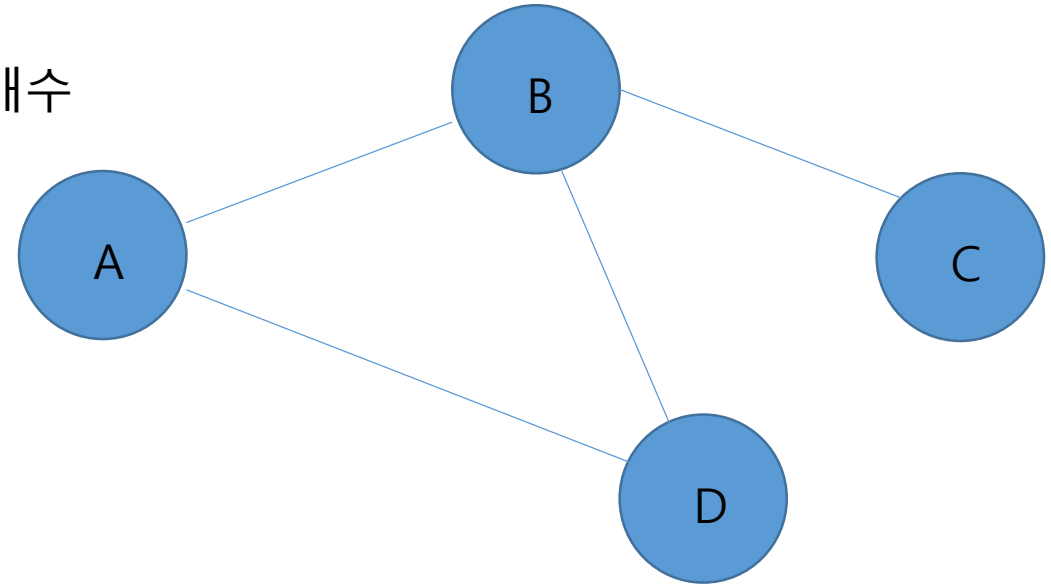
그래프

그래프란?

정점(노드)과 간선(edge)의 결합

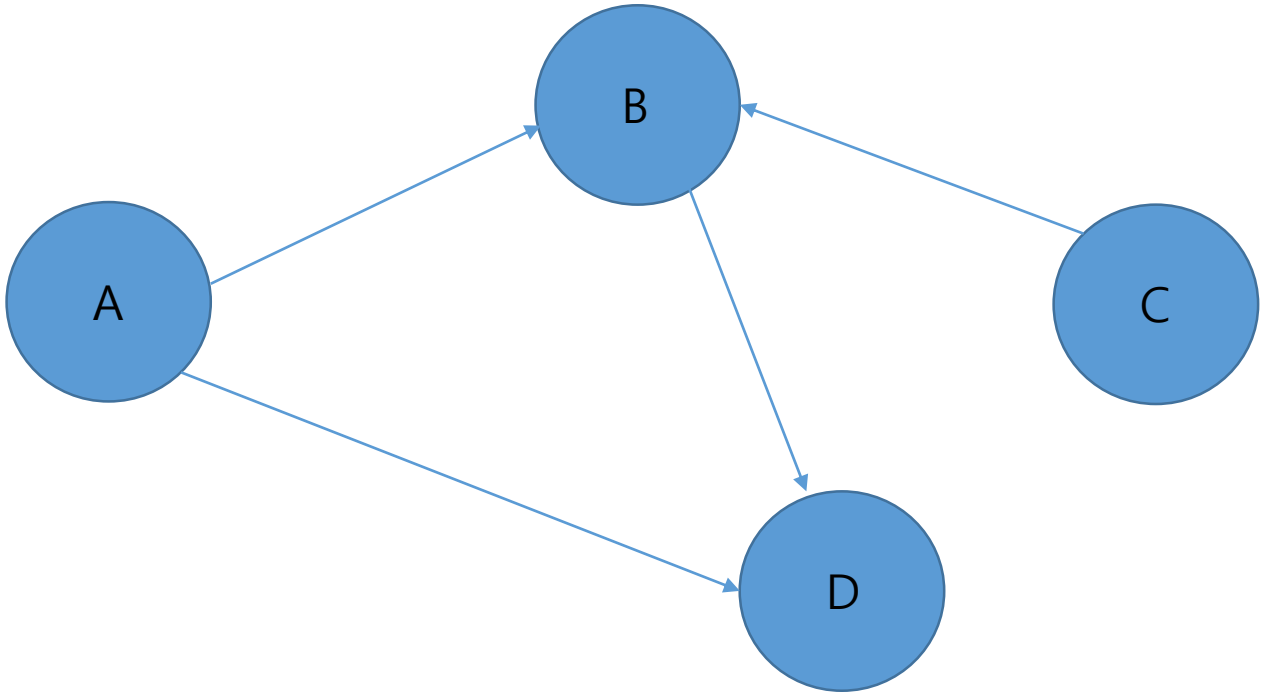
그래프 용어

- 정점: 노드라고도 하며 데이터가 저장된다.
- 간선: 정점간의 관계
- 사이클(cycle) : 출발 정점과 도착정점이 같은 경우
- 정점 차수(degree) : 각 정점에 연결되어 있는 간선의 개수
  - in-degree : 각 정점에 들어오는 간선의 개수
  - out-degree : 각 정점으로부터 나가는 간선의 개수



방향 그래프

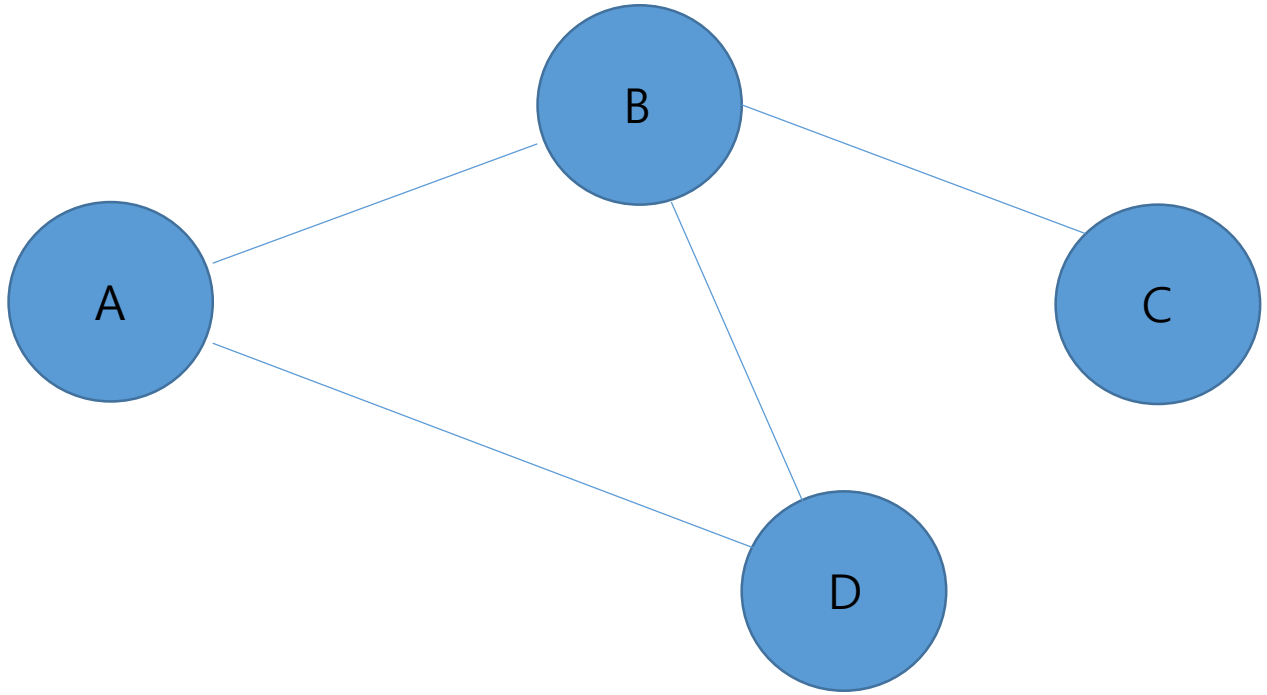
방향성이 있는 그래프
A에서 B로만 갈 수 있고 <A, B>로 표시



무방향 그래프

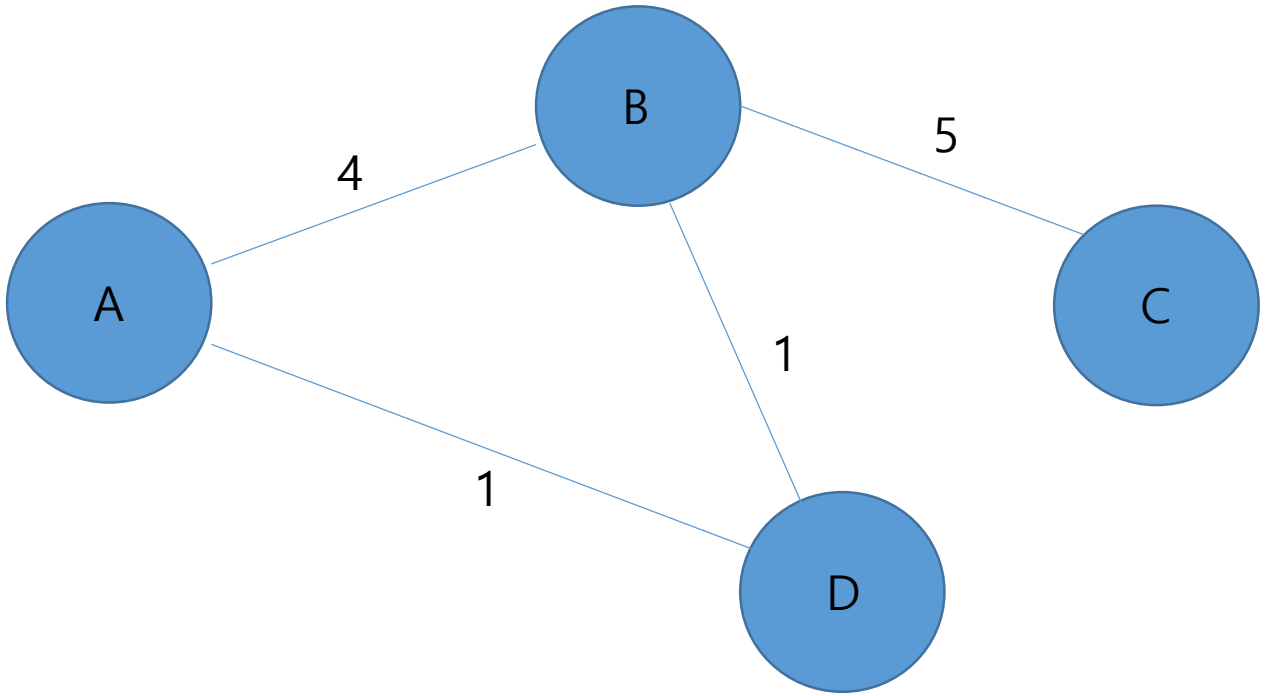
방향성이 없는 그래프

A -> B, B -> A 양방향으로 갈 수 있고 (A, B)로 표시



가중치 그래프

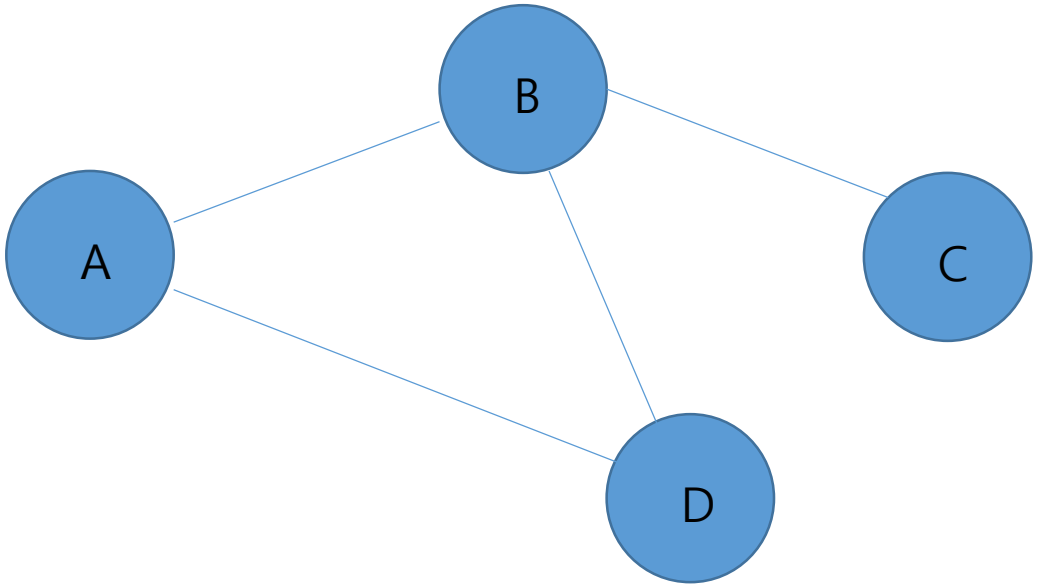
간선에 값이 존재하는 그래프



그래프 구현 방법

1. 인접 행렬

이차원 배열을 사용하여 그래프를 나타내는 방법



	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	0
D	1	1	0	0

인접 행렬

특정한 정점과 인접한 정점을 1
그렇지 않을 경우 0

```javascript
//i, j는 정점을 의미한다.
if(i, j가 인접한 정점일 경우){
  M[i][j] = 1;
} else{
  M[i][j] = 0;
}
```

JavaScript

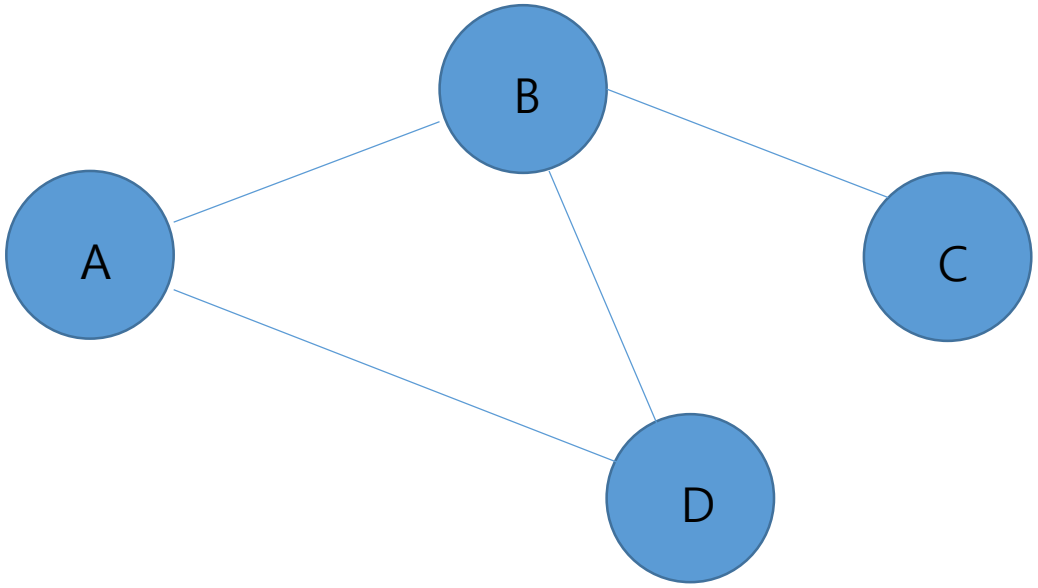
인접 행렬

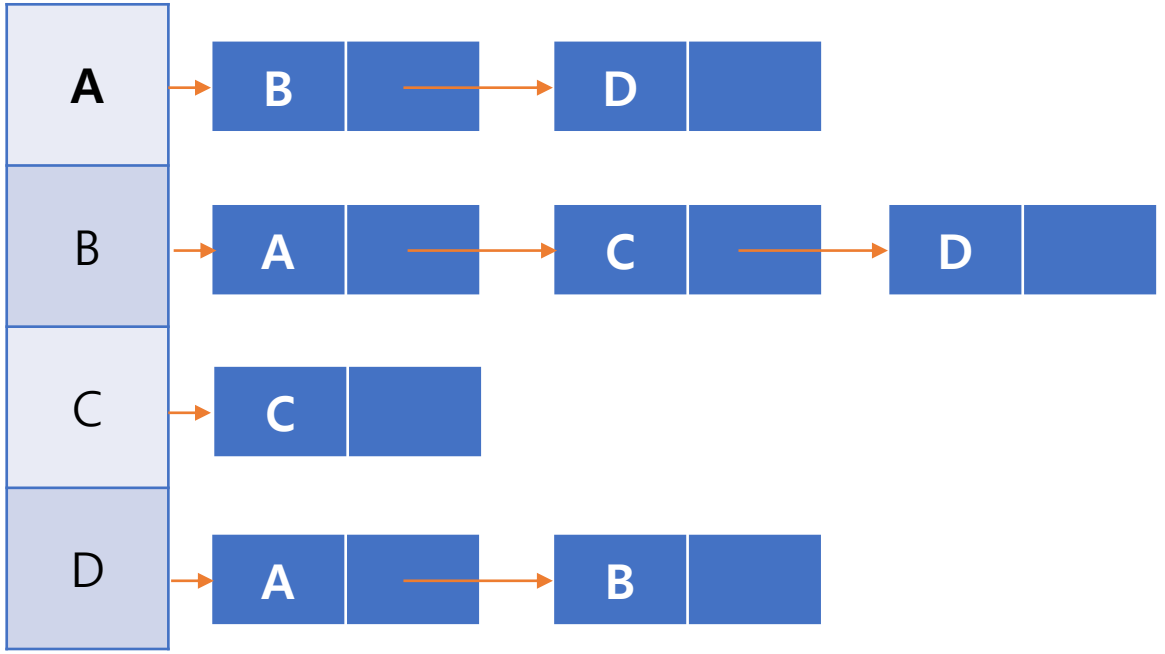
- 탐색 : O(1)
- 모든 간선의 수를 알아내기 위해선 **O(V^2)**
- 그래프에 간선이 많이 존재하는 **밀집 그래프**의 경우 유리하다.
    - 인접 리스트보다 많은 메모리 공간을 요구하지만 빠른 접근이 가능하다.

그래프 구현 방법

2. 인접 리스트

연결 리스트를 이용하여 그래프를 구현하는 방법





인접 리스트

- 탐색 : 정점 차수만큼의 시간이 필요하다.
- 모든 간선의 수 : O(V+E)
- 그래프에 간선이 적은 희소 그래프(Sparse Graph)의 경우 유리하다.
  - 적은 메모리 공간을 요구한다. => 인접한 정점들만 표시하기 때문

그래프의 탐색

1. 깊이 우선 탐색(DFS, Depth-First Search)

   - 그래프에 존재하는 임의의 정점에서 계속해서 연결된 정점으로 나아간다.

   - 더 이상 연결된 정점이 없다면 이전 정점으로 돌아간다.

   - **Stack** 또는 **재귀**를 이용하여 구현할 수 있다.

그래프의 탐색

2. 너비 우선 탐색(BFS, Breadth-First Search)

- 그래프에 존재하는 임의의 정점에서 인접한 정점들을 탐색한다.

- Queue를 이용하여 구현할 수 있다.

Q&A