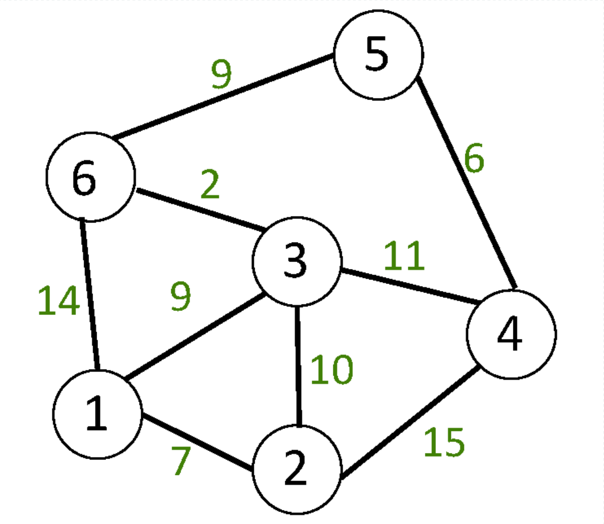
알고리즘 스터디

5주차: 너비우선 탐색(BFS)과 깊이우선 탐색(DFS)[1]

[1] Introduction to Algorihtm 3rd ed. 594p-612p

그래프: basic

- **개체와 개체간의 관계를 표현하는 자료구조**
  - 개체(정점, Vertex), 관계(간선, Edge)



그래프: basic

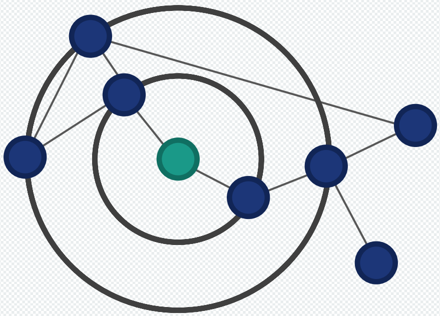
- 개체와 개체간의 관계를 표현하는 자료구조
  - 개체(정점, Vertex), 관계(간선, Edge)
- 정점은 관계의 주체가 되는 모든 데이터
  - 개체가 갖는 고유 값, 상태 값
- 간선은 방향이 있는(유향, Directed), 없는(무향, Undirected) 간선으로 구분
  - 무향 간선은 양방향 간선으로 생각해볼 수 있다.
- 간선은 가중치(weight)가 있는(가중, Weighted), 없는(비가중, Unweighted) 간선으로 구분
  - 무향 간선은 양방향 간선으로 생각해볼 수 있다.

그래프: BFS

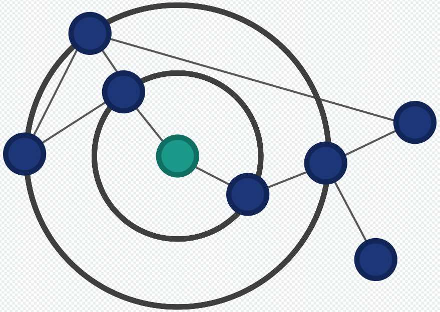
- Tree에서의 Traversal
  - 규칙에 따라 저장된 트리에서, 원하는 데이터의 조건에 따라 child를 선택
- Graph에서의 Traversal
  - 인접하지 않은 정점이라면, 경로(path)를 따라 원하는 데이터를 찾아야함
  - 전략없이 탐색한다면, 같은 경로를 계속 반복할지도 모른다
  - BFS(Breadth-First Search)
  - DFS(Depth-First Search)

그래프: 너비우선탐색(BFS, Breadth-First Search)

- 어떤 그래프 G = (V, E)를 생각하자
  - $s^{\forall} \in V$로부터, 도달가능한 "**모든**" vertex를 **"발견"**하기위해 탐색
- 왜 "너비우선" 탐색인가
  - "발견"한 정점과 "미발견"정점 사이의 경계를 level을 기준으로 넓게 확장
  - level K에 있는 정점을 모두 **"발견"** 후 level K+1의 정점을 **"발견"**



그래프: 너비우선탐색(BFS, Breadth-First Search)



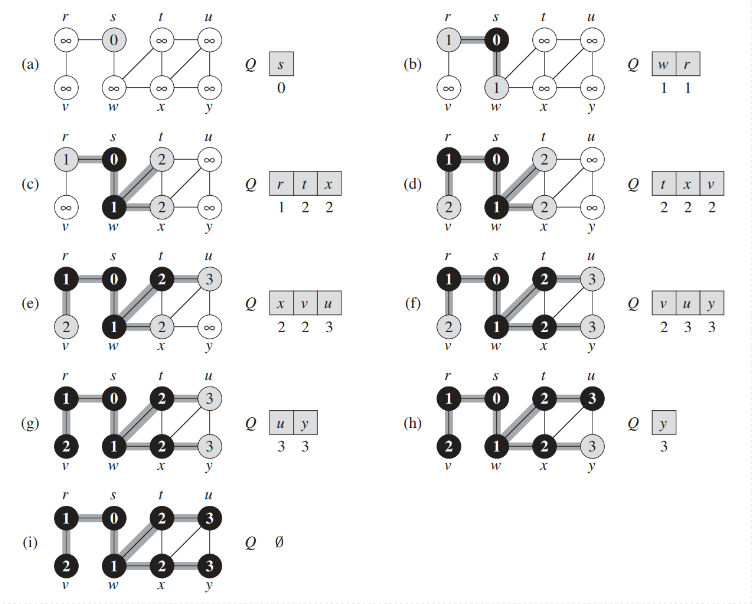
어떤 과정으로 이루어지는가

- 이전시간에 배운 큐(queue)를 활용해보자
  - level 1의 모든 노드를 큐에 넣는다
  - level 2의 모든 노드를 큐에 넣는다
  - …
- v_i는 $source\ node\ s$에 대해 거리가 i인 어떤 정점
- $u^{\forall} \in v_i.adjacent_nodes - \sum_{k=0}^{i} v_k$는 항상 v_{i+1}임을 유추할 수 있다
- 따라서, 이미 발견된 정점을 제외하고 인접한 정점을 계속해서 큐에 추가!

그래프: 너비우선탐색(BFS, Breadth-First Search)

어떤 과정으로 이루어지는가

- 검정색: 방문 / 회색: 발견 / 흰색: 미발견



그래프: 너비우선탐색(BFS, Breadth-First Search)

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

그래프: 너비우선탐색(BFS, Breadth-First Search)

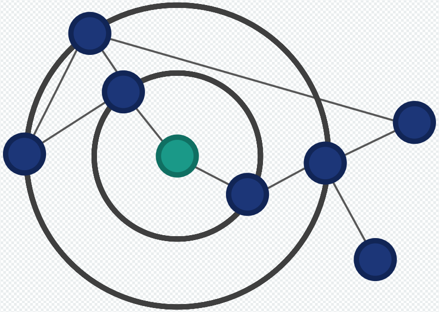
그래서 이걸 어디에 쓰는거죠

- 어떤 조건을 만족하는 정점 중 source로 부터 가장 가까운 정점을 찾는 문제
  - level(distance)순으로 방문함을 알고있다
  - 따라서, 방문한 모든 노드에 대해 조건 검사를 하면, 해결가능

- Unweighted Graph에 대해서 최단거리 문제를 풀 수 있다
  - Directed, Undirected 관계없이 가능하다
  - 단, 방향 그래프의 adjacent list 구성에 신경써야한다
  - single-source shortest-paths algorithm

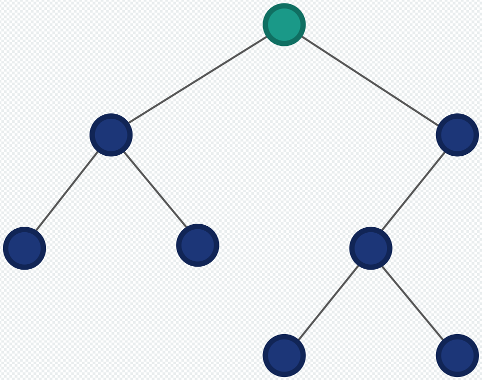
그래프: 너비우선탐색(BFS, Breadth-First Search)

- 아래 그래프에서 정점을 발견하는데 쓰인 간선만 남겨보자
  - 그리고 Source를 잡아서 들어올려보자 (상상력 필요) 🧐



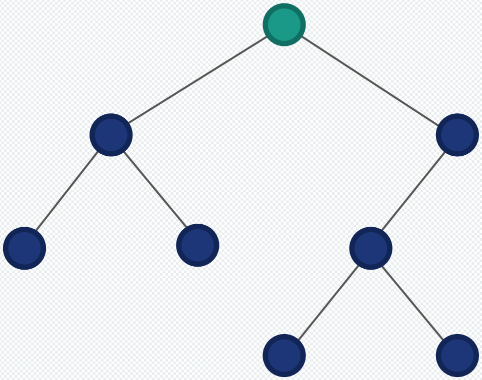
그래프: 너비우선탐색(BFS, Breadth-First Search)

- **아래 그래프에서 정점을 발견하는데 쓰인 간선만 남겨보자**
  - 그리고 Source를 잡아서 들어올려보자 (상상력 필요) 🥳
  - 트리가... 나왔네..?



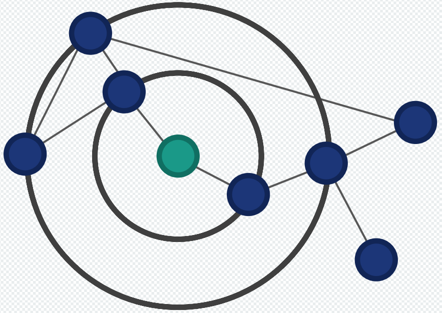
그래프: 너비우선탐색(BFS, Breadth-First Search)

- Spanning Tree, Breadth-First Tree
  - Source부터 도달가능한 모든 노드를 표현하는 트리
  - 각 level은 Source를 기준으로 한 거리
  - Source → Leaf 까지의 Path는 최단거리(비-가중 그래프에서)



그래프: 깊이우선탐색(DFS, Depth-First Search)

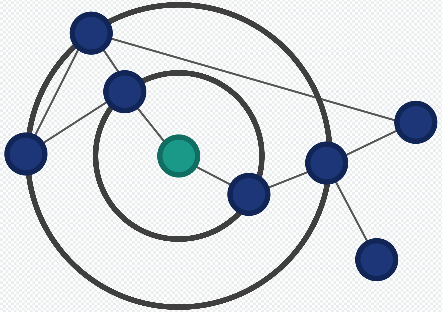
- 어떤 그래프 G = (V, E)를 생각하자
  - $^\forall s \in V$로부터, 도달가능한 "**모든**" vertex를 **"발견"**하기위해 탐색
- 왜 "깊이우선" 탐색인가
  - 가장 **"최근"**에 **"발견"**한 정점에서 나가는 간선 중, 가장 level이 높은 정점으로 향하는 간선을 우선 탐색한다
  - 마치 Tree traversal과 유사한 전략



그래프: 깊이우선탐색(DFS, Depth-First Search)

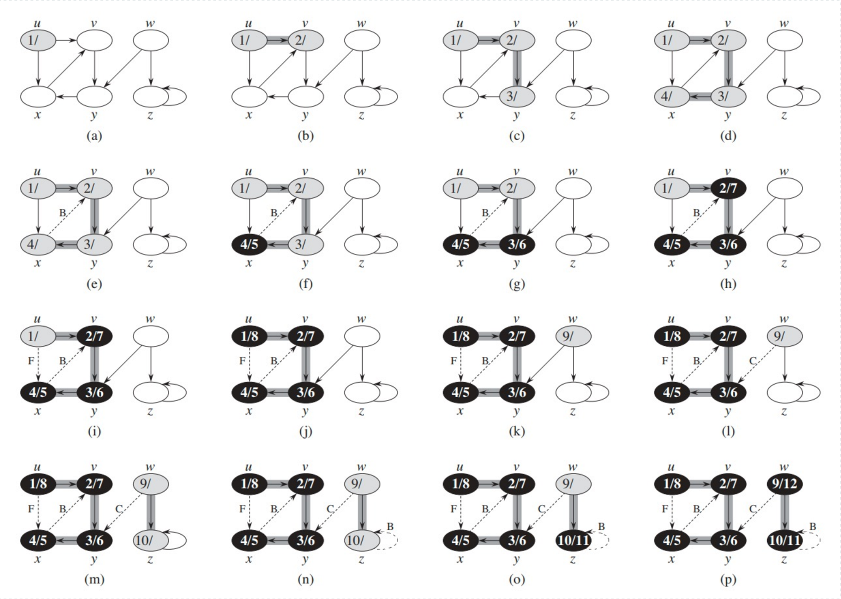
어떤 과정으로 이루어지는가

- 이전시간에 배운 스택(Stack)를 활용해보자
  - Level 0(Source)에서 도달가능한 정점 중 가장 깊은 정점을 "발견"
  - 앞서 선택한 정점을 "방문"
  - "방문"한 정점에서 방문한 적 없는 인접한 정점을 "발견"
  - …
- 더이상 깊은 정점을 찾을 수 없다면, 자신을 발견한 "부모노드"의 인접 정점 중 "발견"하지 않은 정점을 발견하고 위의 프로세스를 반복



그래프: 깊이우선탐색(DFS, Depth-First Search)

- 어떤 과정으로 이루어지는가
  - 검정색: 방문 / 회색: 발견 / 흰색: 미발견



그래프: 깊이우선탐색(DFS, Depth-First Search)

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1                 // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]           // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK                 // blacken u; it is finished
time = time + 1
u.f = time
```

그래프: 깊이우선탐색(DFS, Depth-First Search)

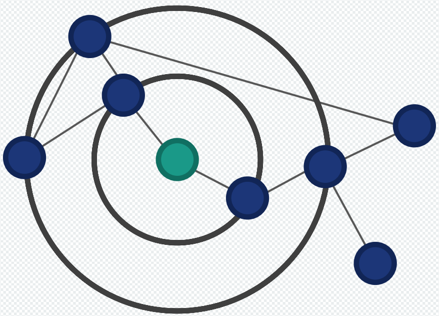
그래서 이걸 어디에 쓰는거죠

- BFS와 마찬가지로, 어떤 두 노드 u, v가 연결되어 있는지 즉, 경로(path)가 존재하는지 확인
  - 결국엔 Spanning Tree를 형성할 것이므로

- 각 정점에 **"상태정보"**를 저장한다고 가정해보자
  - 현재 방문한 정점에서 더이상 깊은 정점을 찾을 수 없는 경우, 하나의 시나리오(초기상태 → 여러 상태를 거쳐 → 최종상태) 를 마쳤다고 볼 수 있다
  - $u \rightarrow v \rightarrow w_0$ 로 추상화 해 보자
  - 이후, 다른 시나리오($u \rightarrow v \rightarrow w_1$) 를 탐색하는 경우, $u \rightarrow v$ 까지의 과정은 생략
  - 왜냐하면 u 까지의 **"상태정보"**를 정점 u에 저장했으니까!
  - 이러한 방법을 **"백트래킹(Back-tracking)"**이라고 부른다

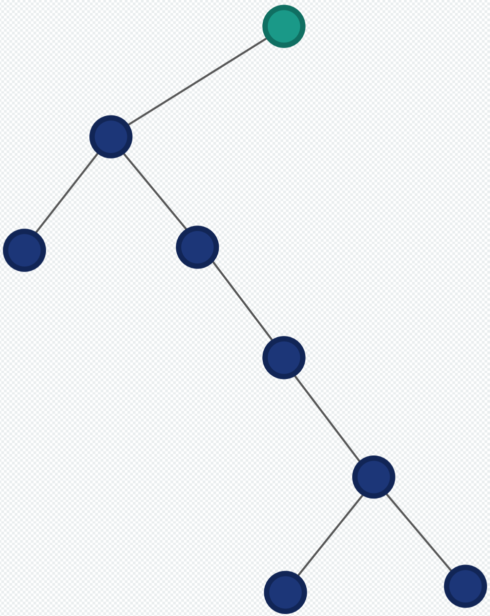
그래프: 깊이우선탐색(DFS, Depth-First Search)

- **아래 그래프에서 정점을 발견하는데 쓰인 간선만 남겨보자**
  - 그리고 Source를 잡아서 들어올려보자 (상상력 필요) 🧐



그래프: 깊이우선탐색(DFS, Depth-First Search)

- 아래 그래프에서 정점을 발견하는데 쓰인 간선만 남겨보자
  - 그리고 Source를 잡아서 들어올려보자 (상상력 필요)
  - 트리가... 나왔네..?



그래프: 깊이우선탐색(DFS, Depth-First Search)

- Spanning Tree, Depth-First Tree
  - DFS를 통해 탐색한 시나리오를 시각화 하는 트리
  - Source → Leaf 까지의 Path는 특정 정점을 지나는 모든 시나리오에 대한 정보를 담는다