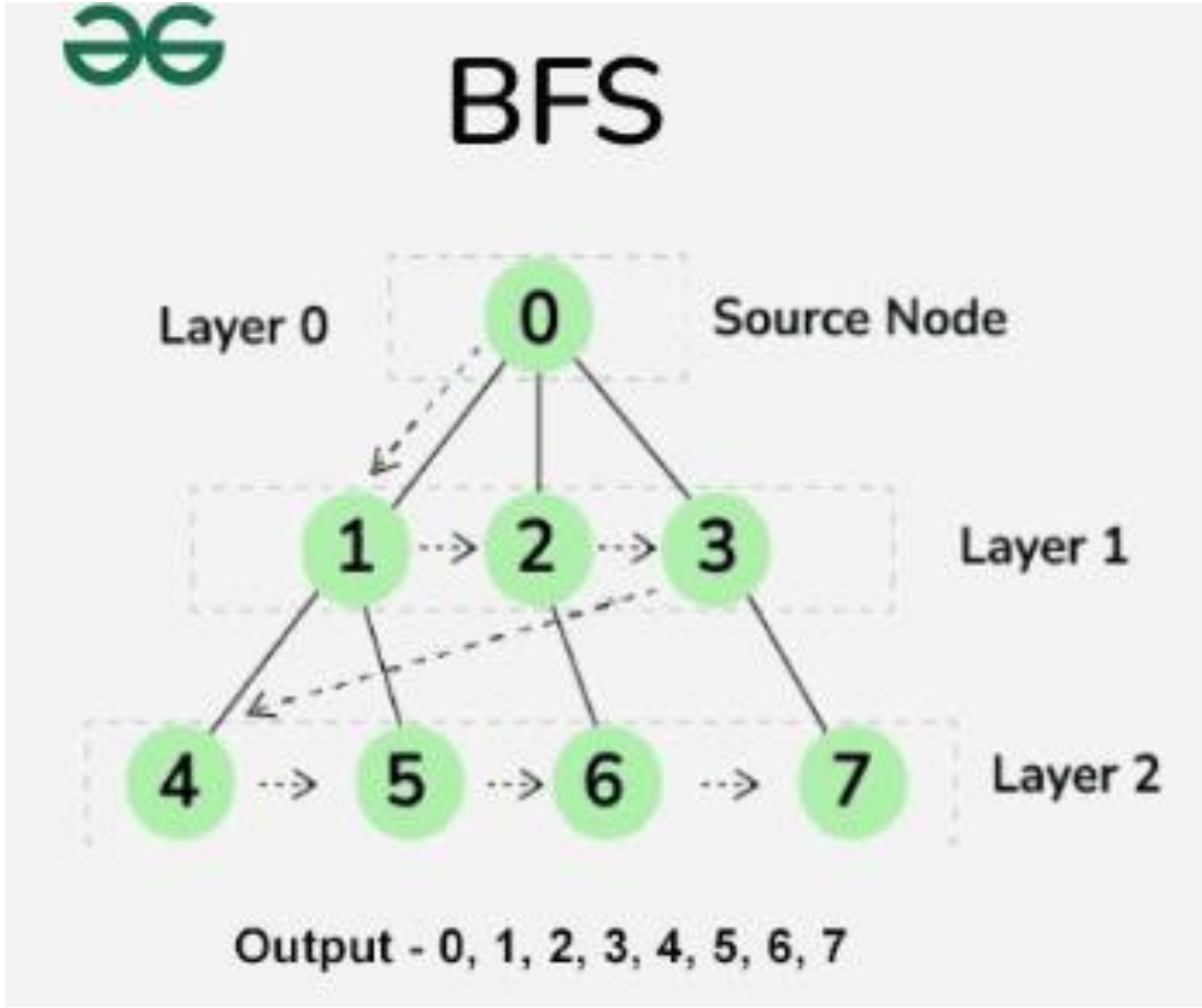
코테스터디

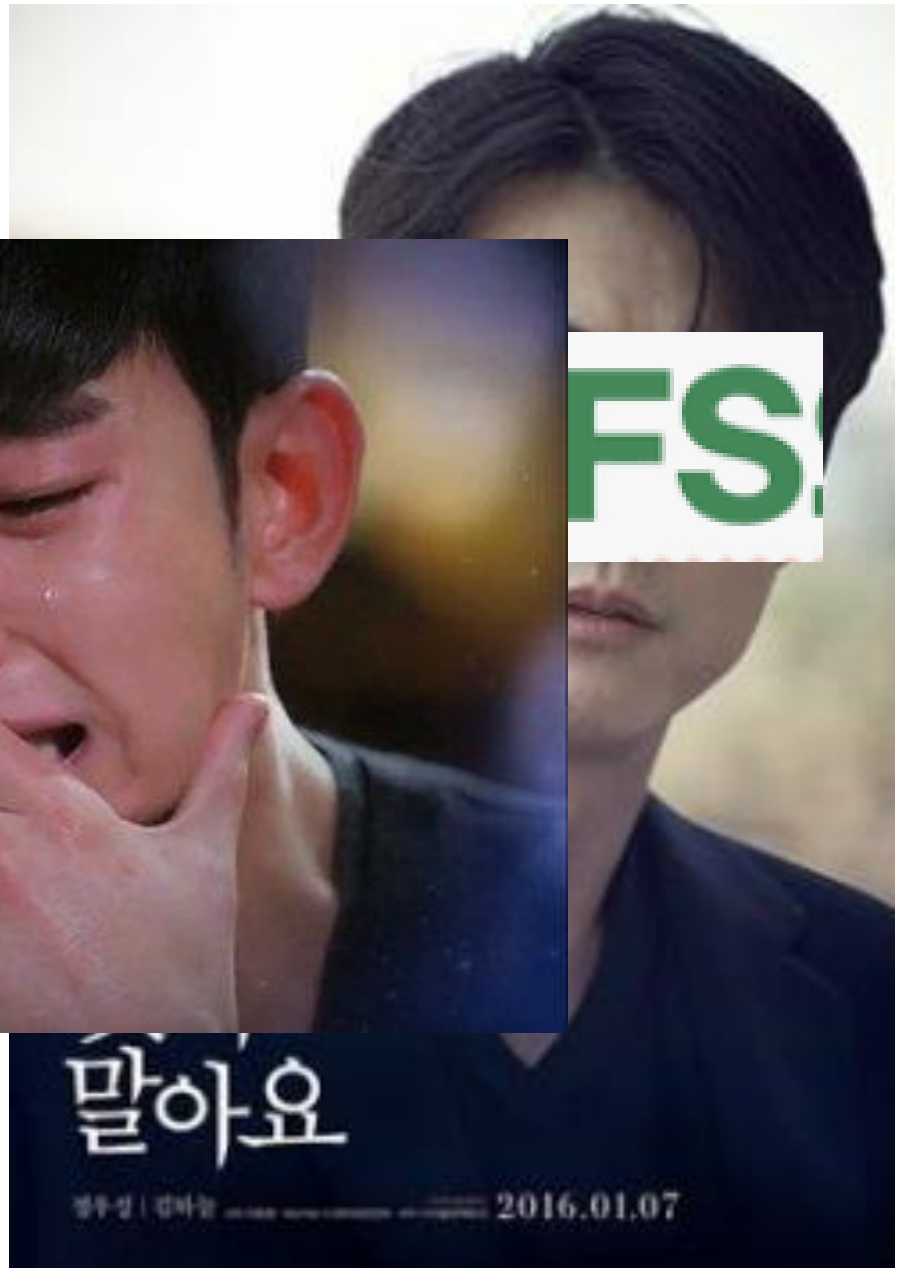
BFS(Breadth-first search)
너비 우선 탐색

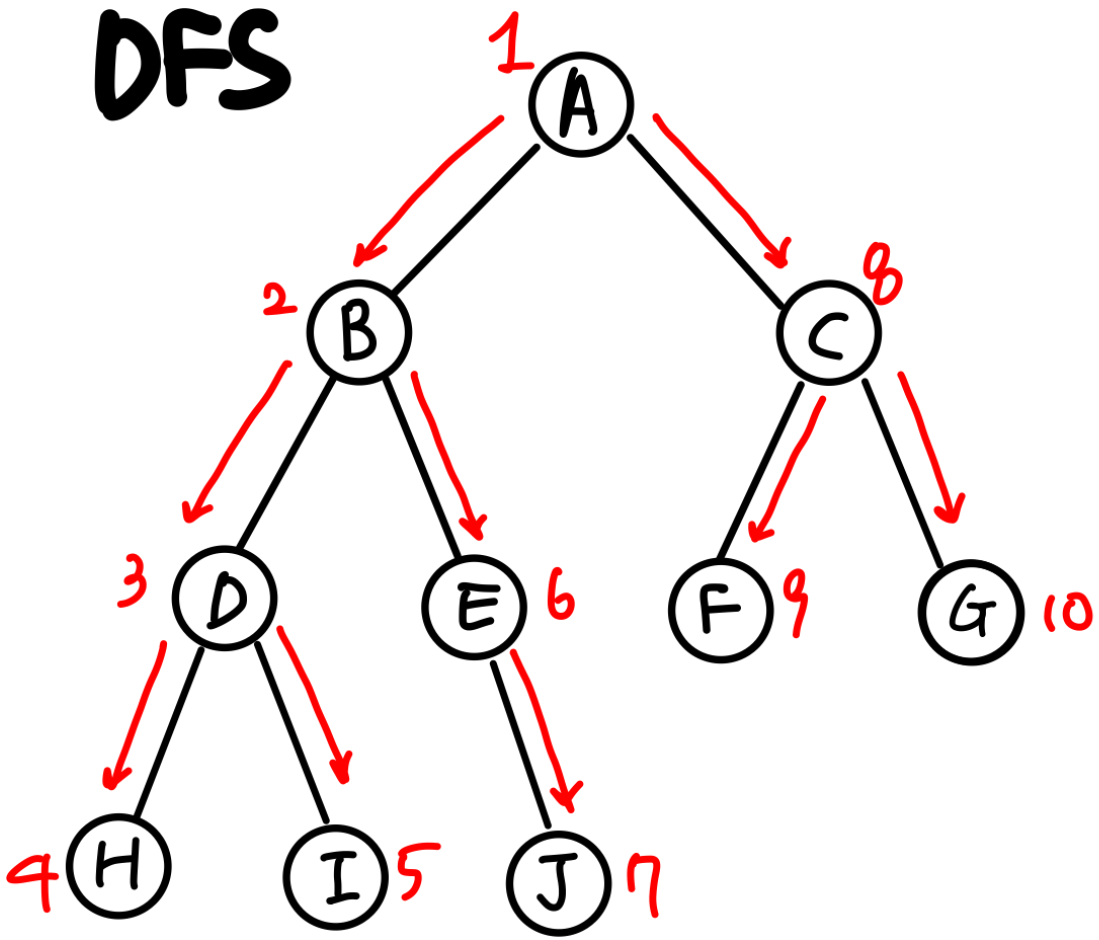

BFS
Layer 0
0
Source Node
1
2
3
Layer 1
4
5
6
7
Layer 2
Output - 0, 1, 2, 3, 4, 5, 6, 7

그래프를 탐색 하는 방법 중 하나!

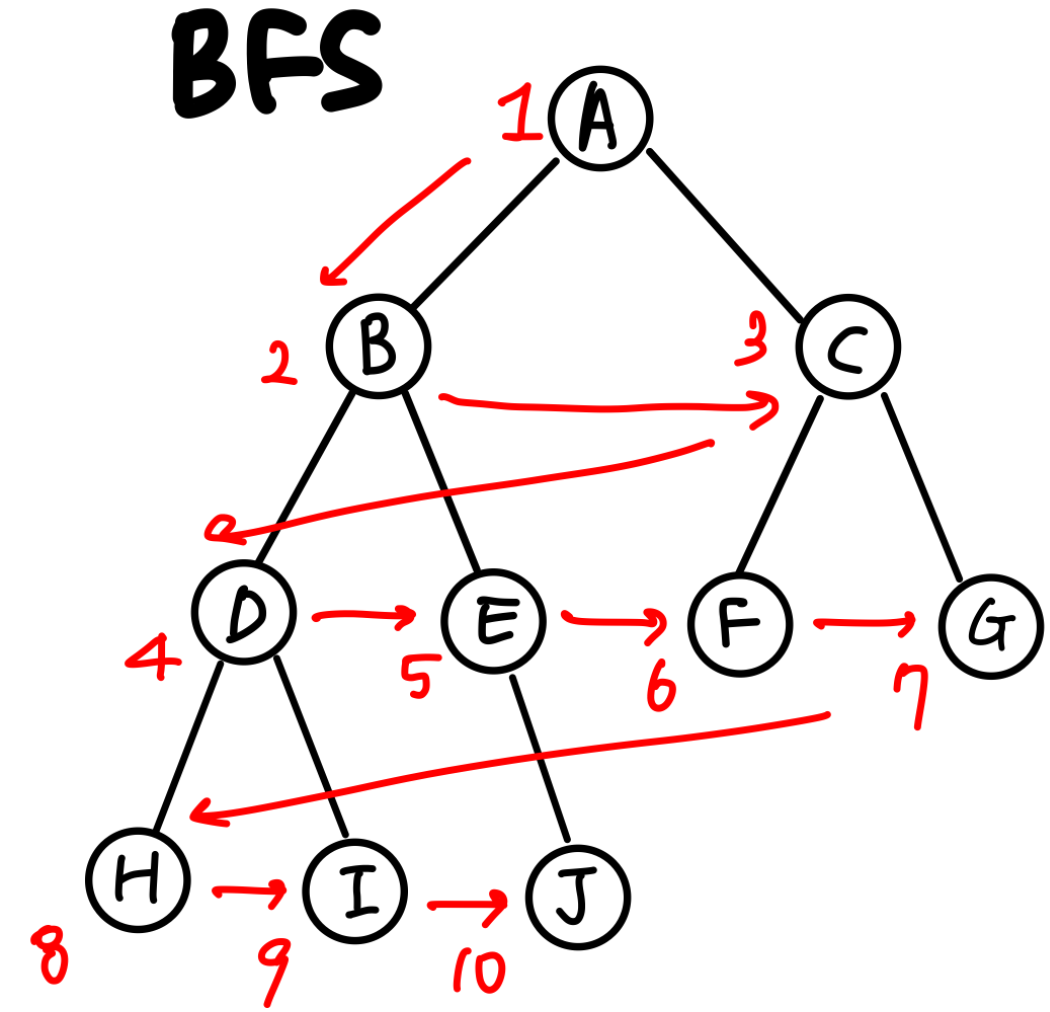
DFS는 기억나시죠?


에필로그


FS!
말아요
2016.01.07



스택, 재귀



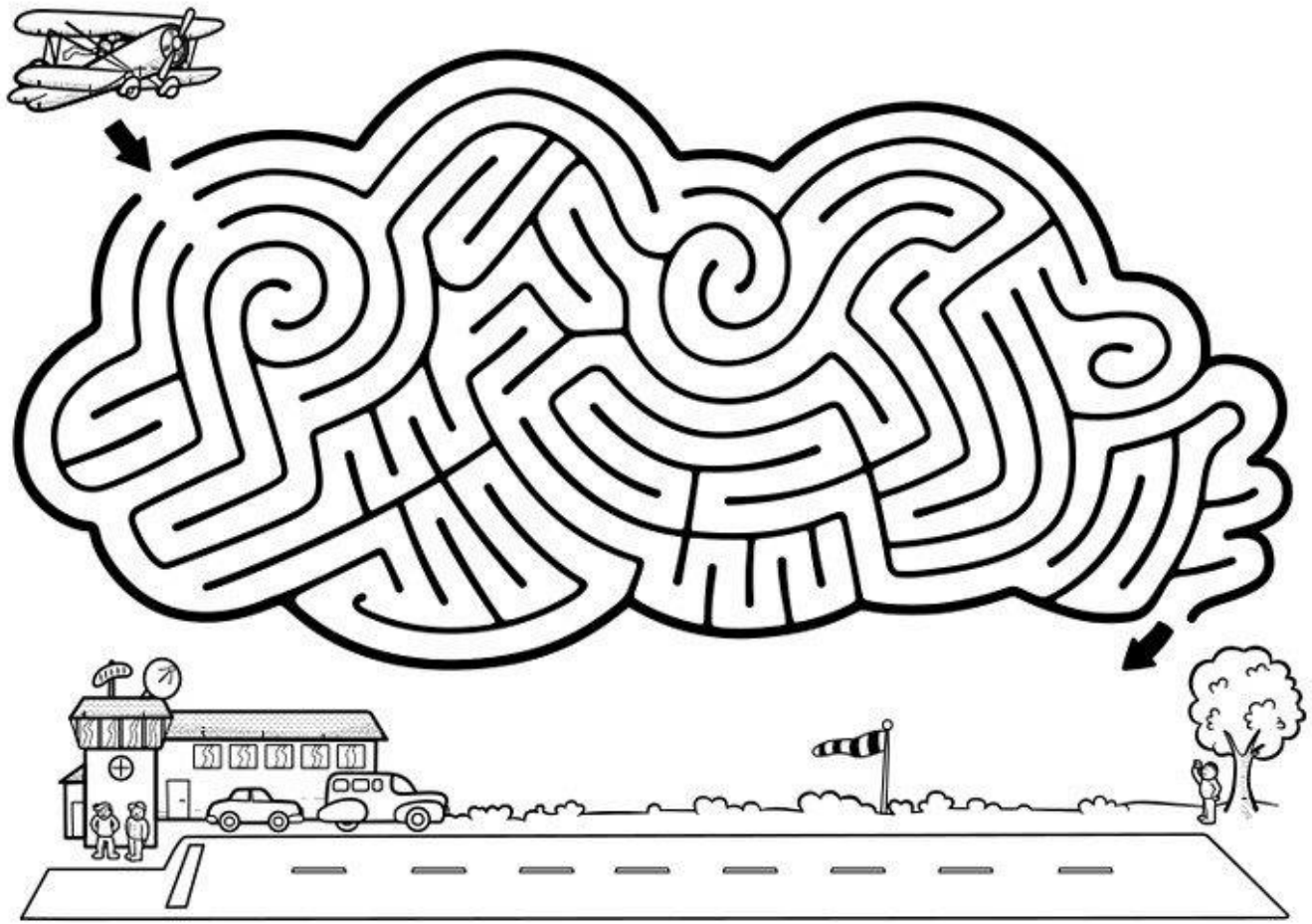
큐

BFS(Breadth-First-Search) 란?

- **하나의 정점으로부터 시작하여 차례대로 모든 정점들을 한 번씩 방**문하는 것
- **루트 노드**(혹은 다른 임의의 노드)에서 시작해서 **인접한 노드를 먼저 탐색**하는 방법
- **시작 정점으로부터 가까운 정점을 먼저 방문**하고 **멀리 떨어져 있는 정점을 나중에 방문**하는 순회 방법이다.
- 즉, **깊게(deep) 탐색하기 전에 넓게(wide) 탐색**하는 방법이다.
- BFS는 시작 노드에서 시작하여 거리에 따라 단계별로 탐색하는 방식이다.
- 방문한 노드들을 차례로 저장한 후 꺼낼 수 있는 **자료구조인 큐(Queue)를 사용**한다.
- 재귀적으로 동작하지 않는다.
- 무한 루프에 빠질 위험이 있기 때문에, **어떤 노드를 방문했었는지 여부를 반드시 검사**해야 한다.

BFS는 언제 주로 사용될까요?

- 두 노드 사이의 최단 경로 혹은 임의의 경로를 찾고 싶을 때 이 방법을 선택합니다!




... 어쩌라고...?

출발

1					

도착

출발

1					
2					

도착

출발

1					
2	3				
3					

도착

출발

1					
2	3	4			
3					
4					

도착

출발

1		5			
2	3	4			
3		5	6	7	
4					
5	6	7			
6					

도착

출발

1		5			
2	3				
3				7	
4				8	
5	6	7	8	9	10
6					11

미로 탐색

1 실버 I

도착


나 혼자 산다
연구소
KB DREAM WAVE 2030
난 DFS / BFS 부터
기억이 안 나


YouTube
그런데, 짜잔!

BFS의 구현

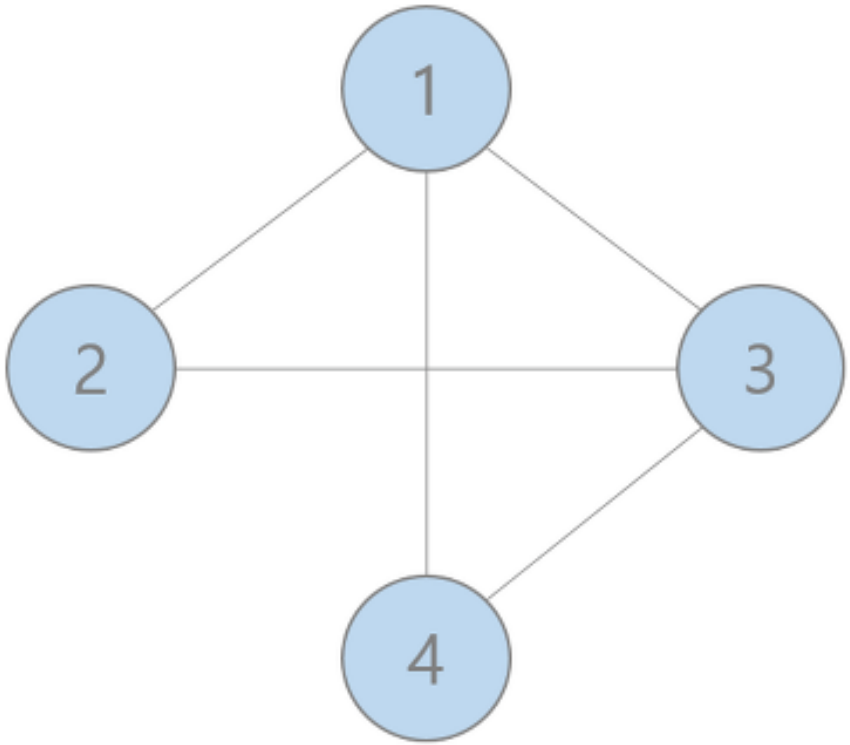
DFS의 구현 – 관계 구현(인접행렬)

인접 행렬은 그래프의 연결 관계를 **이차원 배열**로 나타내는 방식입니다. 인접 행렬을 adj[][][1]라고 한다면 adj[i][j]에 대해서 다음과 같이 정의할 수 있습니다.

> **adj[i][j]** : 노드 i에서 노드 j로 가는 간선이 있으면 1, 아니면 0

cf) 만약 간선에 가중치가 있는 그래프라면 1 대신에 가중치의 값을 직접 넣어주는 방식으로 구현할 수 있습니다.

DFS의 구현 – 관계 구현(인접행렬)



0	1	1	1
1	0	1	0
1	1	0	1
1	0	1	0

DFS의 구현 – 관계 구현

```java
static int[][] graph;
```

```java
static boolean[] visited;
```

```java
for (int i = 1; i <= N; i++) {
    graph[i][i] = 1;
}
```

```java
for (int i = 0; i < M; i++) {
    st = new StringTokenizer(br.readLine());
    int a = Integer.parseInt(st.nextToken());
    int b = Integer.parseInt(st.nextToken());
    graph[a][b] = 1;
    graph[b][a] = 1;
}
```

Graph = 관계를 표시할 저장소

Visited = 방문할 노드를 표시할 Boolean 배열

본인과 본인은 연결되어있으므로 초기화

선이 방향을 아닌 양방향이므로 서로 연결해준다.

사실.. 구현은 비슷합니다

BFS의 구현 - 큐

```java
public static void bfs(int start) {  no usages  new *
    Queue<Integer> queue = new LinkedList<>(); // BFS에 사용할 큐 초기화
    queue.add(start); // 시작 노드를 큐에 추가

    while (!queue.isEmpty()) { // 큐가 비어 있지 않은 동안 반복
        int currentNode = queue.poll(); // 큐에서 현재 노드를 꺼냄

        // 현재 노드가 아직 방문되지 않았다면
        if (!visited[currentNode]) {
            System.out.println(currentNode);// 방문한 노드를 기록
            visited[currentNode] = true; // 현재 노드를 방문으로 표시

            // 현재 노드의 모든 인접 노드를 탐색
            for (int i = 1; i <= N; i++) {
                // 인접 노드이고 아직 방문하지 않은 경우
                if (graph[currentNode][i] == 1 && !visited[i]) {
                    queue.add(i); // 인접 노드를 큐에 추가
                }
            }
        }
    }
}
```

DFS의 구현 - 스택

```java
public static void dfs2(int start) {  1 usage
    Stack<Integer> stack = new Stack<>();
    stack.push(start);

    while (!stack.isEmpty()) {
        int top = stack.pop();
        if(!visited[top]){
            visited[top] = true;
            for (int i = N; i >= 1; i--) {
                if (graph[top][i] == 1 && !visited[i]) {
                    stack.push(i);
                }
            }
        }
    }
}
```

비교해보기

```java
public static void dfs2(int start) {  1 usage
    Stack<Integer> stack = new Stack<>();
    stack.push(start);

    while (!stack.isEmpty()) {
        int top = stack.pop();
        if(!visited[top]){
            visited[top] = true;
            for (int i = N; i >= 1; i--) {
                if (graph[top][i] == 1 && !visited[i]) {
                    stack.push(i);
                }
            }
        }
    }
}
```

```java
public static void bfs(int start) {  no usages  new *
    Queue<Integer> queue = new LinkedList<>(); // BFS에 사용할 큐 초기화
    queue.add(start); // 시작 노드를 큐에 추가

    while (!queue.isEmpty()) { // 큐가 비어 있지 않은 동안 반복
        int currentNode = queue.poll(); // 큐에서 현재 노드를 꺼냄

        // 현재 노드가 아직 방문되지 않았다면
        if (!visited[currentNode]) {
            System.out.println(currentNode);// 방문한 노드를 기록
            visited[currentNode] = true; // 현재 노드를 방문으로 표시

            // 현재 노드의 모든 인접 노드를 탐색
            for (int i = 1; i <= N; i++) {
                // 인접 노드이고 아직 방문하지 않은 경우
                if (graph[currentNode][i] == 1 && !visited[i]) {
                    queue.add(i); // 인접 노드를 큐에 추가
                }
            }
        }
    }
}
```

BFS 의 장단점

장점

- 답이 되는 경로가 여러 개인 경우에도 **최단경로임을 보장**한다.
- 최단 경로가 존재하면 깊이가 무한정 깊어진다고 해도 답을 찾을 수 있다.

단점

- 경로가 매우 길 경우에는 탐색 가지가 급격히 증가함에 따라 보다**많은 기억 공간을 필요로** 하게 된다.
- **해가 존재하지 않는다면** 유한 그래프(finite graph)의 경우에는 모든 그래프를 탐색한 후에 실패로 끝난다.
- 무한 그래프(infinite graph)의 경우에는 결코 해를 찾지도 못하고, 끝내지도 못한다.


DFS / BFS
KB
DREAM
WAVE
2030
BAEKJOON
ONLINE JUDGE
programmers
BOOK

DFS와 BFS

https://www.acmicpc.net/problem/1260


KB
DREAM
WAVE
2030


미로 탐색
1 실버 I
나를 알하는 건가

미로 탐색

실버 I

https://www.acmicpc.net/problem/2178


다시
오늘하루고생했다