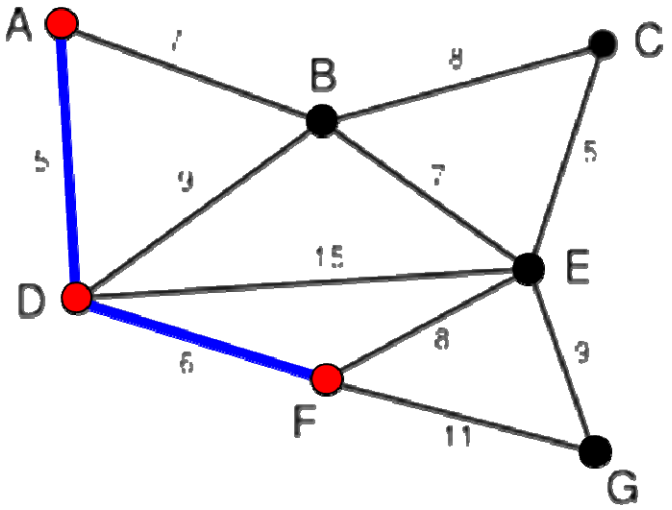
5.1.2 Prim's Algorithm

- Vertex-based algorithm
  - Basic strategy
    - Vertices on a Graph is classified into three categories:
      - U → Vertices in minimum-cost spanning tree (T)
      - Vertices incident to U
      - Other vertices



5.1.2 Prim's Algorithm

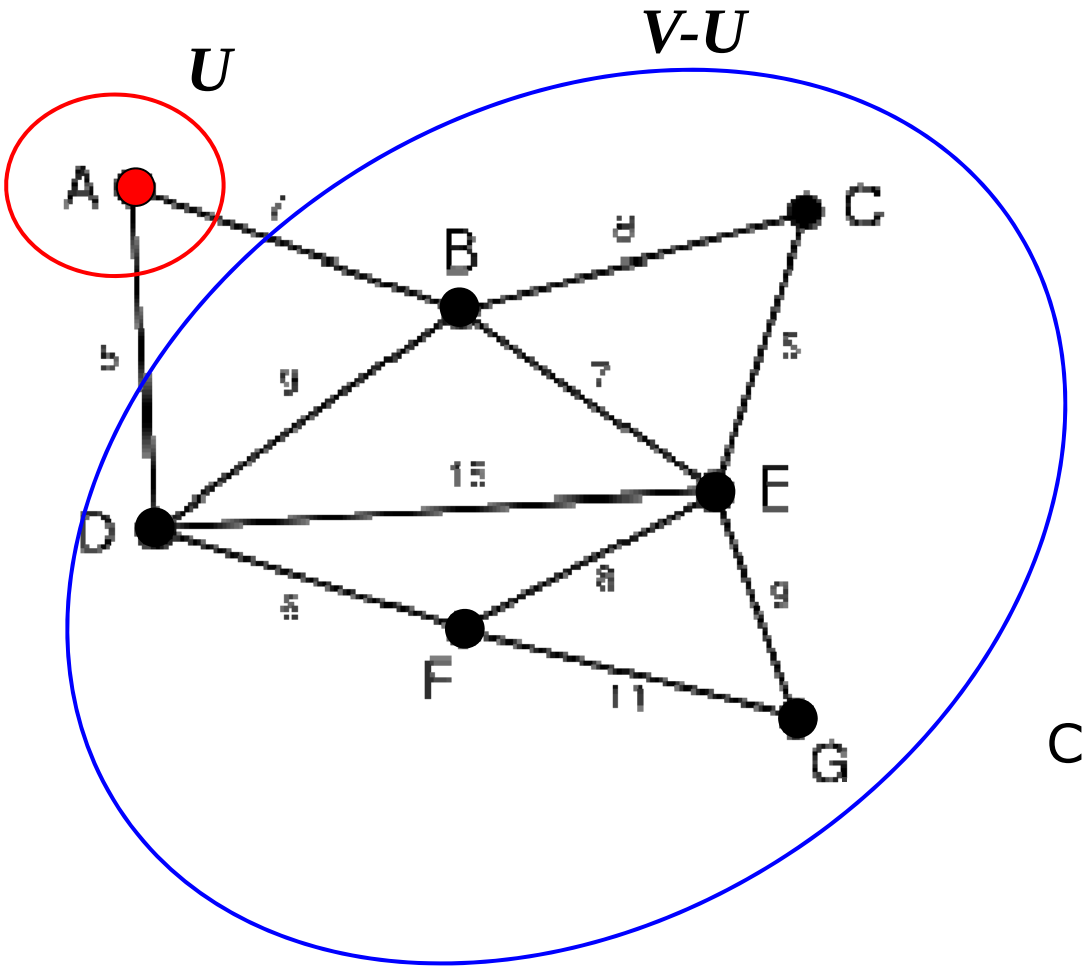
- Algorithm
  - Initially, set T as Φ.
  - Find all the vertices incident to the vertices in T.
  - Find the minimum-weight edge among the edges that connect a vertex belongs to T and a vertex that does not belong to T.
  - Add the vertex on the edge to T.
  - Repeat this process until all vertices belong to U.

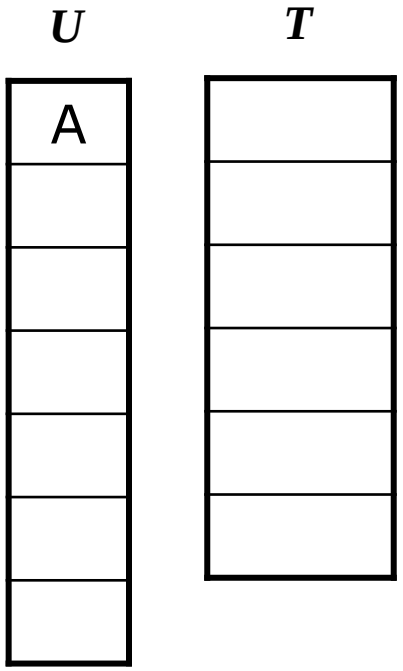
5.1.2 Prim's Algorithm

```
Tree Prim( Vertex V, Edge E )
{
    Vertex *U;
    vertex u,v;
    T = { };
    U = { A };
    while (U != V) {
        (u,v) = lowest cost edge with u in U and v in (V - U);
        if ( (u,v) is NULL )
            return NULL;
        T += (u,v);
        U += v;
    }
    return T;
}
```

5.1.2 Prim's Algorithm

- Ex)

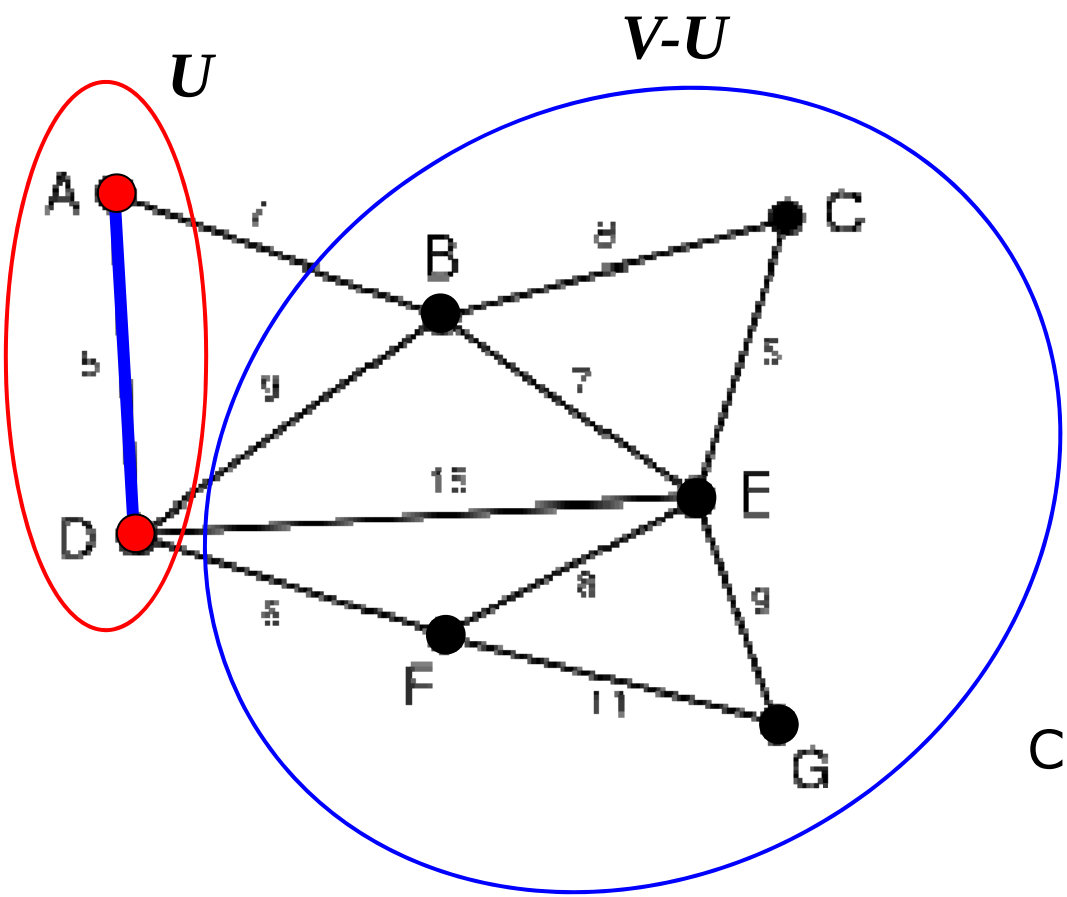




Candidate edges: (A, B): 7
(A, D): 5

5.1.2 Prim's Algorithm

- Ex)

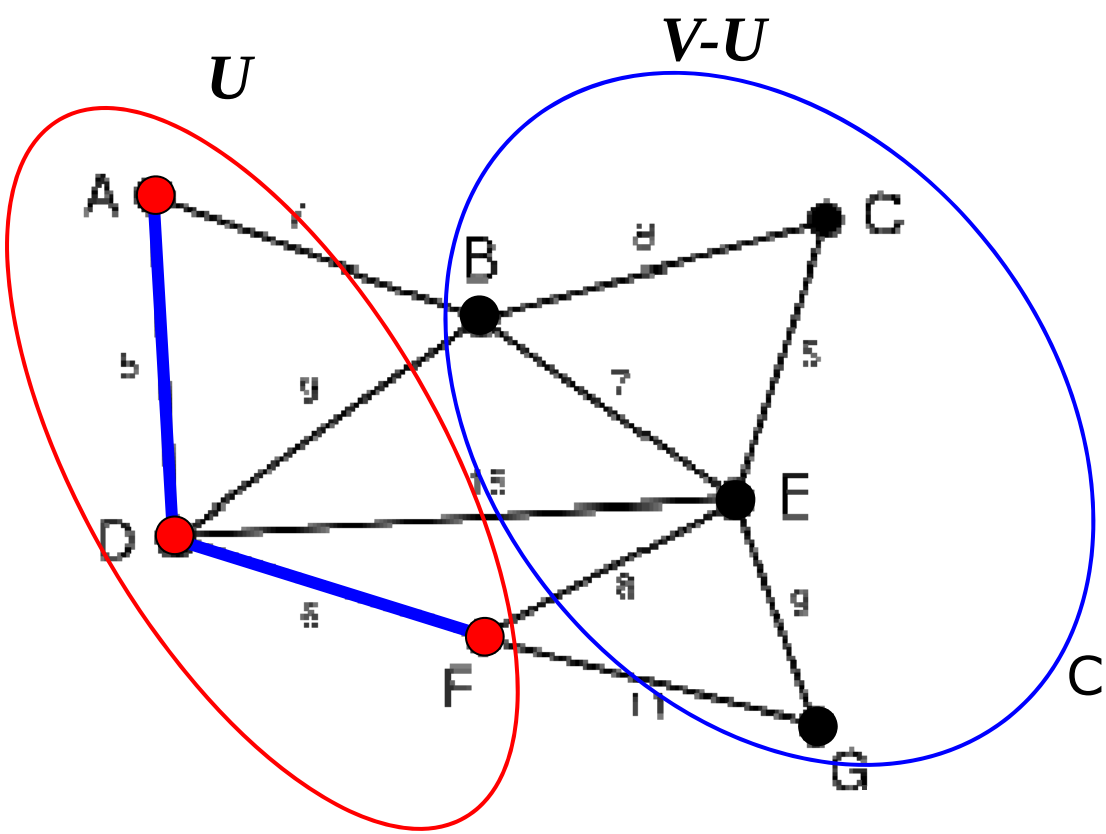


U	T
A	(A,D)
D	

Candidate edges: (A, B): 7
(D, B): 9
(D, E): 15
(D, F): 6

5.1.2 Prim's Algorithm

- Ex)

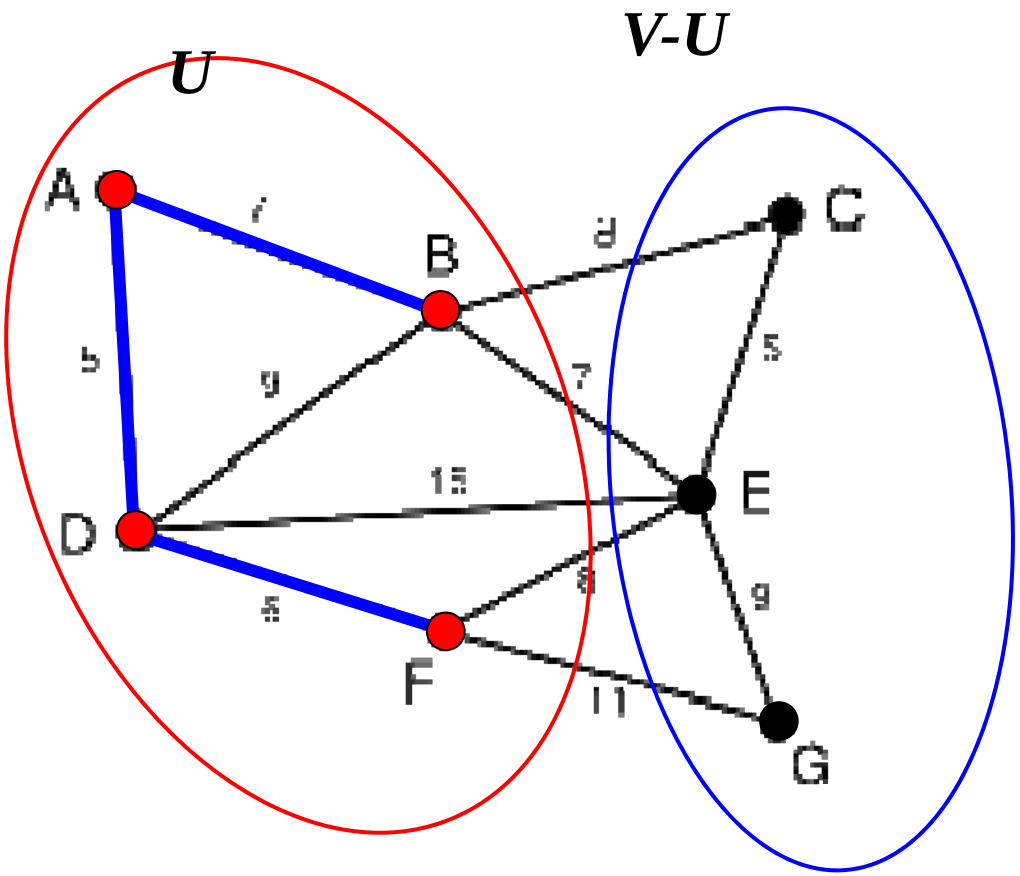


U	T
A	(A,D)
	(D,F)
D	
F	

Candidate edges: (A, B): 7
(D, B): 9
(D, E): 15
(F, E): 8
(F, G): 11

5.1.2 Prim's Algorithm

- Ex)

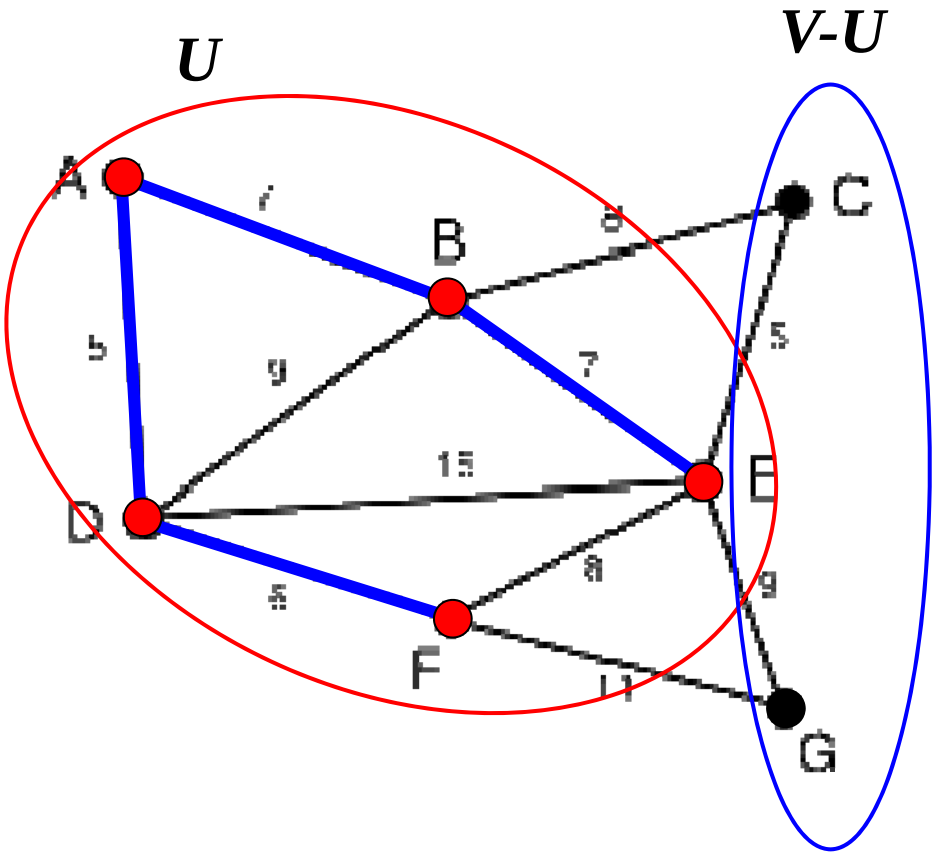


U	*T*
A	(A,D)
B	(D,F)
	(A,B)
D	
F	

Candidate edges: (B, C): 8
(B, E): 7
(D, E): 15
(F, E): 8
(F, G): 11

5.1.2 Prim's Algorithm

- Ex)

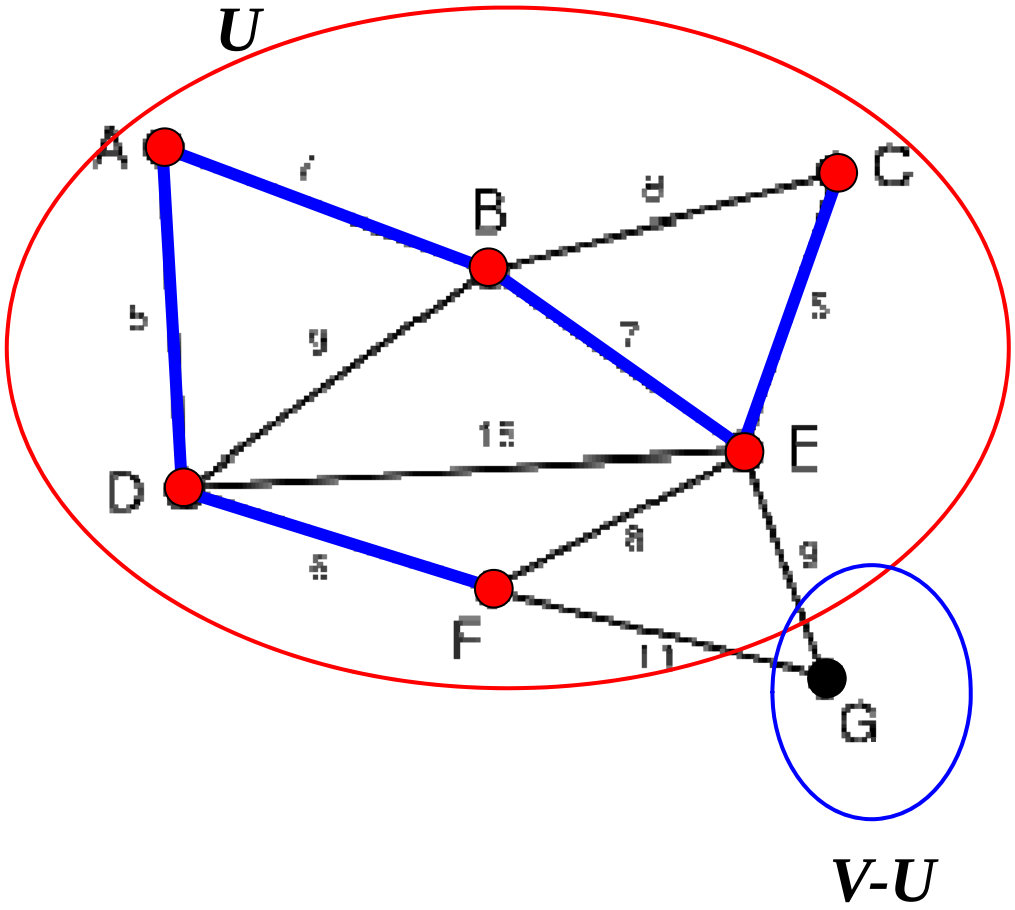


U	*T*
A	(A,D)
B	(D,F)
	(A,B)
D	(B,E)
E	
F	

Candidate edges: (B, C): 8
(E, C): 5
(E, G): 9
(F, G): 11

5.1.2 Prim's Algorithm

- Ex)

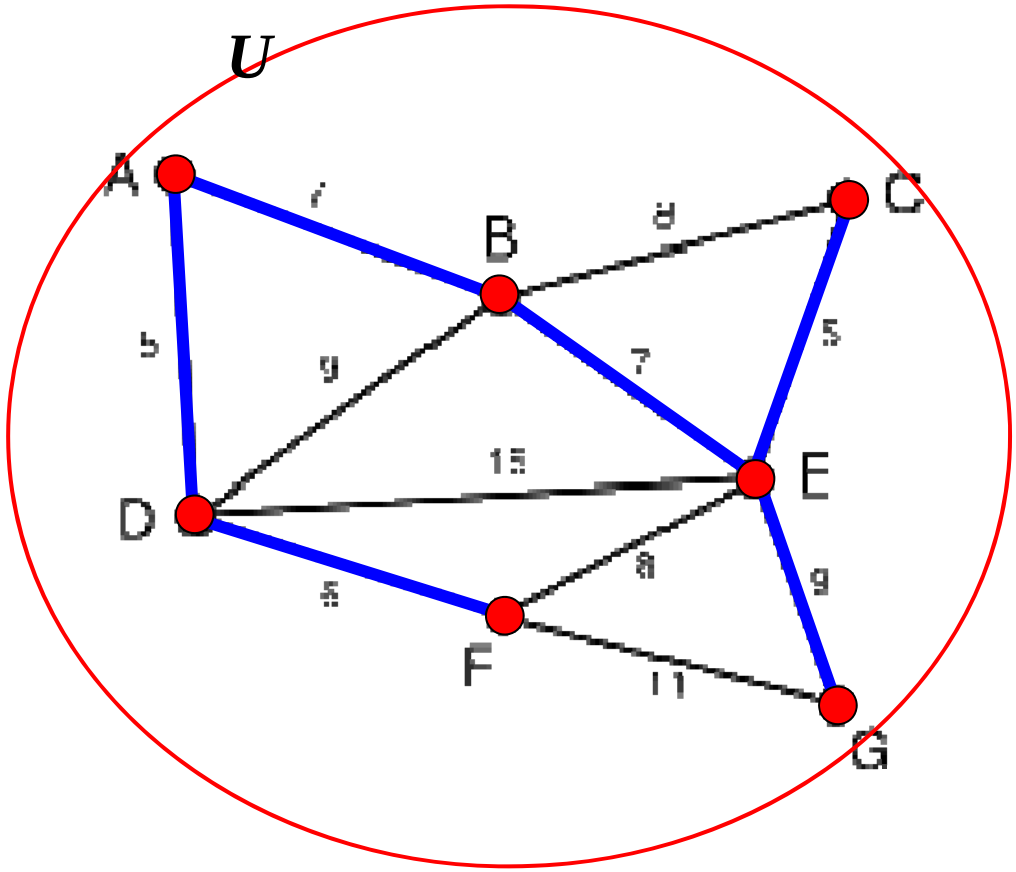


U	T
A	(A,D)
B	(D,F)
C	(A,B)
D	(B,E)
E	(C,E)
F	

Candidate edges: (E, G): 9
(F, G): 11

5.1.2 Prim's Algorithm

- Ex)



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

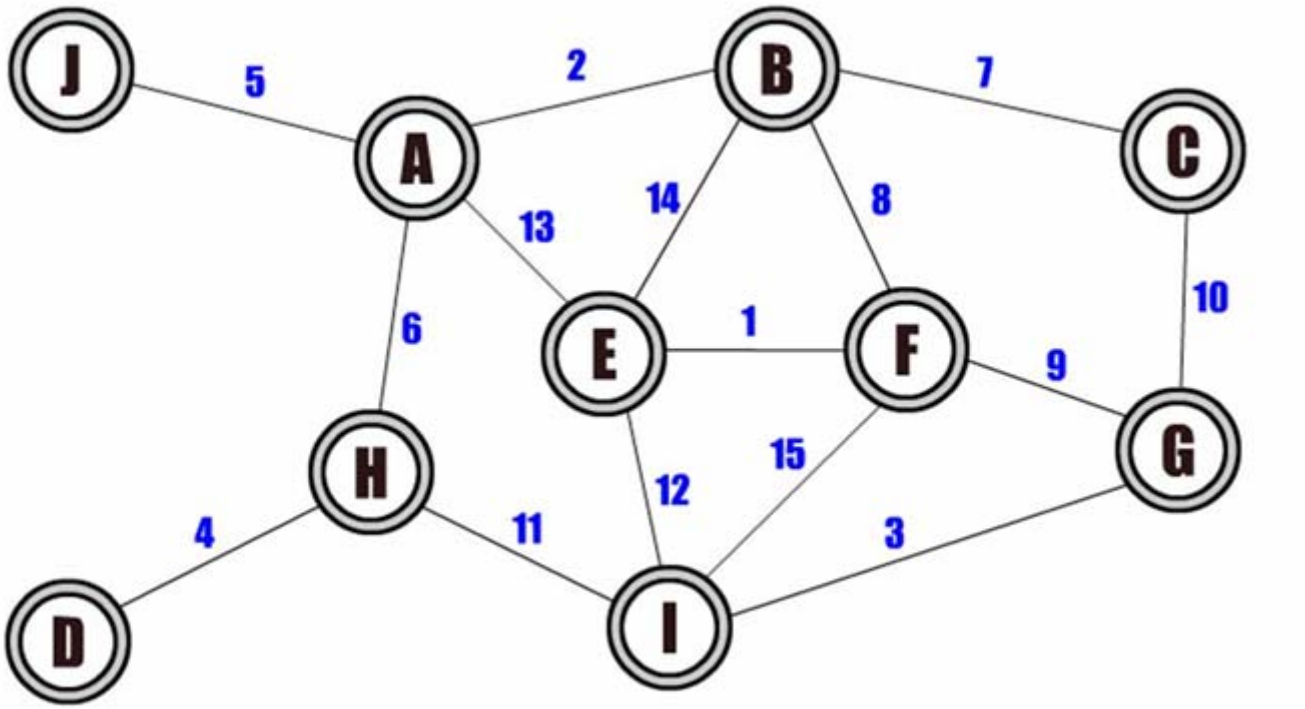
5.1.2 Prim's Algorithm

- Performance analysis
  - All edges in the graph are processed → O(m)
  - All edges are managed in a heap → O(m log m)

```
Tree Prim( Vertex V, Edge E )
{
    Vertex *U;
    vertex u,v;
    T = { };
    U = { A };
    while (U != V) {
        (u,v) = lowest cost edge with u in U and v in (V - U);
        T += (u,v);
        U += v;
    }
    return T;
}
```

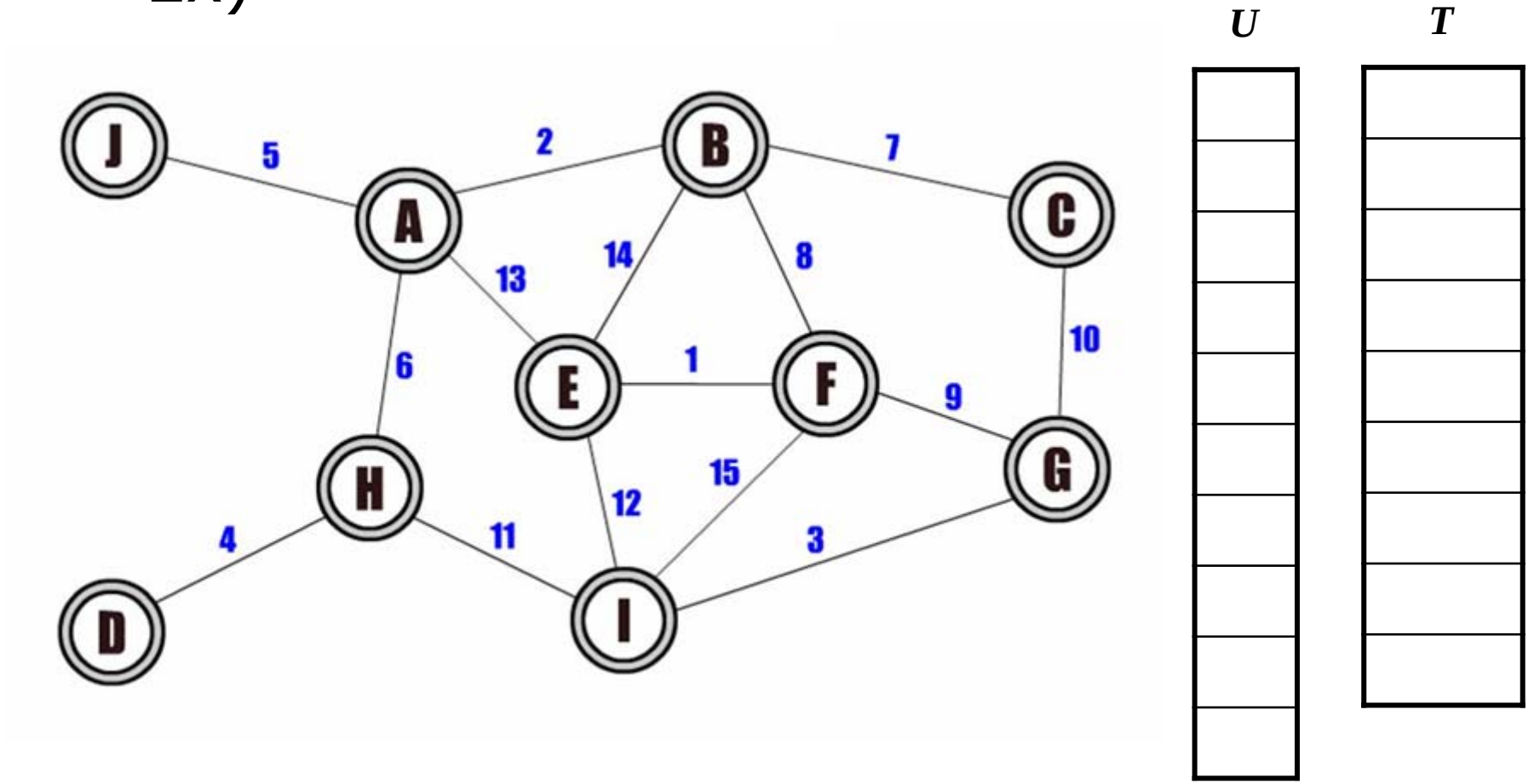
5.1.2 Prim's Algorithm

- Ex)



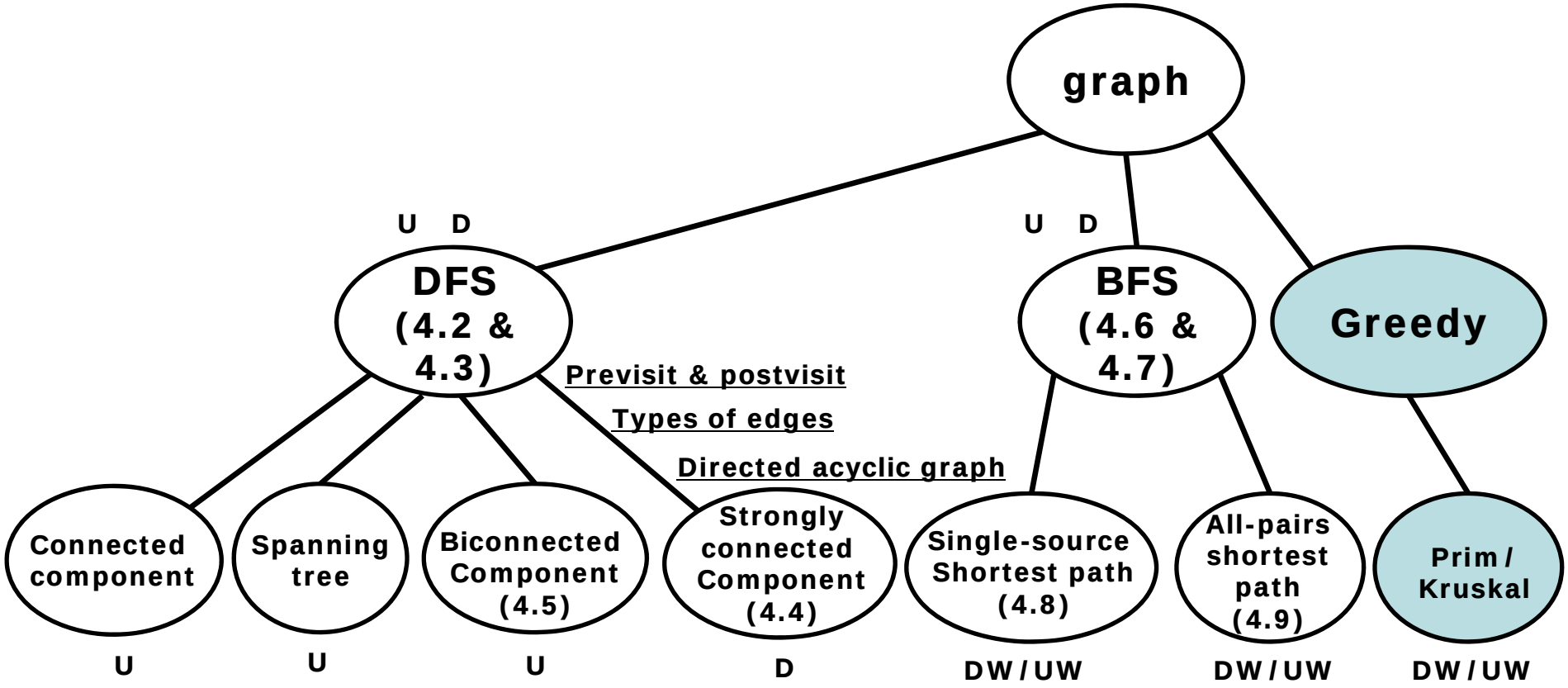
5.1.2 Prim's Algorithm

- Ex)



5.1.2 Prim's Algorithm

(5) Classification of graph algorithms



All about graph

Type	Purpose	Operations	Performance
DFS	Traverse all vertices	Visiting all vertices & visiting all edges	O(n) + O(m)
SCC	Finding SCC	DFS on G^R and G	O(DFS)
BFS	Traverse all vertices	Visiting all vertices & visiting all edges	O(n) + O(m)
Dijkstra	Single source shortest path	Visiting all edges & managing queue	O(n)*get_min + O(m)*modify → $O(n^2)$ or O(n log n)
Floyd	All pairs shortest path	Incrementing k	$O(n^3)$
Kruskal (Greedy)	Minimum-cost spanning tree	Sorting edge & selecting (cycle checking)	O(m log m) + $O(n^2)$ → O(m log m)
Prim (Greedy)	Minimum-cost spanning tree	Managing heap of edges	O(m log m)
MultiStage (Dynamic)			

5.1.2 Prim's Algorithm

다음 설명 중 옳은 것을 모두 고르시오.

(a) Prim 알고리즘은 cycle check를 하지 않기 때문에 Kruskal 알고리즘보다 성능이 더 좋다

(b) Prim 알고리즘은 우선순위 큐를 이용해서 edge를 관리한다

(c) Prim 알고리즘은 n > m인 모든 graph에 대해서 NULL을 return한다

(d) Prim 알고리즘에서 while-loop은 O(m)번 수행된다