[SWEA]1219,4871

0223

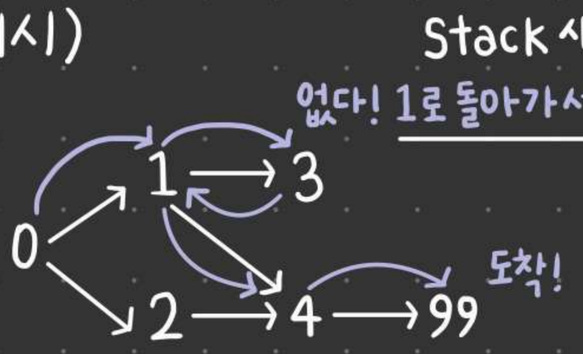
그림과 같이 도식화한 지도에서 A도시에서 출발하여 B 도시로 가는 길이 존재하는 지 조사하려고 한다.
길 중간 중간에는 최대 2개의 갈림길이 존재하고, 모든 길은 일방 통행으로 되돌아오는 것이 불가능하다.
다음과 같이 길이 주어질 때, A도시에서 B도시로 가는 길이 존재하는지 알아내는 프로그램을 작성하여라.

1. A와 B는 숫자 0과 99로 고정된다.
2. 모든 길은 순서쌍으로 나타내어진다. 위 예시에서 2번에서 출발 할 수 있는 길의 표현은 (2, 5), (2, 9)로 나타낼 수 있다.
3. 가는 길의 개수와 상관없이 한가지 길이라도 존재한다면 길이 존재하는 것이다.
4. 단 화살표 방향을 거슬러 돌아갈 수는 없다.

[제약 사항]
출발 점은 0, 도착점은 99로 표현된다. 정점(분기점)의 개수는 98개(출발점과 도착점 제외)를 넘어가지 않으며, 한 개의 정점에서 선택할 수 있는 길의 개수도 2개를 넘어가지 않는다.

[입력]
각 테스트 케이스의 첫 줄에는 테스트 케이스의 번호와 길의 총 개수(N)가 주어지고 그 다음 줄에는 순서쌍(arr)이 주어진다. 순서쌍의 경우, 별도로 나누어 표현되는 것이 아니라 숫자의 나열이며, 나열된 순서대로 순서쌍을 이룬다.

예시)



Stack 사용!
없다! 1로 돌아가서 다시 길찾기 → 방문한 적 있는 길인지 확인하는 리스트 만들기!

0	1	2	3	4	99
T/F	T/F	T/F	T/F	T/F	T/F

99가 T가 되면 탐색 중단!

(1) 지도를 리스트로 표현하기 (인접행렬, 반복문)

(2) 스택을 이용하여 지도 탐색 (반복문)

조건 i) 시작정점에서 인접한 정점이 방문하지 않은곳이면 이동!
조건 ii) 인접한 정점이 모두 방문한 곳이면 이전정점으로 되돌아가기
└ 방문하지 않은 인접정점 발견 or 이전정점이 없을 때까지 반복
조건 iii) 방문한 정점이 목표지점이면 반복중단

```
Stack = [0]  시작정점을 기본값!
visited = [False] * 100  방문리스트
visited[0] = True

while stack:  stack이 비면 이전정점이 없는 것! → 반복문 중단
    현재도착한정점 = stack[-1]
    if 현재도착한정점 == 99 → break!
    for i in 인접행렬 column
        if 정점에서 인접한 정점이 방문하지 않은곳이면 이동!
        else stack.pop()
else:
    실패 ㅠㅠ
```

* 재귀

```
def DFS(v):    v는 정점
    visited[v] = 1
    for i in 인접행렬 column
        if 정점에서 인접한 정점이 방문하지 않은곳이면
            DFS(i)
```

[SWEA]1224

0223

문자열로 이루어진 계산식이 주어질 때, 이 계산식을 후위 표기식으로 바꾸어 계산하는 프로그램을 작성하시오.
문자열 계산식을 구성하는 연산자는 +. * 두 종류이며 문자열 중간에 괄호가 들어갈 수 있다.
이 때, 괄호의 유효성 여부는 항상 옳은 경우만 주어진다.

피연산자인 숫자는 0~ 9 정수만 주어진다.

ex) word = 3+(4+5)*6+7

(1) 중위표기식의 후위표기식 변환 — 순회

토큰	iSP	icp
*, /	2	2
+, −	1	1
(	0	3

i) 피연산자면 list에 추가

ii) 연산자면
- Stack이 비어있다 → 추가!
- Stack[-1]의 iSP 점수와 토큰의 icP 점수비교!
  - 토큰의 icP가 더크면 Stack에 추가
  - 토큰의 icP가 더작거나 같으면 Stack[-1]제거후 list에추가
    (Stack이 비거나 토큰의 icP가 Stack[-1]의 iSP 보다 커질 때까지 제거)

iii) 닫는 괄호면, Stack[-1]이 여는괄호일때까지 Stack[-1]제거후 list에추가

for token in word:
  if 연산자라면
    if 스택이 비어있으면 ⟶ 추가!
    elif 토큰의 icP가 Stack[-1]의 iSP 점수보다 더크면 ⟶ 추가!
    else 토큰의 icP가 Stack[-1]의 iSP 보다 커질 때까지 Stack[-1]제거후 list에추가
      └ Stack이 비면 break!
  elif: 닫는괄호
  else: 피연산자

while stack
  스택이 빌 때까지 POP! & 리스트에 추가

(2) 후위표기법의 수식을 Stack을 이용해 계산

i) 피연산자면 Stack에 저장

ii) 연산자면 Stack[-2][-1] 이용해서 수식계산

for i in 리스트 :
  if 피연산자 ⟶ Stack 저장
  elif 연산자 ⟶ 연산결과 Stack에 저장