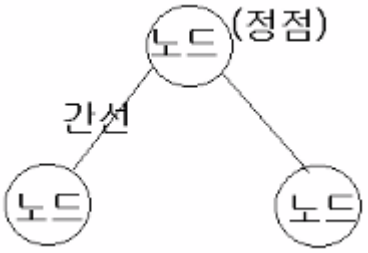
Day 5 - DFS, BFS

완전 탐색

- 모든 노드를 전체적으로 탐색하여 원하는 데이터를 찾는다 ⇒ 100% 코딩문제
- 크게 2가지로 나눔
    1. DFS(Depth-First-Search)
        a. 깊이 우선 탐색
    2. BFS(Breadth-First-Search)
        a. 너비 우선 탐색
- 스택및 큐를 이용하여 결과를 출력한다.
    - Stack(DFS)
    - Queue(BFS)

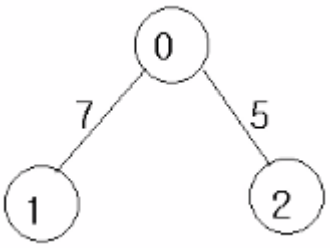


인접 행렬 vs 인접 리스트

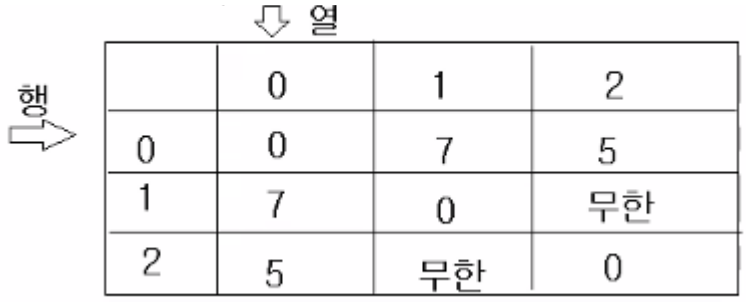
- **코딩 테스트에서 이 두가지 방식 모두 사용하여 프로그래밍을 한다.**

1. 인접 행렬: **2차원 배열로 그래프의 연결 관계 표현**
2. 인접 리스트: **리스트로 그래프의 연결 관계 표현**

그래프 배열화 예시)



1. **인접 행렬 (2차원 배열은 행과 열)**
    a. 각 노드가 연결된 형태를 기록하는 방식을 말한다.
    b. 간선의 가중치값을 기준으로 만들어진다.



	0	1	2
0	0	7	5
1	7	0	무한
2	5	무한	0

키워드: INF ⇒ 무한 (길의 가중치가 정확하게 정해져 있지 않는다)

```
INF = 999999999       # 무한 비용을 이야기한다 (가중치값)
```

 # 2차원 리스트로 인접 행렬을 표현 (인접 리스트 방식)

```
graph = [
        [0, 7, 5] ,
        [7, 0, INF],
        [5, INF, 0]
    ]
print(graph)
```

2. **인접 리스트 방식**
    a. **노드의 데이터를 저장하는 방식**
    b. **노드와 노드가 연결되어있는 데이터를 나열한다**

        0 → 1 → 2 # 0은 1, 2에 연결되어있다.

        1 → 0 # 1은 0에 연결되어있다.

        2 → 0 # 2는 0에 연결되어있다.

***Python에서는 Array를 리스트 자료형으로 표현한다.**

code example:

```
graph = [[] for _ in range(3)]
#노드 0에 연결된 노드 정보 저장(노드, 가중치)
graph[0].append((1, 7))
graph[0].append((2, 5))

# 노드 1에 연결된 정보
graph[1].append((0, 7))


# 노드 2에 연결된 정보
graph[2].append((0, 5))

print(graph)

# 결과물:
    [[(1, 7), (2, 5)],
     [(0, 7)],
     [(0, 5)]
    ]
```

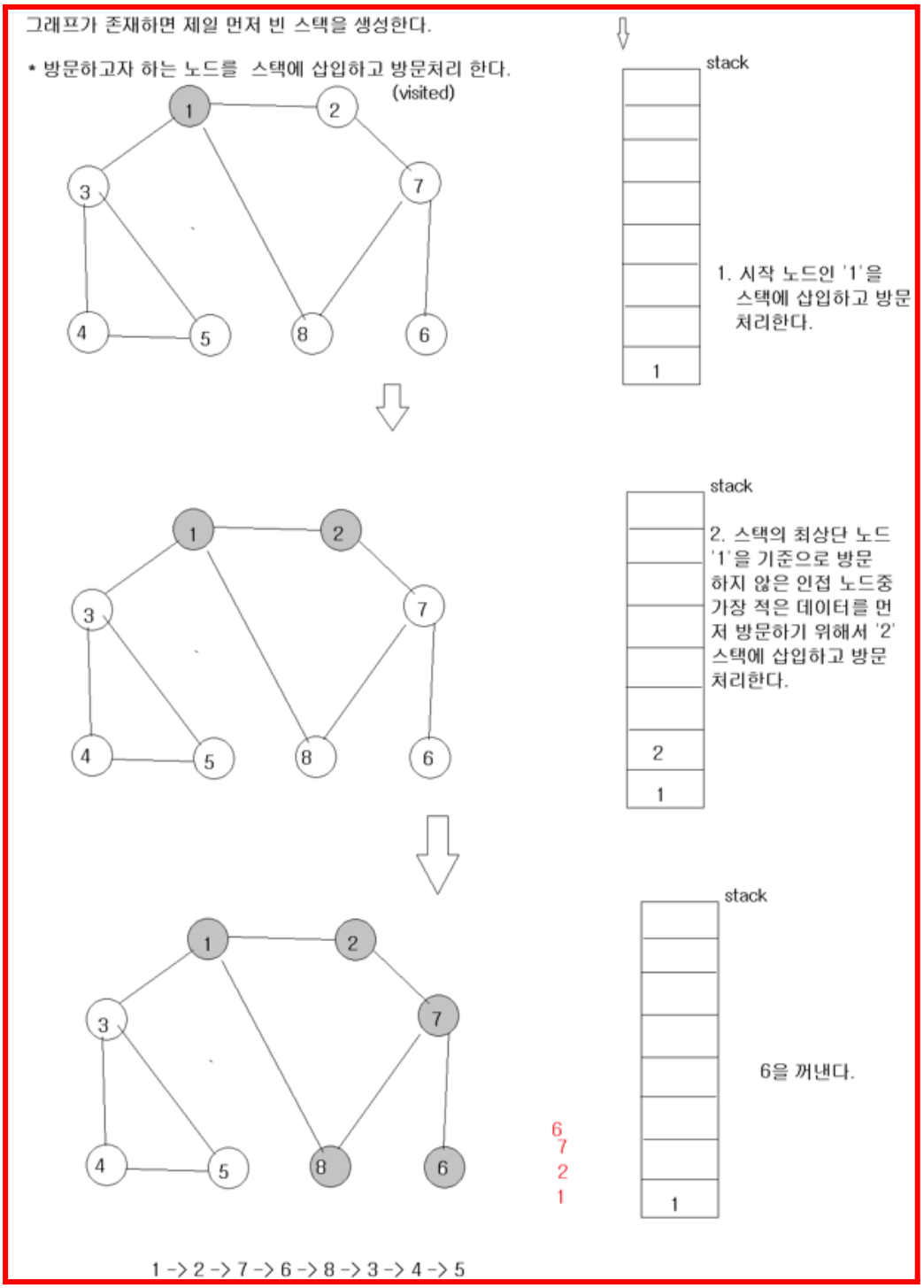
DFS(깊이 우선 탐색)

- stack의 방식으로 작용
- **순서와 상관없이 처리 되지만, 일반적으로 코딩 테스트에서는 어떠한 조건이 주어지지 아니하면 낮은 순서부터 처리하는 것이 기본.**

DFS 순서:

그래프가 존재하면 제일 먼저 빈 스택을 생성한다.

- 방문하고자 하는 노드를 스택에 삽입하고 방문처리(visited) 한다.



DFS 메서드 정의 구문:

```
def dfs(graph, v, visited):
    visited[v] = True
    print(v, end='')

    for i in graph[v]:
        if not visited[i]:
          dfs(graph, i, visited)

graph = [
    [],                 #node 0
    [2, 3, 8],          #node 1
    [1, 7],             #node 2
    [1, 4, 5],          #node 3
    [3, 5],             #node 4
    [3, 4],             #node 5
    [7],                #node 6
    [2, 6, 8],          #node 7
    [1, 7]              #node 8
    ]

visited = [False] * 9
dfs(graph, 1, visited)
```

문제 유형

1. 바이러스
2. 이동
3. 최단 거리 찾기 (가중치의 합)

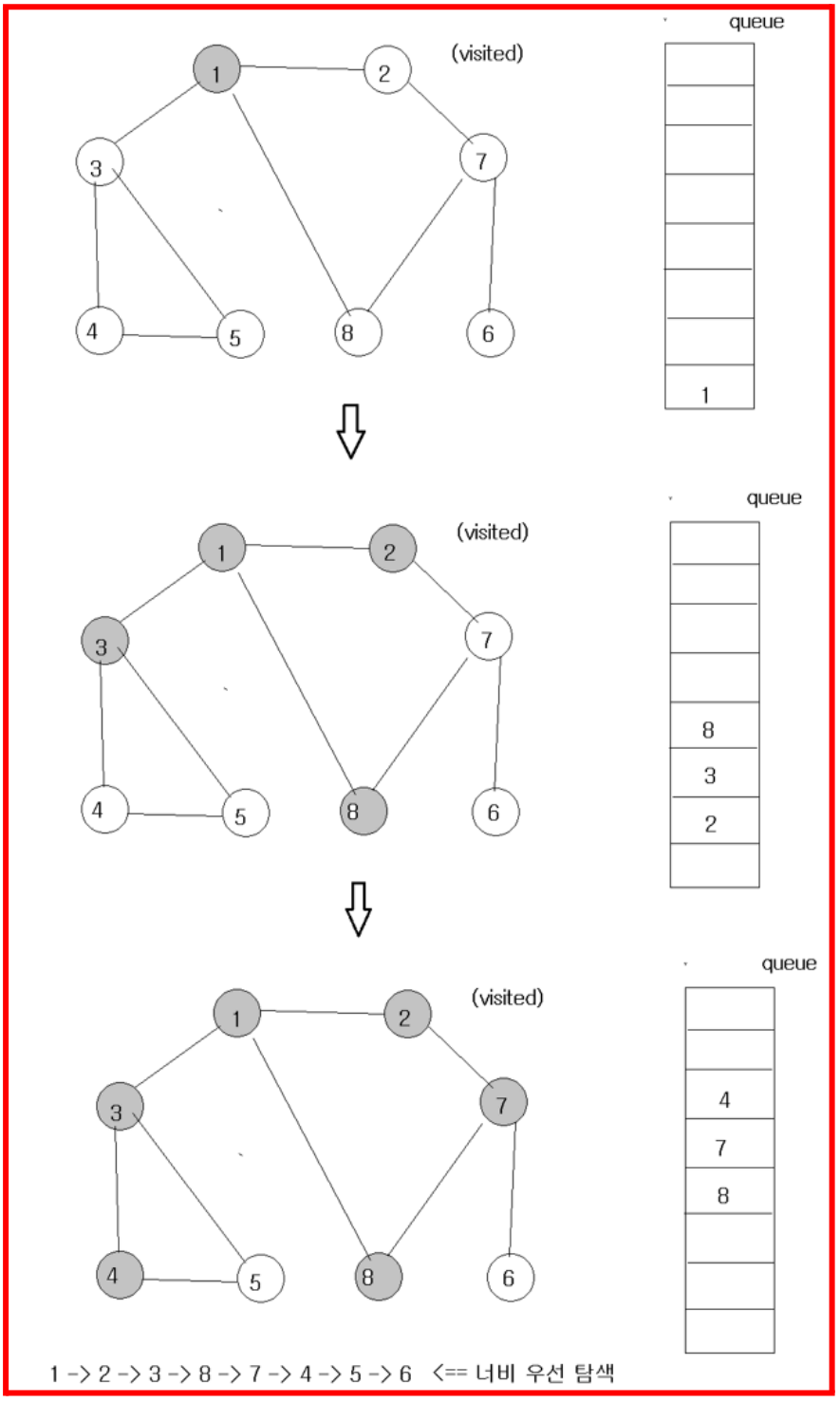
예시문제:

DFS, BFS.ipynb

BFS(너비 우선 탐색)

- **인접한 노드부터 다 탐색**

1. 빈 큐를 생성한다.
2. 탐색 시작 노드를 큐에 삽입한다. (가장 작은 데이터)
3. 큐에서 해당 노드를 꺼내서 인접 노드 중에서 방문하지 않은 노드를 모두 큐에 삽입한다.
4. 위의 3번 과정을 더 이상 수행 할 수 없을 때까지 반복한다.



BFS 정의 구문:

```
# BFS method using Deque

from collections import deque

def bfs(g, start, visited):
    queue = deque([start])
    visited[start] = True

    # 큐가 빌때까지 반복
    while queue:
        v = queue.popleft() ## --> popleft() 메서드가 의미하는 것은?
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
        print(v, end = ' ')

        # 방문한 노드를 기준으로 연결된 모든 모드를 큐에 삽입(방문하지 않은 노드)
        for i in g[v]:
            if not visited[i]:
                queue.append(i)
                visited[i] = True

graph = [
    [],                 #node 0
    [2, 3, 8],          #node 1
    [1, 7],             #node 2
    [1, 4, 5],          #node 3
    [3, 5],             #node 4
    [3, 4],             #node 5
    [7],                #node 6
    [2, 6, 8],          #node 7
    [1, 7]              #node 8
    ]

visited = [False] * 9      # 초기화
bfs(graph, 1, visited)
```

예시문제:

∞ DFS, BFS.ipynb

**** 완전 탐색 (DFS, BFS)**

- 재귀 호출이 적용된다.

	DFS	BFS
원리	스택	큐
방법	재귀 함수 이용	큐 자료구조를 이용