DP 다이나믹 프로그래밍
(동적 계획법 알고리즘)

알고리즘 스터디 4주차 강수지

DP ?

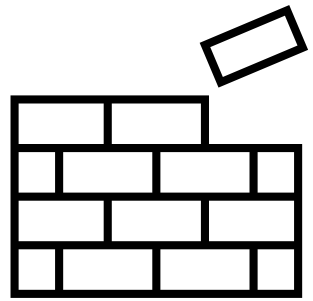
Dynamic Programming

복잡한 문제를 간단한 여러 개의 문제로 나누어 푸는 방법

이런 문제에 활용해요 !

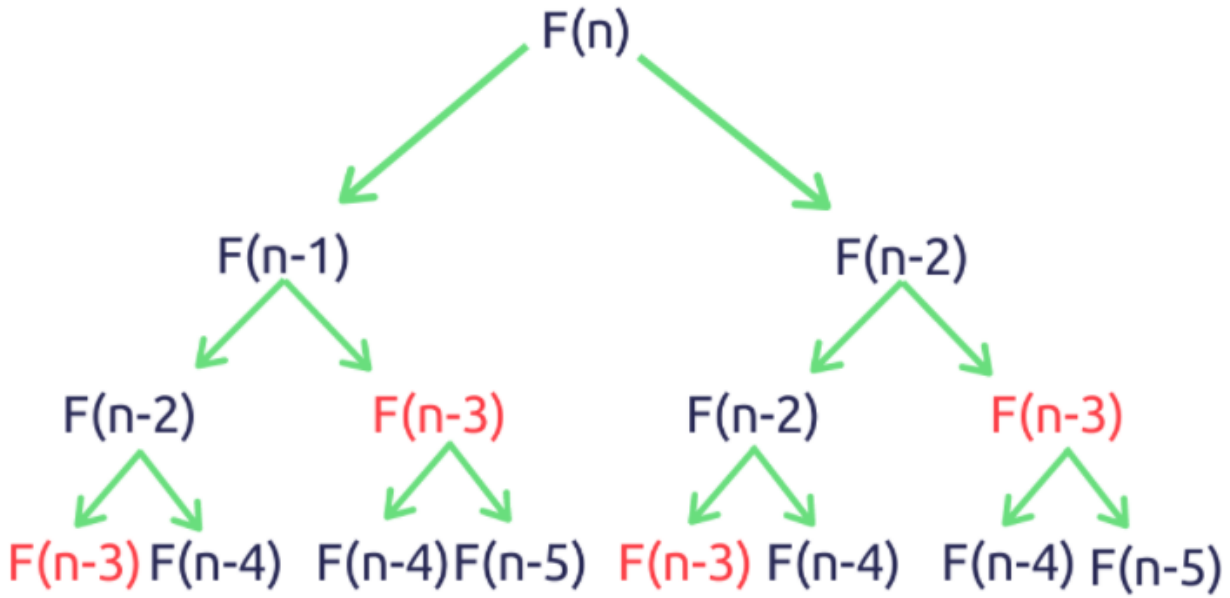
최적성의 원리를 만족하는지 판단

Ex1) 큰 문제를 작은 문제로 나눌 수 있다



Ex2) 작은 문제에서 구한 정답이
그것을 포함하는 큰 문제에서도 동일하다

피보나치 수열로 이해하기



소문제가 상위문제를 해결하기 위해 사용되는 대표적인 예시

중복 연산을 방지하기 위한 방법 (1)

Memoization (하향식)

하위 문제에 대해
정답을 계산했는지 확인

```python
# DP
# memoization (하향식)

dp = [0] * 100  # 소문제 결과를 저장할 리스트
dp[0] = 1
dp[1] = 1


def fib(n):
    # 만약 계산한 적이 없다면 재귀로 계산
    if dp[n] == 0:
        dp[n] = fib(n - 1) + fib(n - 2)

    # 있다면 그대로 반환
    return dp[n]
```

Memoization 을 사용하여 F(6) 을 구해보자 !

```
[1, 1, 2, 3, 5, 8, 13, 0, 0, 0, 0,
13
```

소문제 결과가 담긴 리스트를 확인해가며 해결 !

중복 연산을 방지하기 위한 방법 (2)

Tabulation (상향식)

작은 문제의 정답을
이용해서 큰 문제의 정답을 풀이

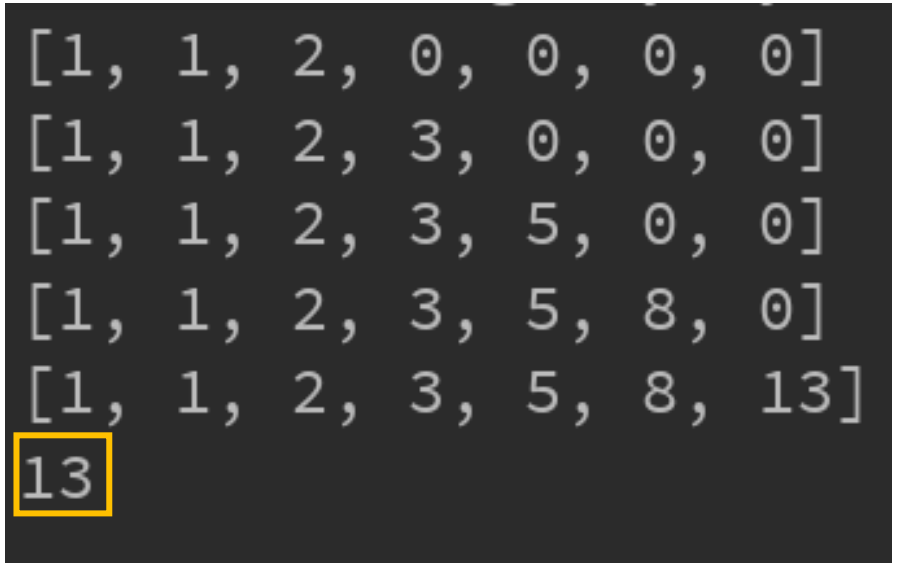
```python
# DP
# 타뷸레이션 (상향식)

def fib(n):
    dp = [0] * (n + 1)
    dp[0] = 1
    dp[1] = 1

    # 작은 값(소문제)부터 직접 계산하며 진행
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]

    return dp[n]
```

Tabulation 을 사용하여 F(6) 을 구해보자 !



리스트 값을 채워 나가며
최종 값을 반환 !

DP 속도 비교 !

결과 : 0.0002186 vs 0.017931199999999998

```python
# 메모장!
factorial_memo = {}

# memoization을 이용한 팩토리얼
def factorial_with_memo(n):
    if n<2:
        return 1
    if n not in factorial_memo:
        factorial_memo[n] = n * factorial_with_memo(n - 1)
    return factorial_memo[n]

# 그냥 팩토리얼
def factorial(n):
    if n < 2:
        return 1
    return n * factorial(n - 1)


####################팩토리얼 100!을 1000번 반복해보기####################
from timeit import *
t1 = Timer("factorial_with_memo(100)", "from __main__ import factorial_with_memo").timeit(number=1000)
t2 = Timer("factorial(100)", "from __main__ import factorial").timeit(number=1000)

print(t1, "vs", t2)
```

Climbing Stairs – 예시 문제로 이해하기

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

1계단 = 1

2계단 = 1+1, 2

3계단 = 1+1+1, 1+2, 2+1

.

.

.

이 문제의 점화식은 ?

Climbing Stairs – Tabulation으로 해결 !

```python
def climbStairs(n: int) -> int:
    c = [0]*(n+1)
    c[0] = 1
    c[1] = 2

    if n < 3:
        return c[n-1]

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

    return c[n-1]
```

n > 2 일 때, C(n) > C(n-1) + c(n-2)

```
[1, 2, 0, 0]
[1, 2, 3, 0]
3
```

climbStairs(3)

c[2] = c[1] + c[0]

다이나믹 프로그래밍 VS 분할정복

- 분할 정복 (Divide and conquer)이란 ?
  하나의 문제를 여러 개의 부분 문제로 나누어 해결하는 방식

다이나믹 프로그래밍과 분할 정복의 차이점

부분 문제의 중복

다이나믹 프로그래밍은 각 부분 문제들이 서로 영향을 미치며 중복
하지만, 분할 정복은 동일한 부분문제가 반복적으로 계산되진 않는다 !