CSEN3001:
DESIGN AND ANALYSIS OF ALGORITHMS

UNIT-II: Minimum Spanning Trees

Greedy Method – general idea

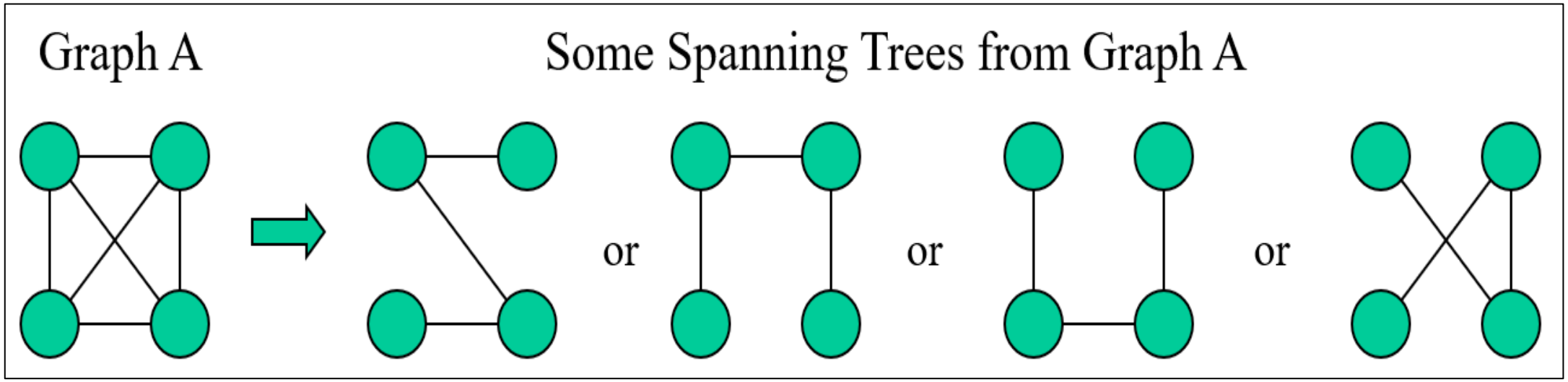
- The Greedy method is the most straightforward designed technique.
- These methods are short-sighted in their approach, deciding based on the information immediately at hand without worrying about the effect this decision may have in the future.

DEFINITION:

- A problem with N inputs will have some constraints, and any subsets that satisfy these constraints are called feasible solutions.
- An optimal solution is a feasible solution that either maximizes or minimizes a given objective function.

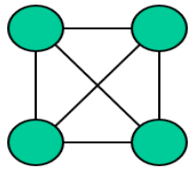
Spanning Tree– general idea

- A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.
- Let G=(V,E) be an undirected connected graph. A subgraph T=(V,E1) of G is a spanning tree of G iff T is a tree
- A graph may have many spanning trees.

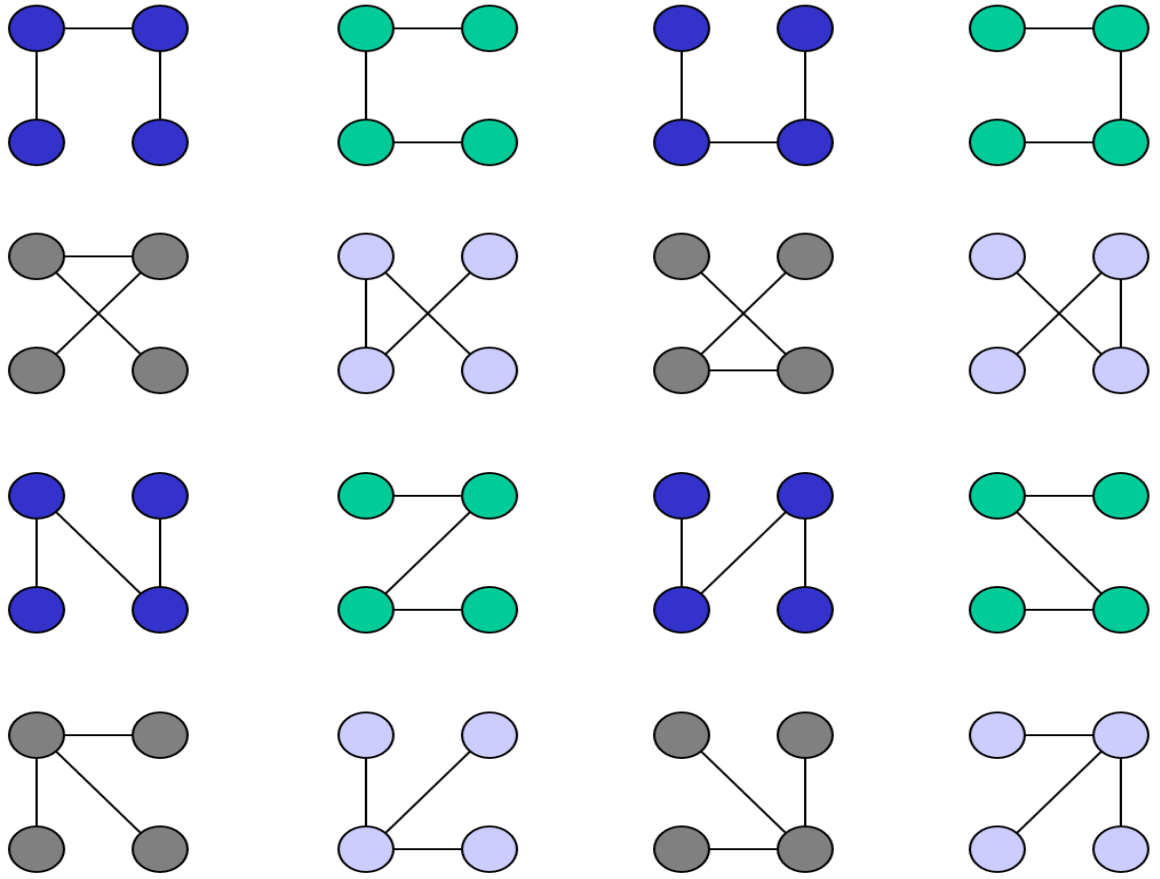


Spanning Tree – Example

Complete Graph

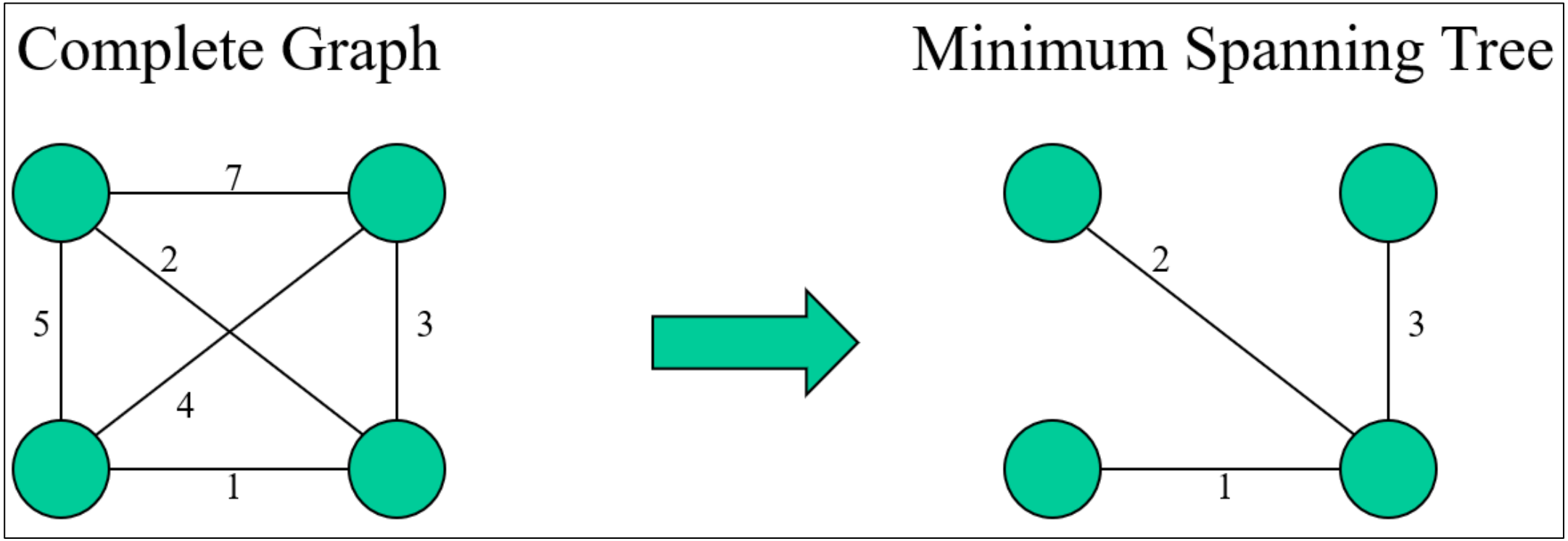


All 16 of its Spanning Trees



Minimum Spanning Tree– general idea

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



Minimum Spanning Tree– Algorithms

- Kruskal's Algorithm
- Prim's Algorithm

Minimum Spanning Tree– Krushkal's Algorithms

- This algorithm creates a forest of trees.
- Initially, the forest consists of n single-node trees (with 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.

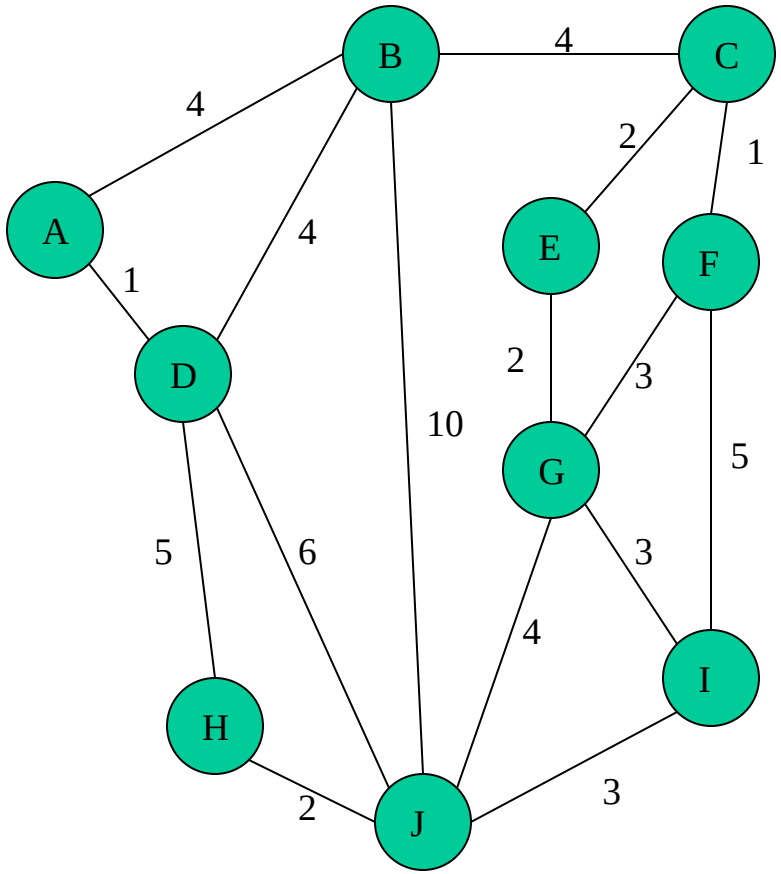
Minimum Spanning Tree– Krushkal's Algorithms

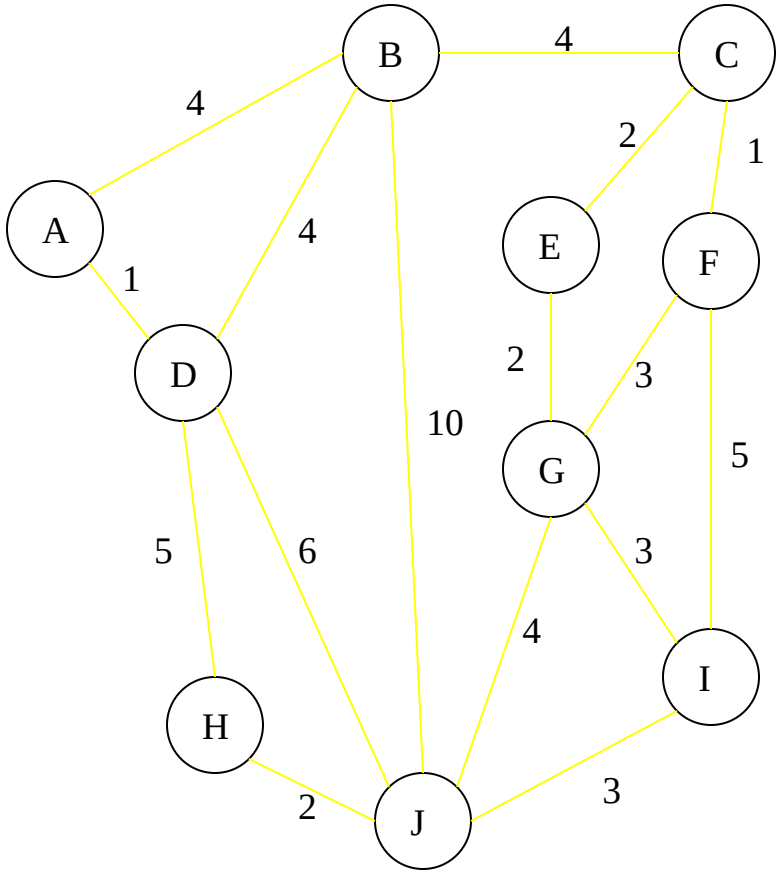
The steps are:

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,
   a. Extract the cheapest edge from the queue,
   b. If it forms a cycle, reject it,
   c. Else add it to the forest. Adding it to the forest will join two trees together.

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

Complete Graph





A 4 B

A 1 D

B 4 C

B 4 D

B 10 J

C 2 E

C 1 F

D 5 H

D 6 J

E 2 G

F 3 G

F 5 I

G 3 I

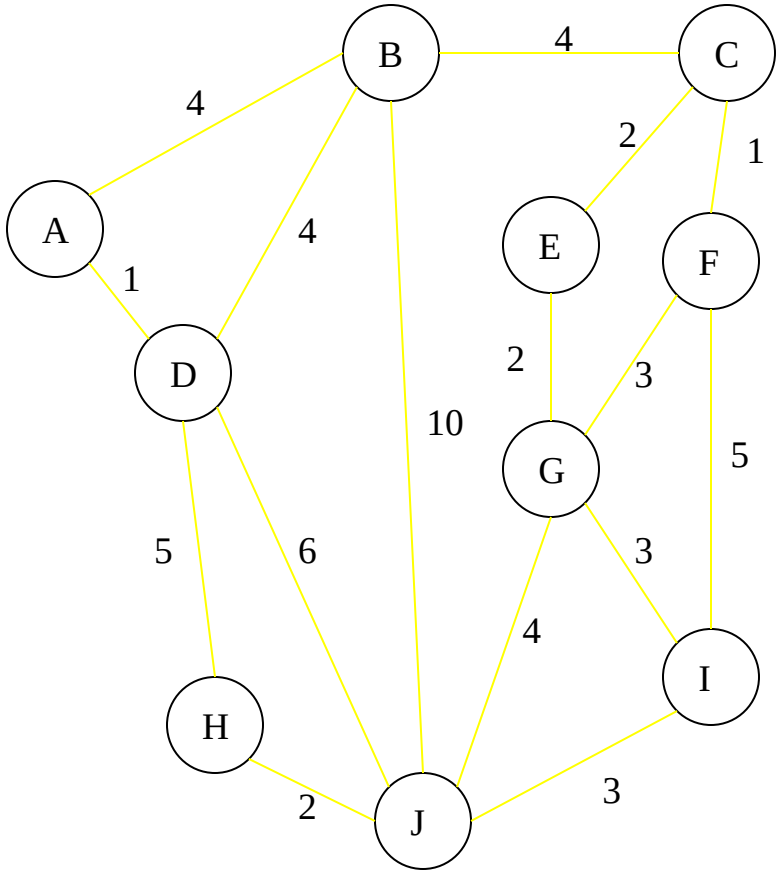
G 4 J

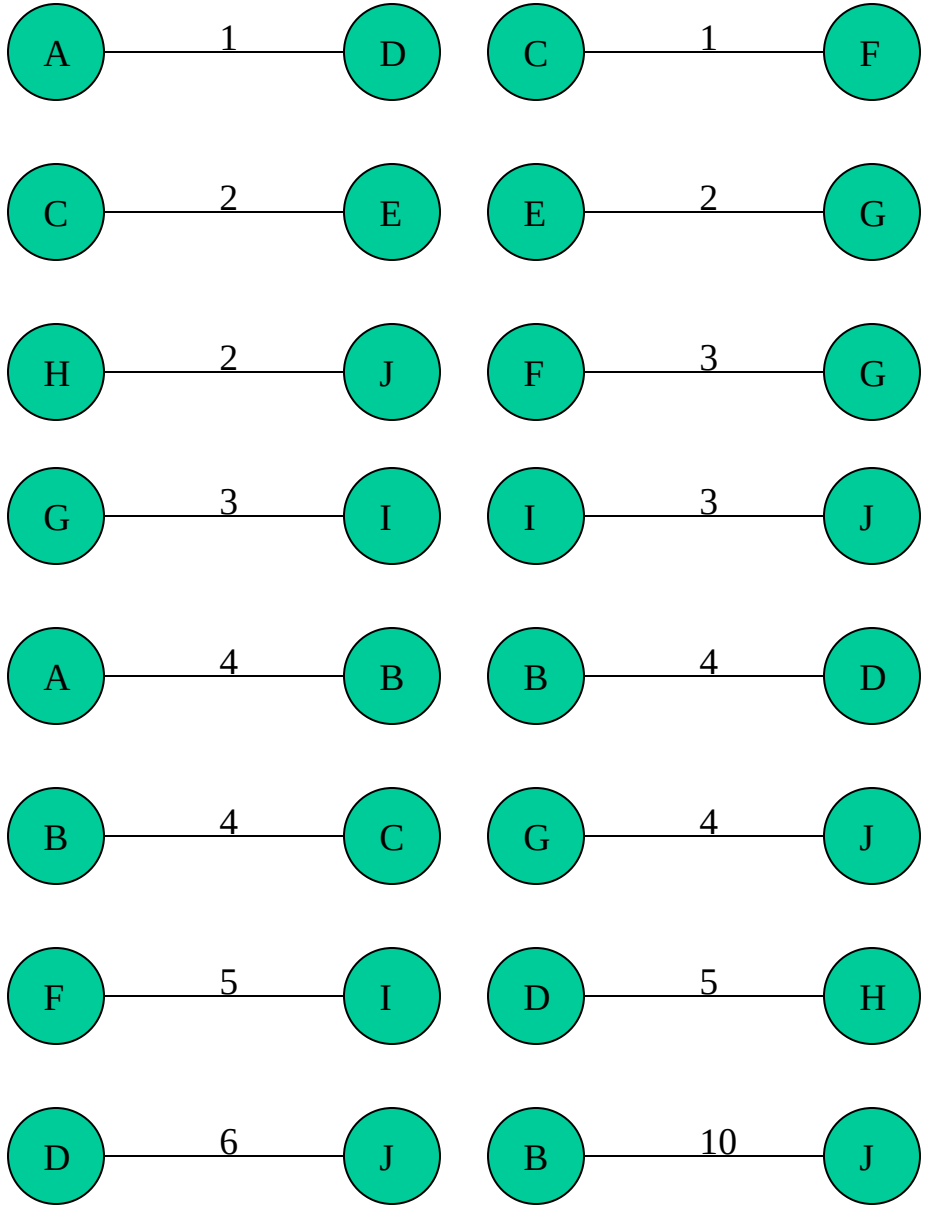
H 2 J

I 3 J

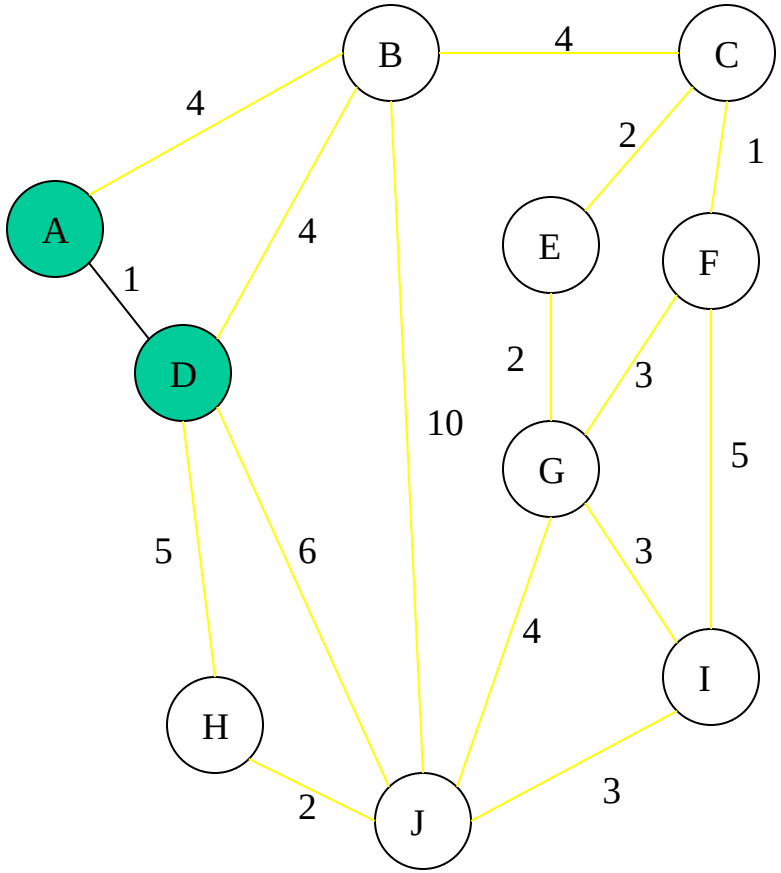
Sort Edges

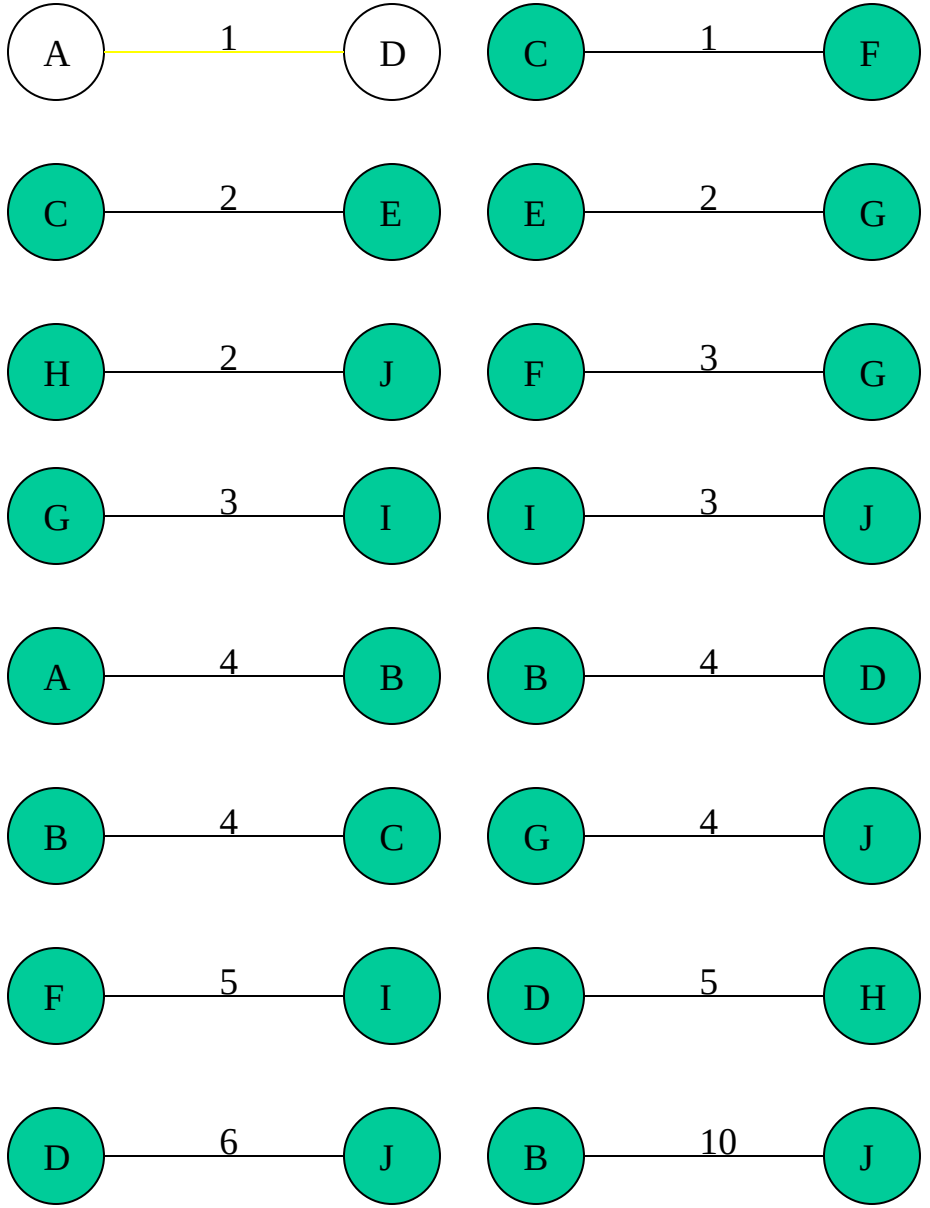
(in reality they are placed in a priority queue - not sorted - but sorting them makes the algorithm easier to visualize)



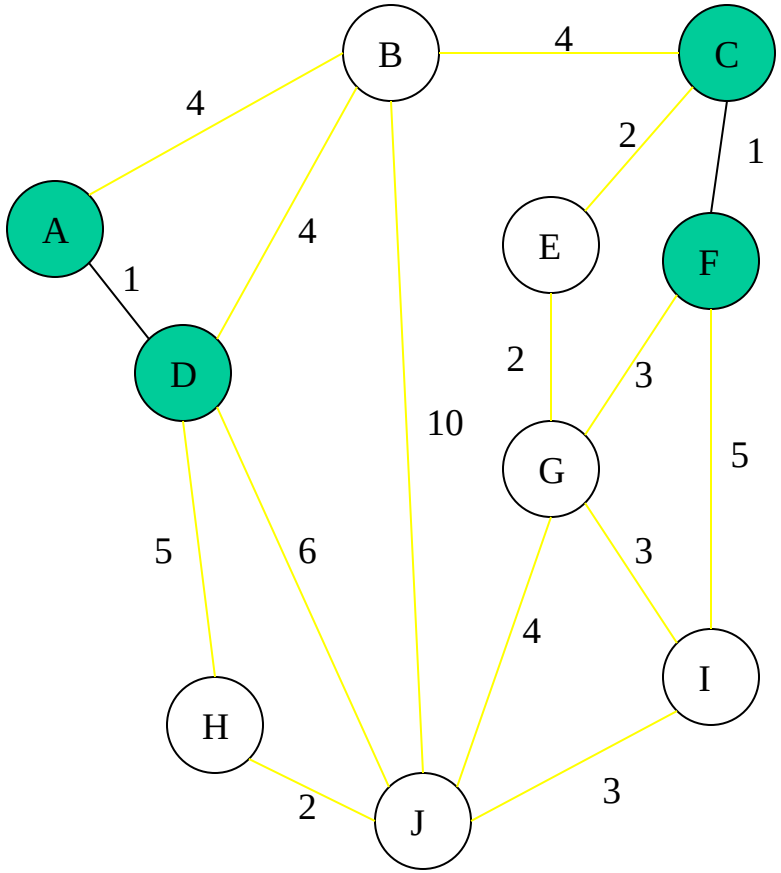


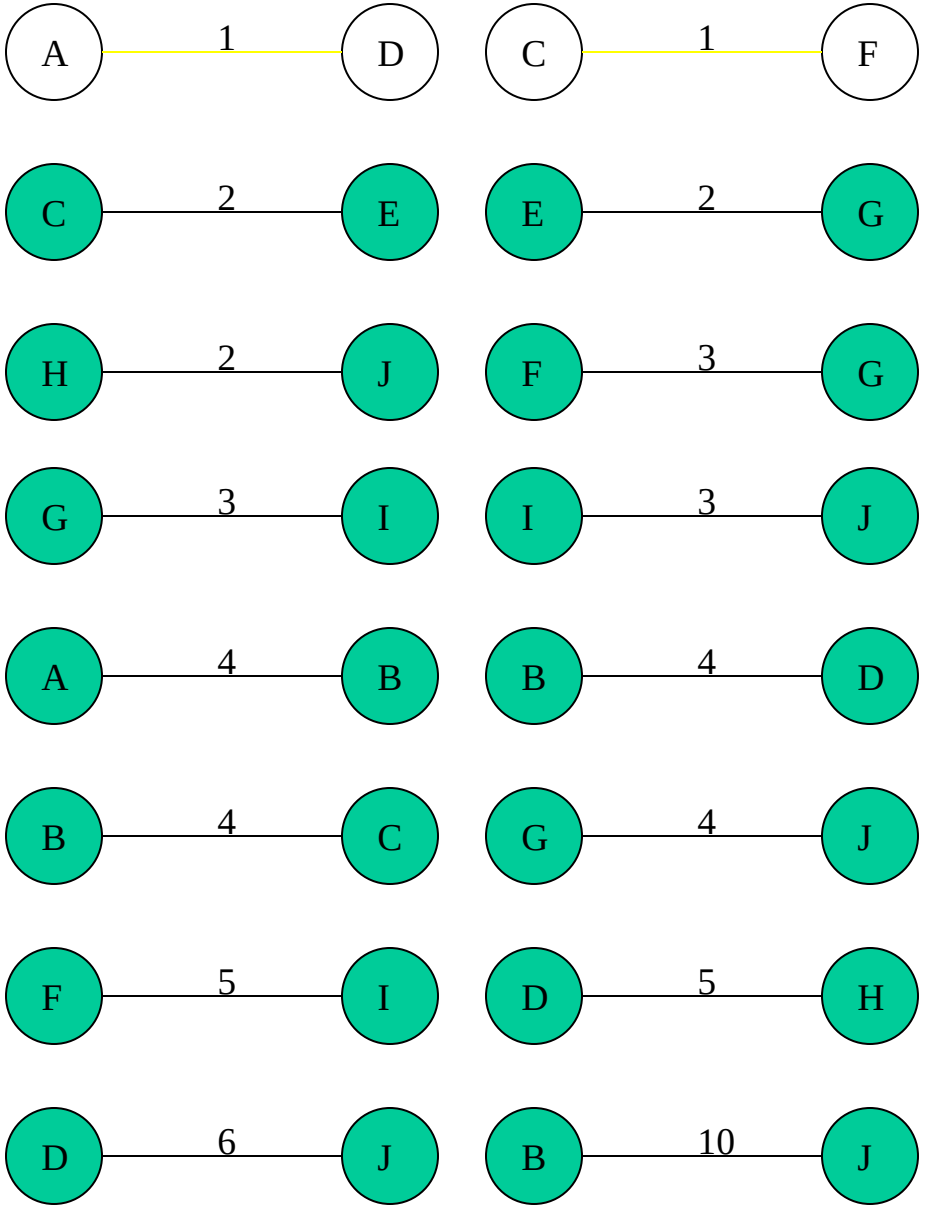
Add Edge


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

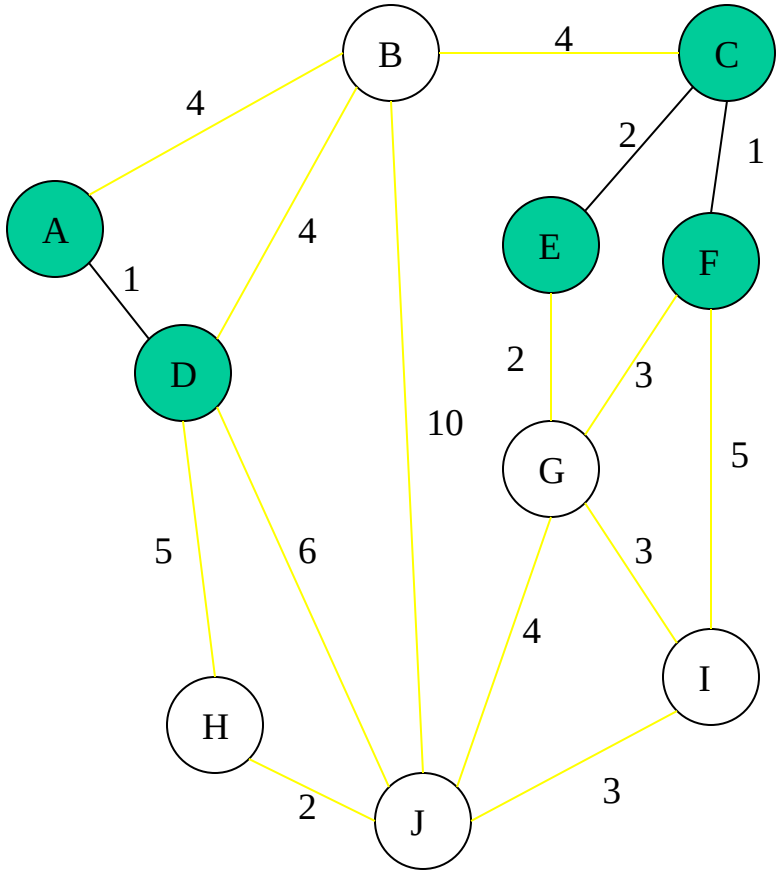

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge


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

Add Edge



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

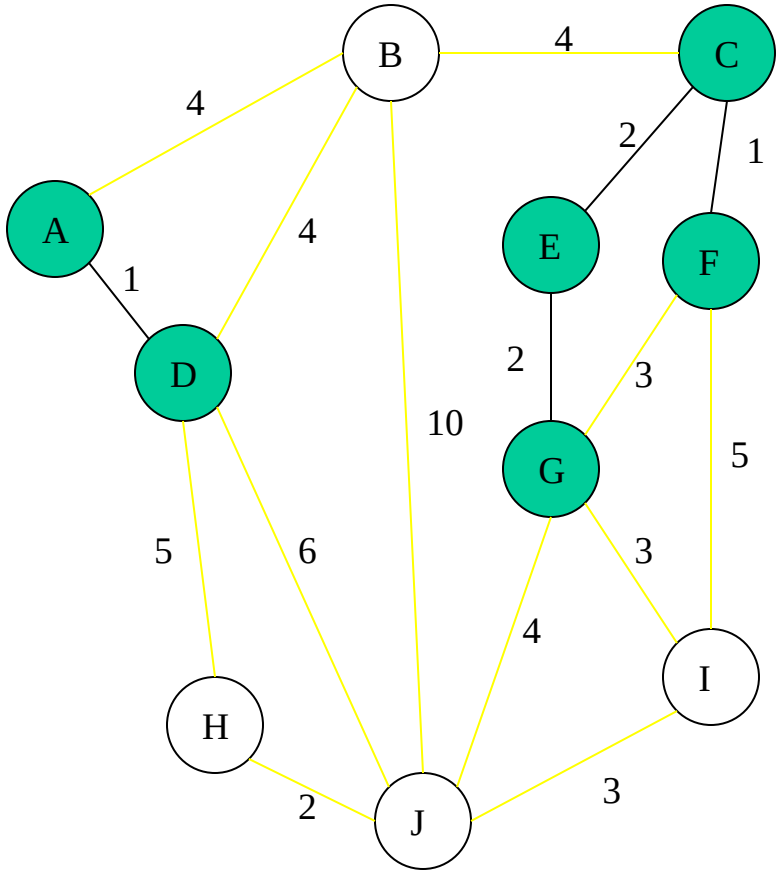
F 5 I

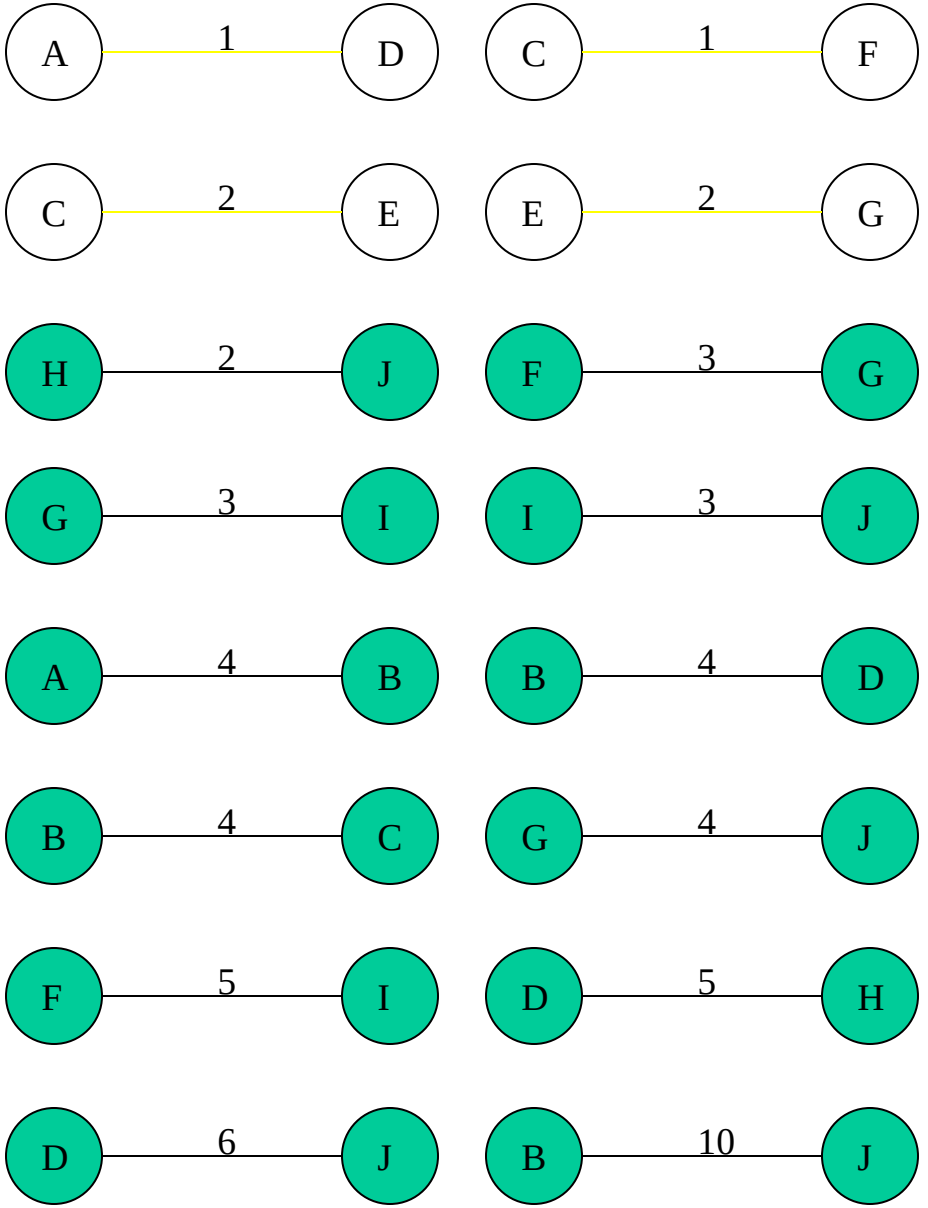
D 5 H

D 6 J

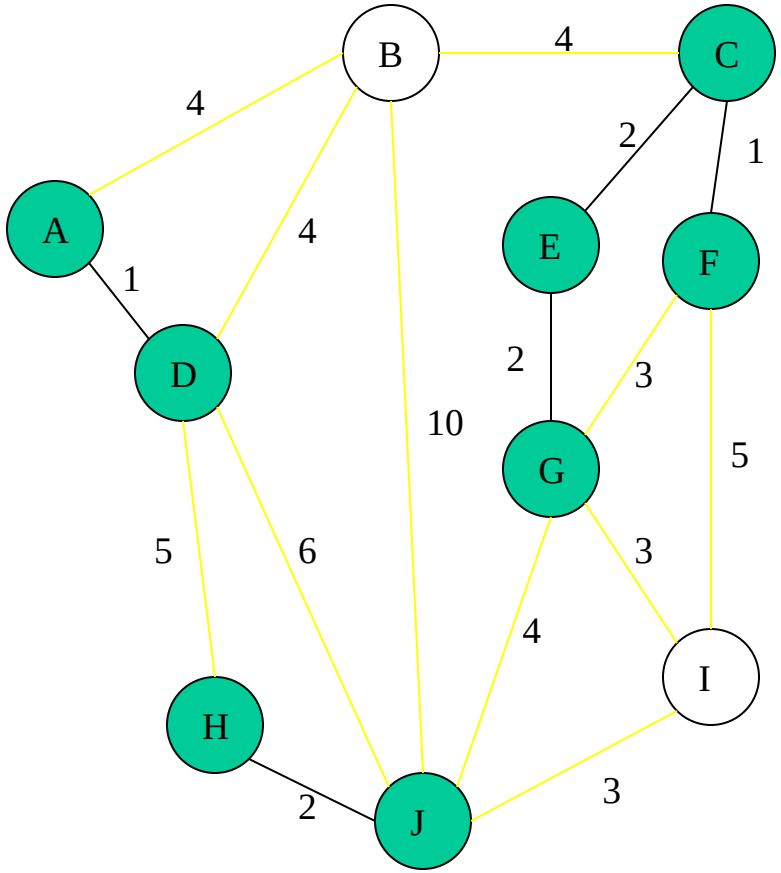
B 10 J

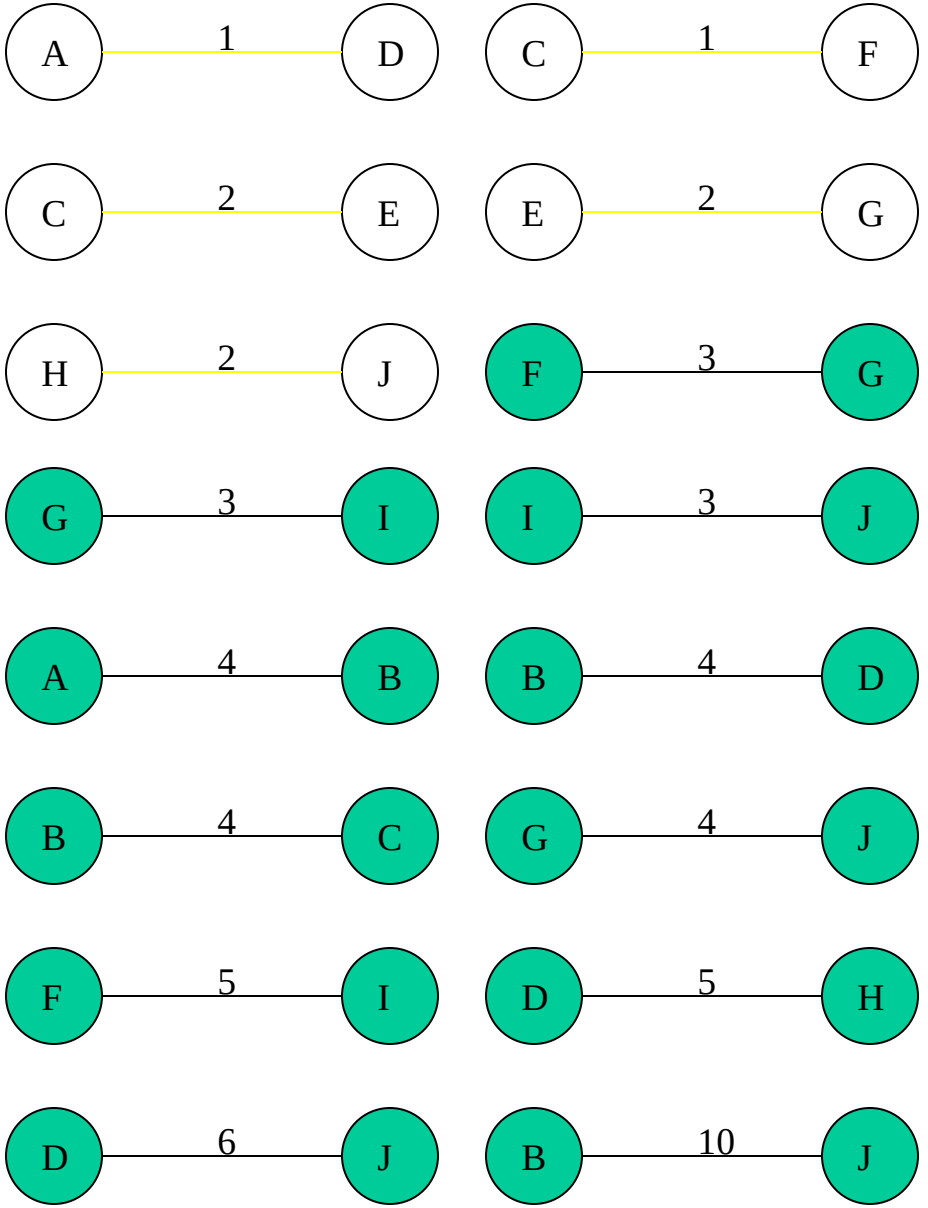
Add Edge


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

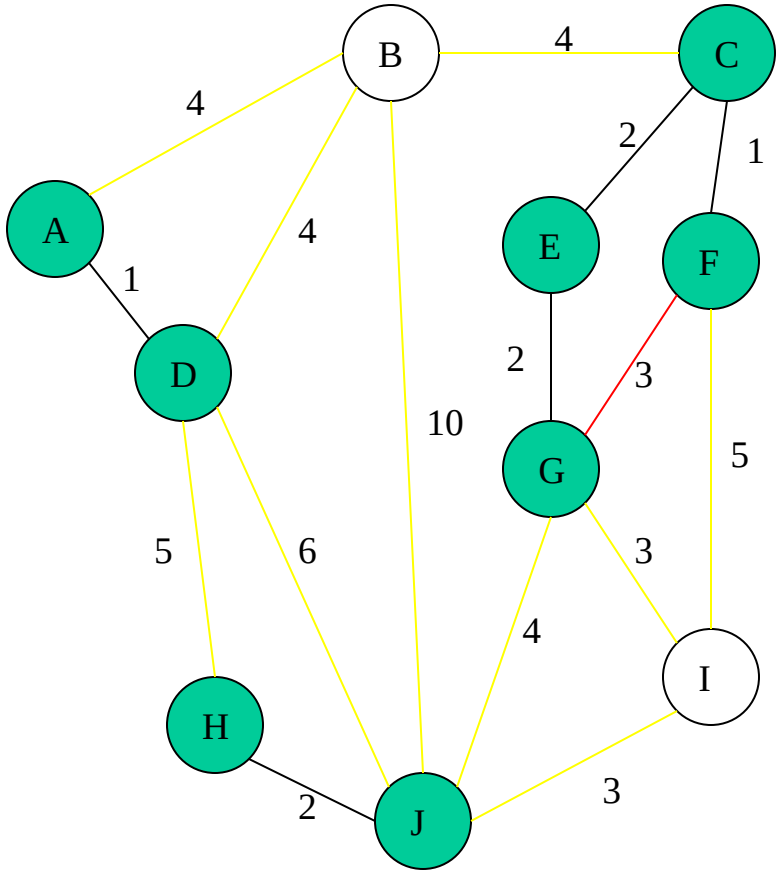
Add Edge

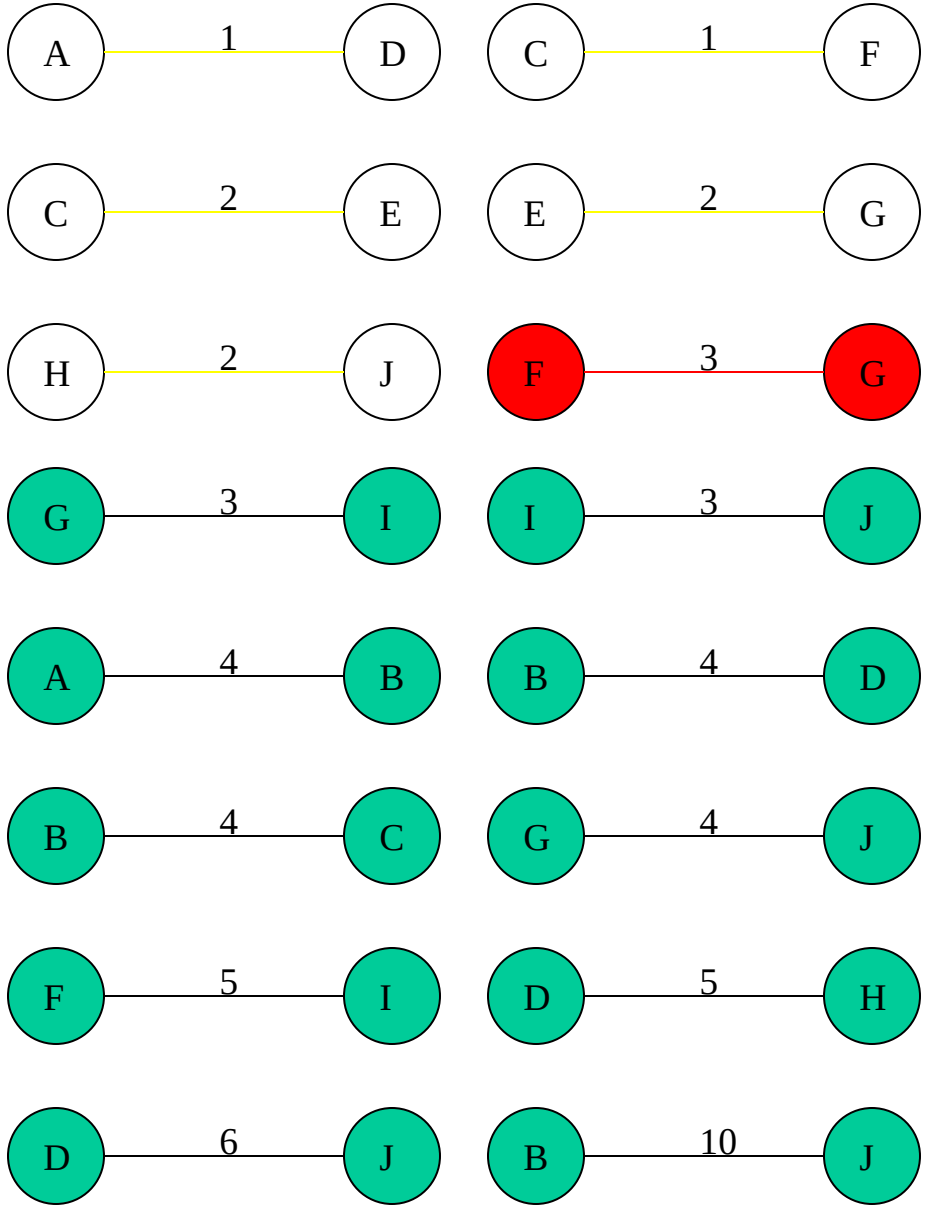

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3


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

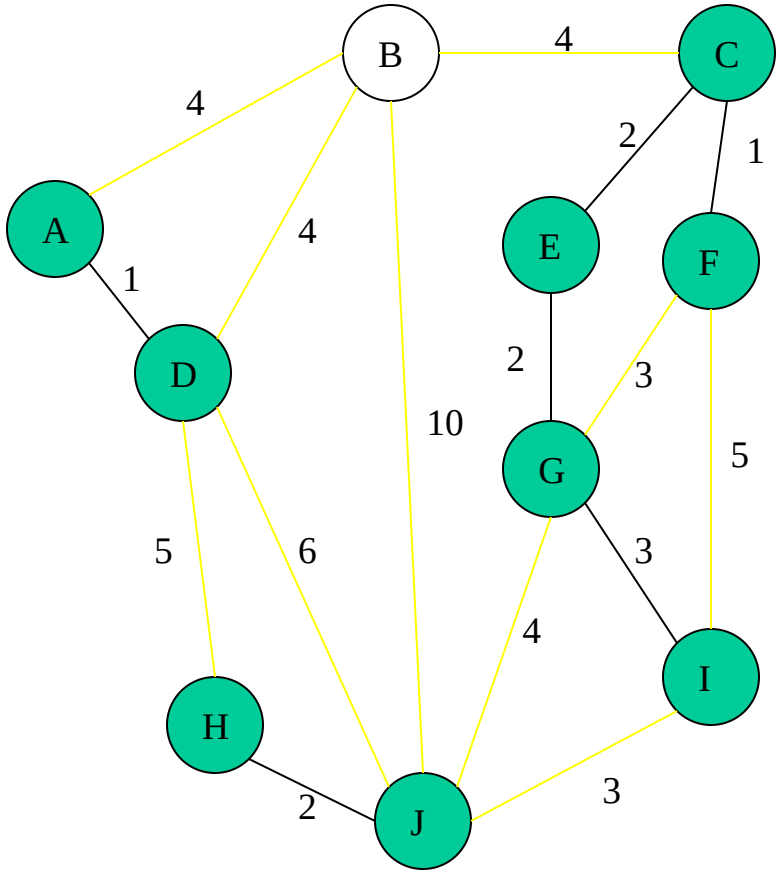
Cycle

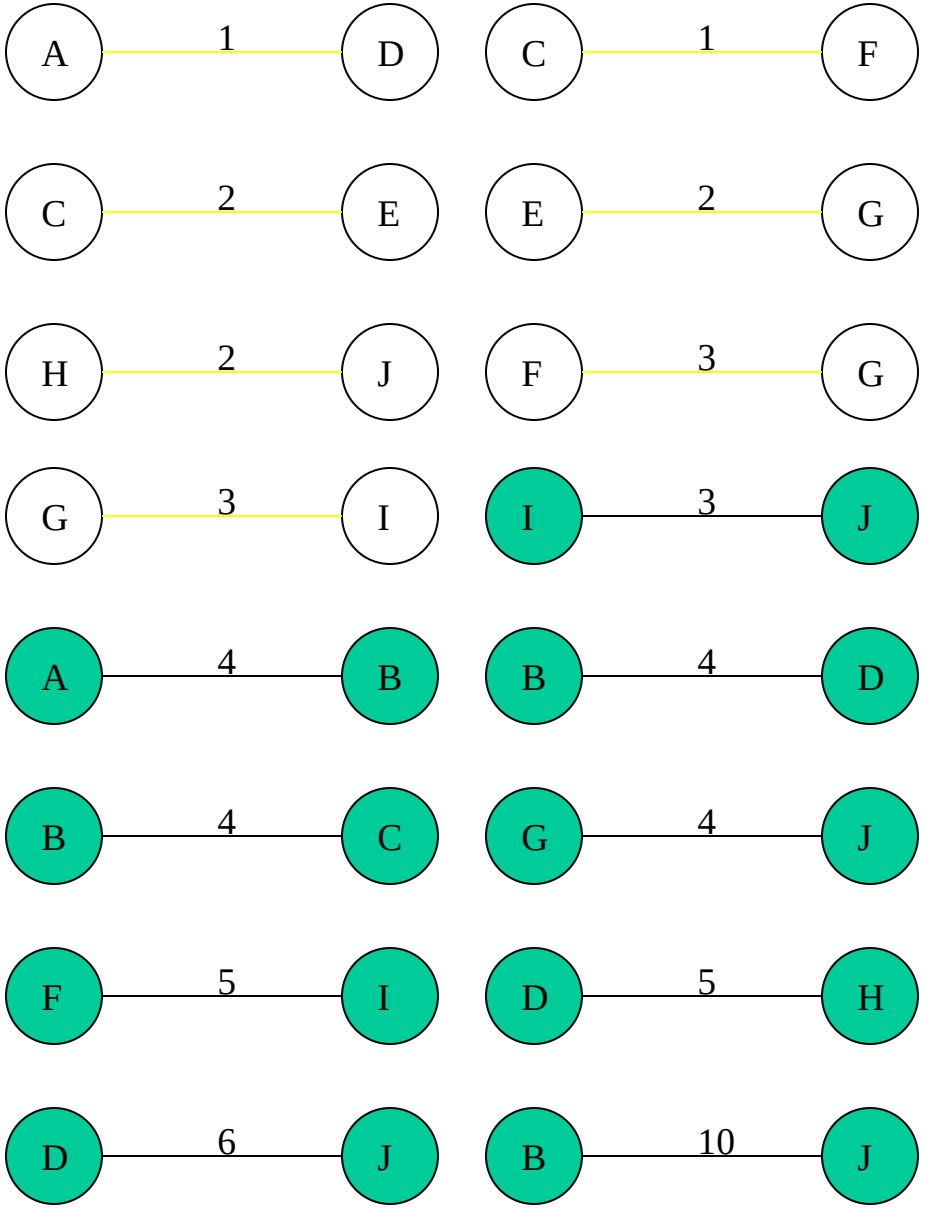
Don’t Add Edge



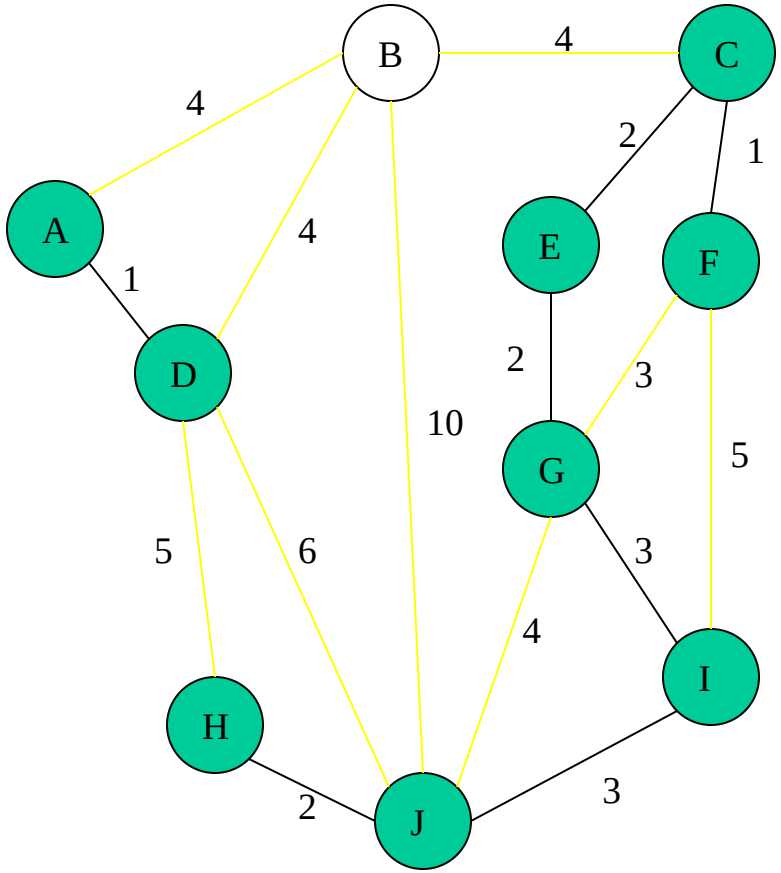


Add Edge


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


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

Add Edge



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

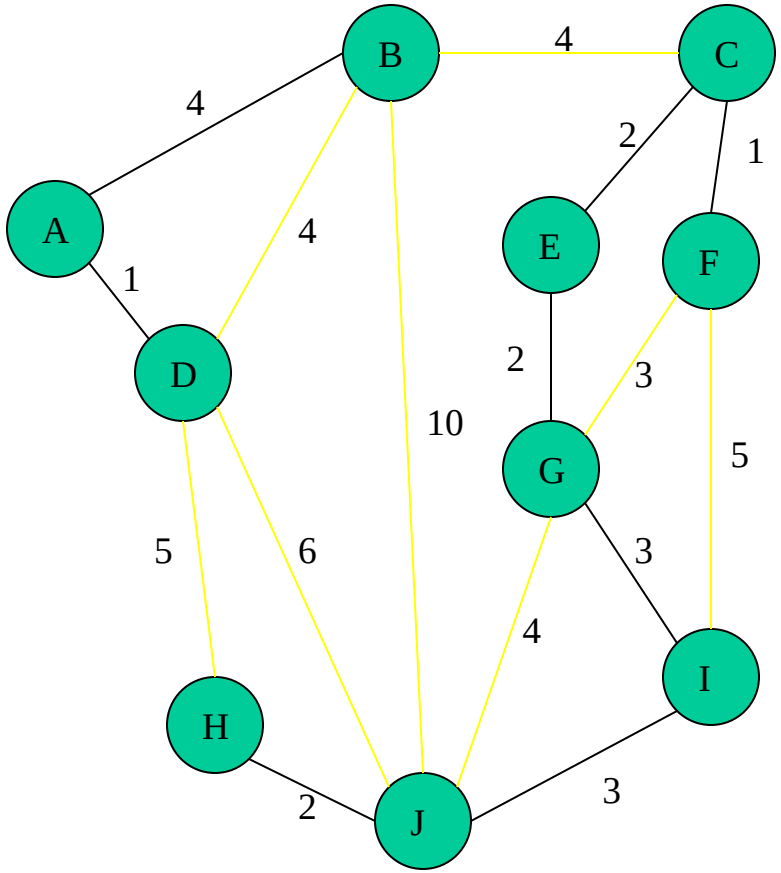
F 5 I

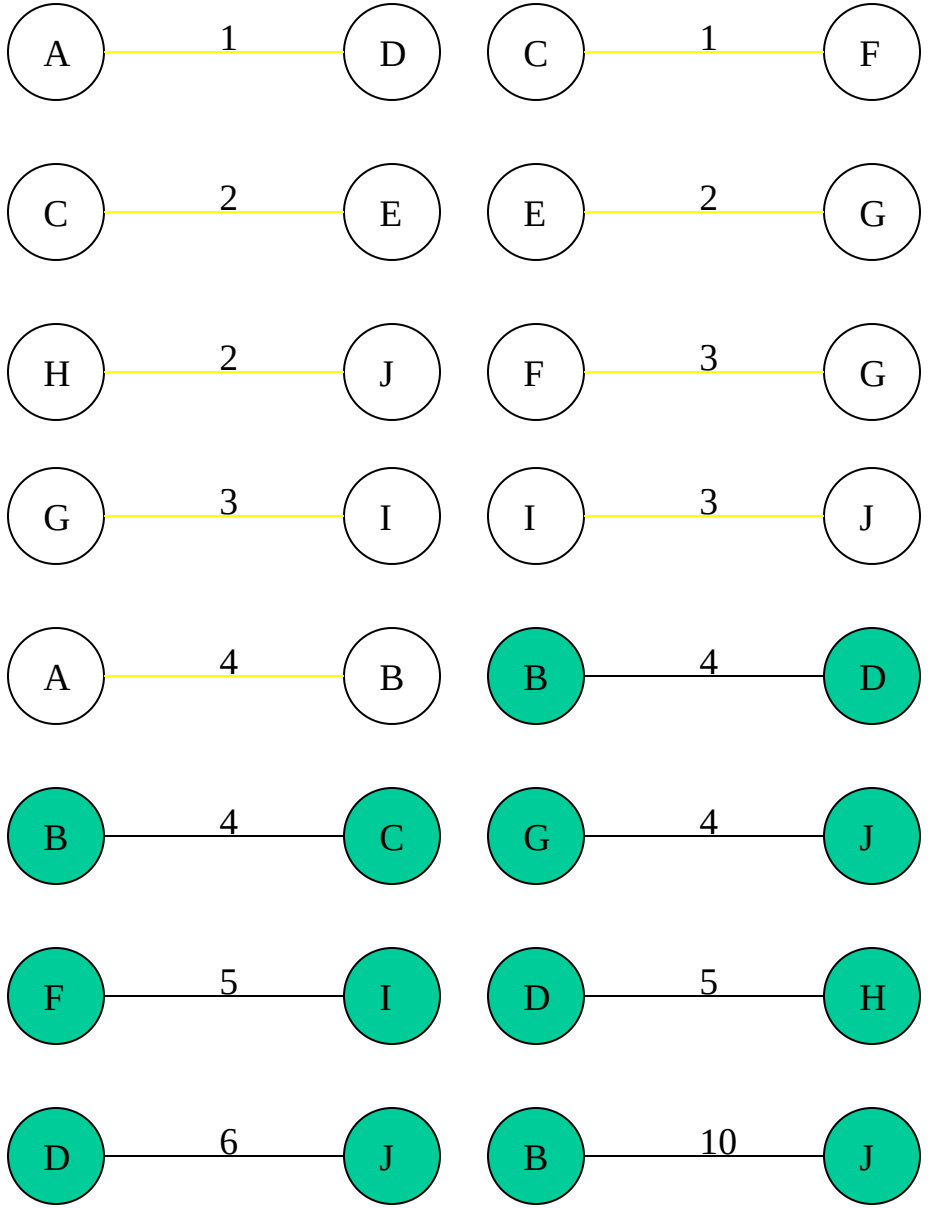
D 5 H

D 6 J

B 10 J

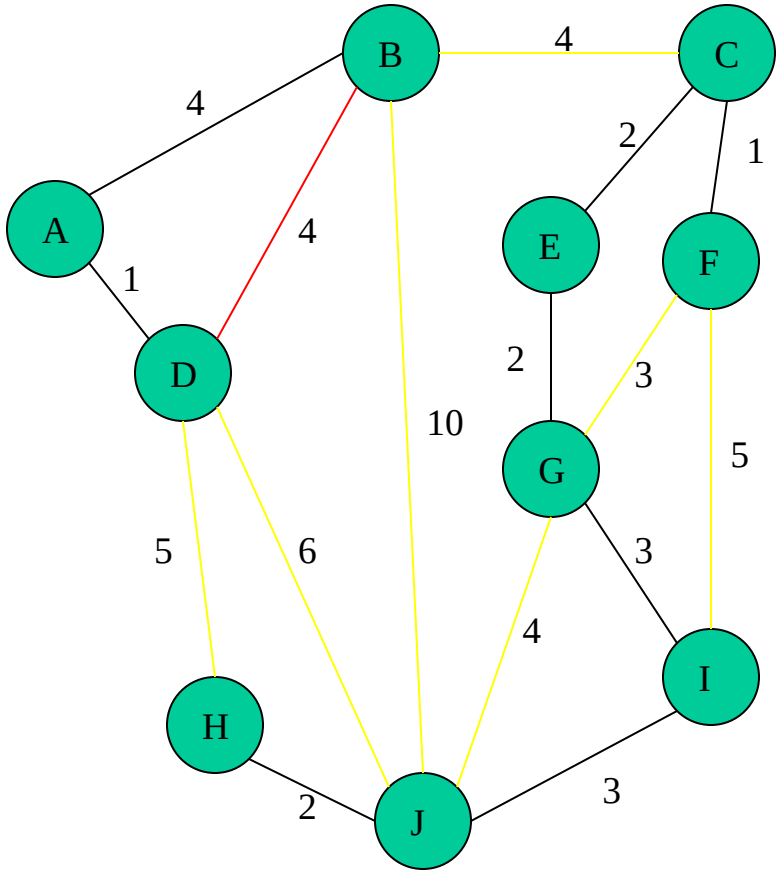
Add Edge

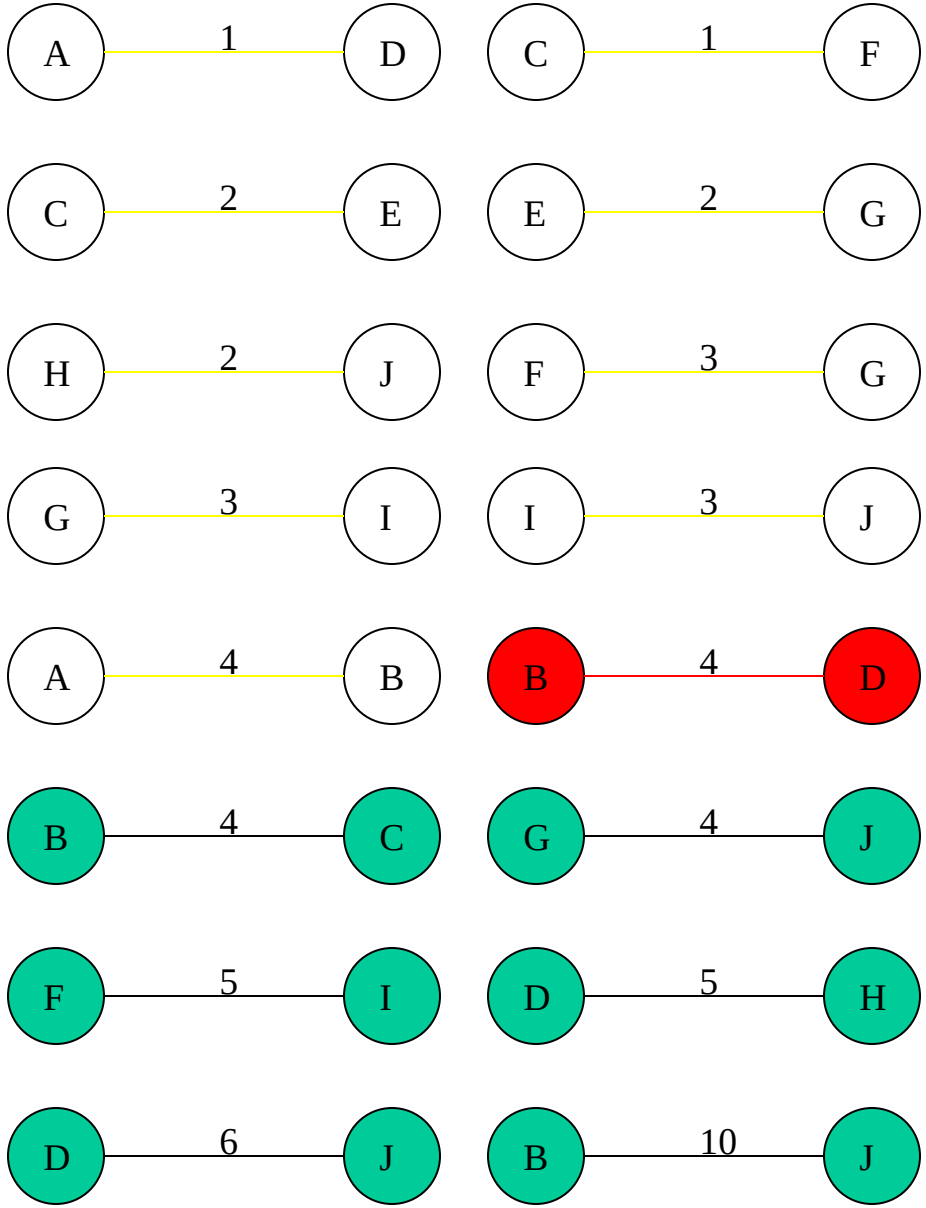

A
B
C
D
E
F
G
H
I
J
4
4
4
1
2
1
4
10
2
3
5
5
6
3
4
2
3


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

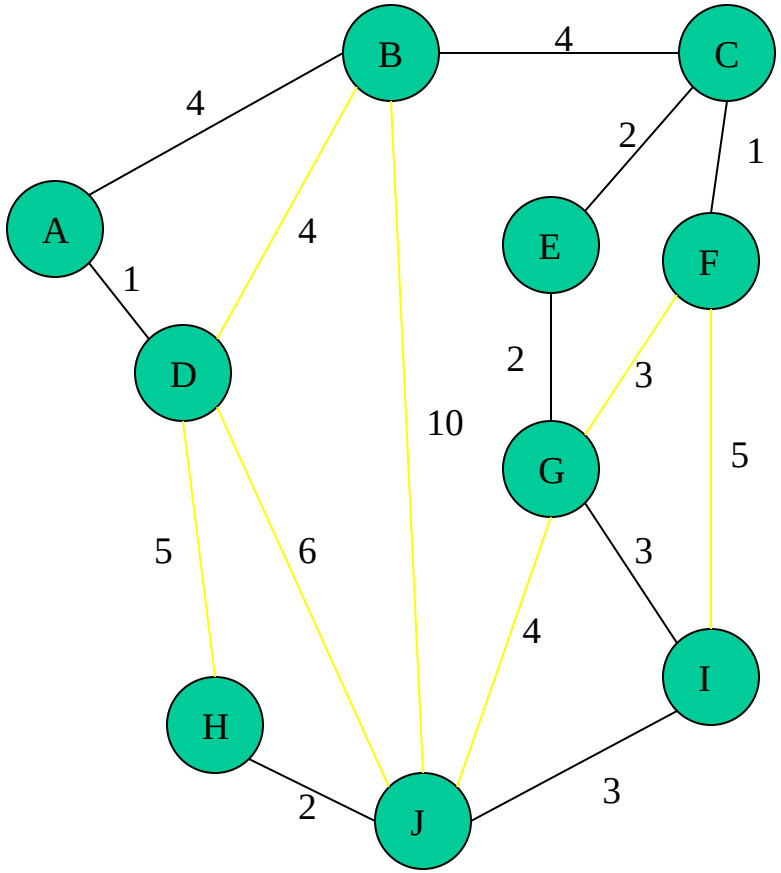
Cycle

Don’t Add Edge





Add Edge



A 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

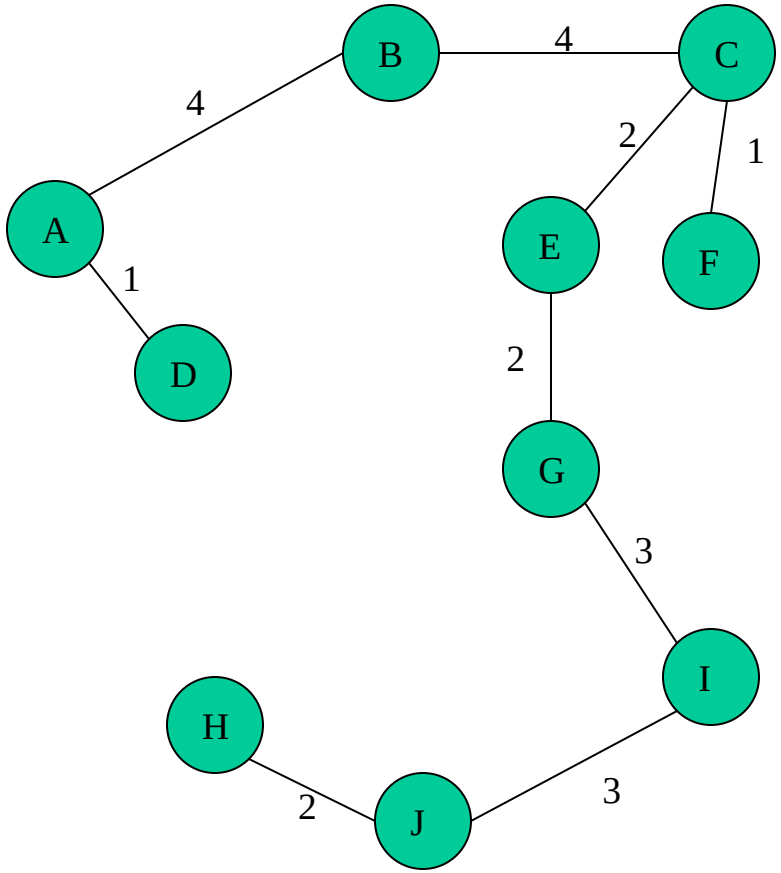
F 5 I

D 5 H

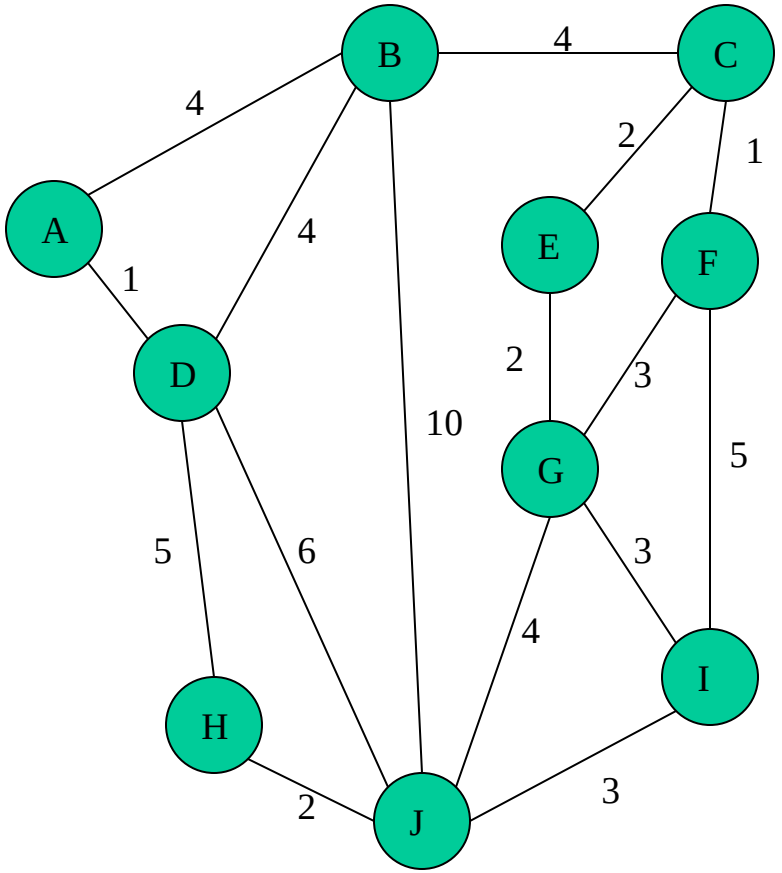
D 6 J

B 10 J

Minimum Spanning Tree



Complete Graph



Minimum Spanning Tree– Krushkal's Algorithm Analysis

- Running Time = O(m log n) (m = edges, n = nodes)
- Testing, if an edge creates a cycle, can be slow unless a complicated data structure, called a "union-find" structure, is used.
- It usually only has to check a small fraction of the edges, but in some cases (like if there was a vertex connected to the graph by only one edge and it was the longest edge) it would have to check all the edges.
- This algorithm works best, of course, if the number of edges is kept to a minimum.

Minimum Spanning Tree– Krushkal's Algorithm

```
Algorithm kruskal(E,cost,n,t)
//E is the set of edgesin G G has n vertices
//cost[u,v] is the cost of edge(u,v). T is the //set
   of edges in the minimum cost spanning tree
{
      for i:=1 to n do parent[i]:=-1;
      i:=0; minicost:=0.0;
      while(i<n-1) and (heap not empty) do
      {
            Delete a minimu cost edge (u,v) from the
               heap and reheapify using Adjust;
            J:=Find(u); k:=Find(v);
            If(j!=k) then
      {
            i:=i+1;
            t[i,1]:=u;t[i,2]:=v;
            minicost:=minicost+cost[u,v];
            Union(j,k);
      }//if ends
      }// while ends
      if(i!=n-1)then
      write("No Spanning Tree);
      else
      return minicost;
}
```

Minimum Spanning Tree– Prim's Algorithm

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

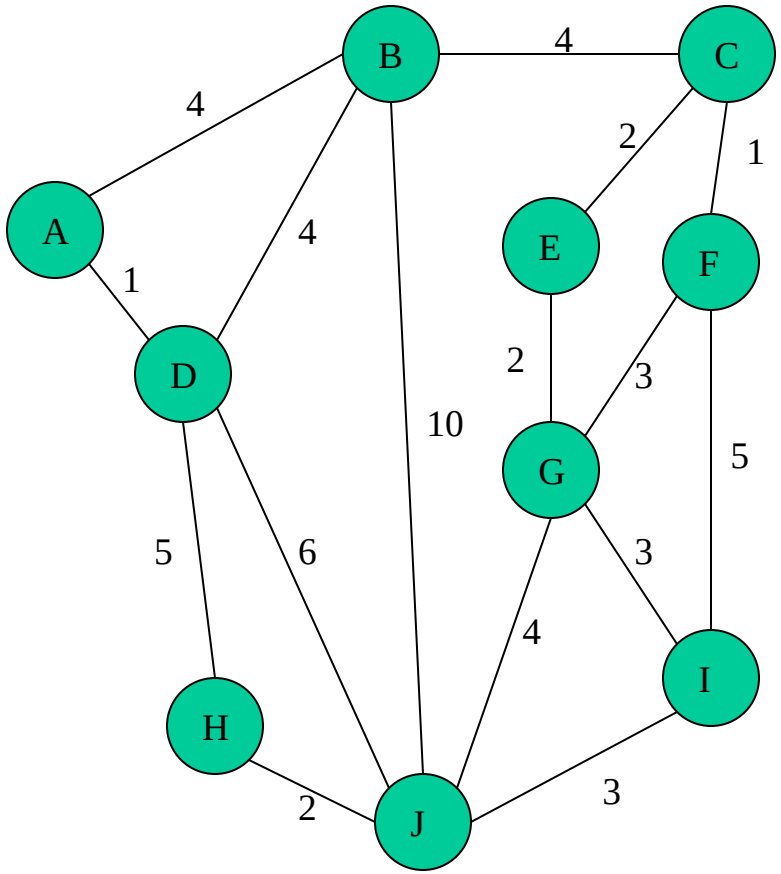
Minimum Spanning Tree– Prim's Algorithm

The steps are:

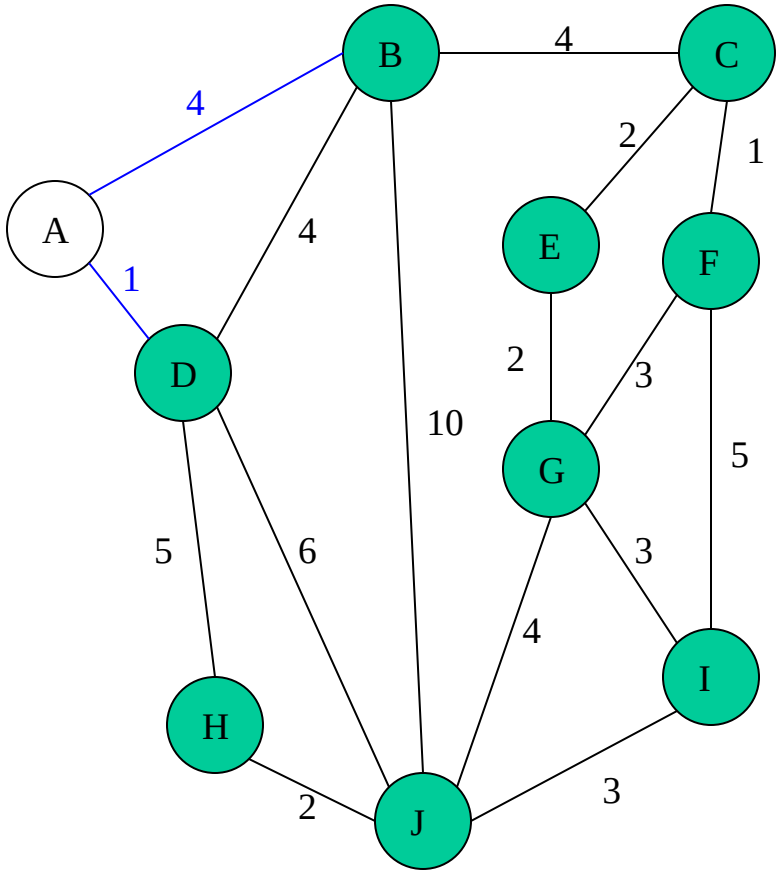
1. The new graph is constructed - with one node from the old graph.
2. While new graph has fewer than n nodes,
    1. Find the node from the old graph with the smallest connecting edge to the new graph,
    2. Add it to the new graph

- Every step will have joined one node, so that at the end, we will have one graph with all the nodes, and it will be a minimum spanning tree of the original graph.

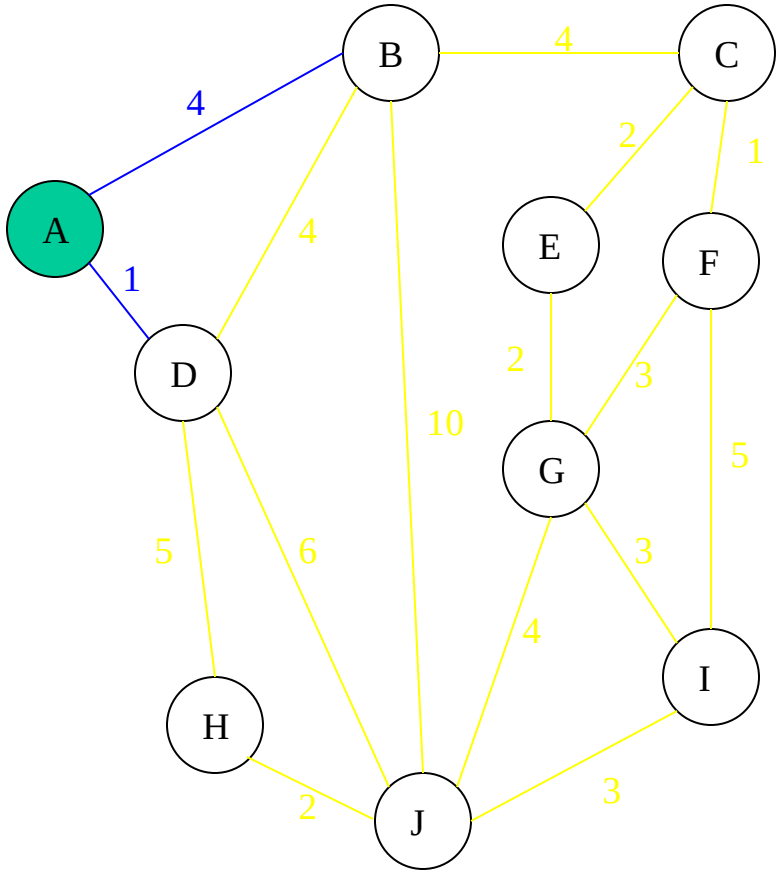
Complete Graph



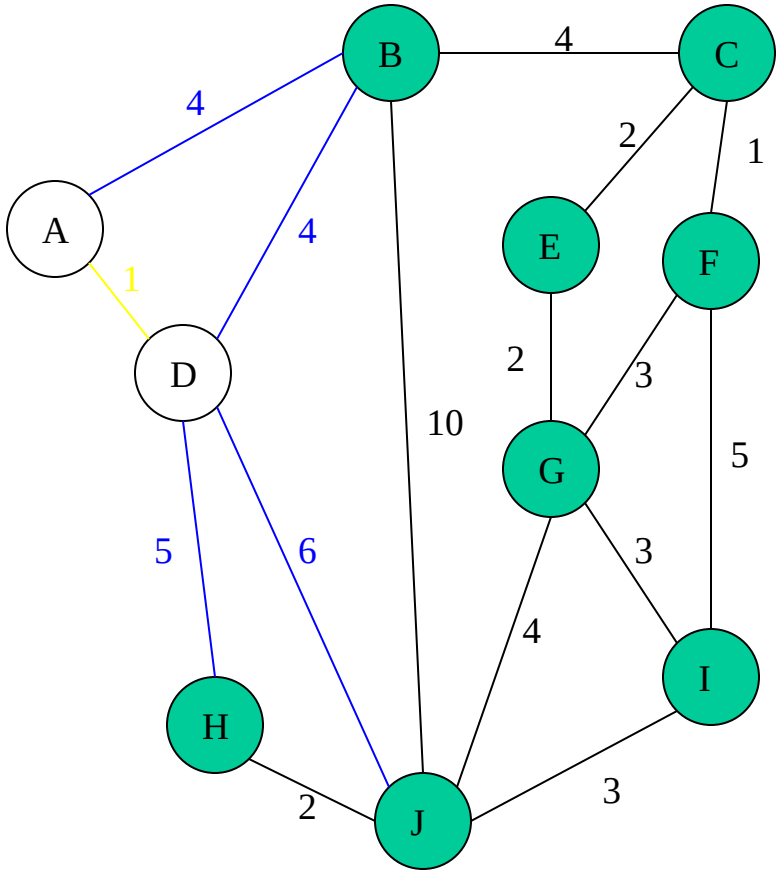
Old Graph



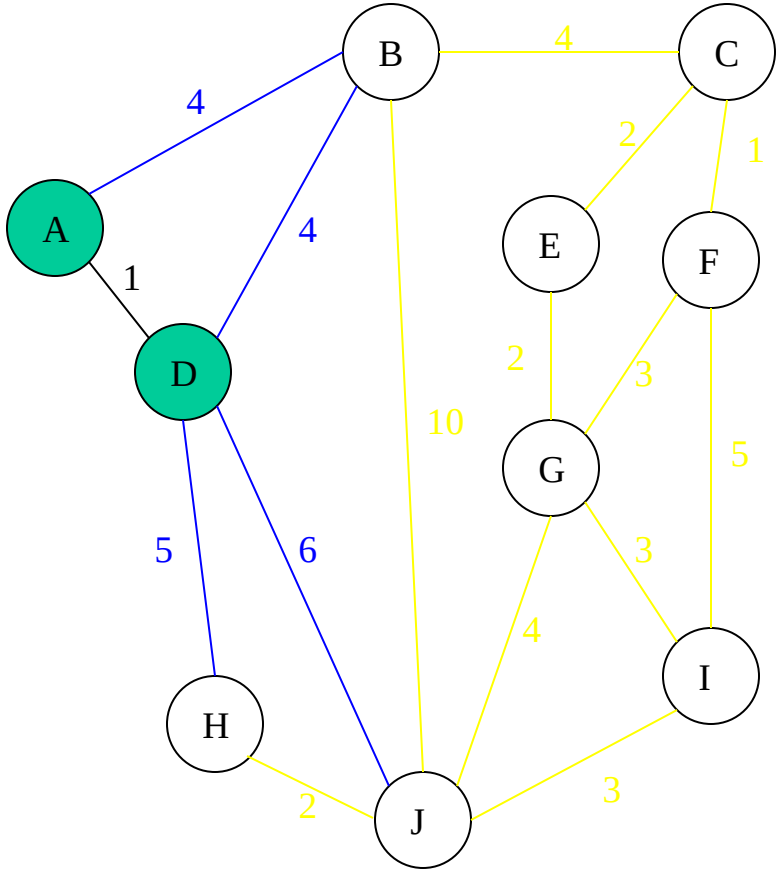
New Graph



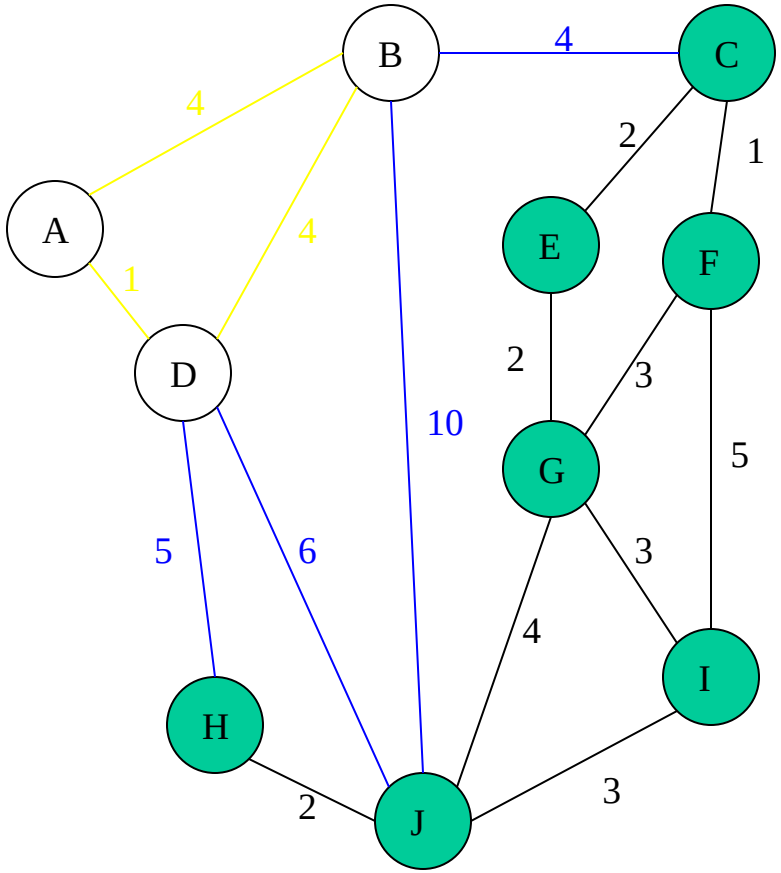
Old Graph



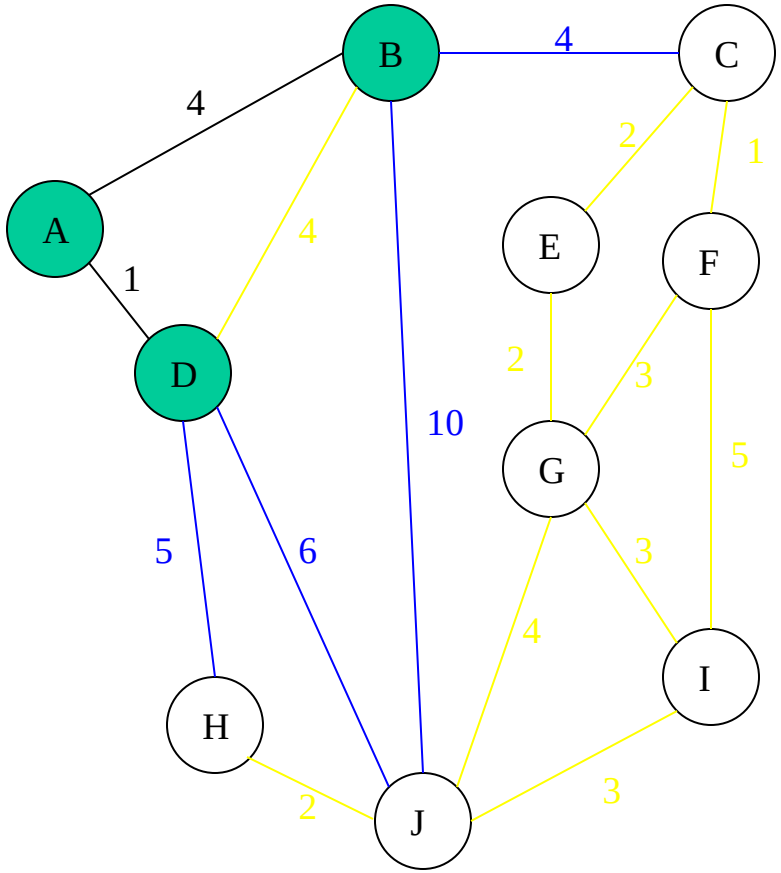
New Graph



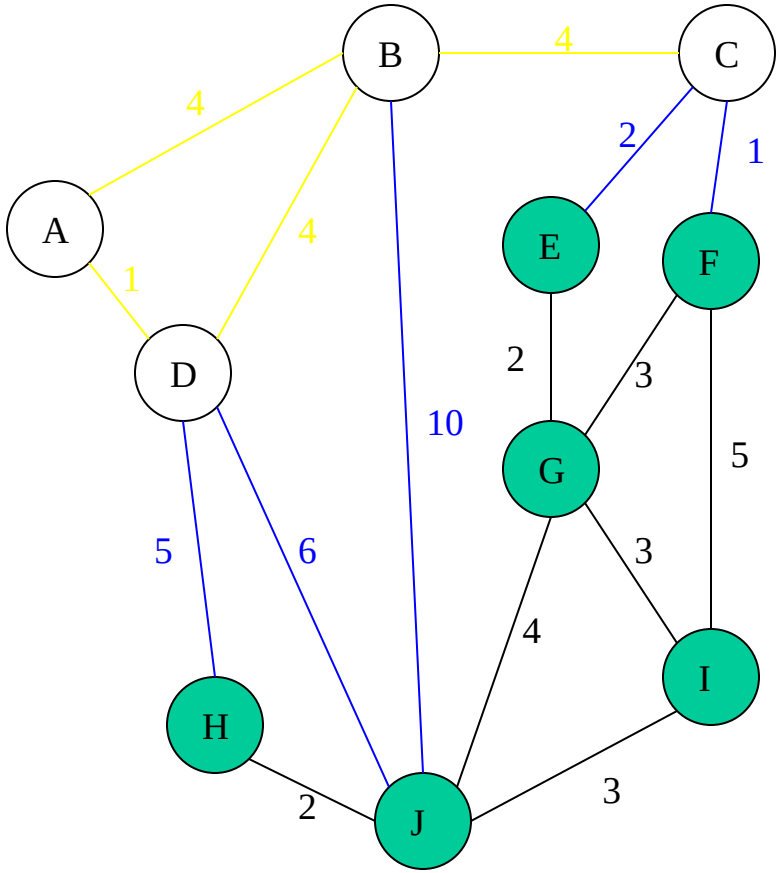
Old Graph



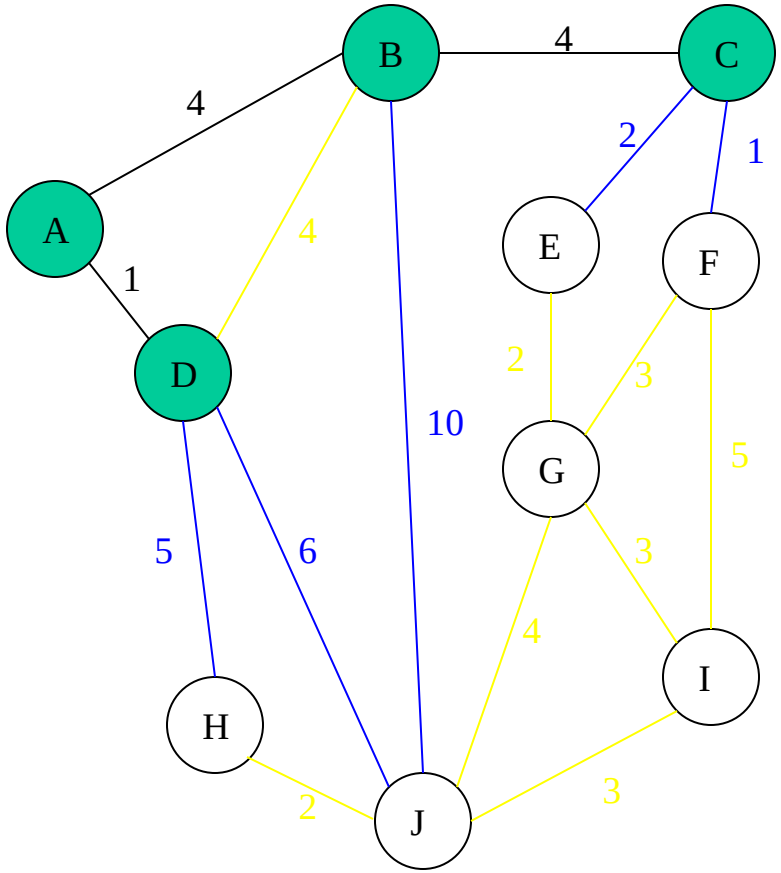
New Graph



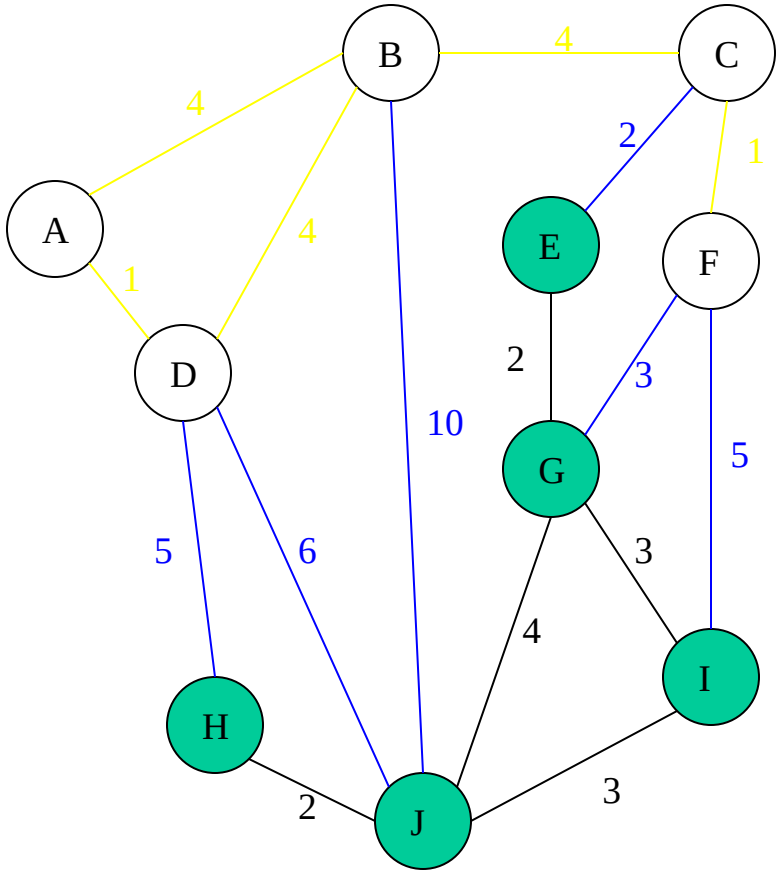
Old Graph



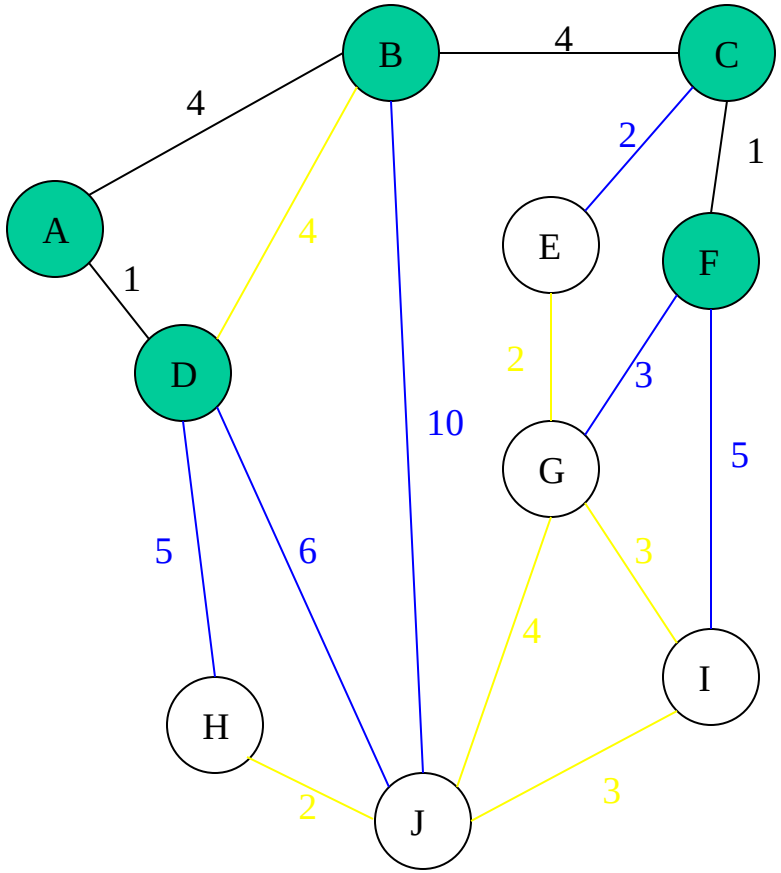
New Graph



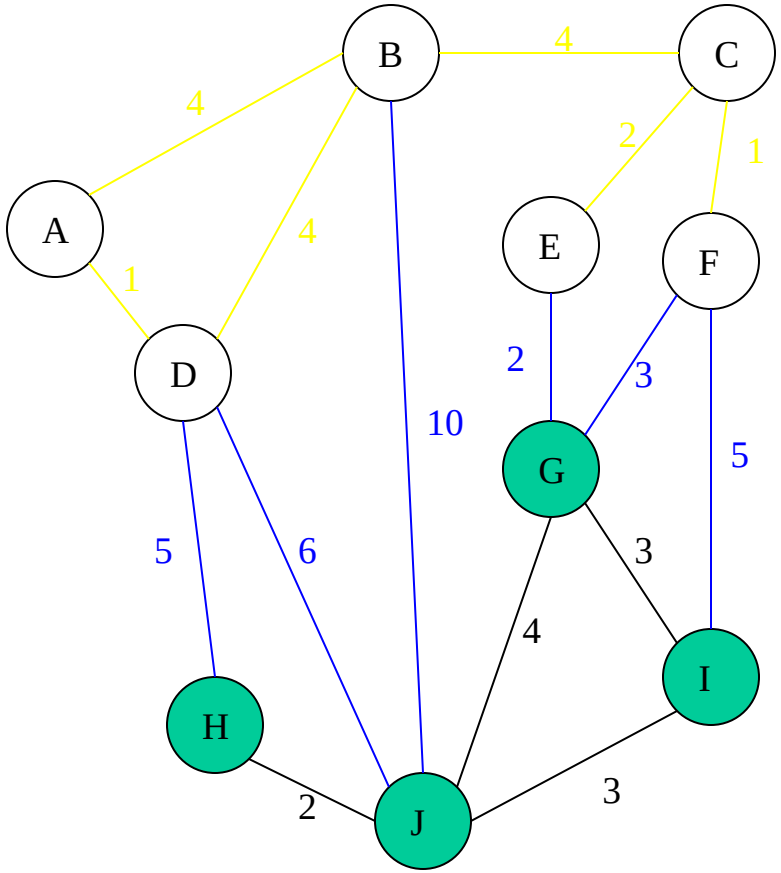
Old Graph



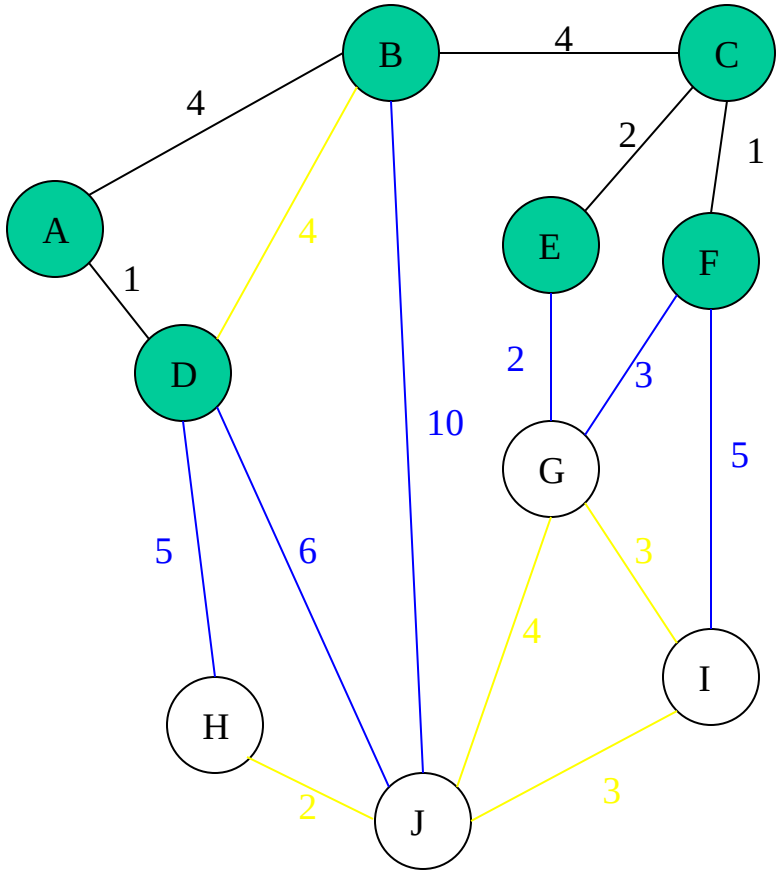
New Graph



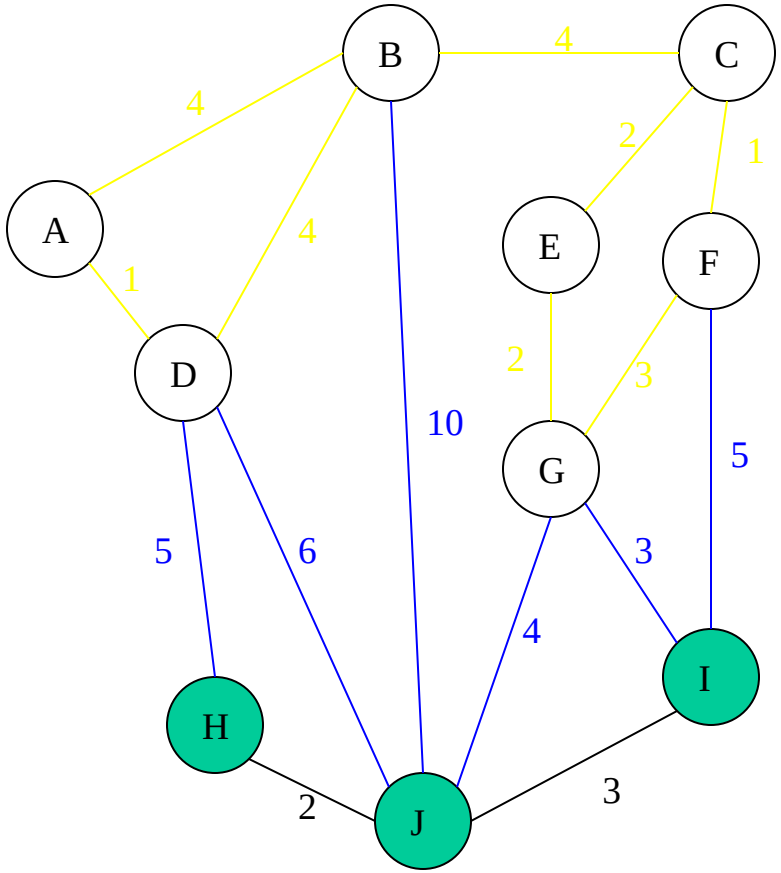
Old Graph



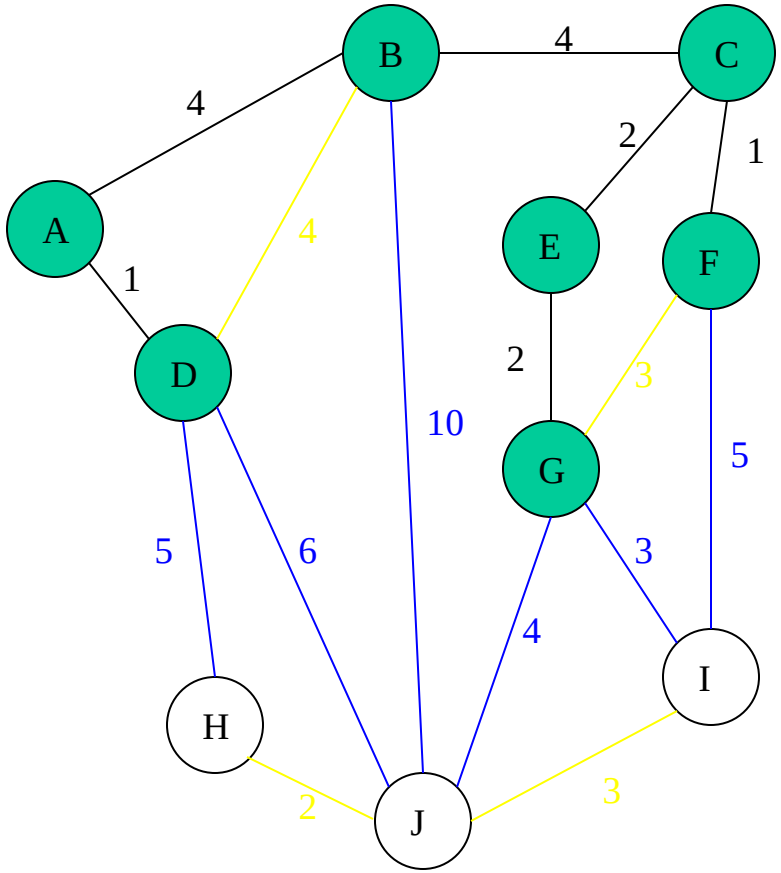
New Graph



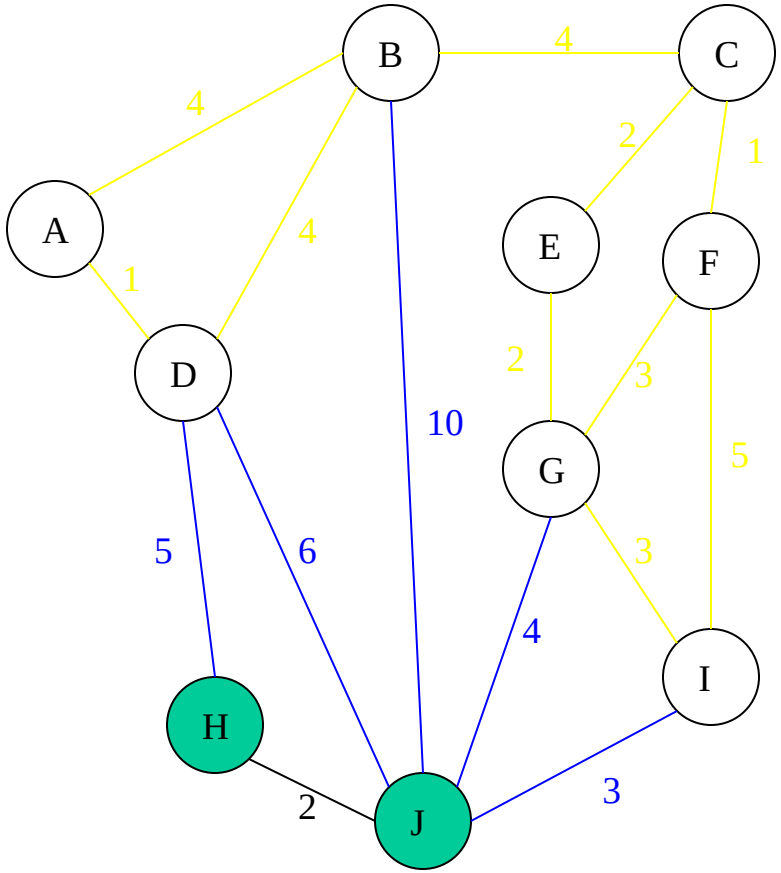
Old Graph



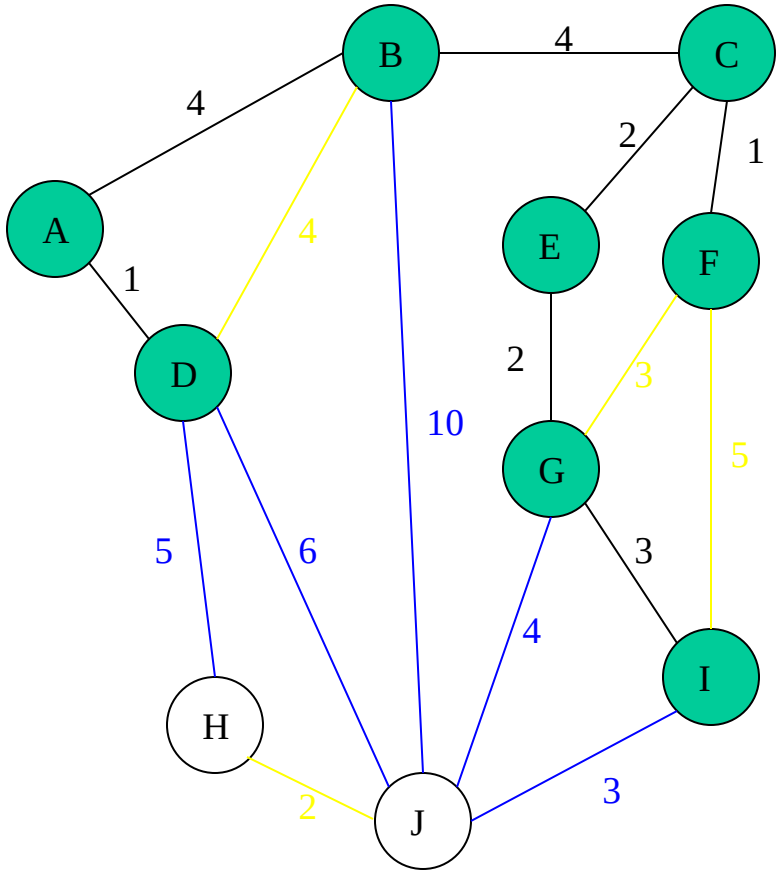
New Graph



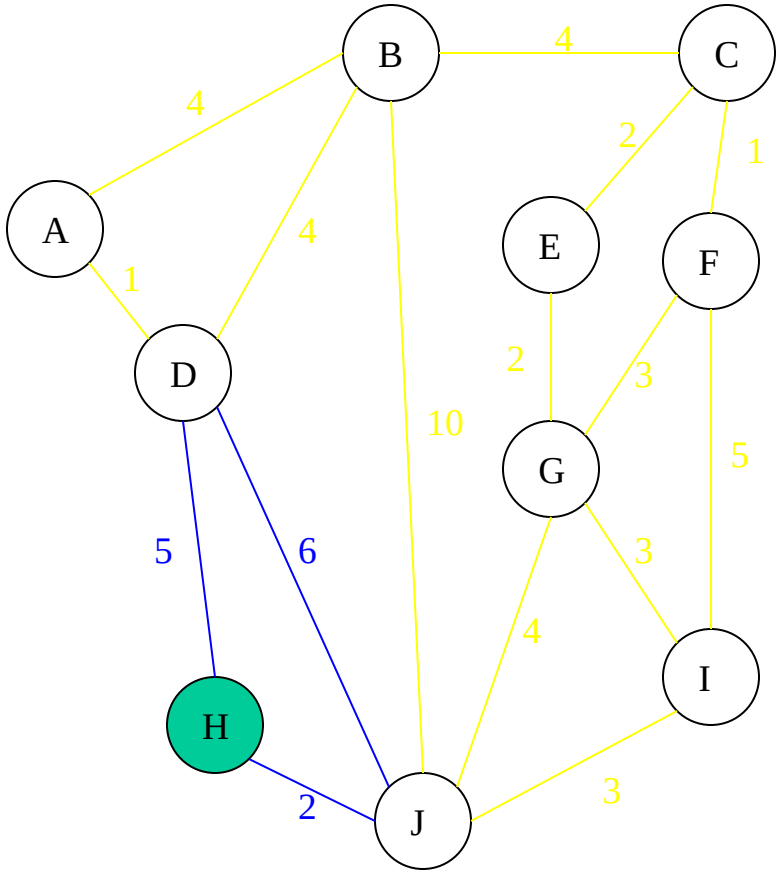
Old Graph



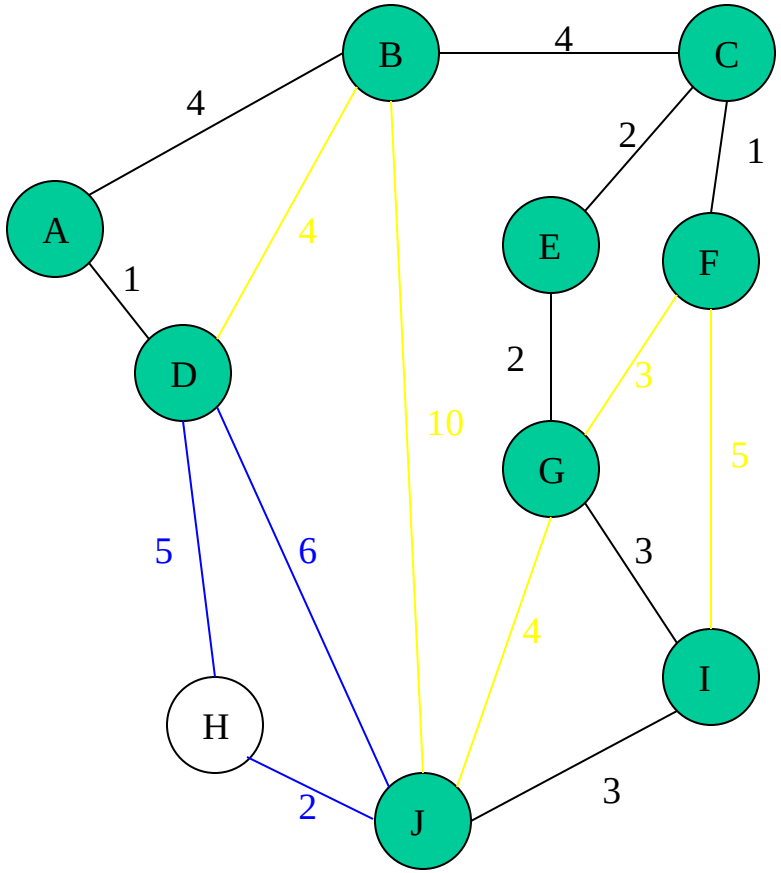
New Graph



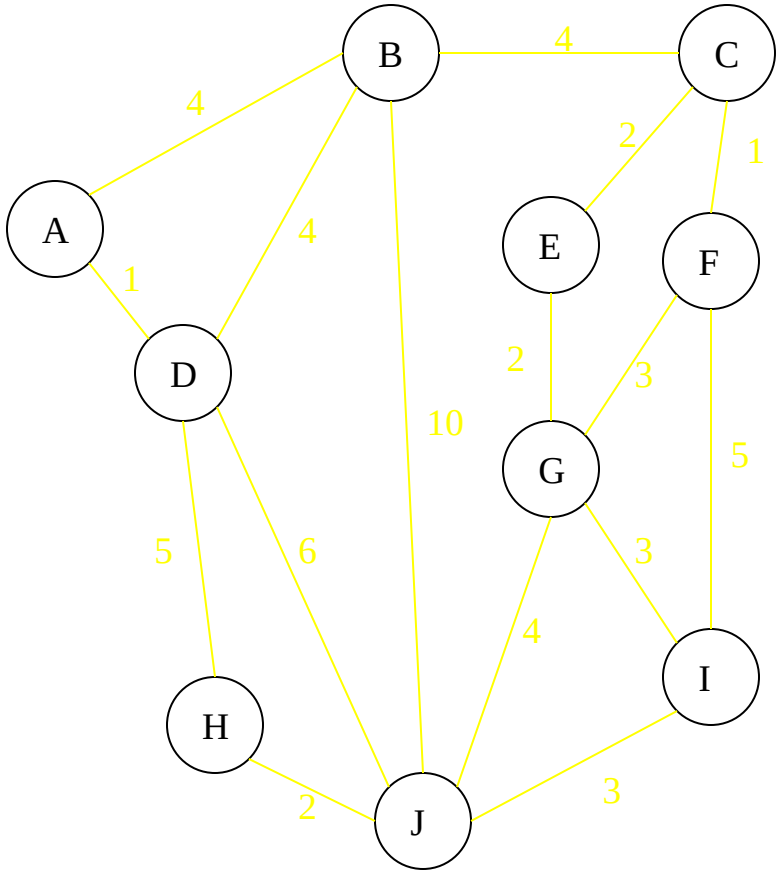
Old Graph



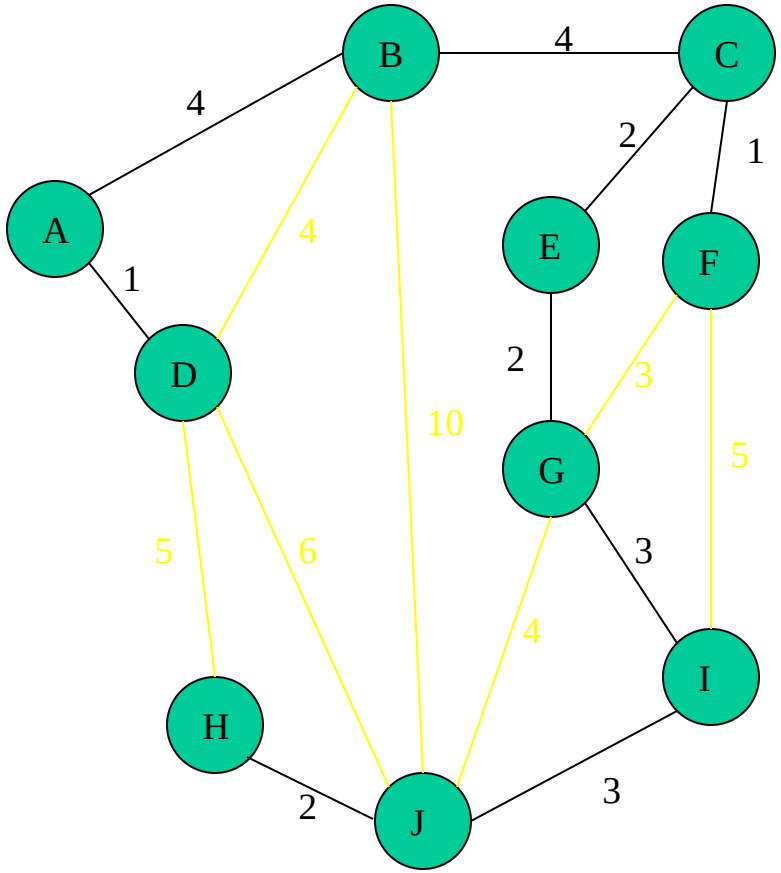
New Graph



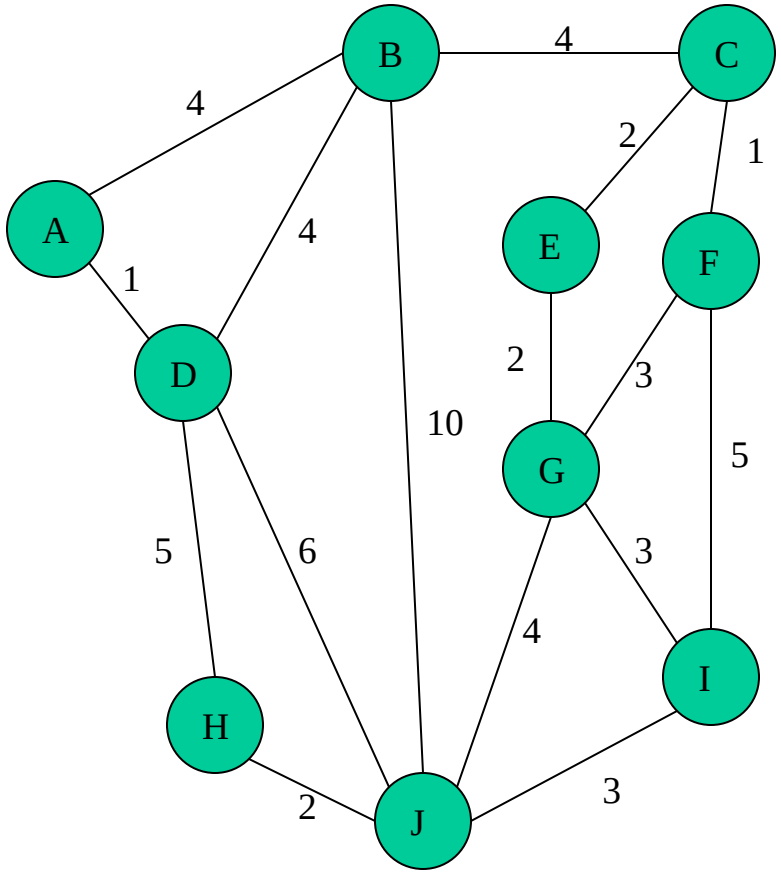
Old Graph



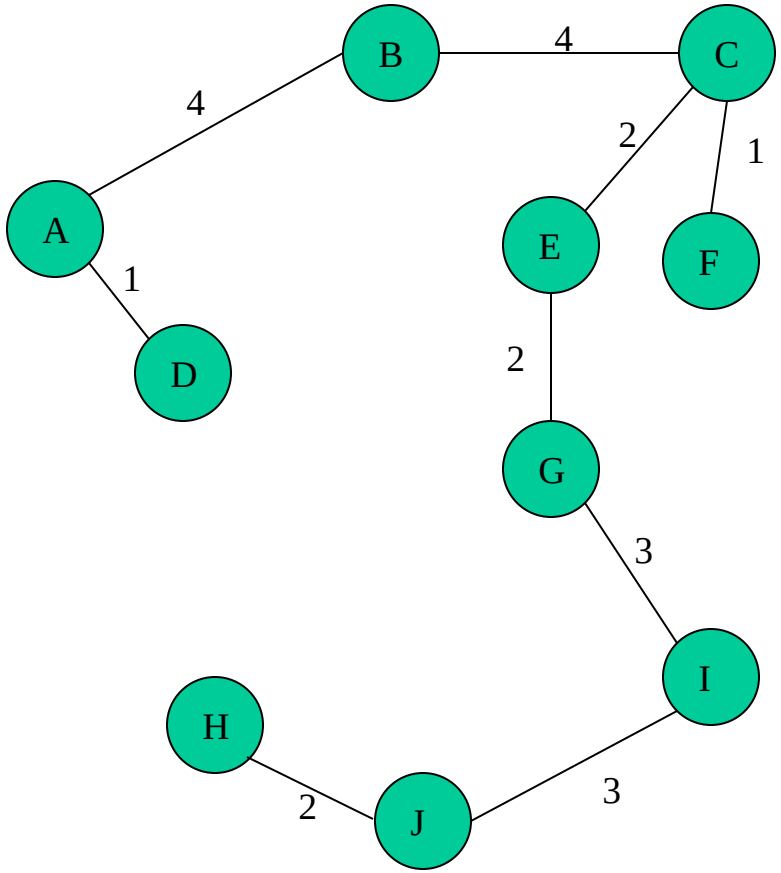
New Graph



Complete Graph



Minimum Spanning Tree



Minimum Spanning Tree– Krushkal's Algorithm

```
Algorithm prim(E,cost,n,t)
{
Let (k,l) be an edge of minimum cost in E;
mincost:=cost[k,l];
t[1,1]:=k;t[1,2]:=l;
for i:=1 to n do
if(cost[i,l]<cost[i,k]) then near[i]:=l;
else near[i]:=k;
near[k]:=near[l]:=0;
for i:=2 to n-1 do
```

```
Let j be an index such that near[j]!=0 and
cost[j,near[j]] is minimum;
t[i,1]:j;t[i,2]:=near[j];

mincost:=mincost+cost[j,near[j]];
near[j]:=0;
for k:= 1 to n do
rf((near[k]!=0) and (cost[k,near[k]]>cost[k,j]))
rhen near[k]:=j;
}
return mincost;
 }
```

Minimum Spanning Tree– Krushkal's Algorithm Analysis

Running Time = O(m + n log n) (m = edges, n = nodes)

If a heap is not used, the run time will be O(n^2) instead of O(m + n log n). However, using a heap complicates the code since you're complicating the data structure. A Fibonacci heap is the best kind of heap to use, but again, it complicates the code.

Unlike Kruskal's, it doesn't need to see all of the graph at once. It can deal with it one piece at a time. It also doesn't need to worry if adding an edge will create a cycle since this algorithm deals primarily with the nodes, and not the edges.

For this algorithm the number of nodes needs to be kept to a minimum in addition to the number of edges. For small graphs, the edges matter more, while for large graphs the number of nodes matters more.

Minimum Spanning Tree– Prim's Algorithm Analysis

1. **Network design.**
   - *i. telephone,*
   - *ii. electrical,*
   - *iii. hydraulic,*
   - *iv. TV cable,*
   - *v. computer,*
   - *vi. road*
2. **Approximation algorithms for NP-hard problems.**
   - **i. Traveling salesperson problem**
3. **Civil Network Planning**
4. **Computer Network Routing Protocol**
5. **Cluster Analysis**

THANK YOU