Minimum Spanning Tree

Instructor: Krishna Venkatasubramanian

CSC 212

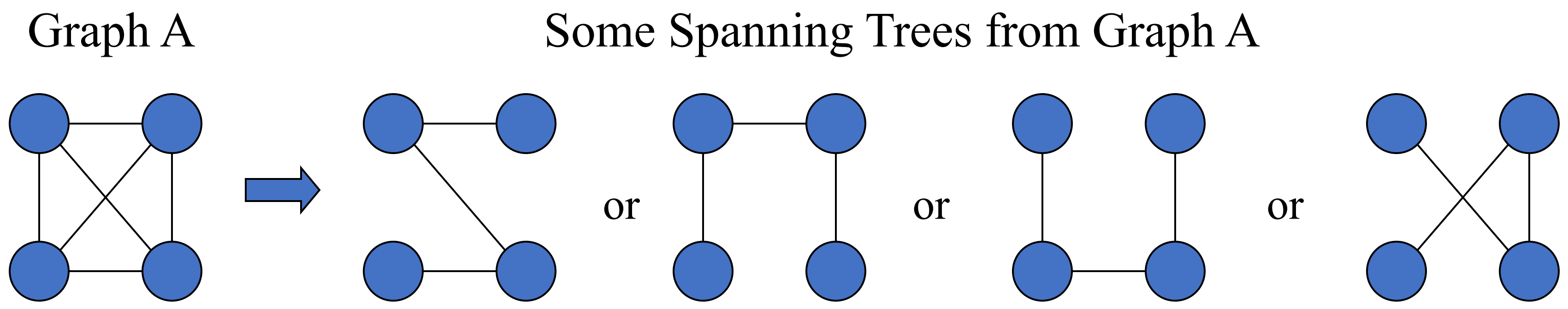
Based on slides from Prof. Carolina Ruiz at WPI

Announcement

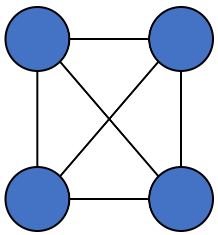
- Project 2 due at midnight
- Project 3 out
  - Due December 6th (last day of class)
- All those who need accommodations for the final exam, please email me.

Spanning Trees

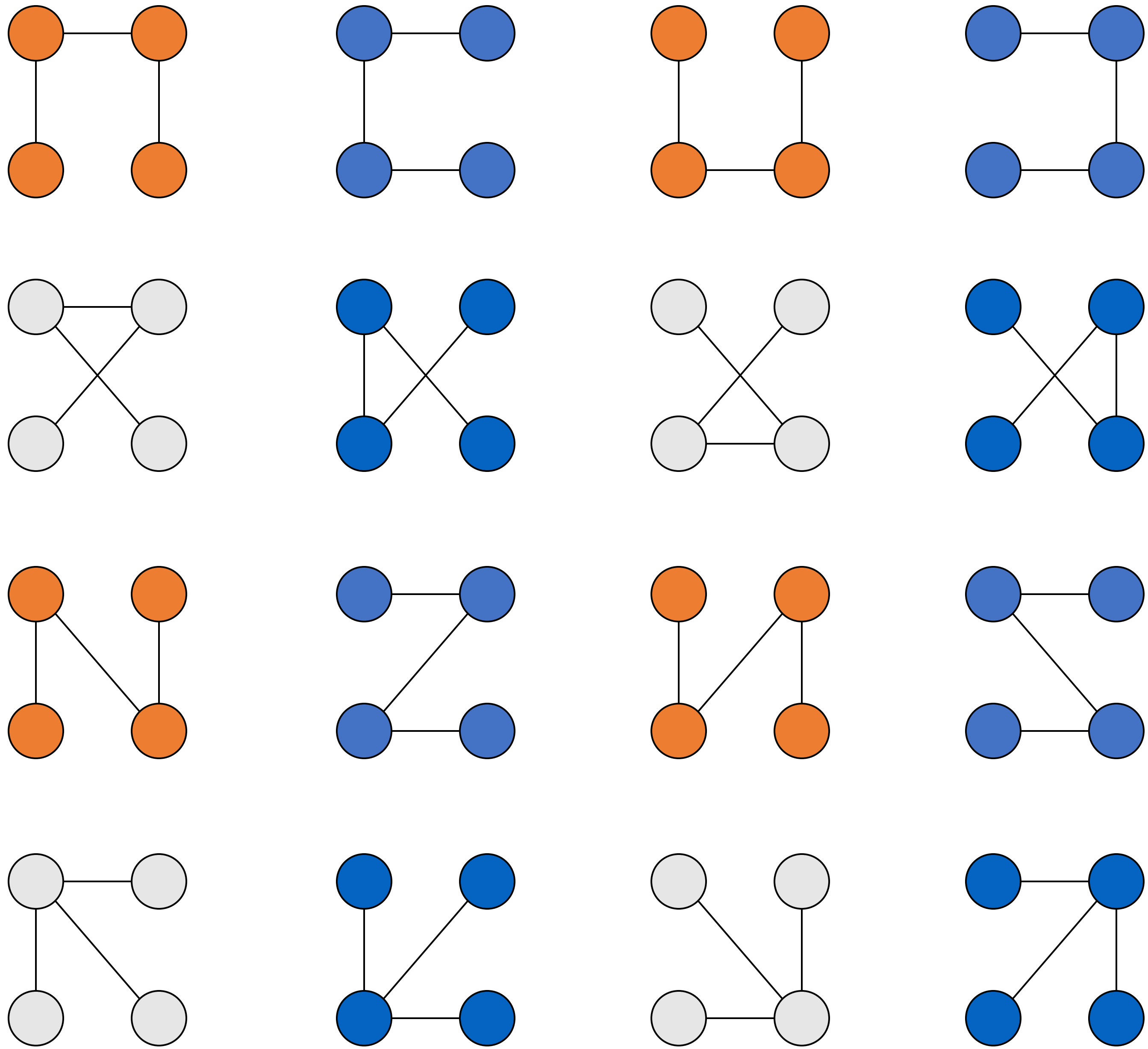
- A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.
- A graph may have many spanning trees.
- Spanning trees are defined for ***connected undirected*** graphs
- Since there are trees ➔ They have no cycles



Complete Graph



All 16 of its Spanning Trees

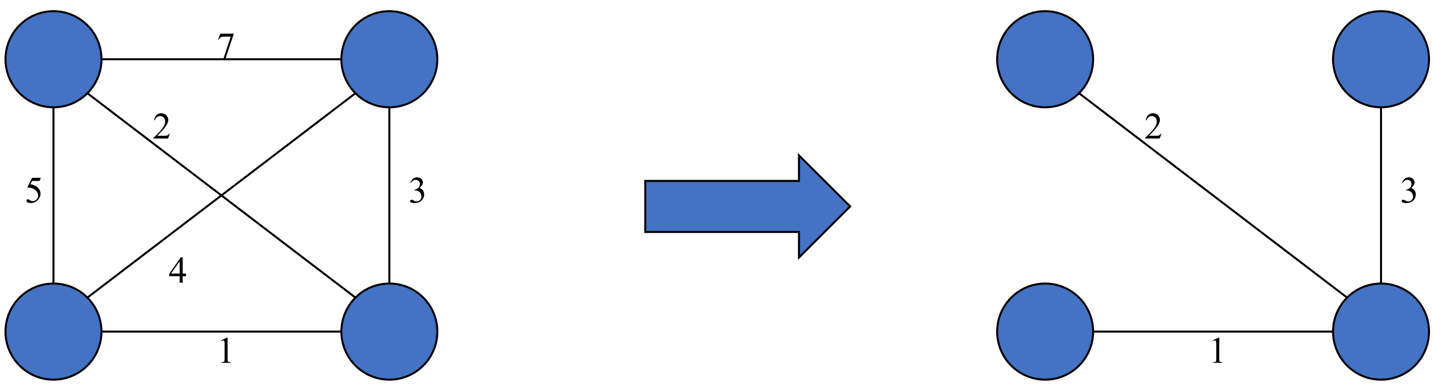


Minimum Spanning Trees

- **The Minimum Spanning Tree for a given graph is the Spanning Tree of minimum cost for that graph.**
- **Defined for *connected, undirected, and weighted* graphs**

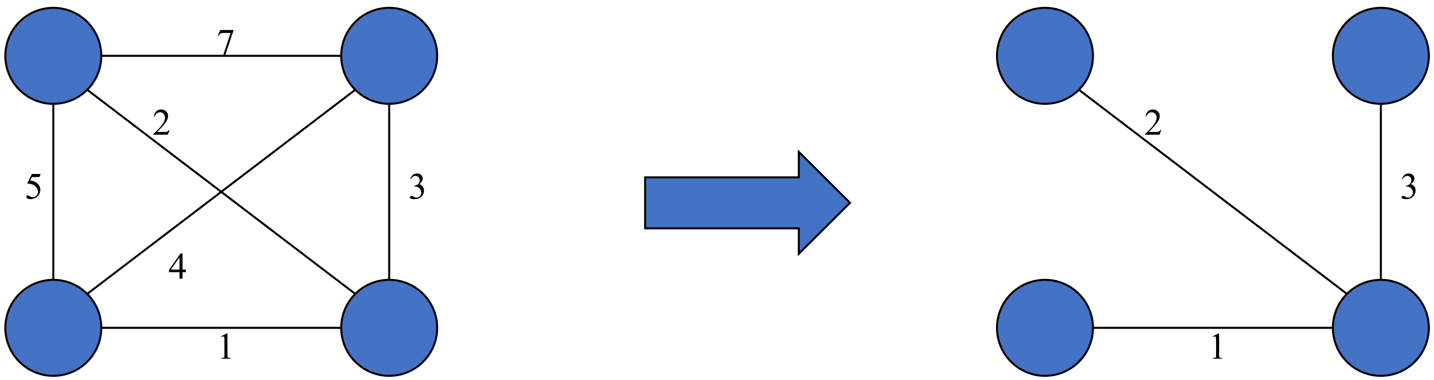
Complete Graph

Minimum Spanning Tree



Minimum Spanning Trees (MST)

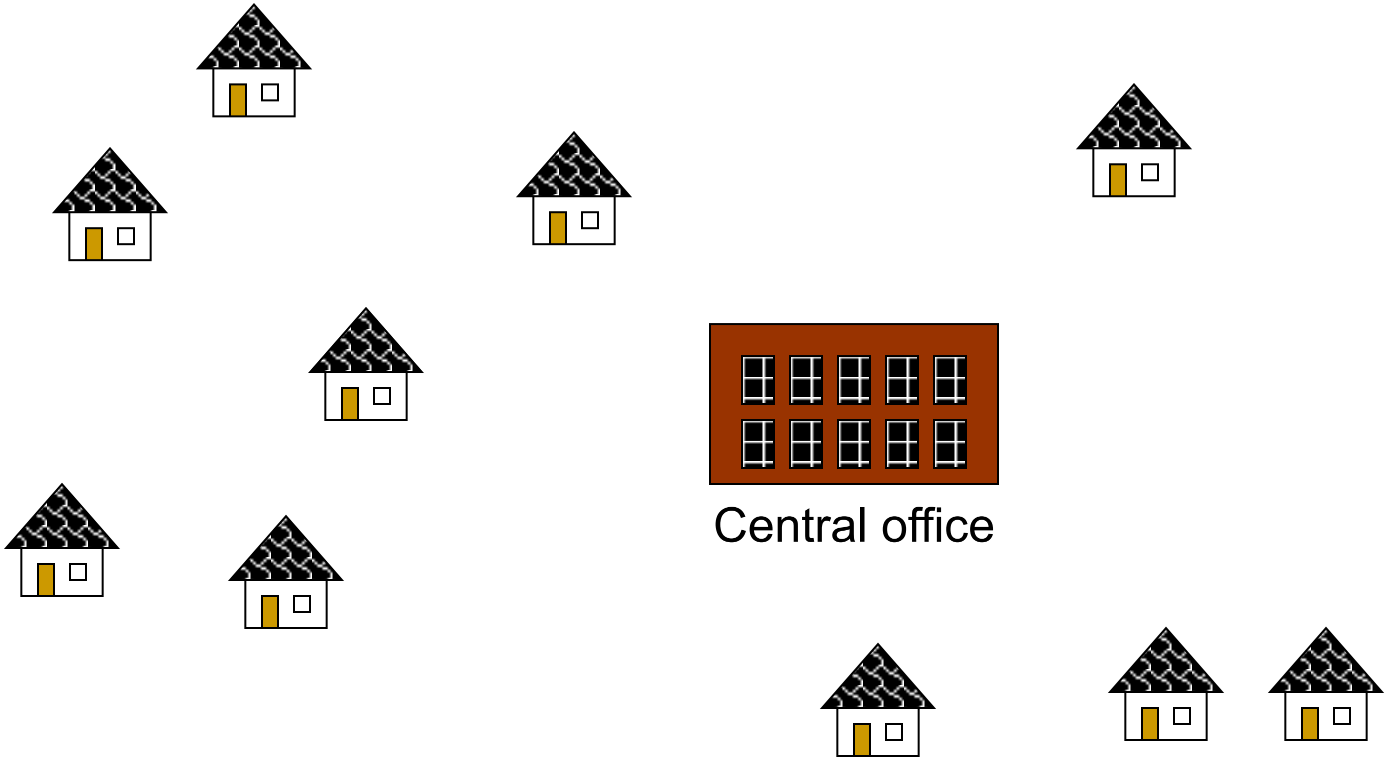
Complete Graph Minimum Spanning Tree



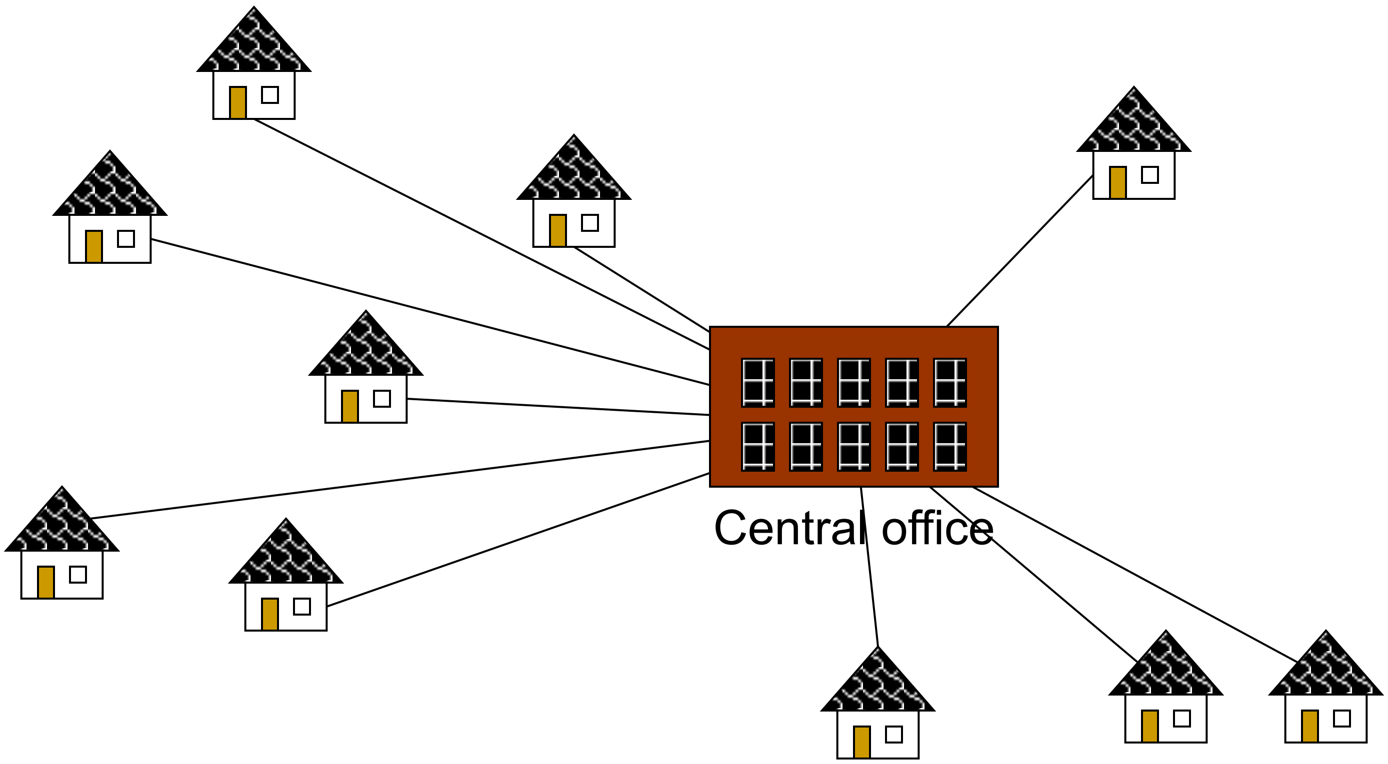
- **If each edge e_i = (u,v) has a weight or cost w_i**
- **Minimum spanning tree is the smallest subset of edges MST that connect all nodes such that:**

$\Sigma\ w_i$ is minimized

Example: Laying Telephone Wire

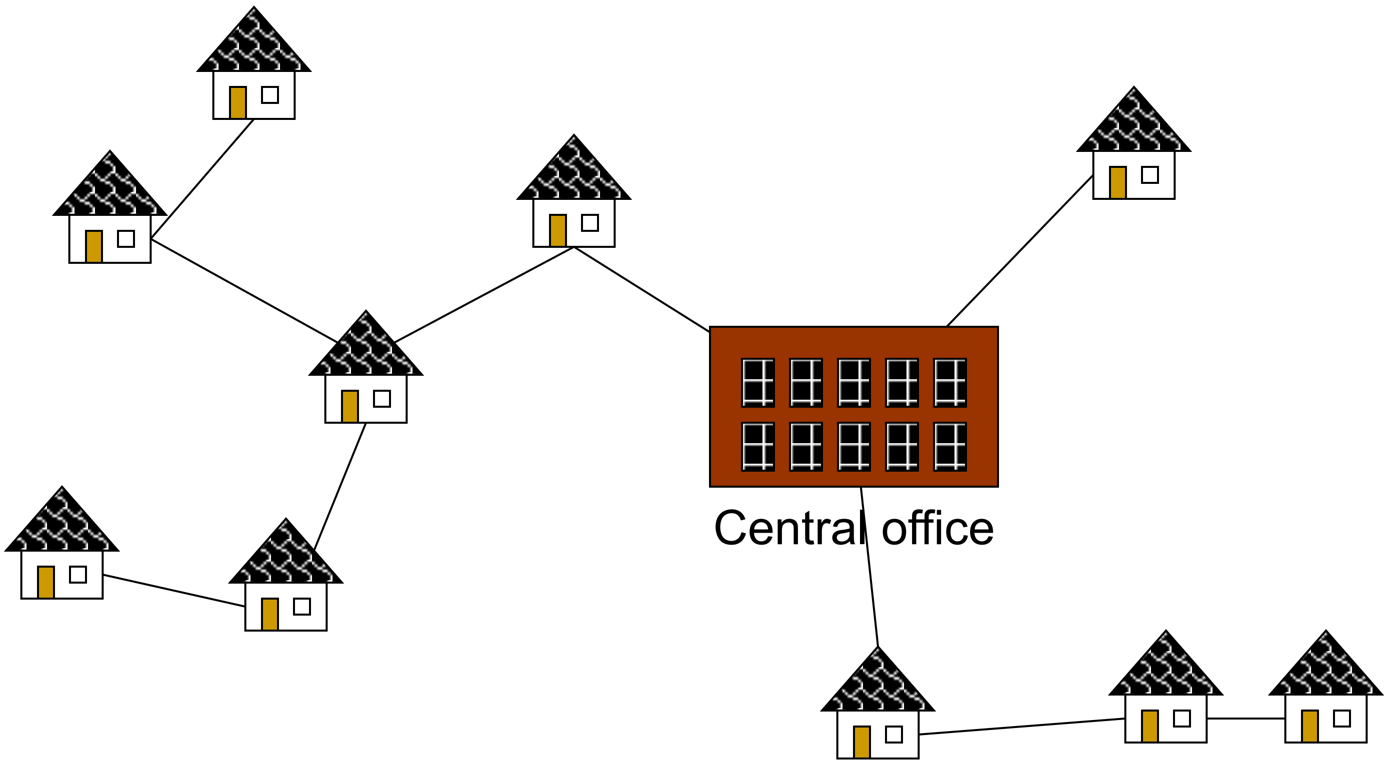


Wiring: Naive Approach



Expensive!

Wiring: Better Approach



Minimize the total length of wire connecting the customers

Algorithms for Obtaining the Minimum Spanning Tree

- **Kruskal's Algorithm**
- **Prim's Algorithm**

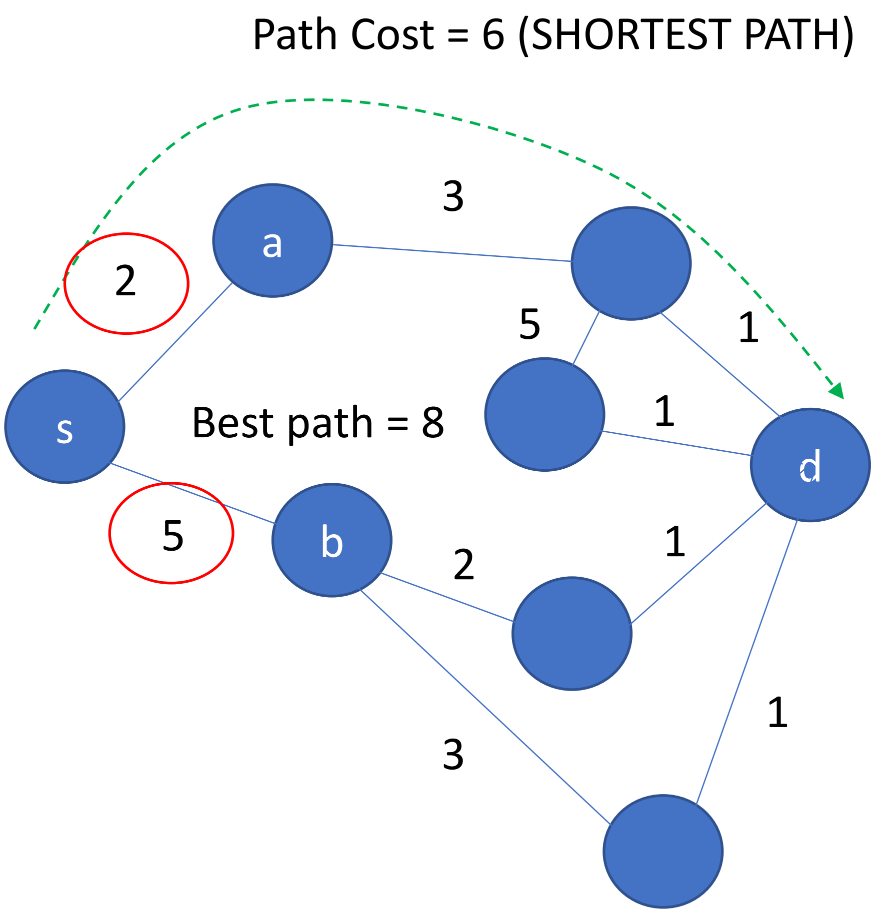
Both of these are Greedy Algorithms

Greedy Algorithms

Find the shortest path route to the destination.

- A **greedy algorithm** is any algorithm that follows the problem-solving heuristic of **making the locally optimal choice at each stage with the intent of finding a global optimum.**

- Can be a very easy to use heuristic and often produces optimal (best solutions)

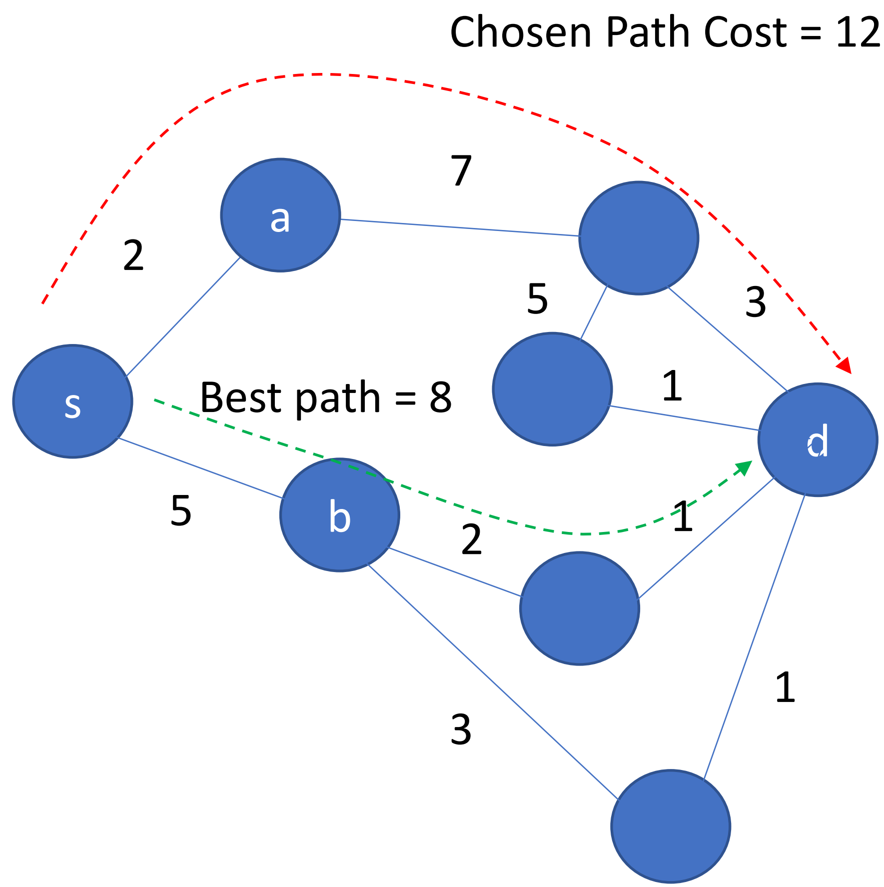


Local optimal solutions: link s-a is preferred over link s-b. It's a locally optimal decision
The hope is that such **local optimal decisions** at every point (pick the shortest link) will produce a **globally optimal decision** (shortest path)

Greedy Algorithms

- Greedy algorithms don't always work, because local optimal is not the global optimal

- Here the green path is better than the red path, but we will never take it because of our greedy approach



Just like Divide and Conquer, Greedy Algorithms are another class of algorithms that we often use

Kruskal's Algorithm: Overview

1. **The algorithm creates a forest of trees (many trees).**
2. **Initially each forest consists of a single node (and no edges).**
3. **At each step, we add one edge (<u>the cheapest one</u>) so that it joins two trees together.**

   The greedy choice

4. **If it were to form a cycle, it would simply link two nodes that were already part of a single connected tree**
   - **Skip this edge.**

The output of the Kruskal's algorithm is a list of edges that together (1) form a tree, (2) have minimum cost when added up

Kruskal's Algorithm: Overview

1. The forest is constructed - with each node in a separate tree.

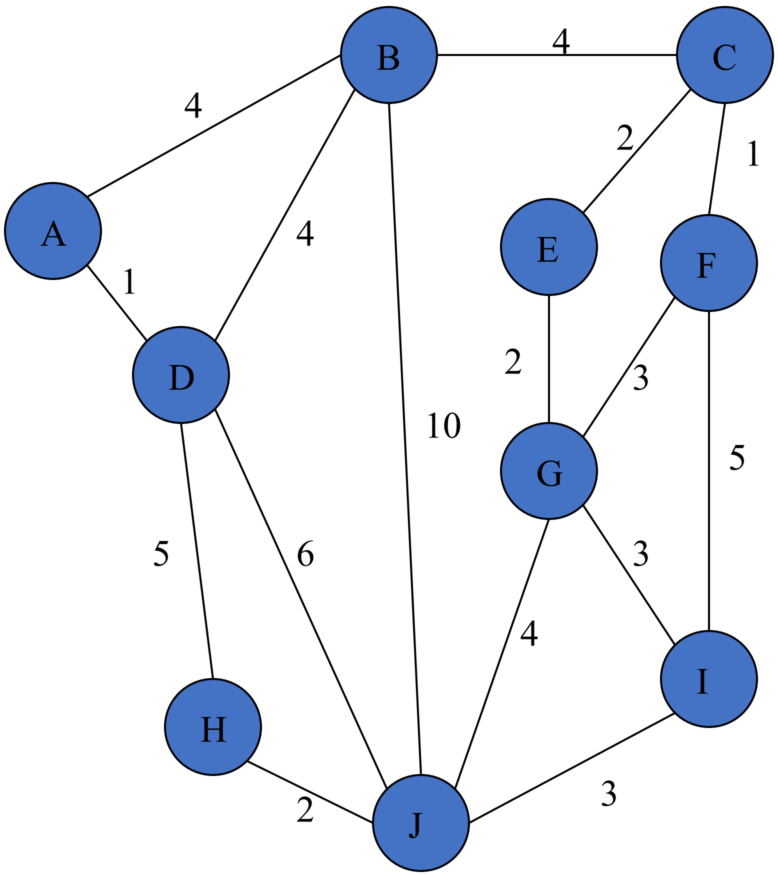
2. The edges are placed in a min-priority queue.

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.

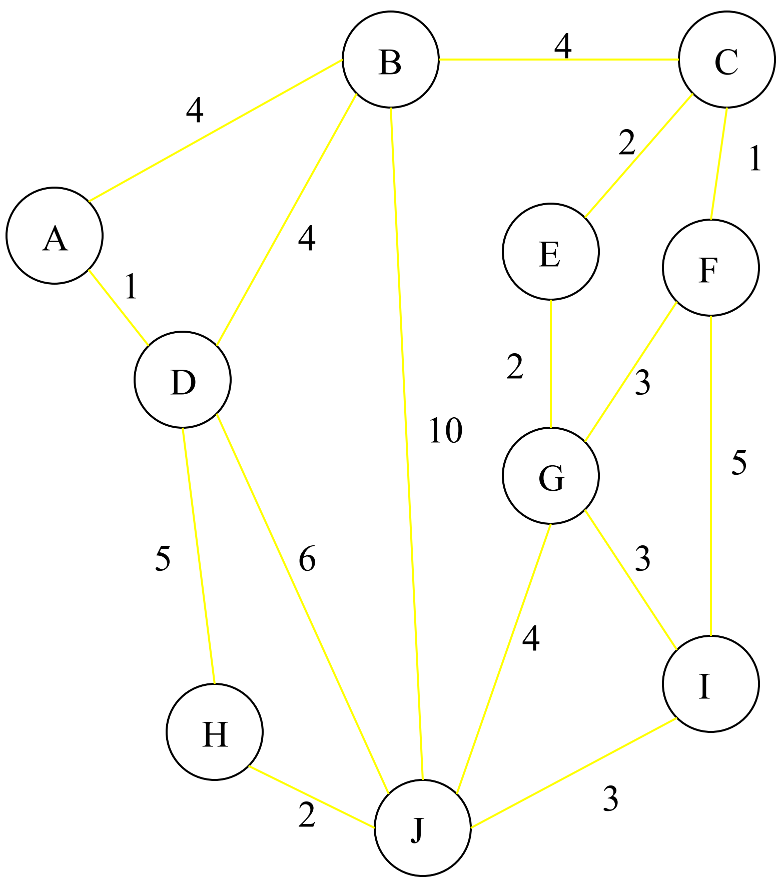
What is this?

- If we start with n nodes (n separate trees)
- Each step we connect two trees
- Then we need (n-1) edges to get a single tree

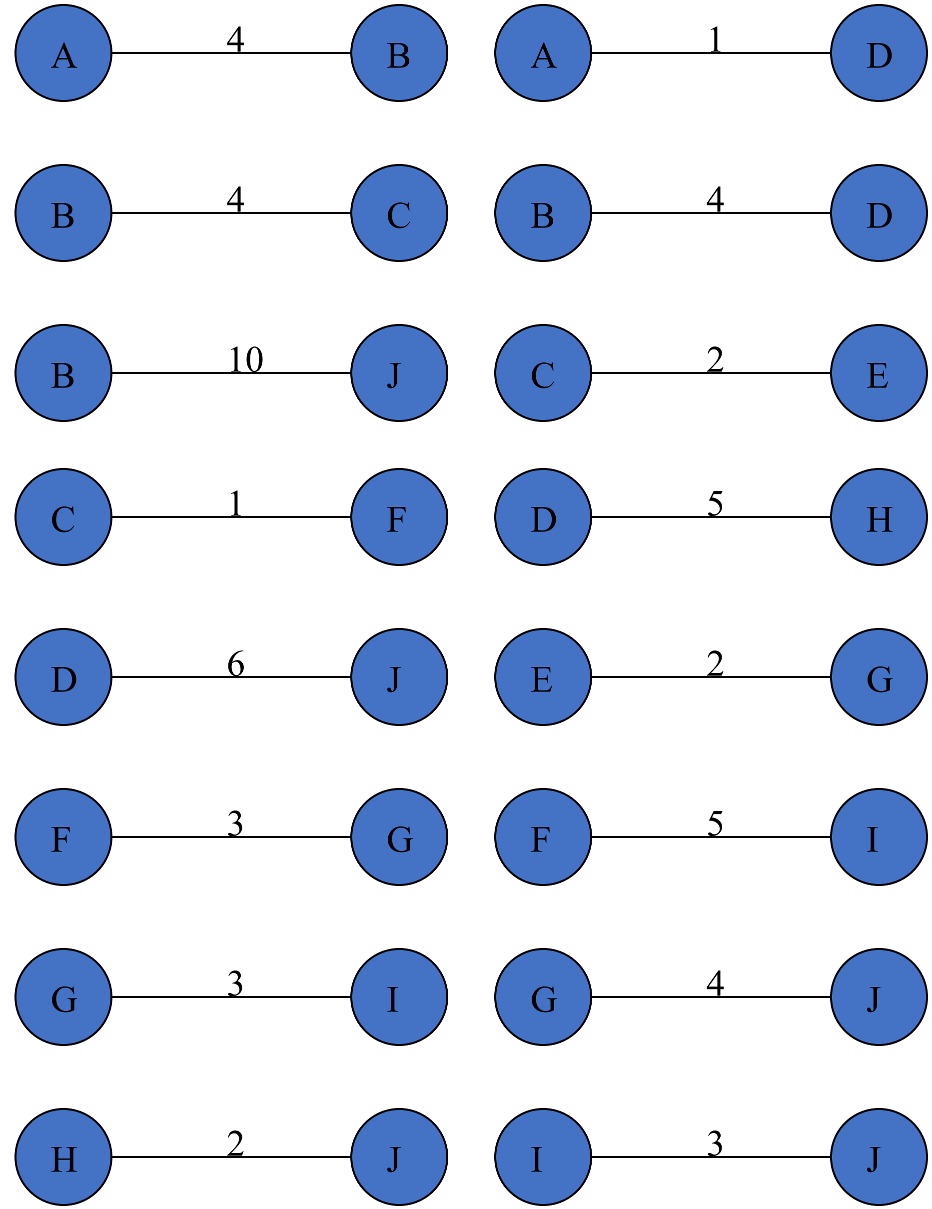
Complete Graph



All nodes, no edges

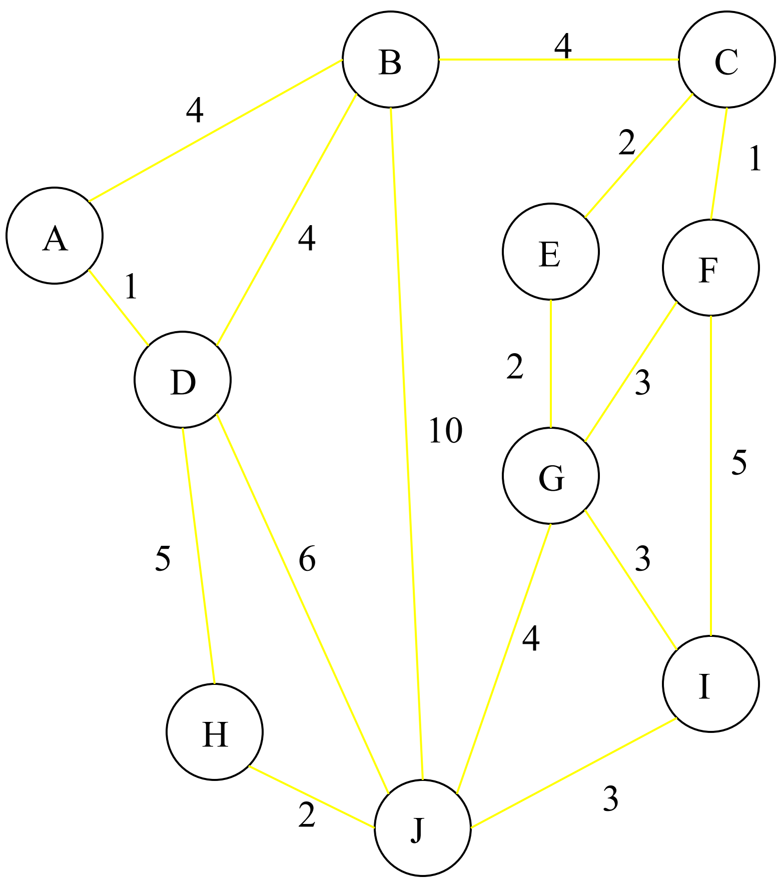


List of all edges

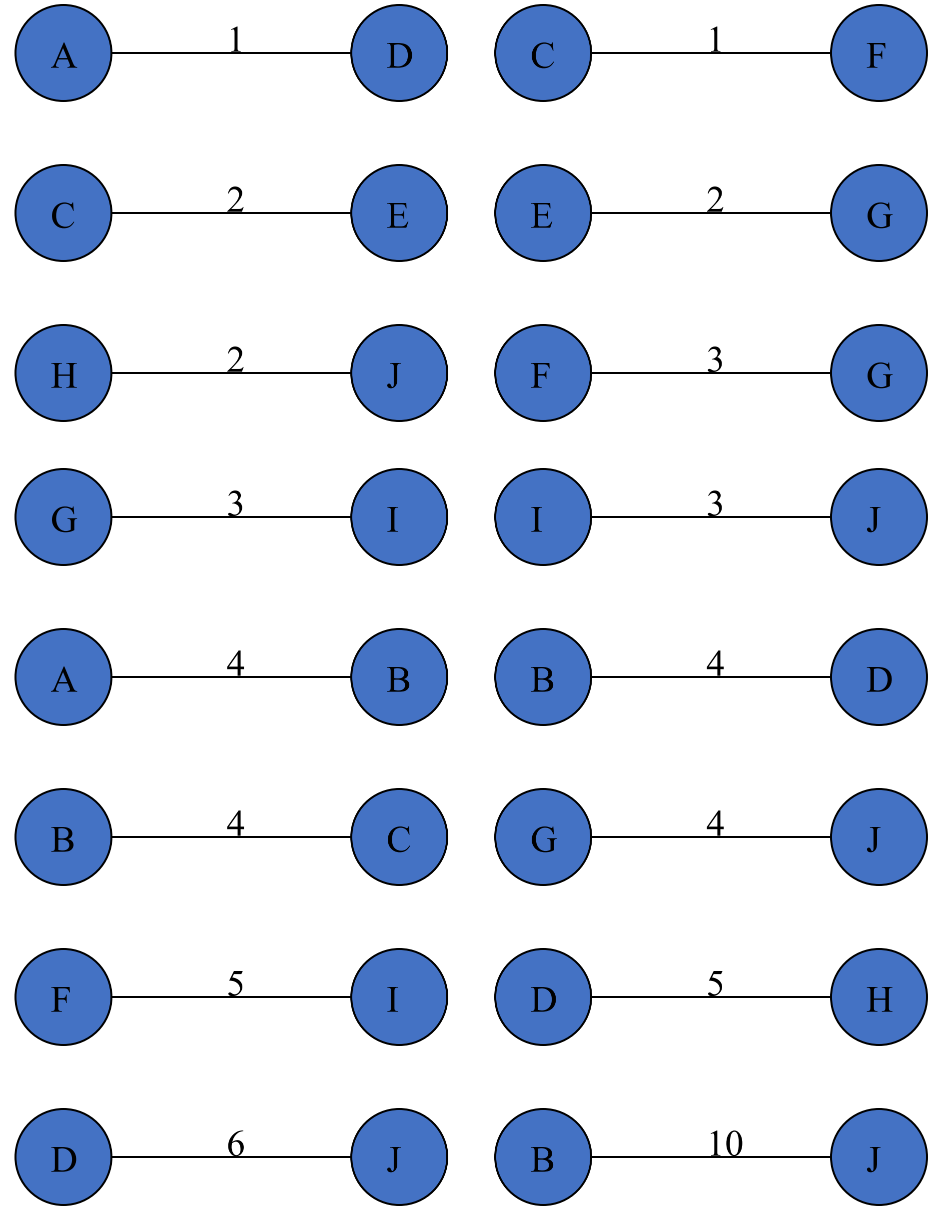


Sort Edges

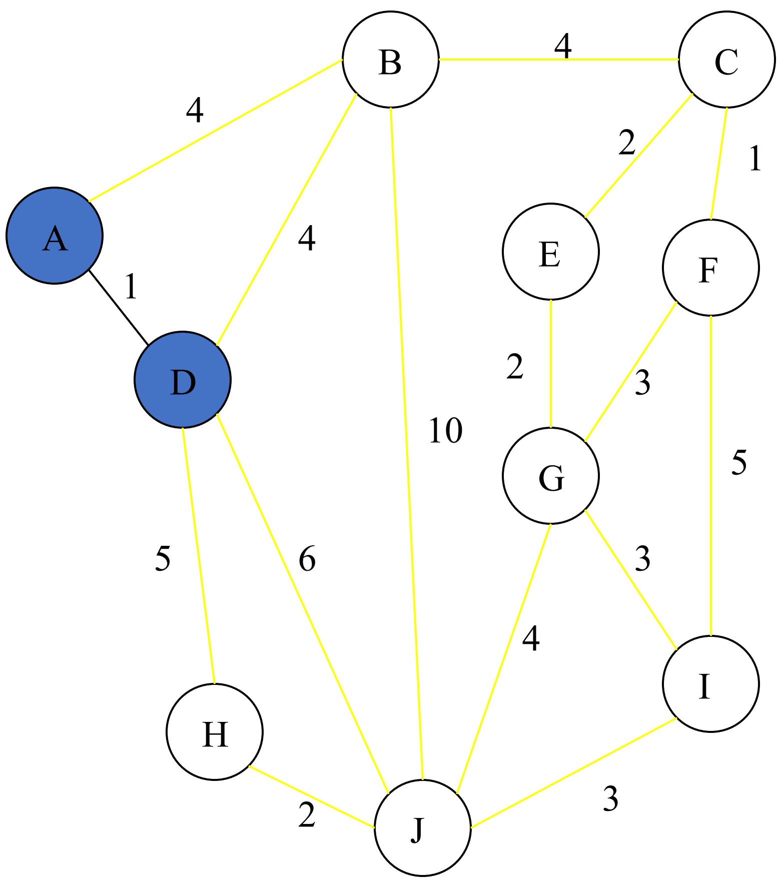
(in reality they are placed in a priority queue - not sorted - but sorting them makes the algorithm easier to visualize)



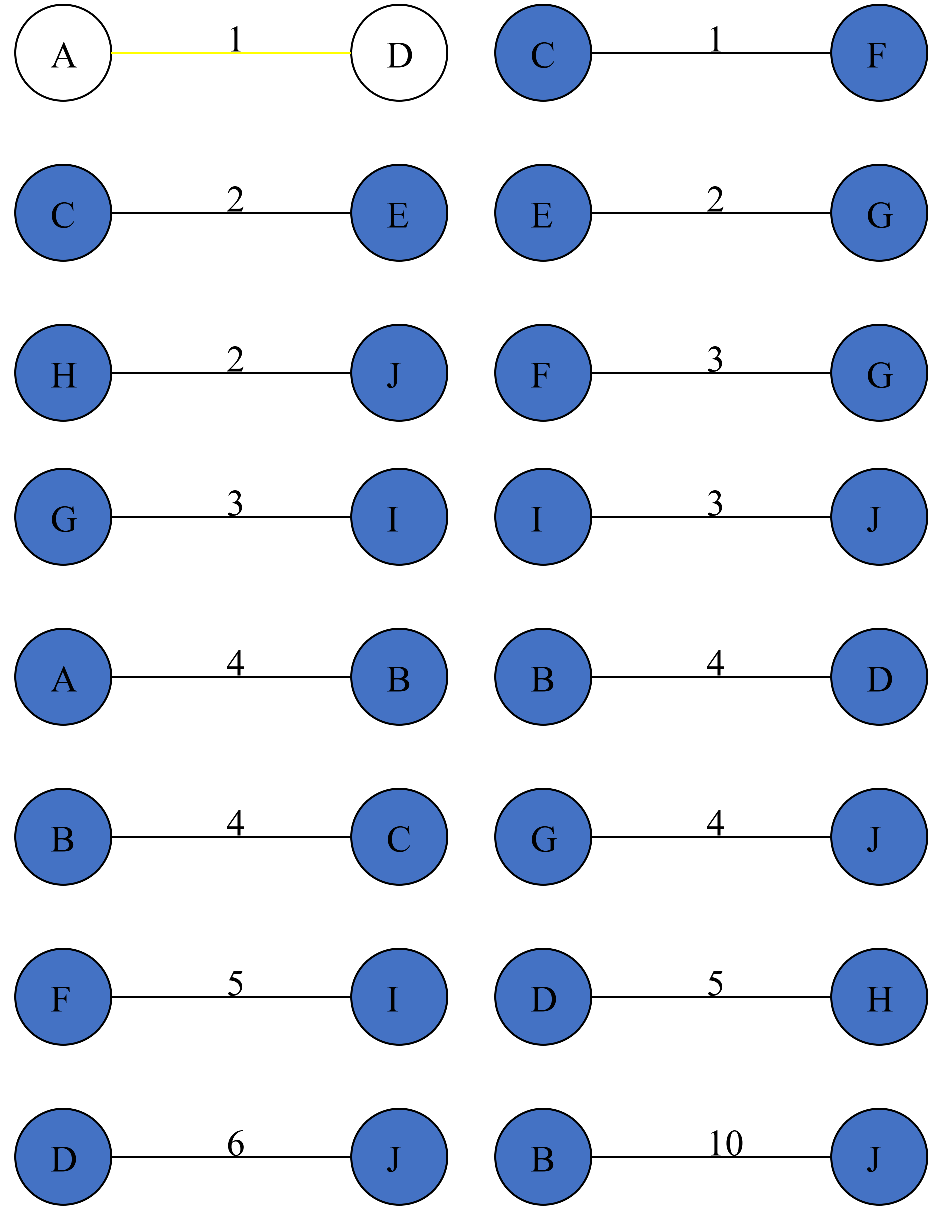
Sort the edges



Add Edge

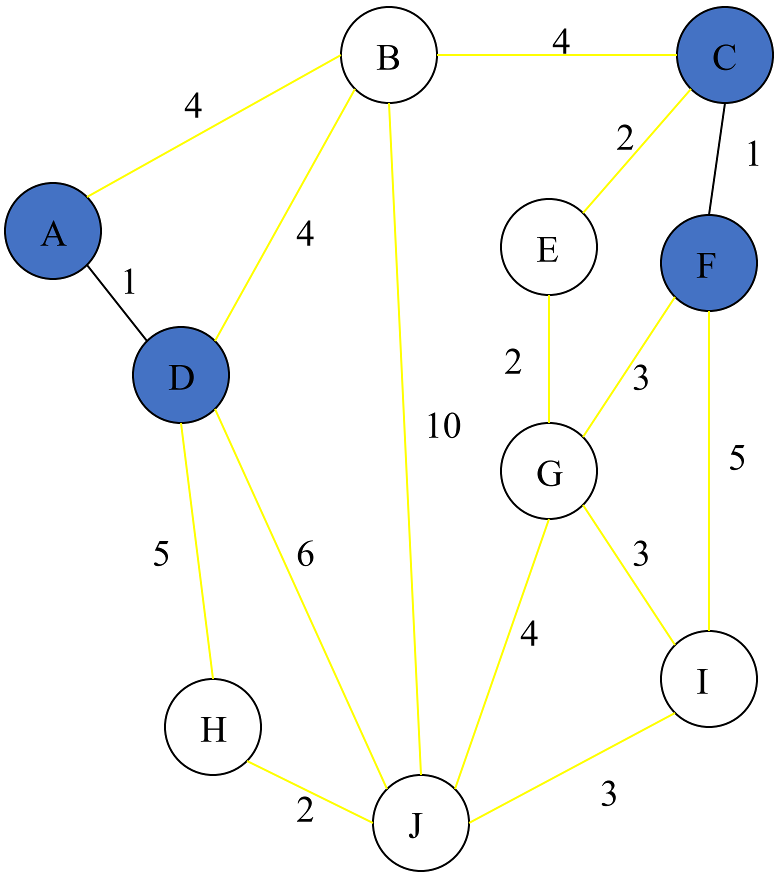


A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
4
3
2
3

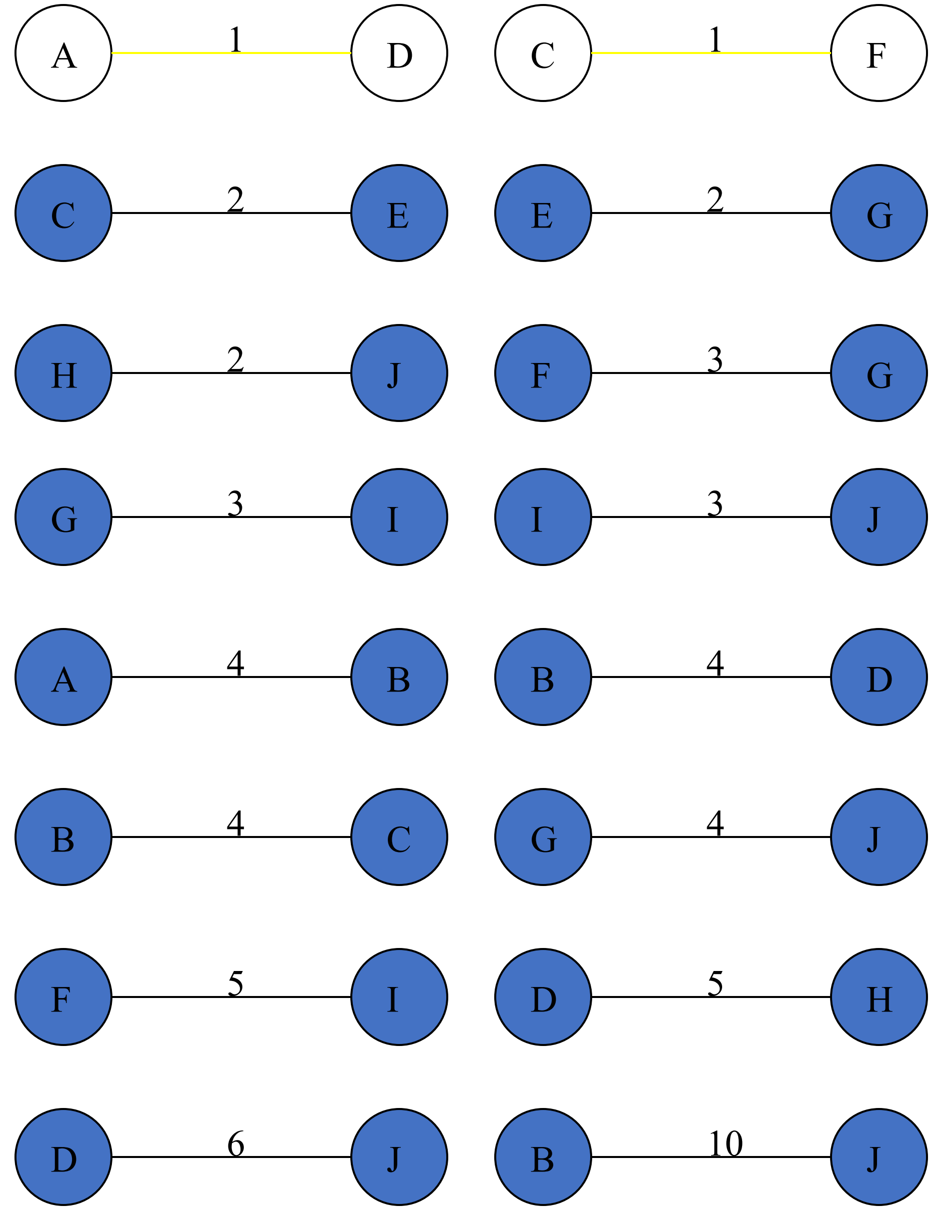


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge

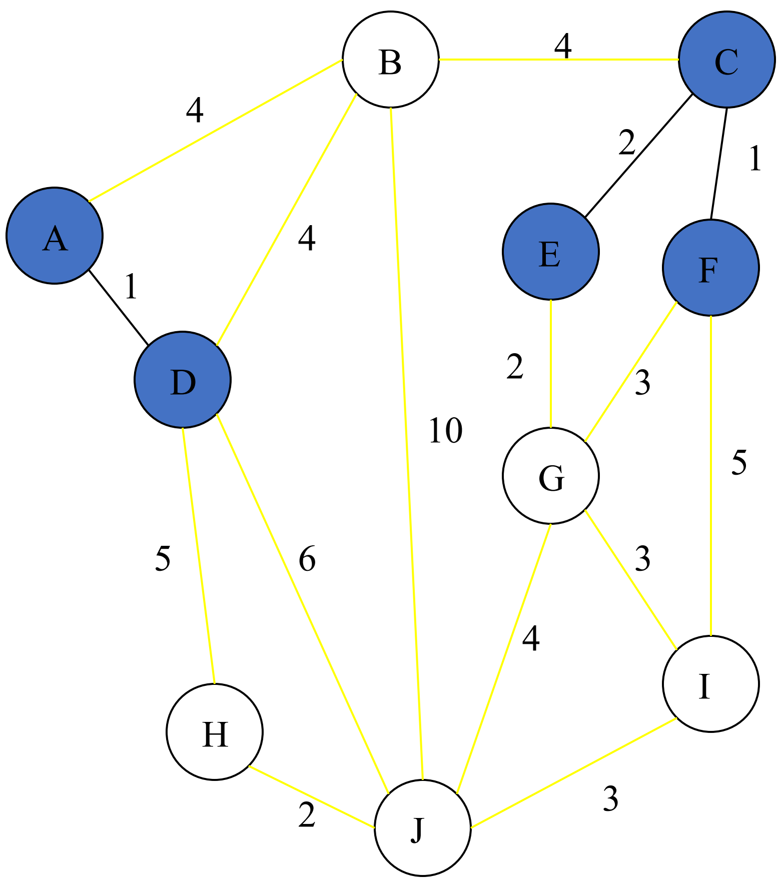


A
B
C
D
E
F
G
H
I
J
4
4
4
1
1
2
2
3
10
5
5
6
4
3
2
3

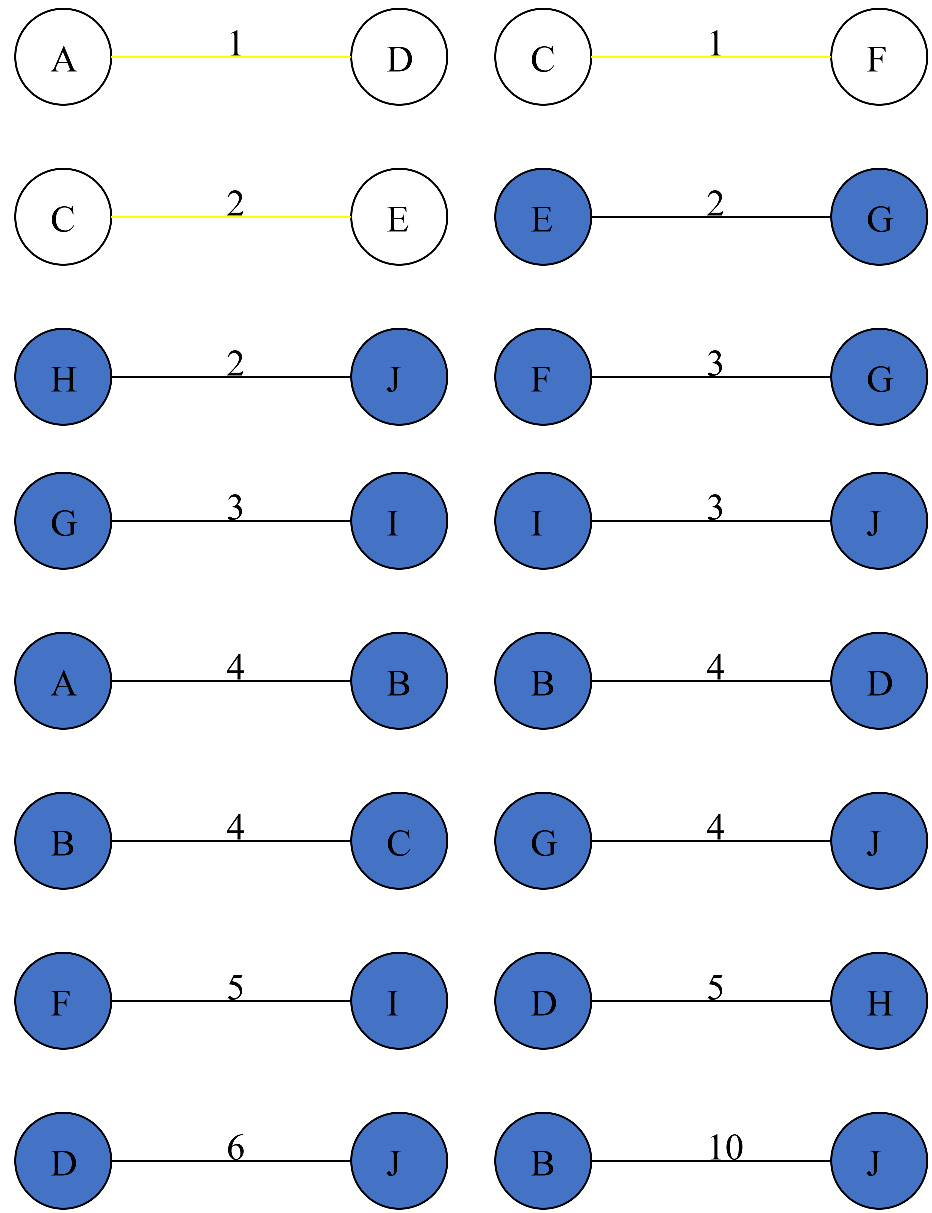


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge

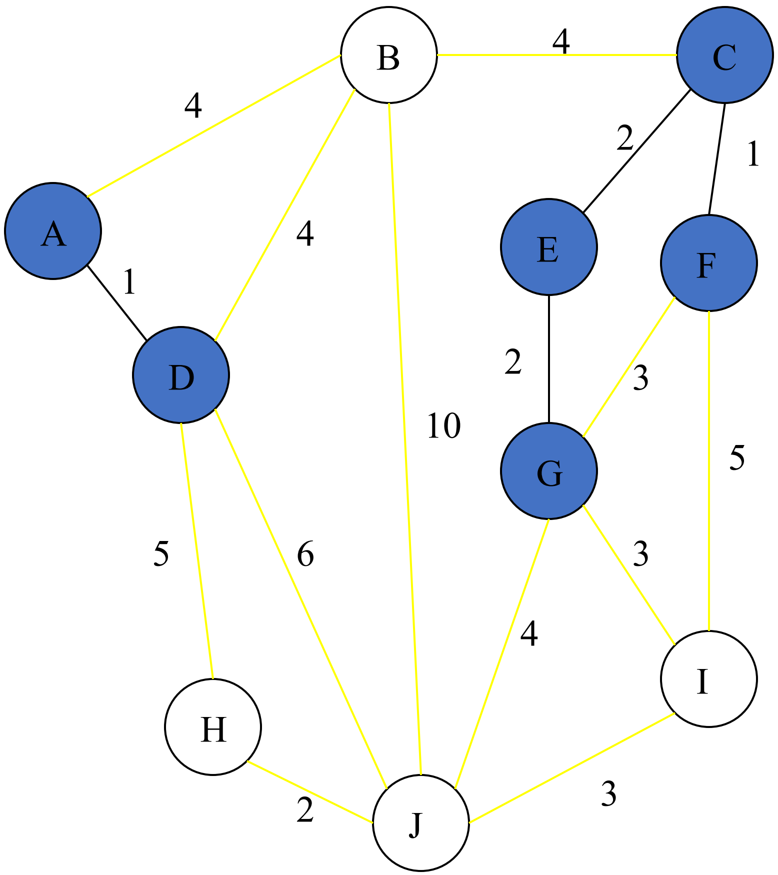


A
B
C
D
E
F
G
H
I
J
4
4
4
1
2
1
2
3
3
5
10
5
6
4
2
3

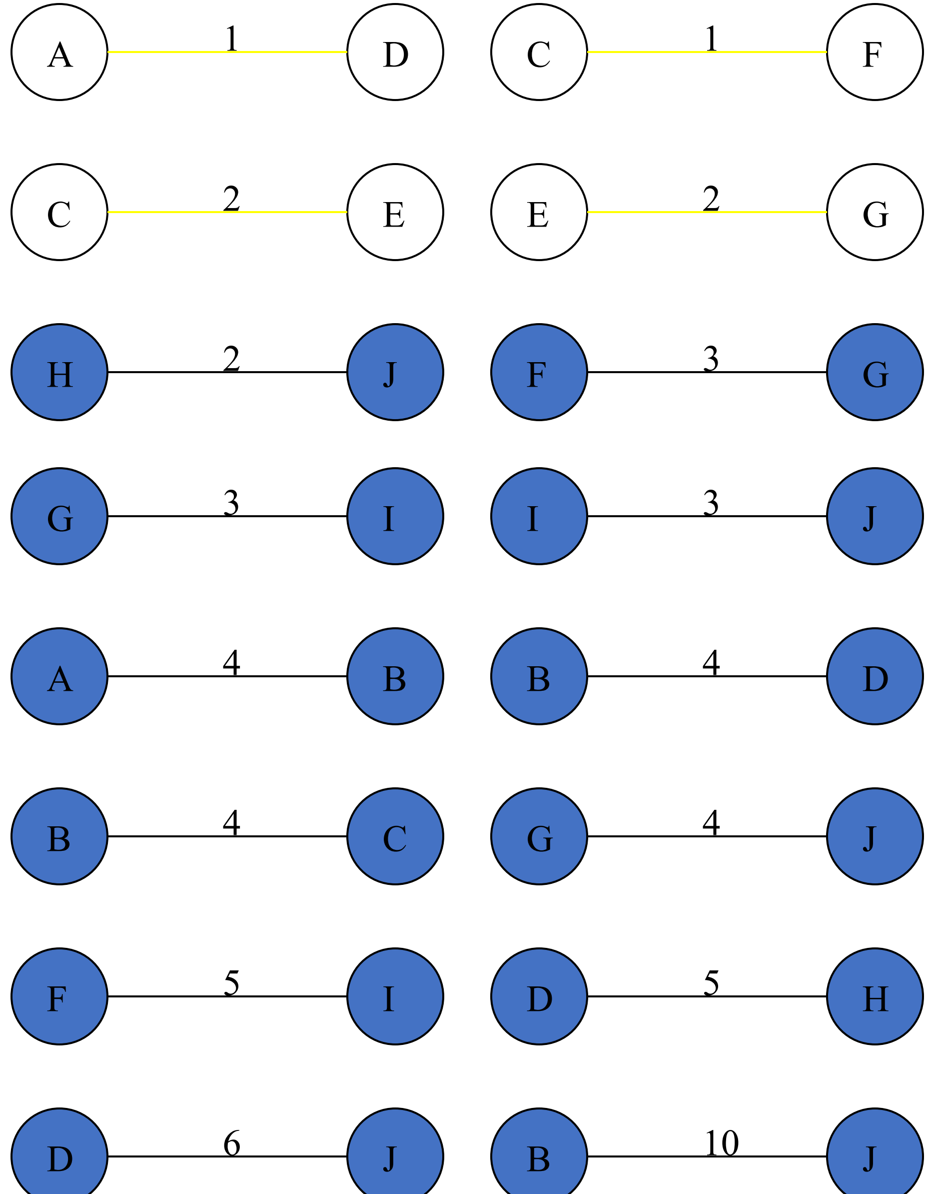


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge

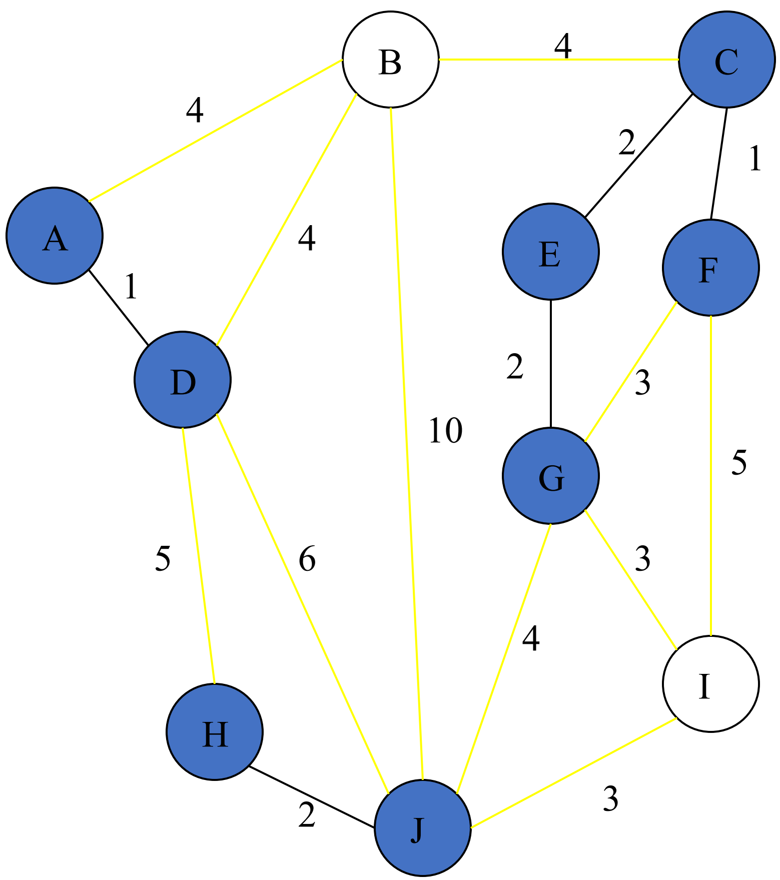


A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
2
3

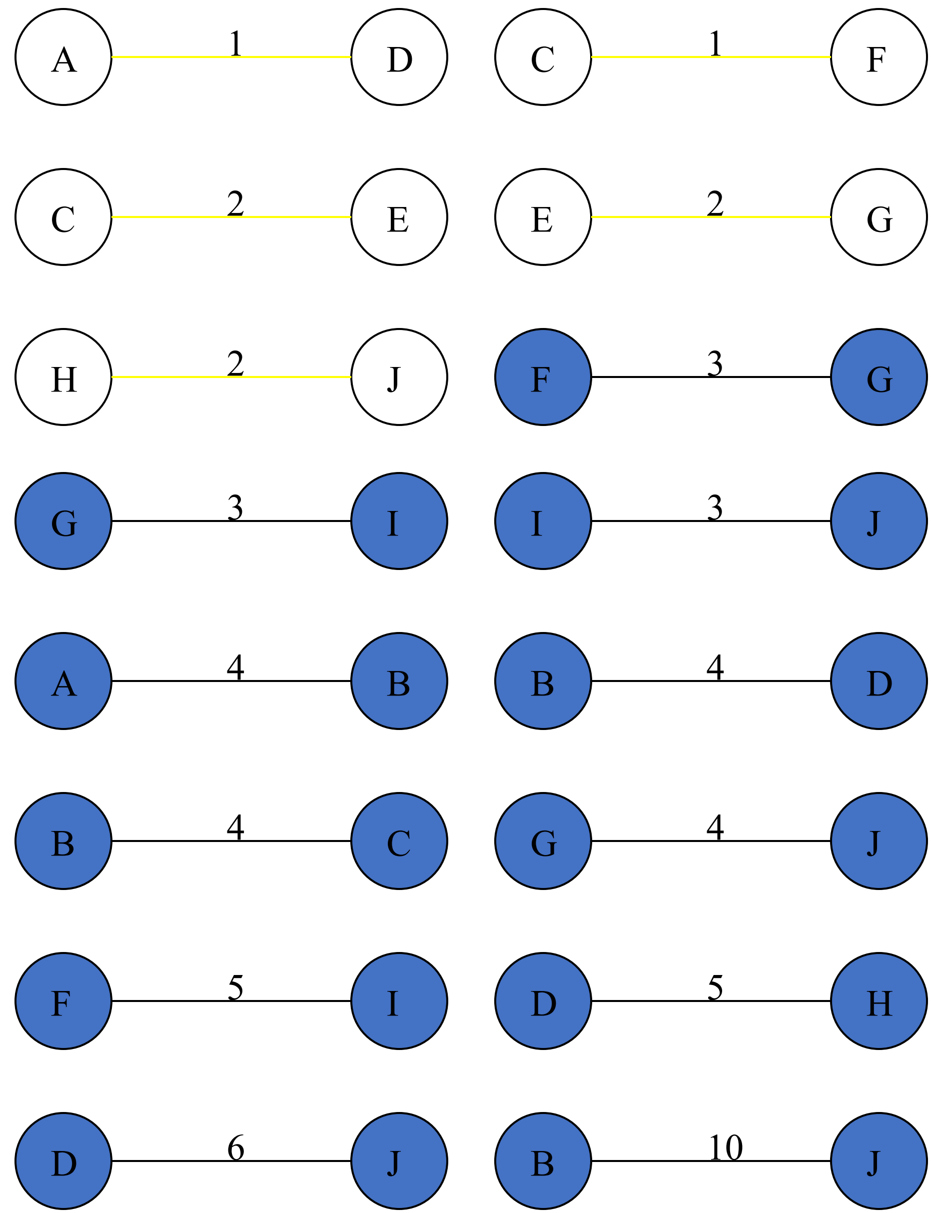


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge



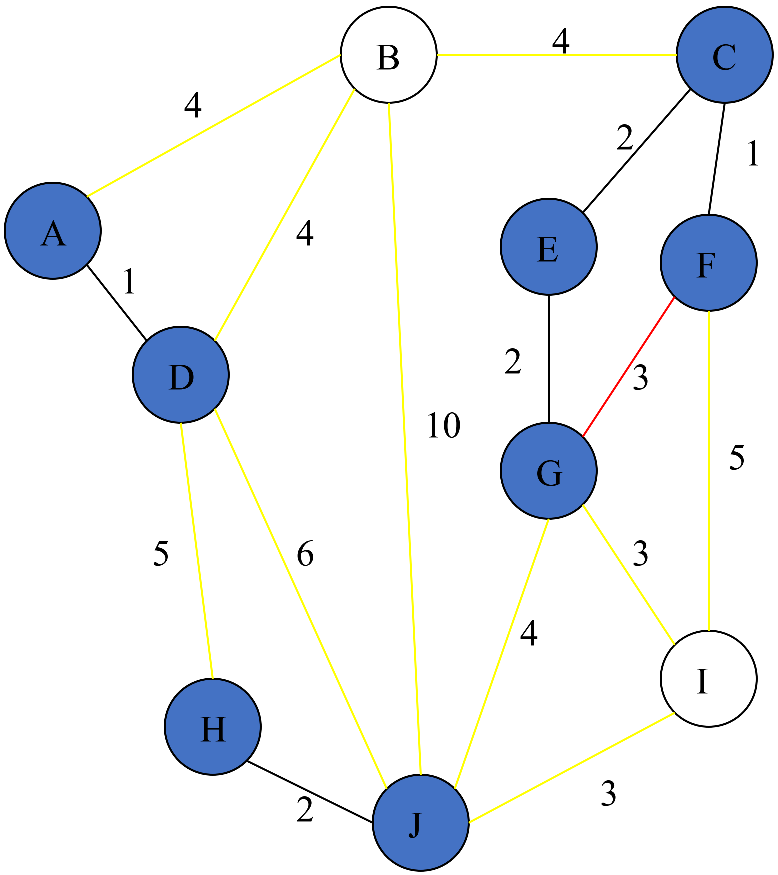
A
B
C
D
E
F
G
H
I
J
4
4
4
1
2
1
2
3
10
5
6
5
3
4
2
3

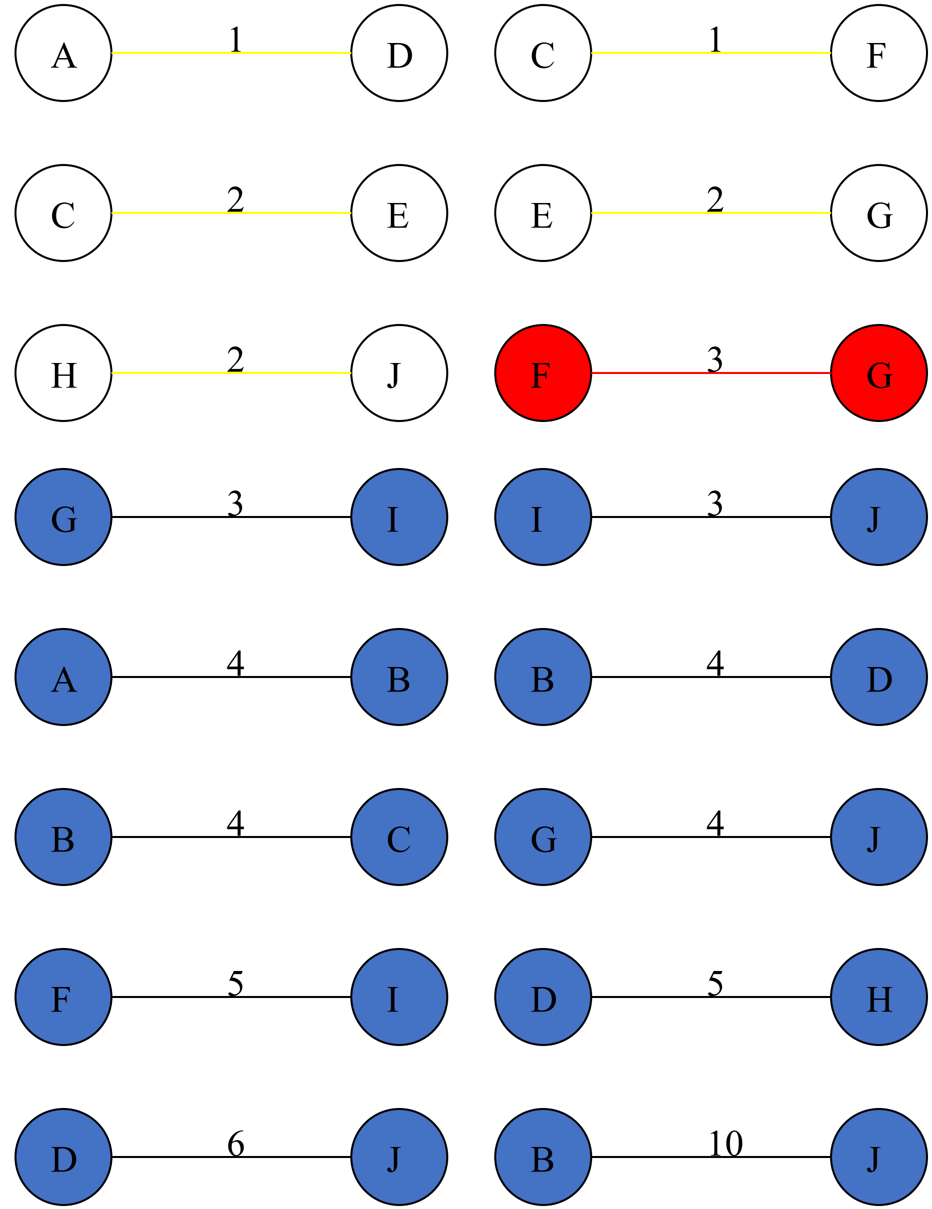


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

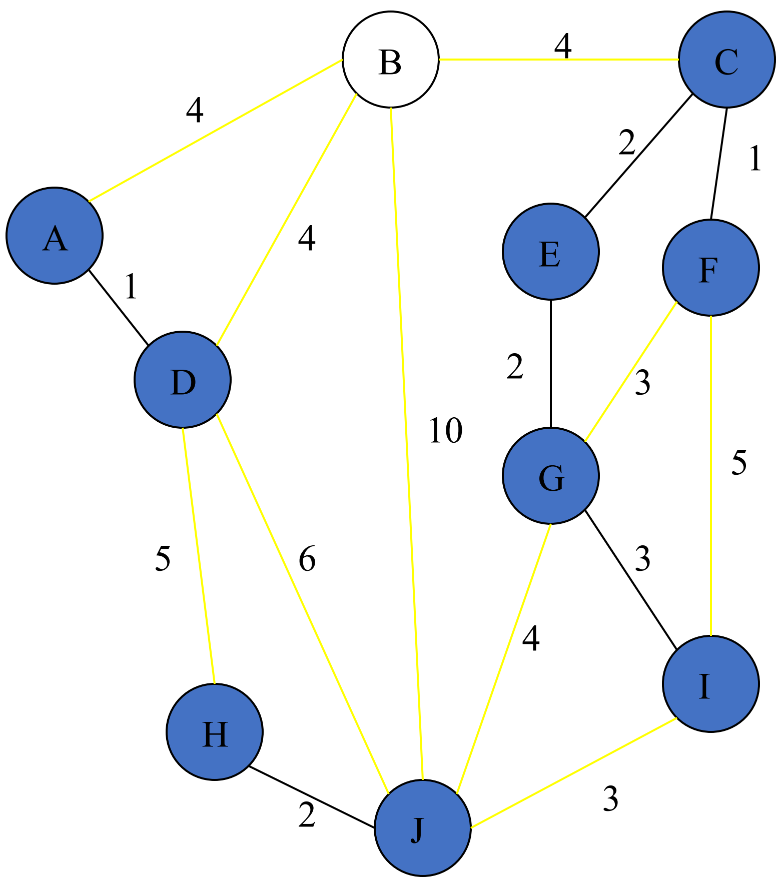
Cycle

Don’t Add Edge

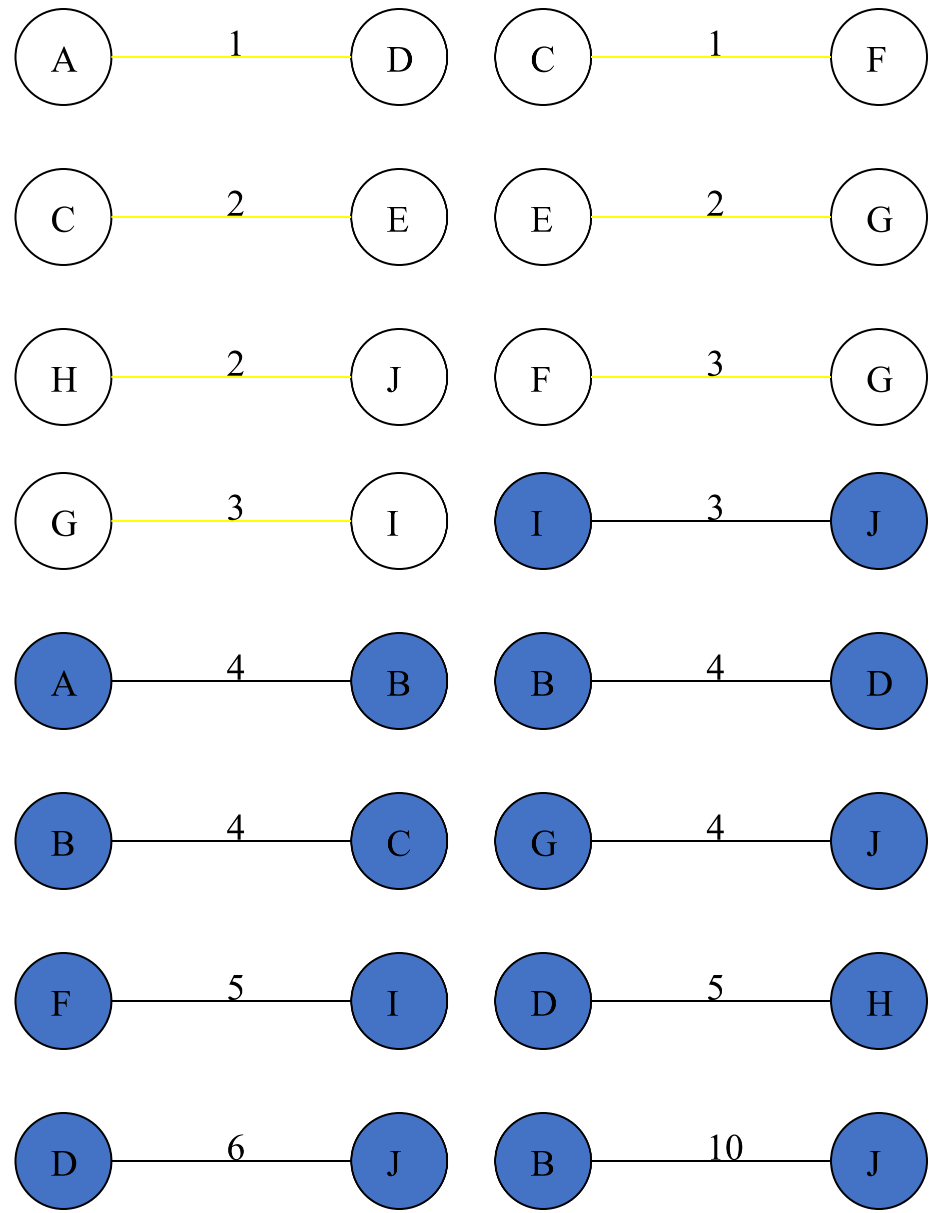




Add Edge

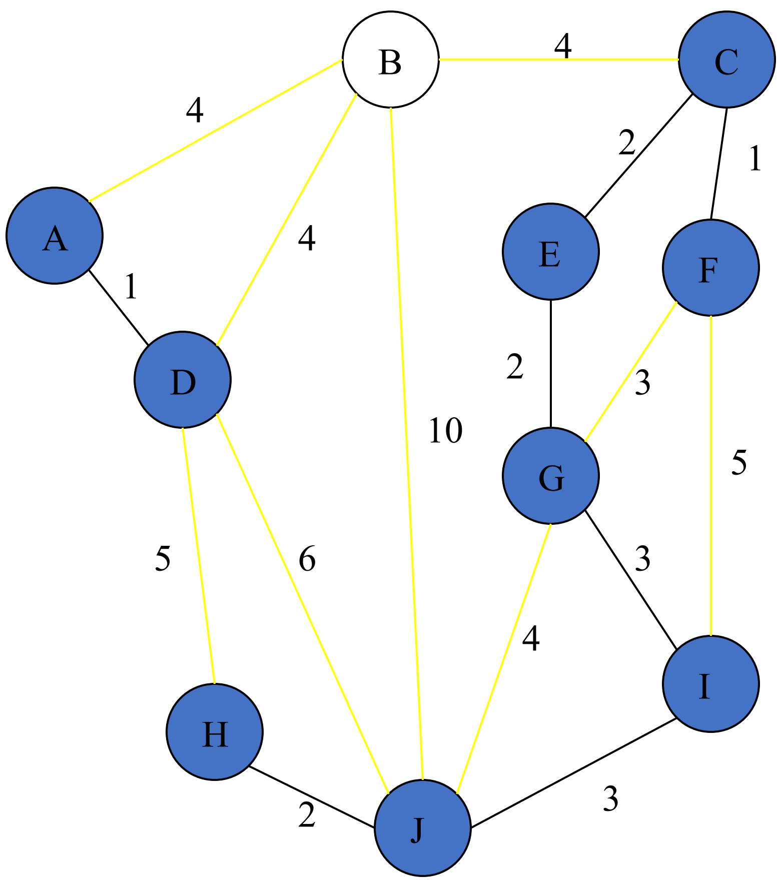


A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
5
10
5
6
3
4
2
3

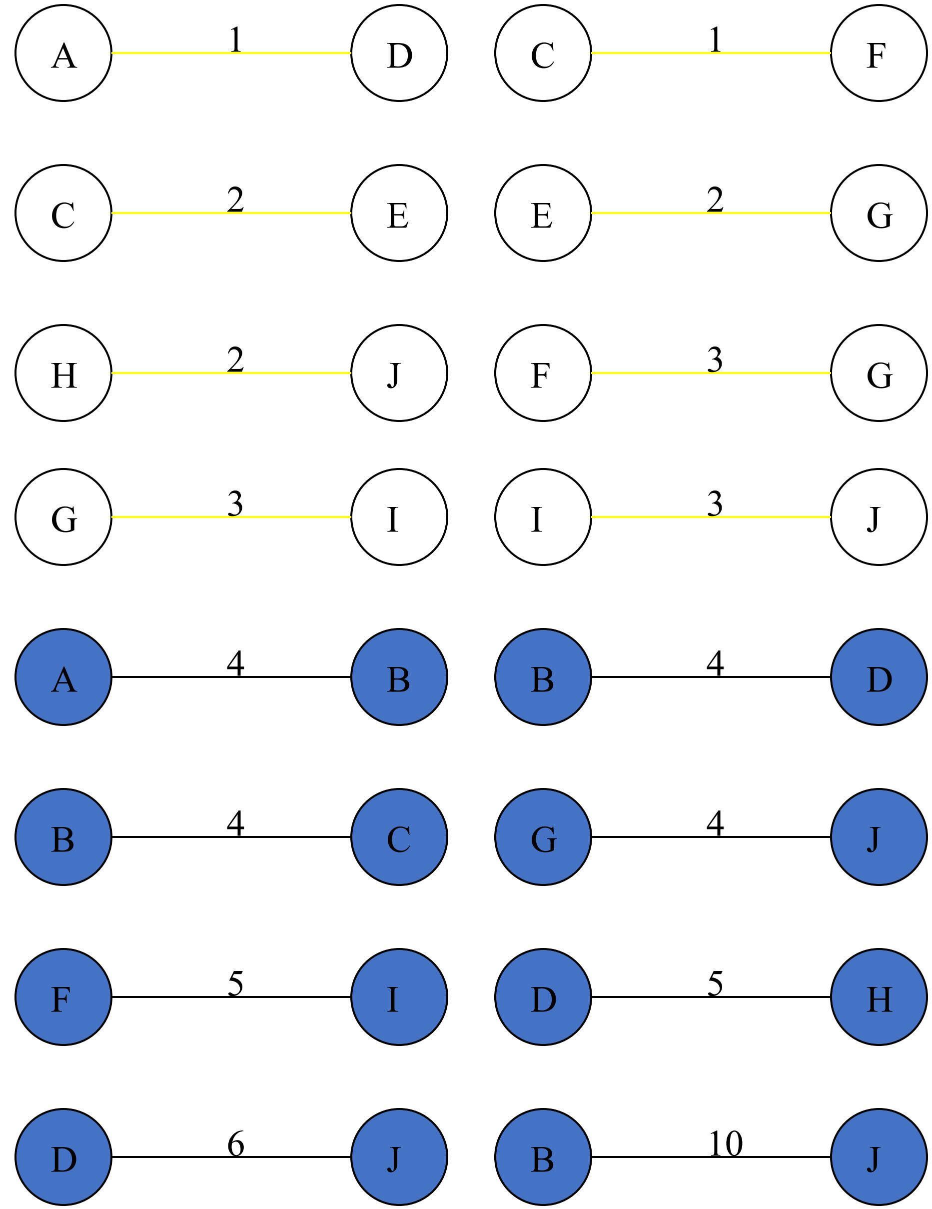


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge

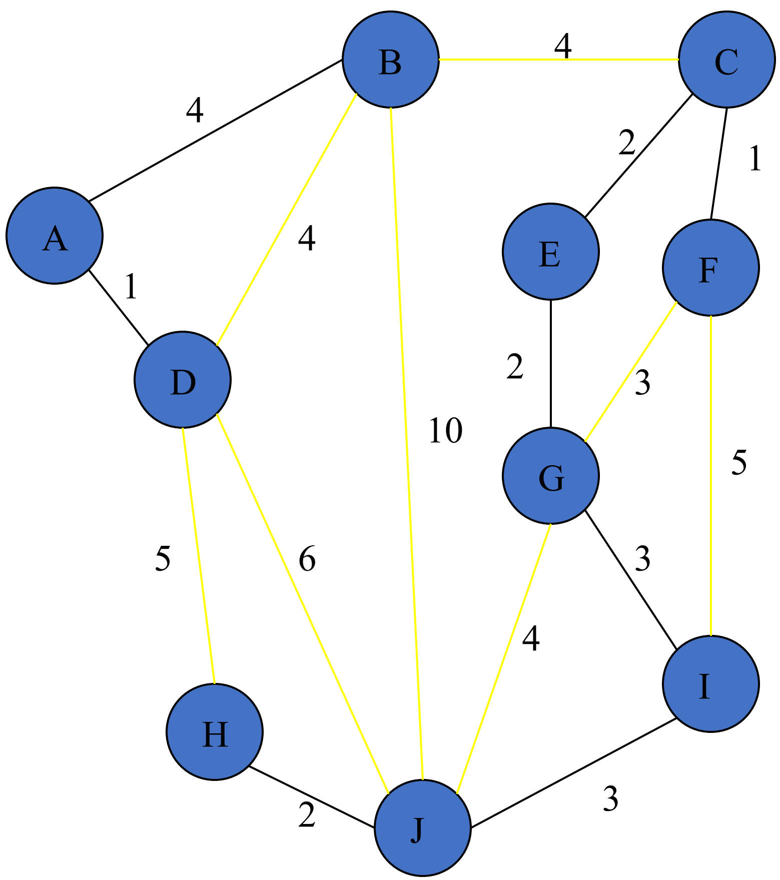


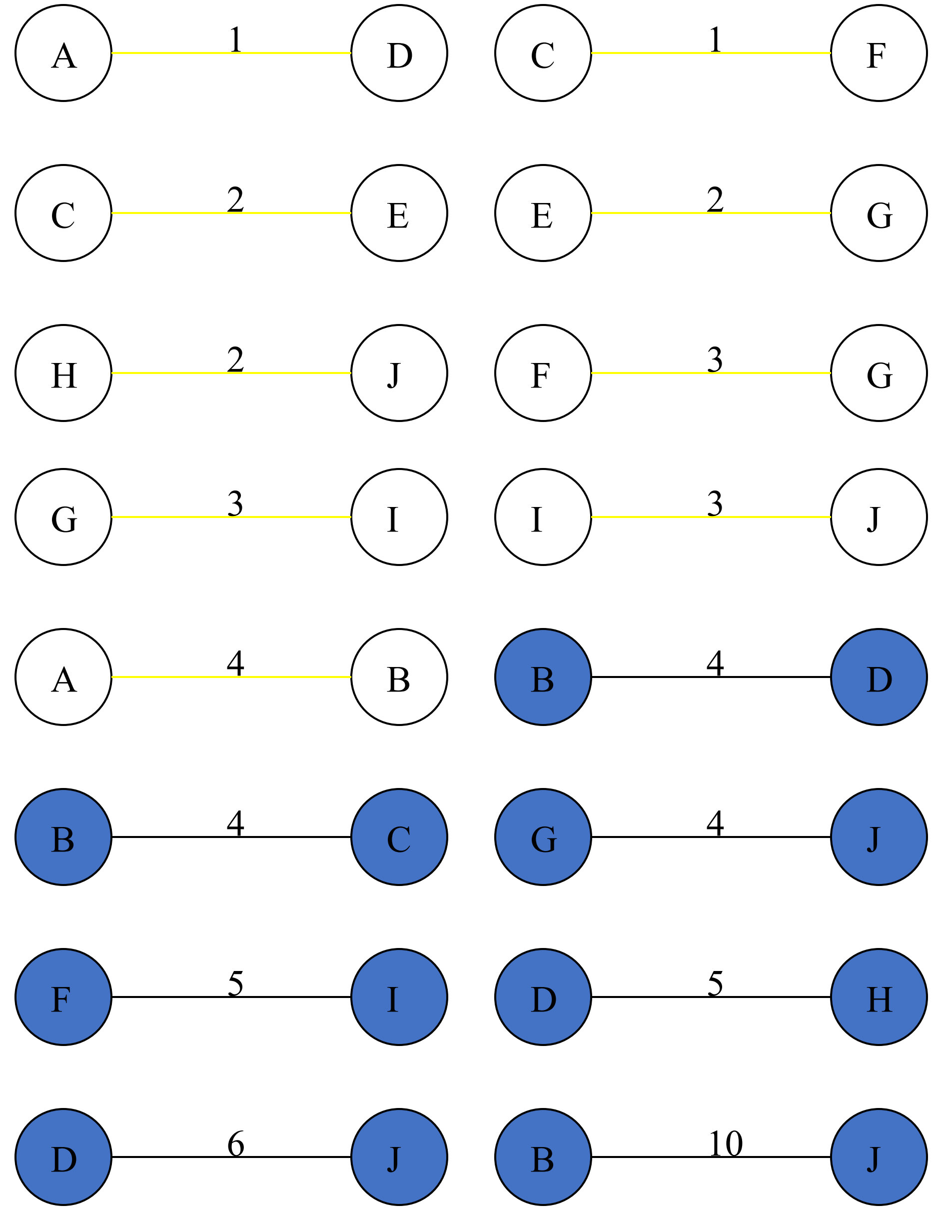
B
C
A
E
F
D
G
H
J
I
4
4
2
1
1
4
2
3
10
5
5
6
3
4
2
3



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

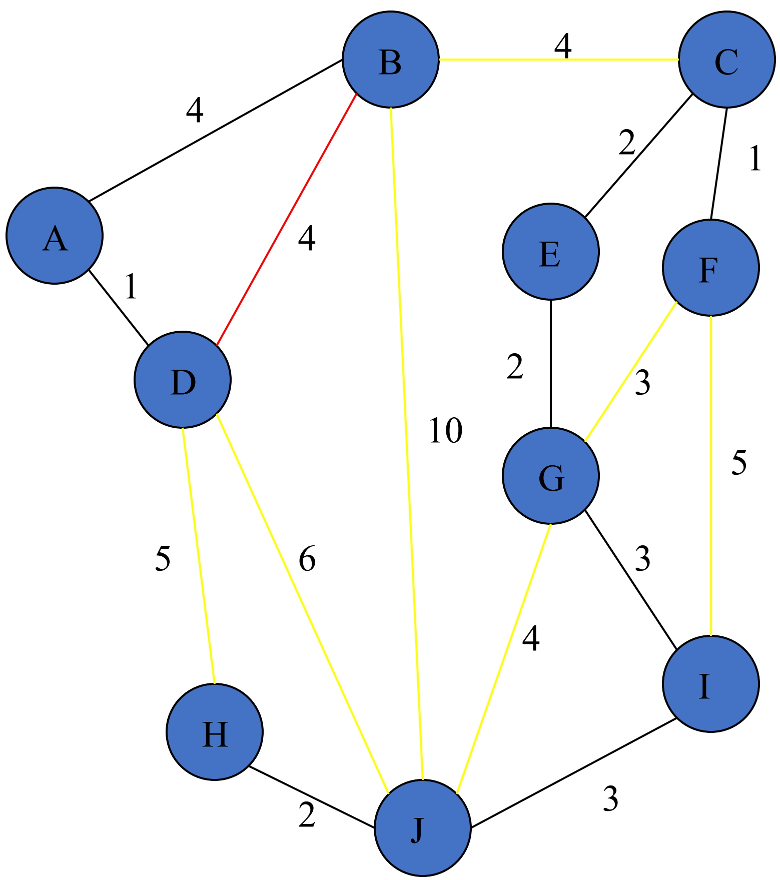
Add Edge

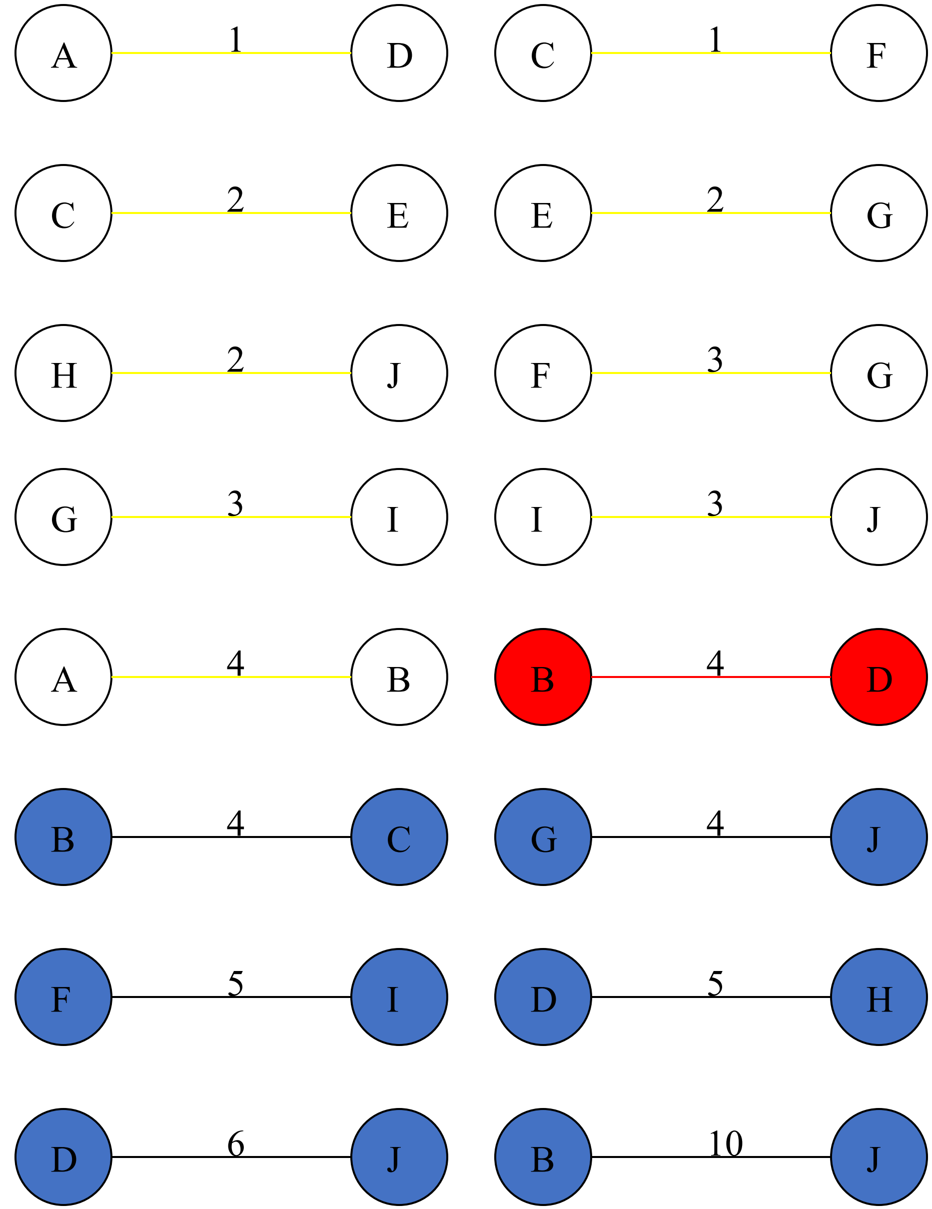

B
4
C
4
2
1
A
4
E
F
1
D
2
3
10
G
5
5
6
3
4
I
H
2
J
3


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

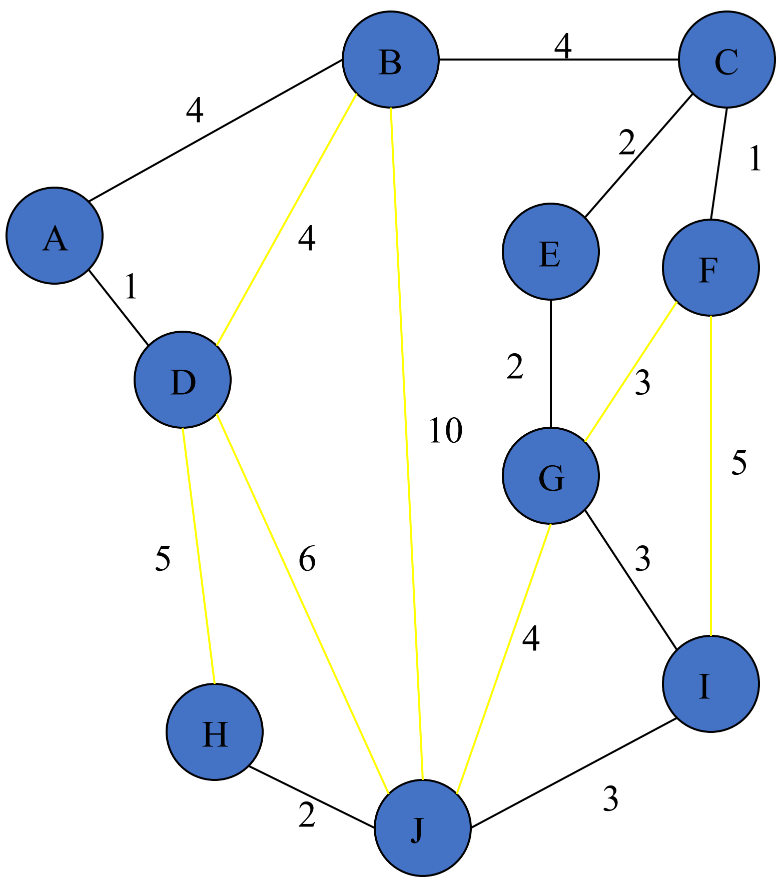
Cycle

Don’t Add Edge

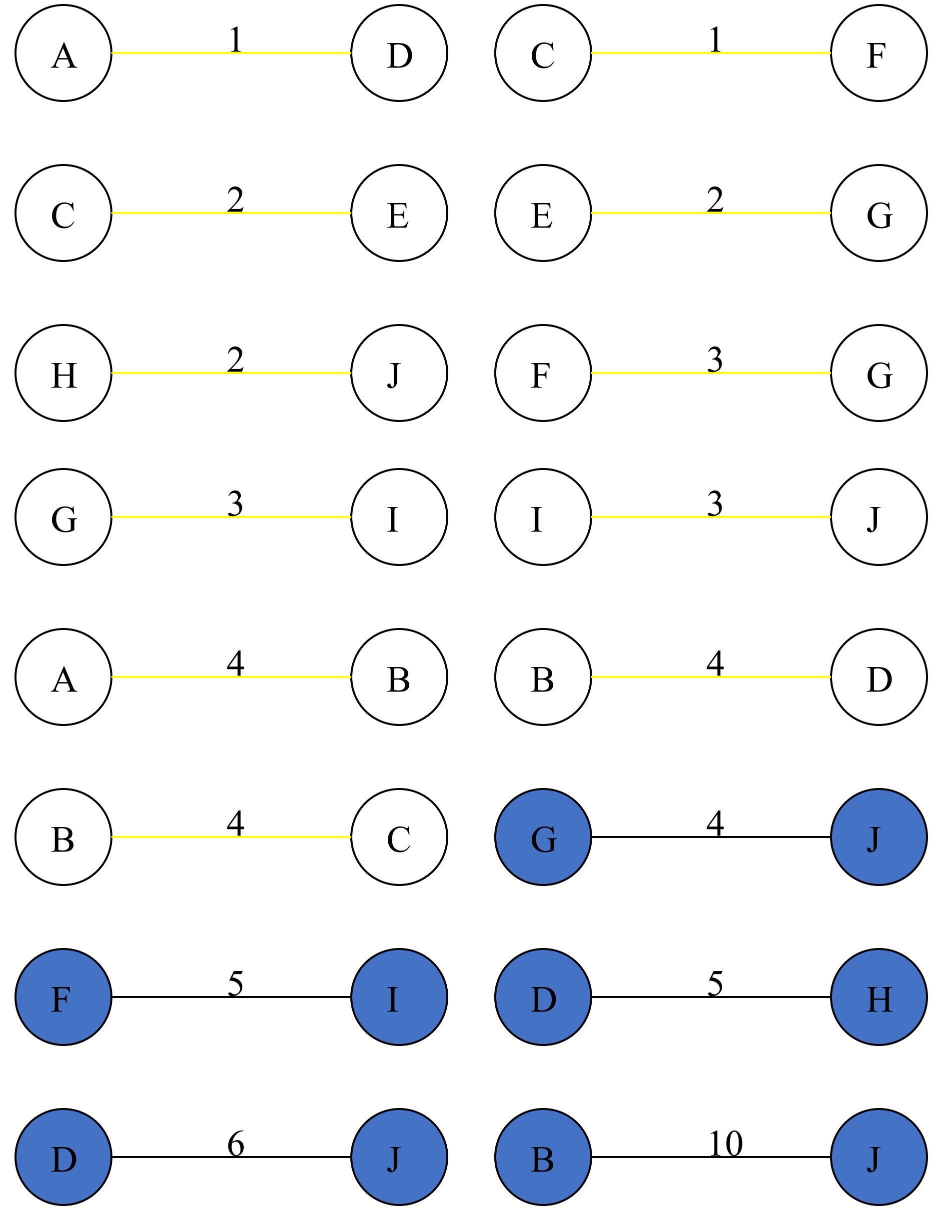




Add Edge

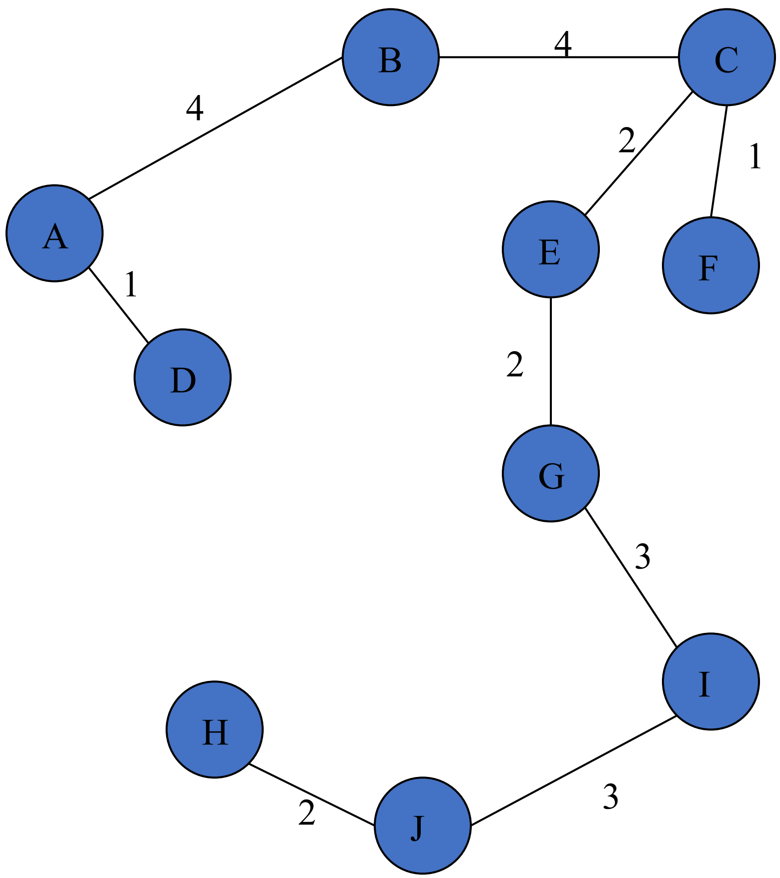


B
4
C
4
2
1
A
4
E
F
1
D
2
3
10
G
5
5
6
3
4
I
H
2
J
3



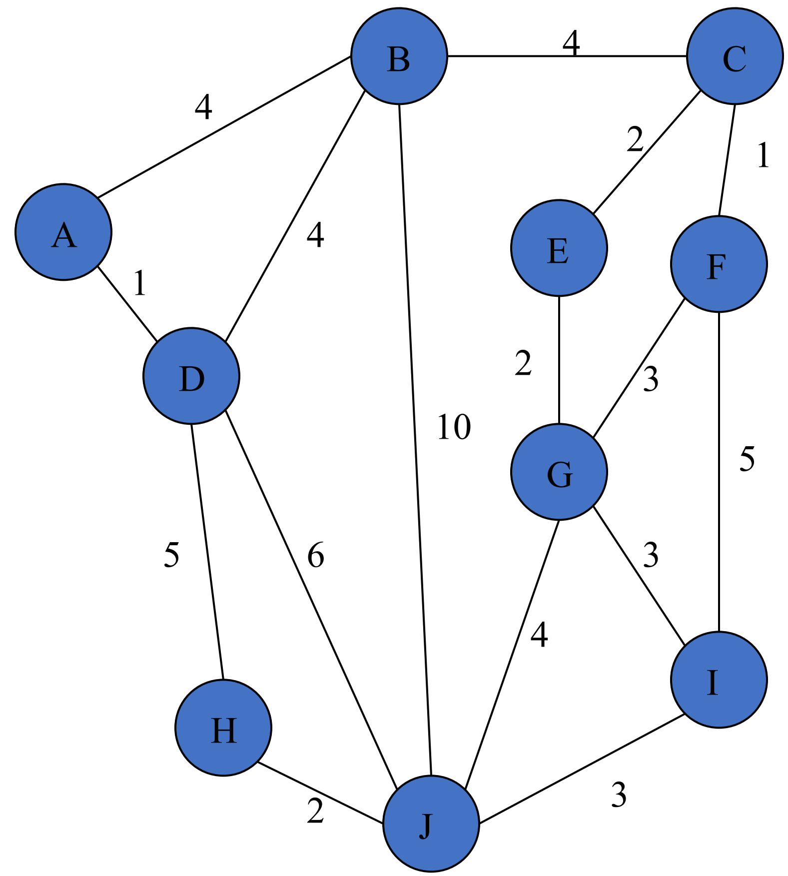
A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Minimum Spanning Tree (10 nodes, 9 edges)



Sum of the weights is the smallest = 22

Complete Graph



Kruskal's Algorithm

Each node is a set by itself

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
  MAKE-SET(v)
foreach (u, v) ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A ∪ {(u, v)}
      UNION(u, v)
return A
```

Get the smallest edge at each time

If the two nodes are in different sets (trees), then this edge will not form a cycle

Add this edge and union the two sets (merge the two trees)

Analysis of Kruskal's Algorithm: Naive

O(V)

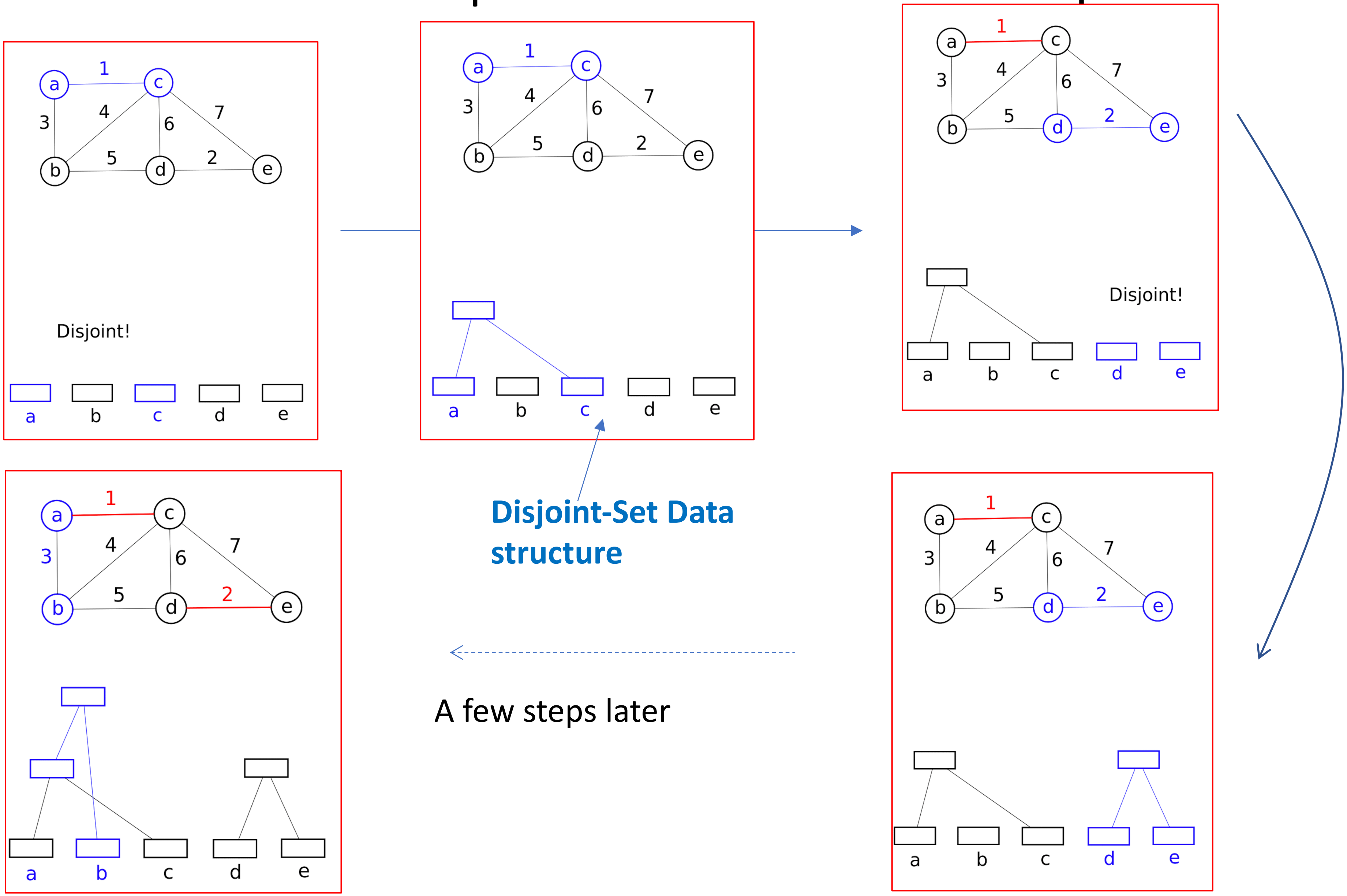
Need to sort the edges first **O(E log E)**

```
KRUSKAL(G):
1 A = ∅
2 foreach v ∈ G.V:
3   MAKE-SET(v)
4 foreach (u, v) ordered by weight(u, v), increasing:
5    if FIND-SET(u) ≠ FIND-SET(v):
6       A = A ∪ {(u, v)}
7       UNION(u, v)
8 return A
```

--Naïve implementation needs O(V) each time
--Will be called E times (for each edge) ➔ **O(EV)**

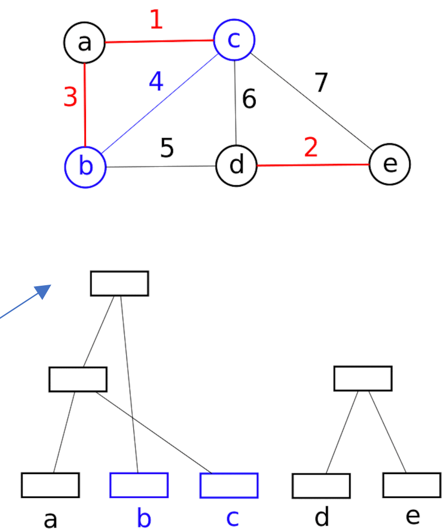
Naïve implementation ➔ O(V) + O(E log E) + O(E V) ➔ O(E V)

FIND-SET Implementation Example

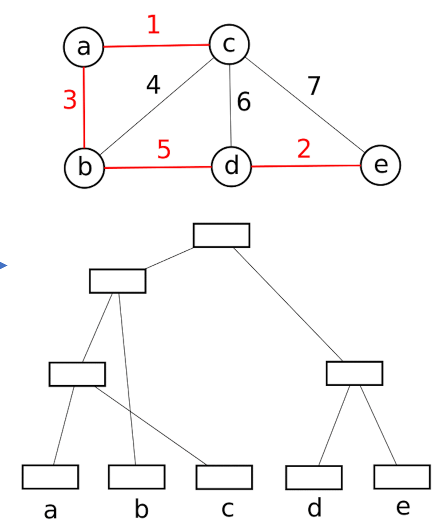


FIND-SET Example

- When two nodes in the same tree, then the are in the same set
  - Edge b-c cannot be part of the MST



- Final **disjoint set data structure** for the Minimum spanning Tree in this example



Analysis of Kruskal's Algorithm: Efficient

O(V)

Put the edges in a minHeap **O(E log E)**
**** Can be done in linear time O(E)**

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
  MAKE-SET(v)
foreach (u, v) ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A ∪ {(u, v)}
      UNION(u, v)
return A
```

--Can be done with some tricks in O (log V)
--Will be called E times (at worst) ➔ **O(E logV)**

Naïve implementation ➔ O(V) + O(E log E) + O(E log V)
Note, O(E log E) < O(E log V) **Since E < V^2 , then Log E < 2 Log V**

ACTUAL COMPLEXITY = O(E Log V)

Algorithms for Obtaining the Minimum Spanning Tree

- Kruskal's Algorithm
- Prim's Algorithm

We will talk about it after the break!


That's all Folks!
Any Question?