Graph

(kruskal,prim)

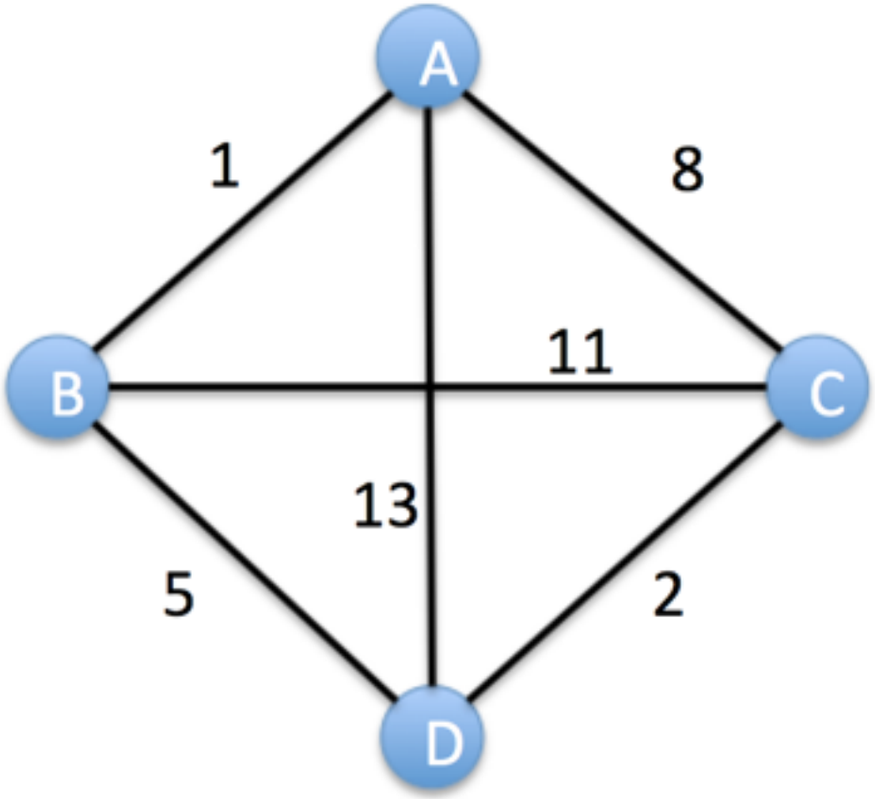
Review

신장트리(spanning tree)

그래프의 모든 정점을 연결하는 서브트리

정점이 n개일 때,

신장트리는n개의 정점과 n-1개의 간선으로 구성



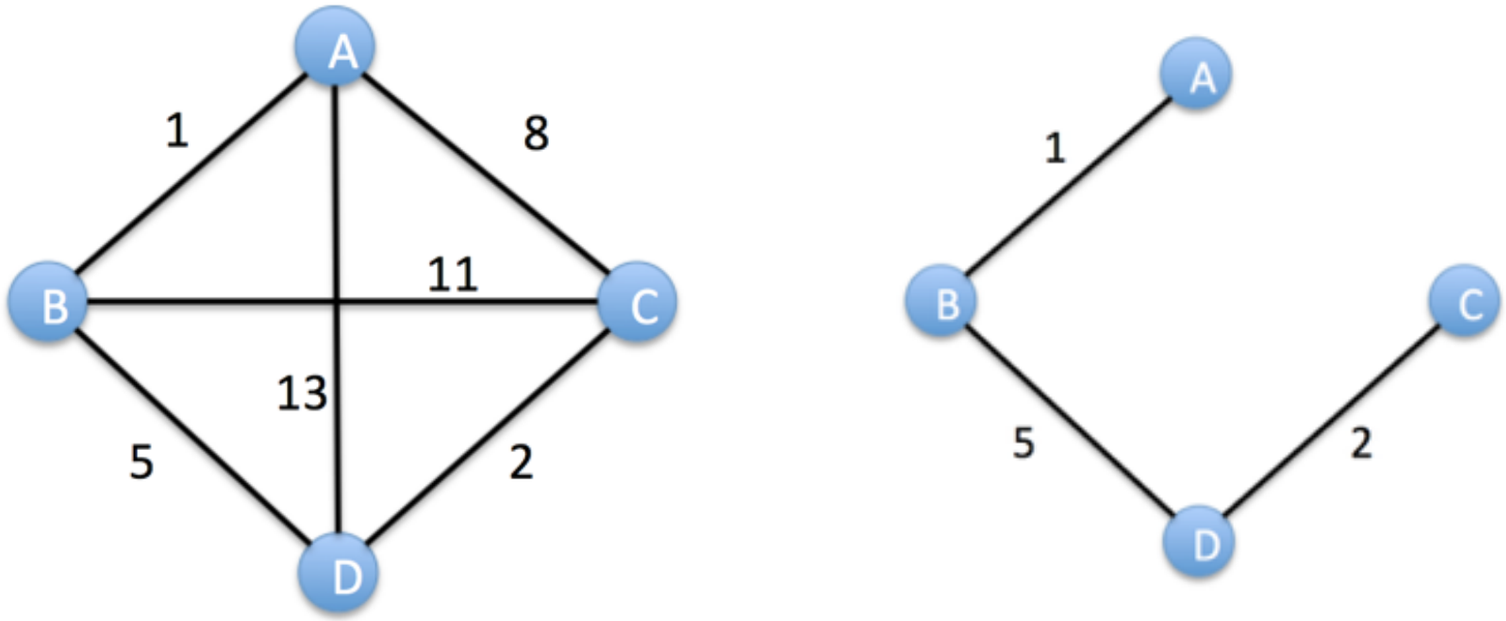
신장트리의 비용(Cost)

: 신장트리 간선들의 가중치의 합

Review

최소비용 신장트리(MST:minimum spanning tree)

가장 적은 비용의 신장트리



MST를 구하는 알고리즘

1) Kruskal's

2) Prim's

3) Sollin's

Kruskal & Prim 알고리즘

공통점

갈망법은 당장 눈앞의 최소비용을 선택하기 때문에 최선의 선택이 아닐 수 있음
다이나믹 프로그래밍은 전체적인 과정에서 최선의 비용을 찾아내는 것

갈망법(**Greedy Method**)?

당장 눈앞의 최소 비용을 선택해서 결과적으로 최적의 Solution을 찾음

√ 그래프 내에 있는 간선만 사용

√ |V(G)|=n 일 때 정확히 n-1개의 간선만 사용

√ 사이클을 생성하는 간선 사용 불가

Kruskal & Prim 알고리즘

Kruskal

√ **간선**위주의 알고리즘

√ 사이클을 이루는지 항상 확인해야함

Union, Find로 계속 확인

√ 일반적인 경우 비교적 유리하다.

√ 크루스칼은 간선을 weight 기준으로 정렬하는 과정이 오래 걸림

시간 복잡도 : O(E logE)

Prim

√ **정점**위주의 알고리즘

√ 사이클 검사 하지 않아도 됨

√ Dense한 그래프의 경우 유리

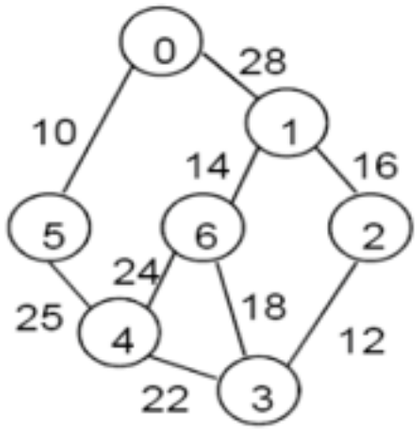
√ 최소 거리의 정점을 찾는 부분에서 자료구조의 성능에 영향을 받음

V는 정점
E는 간선

시간 복잡도 : O(V^2+E) → O(E logV)

$O(n^2)$이거나 그보다 빠른 구현

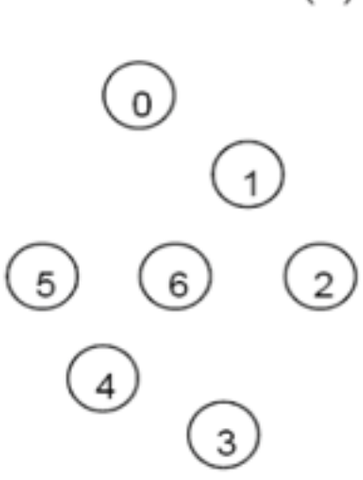
Kruskal 알고리즘



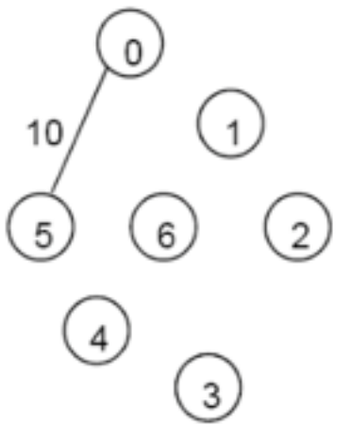
(a)

간선들을 비용이 증가하는 순서로 정렬 여기서 오래걸림

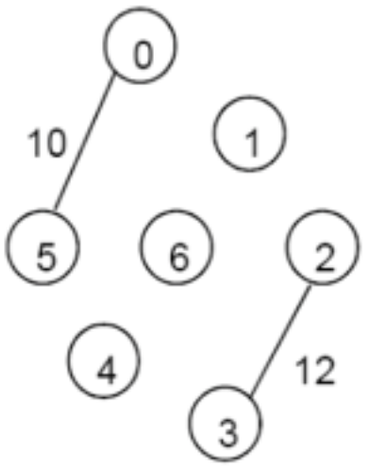
사이클 형성하지 않으며, 적은 비용을 가지는 간선 선택



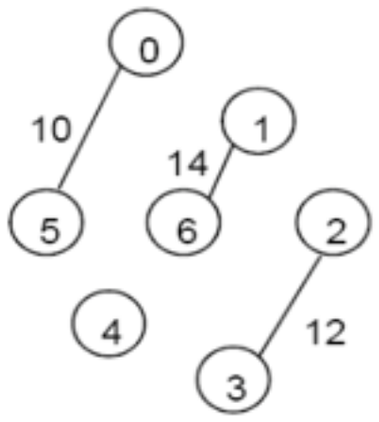
(b)



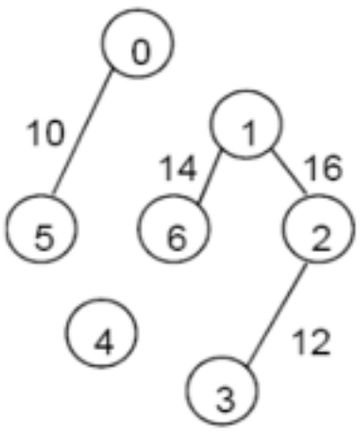
(c)



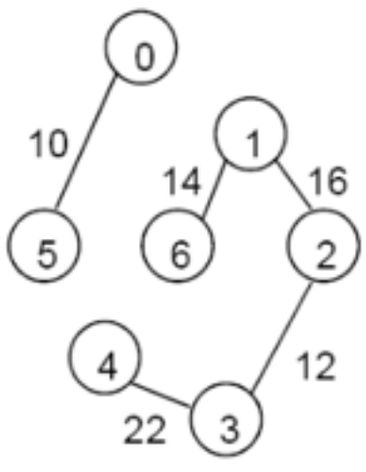
(d)



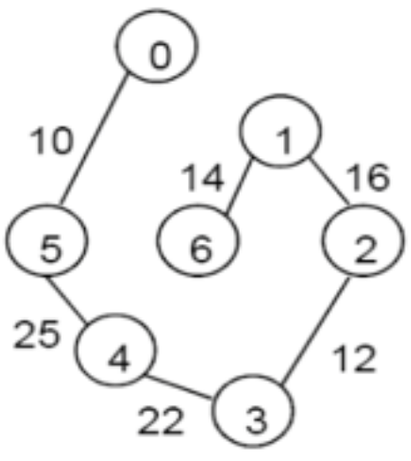
(e)



(f)



(g)



(h)

Kruskal 알고리즘

[min heap] e : 간선의 집합

√ heap 구성할 때: O(e) 가장 적은 간선을 뽑을 때 이 힙이 제일 빠른 자료구조!

√ 최소비용 **간선 찾고 삭제**할 때: O(loge)

√ **사이클 형성하지 않는 것** 확인할 때:

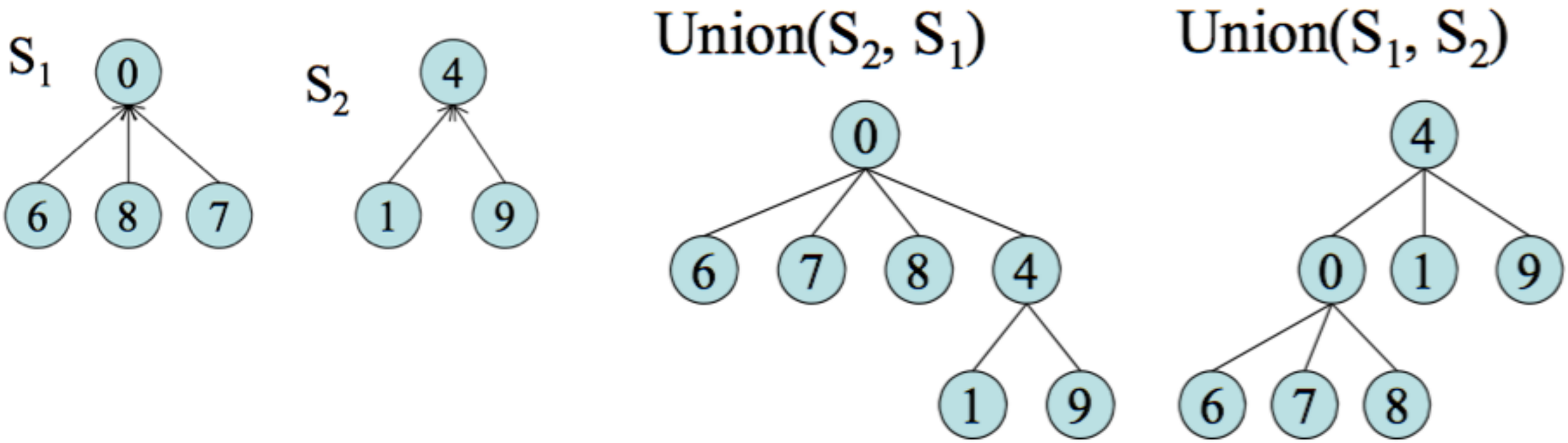
union-find set operation - less than O(loge)

총 시간 복잡도: **O(e loge)**

Kruskal 알고리즘

[Union & Find] 두 간선이 같은 집합에 속해있는지를 확인하기 위해 사용

Union 원소 x와 y가 속해있는 집합을 입력으로 받아 2개의 집합을 합집합으로 함



Weighting Rule for union(i,j)

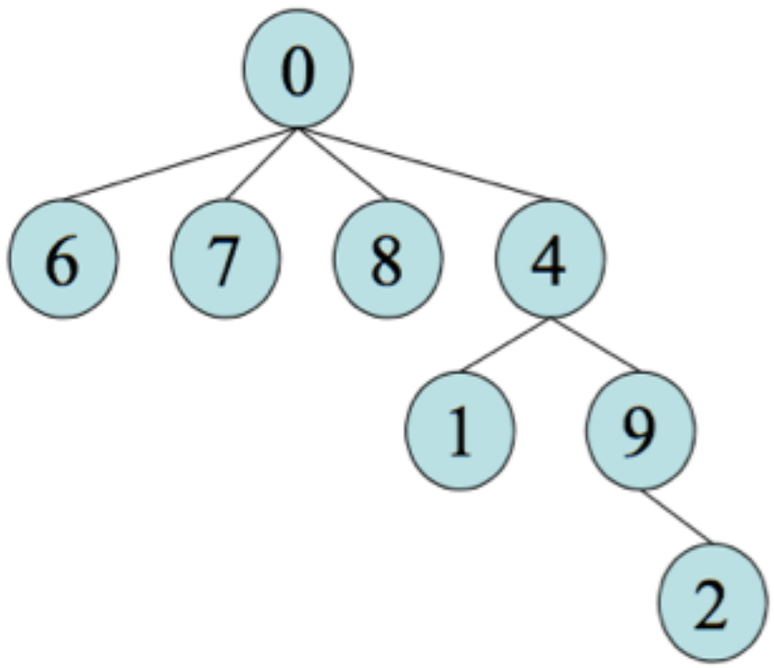
노드 숫자가 더 많은 것이 부모가 됨

Kruskal 알고리즘 Union & Find

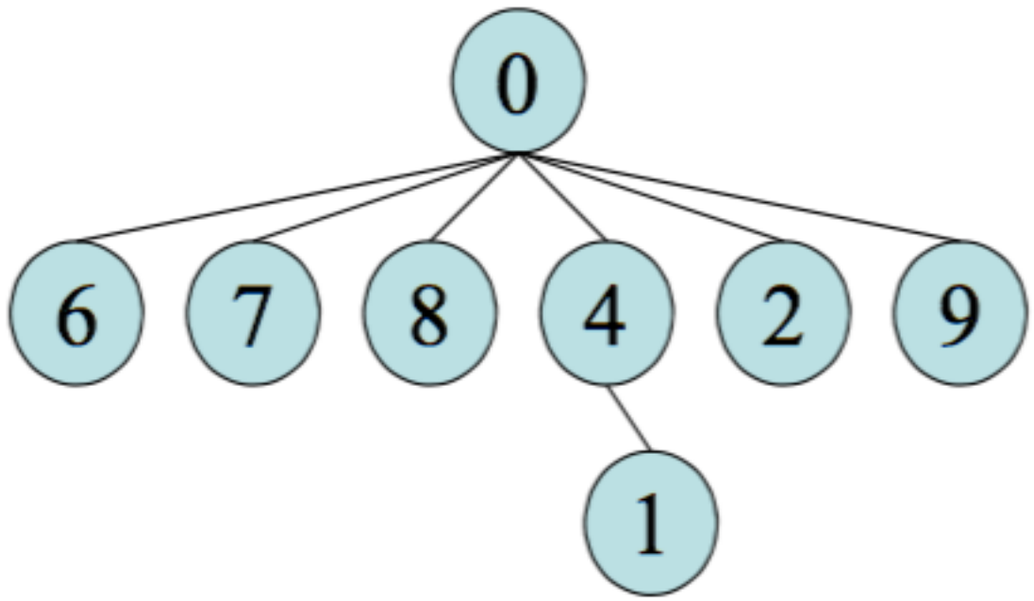
Find

원소 x가 속해있는 집합(대표 원소) 반환

Find(2) = 0



collapsingFind(2) = 0



root값 찾고 root의 자식노드로 넣음

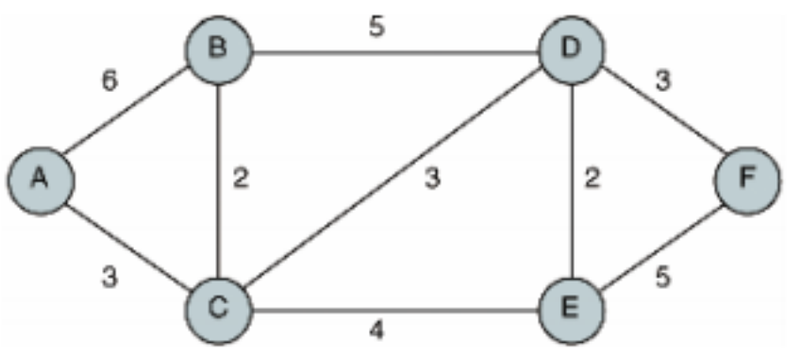
Prim 알고리즘

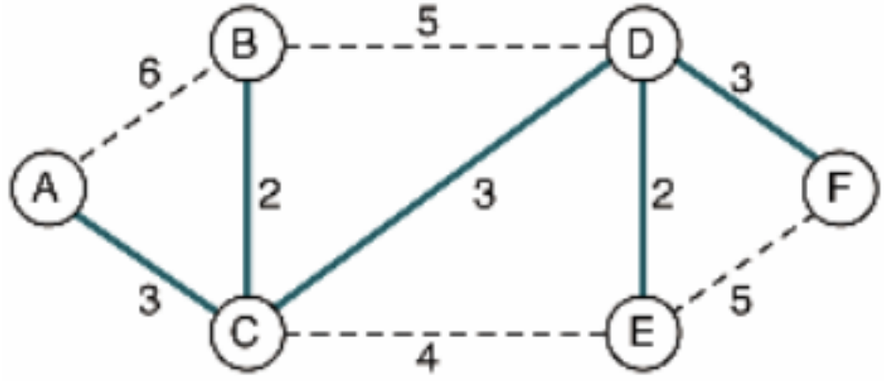
정점 O(A)에서 시작

시작 정점부터 출발하여 신장트리를 조금씩 확장시킴

만들어진 **신장트리에 가장 인접한 정점**들 중 방문을 하지 않은

최소 비용의 이음선을 찾아 연결된 정점 선택



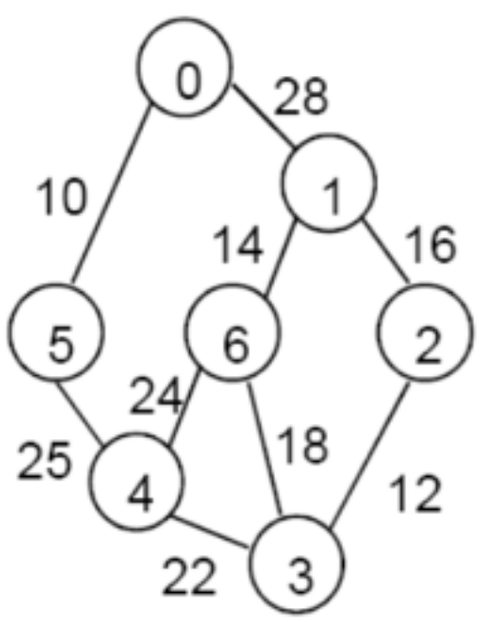


배열을 사용하면 O(V^2), 이진 힙을 사용하면 O(ElgV)

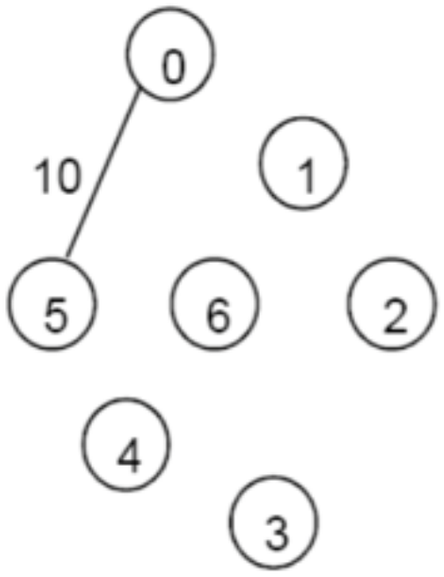
총 시간 복잡도: **O(V^2+E) → O(E lgV)**

$O(V^2)$이거나 그보다 빠른 구현

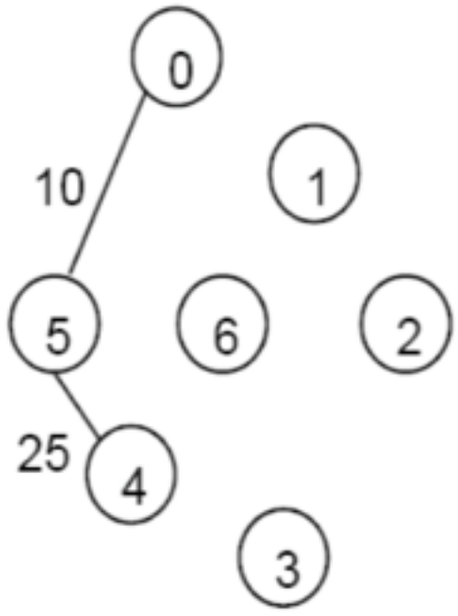
Prim 알고리즘



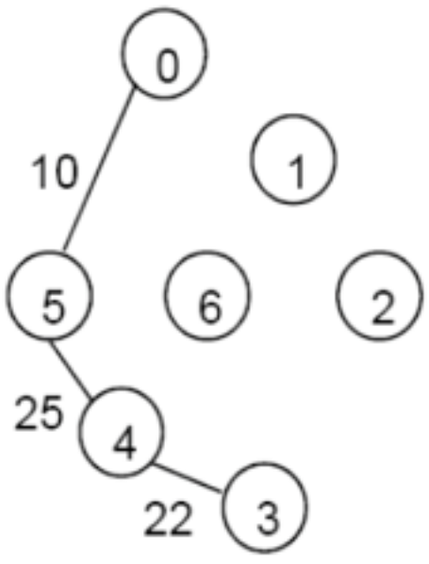
(a)



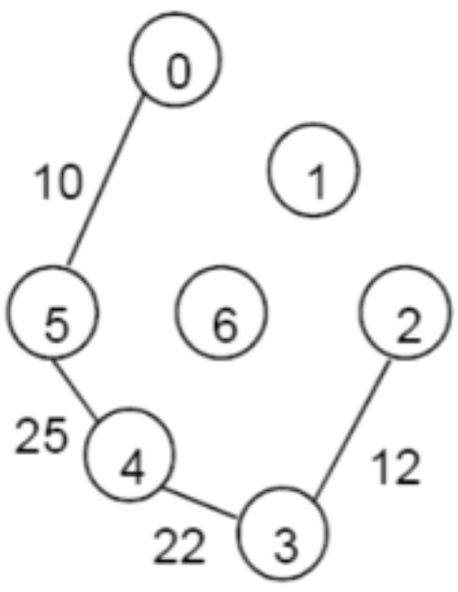
(b)



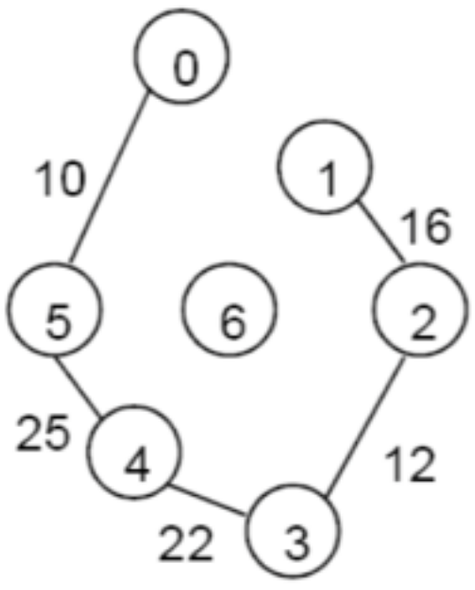
(c)



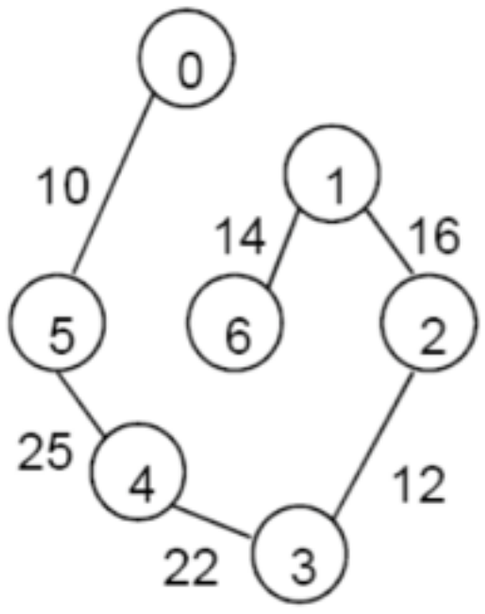
(d)



(e)



(f)



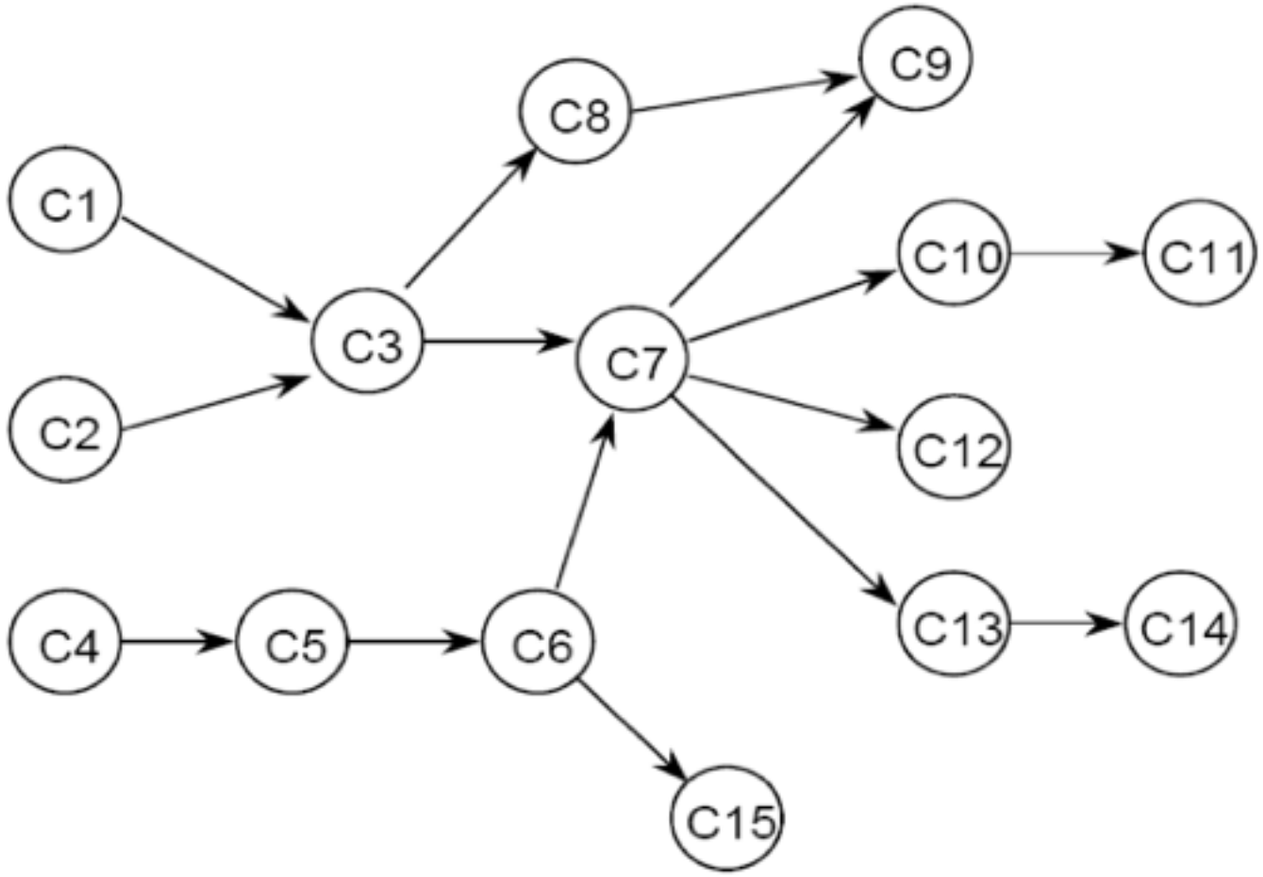
Activity on vertex(AOV) network

간선- 작업 사이의 선행관계

정점 i로부터 정점 j로 방향경로가 존재하면,

정점 i는 정점 j의 선행자(predecessor) , j는 i의 후속자(successor)

<i,j> E(G)이면, 정점 i는 정점 j의 직속선행자(immediate predecessor)



AOV network representing courses as vertices and edges as prerequisites

위상순서(Topological order)

그래프의 정점들의 선형 순서

i가 j의 선행자 -> 위상순서에서 i가 j앞에 있는 선형순서

위상정렬(Topological sort)

DFS 이거 써서 위상정렬 할 수 있음

1) **선행자를 가지지 않는 정점**을 구하여 순서에 추가

2) 그 정점과 그것으로부터 나오는 모든 간선들을 네트워크에서 삭제

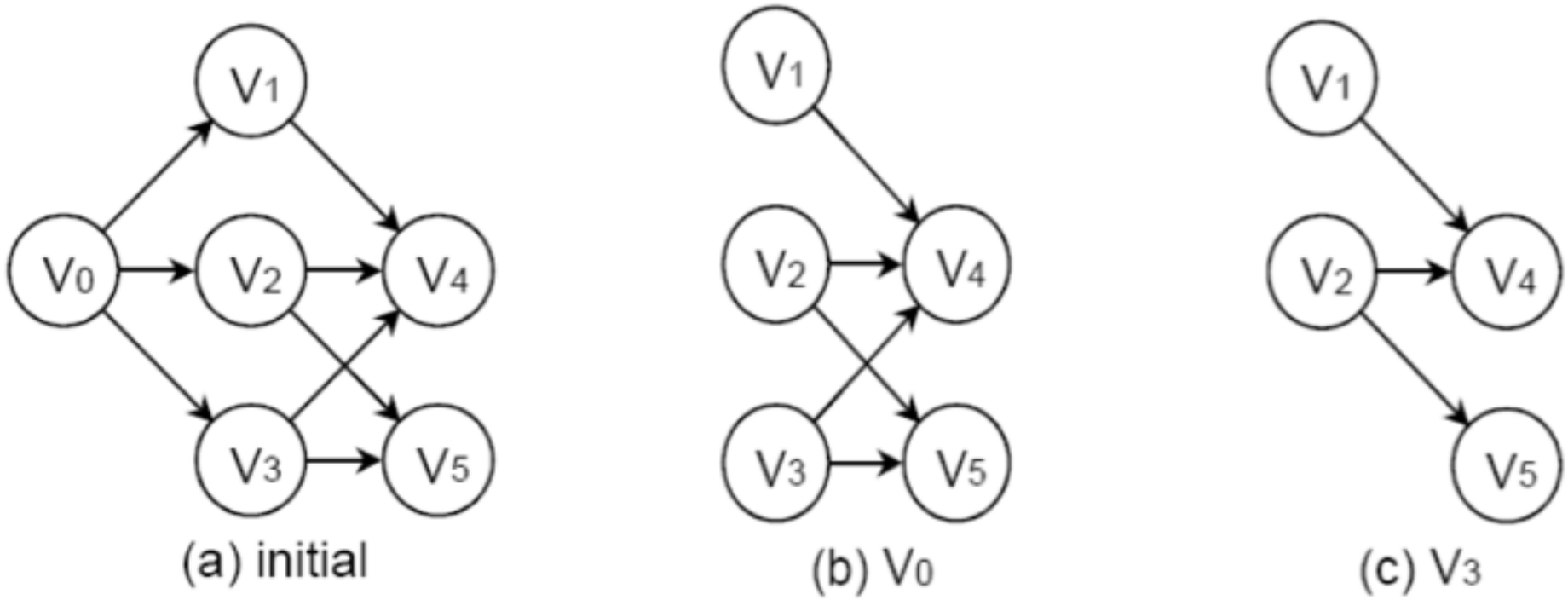
- 다음이 만족할 때까지 위의 1,2과정을 반복

- 모든 남아있는 정점들이 선행자를 가진다 -> 네트워크가 사이클을 가짐

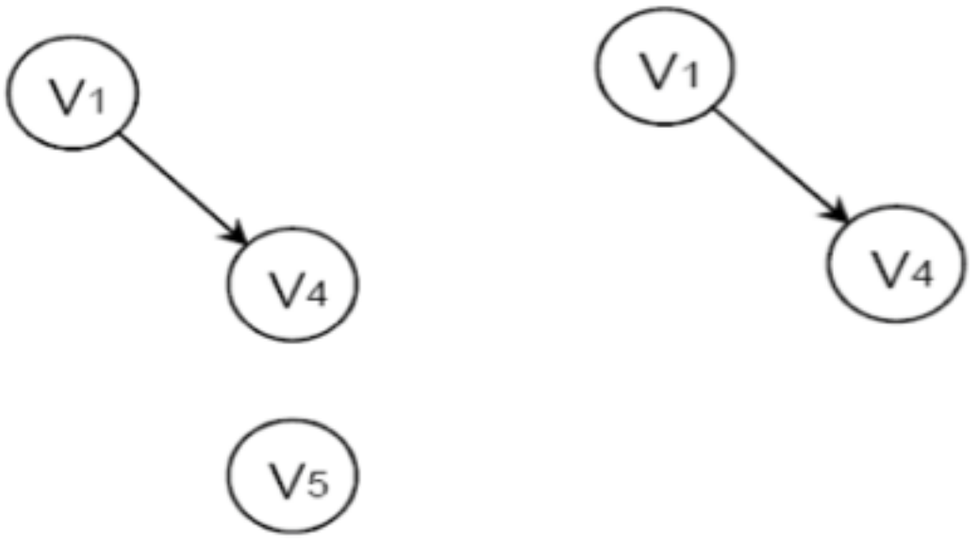
위상정렬(Topological sort)

DFS

Stack사용



(a) initial (b) V_0 (c) V_3



(d) V_2 (e) V_5 (f) V_1 (g) V_4

위상정렬(Topological sort)