5.1.1 Kruskal's Algorithm

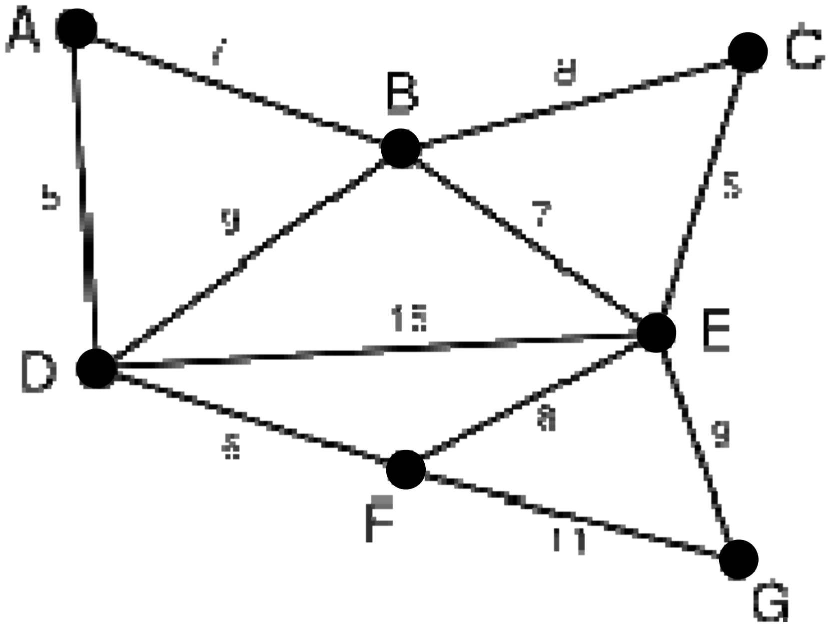
- Edge-oriented algorithm
- Algorithm
  - Sort all the edges of a graph and list them in the ascending order.
  - Choose the edge of the minimum cost from the sorted list, and add it to the minimum cost tree T, if it doesn't make a cycle.
  - Repeat this process until T has (n-1) edges or the sorted list becomes empty.

5.1.1 Kruskal’s Algorithm

```
Tree Kruskal( Vertex V, Edge E )
{
    T = {};
    sort edges in E in ascending order;

    while ( |T| < n-1 && E != empty ) {
       choose a least cost edge (v, w) from E;
       delete (v, w) from E;
       if ( (v, w) does not create a cycle in T )
           add (v, w) to T;
       else
           discard (v, w);
    }
    if ( |T| < n-1 )
       return NULL;

    return T;
}
```

5.1.1 Kruskal's Algorithm

- Ex)

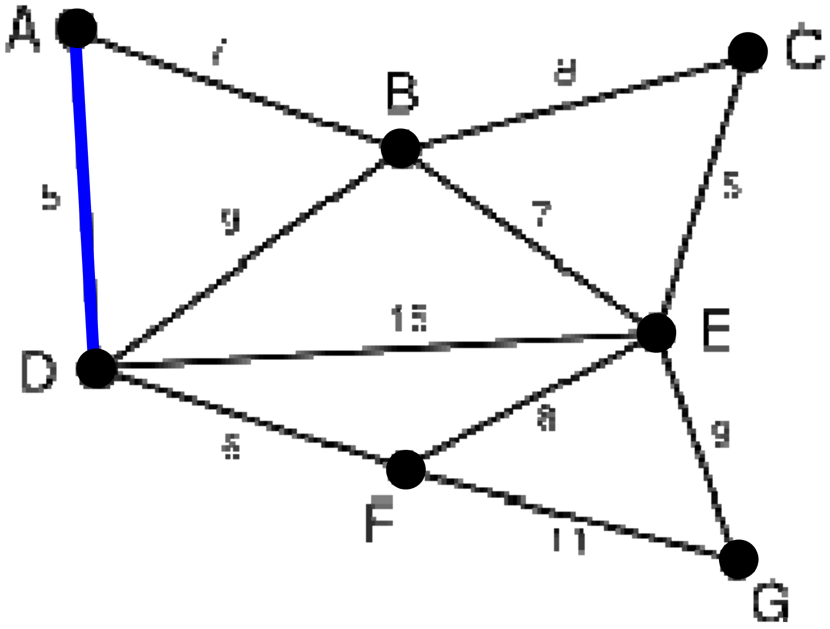


edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T

5.1.1 Kruskal’s Algorithm

- Ex)

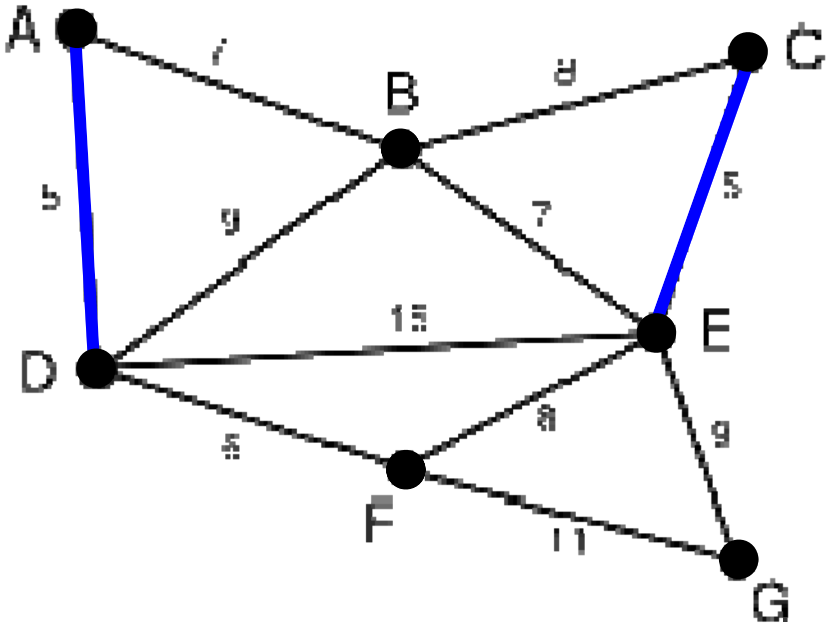


edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)

5.1.1 Kruskal's Algorithm

- Ex)

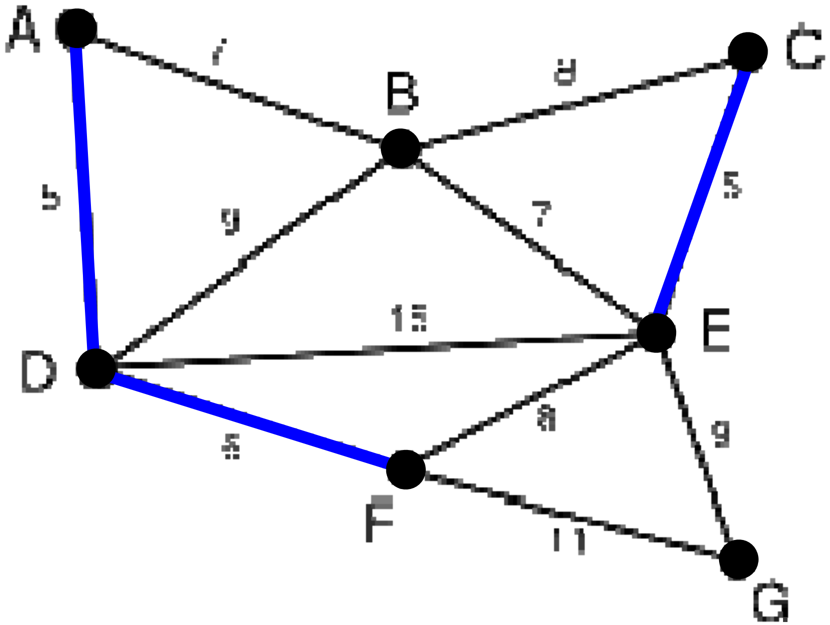


edges	***weights***
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)

5.1.1 Kruskal’s Algorithm

- Ex)

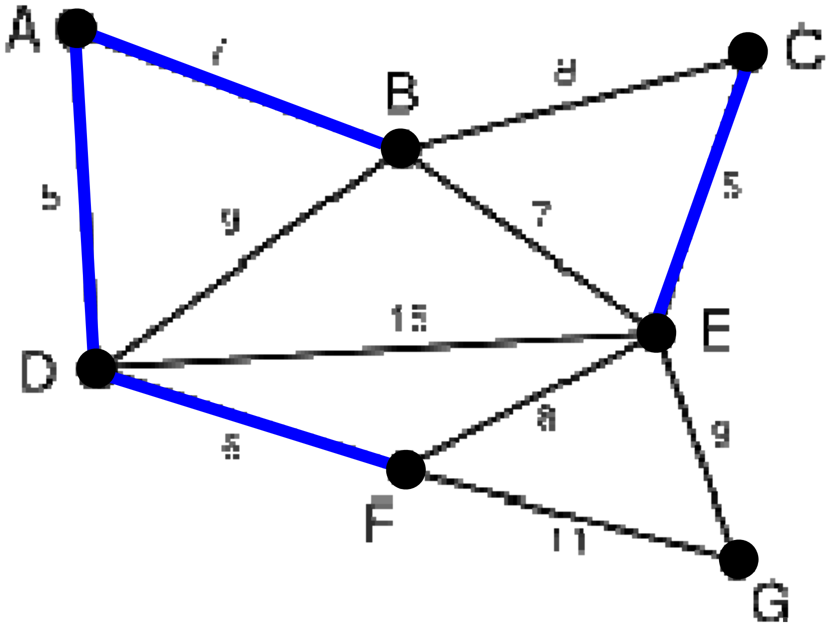


edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)

5.1.1 Kruskal's Algorithm

- Ex)

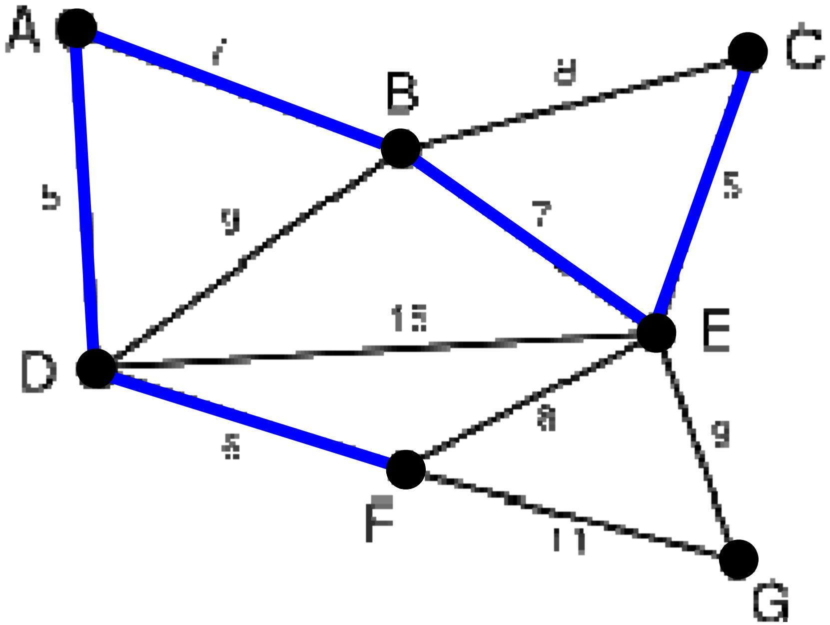


edges	weights
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)

5.1.1 Kruskal's Algorithm

- Ex)

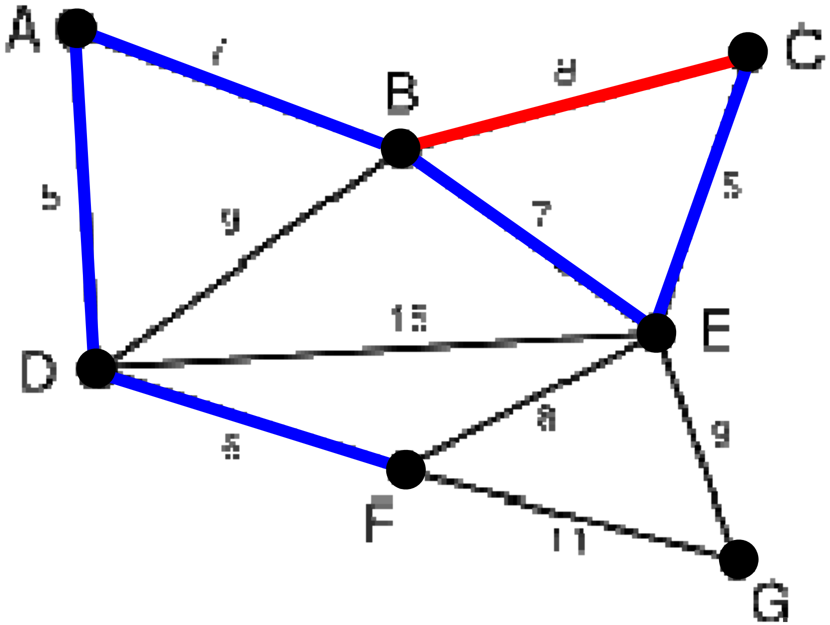


edges	***weights***
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)

5.1.1 Kruskal's Algorithm

- Ex)

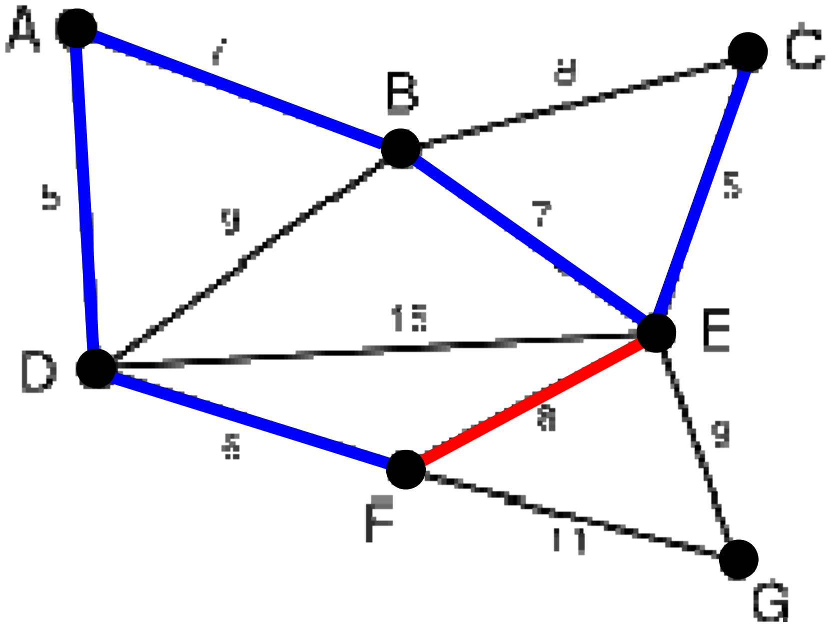


edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)

5.1.1 Kruskal's Algorithm

- Ex)

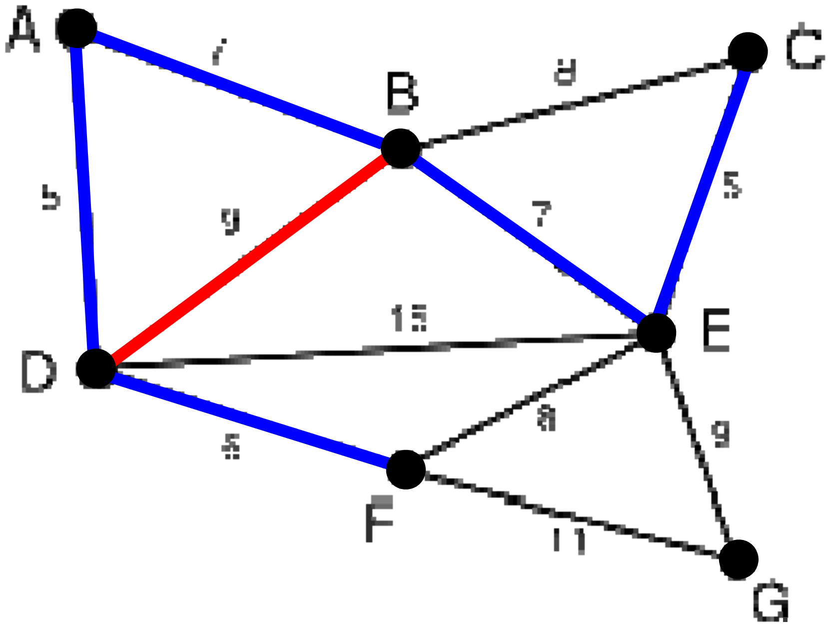


edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)

5.1.1 Kruskal’s Algorithm

- Ex)

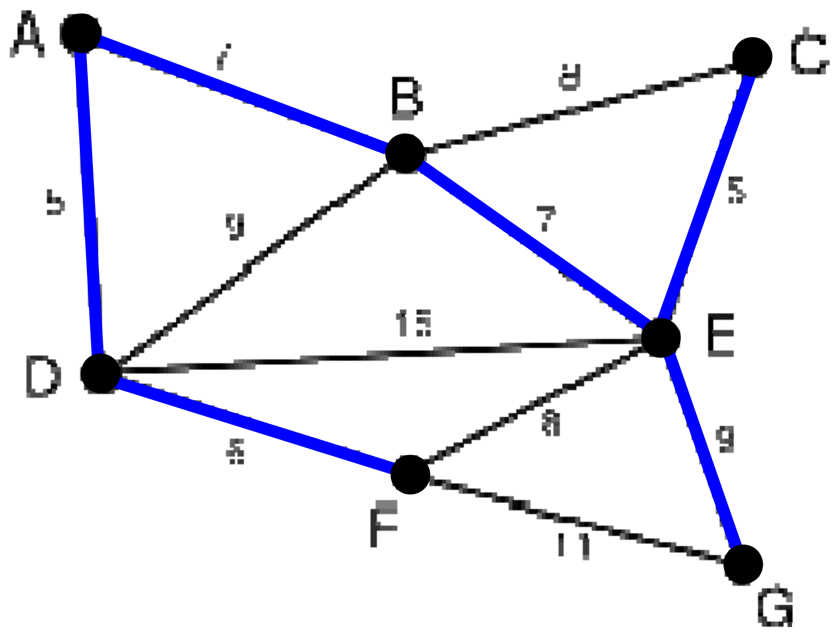


edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)

5.1.1 Kruskal's Algorithm

- Ex)



edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)
(E,G)

5.1.1 Kruskal's Algorithm

- Performance analysis
  - Sorting edges → O(m log m)
  - Adding edges → O(n)
    - Check cycles → O(n)

- How to improve performance?
  - Use UNION-FIND operations for checking cycles
  - Assign labels on the vertices for UNION-FIND
    - Forest-based implementation
    - UNION → O(1)
    - FIND → O(log n)

5.1.1 Kruskal's Algorithm

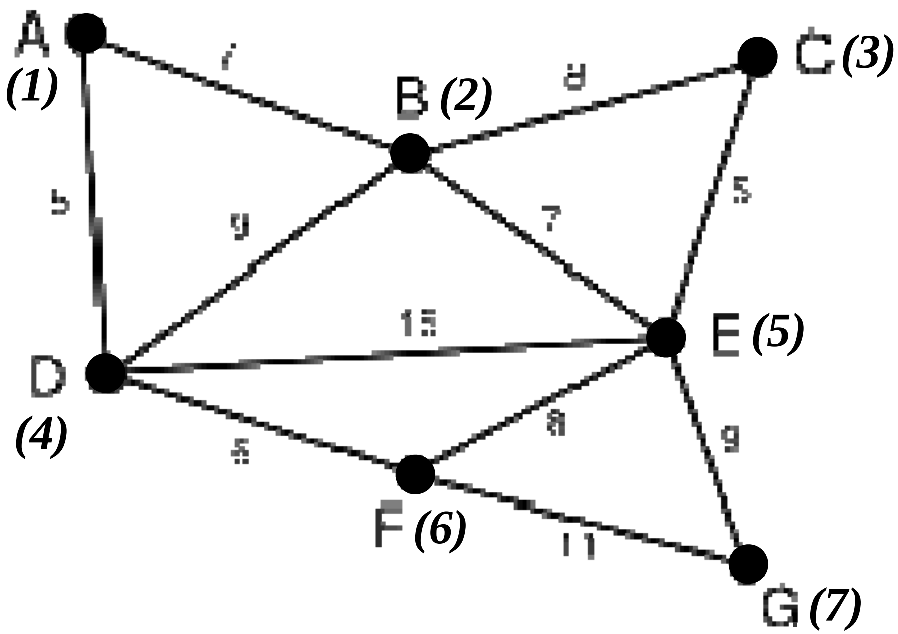
- Another version of Kruskal's algorithm
  - Checking cycle by labeling vertices
    - If two vertices have same labels, then adding the edge that connects the two vertices becomes a cycle

5.1.1 Kruskal's Algorithm

```
Tree Kruskal( Vertex V, Edge E )
{
    T = {};
    sort edges in E in ascending order;

    for each vertex v in the set V
             NEW_LABEL(v);

    for each (u,v) in E, in ascending order of weight {
             if LABEL(u) is not equal to LABEL(v) {
                 add the edge (u,v) to the tree T;
                 UNION( u, v );
             }
    }

    if ( |T| < n-1 )
       return NULL;
    return T;
}
```

5.1.1 Kruskal's Algorithm

- Ex)

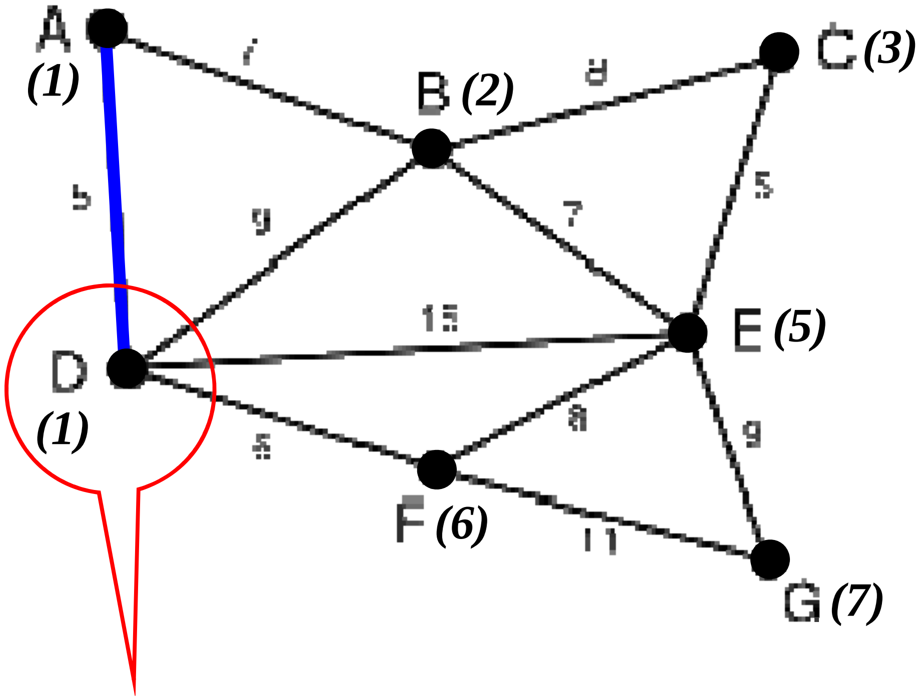


edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T

5.1.1 Kruskal's Algorithm

- Ex)



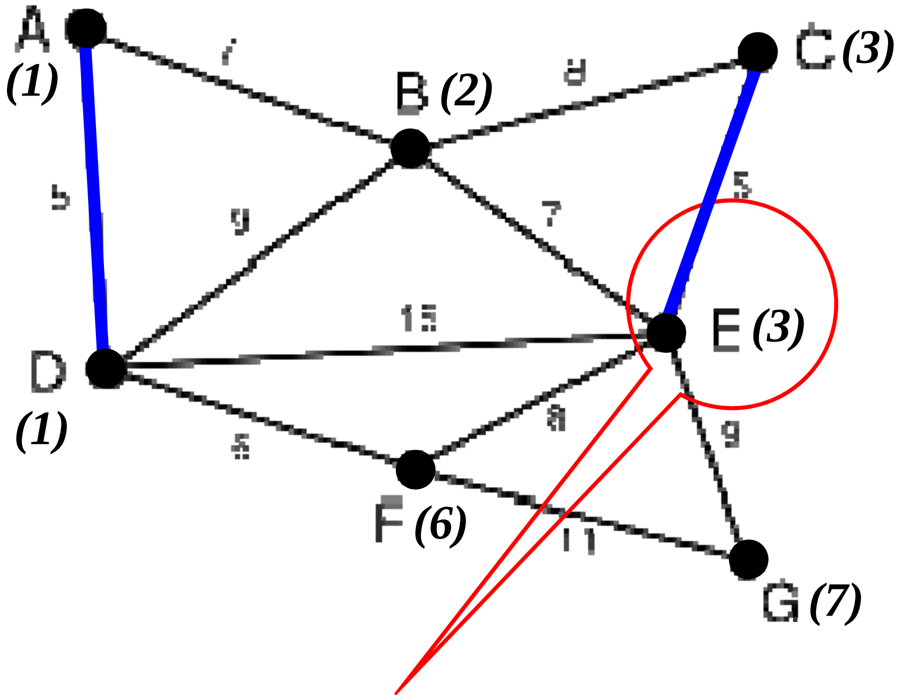
UNION (A, D) changes (4) to (1)

edges	***weights***
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)

5.1.1 Kruskal's Algorithm

- Ex)



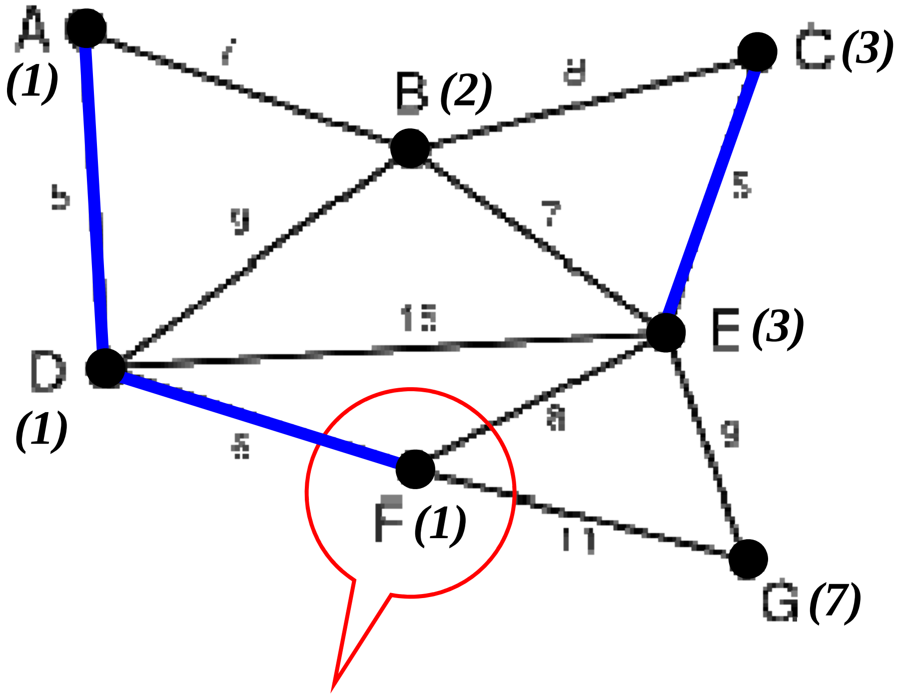
UNION (C, E) changes (5) to (3)

edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)

5.1.1 Kruskal's Algorithm

- Ex)



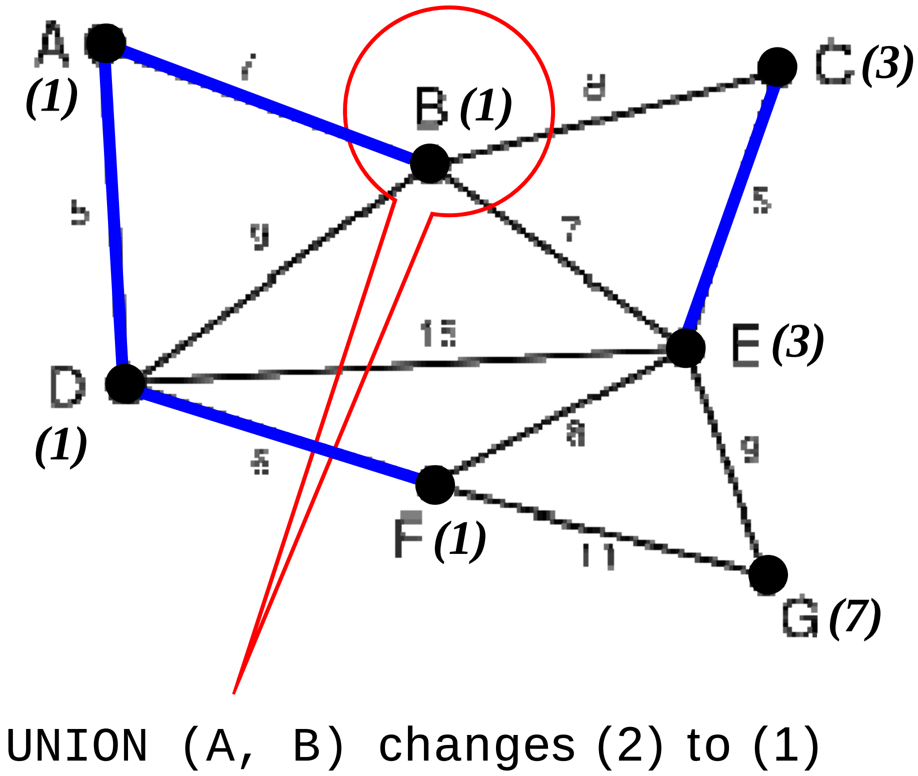
UNION (D, F) changes (6) to (1)

edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)

5.1.1 Kruskal's Algorithm

- Ex)

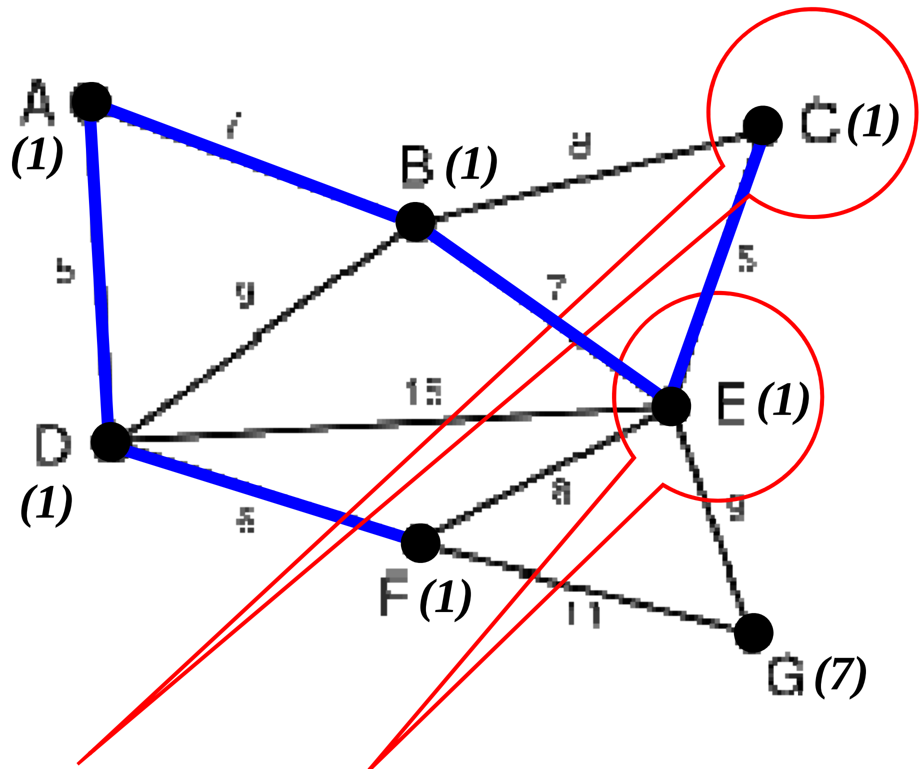


edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)

5.1.1 Kruskal's Algorithm

- Ex)



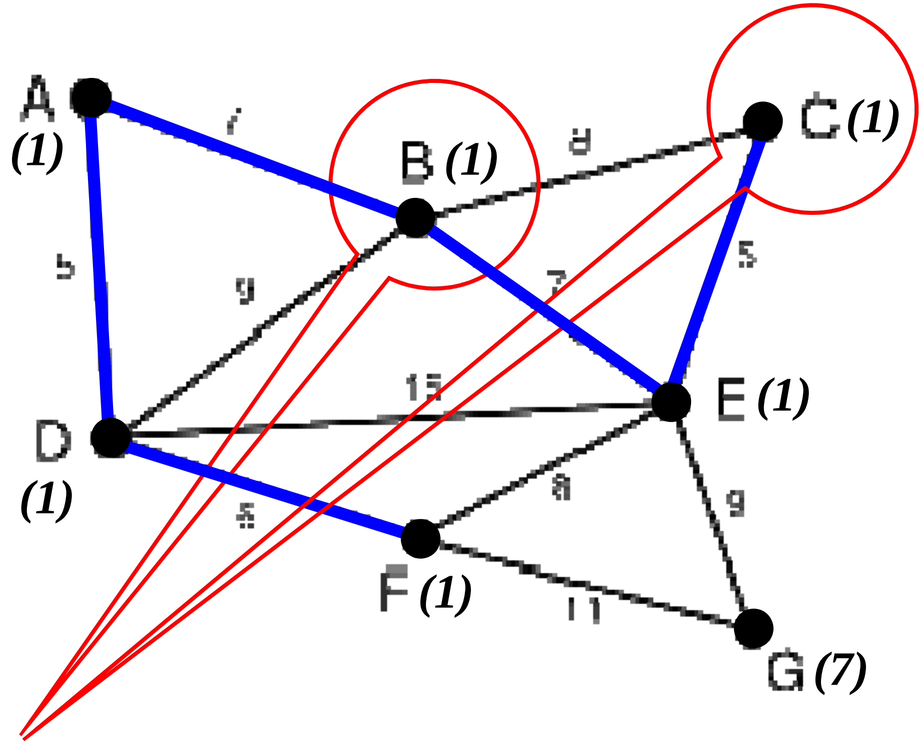
UNION (B, E) changes (3) to (1)

edges	***weights***
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)

5.1.1 Kruskal’s Algorithm

- Ex)



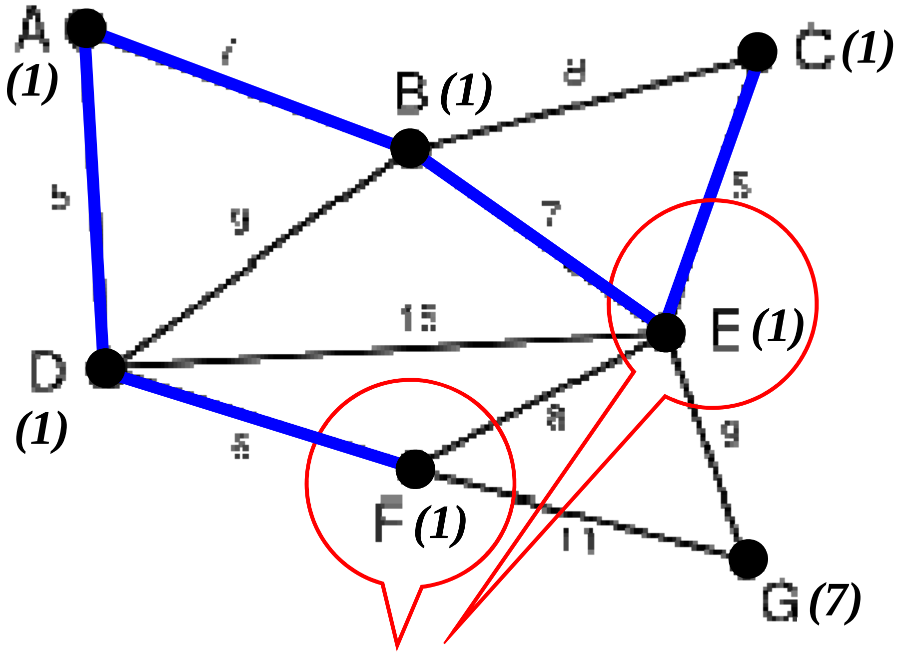
B & C canot be connected,
since both labels are same → cycle

edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)

3.1.1 Kruskal's Algorithm

- Ex)



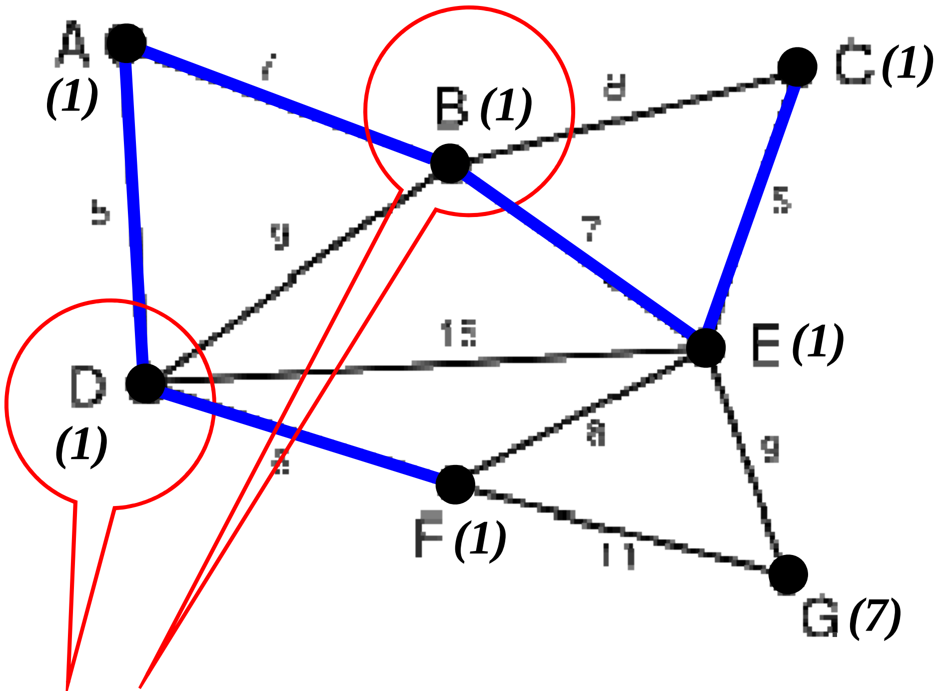
E & F canot be connected,
since both labels are same → cycle

edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)

5.1.1 Kruskal's Algorithm

- Ex)



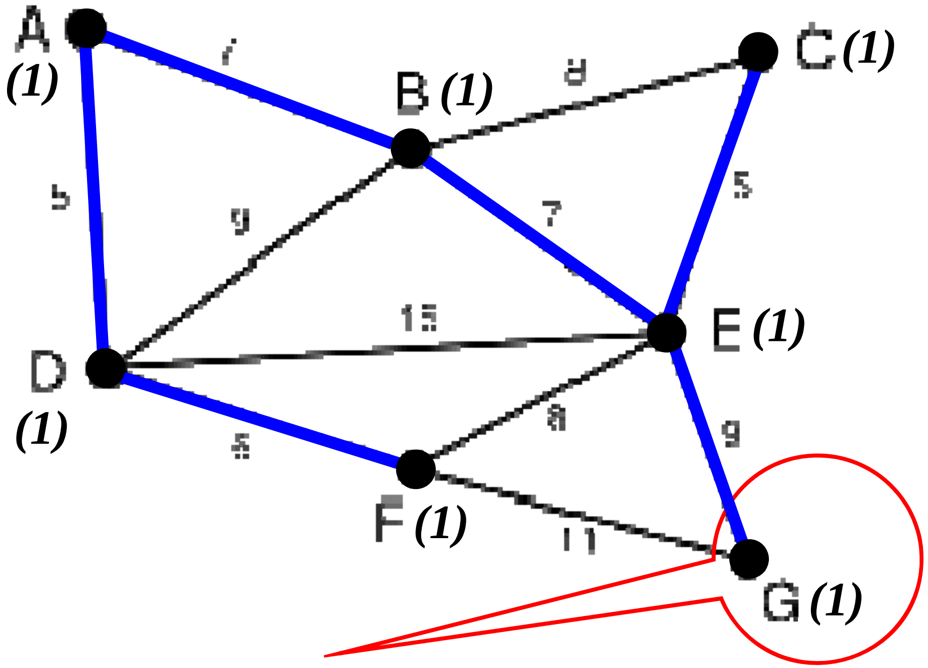
B & D canot be connected,
since both labels are same → cycle

edges	***weights***
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)

5.1.1 Kruskal's Algorithm

- Ex)



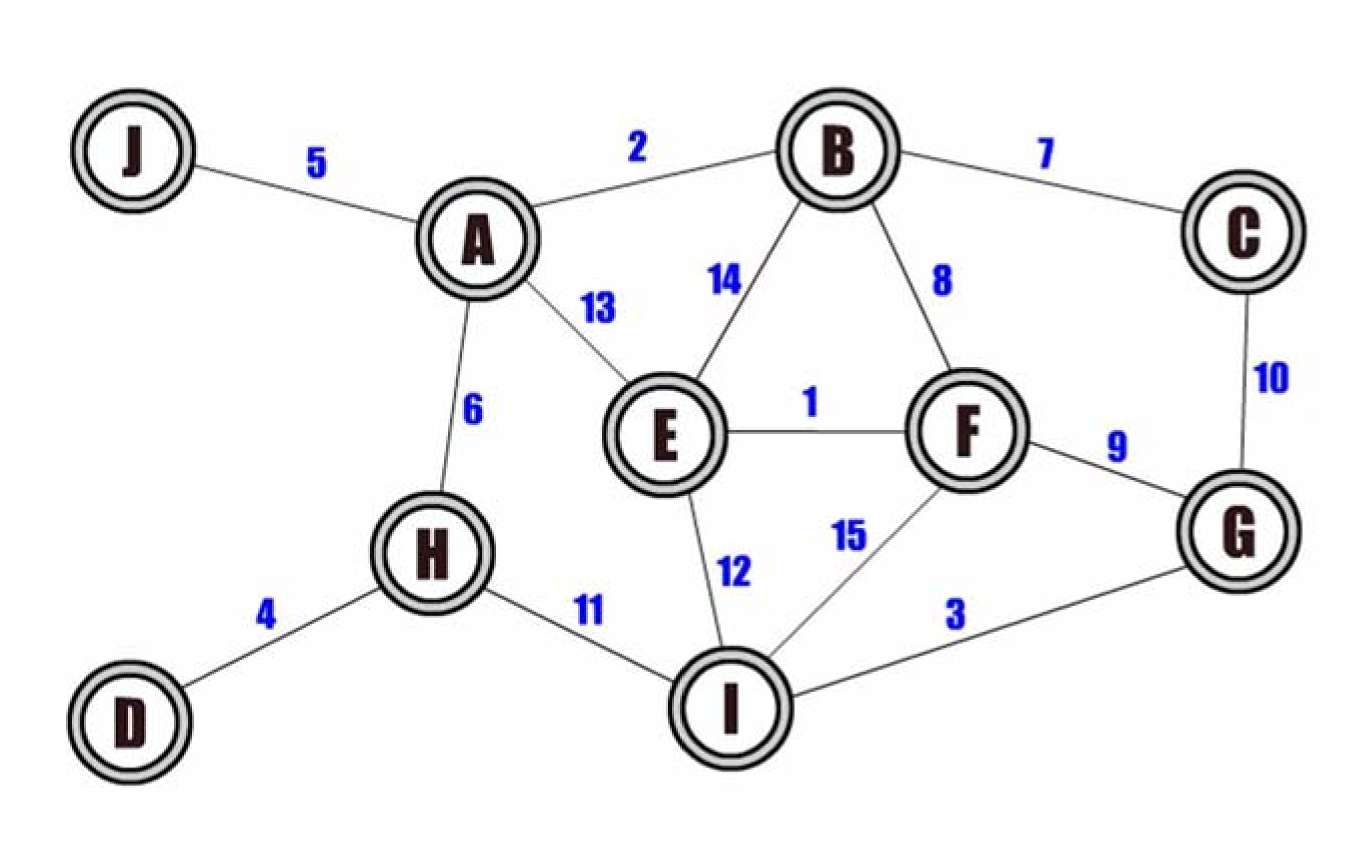
UNION (E, G) changes (7) to (1)

edges	*weights*
(A,D)	5
(C,E)	5
(D,F)	6
(A,B)	7
(B,E)	7
(B,C)	8
(E,F)	8
(B,D)	9
(E,G)	9
(F,G)	11
(D,E)	15

T
(A,D)
(C,E)
(D,F)
(A,B)
(B,E)
(E,G)

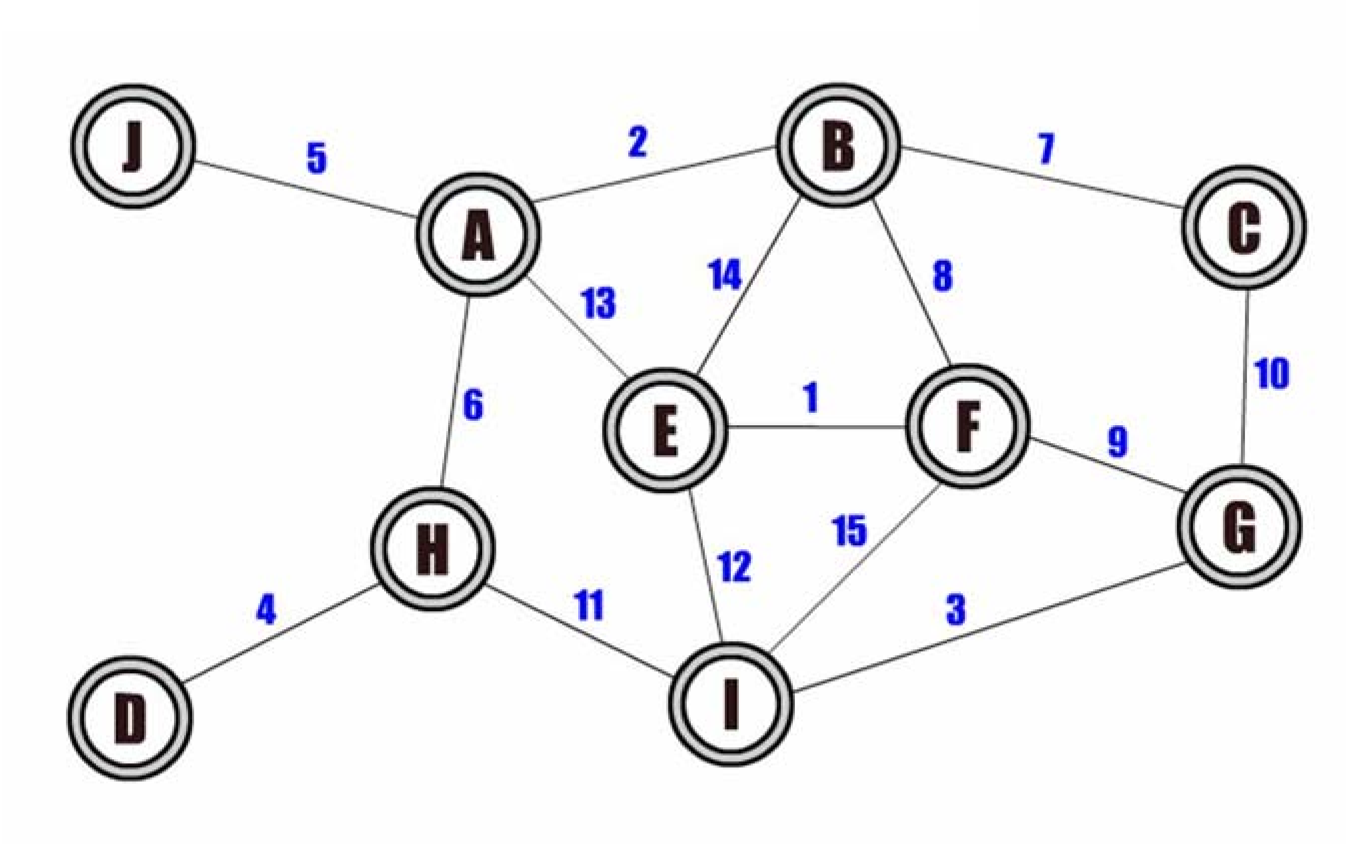
5.1.1 Kruskal's Algorithm

- Ex)



5.1.1 Kruskal's Algorithm

- Ex)



(E,F)	1
(A,B)	2
(G,I)	3
(D,H)	4
(A,J)	5
(A,H)	6
(B,C)	7
(B,F)	8
(F,G)	9
(C,G)	10
(H,I)	11
(E,I)	12
(A,E)	13
(B,E)	14
(F,I)	15

5.1.1 Kruskal’s Algorithm

다음 설명 중 옳지 않은 것을 모두 고르시오.

(a) Kruskal's algorithm은 edge를 하나씩 추가하면서 minimum cost spanning tree를 생성한다

(b) Kruskal's algorithm에서는 edge를 내림차순으로 정렬해야하기 때문에 O(m log m)의 계산 시간을 요구한다 (m: edge의 수)

(c) Kruskal's algorithm에서 정렬된 edge를 heap으로 관리하면 array로 관리하는 경우보다 성능을 향상시킬 수 있다

(d) n > m인 모든 그래프에서 Kruskal's algorithm은 NULL을 return한다