그래프

- 그래프 (Graph) 란?

그래프

- 그래프 (Graph) 란?
  → 노드 (N, Node)와 노드를 연결하는 간선 (E, Edge)을 모아 놓은 자료구조.

그래프

- 그래프 (Graph) 란?
  → 노드 (V, Node)와 노드를 연결하는 간선 (E, Edge)을 모아 놓은 자료구조.

즉, 연결되어 있는 객체 간의 관계를 표현할 수 있는 자료구조.

→ ex. 지도, 지하철 노선도, 도로 등

	그래프	트리
정의	노드(node)와 그 노드를 연결하는 간선(edge)을 하나로 모아 놓은 자료 구조	그래프의 한 종류 DAG(Directed Acyclic Graph, 방향성이 있는 비순환 그래프)의 한 종류
방향성	방향 그래프(Directed), 무방향 그래프(Undirected) 모두 존재	방향 그래프(Directed Graph)
사이클	사이클(Cycle) 가능, 자체 간선(self-loop)도 가능, 순환 그래프(Cyclic), 비순환 그래프(Acyclic) 모두 존재	사이클(Cycle) 불가능, 자체 간선(self-loop)도 불가능, 비순환 그래프(Acyclic Graph)
루트 노드	루트 노드의 개념이 없음	한 개의 루트 노드만이 존재, 모든 자식 노드는 한 개의 부모 노드 만을 가짐
부모-자식	부모-자식의 개념이 없음	부모-자식 관계 top-bottom또는 bottom-top으로 이루어짐
모델	네트워크 모델	계층 모델
순회	DFS, BFS	DFS, BFS안의 Pre-, In-, Post-order
간선의 수	그래프에 따라 간선의 수가 다름, 간선이 없을 수도 있음	노드가 N인 트리는 항상 N-1의 간선을 가짐
경로	-	임의의 두 노드 간의 경로는 유일
예시 및 종류	지도, 지하철 노선도의 최단 경로, 전기 회로의 소자들, 도로(교차점과 일방 통행길), 선수 과목	이진 트리, 이진 탐색 트리, 균형 트리(AVL 트리, red-black 트리), 이진 힙(최대힙, 최소힙) 등

그래프 탐색

- 그래프 탐색 이란?

→ 그래프의 가장 기본적인 연산.
→ 하나의 정점으로부터 시작하여 차례대로 모든 정점을 1회 씩 방문

알고리즘 문제 중 가장 비중이 높고, 중요하다 !

그래프 탐색

- 그래프 탐색 이란?

→ 그래프의 가장 기본적인 연산.

→ 하나의 정점으로부터 시작하여 차례대로 모든 정점을 1회 씩 방문

알고리즘 문제 중 가장 비중이 높고, 중요하다 !

→그래프에 관련된 이론을 잘 알고있자!

1. 무방향 그래프 vs. 방향 그래프
2. 가중치 그래프
3. 연결 그래프 vs. 비연결 그래프
4. 사이클 vs. 비순환 그래프

그래프 탐색 (DFS & BFS)

가장 기본적인 탐색 DFS와 BFS에 대해 알아보자

Depth First Search (DFS)

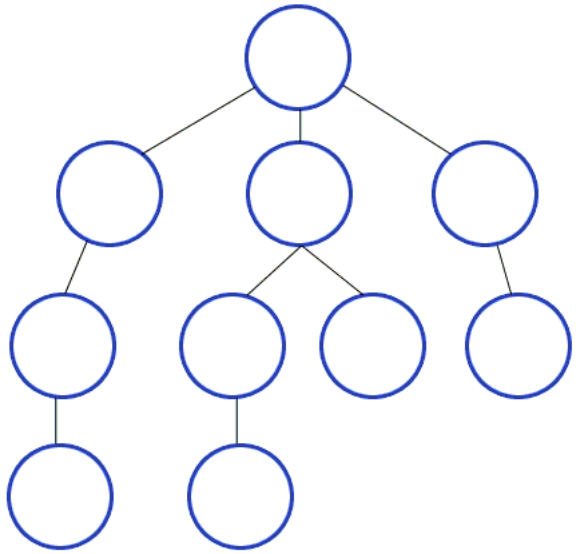
- 깊이 우선 탐색

  → 하나의 특정 노드를 시작으로, 깊이 (일련의 Branch)를 우선하여 탐색한다.

Depth First Search (DFS)

- 깊이 우선 탐색

  → 하나의 특정 노드를 시작으로, 깊이 (일련의 Branch)를 우선하여 탐색한다.



Depth First Search (DFS)

- 시간 복잡도
  - 어떻게 그래프를 구현했느냐에 따라 다르다 !

Depth First Search (DFS)

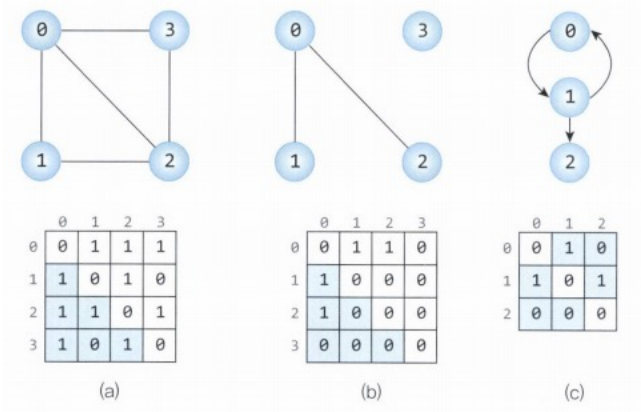
- 시간 복잡도
  - 어떻게 그래프를 구현했느냐에 따라 다르다 !
    - 인접행렬 (Adjacency Metrix) 방식: $O(V^2)$
    - 인접리스트 (Adjacency List) 방식: $O(V+E)$ (참고로, $E < V^2$)

Depth First Search (DFS)

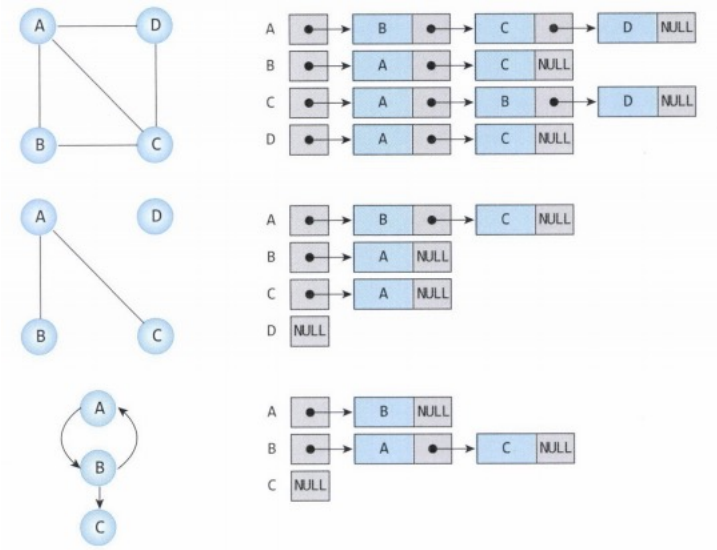
- 시간 복잡도
  - 어떻게 그래프를 구현했느냐에 따라 다르다 !
    - 인접행렬 (Adjacency Metrix) 방식: $O(V^2)$
    - 인접리스트 (Adjacency List) 방식: $O(V+E)$ (참고로, $E < V^2$)

인접행렬 vs. 인접리스트

- 인접 행렬 (Adjacency Metrix)



- 인접 리스트 (Adjacency List)



인접행렬 vs. 인접리스트

- 인접 행렬 (Adjacency Metrix) vs. 인접 리스트 (Adjacency List)

전체 정점 개수 : n
전체 간선 개수 : e

	인접 행렬	인접 리스트
간선(u, v) 검색	O(1)	O(degree(v))
정점(v)의 차수 계산	O(n)	O(degree(v))
전체 노드 탐색	$O(n^2)$	O(e)
메모리	n*n	n+e
구현	비교적 easy	비교적 difficult

인접행렬 vs. 인접리스트

- 인접 행렬 (Adjacency Metrix) vs. 인접 리스트 (Adjacency List)

전체 정점 개수 : n
전체 간선 개수 : e

	인접 행렬	인접 리스트
간선(u, v) 검색	O(1)	O(degree(v))
정점(v)의 차수 계산	O(n)	O(degree(v))
전체 노드 탐색	$O(n^2)$	O(e)
메모리	n*n	n+e
구현	비교적 easy	비교적 difficult

어렵게 생각하지 말고, Vector를 사용하자.

Depth First Search (DFS)

- DFS 자료구조
  - 깊이를 우선적으로 탐색한다. 어떤 자료구조를 사용할 수 있을까?

Depth First Search (DFS)

- DFS 자료구조
  - 깊이를 우선적으로 탐색한다. 어떤 자료구조를 사용할 수 있을까?

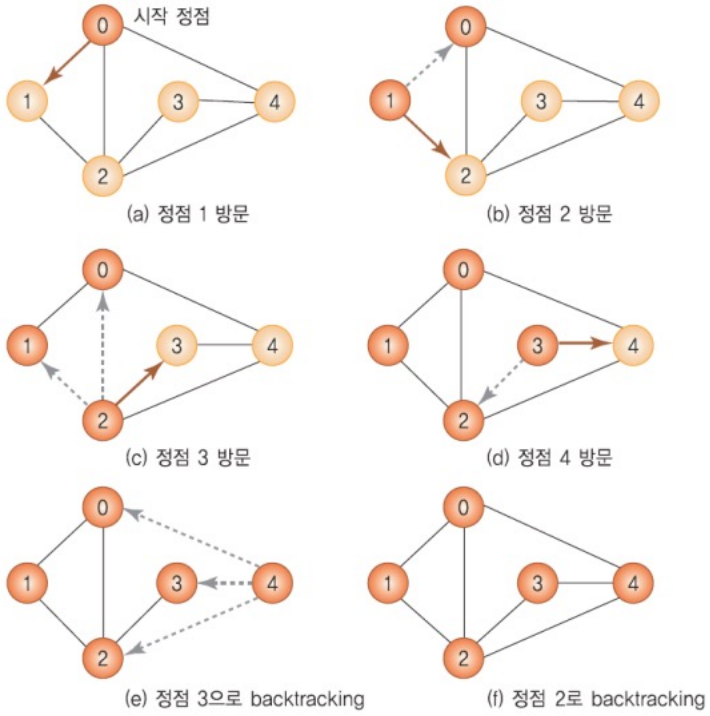
Stack!

하지만 보통, 재귀적으로 표현하는게 편하다.

Depth First Search (DFS)

- DFS – Pseudo code (recursive)

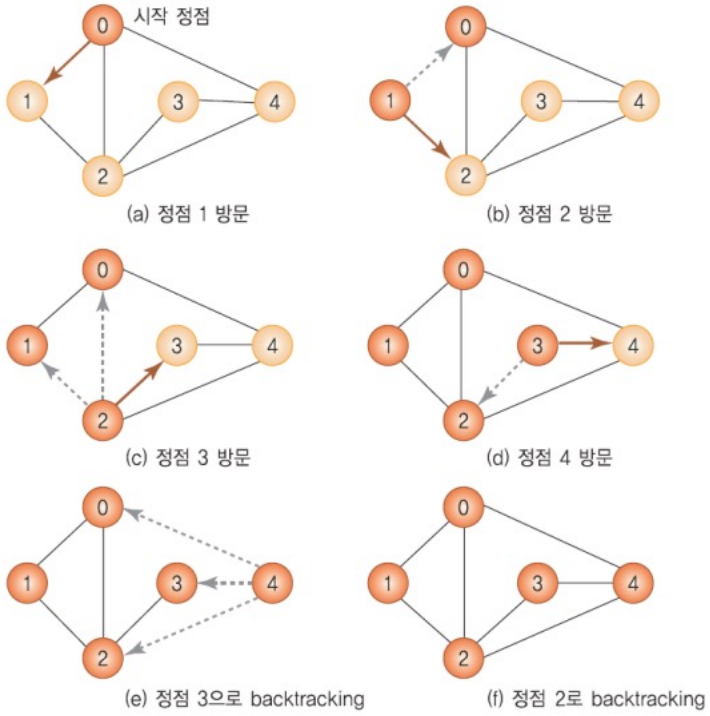
```
depth_first_search(v)
    v를 방문되었다고 표시;
    for all u ∈ (v에 인접한 정점) do
        if (u가 아직 방문되지 않았으면) then depth_first_search(u)
```



Depth First Search (DFS)

- DFS – Pseudo code (stack)

```
depth_first_search(v)
    stack::push(v);
    while stack not empty
        v <- stack::pop
        v를 방문되었다고 표시;
        for all u ∈ (v에 인접한 정점) do
        if (u가 아직 방문되지 않았으면) then stack::push(u)
```



Breadth First Search (BFS)

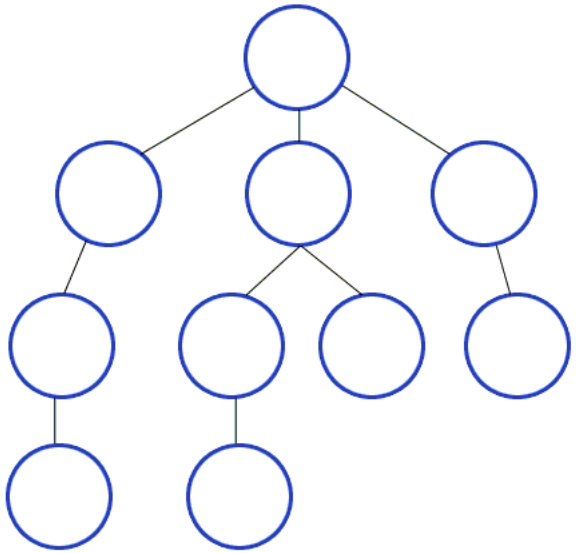
- 너비 우선 탐색

  → 하나의 특정 노드를 시작으로, 너비 (인접한 노드들)를 우선하여 탐색한다.

Breadth First Search (BFS)

- 너비 우선 탐색

  → 하나의 특정 노드를 시작으로, 너비 (인접한 노드들)를 우선하여 탐색한다.



Breadth First Search (BFS)

- 시간 복잡도
  - 어떻게 그래프를 구현했느냐에 따라 다르다 !
    - 인접행렬 (Adjacency Metrix) 방식: $O(V^2)$
    - 인접리스트 (Adjacency List) 방식: $O(V+E)$ (참고로, $E < V^2$)

Breadth First Search (BFS)

- BFS 자료구조
  - 너비를 우선적으로 탐색한다. 어떤 자료구조를 사용할 수 있을까?

Breadth First Search (BFS)

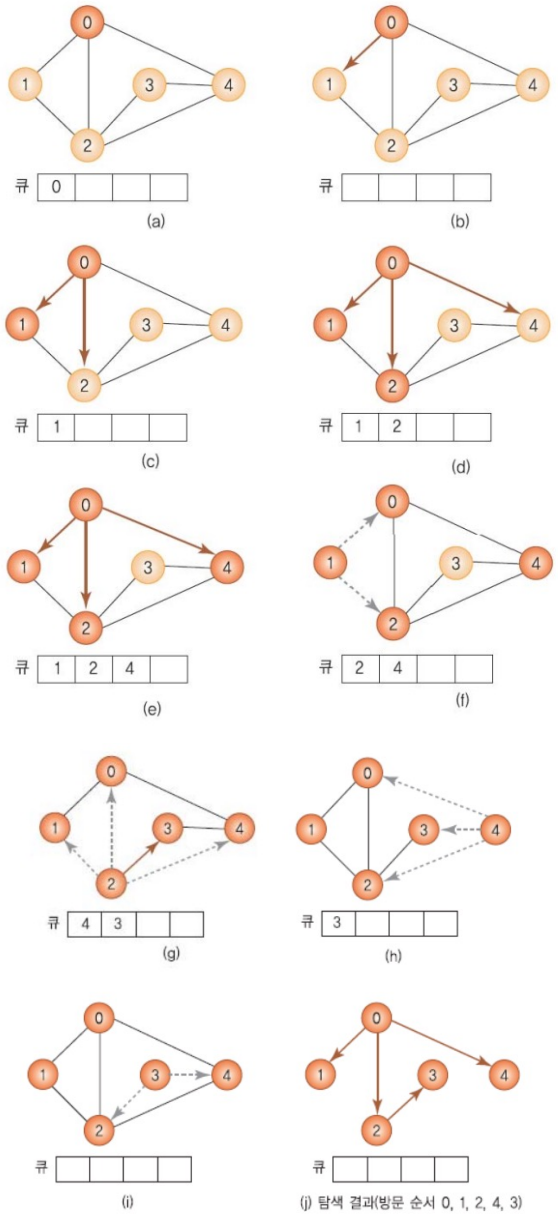
- BFS 자료구조
  - 너비를 우선적으로 탐색한다. 어떤 자료구조를 사용할 수 있을까?

Queue!

Breadth First Search (BFS)

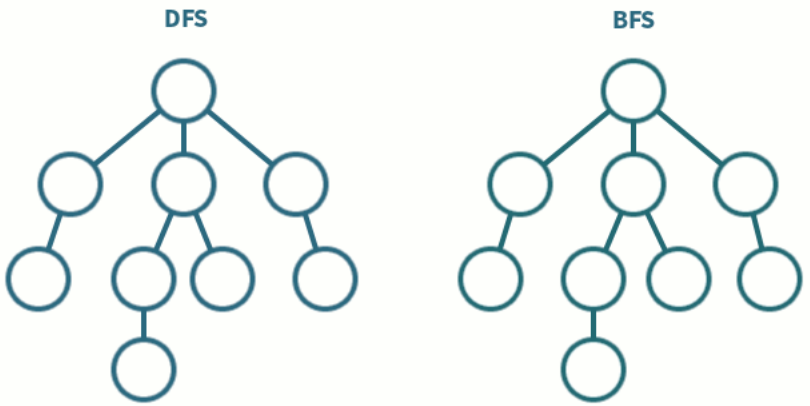
- BFS – Pseudo code (queue)

```
breadth_first_search(v)
    v를 방문되었다고 표시;
    큐 Q에 정점 v를 삽입;
    while (not is_empty(Q)) do
        큐 Q에서 정점 w를 삭제;
        for all u ∈ (w에 인접한 정점) do
            if (u가 아직 방문되지 않았으면) then u를 큐 Q에 삽입;
            u를 방문되었다고 표시;
```



DFS vs. BFS

- DFS와 BFS는 탐색되는 순서가 다르다.



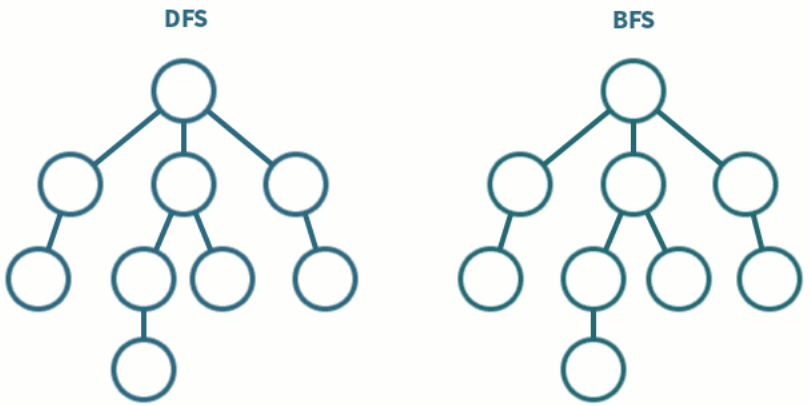
1. 그래프의 모든 정점을 방문해야 한다면?

2. 경로에 특징이 존재한다면?

3. 최단 거리를 구해야 한다면?

DFS vs. BFS

- DFS와 BFS는 탐색되는 순서가 다르다.



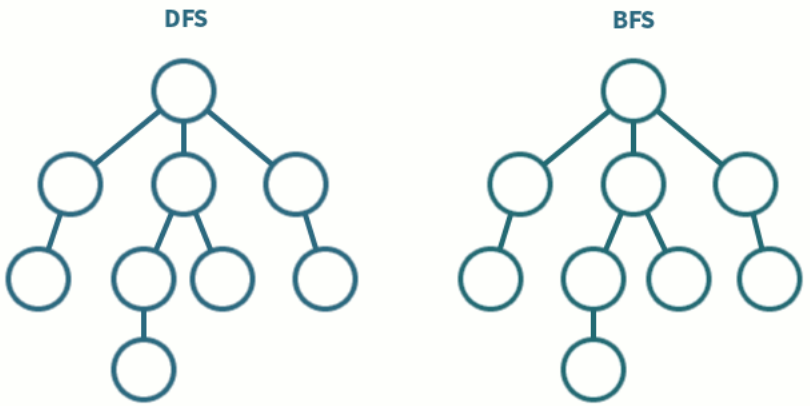
1. 그래프의 모든 정점을 방문해야 한다면?
   → 둘 중 편한거 이용.. 그 외 조건이 중요하다.

2. 경로에 특징이 존재한다면?

3. 최단 거리를 구해야 한다면?

DFS vs. BFS

- DFS와 BFS는 탐색되는 순서가 다르다.



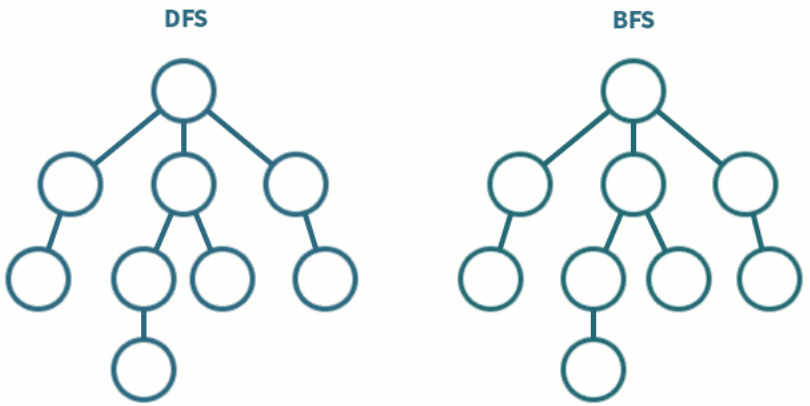
1. 그래프의 모든 정점을 방문해야 한다면?
   → 둘 중 편한거 이용.. 그 외 조건이 중요하다.

2. 경로에 특징이 존재한다면?
   → 각 각의 경로마다 특징을 저장해야한다면, DFS를 사용한다.

3. 최단 거리를 구해야 한다면?

DFS vs. BFS

- DFS와 BFS는 탐색되는 순서가 다르다.



1. 그래프의 모든 정점을 방문해야 한다면?
   → 둘 중 편한거 이용.. 그 외 조건이 중요하다.

2. 경로에 특징이 존재한다면?
   → 각 각의 경로마다 특징을 저장해야한다면, DFS를 사용한다.

3. 최단 거리를 구해야 한다면?
   → BFS, 단 edge에 가중치가 없어야한다.