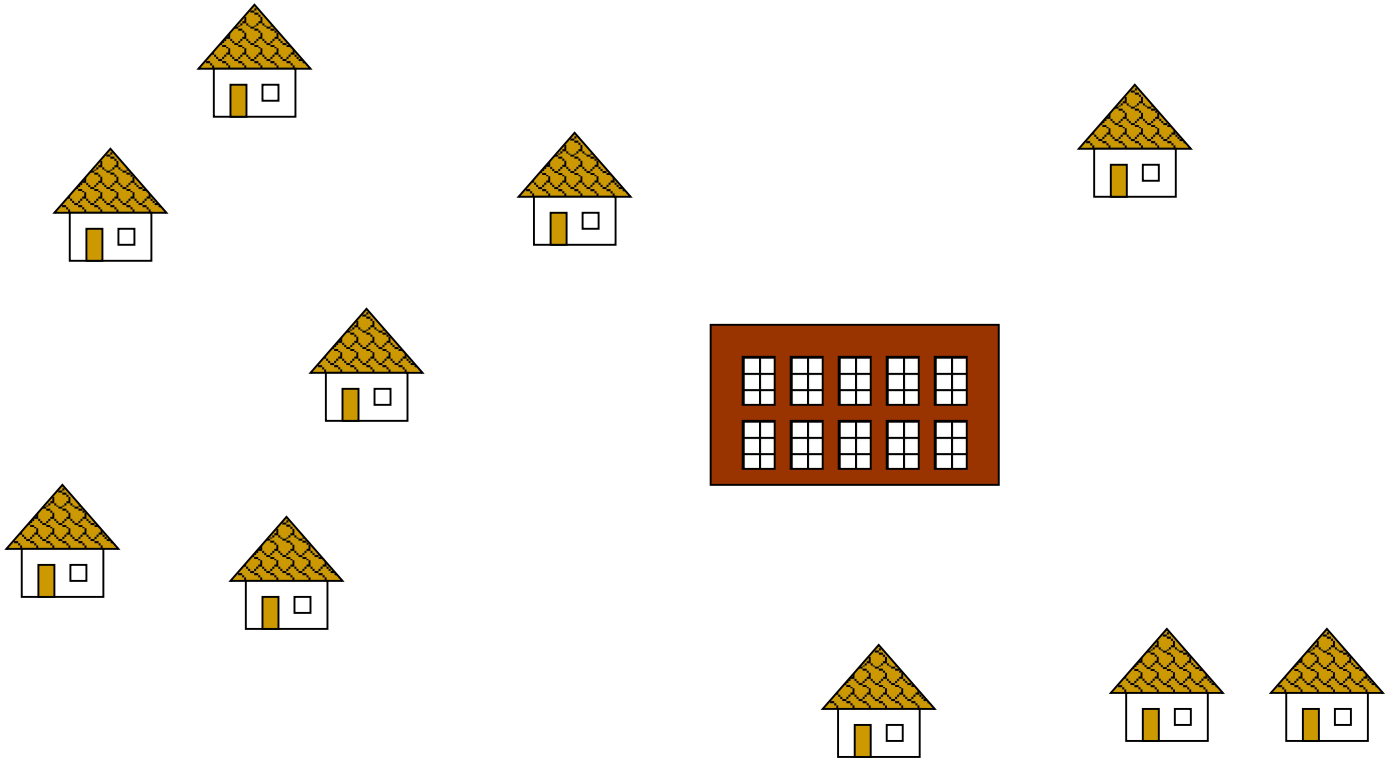
Week 3: Minimum Spanning Tree (MST)

- Definition of MST
- Generic MST algorithm
- Kruskal's algorithm
- Prim's algorithm

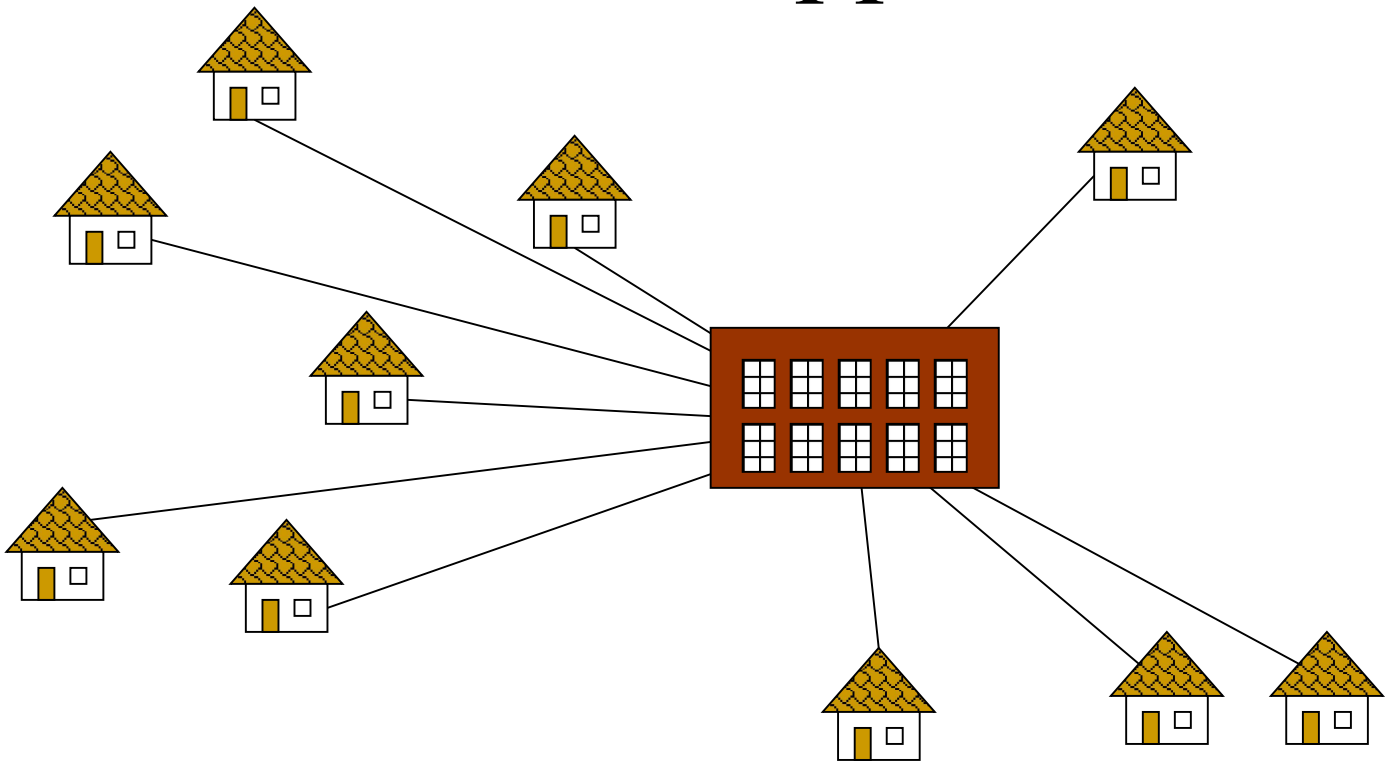
- Graph
  - Subgraph
- Tree: 1) connected, and

  2) no cycle/unique path between two nodes/the no of edges=the no of nodes -1.
- Forest:
  - a disjoint union of trees/a graph without cycles.

Problem: Rail Network

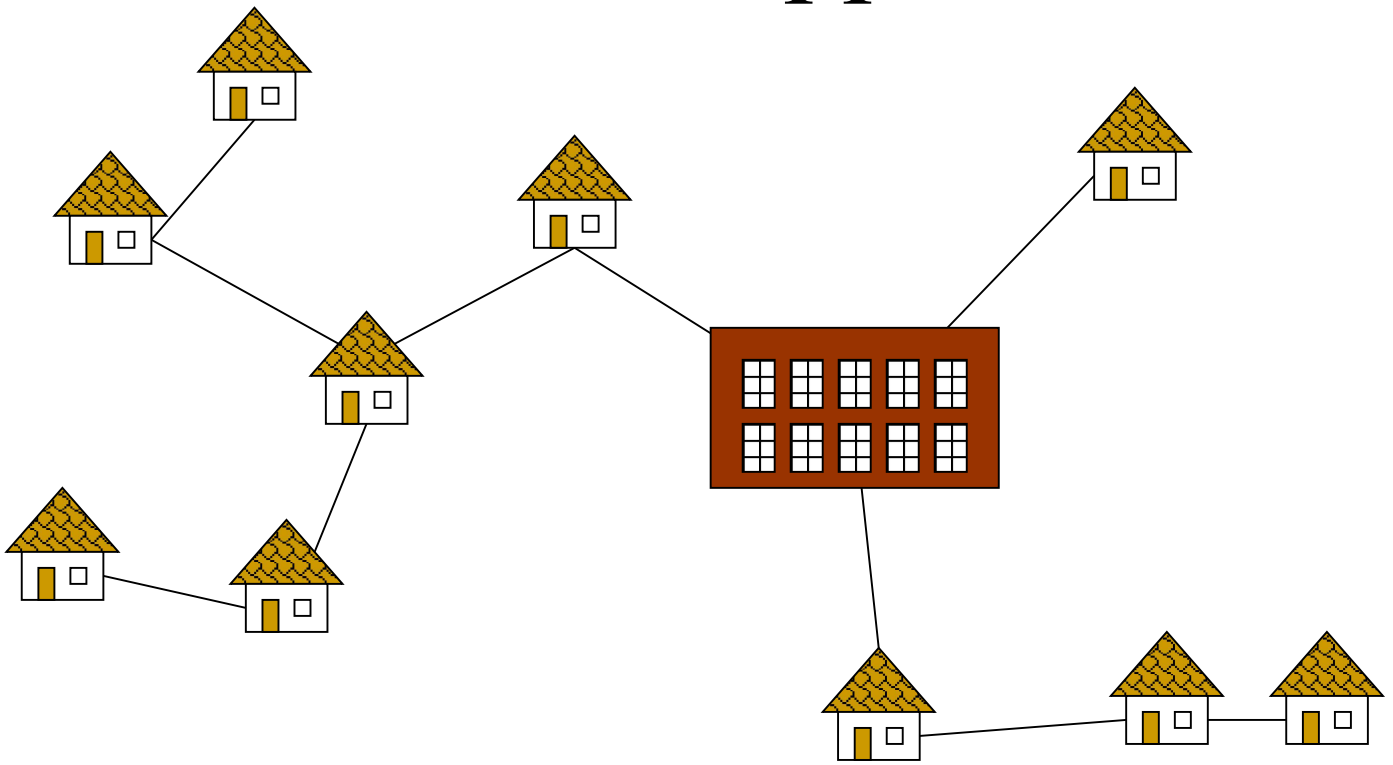


Naïve Approach



Expensive!

Better Approach



Minimize the total length of the railway connecting all the towns

Definition of MST

Spanning tree: Given a undirected and connected graph G=(V, E), a subgraph T of G is a spanning tree of G if

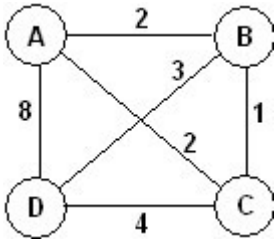
1. T is a tree and
2. T contains all the vertices (nodes) of G.

Given a connected and undirected graph *G=(V,E),* where each edge in E has a (nonnegative) weight (cost, or length),

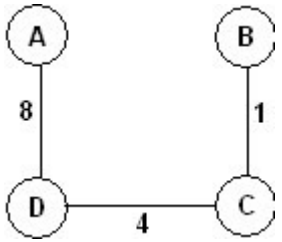
a minimum spanning tree (MST) is a spanning tree with minimum total weight (cost, or length).

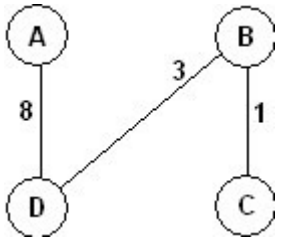
Minimum Spanning Tree: Example

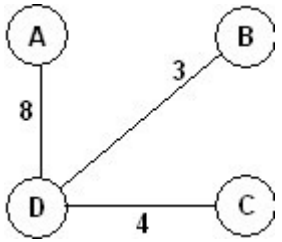
- Example: The graph

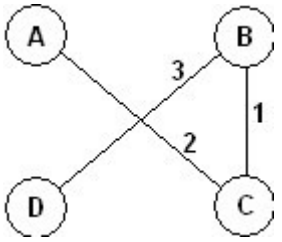


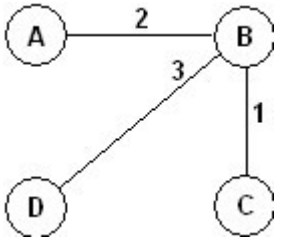
Has 16 spanning trees. Some are:



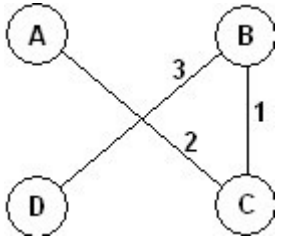


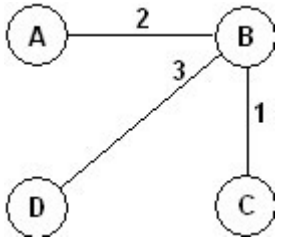






The graph has two minimum spanning trees, each with a length of 6:





Growing a MST(Generic Algorithm)

```
GENERIC_MST(G, w)
1       A:={}
2       while A does not form a spanning tree do
3               find an edge (u,v) that is safe for A
4               A:=A ∪ {(u,v)}
5       return A
```

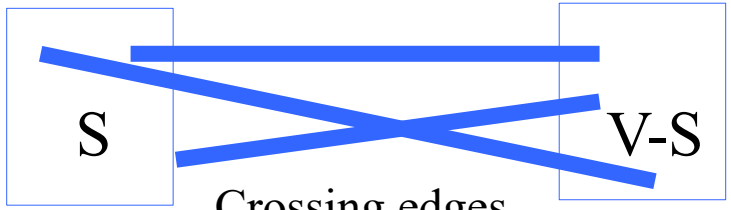
- Invariant: Set A is always a subset of some minimum spanning tree.
  - This property is called the invariant property.
- An edge (u, v) is a *safe edge for A* if adding it to A does not destroy the invariant.

Basically, GENERIC_MST() says that "select the correct *edges one by one until A becomes a minimum spanning tree.*

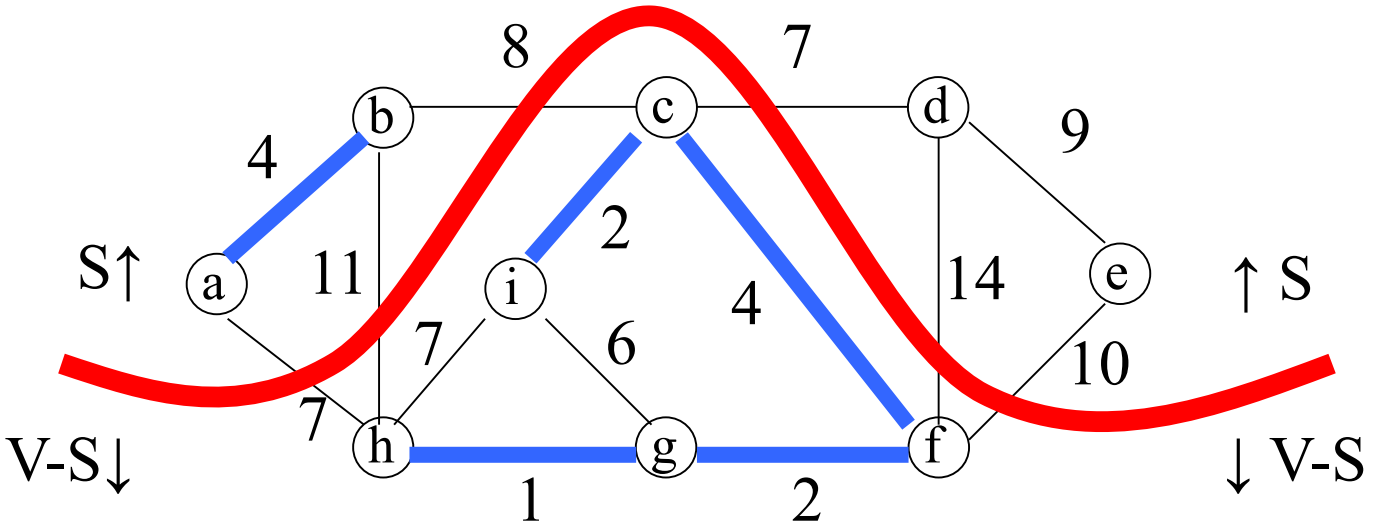
Safe edge

We need some definitions and a theorem.

- A cut (S,V-S) of an undirected graph G=(V,E) is a partition of V, where S is a subset of V.
- An edge crosses the cut (S,V-S) if one of its endpoints is in S and the other is in V-S.
- An edge is a light edge crossing a cut if its length is the **shortest** among all the edges crossing the cut.



Crossing edges



• This figure shows a cut (S,V-S) of the graph.

S={a, b, d, e}, V-S={h, i, g, c, f}

• The edge (d,c) and (a, h) are two light edges crossing the cut.

Theorem

If

- G=(V, E) is a connected and undirected graph with a nonnegative real-valued length function w defined on E.
- A is a subset of E that is included in a minimum spanning tree (MST) for G.
- (S,V-S) is a cut of G such that no edge in A crosses the cut.
- (x, y) is a light edge crossing (S,V-S).

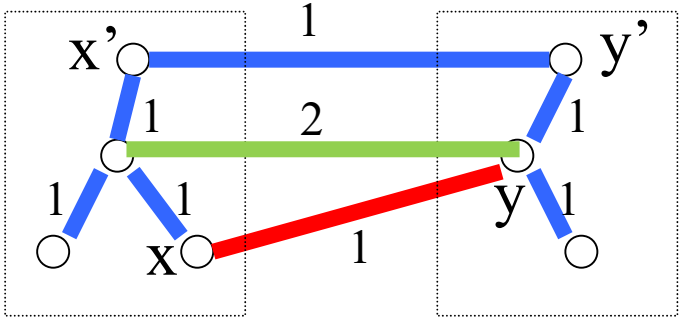
Then, edge (x, y) is safe for A (i.e. A∪ (x,y) is part of a MST too)

Outline of Pf: Let T_{opt} be a MST (blue) that includes A.

Case 1: (x,y) is in T_{opt}, then **A∪ (x,y) is in** T_{opt}, so (x,y) is safe.

Case 2: (x, y) is not in T_{opt} (as in the Figure, red edge is (x,y))

- There MUST be a path linking x and y in T_{opt}. Then there is another edge (x', y') (green) which crosses (S, V-S) and is in this path.
- We replace (x', y') in T_{opt} by (x, y) and get another tree T'_{opt}
- Since (x, y) is light , T'_{opt} is also optimal.
- **A∪ (x,y) is in** T'_{opt}, so (x,y) is safe.



Corollary

- Let G=(V,E) be a connected, undirected graph with a real-valued weight function w defined on E.
- Let A be a subset of E that is included in some minimum spanning tree for G. (A form a forest. See blue trees)
- Let C be a connected component (tree) in the forest G_A=(V,A).
- Let (u,v) be a light edge (shortest among all edges connecting C with other components) connecting C to some other component D in G_A.
- Then, edge (u,v) is safe for A. (For Kruskal's algorithm)

The algorithms of Kruskal and Prim

- The two algorithms are elaborations of the generic algorithm.
- They each use a specific rule to determine a safe edge in line 3 of GENERIC_MST.
- In Kruskal's algorithm,
  - The set A is a forest.
  - The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.
- In Prim's algorithm,
  - The set A forms a single tree.
  - The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

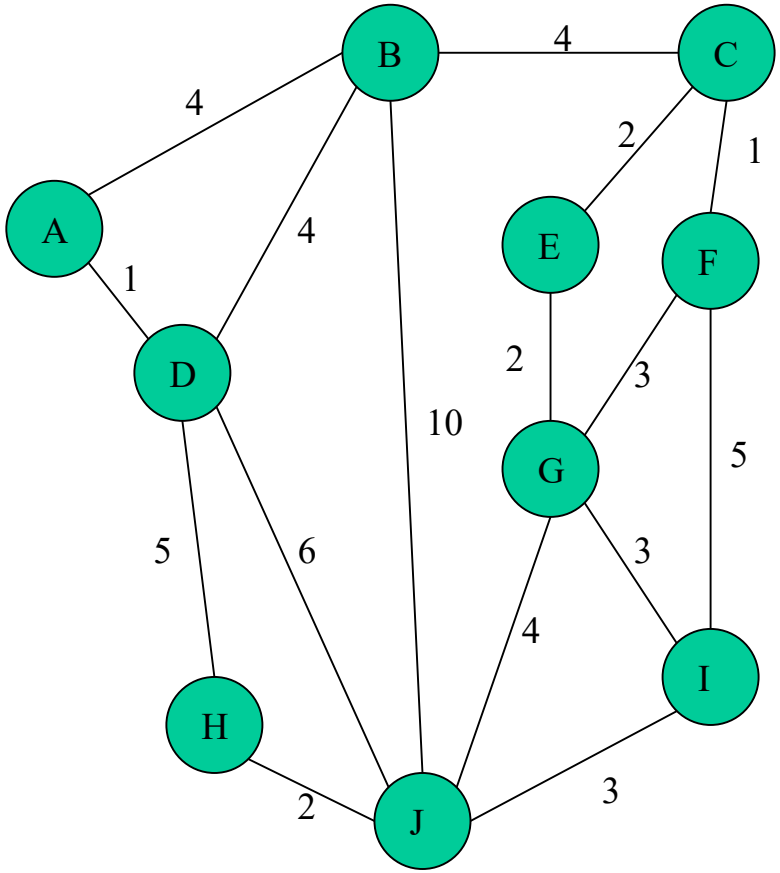
Kruskal's Algorithm

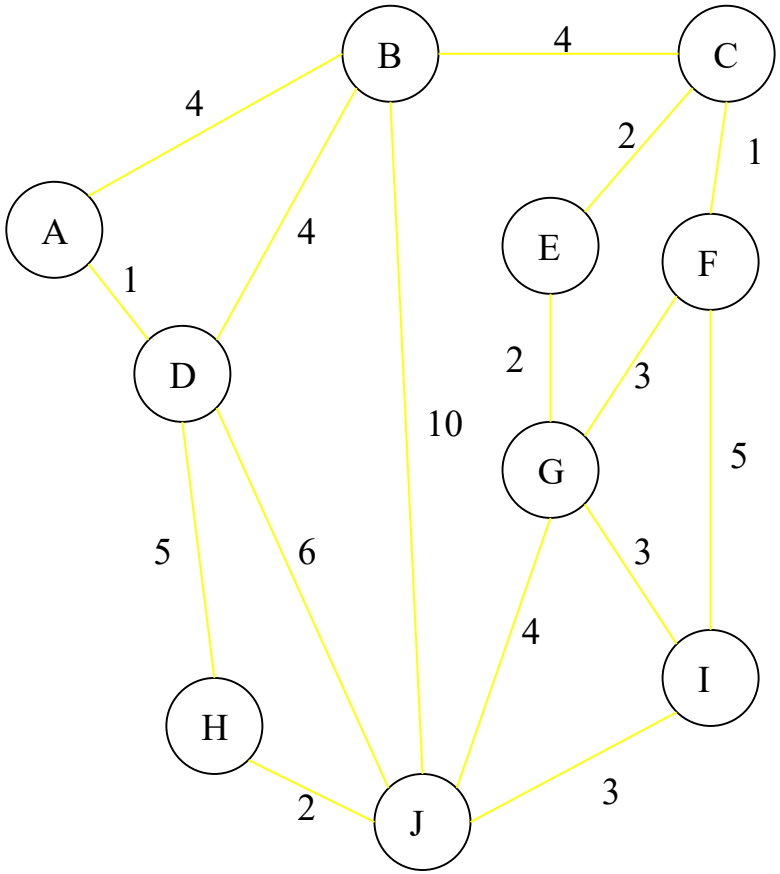
Kruskal's algorithm(basic part)

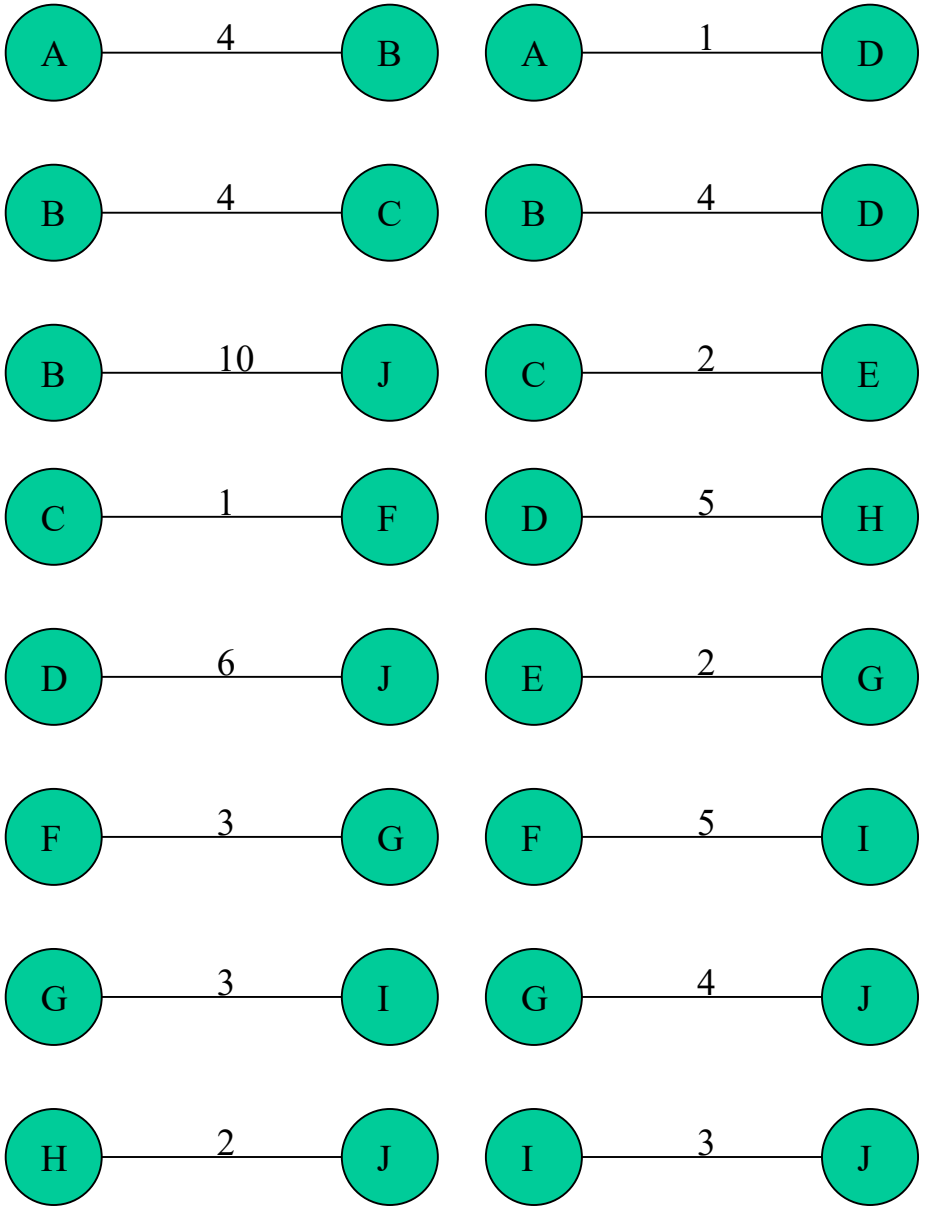
```
1    (Sort the edges in an increasing order)
2    A:={}
3    while E is not empty do {
3        take an edge (u, v) that is shortest in E
         and delete it from E
4        if  u and v are in different components then
                 add (u, v) to A
         }
```

Note: each time a shortest edge in E is considered.

Graph

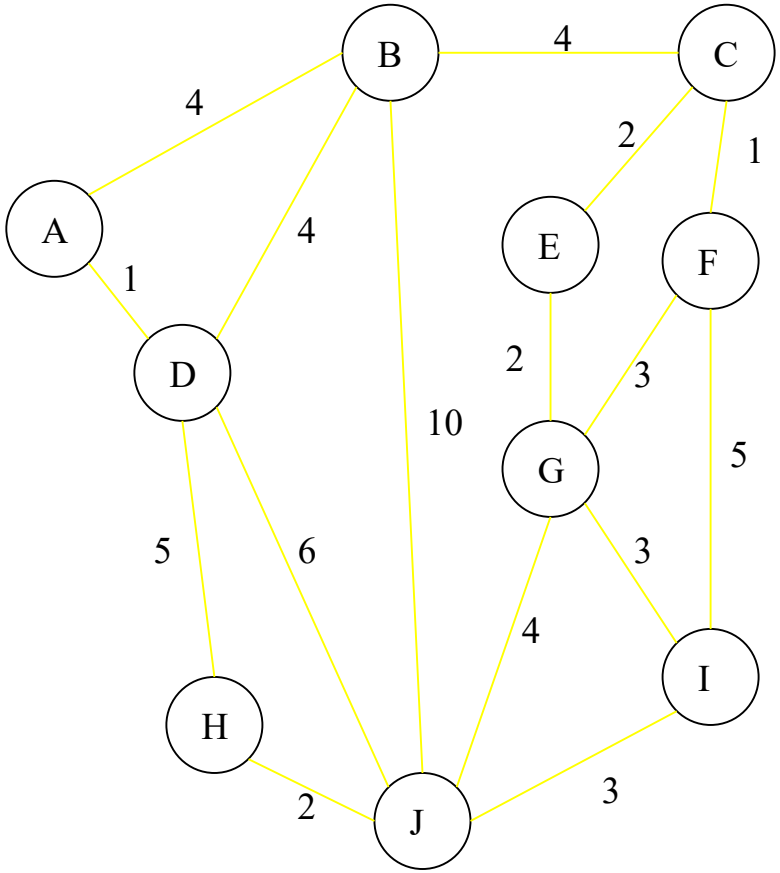


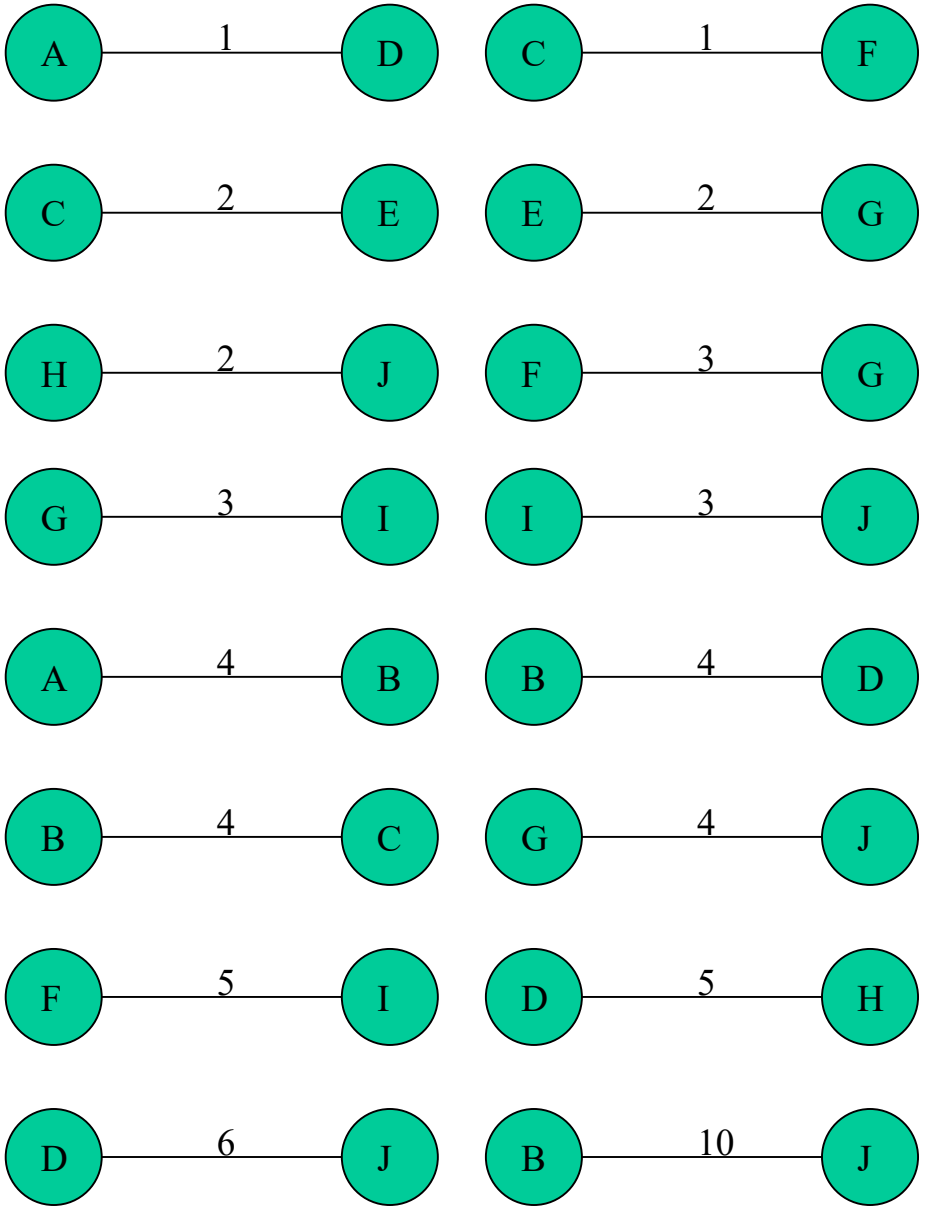

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3


A 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

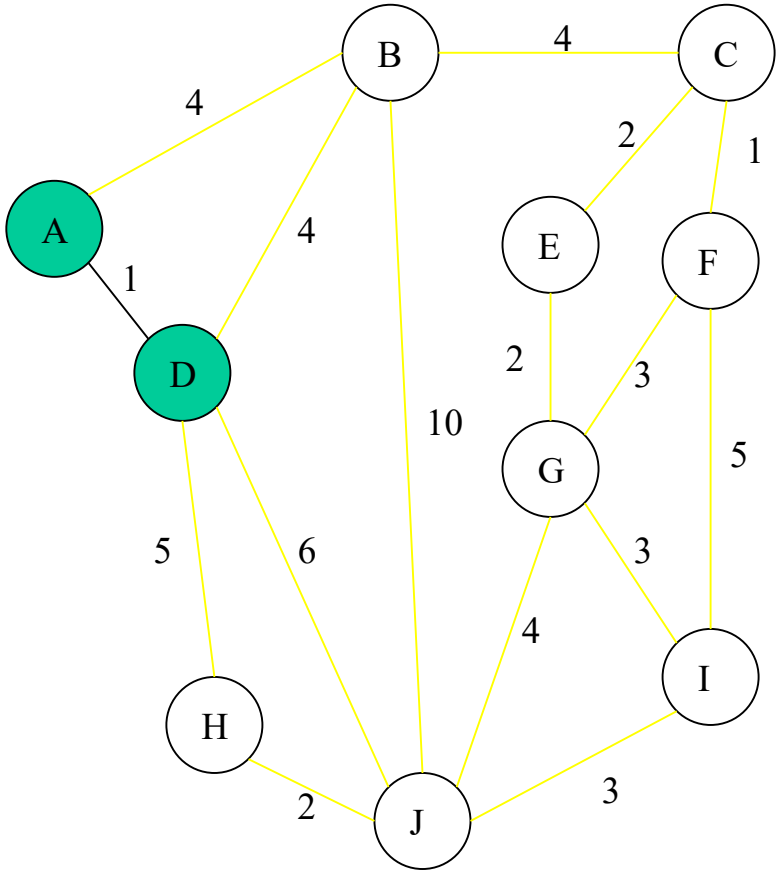
Sort Edges

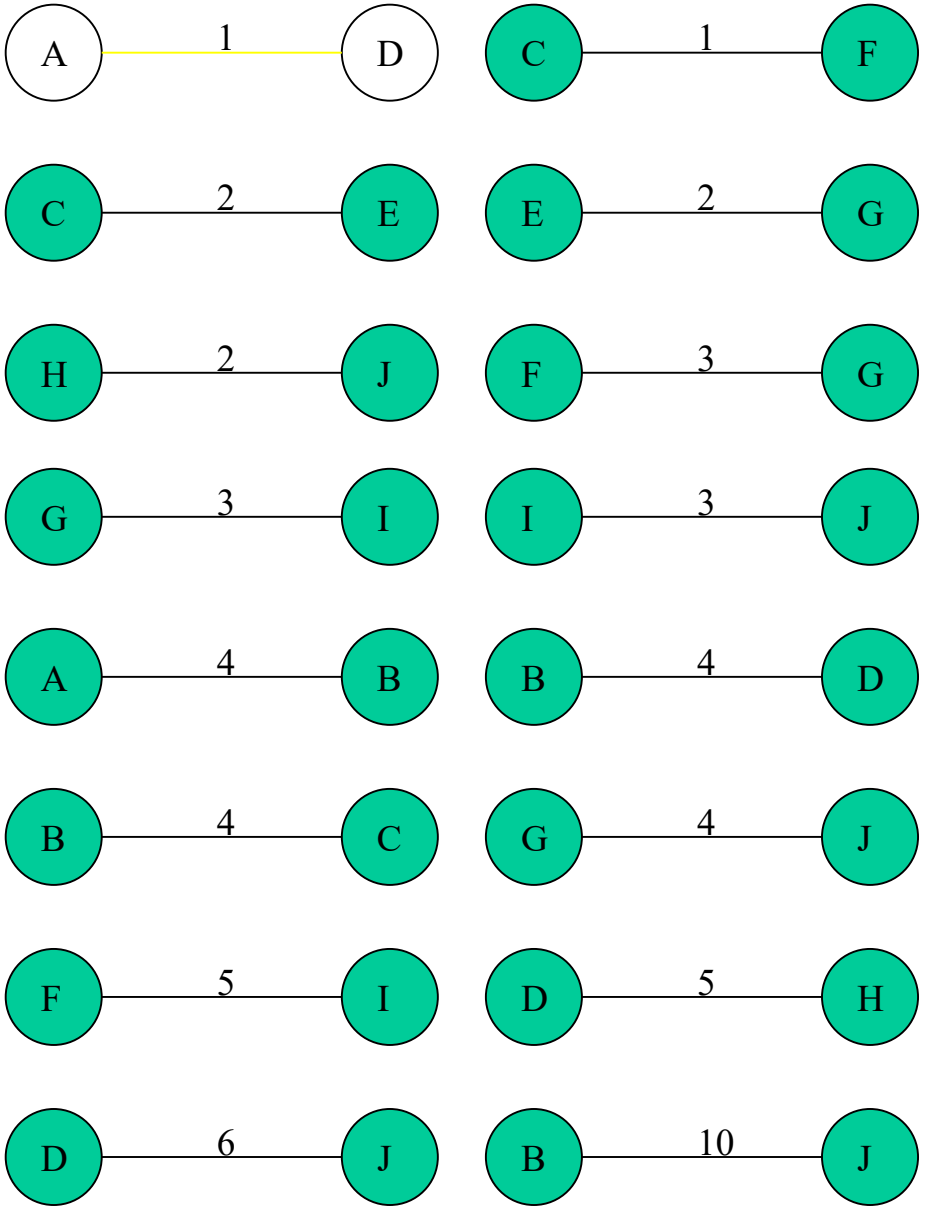
(in reality they are placed in a priority queue - not sorted - but sorting them makes the algorithm easier to visualize)



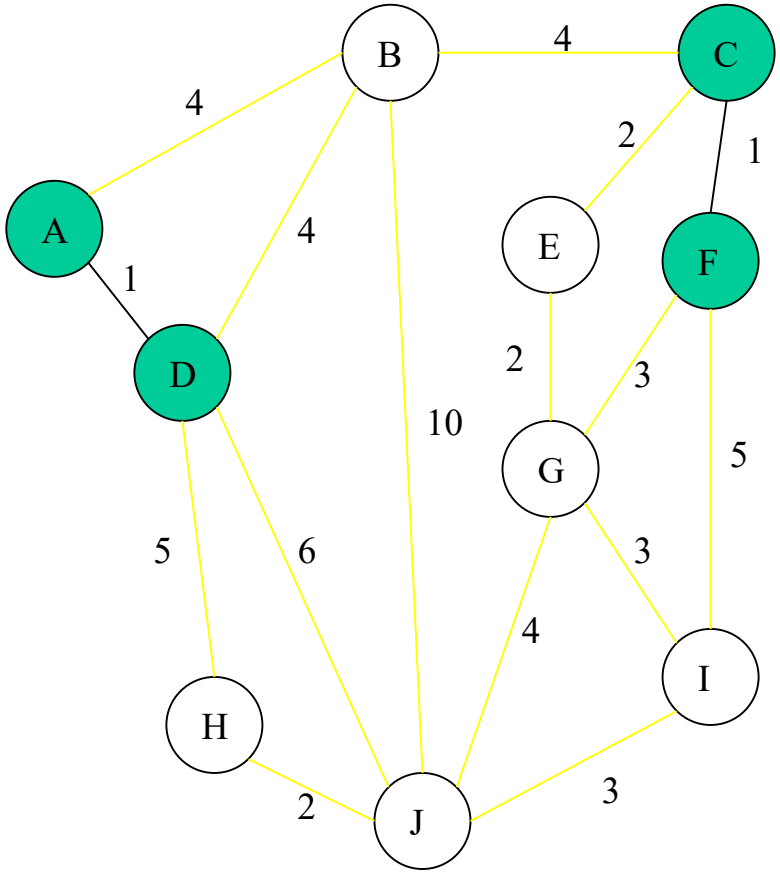


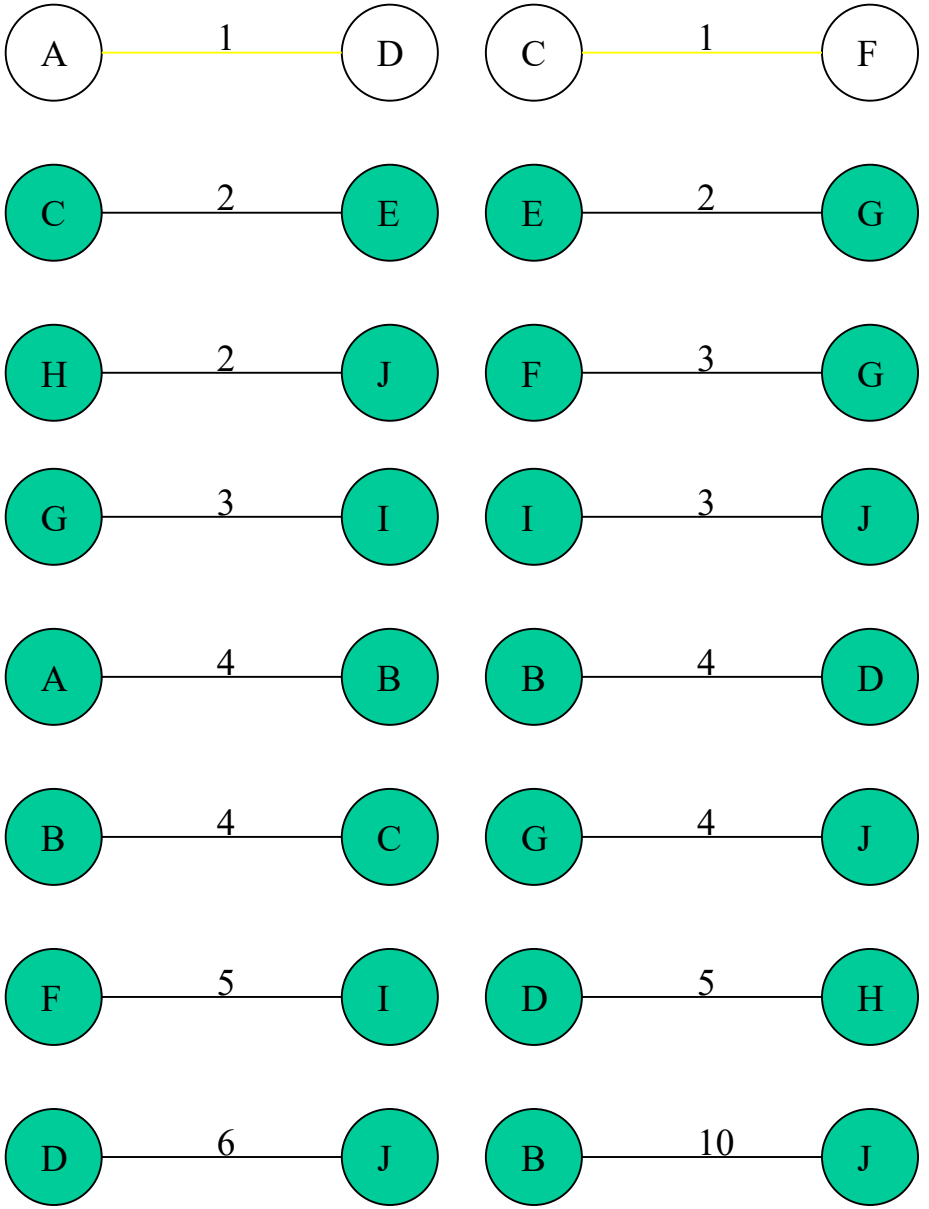
Add Edge


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

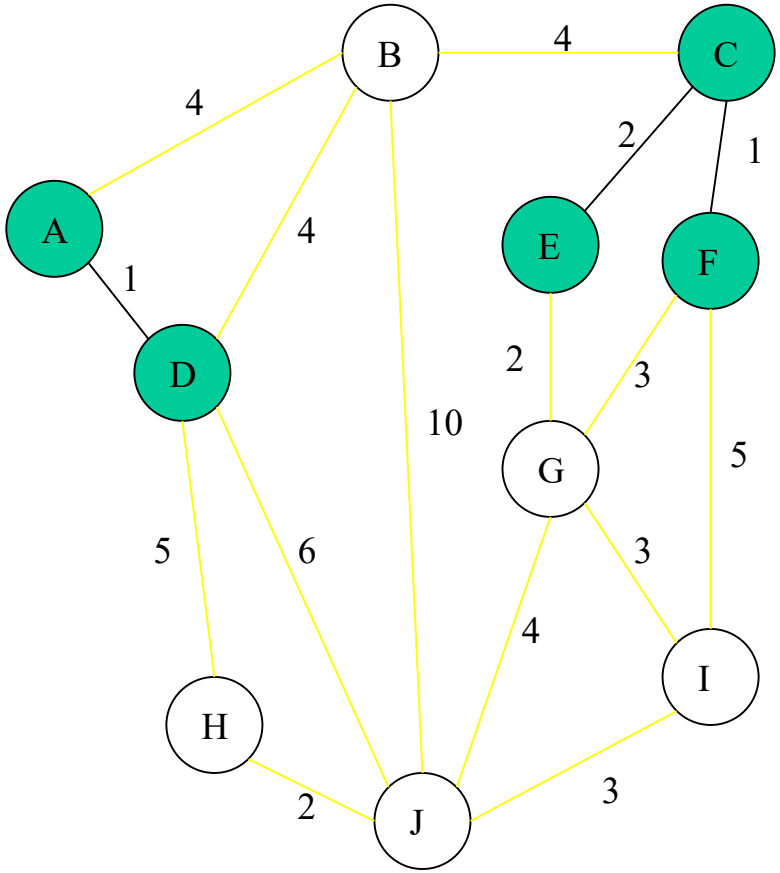

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

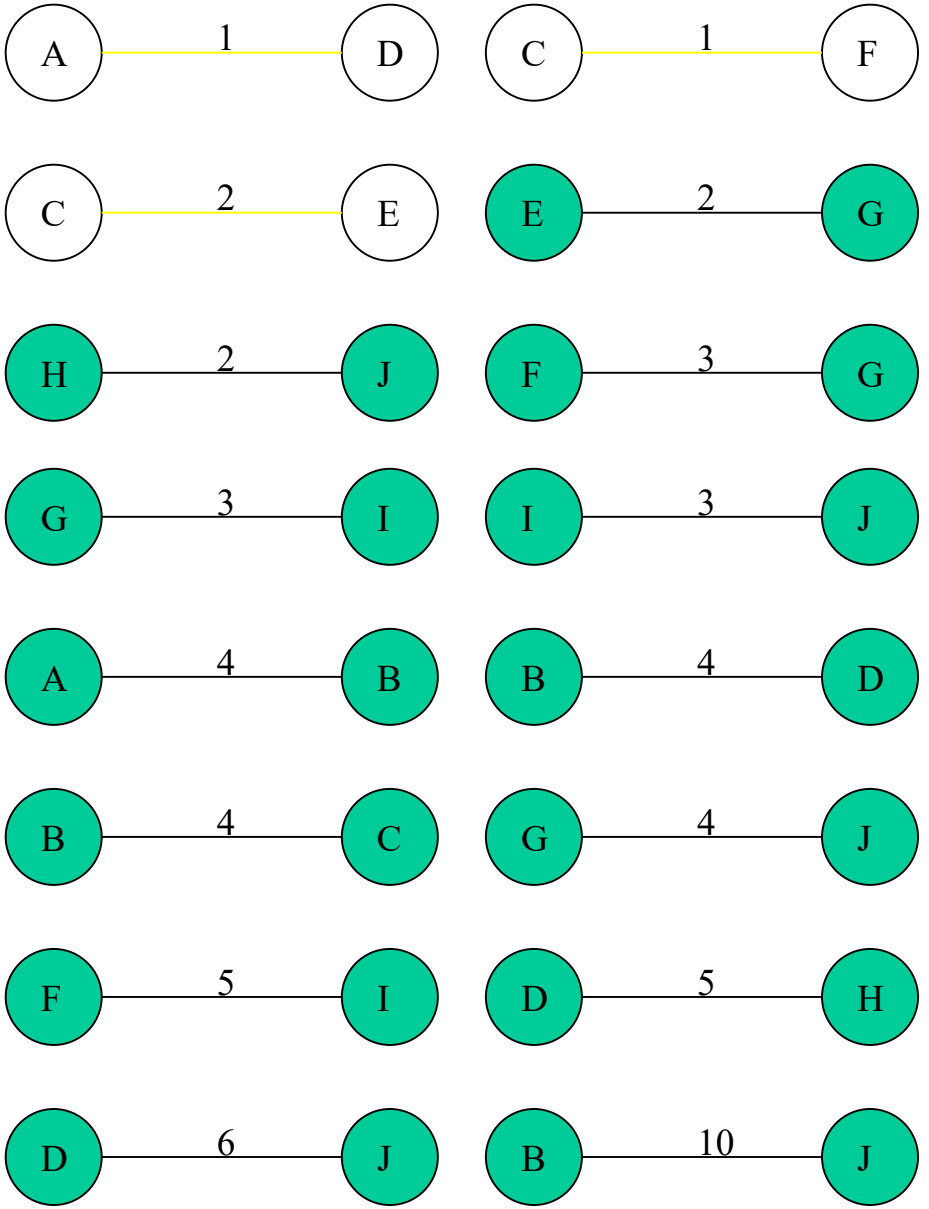
Add Edge


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

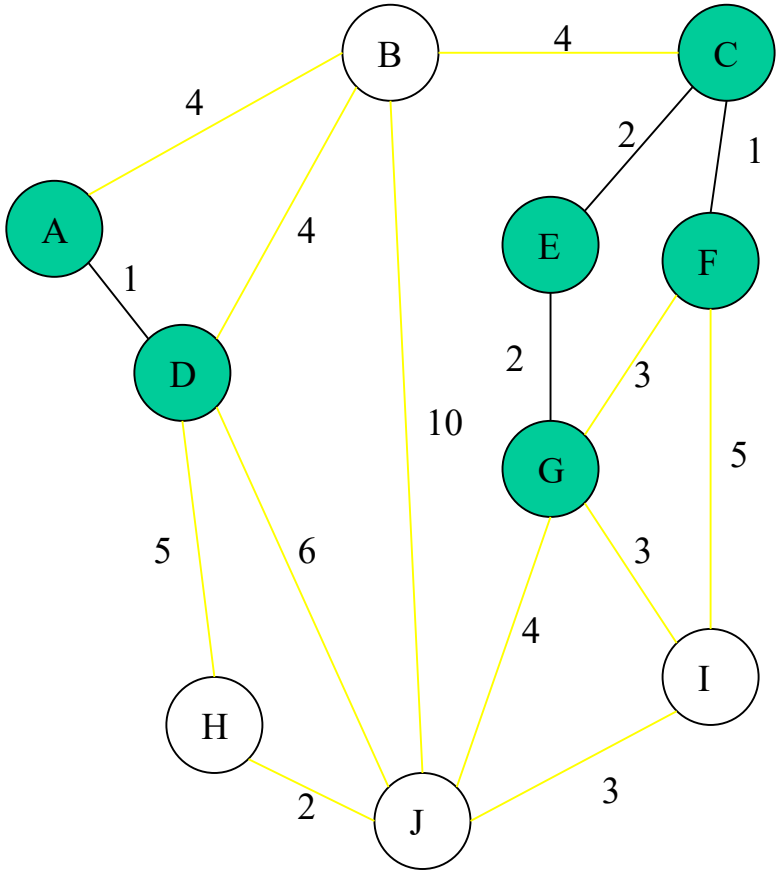

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Add Edge


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



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

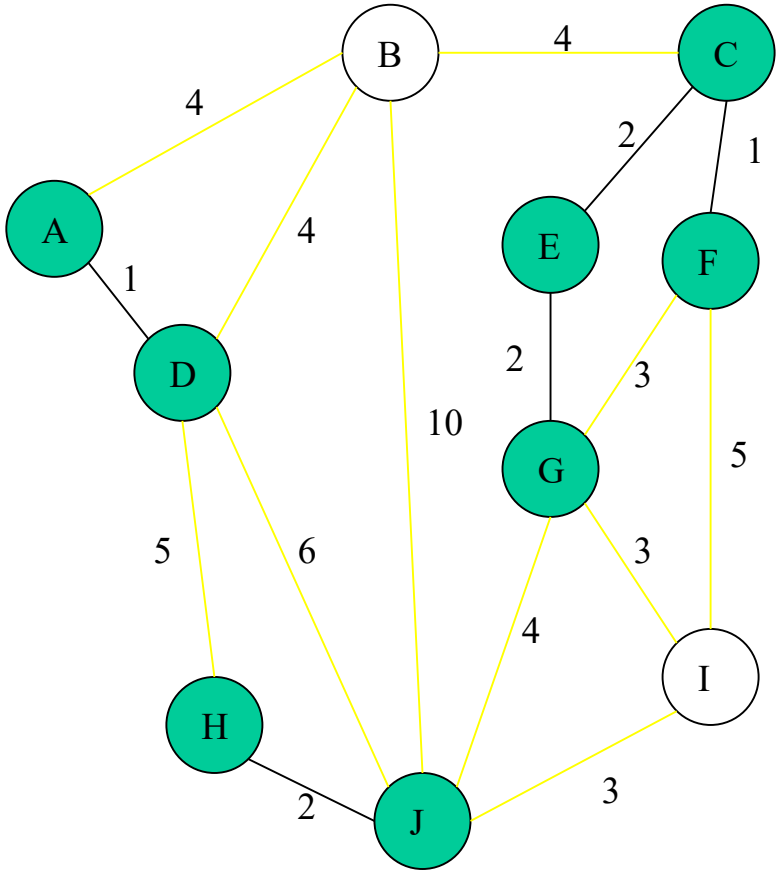
F 5 I

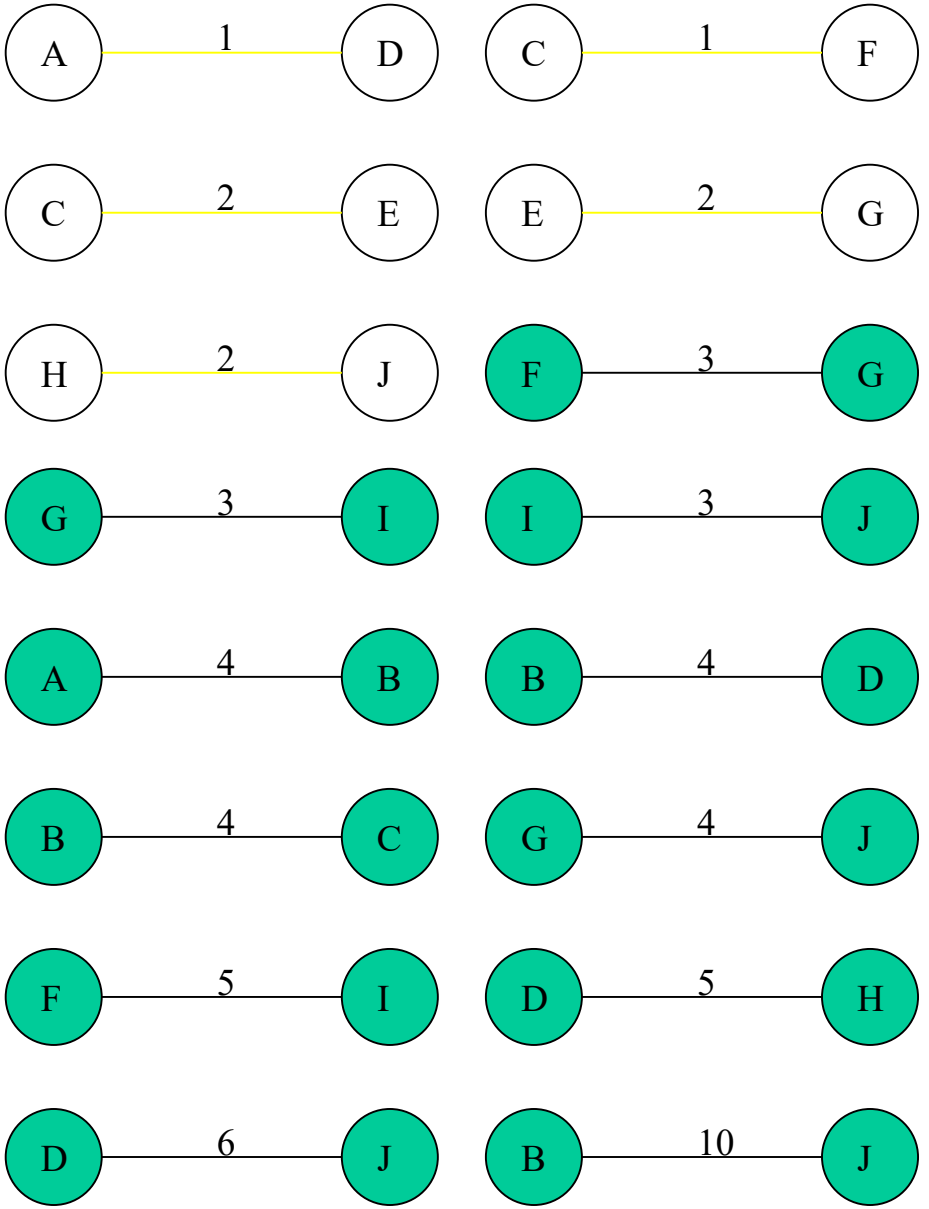
D 5 H

D 6 J

B 10 J

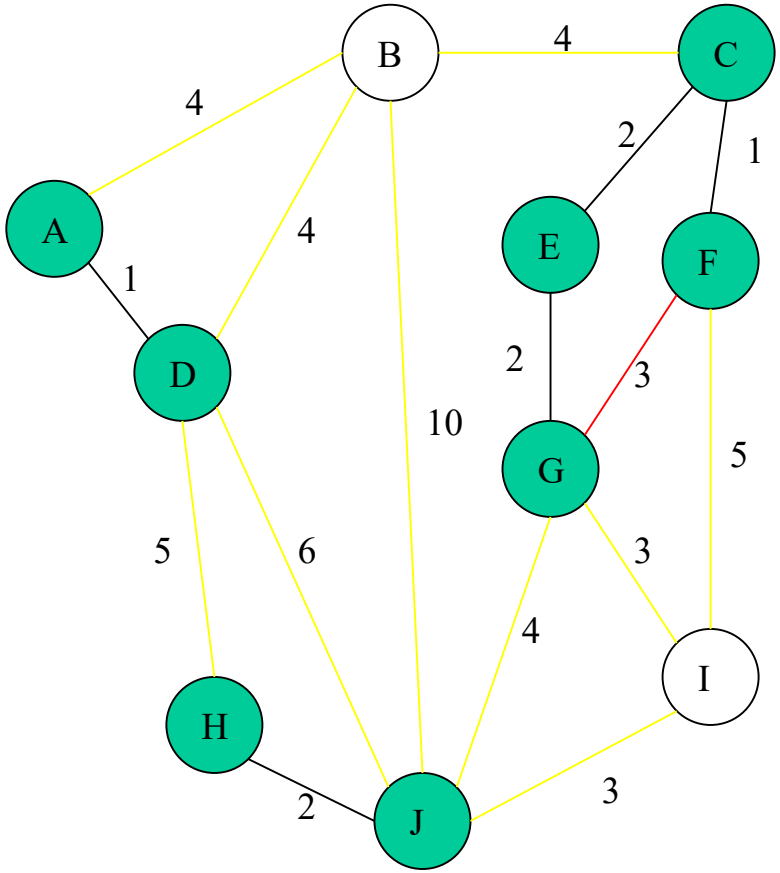
Add Edge

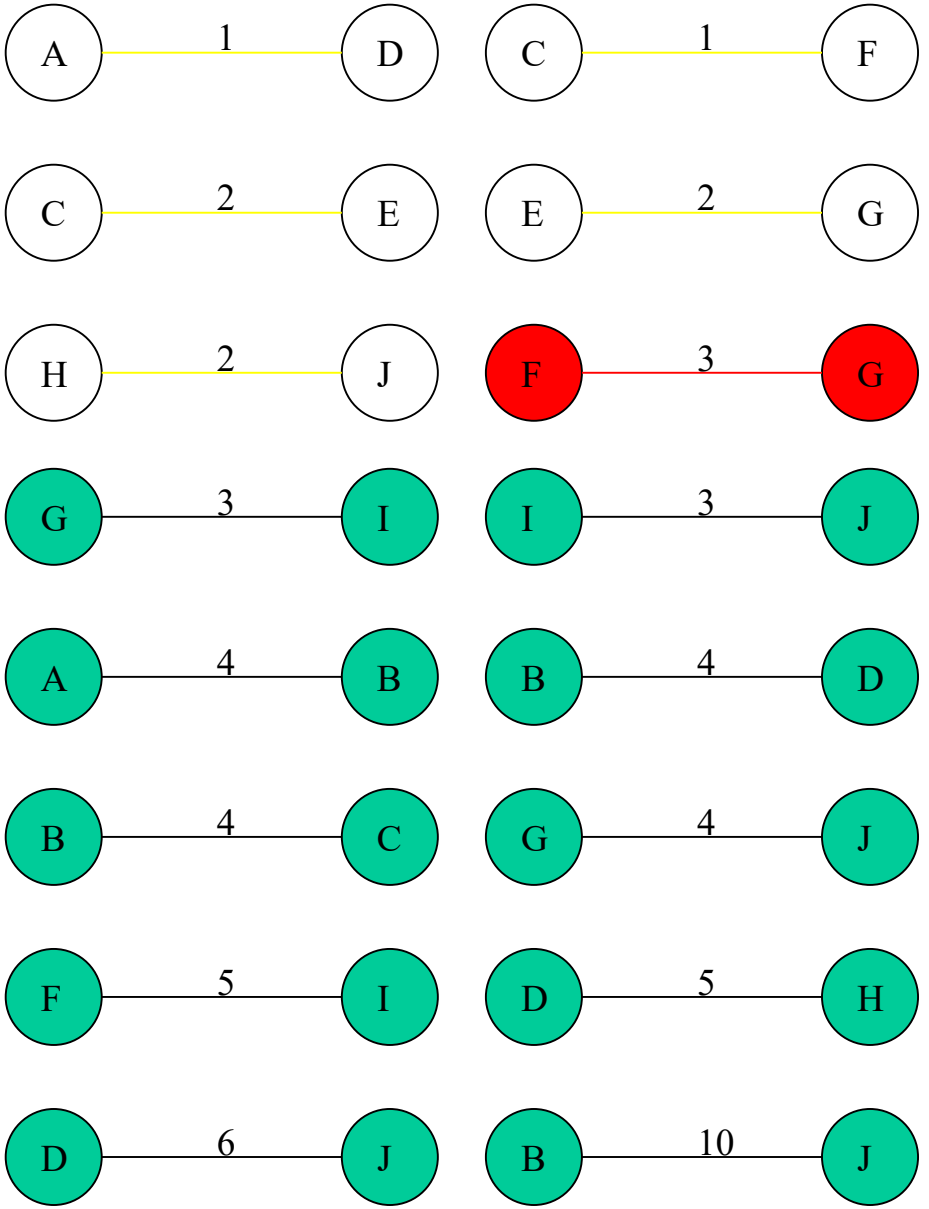

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3


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

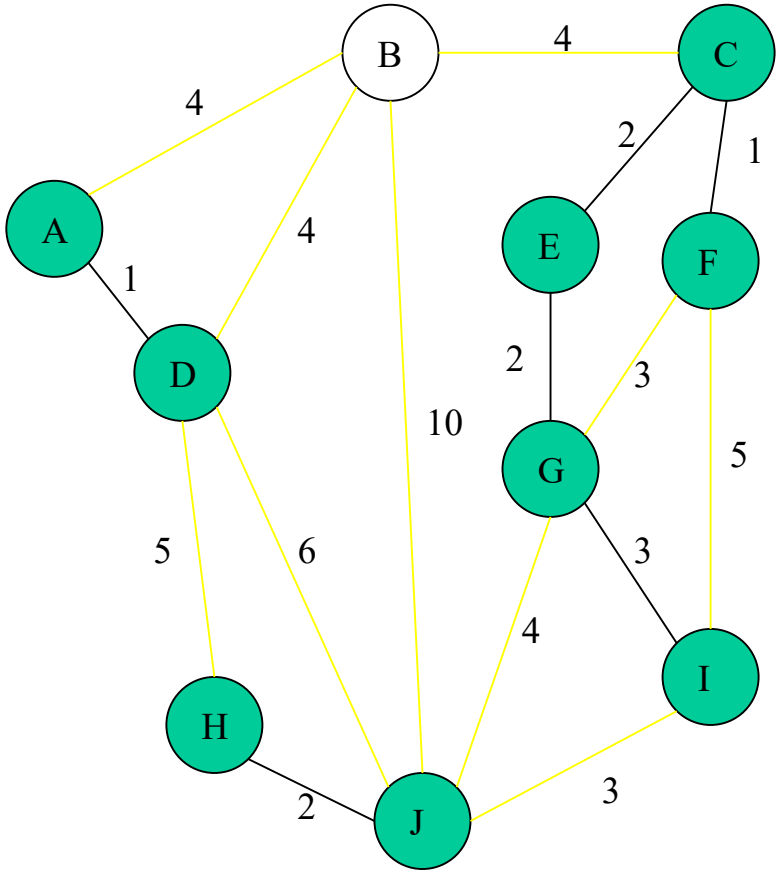
Cycle

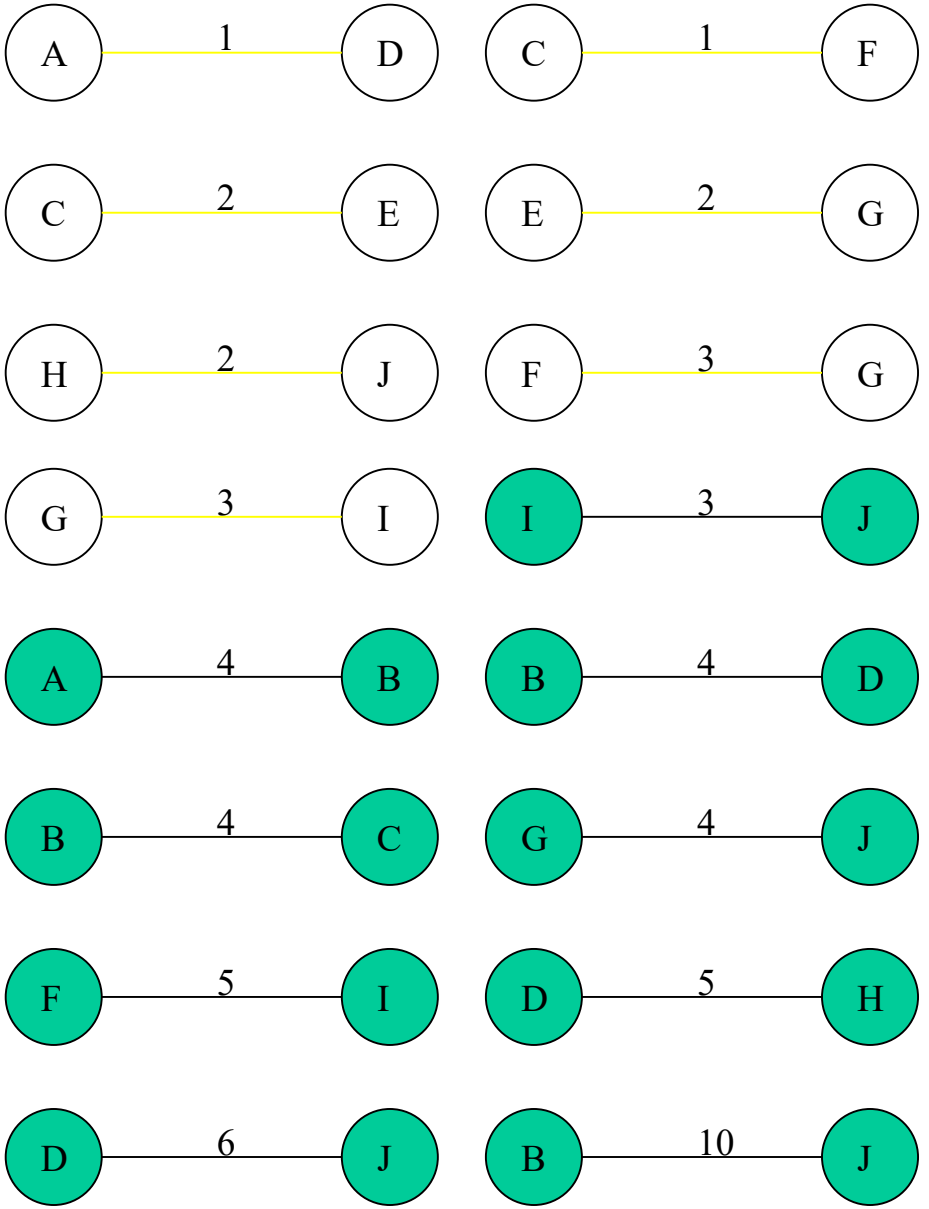
Don’t Add Edge



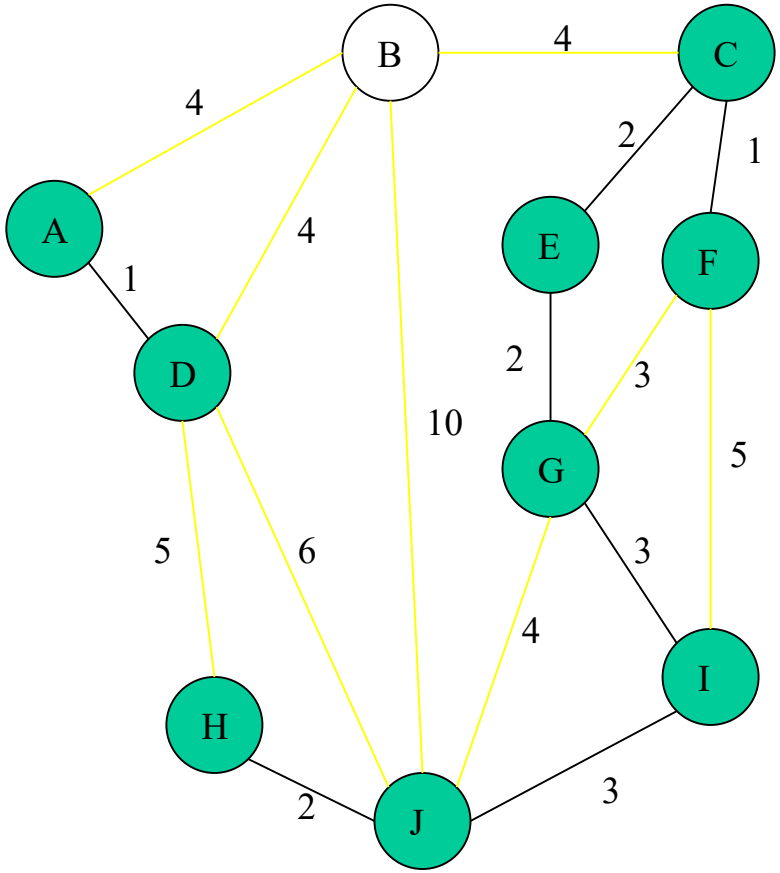


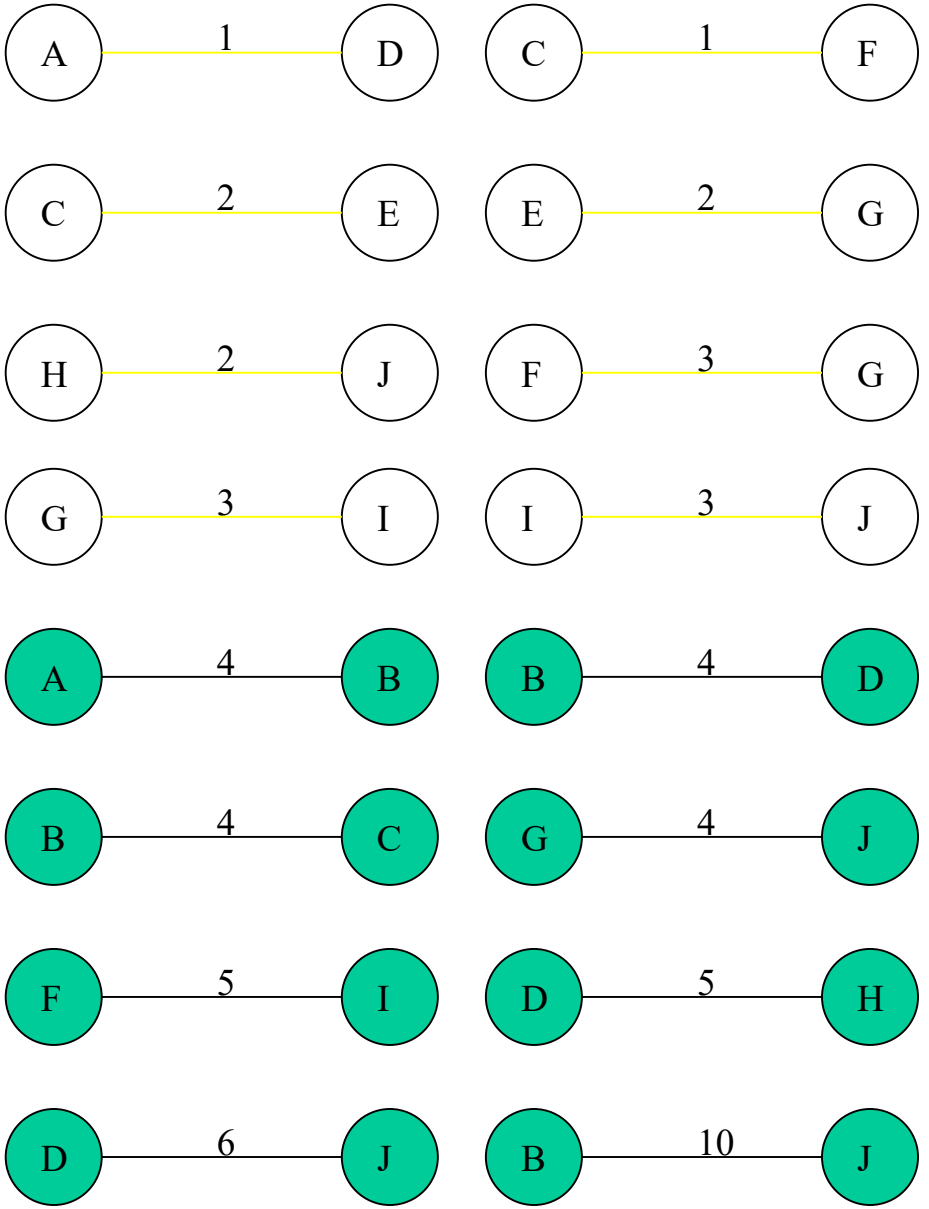
Add Edge


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

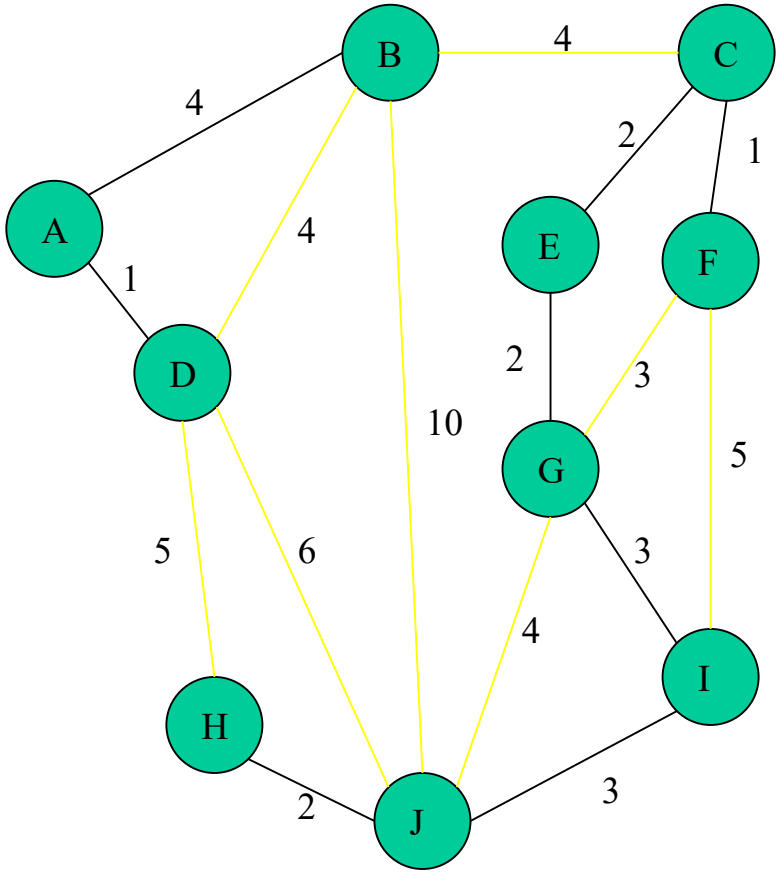

A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

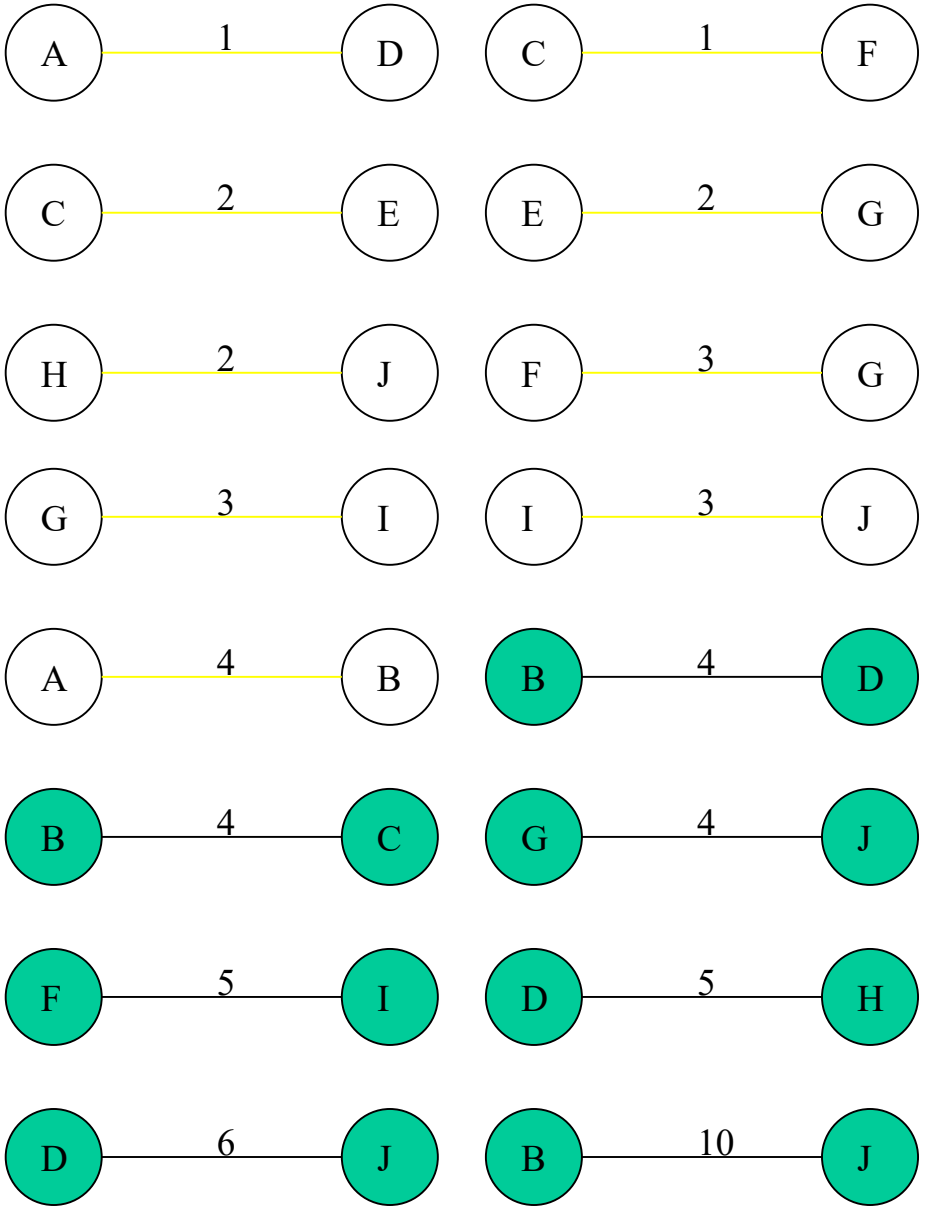
Add Edge


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

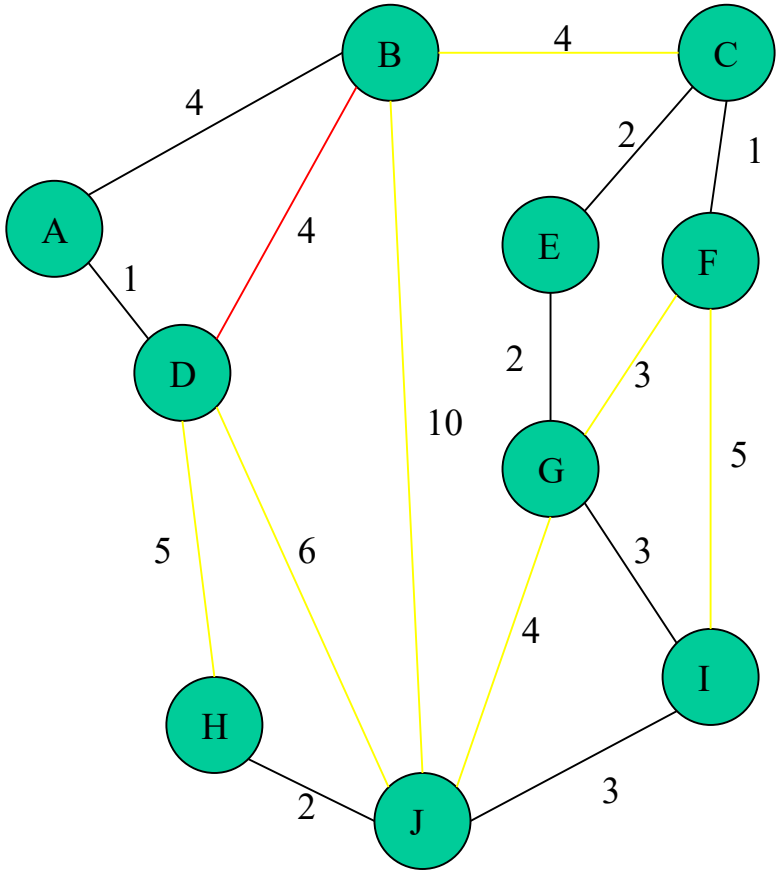
Add Edge

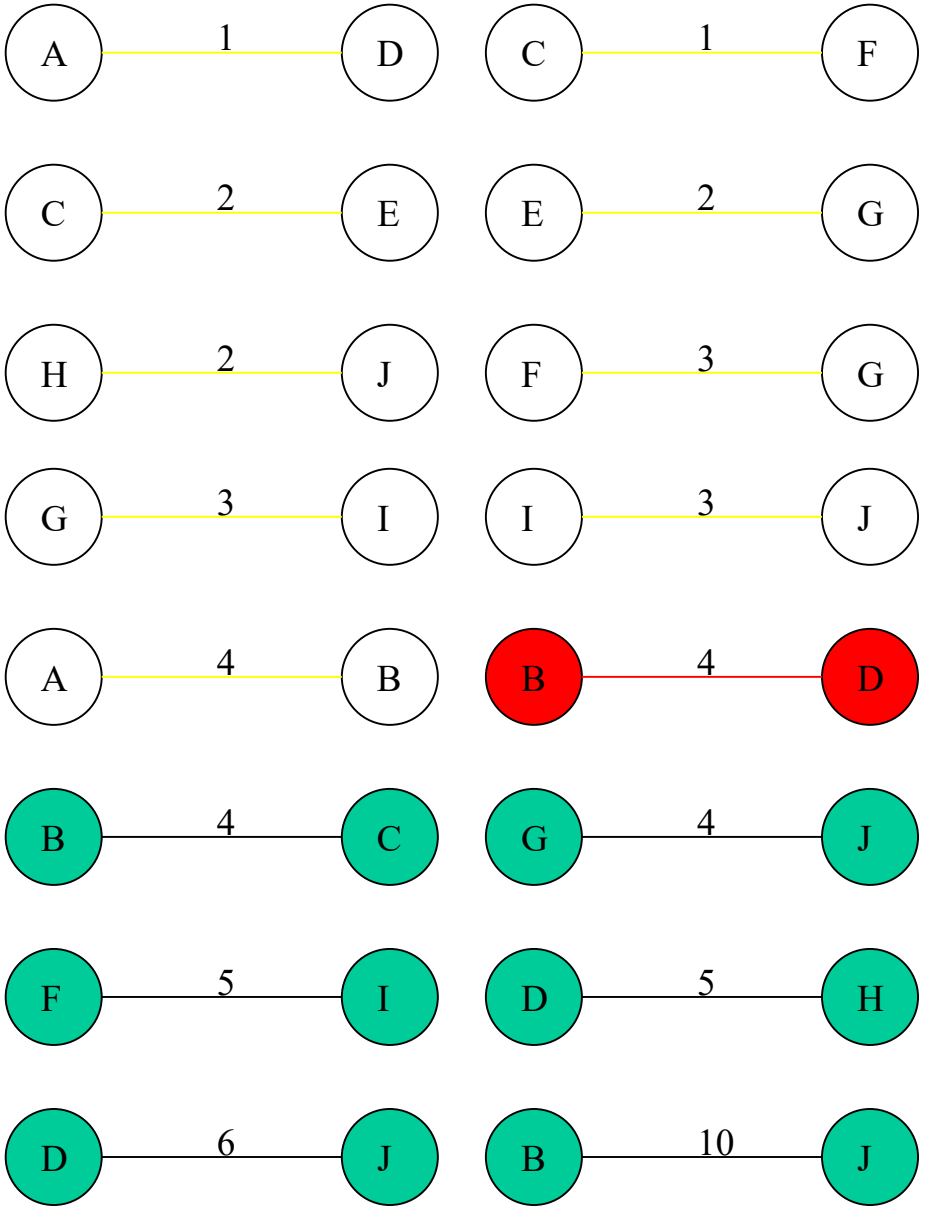

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3


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

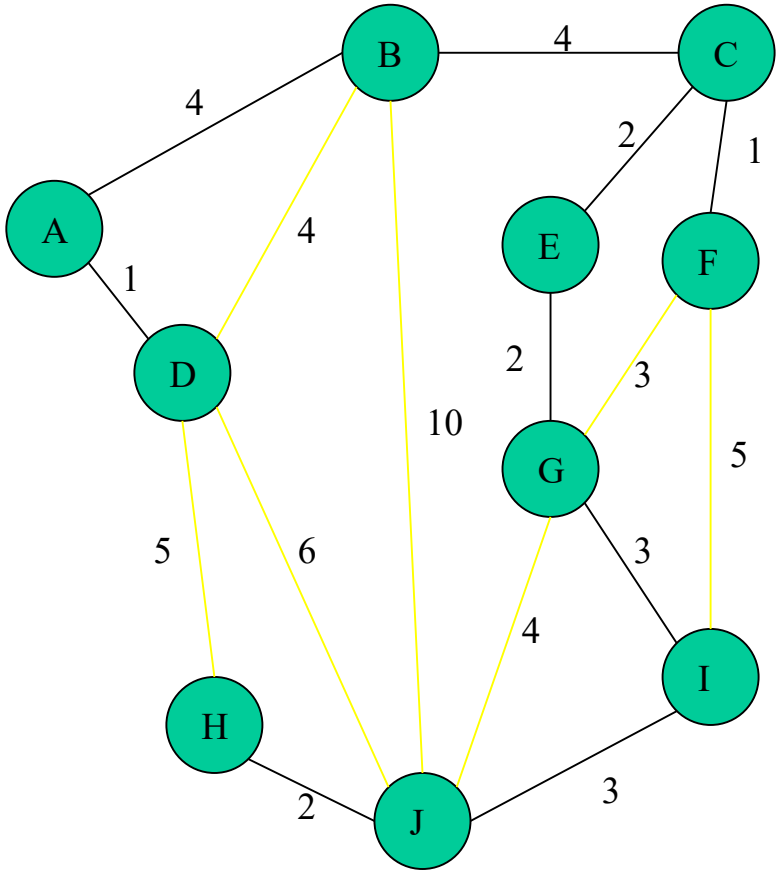
Cycle

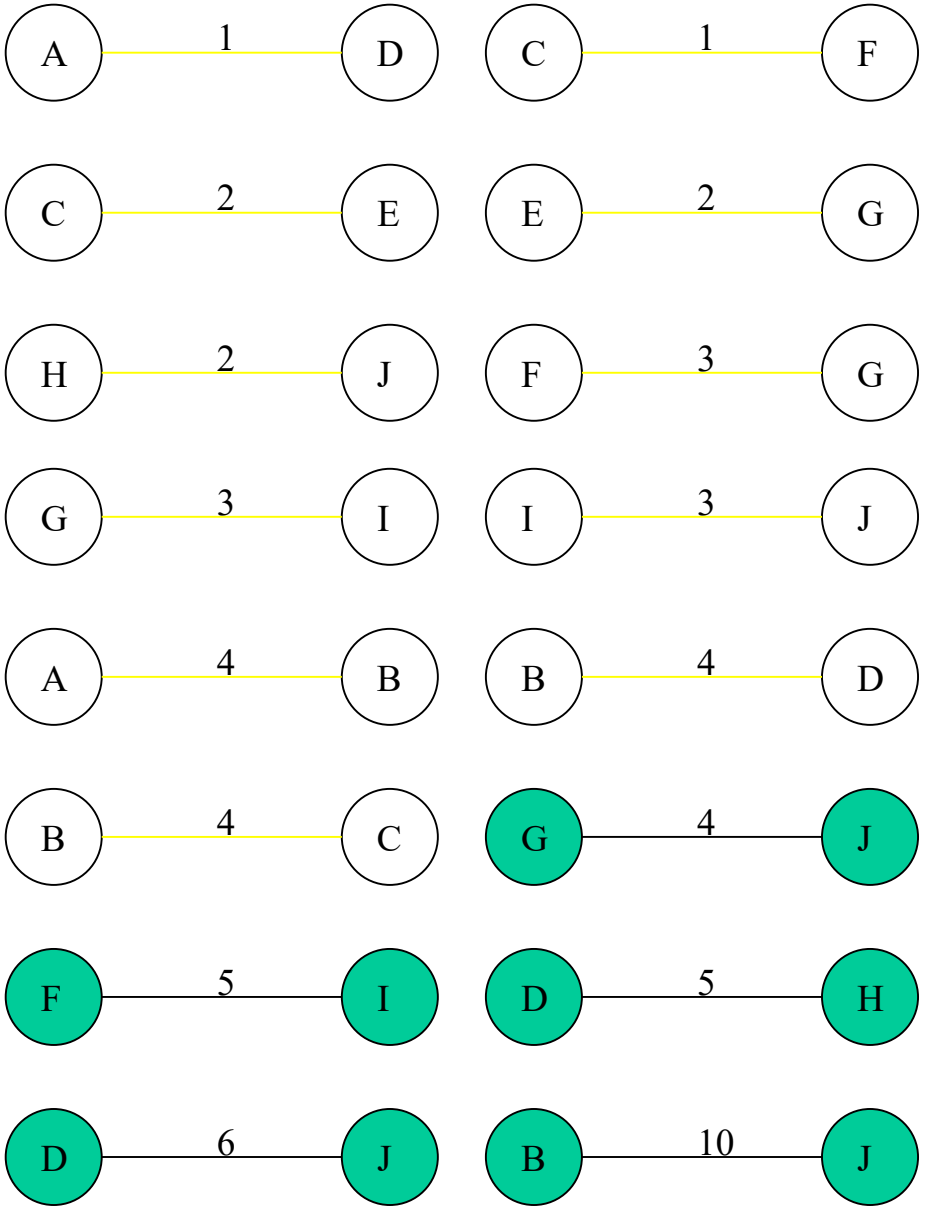
Don’t Add Edge



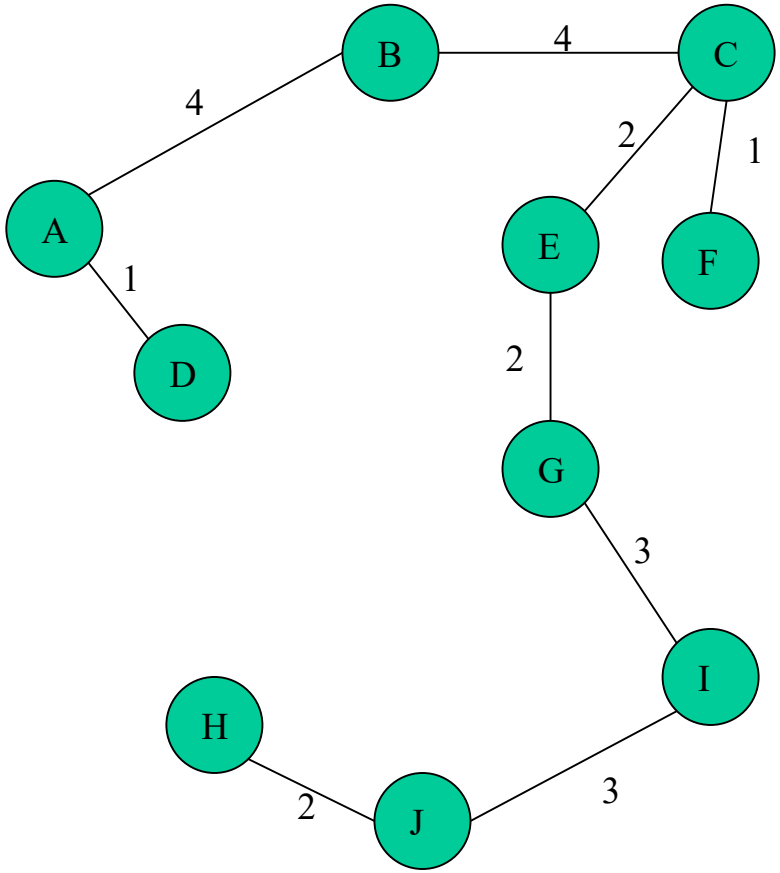


Add Edge

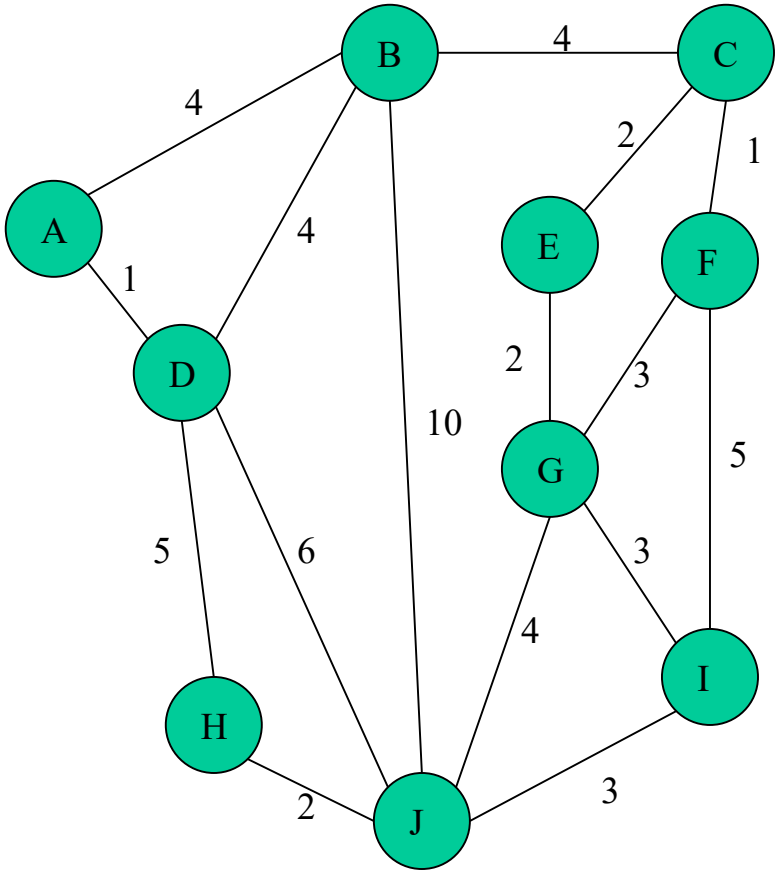

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
3
2
3


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

Minimum Spanning Tree



Graph



Disjoint-Set

- Keep a collection of sets S_1, S_2, .., S_k,
  - Each S_i is a set, e,g, $S_1=\{v_1, v_2, v_8\}$.
- Three operations
  - Make-Set(x)-creates a new set whose only member is x.
  - Union(x, y) –unites the sets that contain x and y, say, S_x and S_y, into a new set that is the union of the two sets.
  - Find-Set(x)-returns a pointer to the representative of the set containing x.
  - Each operation takes O(log n) time.

Kruskal's algorithm

```
MST_KRUSKAL(G,w)
1    A:={}
2    for each vertex v in V[G]
3       do MAKE_SET(v)
4    sort the edges of E by nondecreasing weight w
5    for each edge (u,v) in E, in order by
     nondecreasing weight
6       do if FIND_SET(u) != FIND_SET(v)
7             then   A:=A ∪ {(u,v)}
8                    UNION(u,v)
9    return A
```

- Our implementation uses a disjoint-set data structure to maintain several disjoint sets of elements.
- Each set contains the vertices in a tree of the current forest.
- The operation FIND_SET(u) returns a representative element from the set that contains u.
- Thus, we can determine whether two vertices u and v belong to the same tree by testing whether FIND_SET(u)=FIND_SET(v).
- The combining of trees is accomplished by the UNION procedure.
- Running time O(|E| log (|E|)).

Prim’s Algorithm

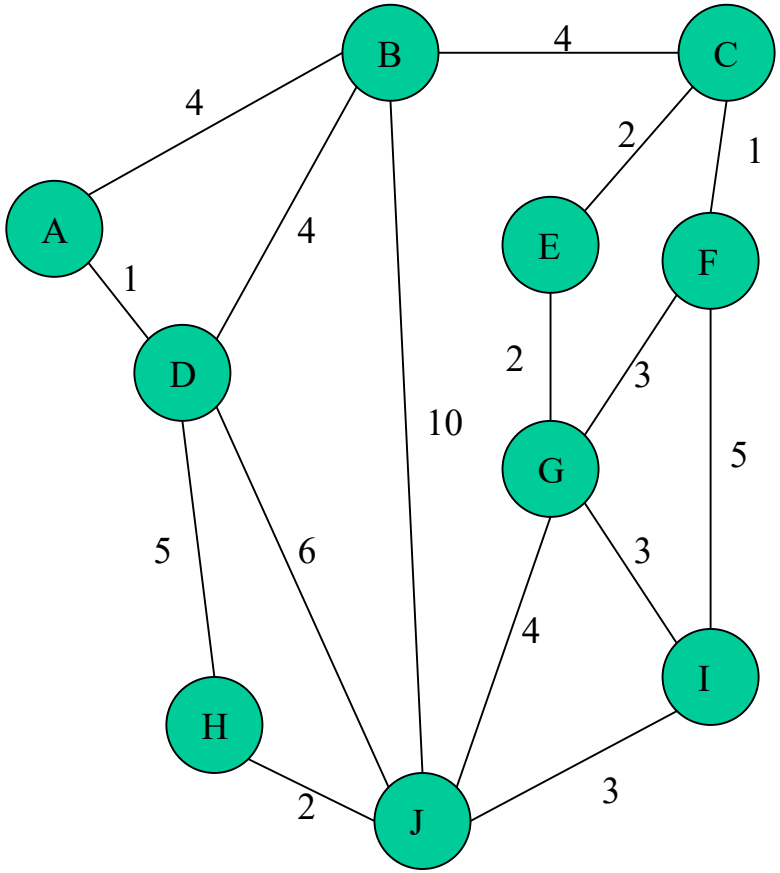
Prim's algorithm(basic part)

MST_PRIM(G,w,r)

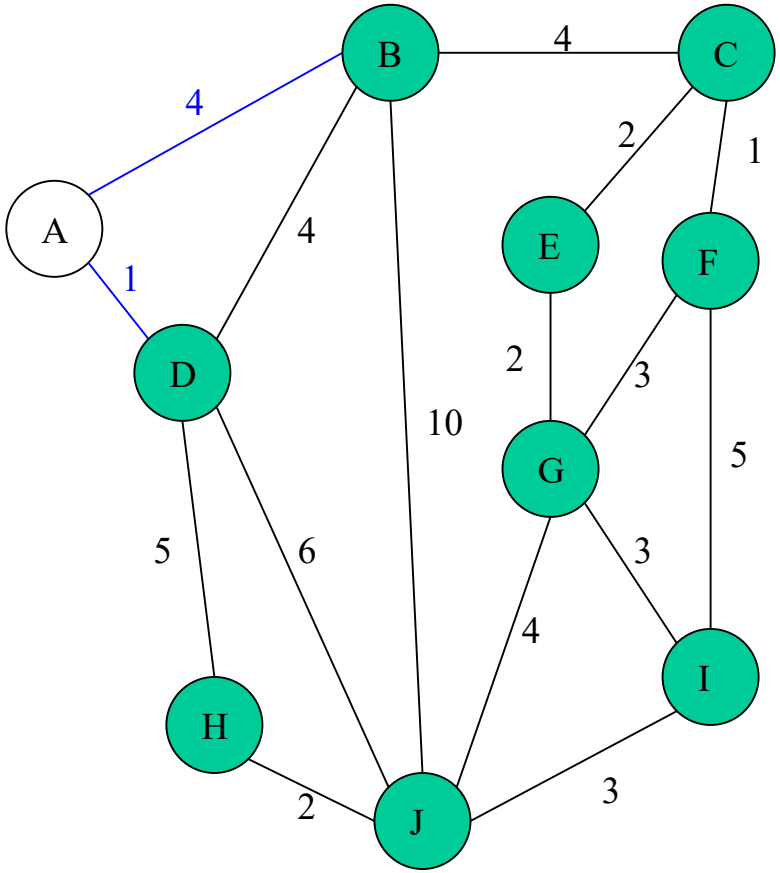
1. A={}
2. S:={r}
3. Q=V-{r};
4. **while** Q is not empty **do {**
5. take an edge (u, v) such that **(1) u ∈S and v ∈ Q (v∉ S)** and (u, v) is the shortest edge satisfying (1)
6. **add** (u, v) to A, add v to S and delete v from Q

}

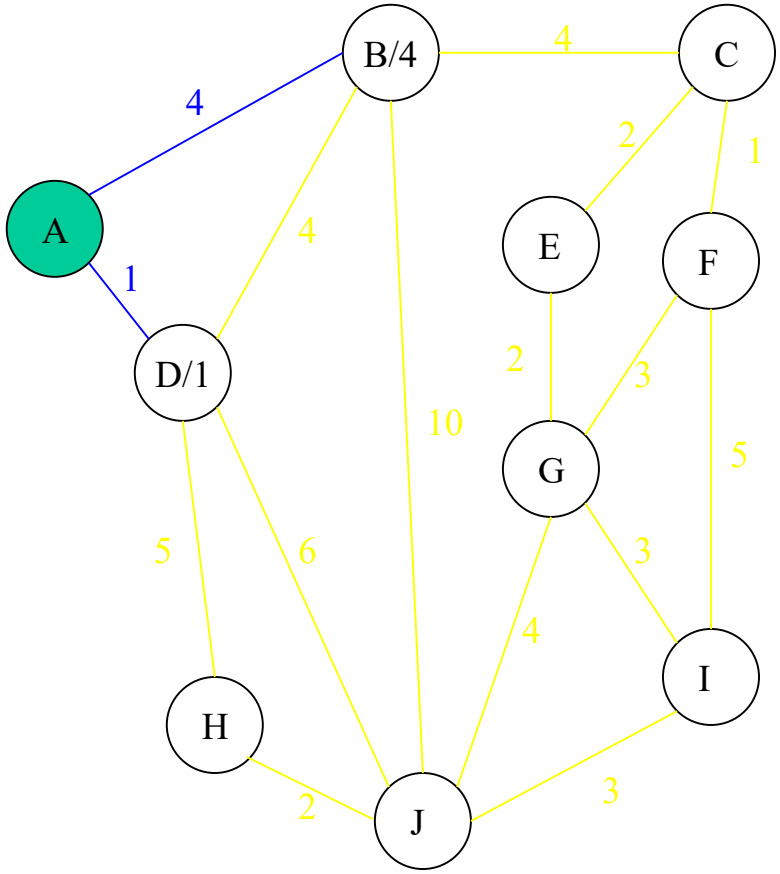
Graph



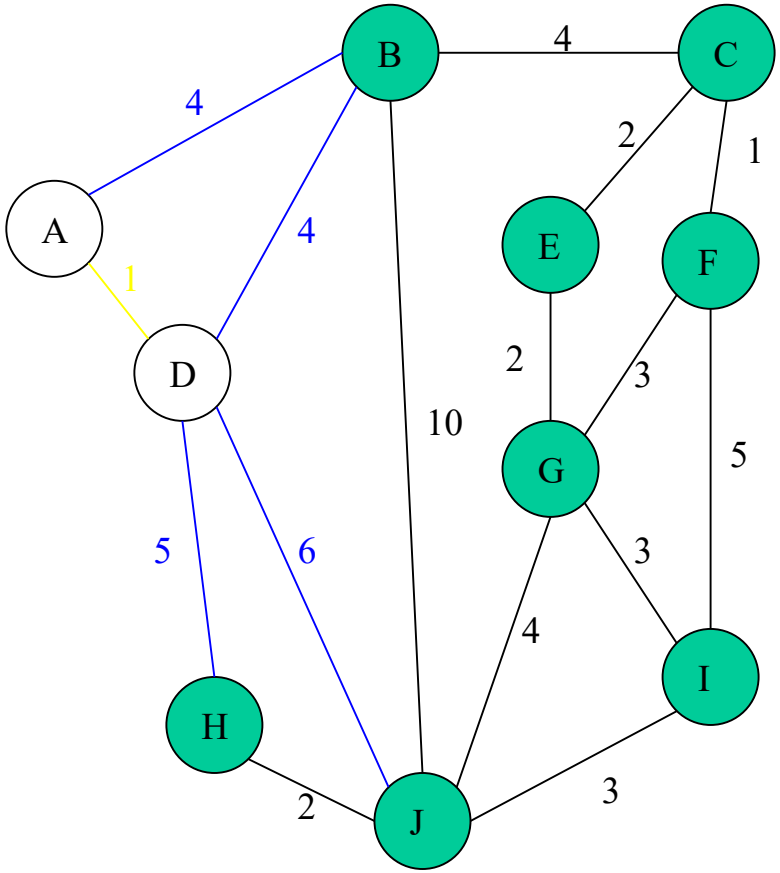
Old Graph



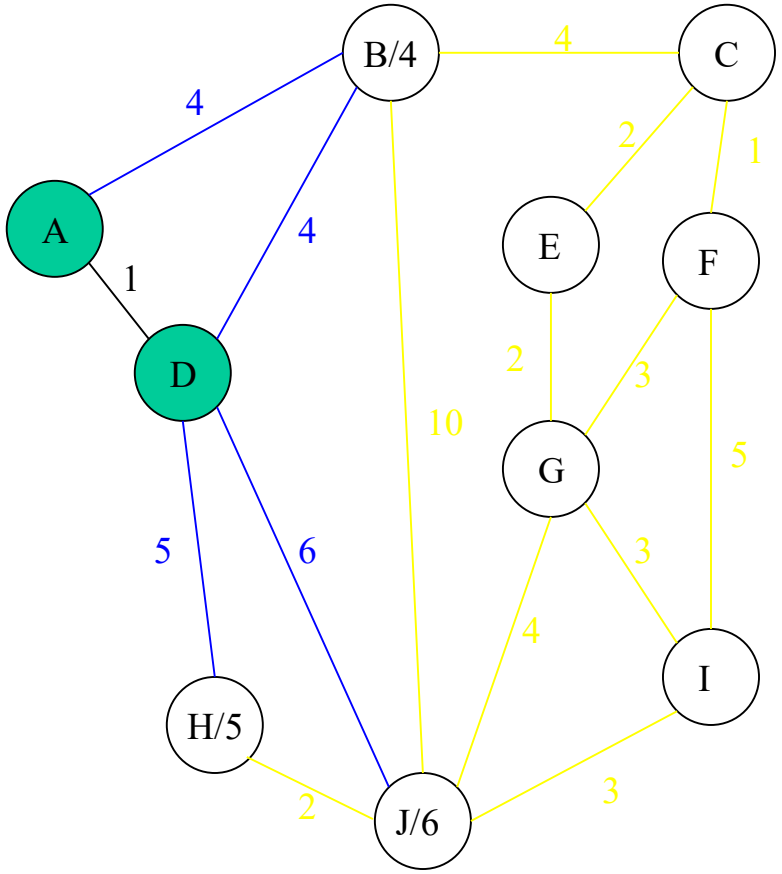
Tree



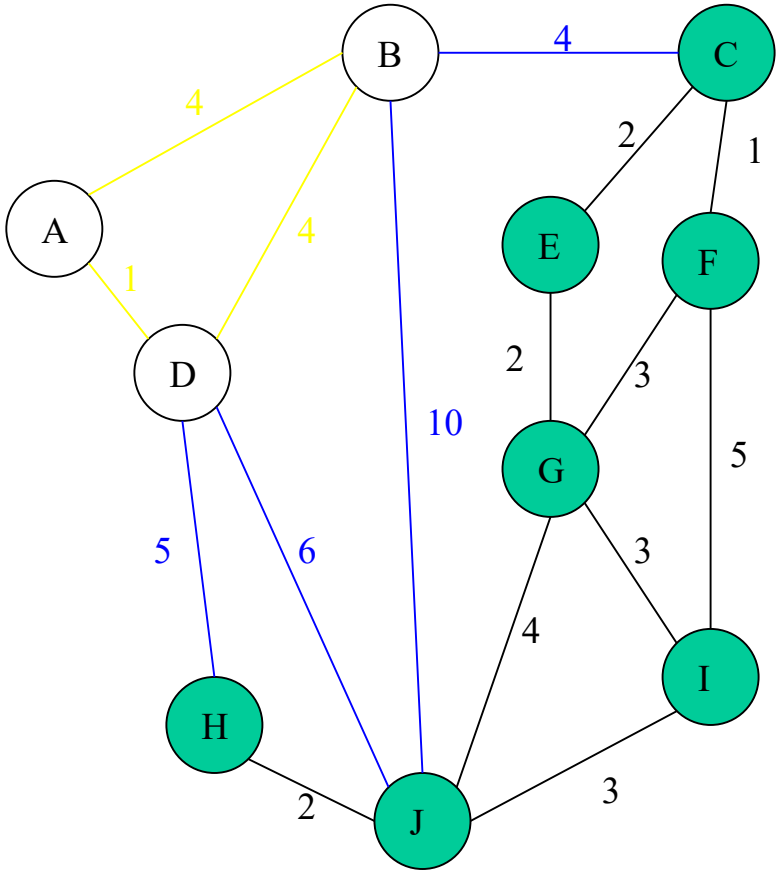
Old Graph



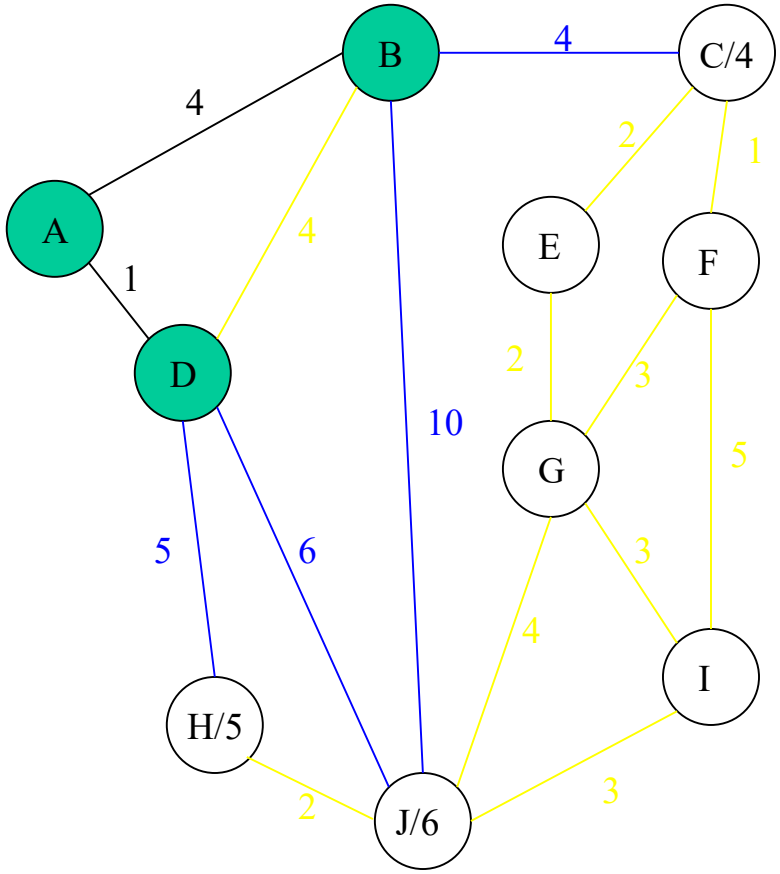
Tree



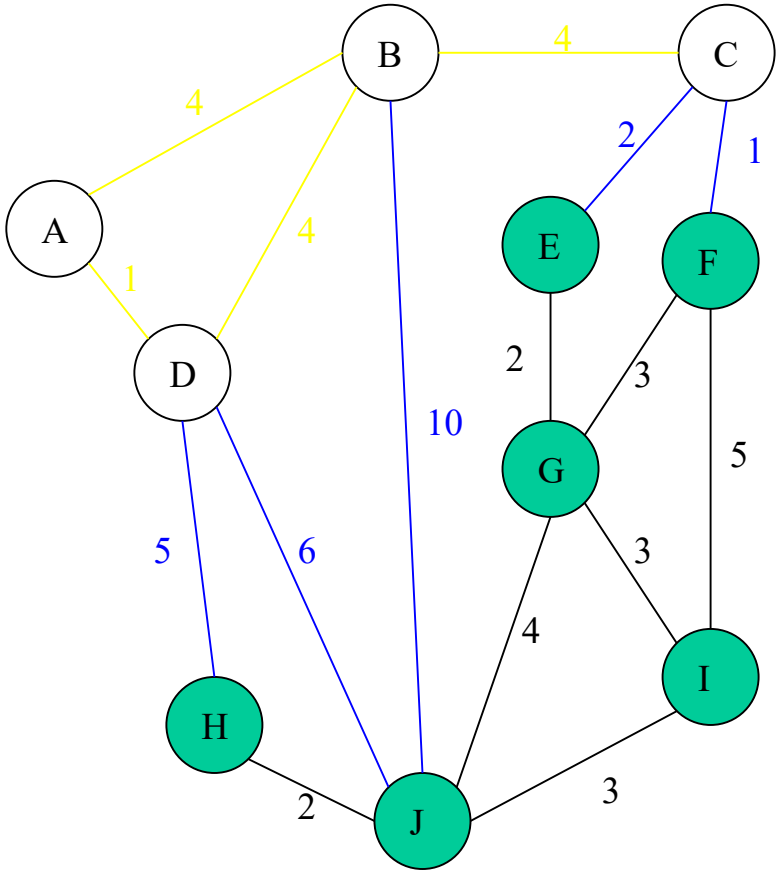
Old Graph



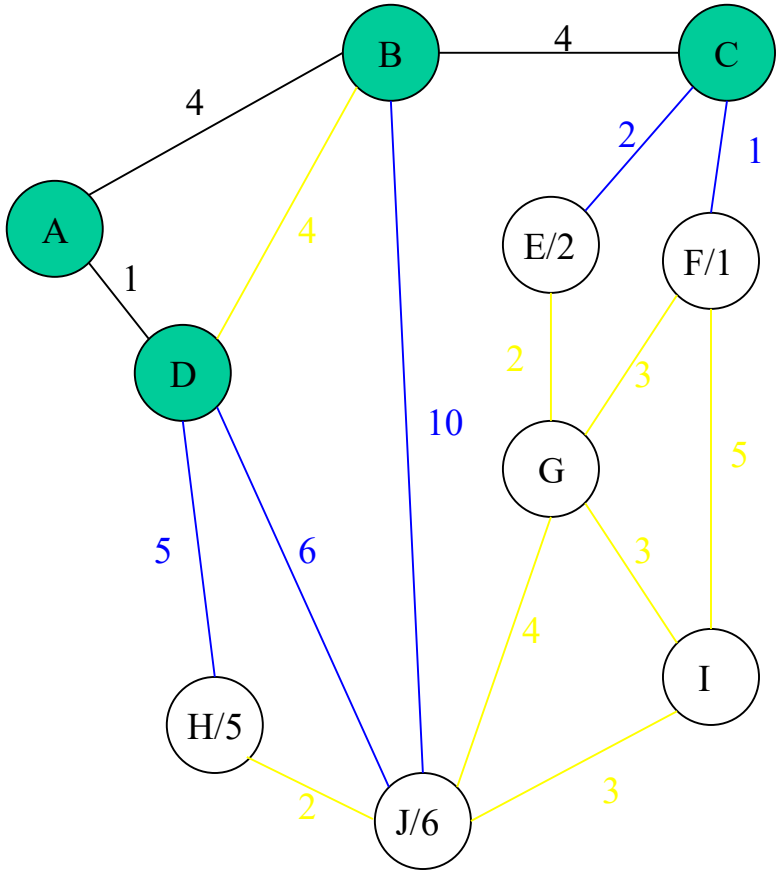
Tree



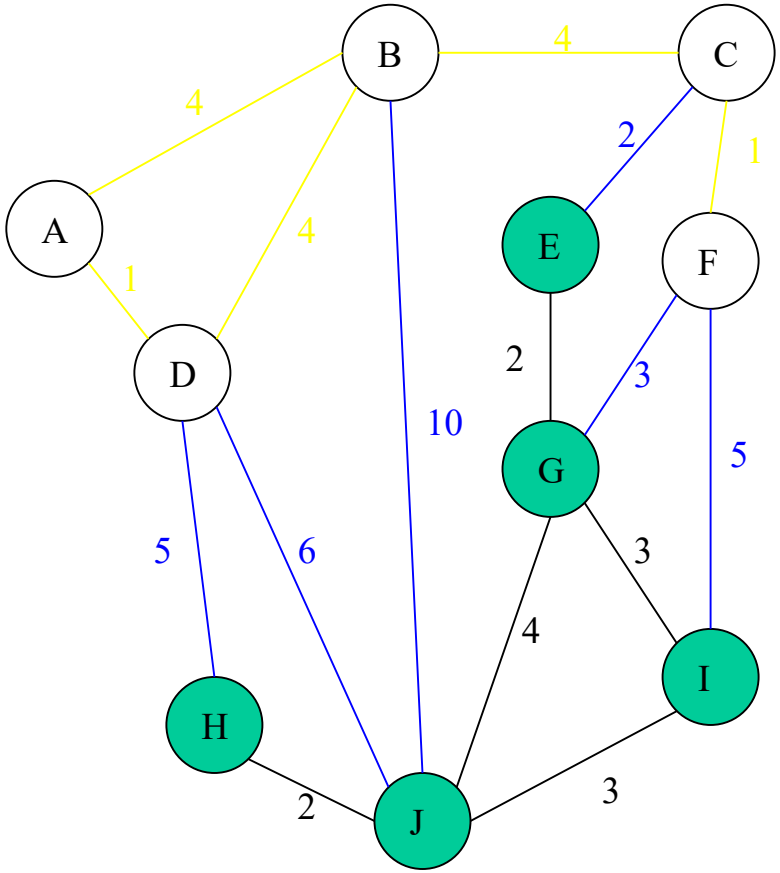
Old Graph



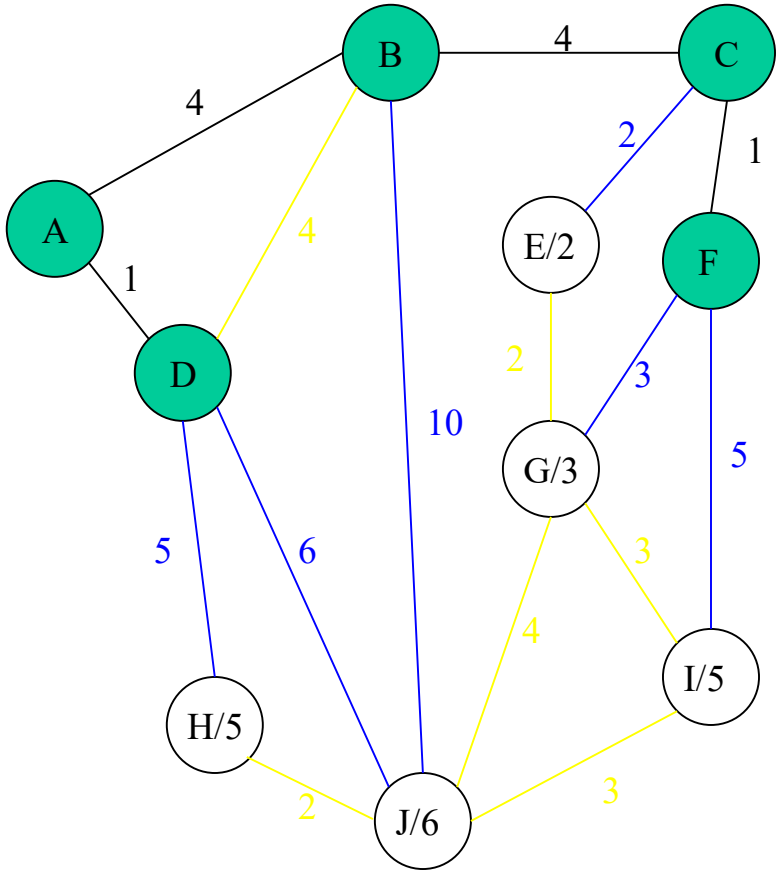
Tree



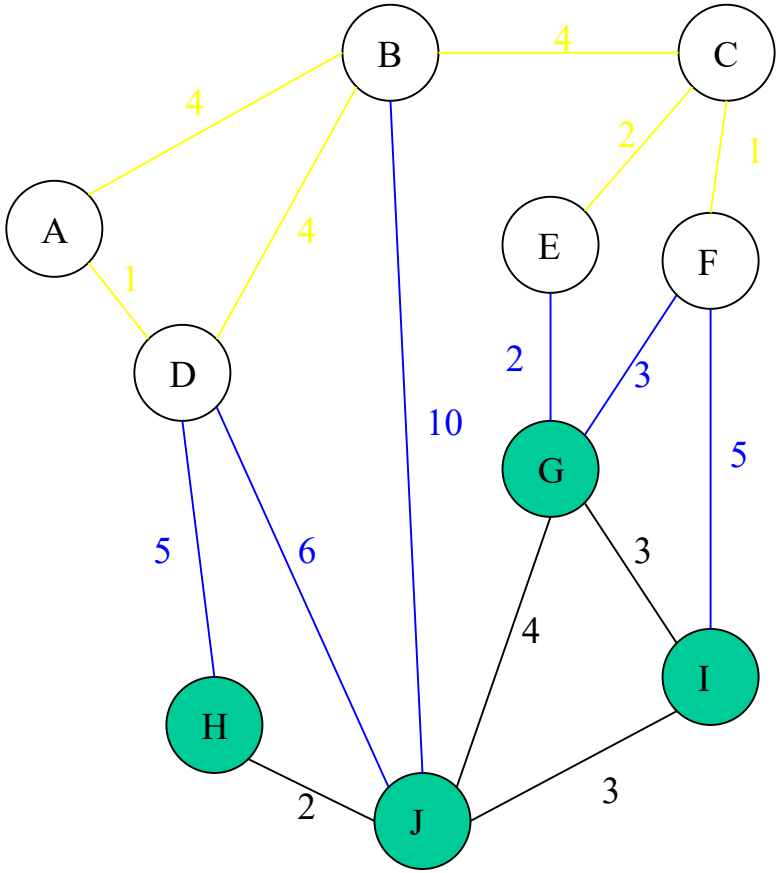
Old Graph



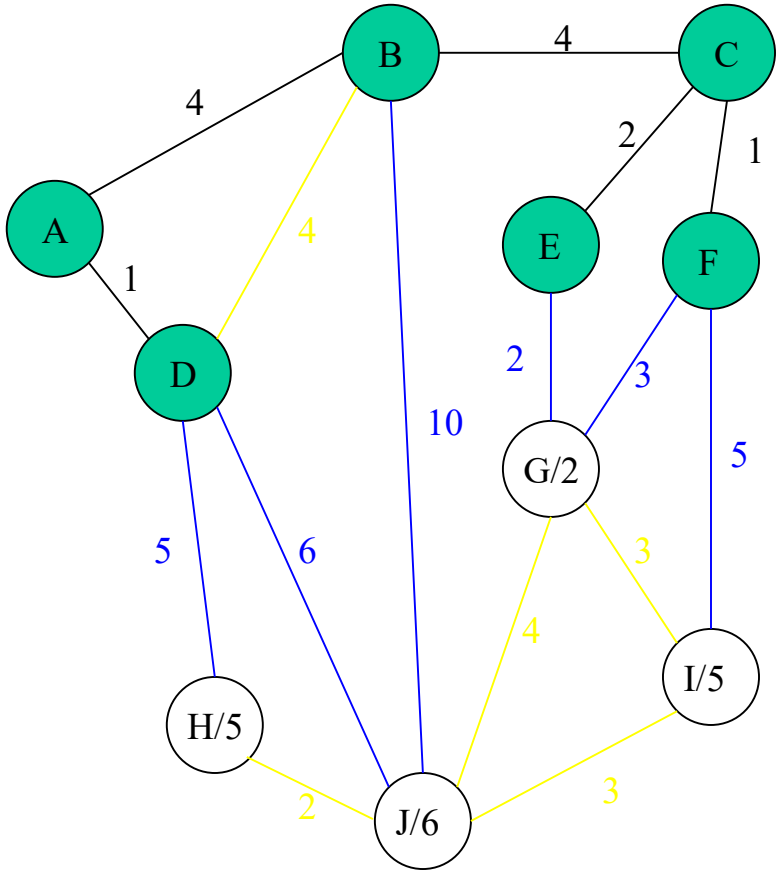
Tree



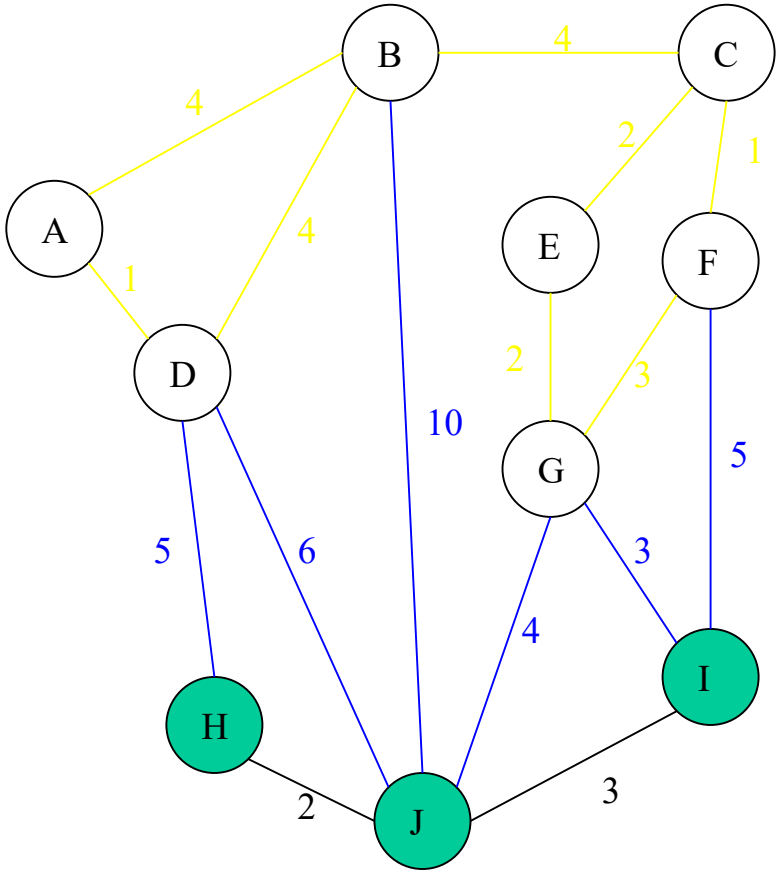
Old Graph



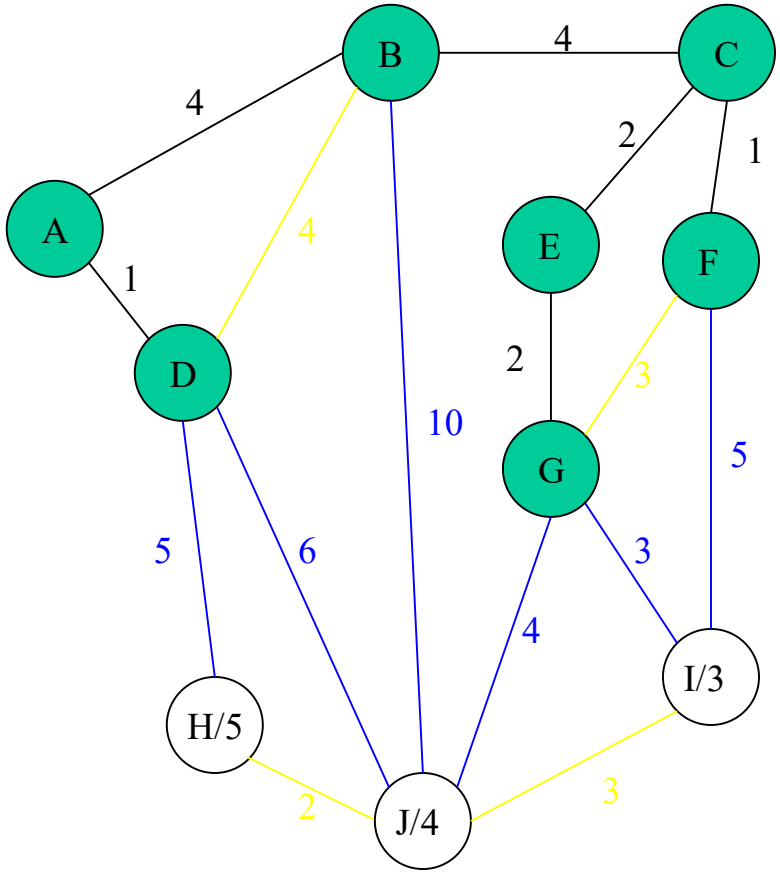
Tree



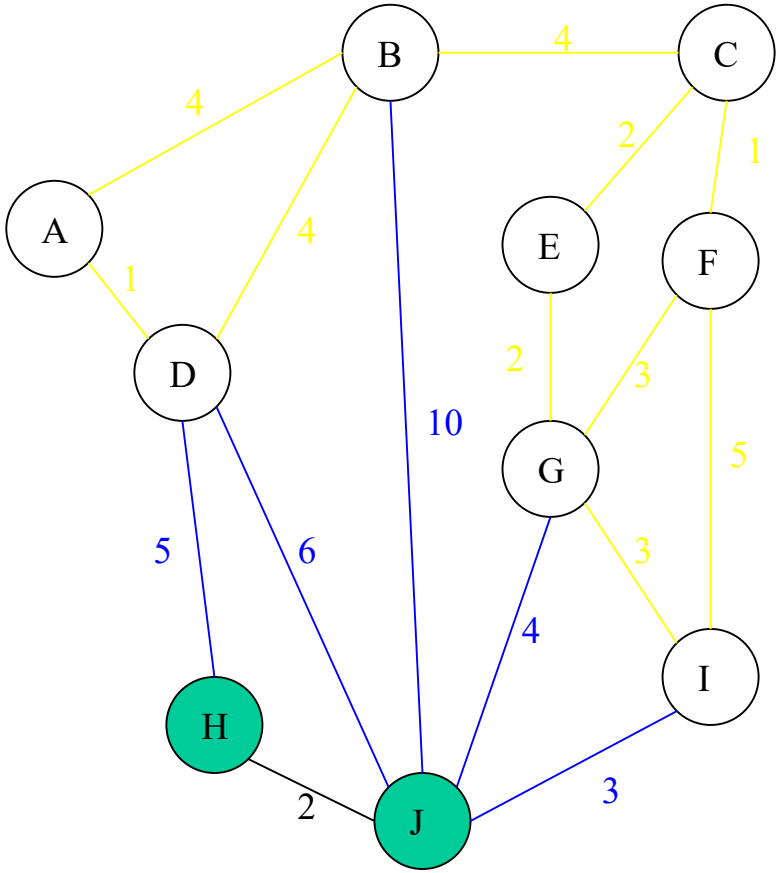
Old Graph



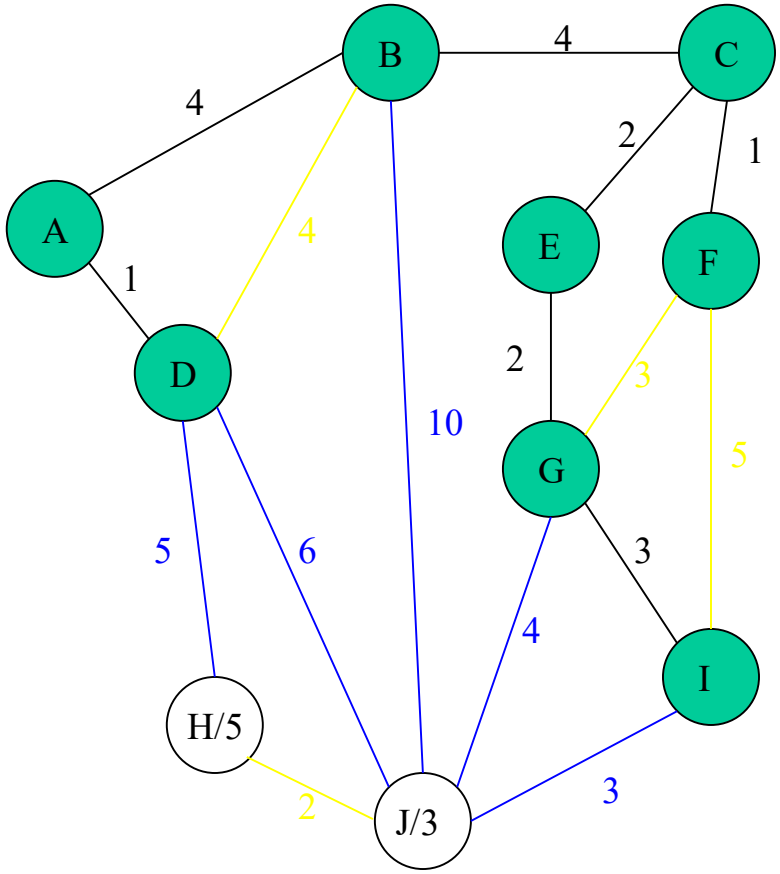
Tree



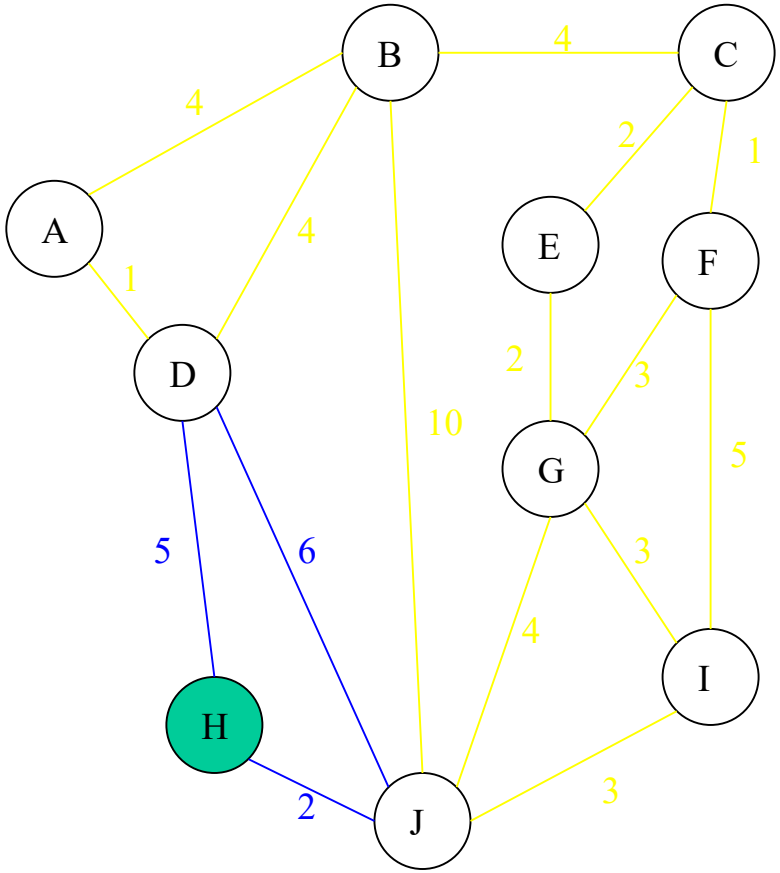
Old Graph



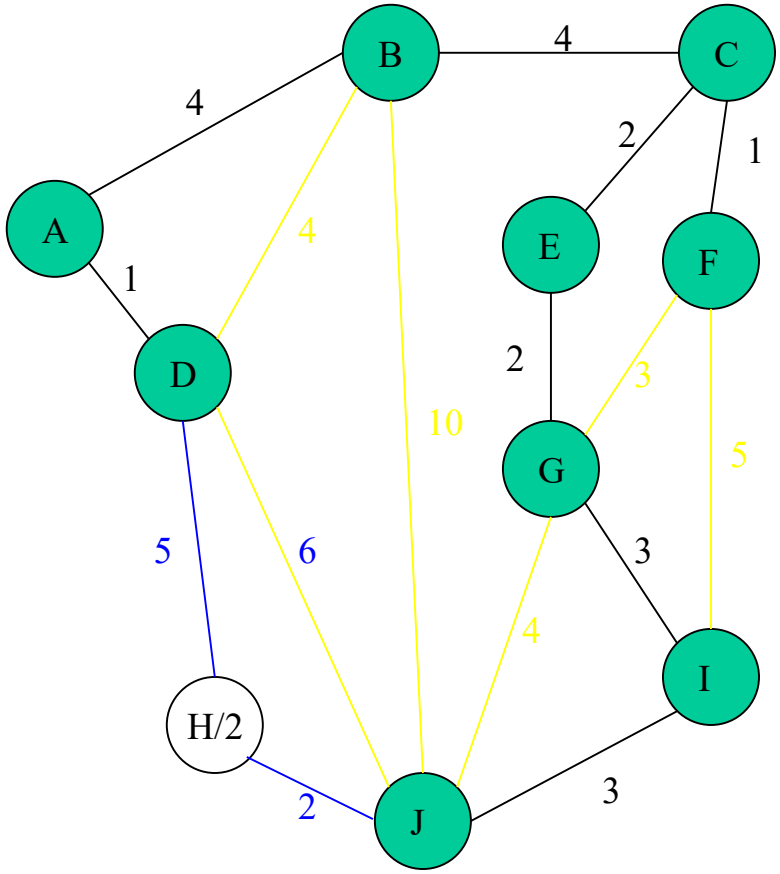
Tree



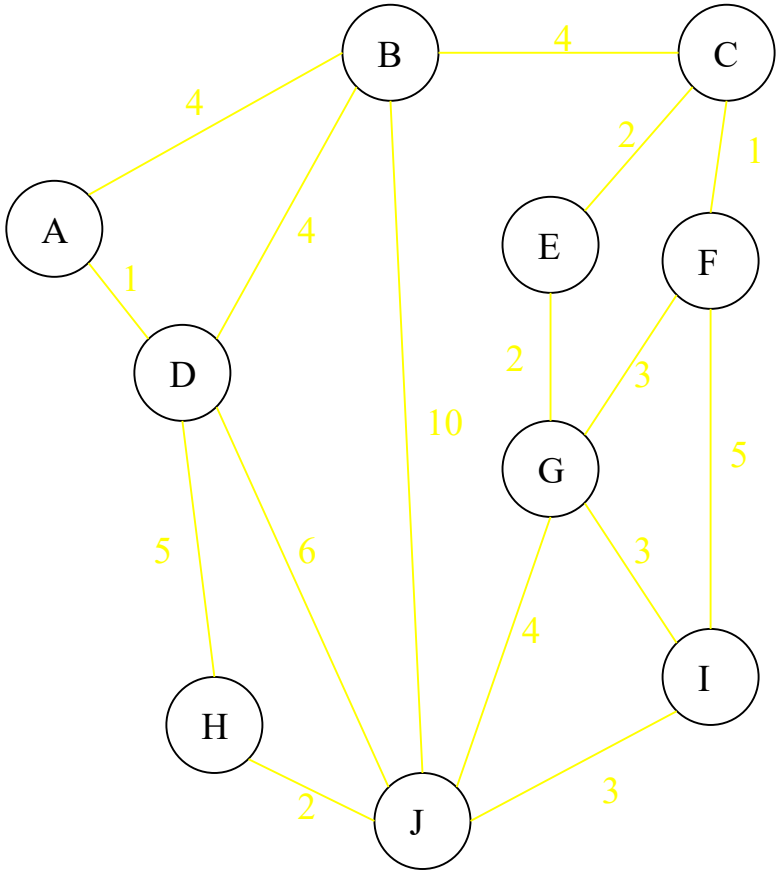
Old Graph



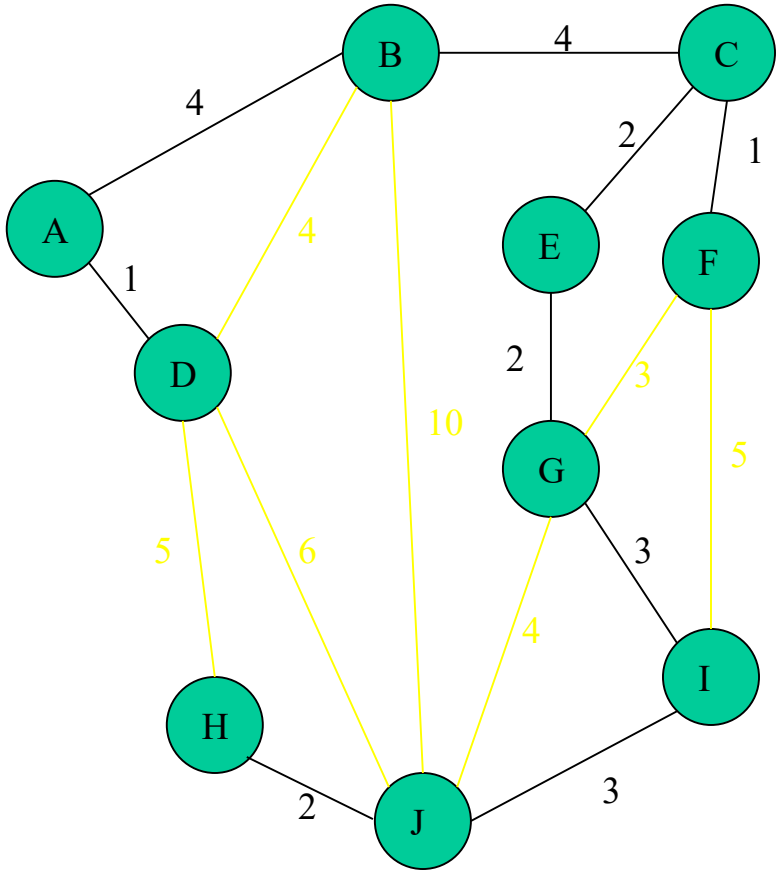
Tree



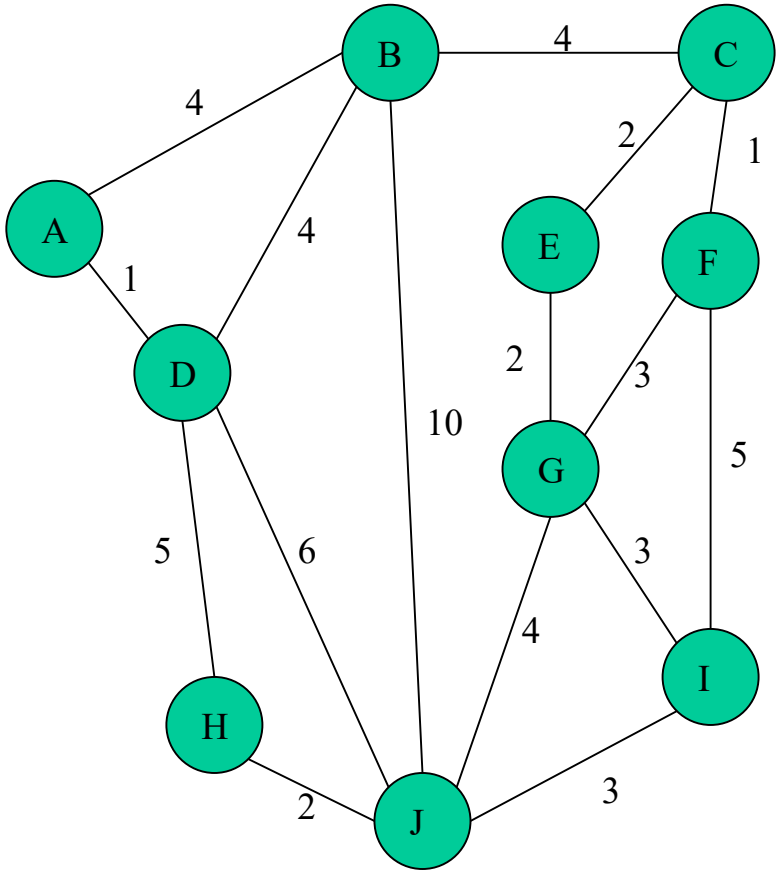
Old Graph



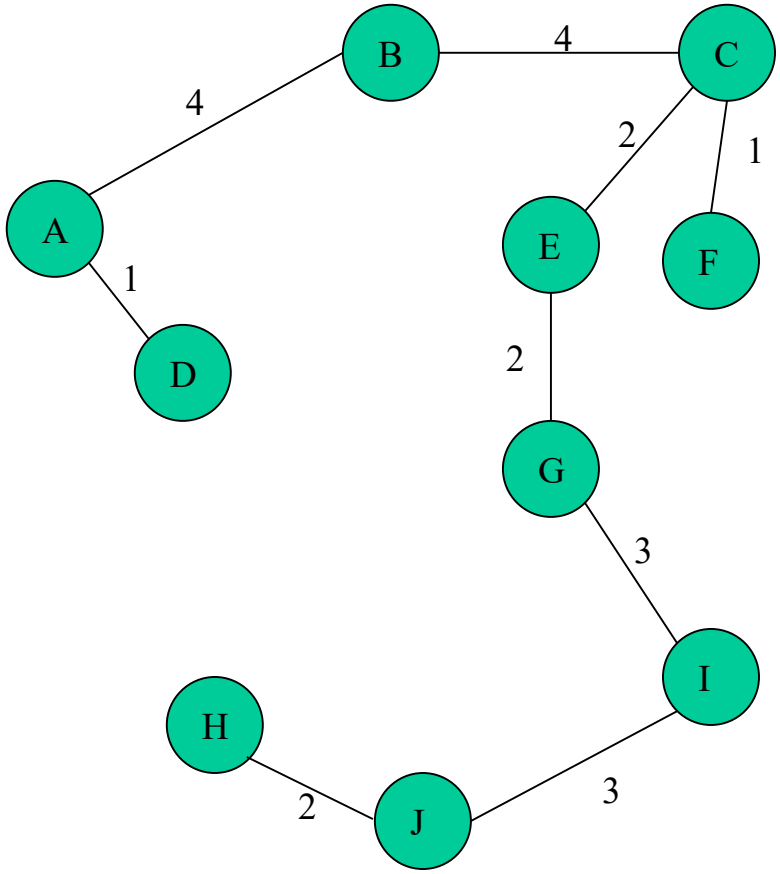
Tree



Complete Graph



Minimum Spanning Tree



Prim's algorithm

```
MST_PRIM(G, w, r)
1   for each v in V do
2      key[v]:=∞,  parent[v]:=NIL
3      A:=∅
4    key[r]:=0; parent[r]=NIL;
5    Q←(V, key)  /* initialize Q.
6    while Q!={} do
7       u:=EXTRACT_MIN(Q); if parent[u]≠NIL, A:=A U (u, parent[u]).
8       for each v in Adj[u] do
9              if v in Q and w(u,v)<key[v]
10             then    parent[v]:=u
11                     k:=w(u,v)
12                     Update(v,k)
```

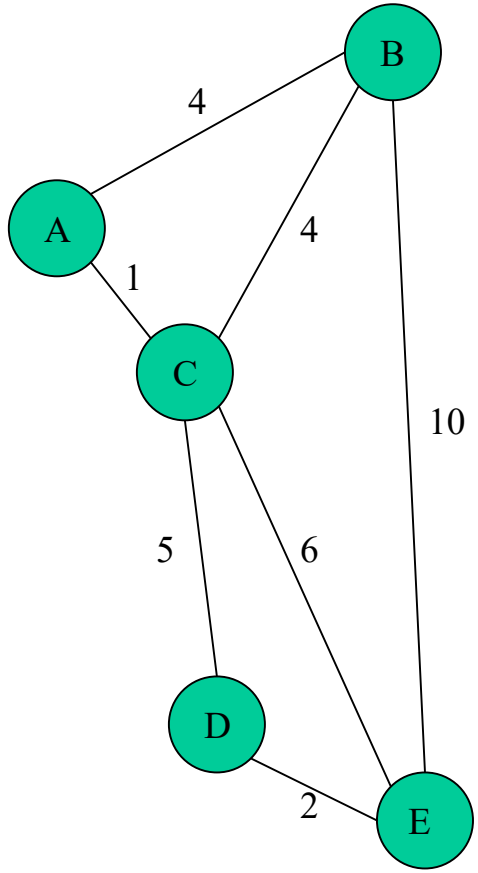
Q (priority queue): contain all the vertices that have not yet been included in the tree. V\S: vertices in the tree

Parent[v]: the nearest vertex in the tree to v
Key[v]: the length of edge (v, parent[v])

i.e. u is not r.

u: the nearest vertex in Q to the tree.
Remove u from Q, i.e., add u to the tree

v has an edge with u



After 1-5: The tree is empty. Q contains:

Node	A	B	C	D	E
Key	0	inf	inf	inf	inf
Parent	NIL	NIL	NIL	NIL	NIL

After one pass in while loop. The tree has one node A. Q is:

Node	B	C	D	E
Key	4	1	inf	inf
Parent	A	A	NIL	NIL

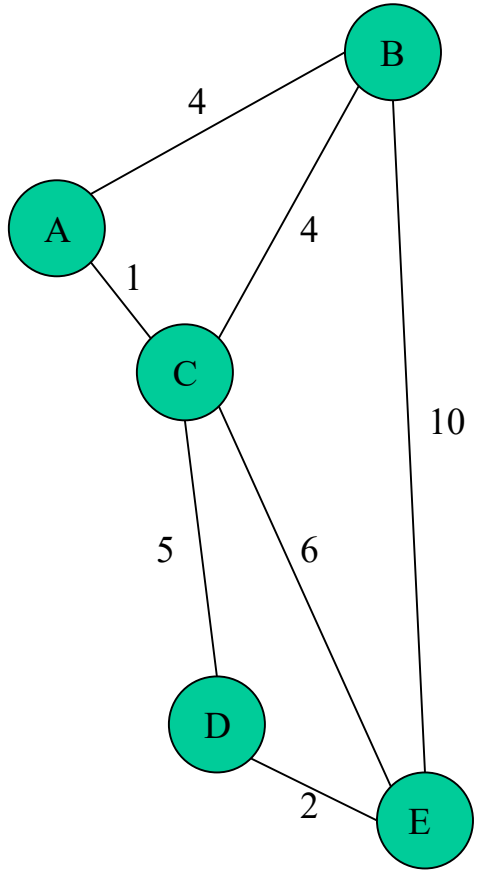
After second pass in while loop. The tree has two node A C, and one link (A,C), Q is:

Node	B	D	E
Key	4	5	6
Parent	A	C	C

After third pass in while loop. The tree has two node A C, B and two links (A,C) and (A,B), Q is:

Node	D	E
Key	5	6
Parent	C	C

Node	E
Key	2
Parent	D



After 1-5: The tree is empty. Q contains:

Node	A	B	C	D	E
Key	0	inf	inf	inf	inf
Parent	NIL	NIL	NIL	NIL	NIL

After one pass in while loop. The tree has one node A. Q is:

Node	B	C	D	E
Key	4	1	inf	inf
Parent	A	A	NIL	NIL

After second pass in while loop. The tree has two node A C, and one link (A,C), Q is:

Node	B	D	E
Key	4	5	6
Parent	A	C	C

After third pass in while loop. The tree has two node A C, B and two links (A,C) and (A,B), Q is:

Node	D	E
Key	5	6
Parent	C	C

Node	E
Key	2
Parent	D

priority queue Q: data structure containing n items: each item u has a key value key[u].

three important operations

- EXTRACT-MIN(Q) –takes O(log (n)) times if there are n items in Q.
- Insert an element (key u) into Q- takes O(log(n)) time.
- Update(u, k): update the key value of element u to k - takes O (log (n)) time.

- Grow the minimum spanning tree from the root vertex r.
- Q is a priority queue, holding all vertices that are not in the tree now.
- key[v]: the length of the shortest edge linking v with a vertex in the tree.
- parent[v]: the parent of v in the tree. (v, partent[v]) is the shortest among all the edges linking v with a vertex in the tree.
- When the algorithm terminates, Q is empty; the minimum spanning tree A for G is thus $A=\{(v,parent[v]):v\in V-\{r\}\}$.
- Running time: $O(||E||\log |V|)$.
- Using *Fibonacci heap* structure to store Q, we can reduce the complexity to $O(|E|+|V|\log|V|)$ since Update(u, k) can be done in O(1) time for Fibonacci heap.

Summary

- MST
- Cut, safe edge.
- Generic MST algorithm
- Kruskal's algorithm
- Prim's algorithm

Challenge Problem

1. Consider the problem of computing a *maximum* spanning tree, namely the spanning tree that maximizes the sum of edge costs. Do Prim and Kruskal's algorithm work for this problem (assuming of course that we choose the crossing edge with maximum cost)?

1. Prove that for any weighted undirected graph such that the weights are distinct (no two edges have the same weight), the minimum spanning tree is unique.