동적 계획법 - Dynamic Programming

동적 계획법 - Dynamic Programming

동적 계획법이란?

- 문제를 해결하는 <u>최적의 해</u>(*Optimal Solution*)를 구하는 알고리즘
- <u>최적 부분 구조</u>(*Optimal Substructure*)를 갖는 경우에
- <u>중복되는 부분 문제</u>(*Subproblem*)를 <u>저장하여 재사용</u>(*Memoization*)하여 <u>재귀를 최적화</u>
- 계획(*Programming*)이란 해를 구하기 위해 테이블을 구축한다는 것을 의미
- 동적 계획 대신 기억하기 프로그래밍이라고 부르기도 한다.

동적 계획법 - Dynamic Programming

동적 계획법이란?

최적의 해(*Optimal Solution*)

최적 부분 구조(*Optimal Substructure*)

중복되는 부분 문제(*Subproblem*) 저장하여 재사용(*Memoization*) 재귀를 최적화

동적 계획법 - Dynamic Programming

동적 계획법이란?

최적의 해(*Optimal Solution*)

최적 부분 구조(*Optimal Substructure*)

중복되는 부분 문제(*Subproblem*) 저장하여 재사용(*Memoization*) 재귀를 최적화

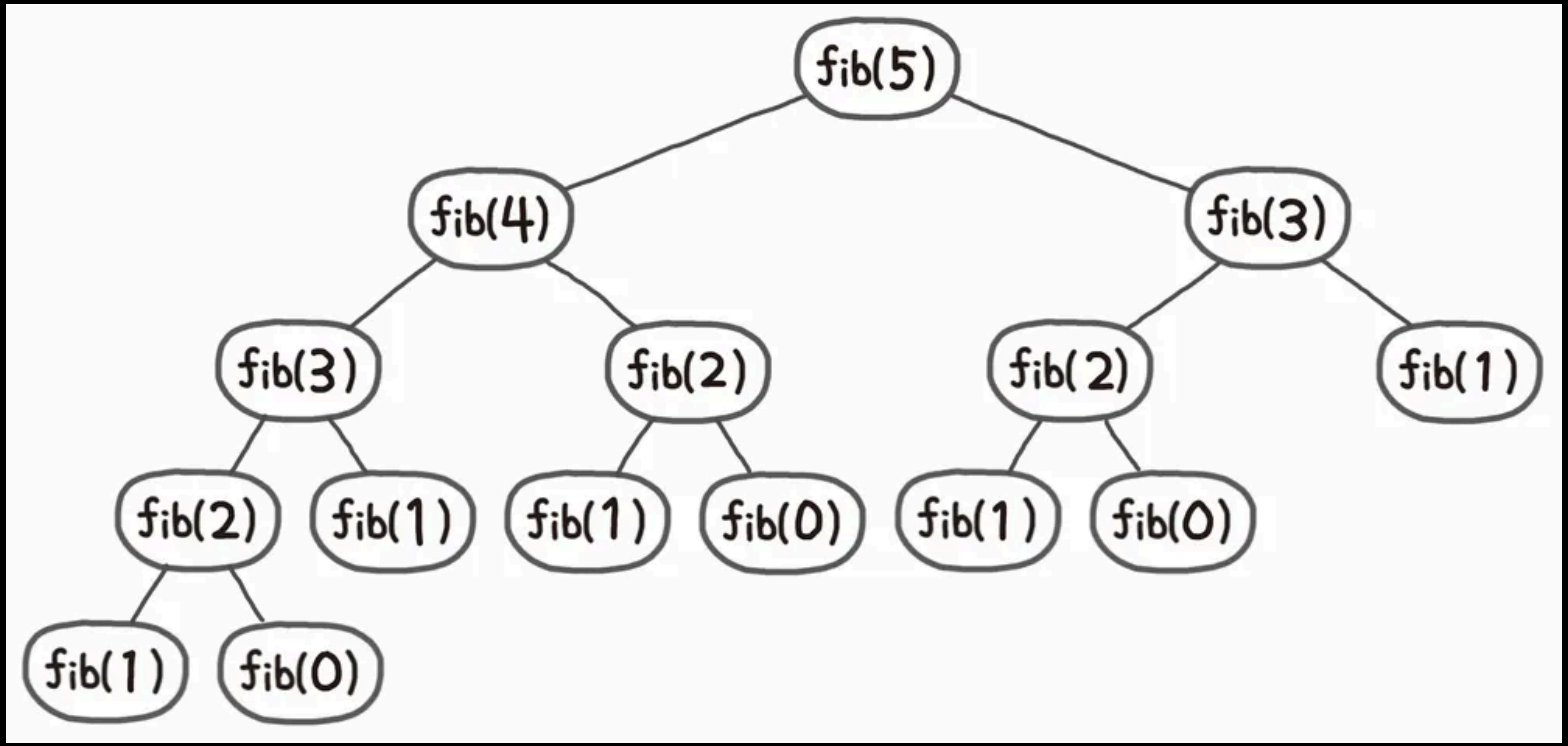
재귀 최적화 최적의 해

동적 계획법 - Dynamic Programming

재귀 최적화 - Fibonacci

```
int fibonacci(int n) {
    if (n <= 1)
        return (n);
    return (fibonacci( n: n - 1) + fibonacci( n: n - 2));
}
```

$$f(n) = f(n-1) + f(n-2)$$

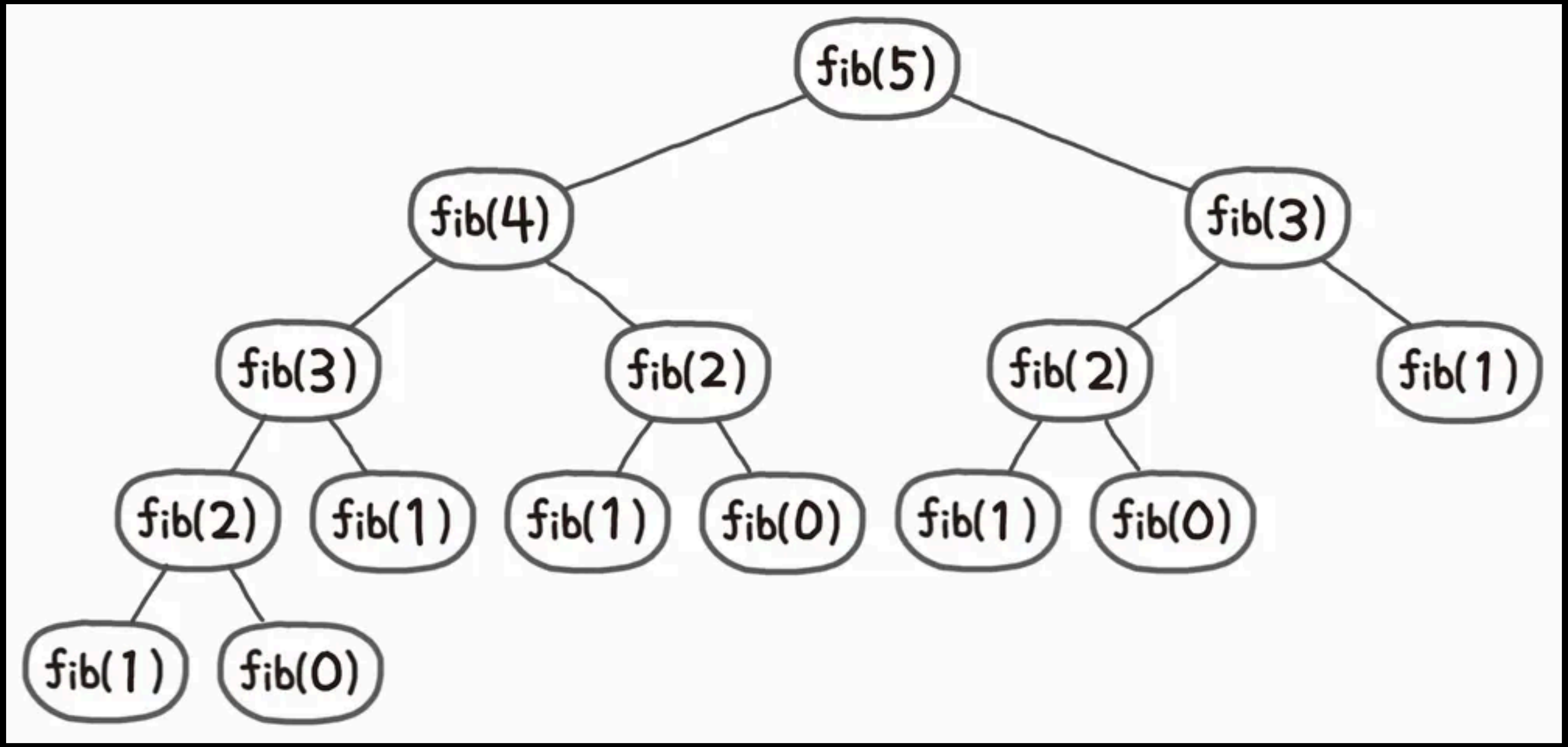


동적 계획법 - Dynamic Programming

재귀 최적화 - Fibonacci

```
int fibonacci(int n) {
    if (n <= 1)
        return (n);
    return (fibonacci( n: n - 1) + fibonacci( n: n - 2));
}
```

$$f(n) = f(n-1) + f(n-2)$$



최적 부분 구조

중복되는 부분 문제

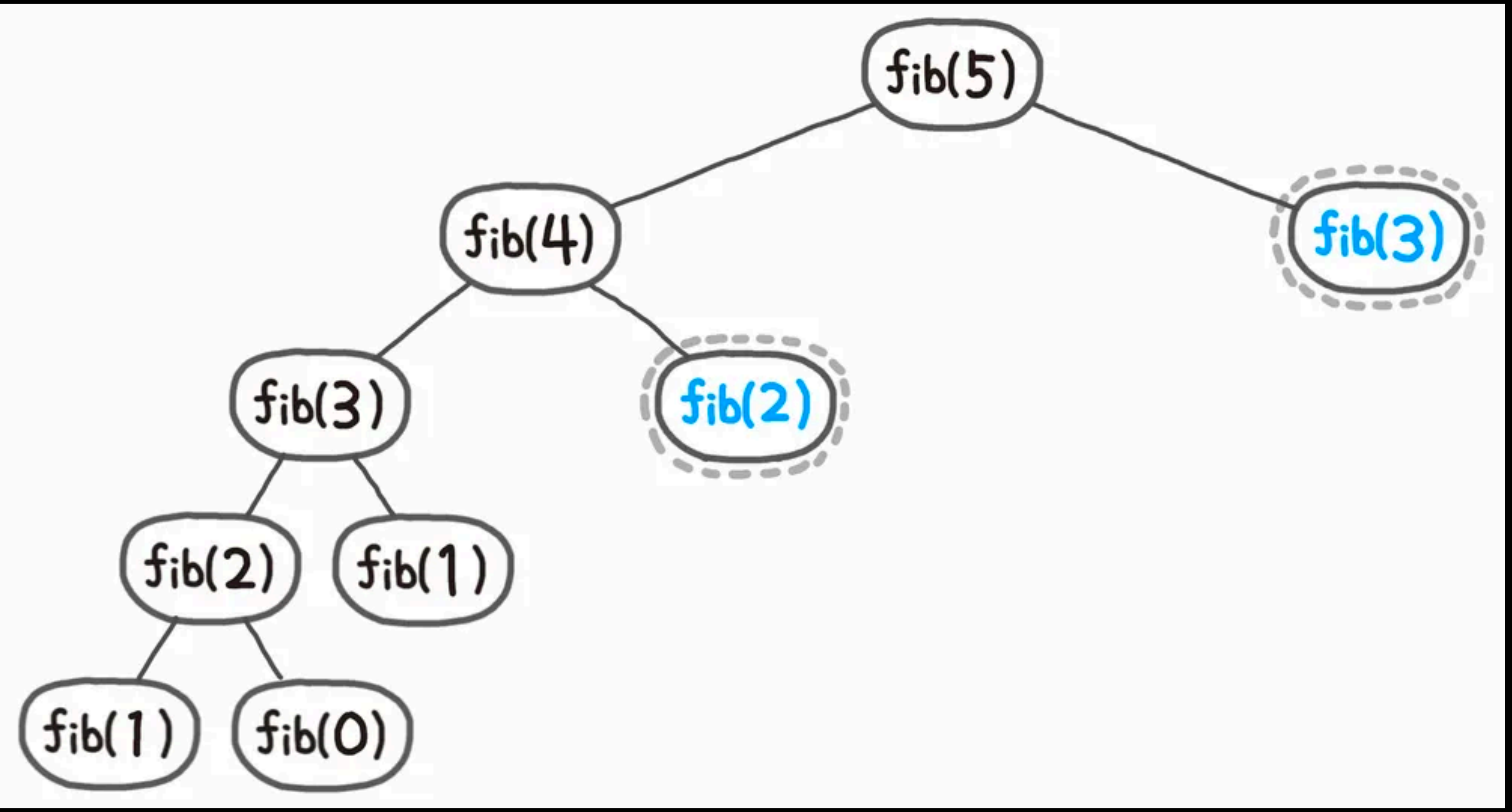
동적 계획법 - Dynamic Programming

재귀 최적화 - Fibonacci

```
int dp[1000001];

int fibonacci(int n) {
    if (dp[n])
        return (dp[n]);
    if (n <= 1)
        return (n);
    dp[n] = fibonacci( n: n - 1) + fibonacci( n: n - 2);
    return (dp[n]);
}
```

중복되는 부분을 저장하여 재사용



동적 계획법 - Dynamic Programming

재귀 최적화 - Fibonacci

```c
int dp[1000001];

int fibonacci(int n) {
    if (dp[n])
        return (dp[n]);
    if (n <= 1)
        return (n);
    dp[n] = fibonacci(n - 1) + fibonacci(n - 2);
    return (dp[n]);
}
```

메모이제이션(Memoization)

```c
int fib[1000001];

int fibonacci(int n) {
    fib[0] = 0;
    fib[1] = 1;
    for (int i = 2; i <= n; i++)
        fib[i] = fib[i - 1] + fib[i - 2];
    return (fib[n]);
}
```

타뷸레이션(Tabulation)

동적 계획법 - Dynamic Programming

재귀 최적화 - Fibonacci

```
int dp[1000001];

int fibonacci(int n) {
    if (dp[n])
        return (dp[n]);
    if (n <= 1)
        return (n);
    dp[n] = fibonacci( n: n - 1) + fibonacci( n: n - 2);
    return (dp[n]);
}
```

메모이제이션(Memoization)

하향식 접근(Top-Down Approach)
재귀(Recursive)
Subproblem 일부만 계산해도 되는 경우

```
int fib[1000001];

int fibonacci(int n) {
    fib[0] = 0;
    fib[1] = 1;
    for (int i = 2; i <= n; i++)
        fib[i] = fib[i - 1] + fib[i - 2];
    return (fib[n]);
}
```

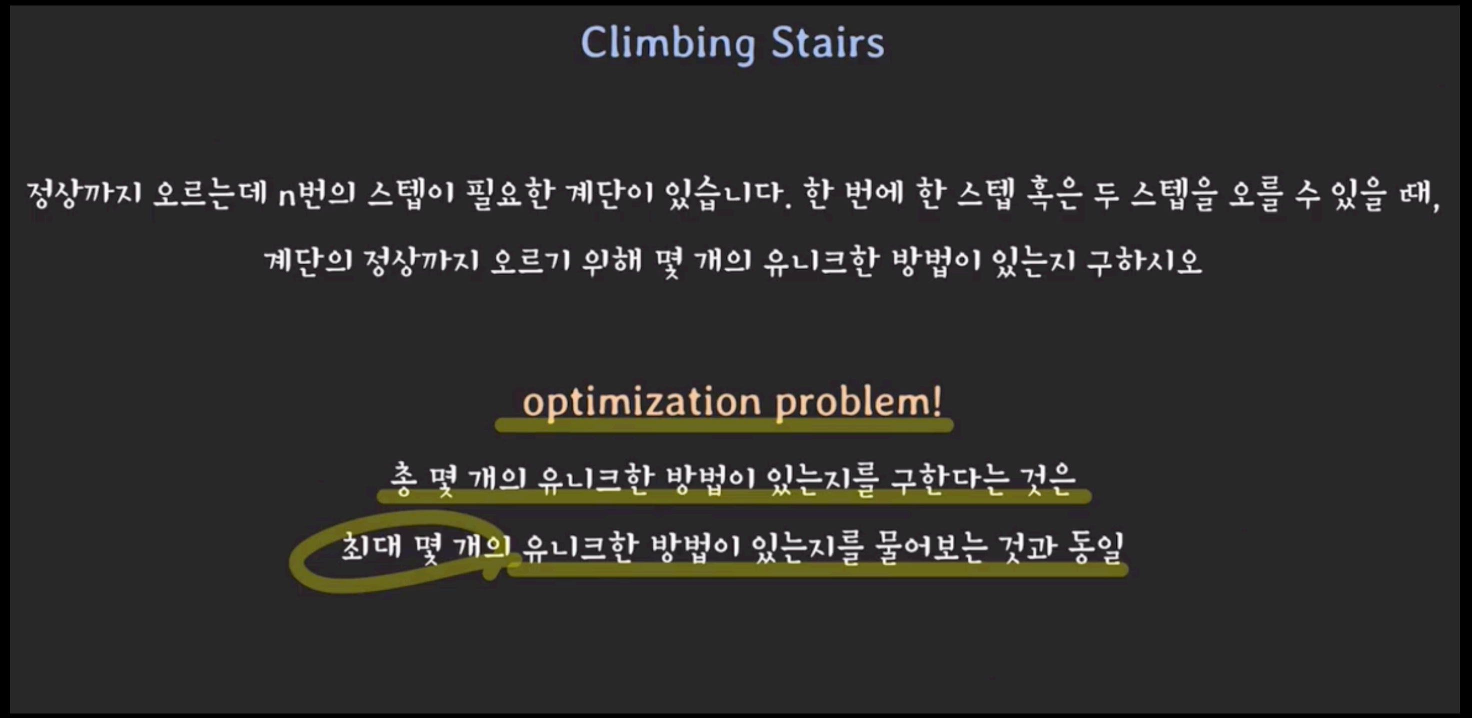
타뷸레이션(Tabulation)

상향식 접근(Bottom-Up Approach)
반복(Iteration)
모든 Subproblem을 계산해야 하는 경우

동적 계획법 - Dynamic Programming

최적의 해 - Climbing Stairs

- Climbing Stairs
- 정상까지 오르는데 n번의 스텝이 필요한 계단이 있다.
- 한 번에 1 스텝 혹은 2 스텝을 오를 수 있을 때,
- 계단의 정상까지 오르기 위한 방법의 총 개수를 구하시오.

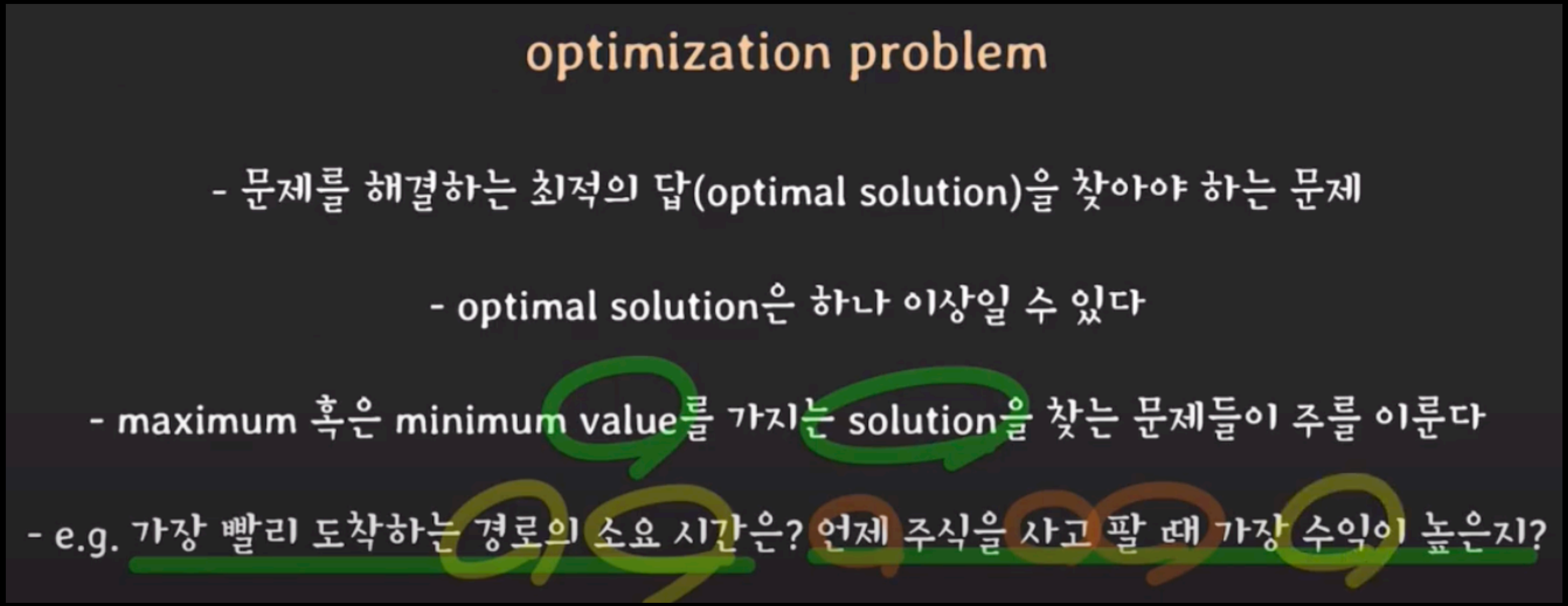


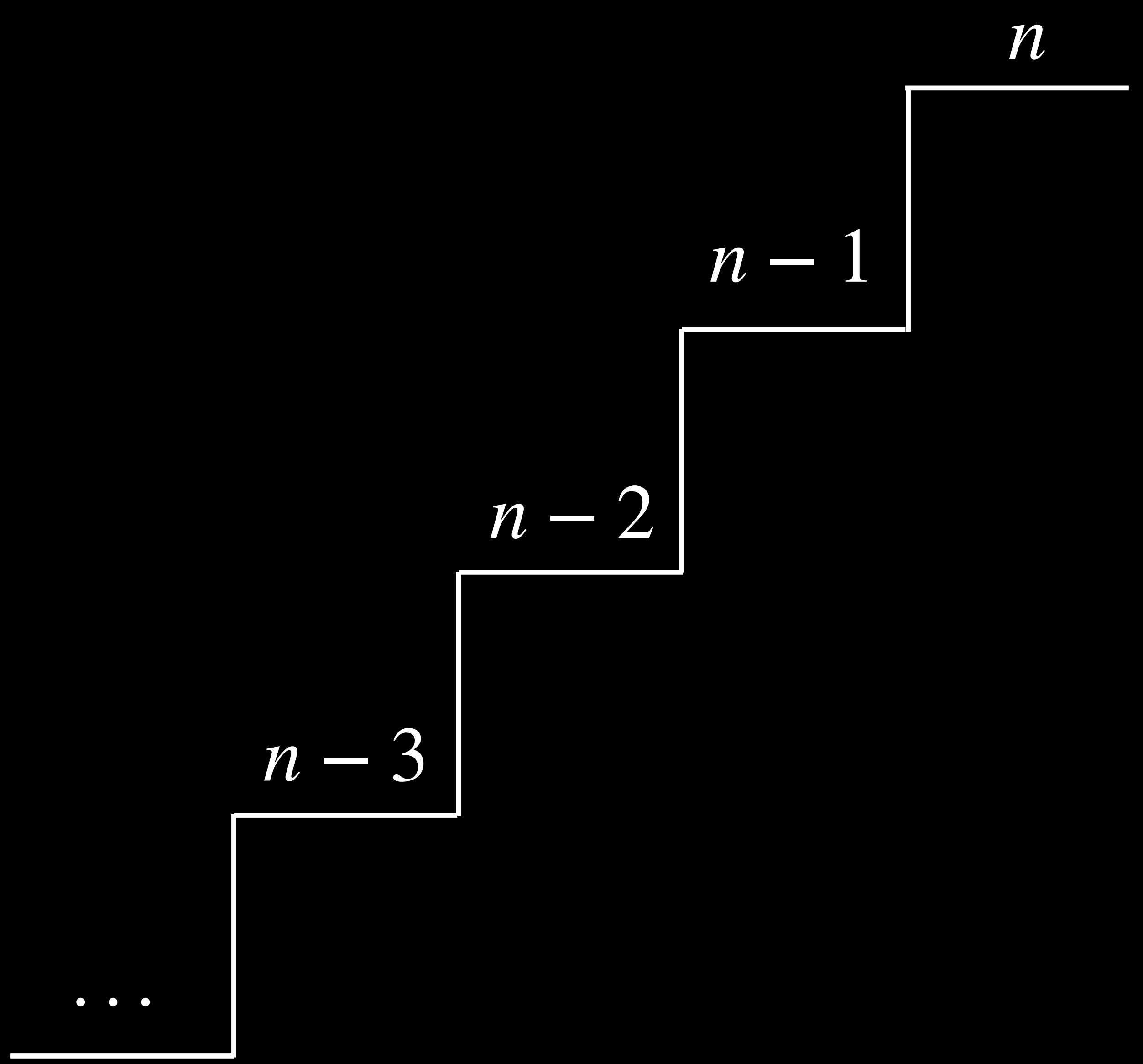
출처 : https://www.youtube.com/watch?v=GtqHli8HIqk

동적 계획법 - Dynamic Programming

최적의 해 - Climbing Stairs

정상까지 오르는데 n번의 스텝이 필요한 계단이 있다. 한 번에 1 스텝 혹은 2 스텝을 오를 수 있을 때,
계단의 정상까지 오르기 위한 방법의 총 개수를 구하시오.



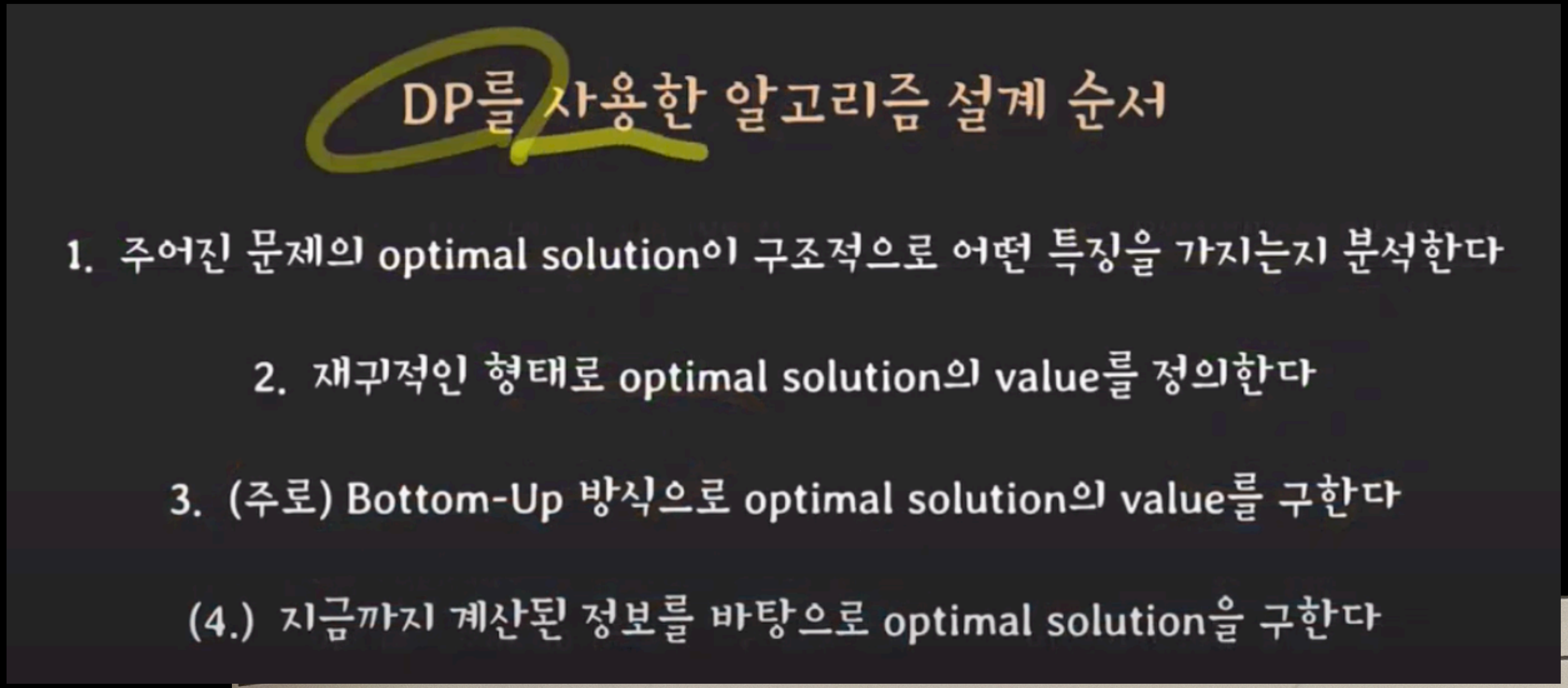


출처 : https://www.youtube.com/watch?v=GtqHIi8HIqk

동적 계획법 - Dynamic Programming

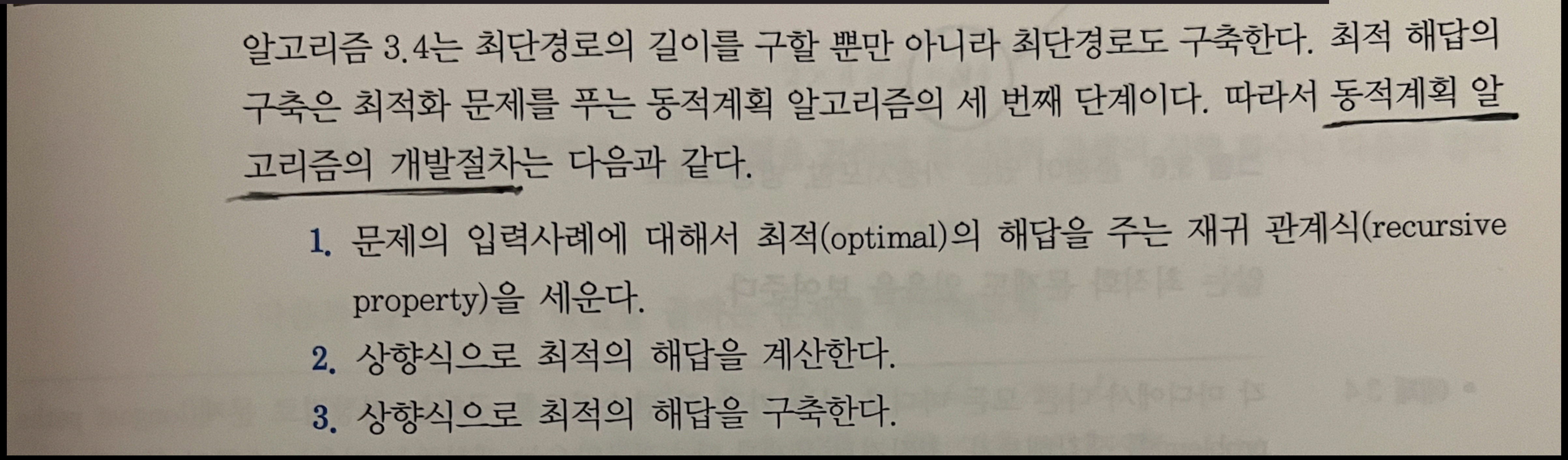
최적의 해 - Climbing Stairs

정상까지 오르는데 n번의 스텝이 필요한 계단이 있다. 한 번에 1 스텝 혹은 2 스텝을 오를 수 있을 때,
계단의 정상까지 오르기 위한 방법의 총 개수를 구하시오.



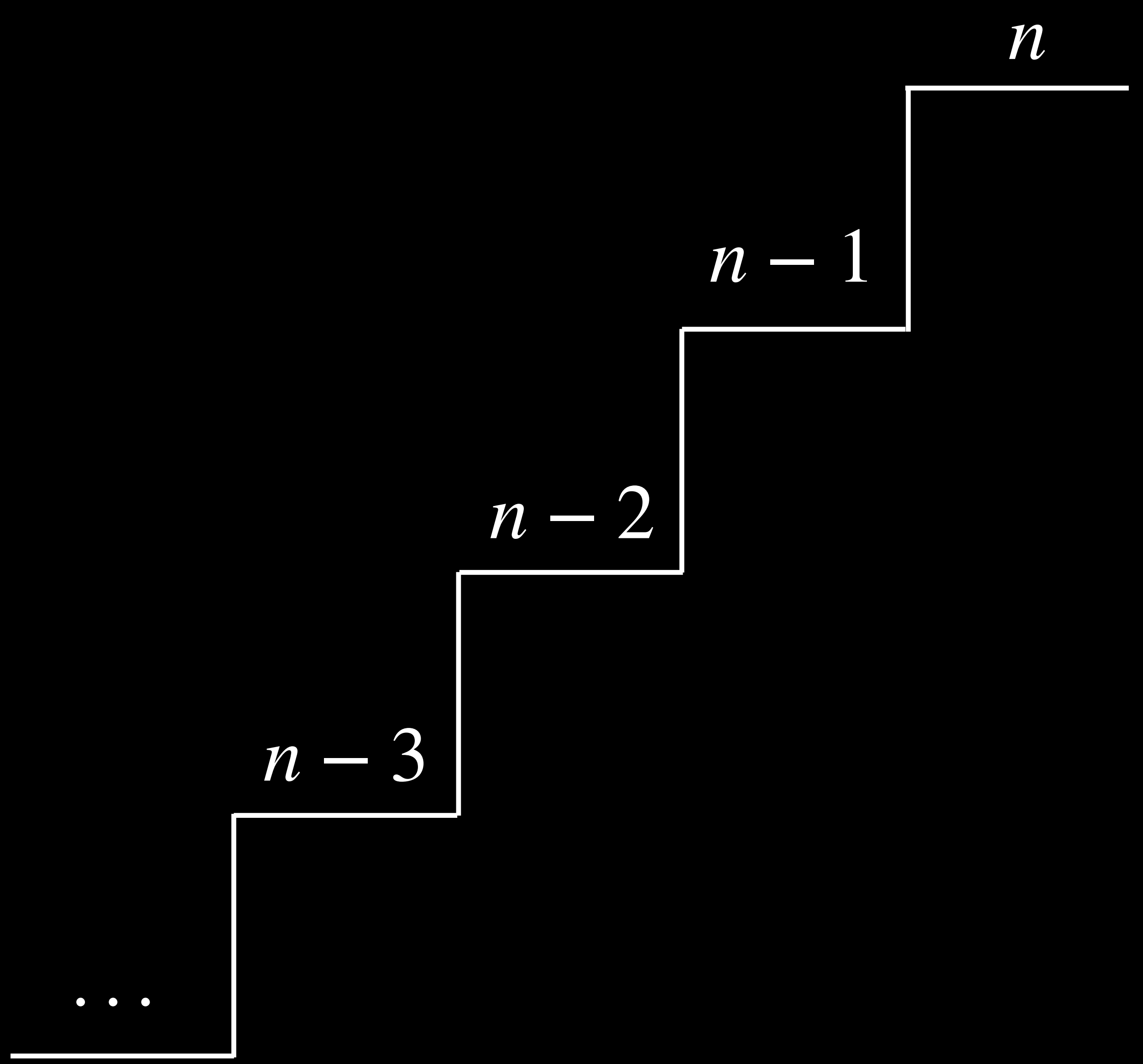
DP를 사용한 알고리즘 설계 순서

1. 주어진 문제의 optimal solution이 구조적으로 어떤 특징을 가지는지 분석한다
2. 재귀적인 형태로 optimal solution의 value를 정의한다
3. (주로) Bottom-Up 방식으로 optimal solution의 value를 구한다
(4.) 지금까지 계산된 정보를 바탕으로 optimal solution을 구한다



알고리즘 3.4는 최단경로의 길이를 구할 뿐만 아니라 최단경로도 구축한다. 최적 해답의 구축은 최적화 문제를 푸는 동적계획 알고리즘의 세 번째 단계이다. 따라서 동적계획 알고리즘의 개발절차는 다음과 같다.

1. 문제의 입력사례에 대해서 최적(optimal)의 해답을 주는 재귀 관계식(recursive property)을 세운다.
2. 상향식으로 최적의 해답을 계산한다.
3. 상향식으로 최적의 해답을 구축한다.



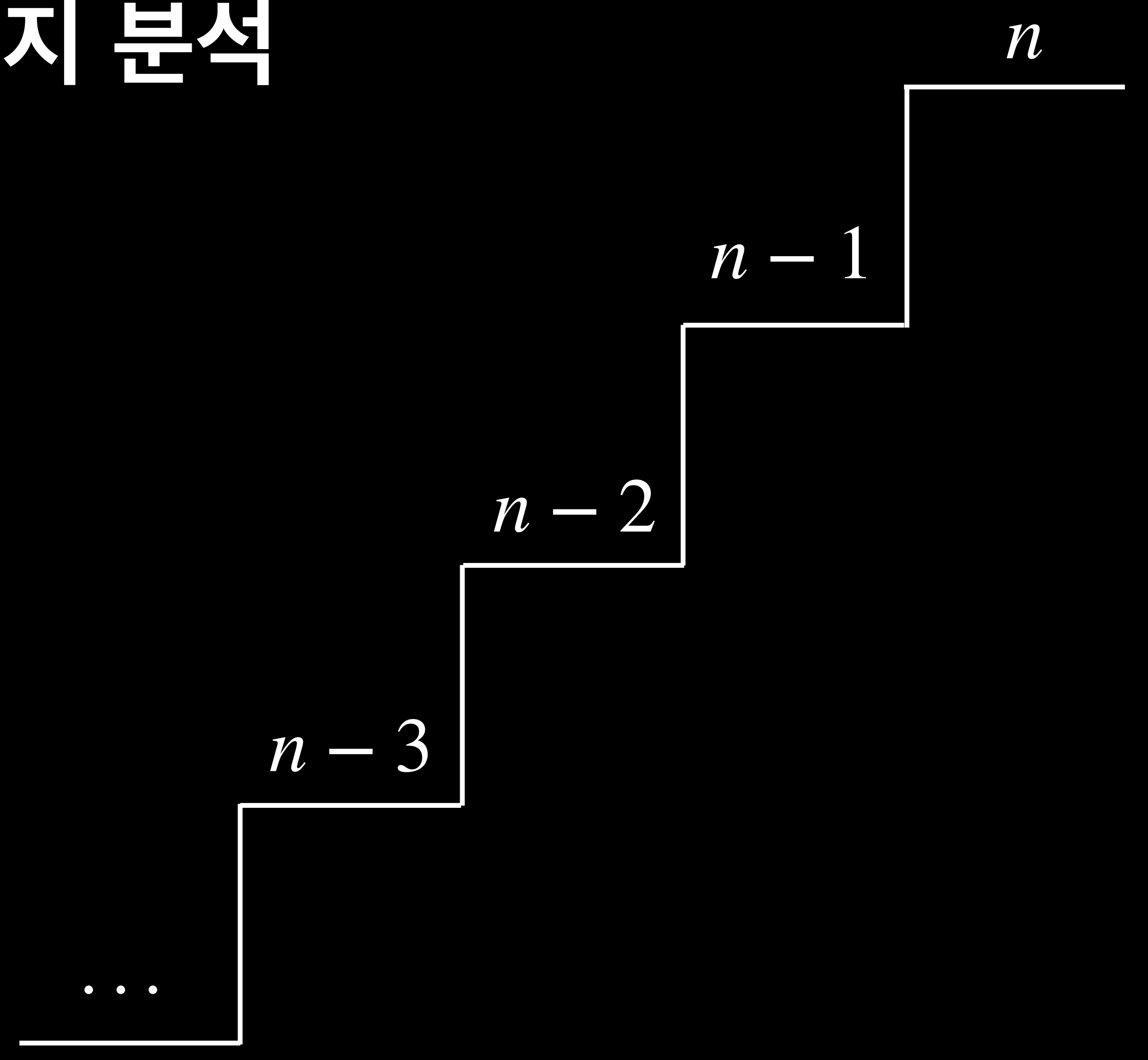
출처 : https://www.youtube.com/watch?v=GtqHIi8HIqk

동적 계획법 - Dynamic Programming

최적의 해 - Climbing Stairs

정상까지 오르는데 n번의 스텝이 필요한 계단이 있다. 한 번에 1 스텝 혹은 2 스텝을 오를 수 있을 때,
계단의 정상까지 오르기 위한 방법의 총 개수를 구하시오.

1. 주어진 문제의 Optimal Solution이 구조적으로 어떤 특징을 가지고 있는지 분석



총 개수 = 최대 개수

출처 : https://www.youtube.com/watch?v=GtqHli8HIqk

동적 계획법 - Dynamic Programming

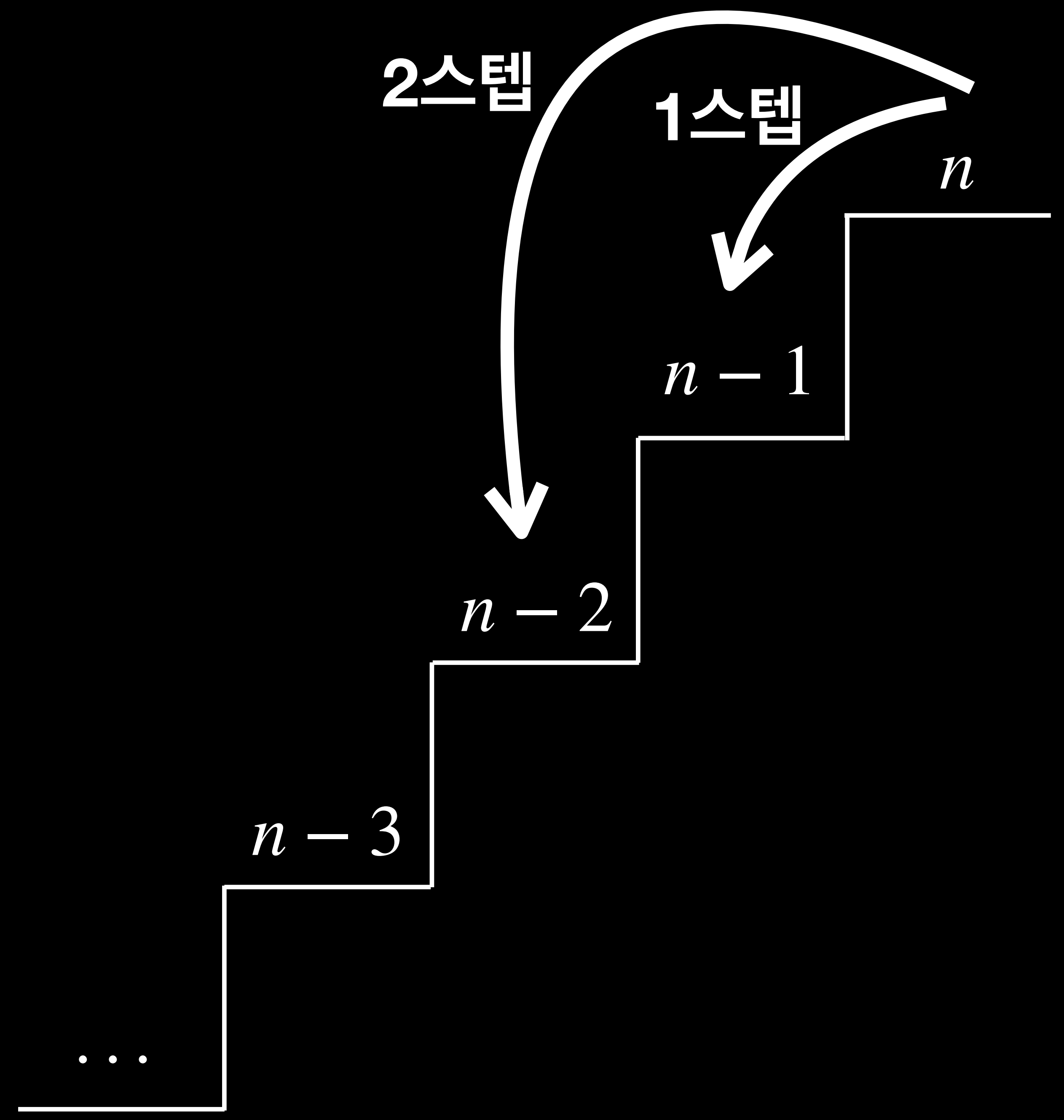
최적의 해 - Climbing Stairs

정상까지 오르는데 n번의 스텝이 필요한 계단이 있다. 한 번에 1 스텝 혹은 2 스텝을 오를 수 있을 때, 계단의 정상까지 오르기 위한 방법의 총 개수를 구하시오.

2. 재귀적인 형태로 Optimal Solution의 Value 정의
→ Value에 대한 재귀 관계식 정의

$$f(n) = f(n-1) + f(n-2)$$

```
int climb(int n) {
    if (n <= 2)
        return (n);
    return (climb(n: n - 1) + climb(n: n - 2));
}
```



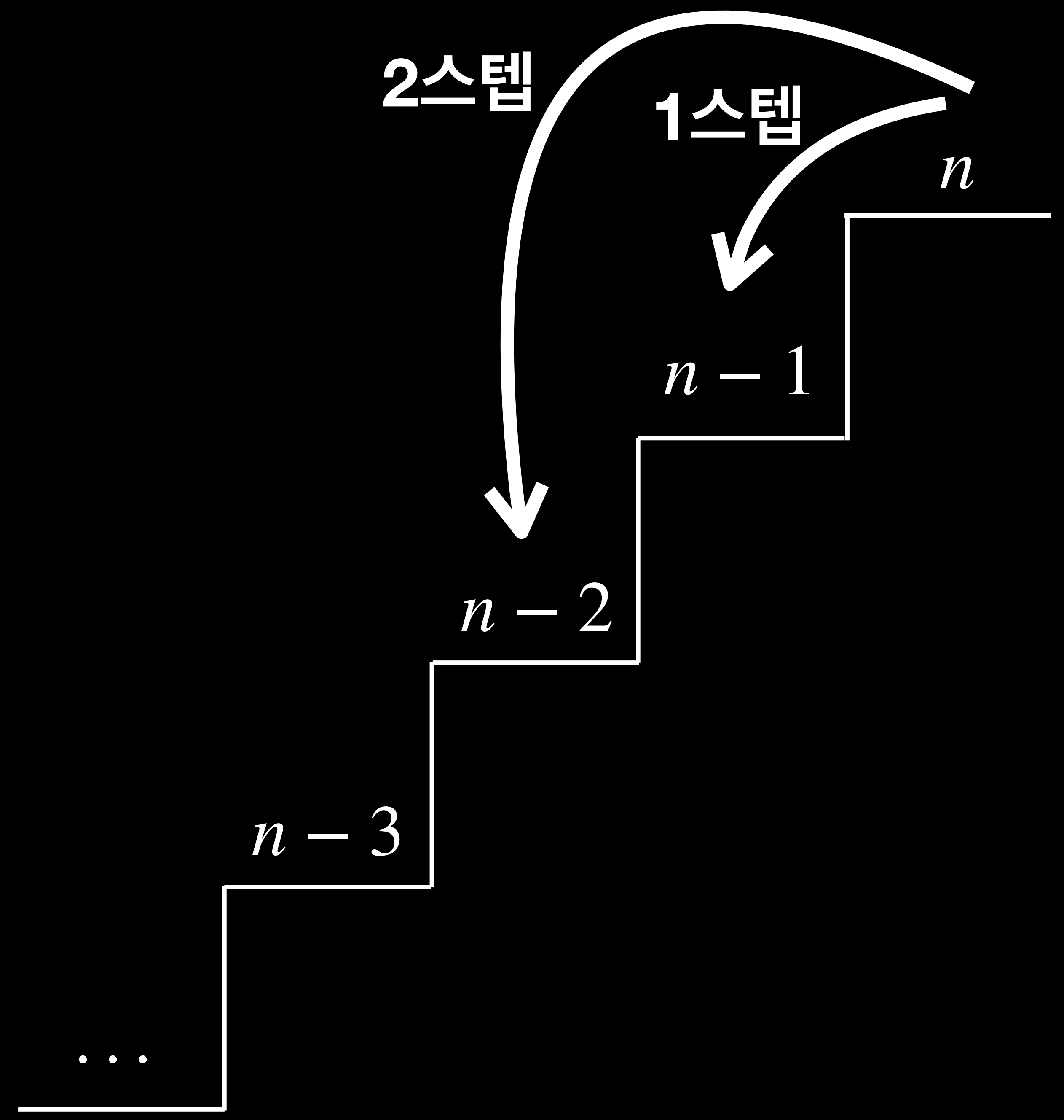
출처 : https://www.youtube.com/watch?v=GtqHli8HIqk

동적 계획법 - Dynamic Programming

최적의 해 - Climbing Stairs

정상까지 오르는데 n번의 스텝이 필요한 계단이 있다. 한 번에 1 스텝 혹은 2 스텝을 오를 수 있을 때,
계단의 정상까지 오르기 위한 방법의 총 개수를 구하시오.

3. Bottom-Up 방식으로 Optimal Solution의 Value 계산
→ 재귀 함수를 반복문 형태로 변환




```
int dp[1000001];

int climb(int n) {
    if (dp[n])
        return (dp[n]);
    if (n <= 2)
        return (n);
    dp[n] = climb( n: n - 1) + climb( n: n - 2);
    return (dp[n]);
}
```

→

```
int dp[1000001];

int climb(int n) {
    dp[1] = 1;
    dp[2] = 2;
    for (int i = 3; i <= n; ++i)
        dp[i] = dp[i - 1] + dp[i - 2];
    return (dp[n]);
}
```

출처 : https://www.youtube.com/watch?v=GtqHli8HIqk

동적 계획법 - Dynamic Programming
정리

- **조건**
  - 최적 부분 구조(Optimal Substructure)
  - 중복되는 부분 문제(Overlapping Subproblem)

- **방식**
  - 메모이제이션(Memoization) : Top-Down Approach
  - 타뷸레이션(Tabulation) : Bottom-Up Approach

- **설계**
  - 주어진 문제의 Optimal Solution이 구조적으로 어떤 특징을 가지고 있는지 분석
  - 재귀적인 형태로 Optimal Solution의 Value 정의 → Value에 대한 재귀 관계식 정의
  - Bottom-Up 방식으로 Optimal Solution의 Value 계산 → 재귀 함수를 반복문 형태로 변환