1. 그래프

노드와 노드 사이에 **연결된 간선의 정보**를

가지고 있는 자료구조

2. 그래프 문제를 해결하는 대표적인 탐색 기법

가. **Breadth-first search (너비 우선 탐색, BFS)**

graph = [

    [],

    [2,3,8],

    [1,7],

    [1,4,5],

    [3,5],

    [3,4],

    [7],

    [2,6,8],

    [1,7]

]

1) BFS는 간선의 수를 기반으로 탐색을 할 때에 적합

2) 최단거리, 최소 횟수, 임의의 경로

3) **현재 나의 위치에서 가장 가까운 노드들을 모두 방문**

4) 큐 자료구조 사용 (트리에서 레벨 탐색이 되기 때문)

from collections import deque

def BFS(graph, start, visited):

    queue = deque([start])

    visited[start] = True

    while queue:

        v = queue.popleft()

        print(v, end= ' ')

        for i in graph[v]:

            if not visited[i]:

                queue.append(i)

                visited[i] = True

BFS(graph, 1, visited)

나. **Depth-first search (깊이 우선 탐색, DFS)**

1) 경우의 수 문제

2) **현재 나의 위치에서 연결된 브랜치를 모두 방문 후 다음 브랜치로 넘거가는 방법**

3) 스택 자료구조 사용

def DFS(graph, v, visited):

    visited[v] = True

    print(v, end=' ')

    for i in graph[v]:

        if not visited[i]:

            DFS(graph, i, visited)

DFS(graph, 1, visited)

\* 참고 : 디스뻐큐 : DFS-스택, BFS-큐 (이렇게까지 해야만 속이 후련했냐.. 꼭 기억해주겠지)

\* 절대는 아님! DFS를 큐로, BFS를 스택으로 해도 됨. 큐로 스택을 구현할 수 있는 것처럼. 다만 보편적이고 편하다는 것

3. 그래프 vs DP ? https://edu.goorm.io/qna/12272

가. 그래프 : 각 층에 대한 모든 정보를 큐에 담아서 다음 계산을 할 때 활용하는 방식

1) 경우의 수가 무한 (재귀적 점화식 불가)

나. DP : 계산 정보를 DP 테이블에 담아두고, 다음 층 계산시 활용하는 방법

1) 재귀적인 점화식 먼저 설계, 조건을 만족하면 캐싱

2) 재귀적인 점화식의 기본적인 조건? 재귀가 종료되는 case가 존재

3) 모든 경우의 수를 따져본다

4) **문제의 해가 중복되는 하위문제 해를 사용하여 풀 수 있음**

def fibo(x):

if x == 1 or x == 2:

return 1

return fibo(x-1) + fibo(x-2)

**DP 메모제이션(캐싱) 기법**

**d = [0] \* 50 # 답 저장할 공간**

def fibo(x):

if x == 1 or x == 2:

return 1

**if d[x] != 0: # 이미 답이 있다면**

**return d[x] # 저장된 값 불러옴**

d[x] = fibo(x-1) + fibo(x-2)

return d[x]