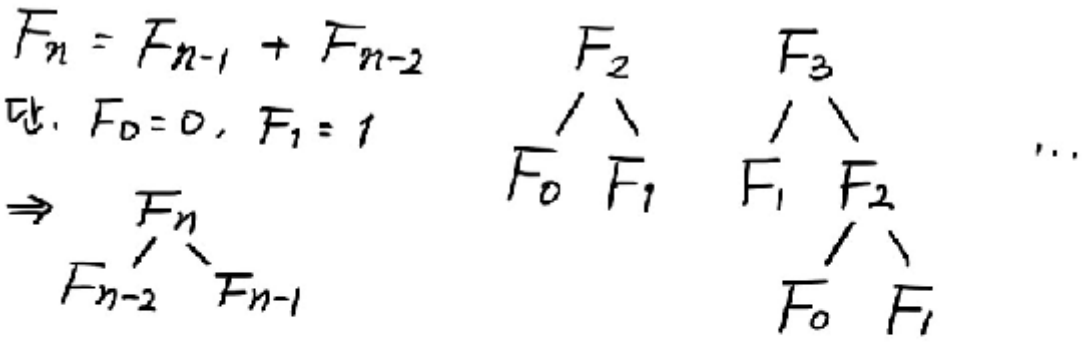
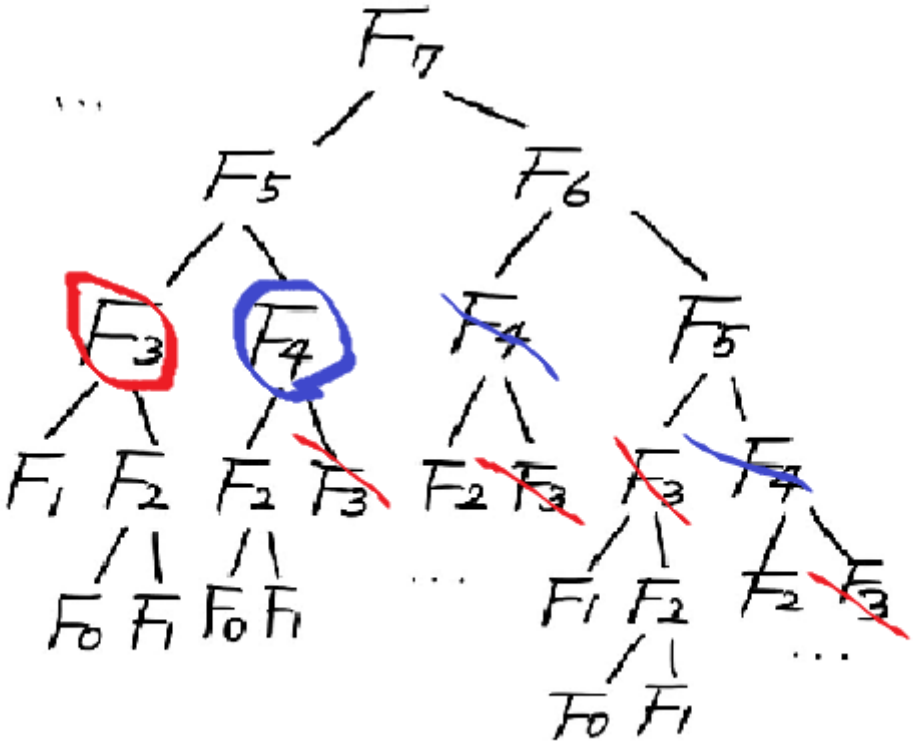
DP

~~최적 부분 문제에 대한~~ 불필요한 반복 연산을

메모리를 활용하여

실행 시간을 줄이는 알고리즘





<이항 계수 nC(n/2)를 계산할 때 필요한 함수 호출의 수>

n	2	3	4	...	18	19	...	24	25	...
recursion	3	5	11		97239	184755		5408311	10400599	
DP	3	5	8		99	109		168	181	

DP 문제 성립 조건

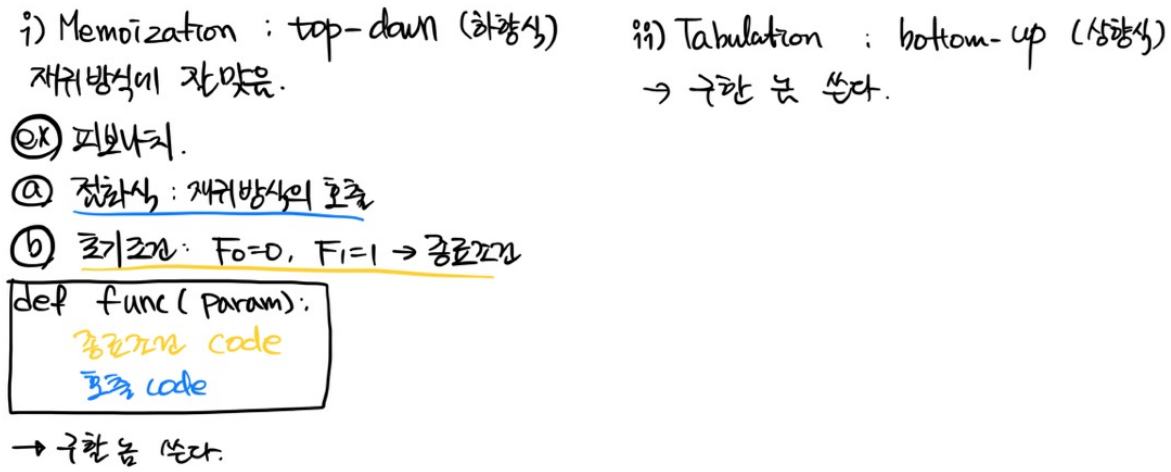
1) 최적 부분 구조 : 부분 문제의 최적해를 통해 전체 문제의 최적해를 얻을 수 있다.

2) 중복된 하위 문제 : 두 번 이상 계산되는 부분 문제

3) 참조적 투명성 : 동일한 입력, 동일한 출력

메모리를 만들어 계산 결과를 담아놓는다.

구현 방식



i)	ii)
def f(n):	

시간 복잡도

(부분 문제의 개수) x (부분 문제 하나당 시간 복잡도)

📌 프로그래머스

N으로 표현

문제 설명

아래와 같이 5와 사칙연산만으로 12를 표현할 수 있습니다.

12 = 5 + 5 + (5 / 5) + (5 / 5)12 = 55 / 5 + 5 / 512 = (55 + 5) / 5

5를 사용한 횟수는 각각 6,5,4 입니다. 그리고 이중 가장 작은 경우는 4입니다.이처럼 숫자 N과 number가 주어질 때, N과 사칙연산만 사용해서 표현 할 수 있는 방법 중 N 사용횟수의 최솟값을 return 하도록 solution 함수를 작성하세요.

제한사항

- N은 1 이상 9 이하입니다.
- number는 1 이상 32,000 이하입니다.
- 수식에는 괄호와 사칙연산만 가능하며 나누기 연산에서 나머지는 무시합니다.
- 최솟값이 8보다 크면 -1을 return 합니다.

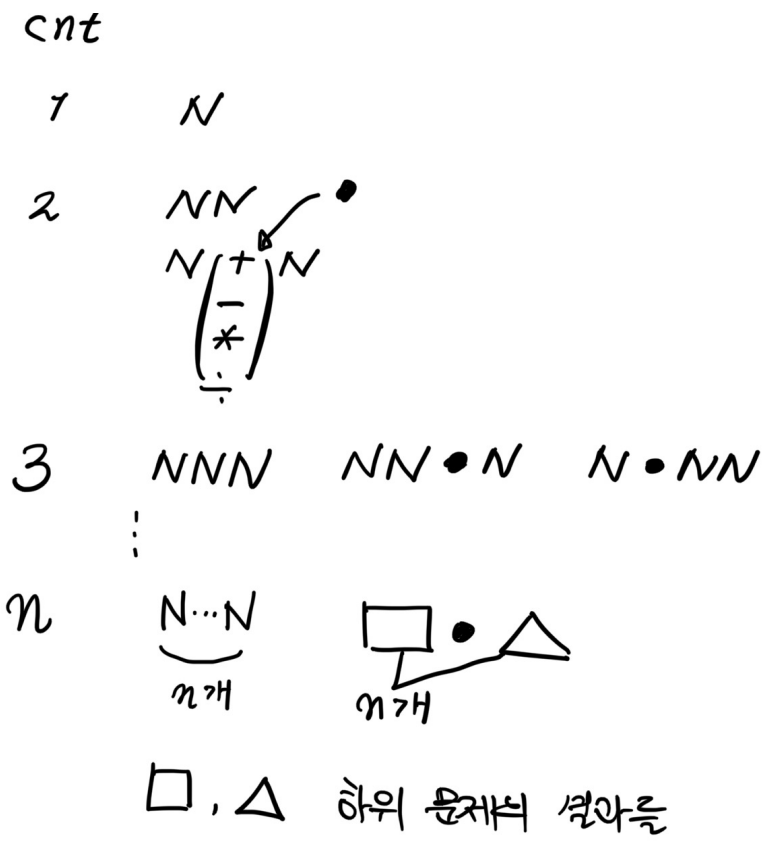
예시

N	number	return
5	12	4
2	11	3

1. 문제 설명 예시
2. 22//2 = 11, 2를 3번 사용

구현 아이디어

1. **최솟값이 8보다 크면 -1을 리턴**
3. 디자인에 대한 힌트가 없다고 판단.
4. 재귀 ⇒ 문제 목적이 문장으로 잘 풀어지는지?
    - N을 k개 써서 number를 만들 때에 최소의 k
    - k를 1부터 호출한다면... {N을 k개 써서 number를 만들 수 있는지?}
4. 손으로 풀기



5. **하위 문제의 결과들 ⇒ DP**

Tabulation이 더 나은 이유

최대 k의 값이 8로 작긴 하지만

k = 1부터 호출하는 타뷸레이션 방식이 더 자연스럽고 효율적이다.

코드

```
def make_num(cnt, memo, N, num):
    if cnt == 9:
        return -1
    temp = set()
    temp.add(int(str(N)*cnt))

    for n1 in range(1, cnt):
        n2 = cnt - n1
        if n2 == 0:
```

```
            break
        for i1 in range(len(memo[n1])):
            for i2 in range(len(memo[n2])):
                temp.add(memo[n1][i1] + memo[n2][i2])
                temp.add(memo[n1][i1] - memo[n2][i2])
                temp.add(memo[n1][i1] * memo[n2][i2])
                if memo[n2][i2]:
                    temp.add(memo[n1][i1] // memo[n2][i2])

    if num in temp:
        return cnt
    else:
        memo[cnt] = list(temp)
        return make_num(cnt + 1, memo, N, num)

def solution(N, number):
    answer = 0
    if N == number:
        return 1
    memo = [[] for _ in range(9)]
    answer = make_num(1, memo, N, number)

    return answer
```

등굣길

문제 설명

계속되는 폭우로 일부 지역이 물에 잠겼습니다. 물에 잠기지 않은 지역을 통해 학교를 가려고 합니다. 집에서 학교까지 가는 길은 m x n 크기의 격자모양으로 나타낼 수 있습니다.

아래 그림은 m = 4, n = 3 인 경우입니다.

가장 왼쪽 위, 즉 집이 있는 곳의 좌표는 (1, 1)로 나타내고 가장 오른쪽 아래, 즉 학교가 있는 곳의 좌표는 (m, n)으로 나타냅니다.

격자의 크기 m, n과 물이 잠긴 지역의 좌표를 담은 2차원 배열 puddles이 매개변수로 주어집니다. **오른쪽과 아래쪽으로만 움직여** 집에서 학교까지 갈 수 있는 최단경로의 개수를 1,000,000,007로 나눈 나머지를 return 하도록 solution 함수를 작성해주세요.

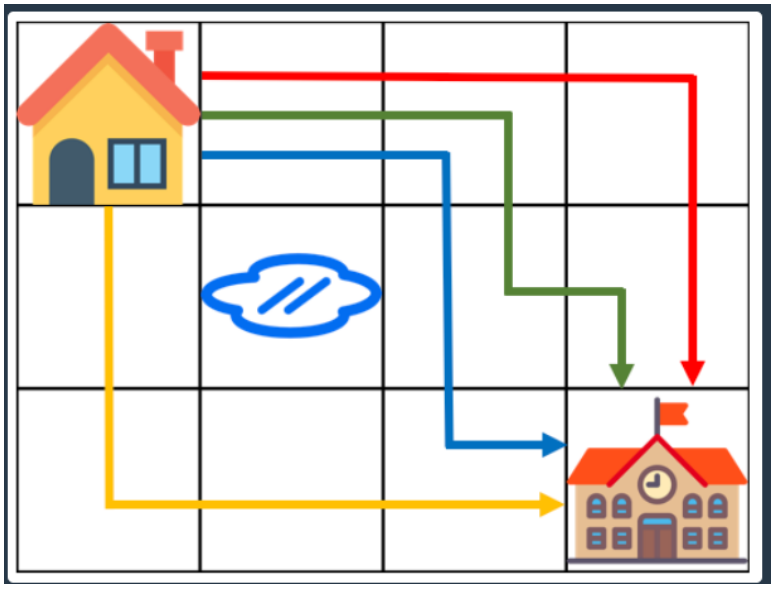
제한사항

- 격자의 크기 m, n은 1 이상 100 이하인 자연수입니다.
    - m과 n이 모두 1인 경우는 입력으로 주어지지 않습니다.
- 물에 잠긴 지역은 0개 이상 10개 이하입니다.

- 집과 학교가 물에 잠긴 경우는 입력으로 주어지지 않습니다.

예시

m	n	puddle	return
4	3	[[2,2]]`	4



구현 아이디어

1. 오른쪽, 아래로만 간다 ⇒ 도착하는 경우의 수가 최단 경로
2. 중학교 최단 경로 문제
3. DP 요건 또한 충족
4. modular 연산

코드

```
from collections import deque

def go_school(graph, m, n):
    D   = [[1, 0], [0, 1]]
```

```
    MOD = 1000000007

    memo  = [[0] * m for _ in range(n)]
    visit = [[0] * m for _ in range(n)]
    memo[0][0] = 1

    q = deque()
    q.append((0, 0))
    visit[0][0] = 1

    while q:
        cr, cc = q.popleft()
        #if cr == n - 1 and cc == m - 1:
            #break

        for i in range(2):
            nr = cr + D[i][0]
            nc = cc + D[i][1]
            if nr < 0 or nr >= n or nc < 0 or nc >= m:
                continue
            if graph[nr][nc]:
                continue
            memo[nr][nc] += memo[cr][cc]
            memo[nr][nc] %= MOD
            if visit[nr][nc]:
                continue
            visit[nr][nc] = 1
            q.append((nr, nc))

    return memo[n - 1][m - 1]

def solution(m, n, puddles):
    graph = [[0] * m for _ in range(n)]

    for p in puddles:
        graph[p[1] - 1][p[0] - 1] = 1

    return go_school(graph, m, n)
```