동적계획법(DP)

⭐ N으로 표현 ⭐

```
아래와 같이 사칙연산만으로 12를 표현할 수 있습니다
12 = 5 + 5 + (5/5) + (5/5)  # 5를 사용한 횟수 = 6
12 = 55/5 + 5/5             # 5를 사용한 횟수 = 5
12 = (55 + 5)/5             # 5를 사용한 횟수 = 4

5를 가장 적게 사용한 경우는 4번 사용했을 때이다
이처럼 숫자 N과 number가 주어질 때 (위의 예에서는 N=5, number=12)
number를 표현할 수 있는 N의 최소개수를 리턴하시오

* 사칙연산(+,-,*,/)만 사용하기 + N이 concatenate해서 만들어지는 수도 인정 (55, 555, 5555 ...)
* 나누기에서 나머지는 무시한다
* N 사용횟수의 최소가 8번보다 크면 return -1



# 풀이
- (op)은 각 사칙연산
- N을 2번 사용했을 경우 만들어지는 수는
(N을 1번 사용했을 경우 만들어지는 수) (op) (N을 1번 사용했을 경우 만들어지는 수)로 표현가능

- N을 3번 사용했을 경우는
(1번 사용) (op) (2번 사용)
(2번 사용) (op) (1번 사용)

- N을 K번 사용했을 경우는
(1번 사용) (op) (K-1번 사용)
(2번 사용) (op) (K-2번 사용)
....
(K-1번 사용) (op) (1번 사용)

각 J번 사용했을 때 만들어지는 수들을 set으로 저장하고(중복된 수는 제거)
outcomes[J]에 set을 저장(memoization)
-> J+1번 사용했을 때 만들어지는 수를 구할 때 사용된다
-> Dynamic programming!
```

- 답

```
def makeOutcomesSet(outcomes, N, usage):
    newSet = set()
    newSet.add(int(str(N)*usage))  # N이 concatenate하는 경우 추가(N, NN, NNN ...)
```

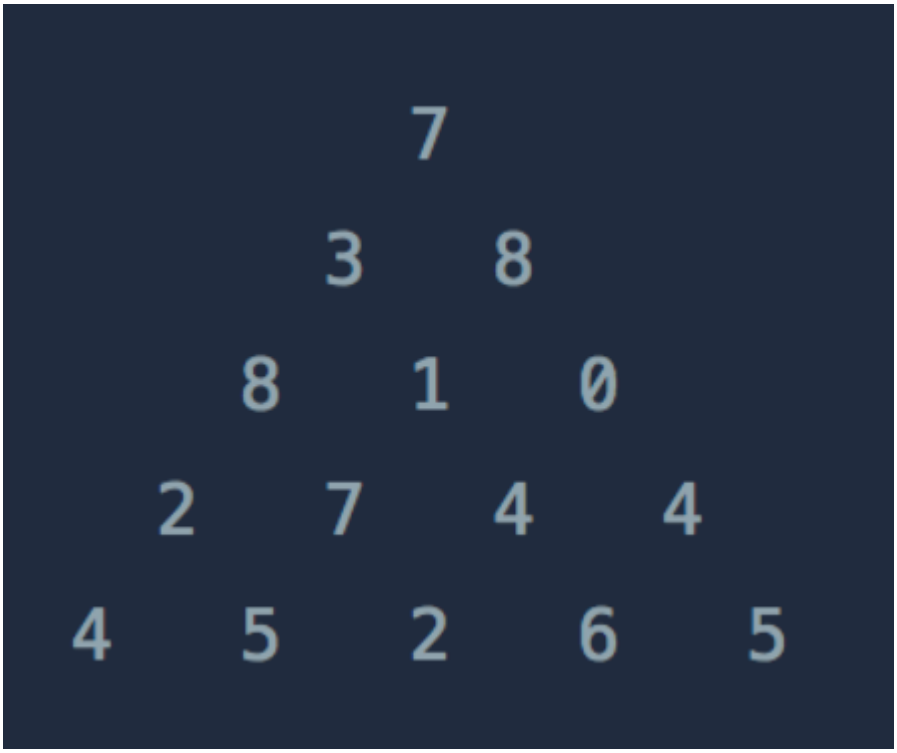
```
    for i in range(1, usage):
        # newSet.add(outcomes[i-1] (op) outcomes[usage-i])
        for item1 in outcomes[i-1]:
            for item2 in outcomes[usage-i-1]:
                newSet.add(item1 + item2)
                newSet.add(item1 - item2)
                newSet.add(item1 * item2)
                if item2 != 0 :
                    newSet.add(item1 // item2) # 나누기에서 나머지는 무시
    return newSet


def solution(N, number):
    answer = 0
    outcomes = [] # N을 1번, 2번, 3번... 8번 사용했을 경우 만들어지는 수의 set들을 저장

    for i in range(0, 8):
        if i == 0 : # N을 1번 사용할 경우 만들어지는 수는 N 한개임
            outcomes.append({N})
        else :
            outcomes.append(makeOutcomesSet(outcomes, N, i+1))
        if number in outcomes[i]:
            answer = i+1
            return answer

    answer = -1
    return answer
```

정수 삼각형



- 삼각형의 꼭대기부터 바닥까지 이동하면서 거쳐가는 수들을 모두 더한다. 마지막 레벨까지 완료했을 때 거쳐간 숫자의 합이 가장 큰 경우를 리턴
    - 아래 레벨로 이동할 때는 자신의 대각선 방향으로밖에 이동할 수 없음
- 풀이
    - 그리디와 구분! : 현재 위치에서 좌우 대각선 방향을 보고 더 큰 값을 골라간다고 해서 결과가 최댓값이 되지는 않음.
    - 각 레벨의 각 위치에 도달할 때까지의 최대값을 저장, 다음 레벨에서는 그 전 레벨에서의 최대값들을 보고 해당 레벨 위치들의 최대값 산정
    - 작은 삼각형부터 해결 (bottom-up) 방식

```
def convertLevel(target, ref):
    # 현재 레벨의 값들을 각 위치에 도달하기까지의 최댓값으로 변경함
    # target : 현재 레벨의 값 리스트
    # ref : 그 전 레벨(각 위치에서의 최댓값들을 보관중)
    for idx in range(0, len(target)):
        if idx == 0: # 가장 왼쪽 자리 (부모가 1개)
            target[idx] += ref[idx]
        elif idx == len(target) - 1: # 가장 오른쪽 자리 (부모가 1개)
            target[idx] += ref[idx-1]
        else: # 부모가 2개일 경우 각 부모를 현재 값에 더해보고 더 큰 값 선택
            target[idx] = max(ref[idx-1]+target[idx], ref[idx]+target[idx])


def solution(triangle):
    answer = 0
    for i in range(0, len(triangle)):
        if i == 0:
            continue
        convertLevel(triangle[i], triangle[i-1])

    answer = max(triangle[-1]) # 마지막 레벨의 값들 중 최댓값이 답
    return answer
```