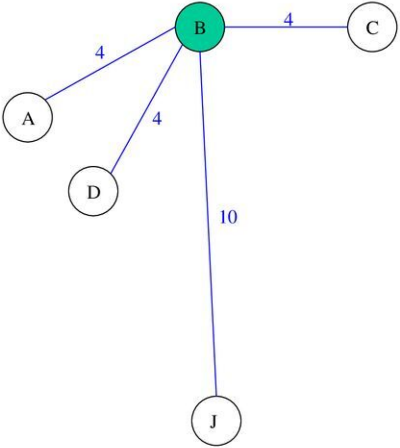
Edge A-D



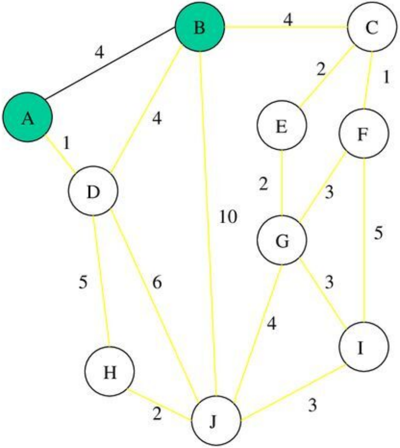
Round 1



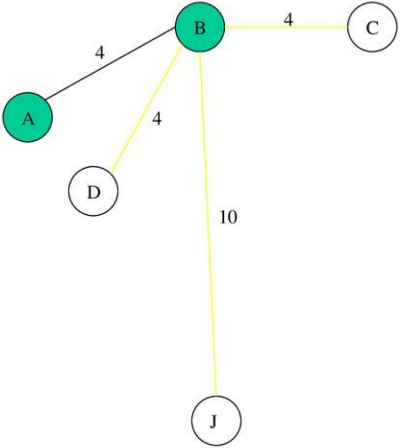
Tree B



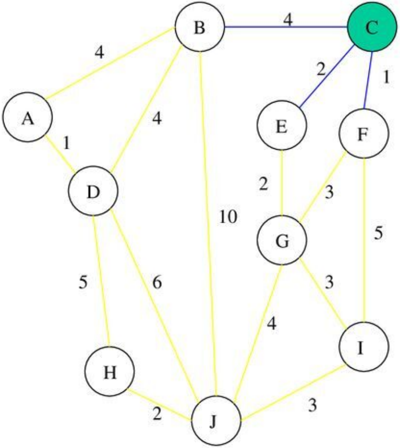
Round 1



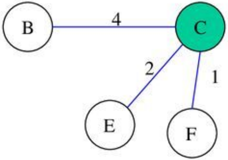
Edge B-A



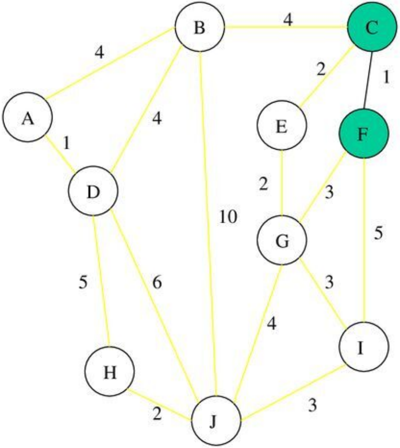
Round 1



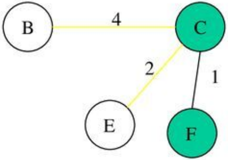
Tree C



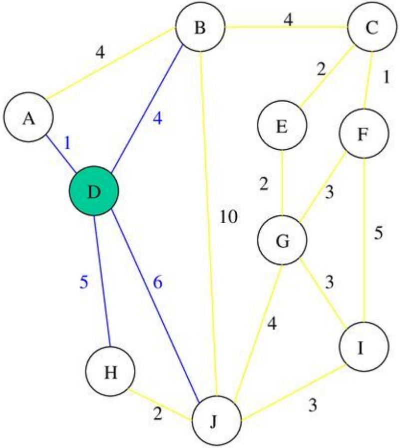
Round 1



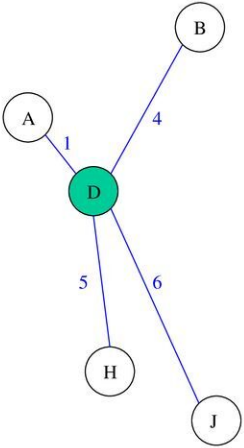
Edge C-F



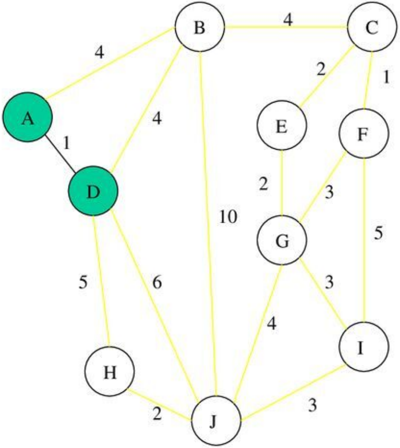
Round 1



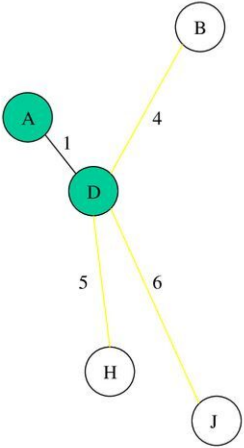
Tree D



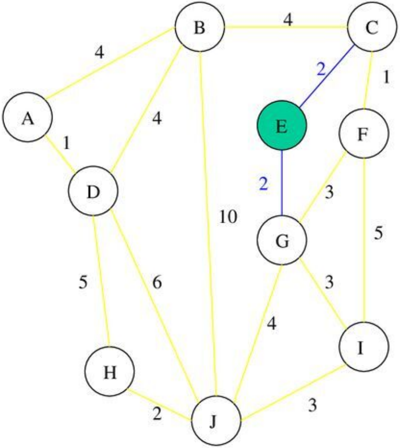
Round 1



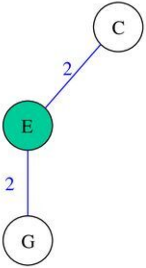
Edge D-A



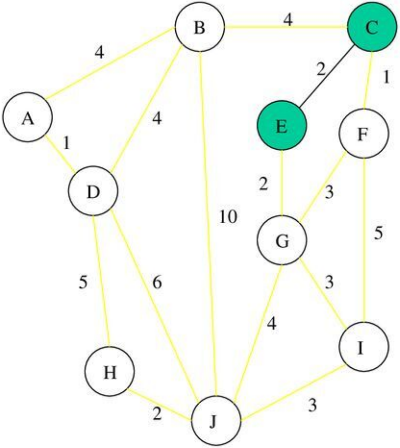
Round 1



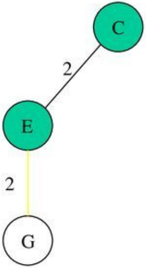
Tree E



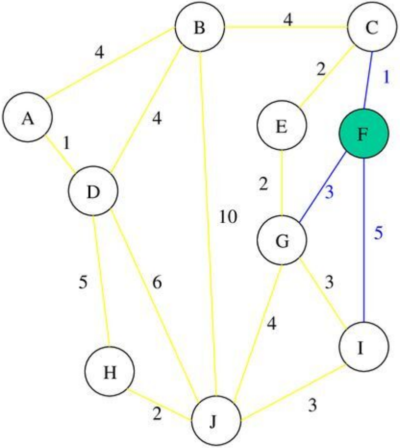
Round 1



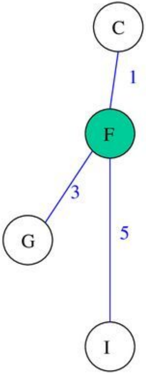
Edge E-C



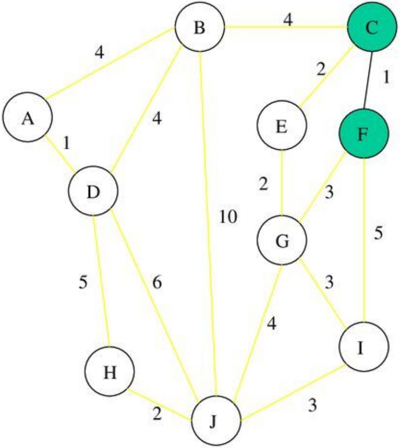
Round 1



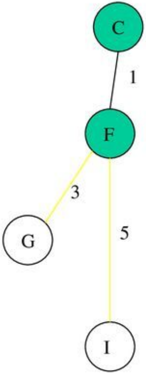
Tree F



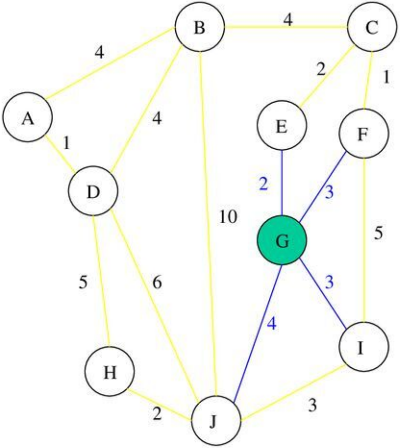
Round 1



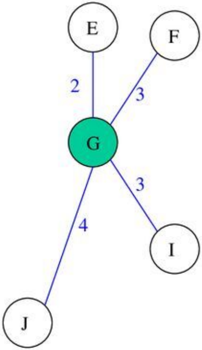
Edge F-C



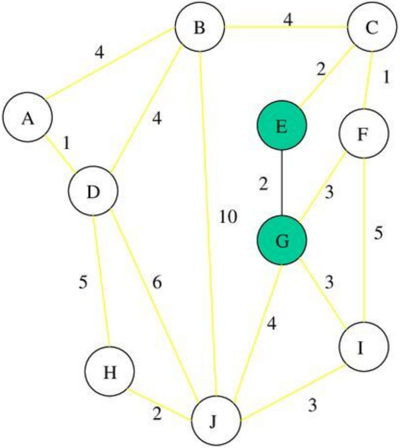
Round 1



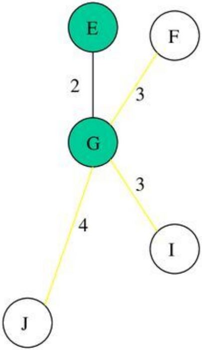
Tree G



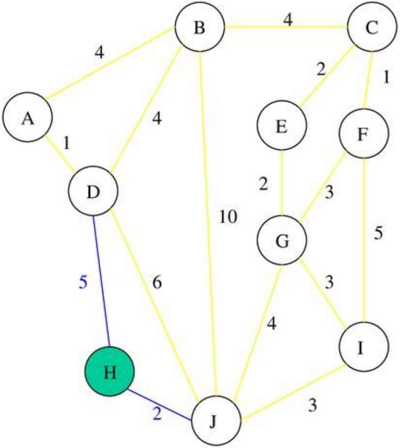
Round 1



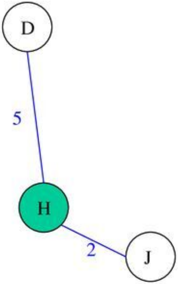
Edge G-E



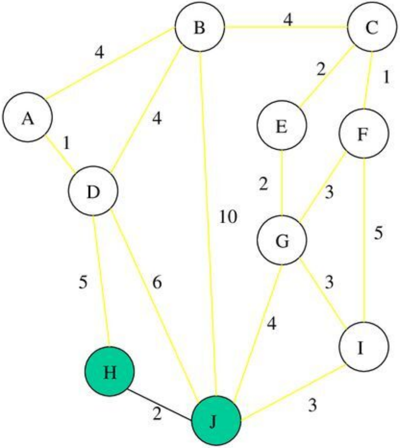
Round 1



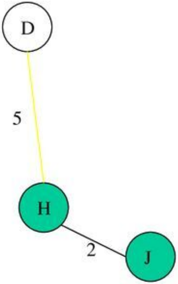
Tree H



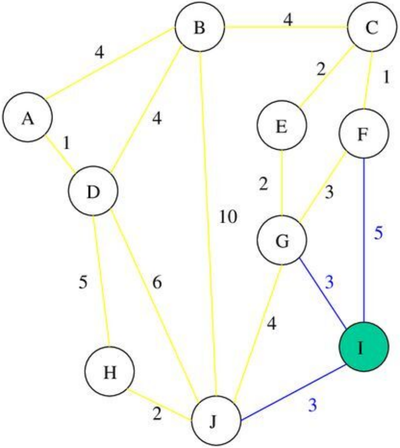
Round 1



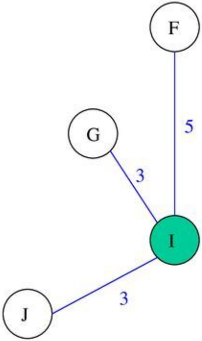
Edge H-J



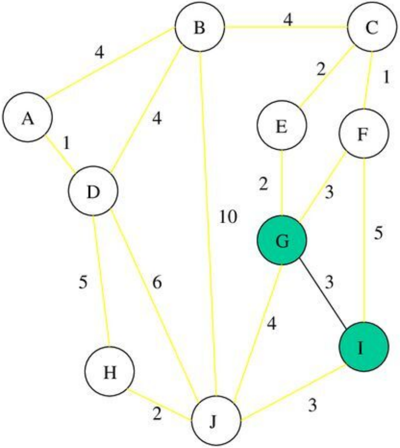
Round 1



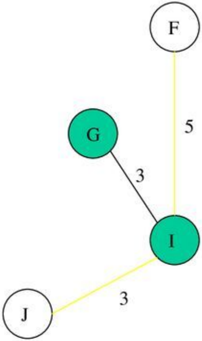
Tree I



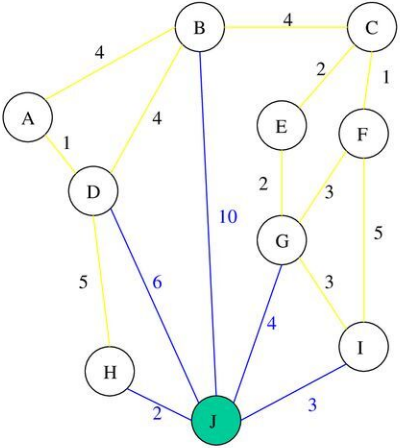
Round 1



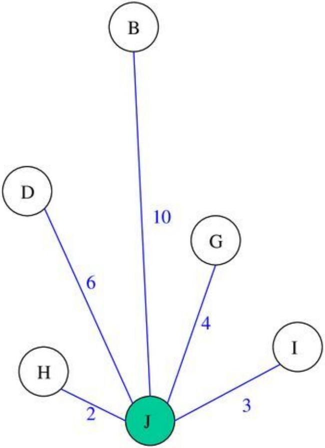
Edge I-G



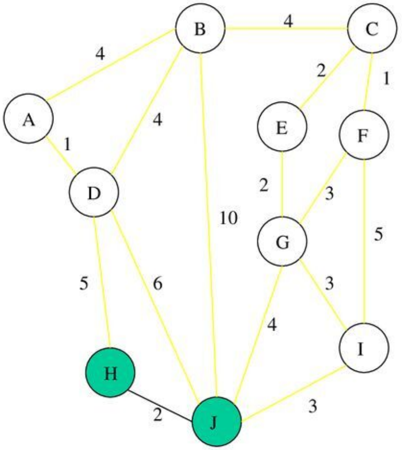
Round 1



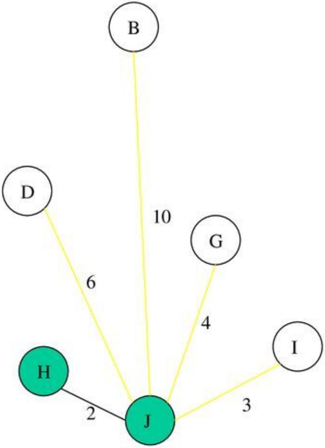
Tree J



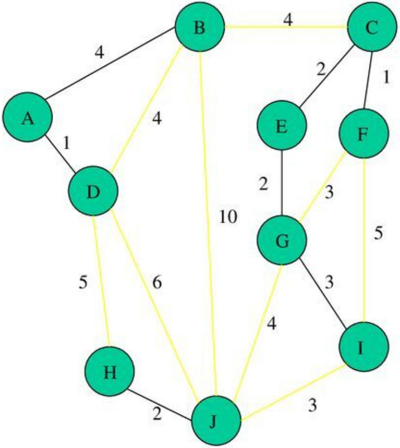
Round 1



Edge J-H



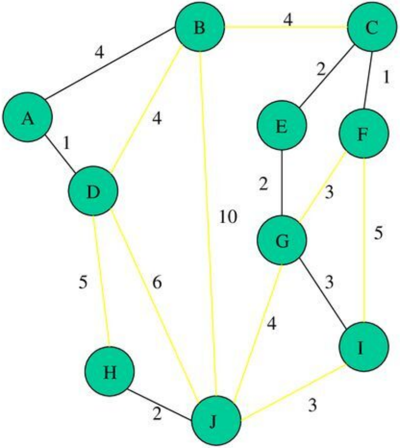
Round 1 Ends - Add Edges



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

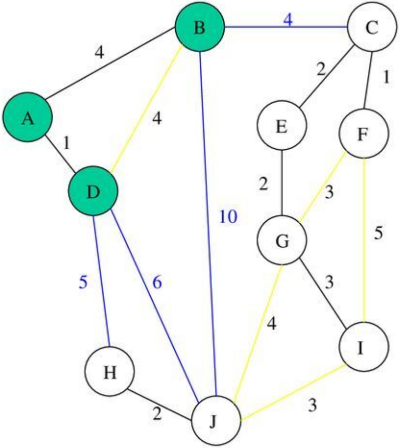
Trees of the Graph at Beginning of Round 2



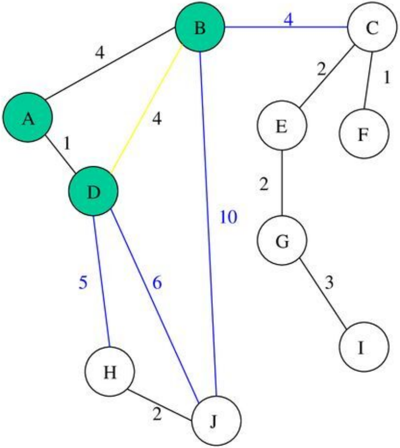
List of Trees

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

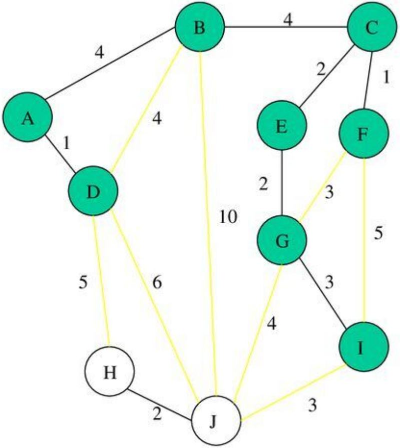
Round 2



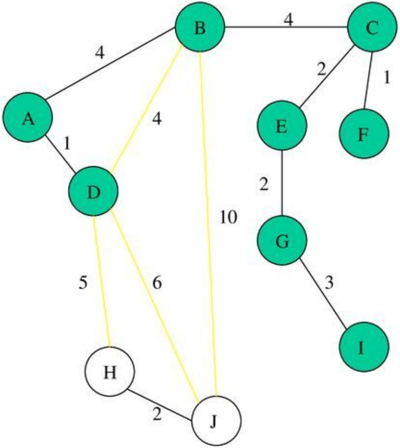
Tree D-A-B



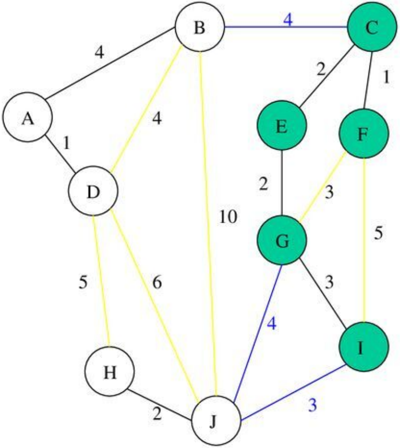
Round 2



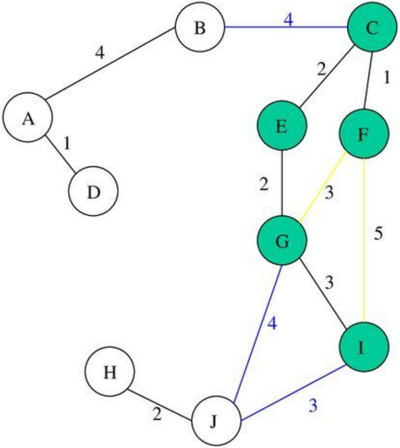
Edge B-C



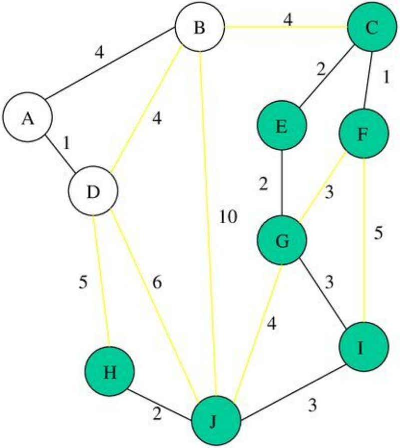
Round 2



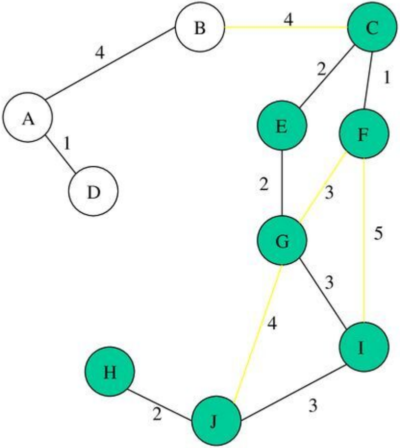
Tree F-C-E-G-I



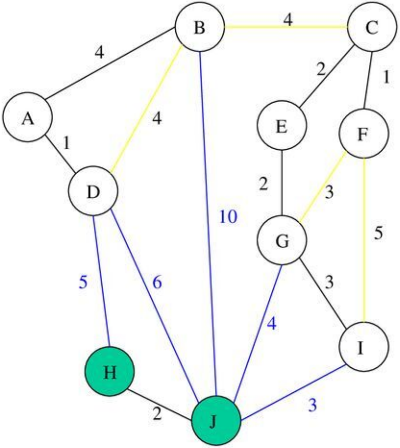
Round 2



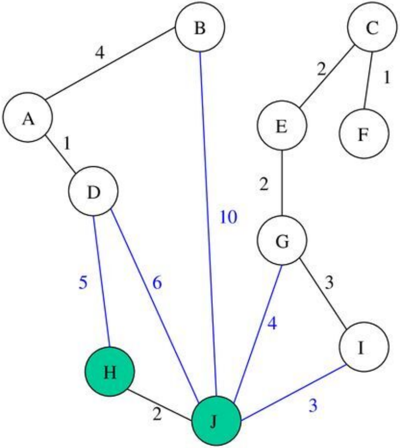
Edge I-J



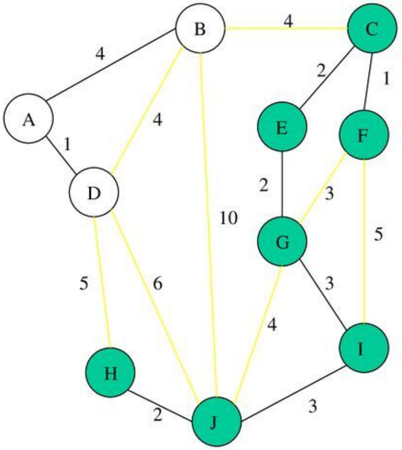
Round 2



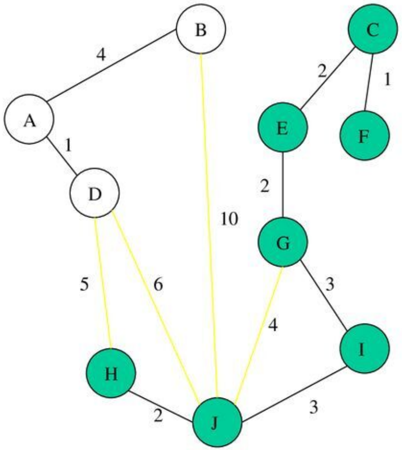
Tree H-J



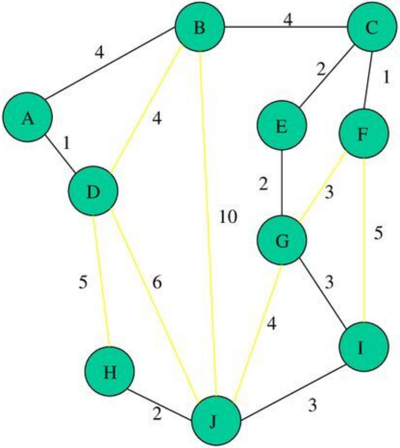
Round 2



Edge J-I



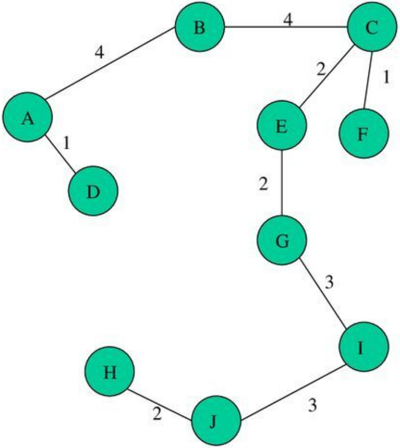
Round 2 Ends - Add Edges



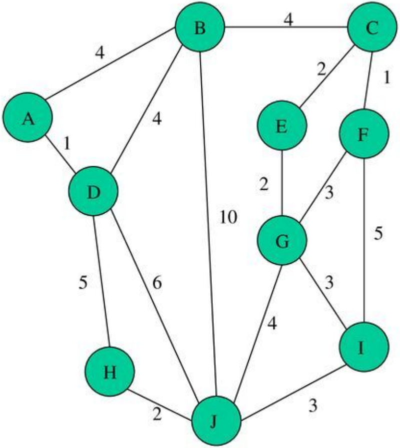
List of Edges to Add

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

Minimum Spanning Tree



Complete Graph



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.

The best algorithm that I know of is a hybrid of Boruvka's and Prim's, which I did not examine here. It does O(log log n) passes of Boruvka's and then switches to Prim's, resulting in a running time of O(m log log n). So, it's the fastest algorithm, but would, of course, require the Fibonacci heap for Prim's which Boruvka's avoids when used by itself. However, in order to keep things simple, I did not explore it here.