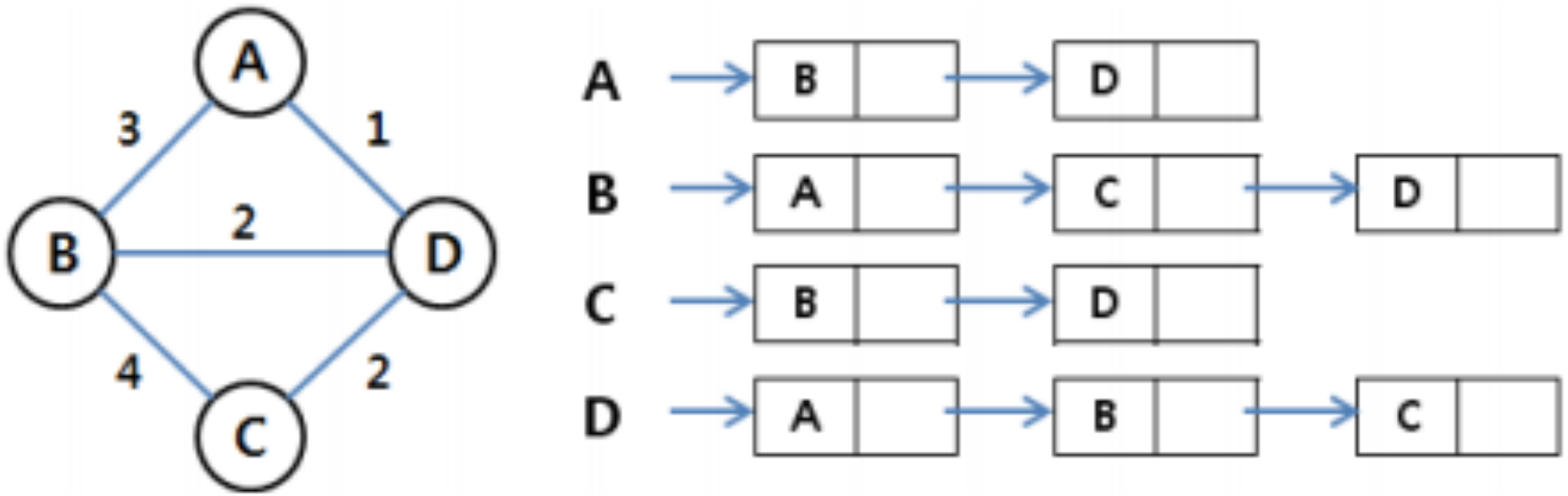
Graph

- ver2

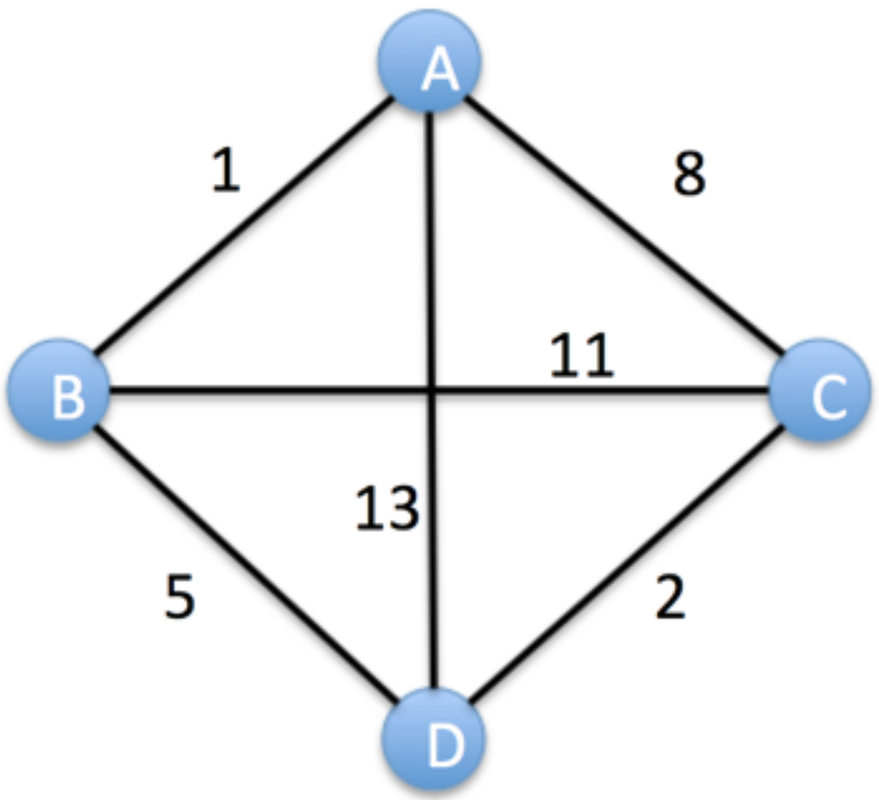
가중치 그래프



신장트리 그래프 내에서 모든 정점을 포함하는 트리

조건

- 그래프 내의 **모든 정점을 포함**하고 있어야 함
- 사이클을 포함하고 있으면 안됨
- 간선에 **가중치 값** 있음

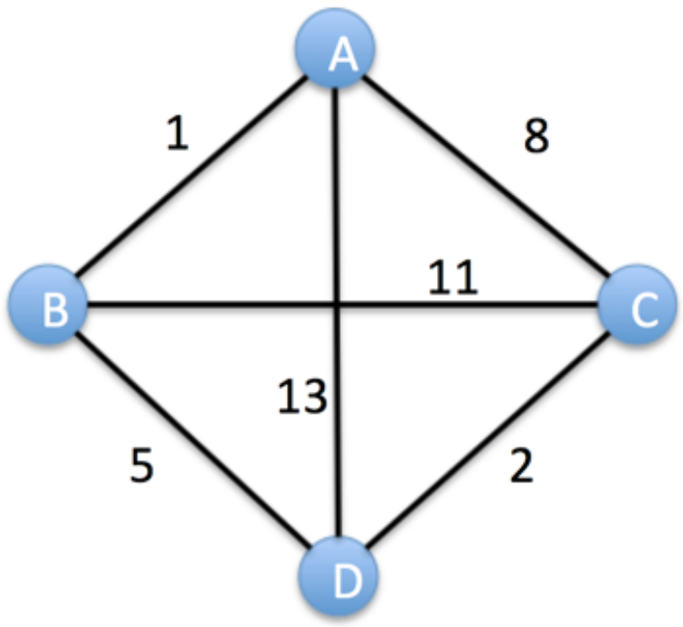


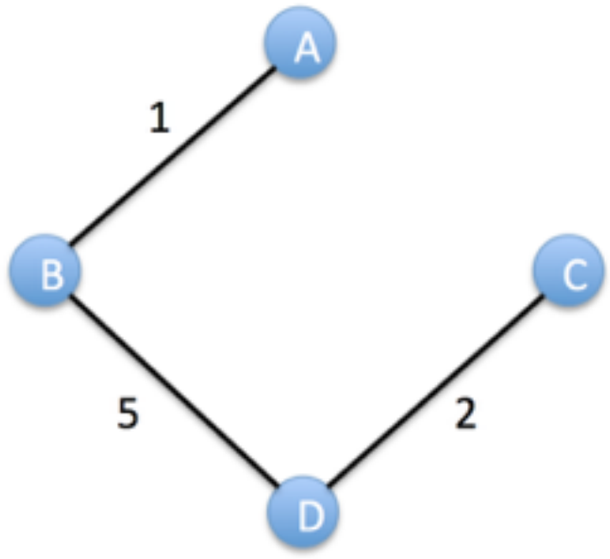
최소비용 신장트리(MST:minimum spanning tree)

모든 정점들을 가장 적은 수의 간선과 비용으로 연결하는 신장트리

간선에 **가중치 값** 있음

weight값이 최소가 되는 신장트리

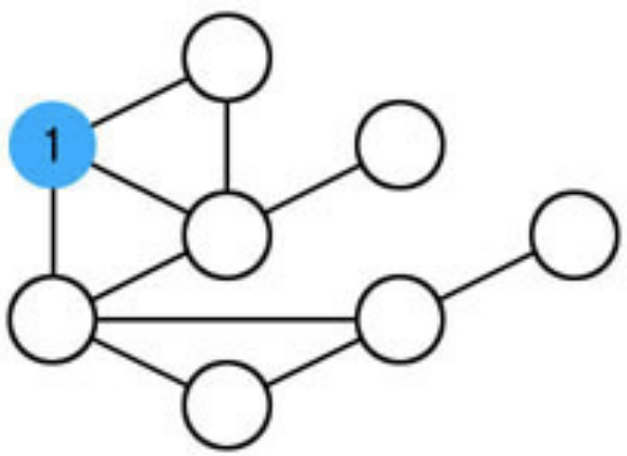


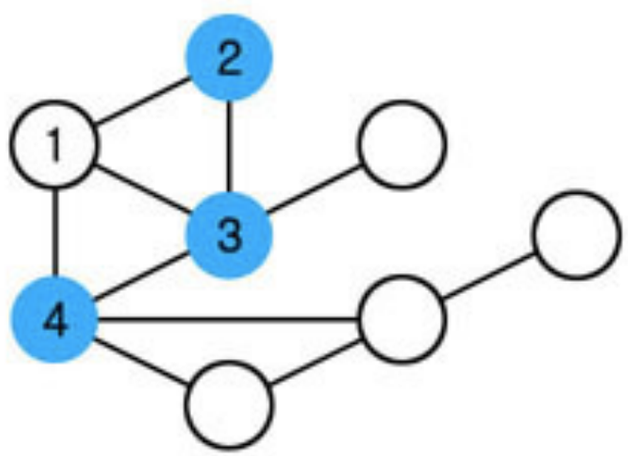


Kruskal , Prim 알고리즘 사용

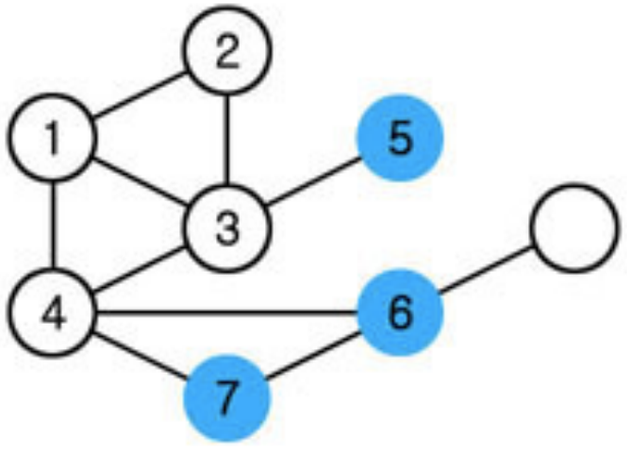
너비우선탐색

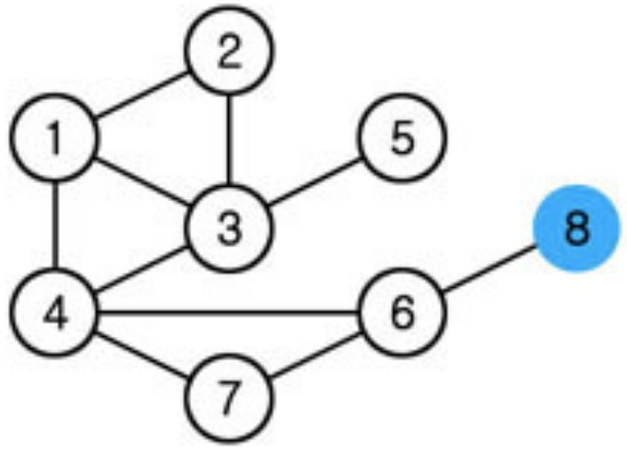
‘Queue’ 사용

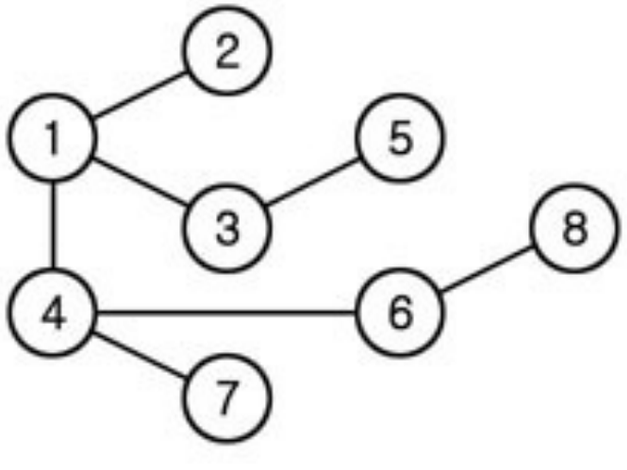




[BFS신장트리]

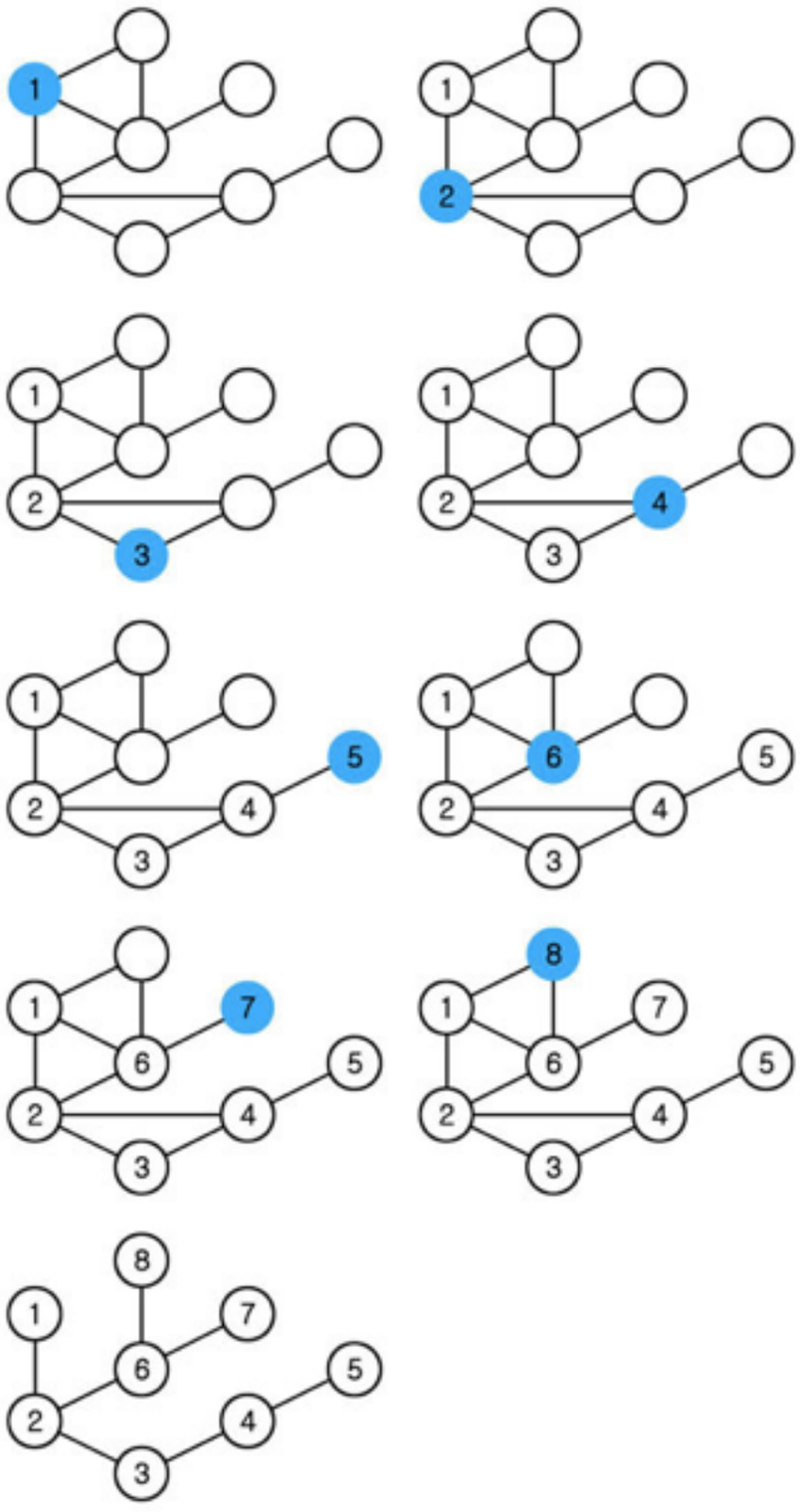






깊이우선탐색

'Stack' 사용 - 갈 때까지 간다

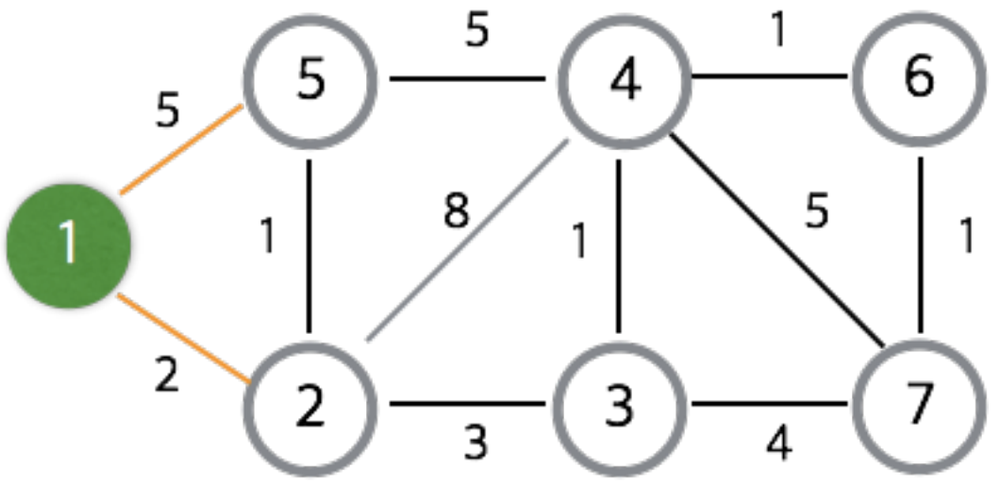


[DFS신장트리]

Dijkstra 알고리즘 그래프의 노드 사이에서 최단 경로를 찾는 알고리즘

현재 위치 방문 완료 연결된 간선

i	1	2	3	4	5	6	7
거리	0	2	∞	∞	5	∞	∞
체크	1						

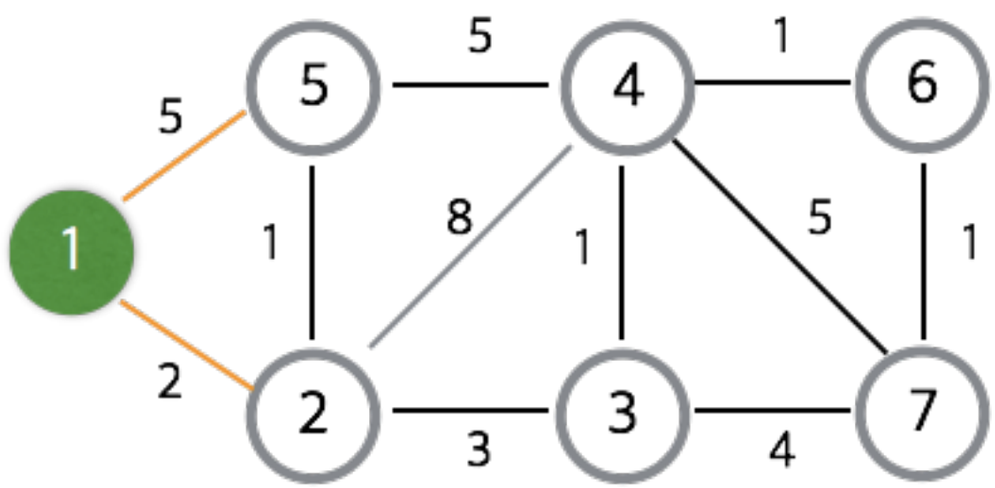


1. 체크되어 있지 않은 정점 중에서 거리의 값이 가장 작은 정점 x 선택
2. **x 체크**
3. **x**와 연결된 모든 정점 검사
4. 더 짧은 경로가 발견되었을 때, 갱신
5. 위의 과정을 모든 정점을 체크할 때까지 계속한다.

Dijkstra 알고리즘

현재 위치 방문 완료 연결된 간선

i	1	2	3	4	5	6	7
거리	0	2	∞	∞	5	∞	∞
체크	1						



1. 최초에 방문한 노드를 체크

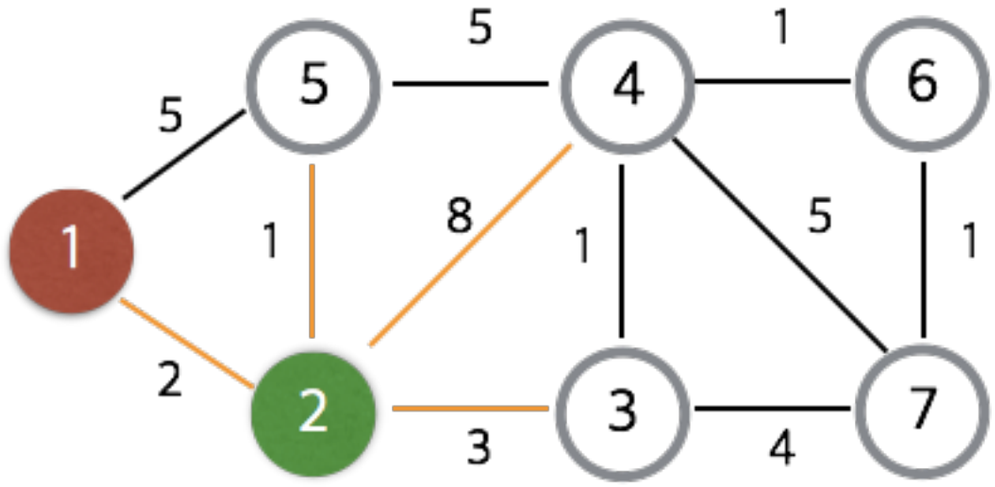
2. d[2], d[5]에 간선에 값을 입력

3. 체크가 되어 있지 않으면서

거리가 가장 작은 정점을 선택합니다. (2번 정점)

Dijkstra 알고리즘

i	1	2	3	4	5	6	7
거리	0	2	5	10	5->3	∞	∞
체크	1	1					



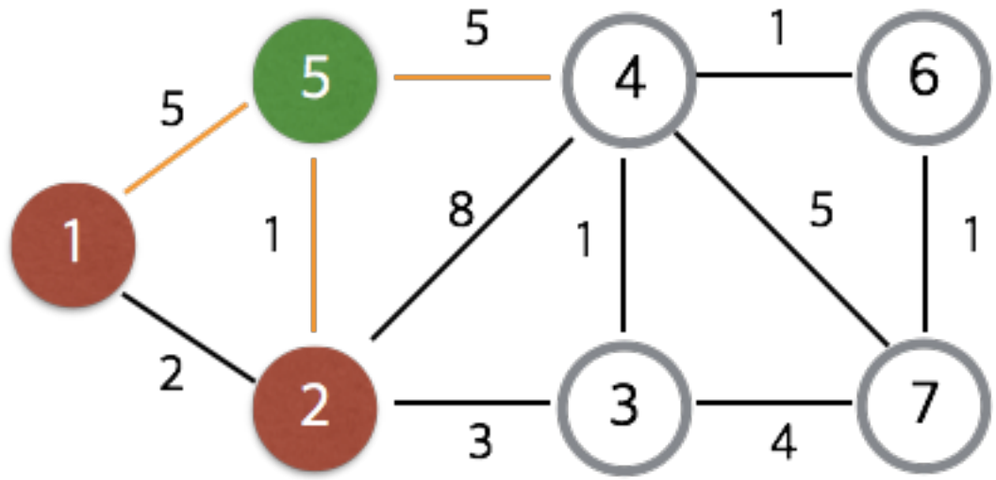
1. 방문하는 정점 체크
2. 2번 정점에서 접근 가능한 모든 노드들에 관하여 거리 계산 연산과 갱신

* 여기서는 5번 정점의 경우, 1번 -> 2번 -> 5번으로 가는 것이

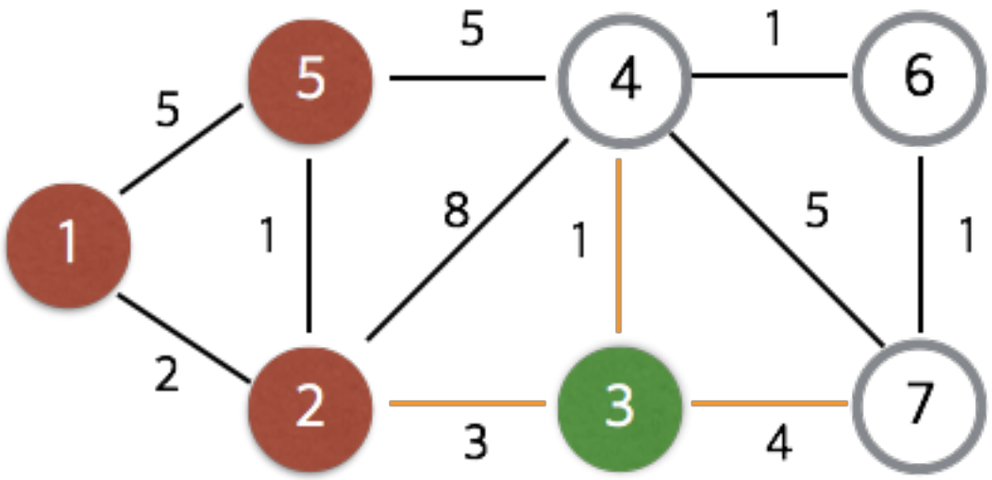
1 -> 5번으로 가는 것보다 빠르기 때문에 갱신

Dijkstra 알고리즘

i	1	2	3	4	5	6	7
거리	0	2	5	10->8	3	∞	∞
체크	1	1	1		1		

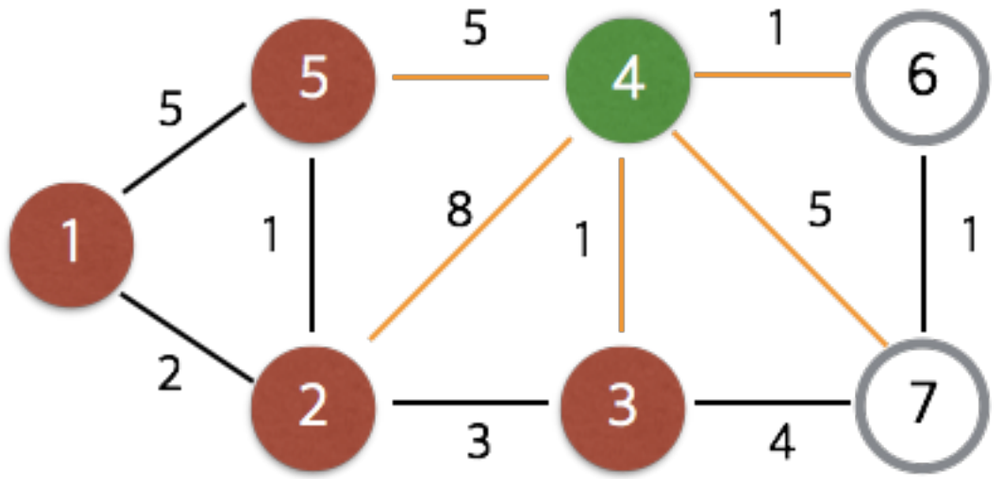


i	1	2	3	4	5	6	7
거리	0	2	5	8->6	3	∞	9
체크	1	1	1		1		

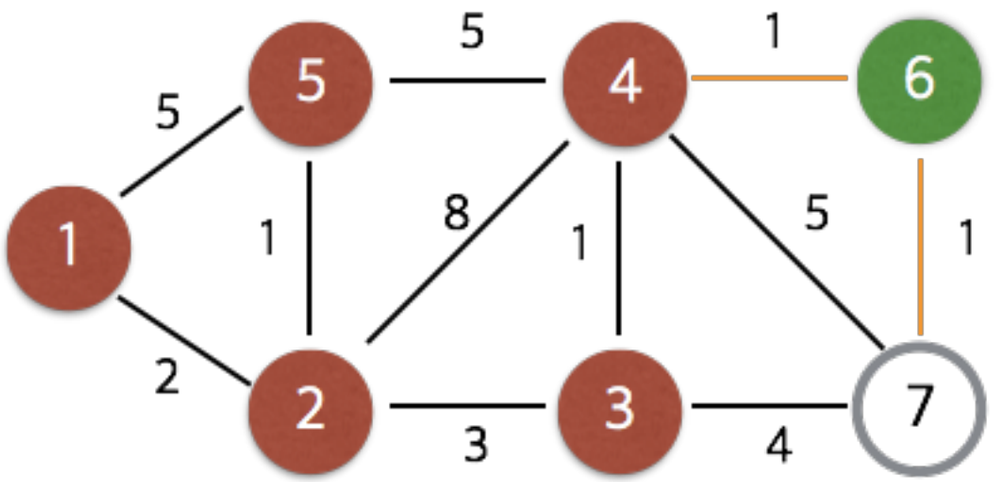


Dijkstra 알고리즘

i	1	2	3	4	5	6	7
거리	0	2	5	6	3	7	9
체크	1	1	1	1	1		

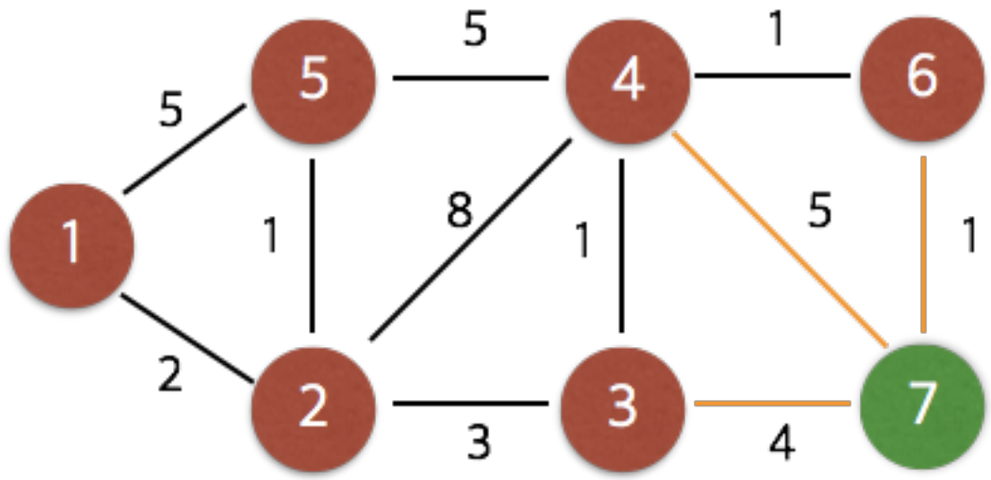


i	1	2	3	4	5	6	7
거리	0	2	5	6	3	7	9->8
체크	1	1	1	1	1	1	



Dijkstra 알고리즘

i	1	2	3	4	5	6	7
거리	0	2	5	6	3	7	8
체크	1	1	1	1	1	1	1



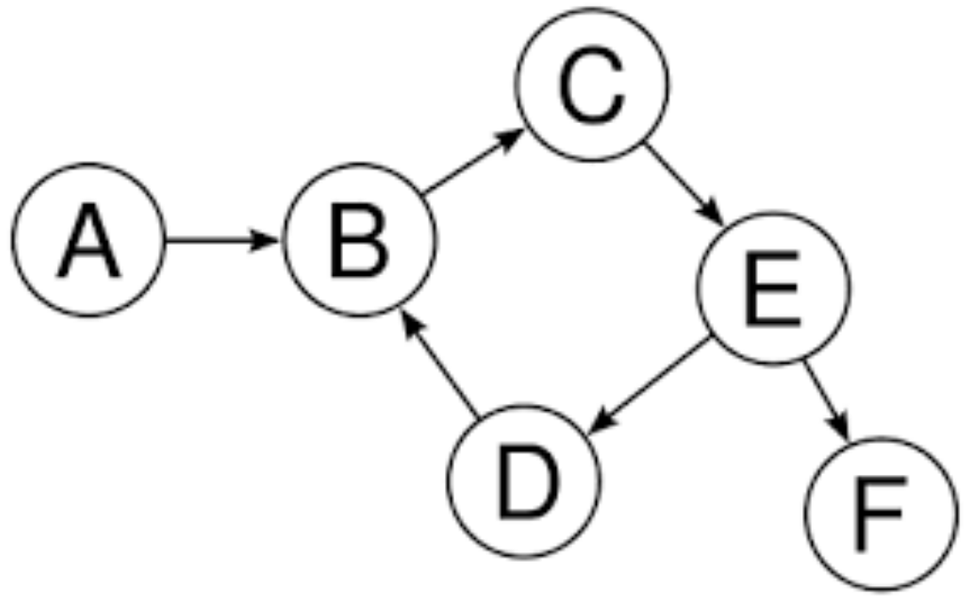
- **연결 리스트나 배열 구조**로 구현하고 Extract-Min(Q) 함수를 단순한 **선형 탐색**으로 구현했을 때,
  실행시간 = O(n^2)

- 실제 현장에서 사용되는 좋은 사례 :
  인터넷 라우팅에서 사용되는 OSPF(Open Shortest Path First) 방식의 프로토콜

http://manducku.tistory.com/29

방향그래프

방향성을 부여한 것 (간선을 화살표로 표시)



- 정점 A에서 B로 갈 수는 있어도, B에서 A로는 갈 수 없는 상황을 모델링하는데 유용
- 방향 그래프는 어떤 두 정점간에 연결될 수 있는가(Connectivity)를 알아보는 것이 주요 이슈