AIVLE SCHOOL STUDY

다이나믹 프로그래밍 (DP)

01 다이나믹 프로그래밍 이란 _ 개념

= 동적 계획법

하나의 큰 문제를 여러 개의 작은 문제로 나누어서 그 결과를 저장하여
다시 큰 문제를 해결할 때 사용하는 문제해결 패러다임

재귀와 비슷해 보이는데 비효율적인 재귀를 더 효율적으로 하기 위해 만들어짐

02 다이나믹 프로그래밍 이란 _ 조건

01 최적 부분 구조 (Optimal Substructure)

큰 문제를 작은 문제로 나눌 수 있으며, 작은 문제의 답을 모아서
큰 문제를 해결 할 수 있는 경우

02 중복되는 부분 문제(Overlapping Subproblems)

동일한 작은 문제들이 반복하여 나타나는 경우

02 다이나믹 프로그래밍 이란 _ 조건

최적 부분 구조 (Optimal Substructure)

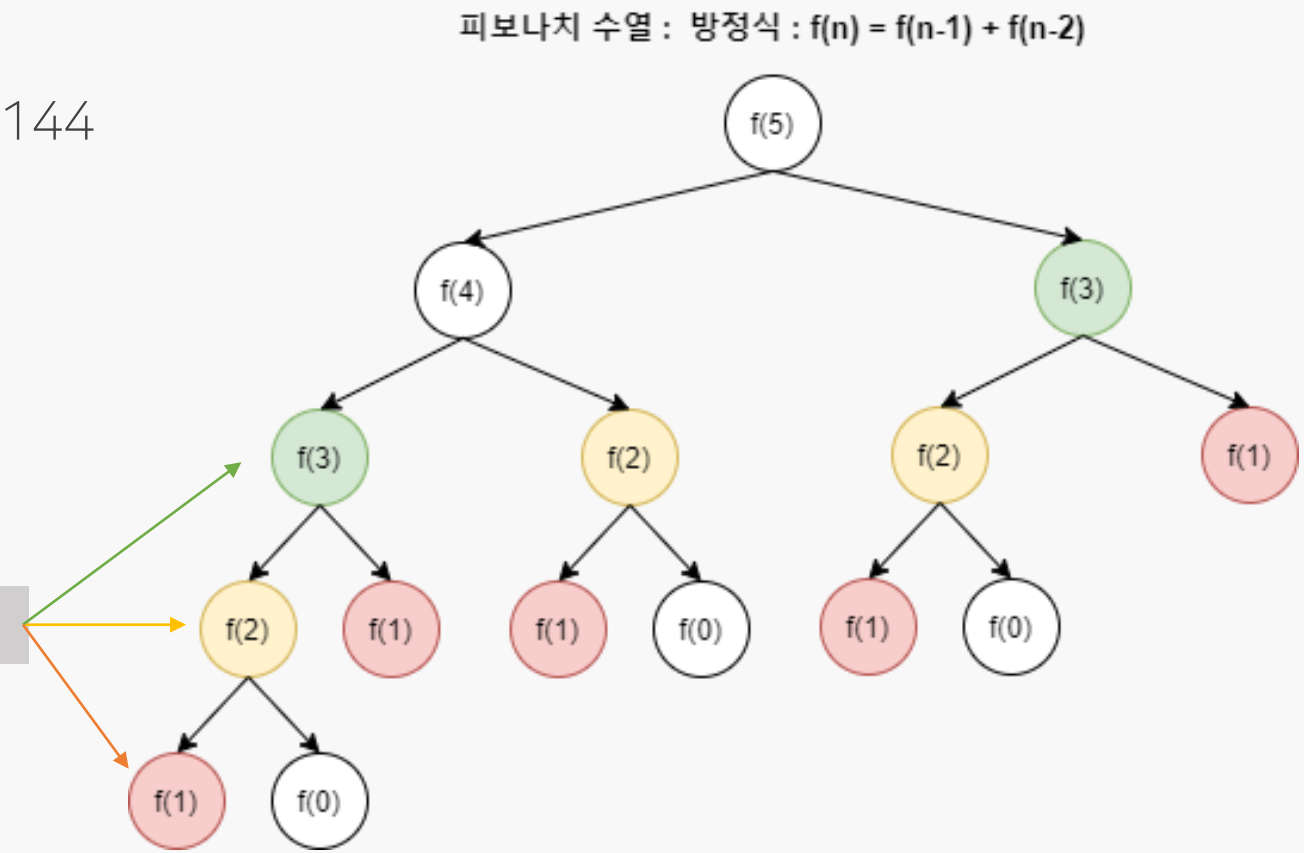
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144

3항(2) = 1항(1)+2항(2)

f(3) = f(1)+f(2)

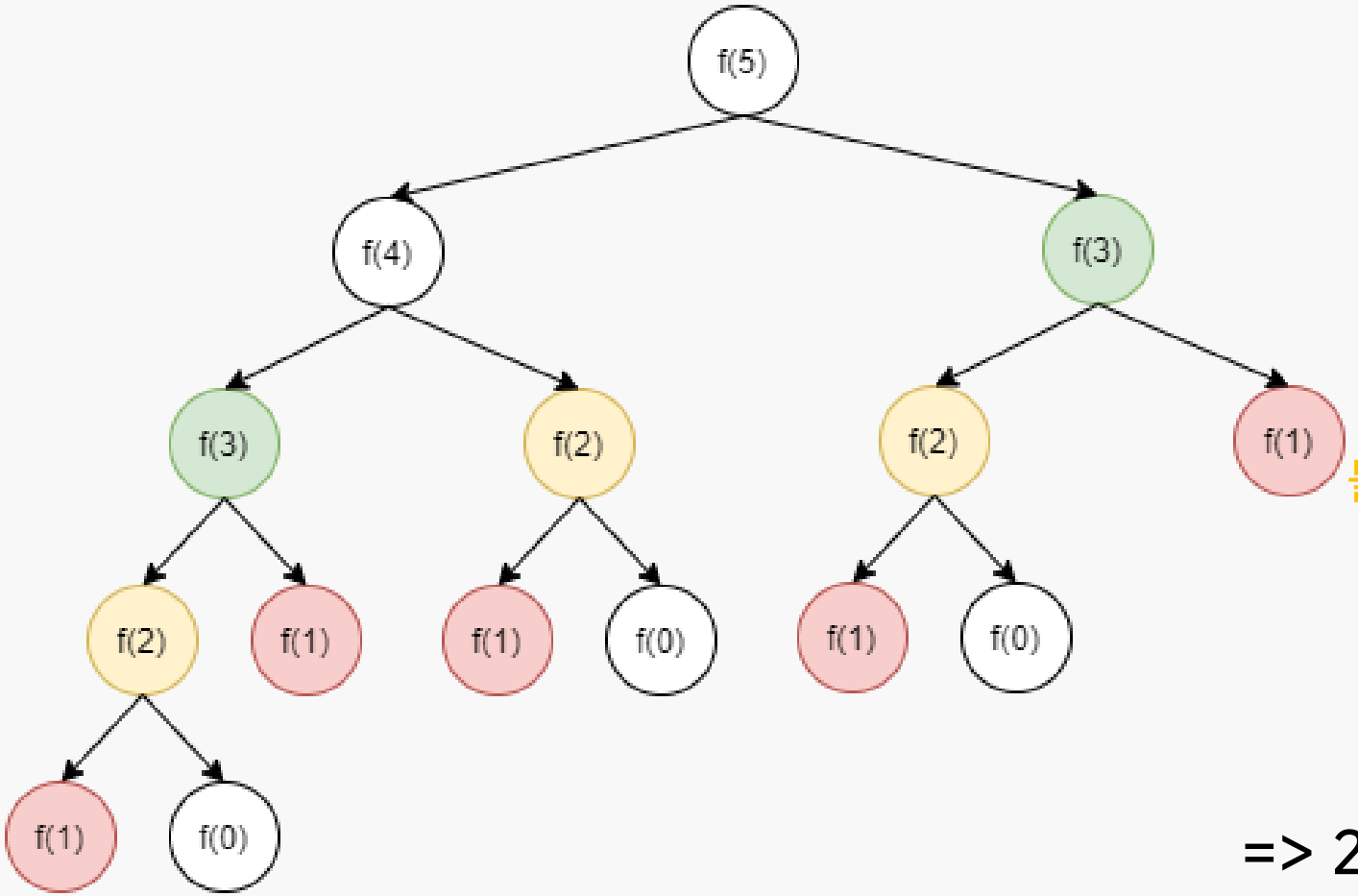
⇒ f(n) = f(n-1) + f(n-2)

중복되는 부분 문제(Overlapping Subproblems)



02 다이나믹 프로그래밍 이란 _ 조건 : 피보나치

피보나치 수열 : 방정식 : f(n) = f(n-1) + f(n-2)



시간 복잡도 $O(2^N)$

기존에 사용하던 재귀 함수

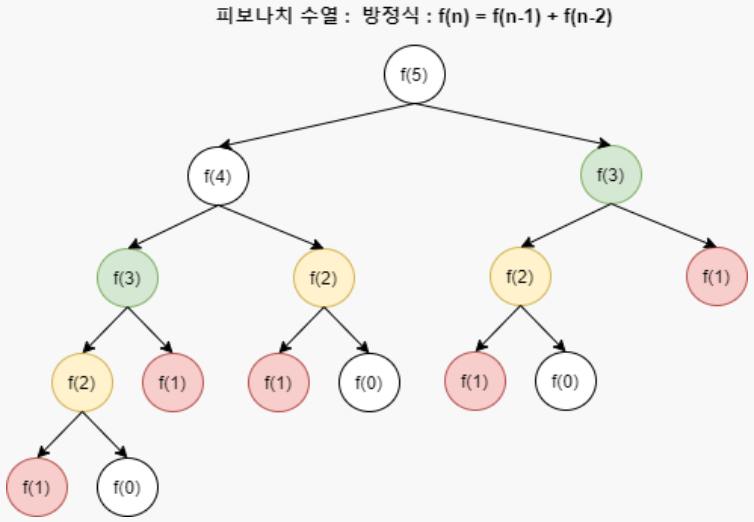
f(5) 만 !
f(5) = f(4) + f(3)
f(4) = f(3) + f(2)
f(3) = f(2) + f(1)
…

불 필요함

f(4) 차례
f(4) = f(3) + f(2)
f(3) = f(2) + f(1)
…

=> 2가지 조건을 만족, DP 사용에 적절 ! !

03 기법 1 : 메모이제이션



f(5) 차례
f(5) = f(4) + f(3)
f(4) = f(3) + f(2)
f(3) = f(2) + f(1)
…

얘를 저장해 놓으면 문제 해결

f(4) 차례
f(4) = f(3) + f(2)
f(3) = f(2) + f(1)
…

한 번 구한 결과를 메모리 공간에 **메모(저장)**해두고 -->
같은 식을 호출하면 메모한 결과를 그대로 가져오는 기법 // 중복 계산 해결

(값을 기록해 놓는다는 점에서 **캐싱(Caching)**이라고도 한다.)

탑다운(하향식) _ 재귀

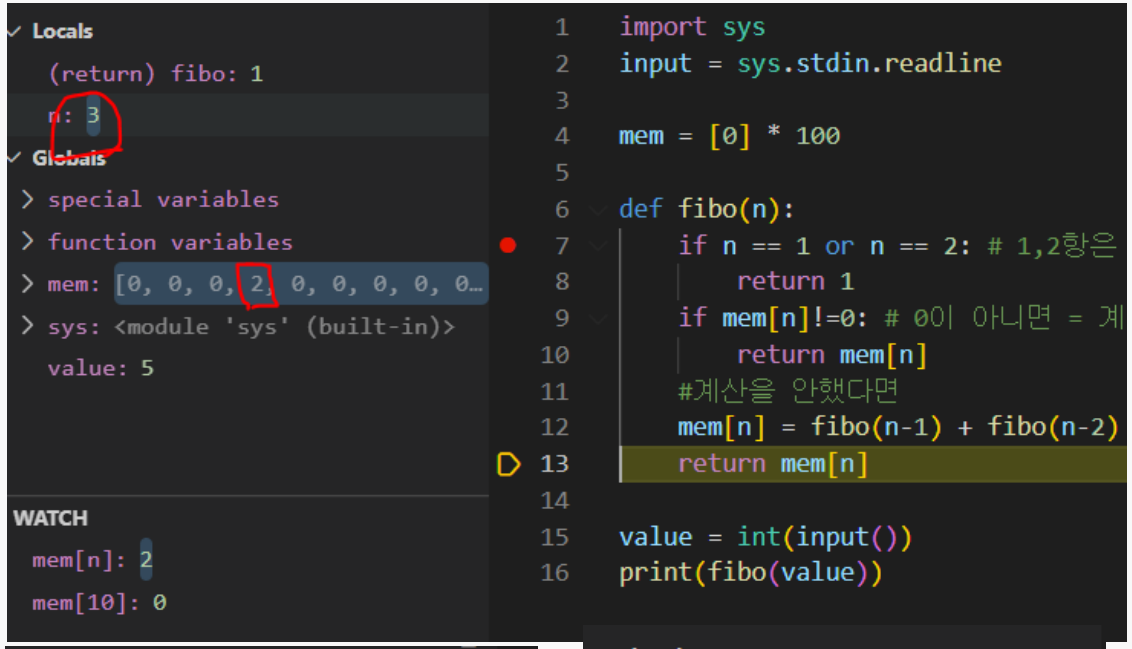
study > DP > fibo.py > fibo

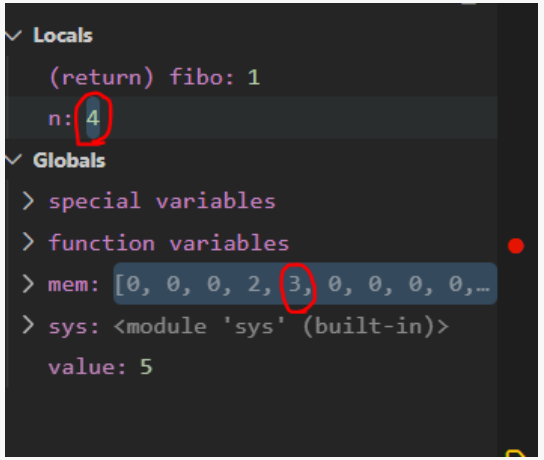
```python
import sys
input = sys.stdin.readline

mem = [0] * 100

def fibo(n):
    if n == 1 or n == 2: # 1,2항은 식에 대입하면 -이므로
        return 1
    if mem[n]!=0: # 0이 아니면 = 계산을 이미 했으면
        return mem[n]
    #계산을 안했다면
    mem[n] = fibo(n-1) + fibo(n-2)
    return mem[n]

value = int(input())
print(fibo(value))
```

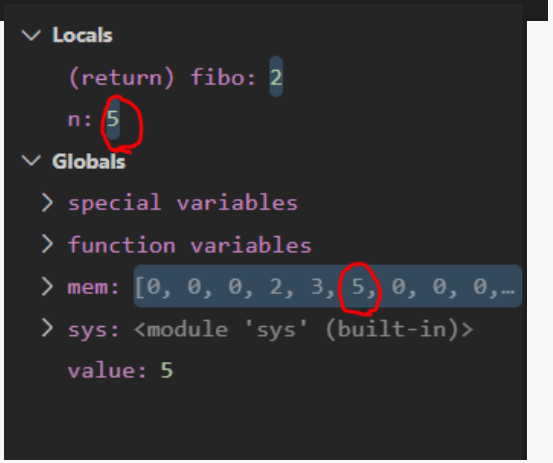
03 기법 1 : 메모이제이션 _ 탑다운 예제 f(5)

study > DP > fibo.py > fibo

```python
import sys
input = sys.stdin.readline

mem = [0] * 100

def fibo(n):
    if n == 1 or n == 2: # 1,2항은 식에 대입하면 -이므로
        return 1
    if mem[n]!=0: # 0이 아니면 = 계산을 이미 했으면
        return mem[n]
    #계산을 안했다면
    mem[n] = fibo(n-1) + fibo(n-2)
    return mem[n]

value = int(input())
print(fibo(value))
```





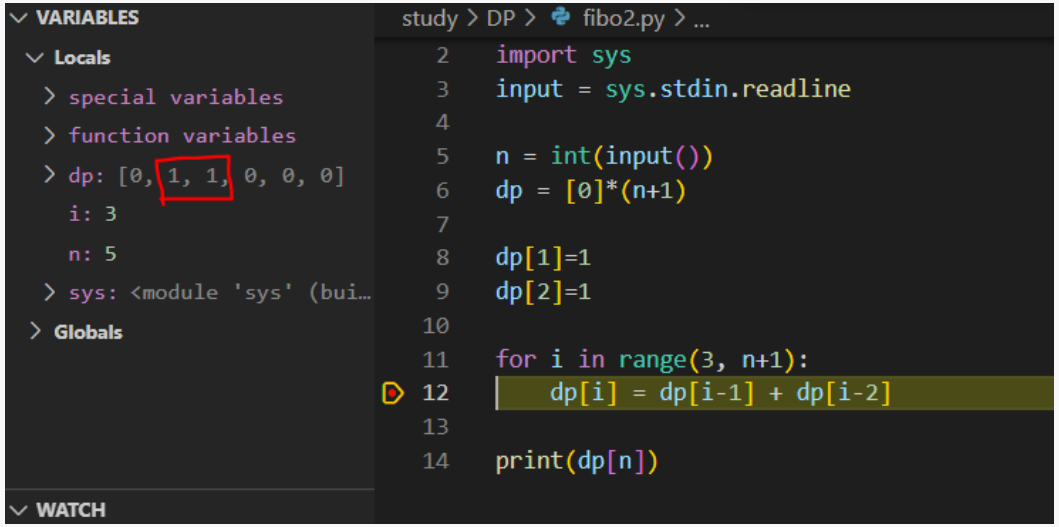


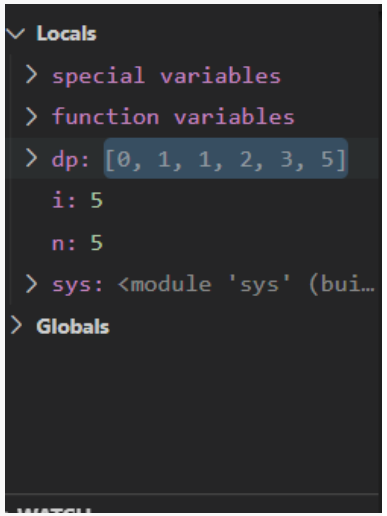
답 : 5

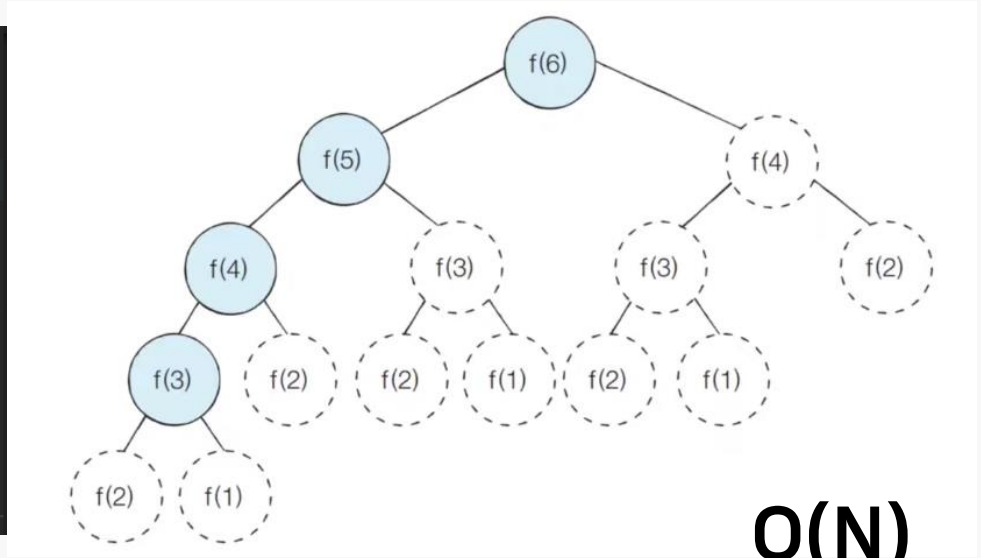
03 기법 2 : 메모이제이션 _ 바텀업

study > DP > fibo2.py > ...

```python
#바텀업
import sys
input = sys.stdin.readline

n = int(input())
dp = [0]*(n+1)

dp[1]=1
dp[2]=1

for i in range(3, n+1):
    dp[i] = dp[i-1] + dp[i-2]

print(dp[n])
```







O(N)

04 예제 : 바닥공사

5401. **바닥 공사 1**

✔ 성공 초

모 모두의 코딩

0 | 953 읽음

가로의 길이가 N, 세로의 길이가 2인 직사각형 형태의 복도가 있습니다. 태혁이는 이 복도의 바닥을 1 X 2의 타일와 2 X 1의 타일을 이용해 채우고자 합니다. 단, 타일을 겹쳐 놓거나 타일을 작게 쪼갤 수 없습니다. 바닥을 타일로 가득 채우는 방법의 수를 출력하는 프로그램을 작성해주세요

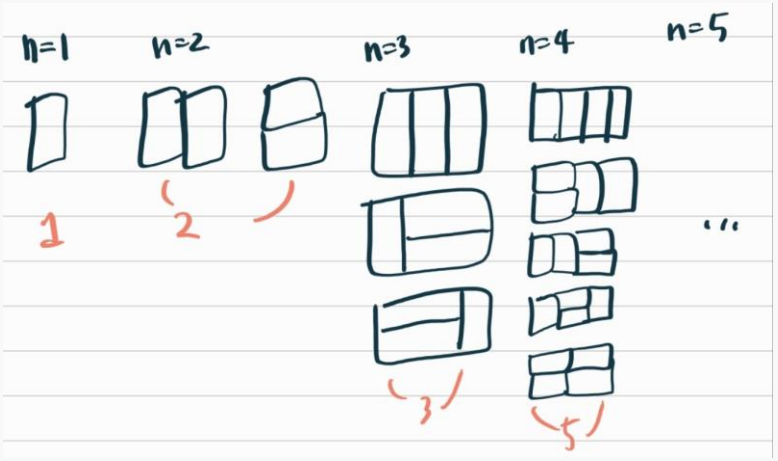
예를 들어 가로의 길이가 3, 세로의 길이가 2인 경우 바닥을 채우는 방법은 총 3가지이므로 3을 출력해야 합니다.

예제 입력1

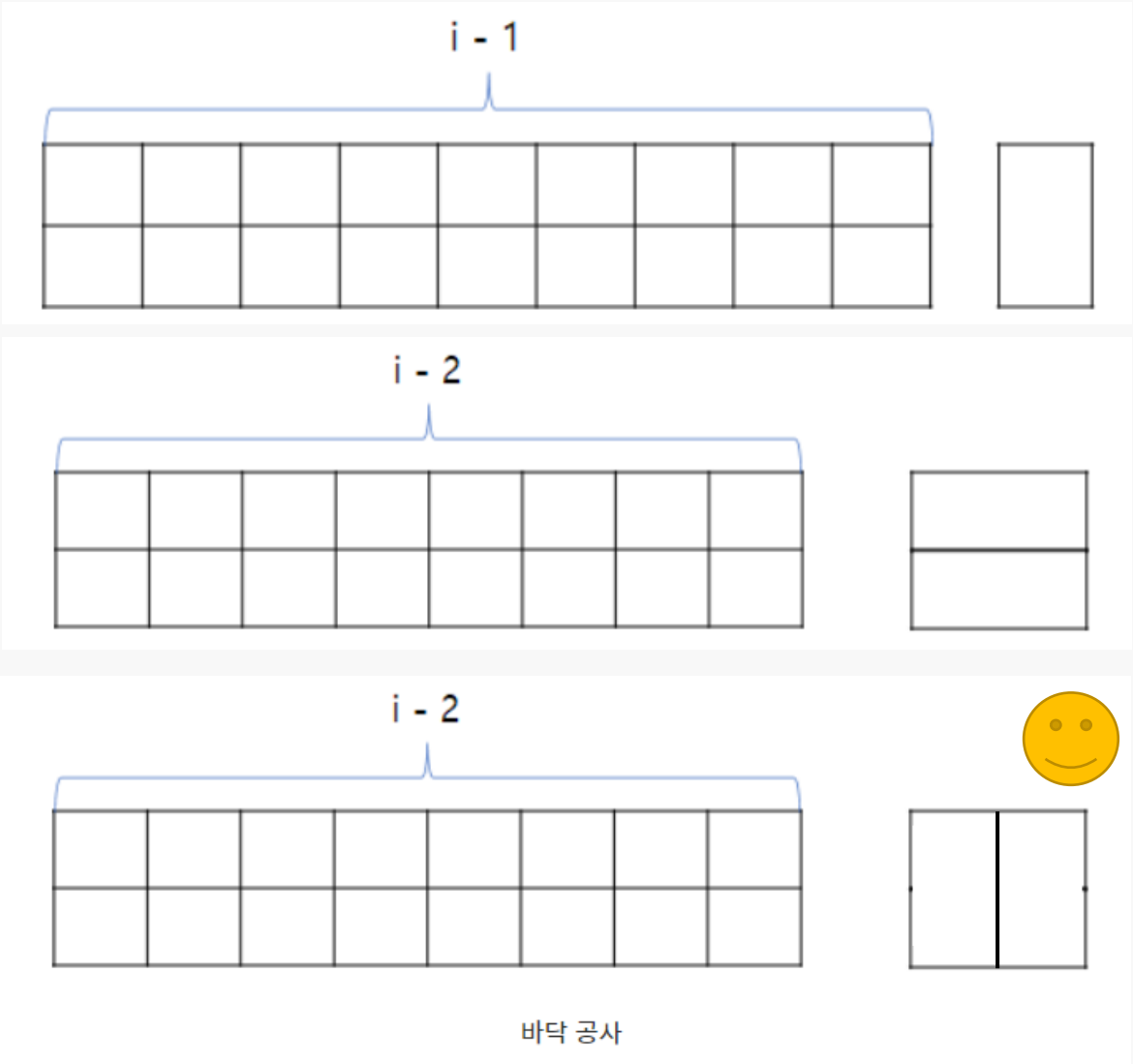
```
3
```

예제 출력1

```
3
```



04 예제 : 바닥공사



$$dp[n] = dp[n-1] + dp[n-2]$$

```python
n = int(input())
dp = [0]*(n+1)

dp[1]=1
dp[2]=1

for i in range(3, n+1):
    dp[i] = dp[i-1] + dp[i-2]

if n == 1:
    print(1)
else :
    print(dp[n])
```

수정 필요

Practice

1+2+3 더하기) https://www.acmicpc.net/problem/9095

계단 오르기) https://www.acmicpc.net/problem/2579

바닥 타일 3) 코딩마스터즈

퇴사 2) https://www.acmicpc.net/problem/15486

THANK YOU –

감사합니다 ^_^