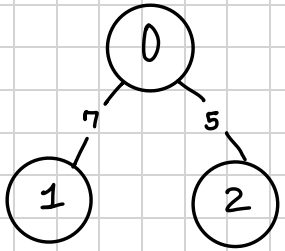
5장 DFS | BFS

✦ DFS | BFS 를 수행하기 위해서 가장 중요한 개념
1. 스택
2. 큐
3. 재귀 함수 (~= 스택)



그래프를 표현하는 두 가지 방법

✦ 인접 행렬 -> 2차원 배열로 그래프의 연결 관계를 표현함

```
graph = [
        [0, 7, 5],
        [7, 0, INF],
        [5, INF, 0]
    ]
```

✦ 인접 리스트 -> 리스트로 그래프의 연결 관계를 표현함

```
graph = [[] for _ in range(3)]
graph[0].append((1,7))
graph[0].append((2,5))
graph[1].append((0,7))
graph[2].append((0,5))
```

DFS : 깊이 우선 탐색

그래프에서 깊은 부분을 우선적으로 탐색하는 알고리즘 -> 노드 Node와 간선 Edge로 표현
선입후출 방식인 스택 자료구조 사용 (python에서는 list가 stack 자료형임)

1. 탐색 시작 노드를 스택에 삽입 (dfs 함수에 시작 노드 v 전달)
2. 해당 노드 방문 처리 (visited 배열 True로)
3. 스택 최상단 노드의 인접 노드 중 방문하지 않은 노드 k가 있으면 (visited 배열이 False라면), k를 스택에 넣고 방문 처리함.
4. 최상단 노드의 인접 노드를 모두 방문했다면 스택의 최상단 노드를 꺼냄. (stack에서 pop)
5. 스택이 빌 때까지 반복함.

☆ 재귀함수가 곧 스택임.
가장 최근에 불린 재귀함수부터 차례대로 처리하기 때문

3번 음료수 얼려 먹기

입력 :
001
010
101

Graph =
[[0,0,1],
[0,1,0],
[1,0,1]]

Dfs 함수가 수행해야 할 일
1. 접근하는 graph의 요소가 0인 경우 1로 바꿈
2. 접근하는 graph의 요소가 1인 경우 False 반환함
3. 해당 요소에서 상,하,좌,우 위치도 모두 재귀적으로 호출함 ex.dfs(x-1,y)
4. 요소의 상, 하, 좌, 우가 모두 1인 경우(방문된 경우) return하며 결과값에 1을 더함.

BFS : 넓이 우선 탐색

가까운 노드부터 탐색하는 알고리즘 -> 노드 Node와 간선 Edge로 표현

선입선출 방식인 큐 자료구조 사용 (python에서는 collections 모듈의 deque)

1. 탐색 시작 노드를 큐에 삽입하고 (deque에 append) 방문 처리(visited 배열 True로)
2. 큐에서 노드 k를 꺼내 (popleft()) 해당 노드 k의 인접 노드 중 방문하지 않은 노드(graph[k]의 원소 중 visited가 False인)를 모두 큐에 삽입(deque에 append)
3. 큐에 넣은 노드들 방문 처리(visited 배열 True로)
3. 큐가 빌 때까지 반복 (while queue)

```
nx = x + dx[i]
ny = y + dy[i]

if graph[nx][ny] == 1:
    graph[nx][ny] = graph[x][y] + 1
    queue.append((nx, ny))
```

3장 그리디

✦ 현재 상황에서 지금 당장 좋은 것만 고르는 방법

✦ 즉, 가장 좋아 보이는 것만을 선택해도 문제를 풀 수 있는지를 파악해야 함.

✓ '가장 큰 순서대로', '가장 작은 순서대로'와 같은 기준에 유의할 것

거스름돈

✦ '가장 큰 화폐 단위부터' 돈을 거슬러 준다 -> 배수 형태이므로 큰 화폐부터 차례대로 검사 진행하기

1이 될 때까지

✦ '빼기'보다 '나누기'를 많이 수행해야 수행 횟수를 줄일 수 있음.

✦ '최대한 많이 나누는' 코드를 작성하면 된다.

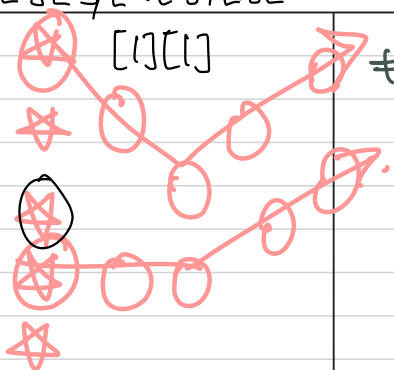
3109번

dfs[ ][y+1]

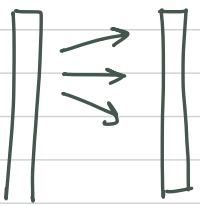
[-1][1]

[0][0]/[0][1]/[0][2]

[1][1]



입력 → mapping



visited 배열 → 방문한적 있으면 끝

최대개수이므로 이중for로

cnt 개수 → global로 처리

dx = [-1, 0, 1]

dx = x + (-1)
dx = x + (0)
dx = x + (1)

DFS

파이프라인의 최대 개수를 구해야 함

✦ 각 열마다 진행하되, 파이프끼리 겹치지 않아야 함.

✦ 즉, '겹치지 않도록'(visited 배열 이용하기) 코드를 작성해야 함.

✦ '오른쪽 위, 오른쪽, 오른쪽 아래 대각선 순서로 연결 시도' 하는 코드를 작성해야 함. -> 어차피 파이프를 설치할 때에는 가장 위쪽에 밀착되도록 하는 시도를 먼저 해야 하기 때문.

#2839번

5를 최대한 많이. → (N-5)%3 / (N-5-5)%3) until N-5*k < 3

min 변수

k는 N//5

```
for i in range(k):
    if (N-5*k)%3 == 0:
        min_cnt = k+(N-k)%3
```

만약 (k==어쩌고) and min_cnt==0

(N-5*k)%3 == 0 인경우

k개 + (N-5*k)//3

예제 3-1 거스름돈과 유사하지만, 3킬로그램과 5킬로그램이 서로소 관계라는 차이점이 존재함.

✦ '5를 최대한 많이 포함'해야 하므로 증가하는 수 k를 통해 5*k를 뺀 값이 3으로 나누어지는지를 확인하는 코드

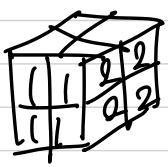
✦ '거스름돈' 문제와 달리 배수 형태도 아니고, 정확하게 N킬로그램을 만들 수 없을 수도 있음. -> min_count가 초기화되지 않고 0인 경우, -1을 출력하도록 작성

#1041번

<6을 겹치게>

(1+2+3+4+5)*

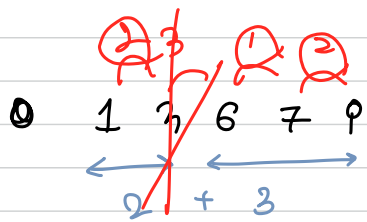
1*8+



#2212번

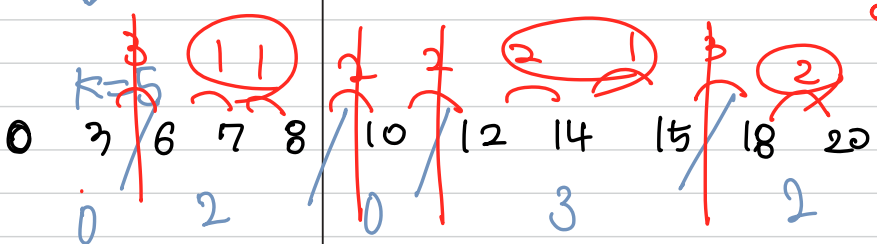
~~센서 → 문자 처리~~

예제1 ⇒



4개 (K-1) ⇒ 안씀

예제2 ⇓



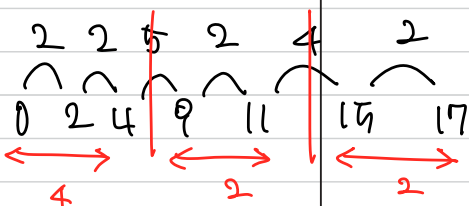
9개 (K-1) ⇒ 안씀

10%

각 index마다 차이를 구하기

ex) lst[1] - lst[0]

lst[2] - lst[1]



3개 설치

→ 거리가 가장 긴 (3-1)개의 구간 제외

수신 가능 영역의 길이의 합의 최솟값

✦ 센서 간의 거리가 가장 긴 구간에 집중국 설치 -> 최솟값을 구하기 위해서 '가장 긴 거리'들을 제외함

✦ set 자료형이 아니어도 되는 이유 - 어차피 sorted 되어 있는 상태이므로, 중복을 제거하지 않아도 같은 수(위치)끼리의 거리는 0으로 계산됨.