# 알고리즘 특강 동적계획법

메모이제이션을 활용한 문제 풀이 기법입니다.
굉장히 생소한 개념이며, 초보자분들은 많이 어려워 하지만, 차근차근 하면 됩니다!

# 문제부터 보자.

## Silver 3 - 피보나치 수 7 (#15624)

### 요약

- n이 주어졌을 때, n번째 피보나치 수를 1,000,000,007으로 나눈 값을 출력하는 프로그램을 작성하시오.

### 제약조건

- n의 범위는 1 <= n <= 1,000,000 이다.

# 피보나치 수가 뭐였지?

## Silver 3 - 피보나치 수 7 (#15624)

1, 1, 2, 3, 5, 8, 13, 21, 34, 55 …

$$P_n = P_{n-1} + P_{n-2}$$

# 재귀로 풀면 되지 않나?

## Silver 3 - 피보나치 수 7 (#15624)

```
function fibbo(n)
    if n <= 1
        return 1
    else return fibbo(n - 1) + fibbo(n - 2)
```

# 그래서 풀어보았습니다.

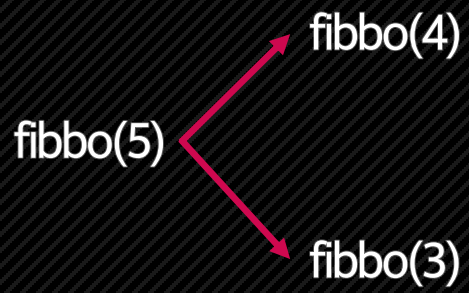
## Silver 3 - 피보나치 수 7 (#15624)

```
function fibbo(n)
    if n <= 1
        return 1
    else return fibbo(n - 1) + fibbo(n - 2)
```

~25: 1ms 미만
30: 4.4ms
35: 55.1ms
40: 570.9ms
45: 6.403초
50: 76.703초

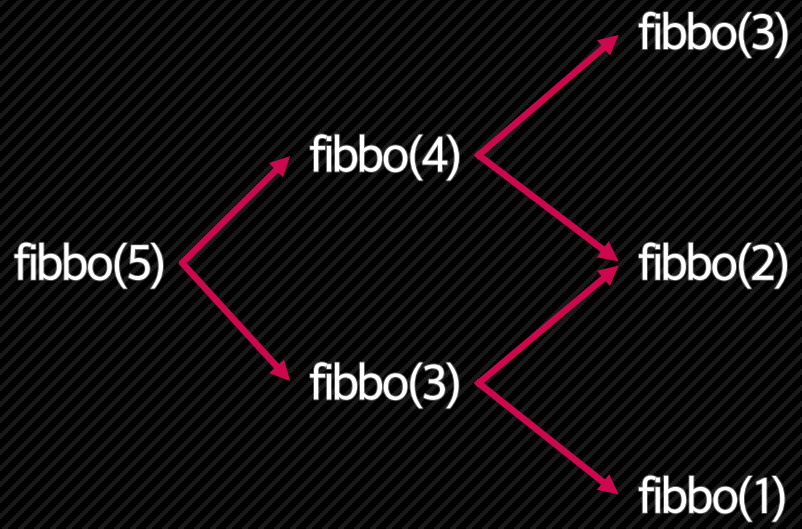
# 무슨 일이죠?

## Silver 3 - 피보나치 수 7 (#15624)



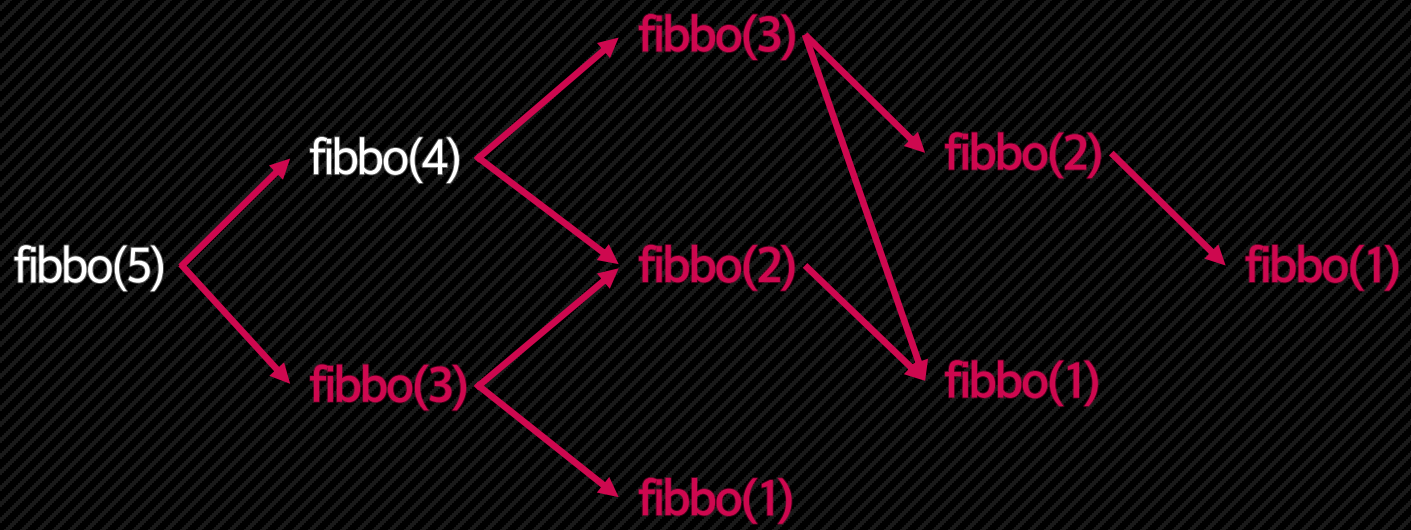
# 무슨 일이죠?

## Silver 3 - 피보나치 수 7 (#15624)



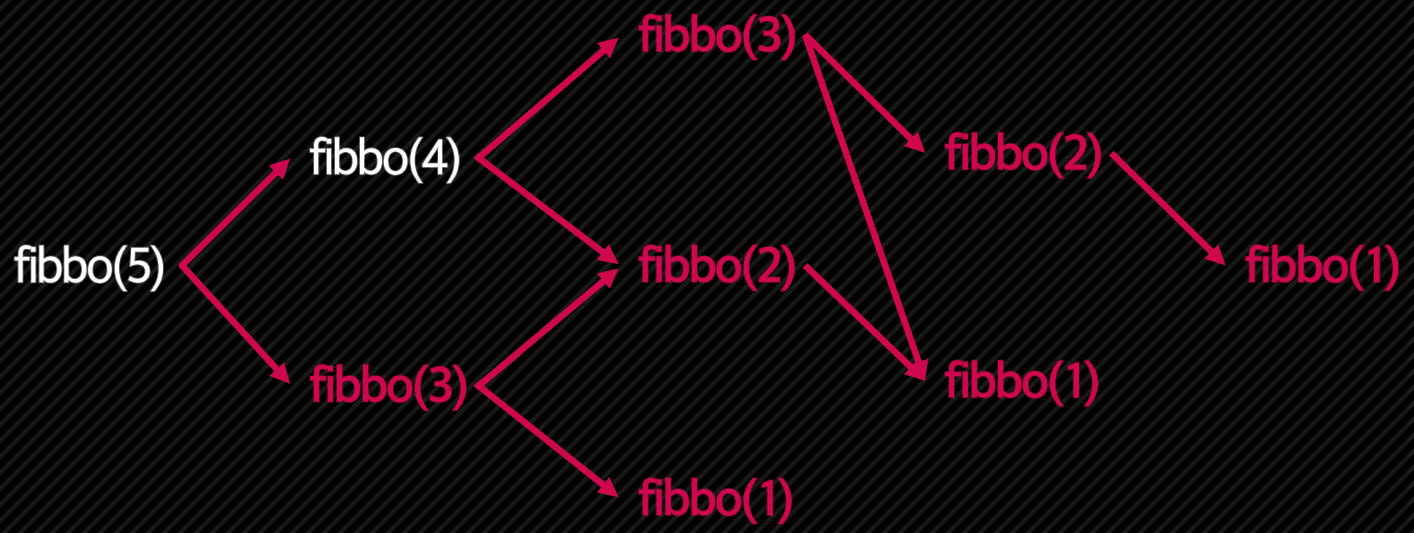
# 무슨 일이죠?

## Silver 3 - 피보나치 수 7 (#15624)



# 무슨 일이죠?

## Silver 3 - 피보나치 수 7 (#15624)




→ 이미 구했던 결과를 다시 구할 필요가 있을까?

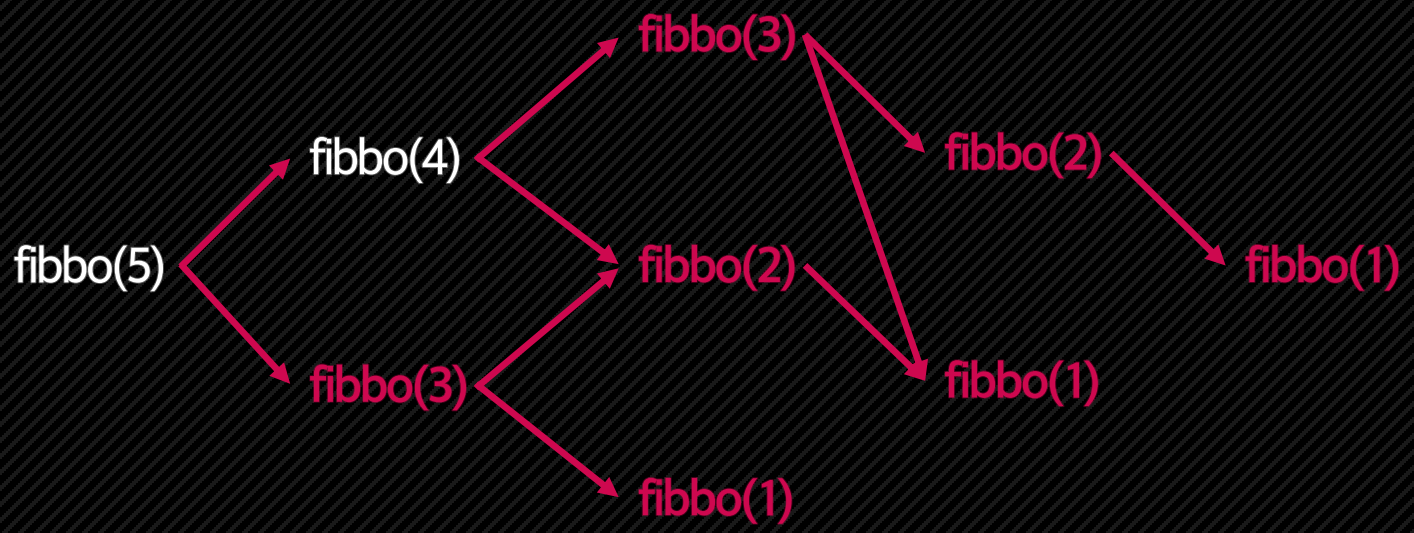
# 동적계획법

## Dynamic Programming

- 특정 범위까지의 값을 구하기 위해 **이전 범위의 값을 활용**하여 효율적으로 값을 얻는 기법.
- 이전 범위의 값을 저장하여, **(Memoization)** 똑같은 문제가 발생했을 때 이를 참조하여 해결함.

# 기본 아이디어를 떠올리자.

## Silver 3 - 피보나치 수 7 (#15624)




→ 이미 구했던 결과를 다시 구할 필요가 있을까?

# 기본 아이디어를 떠올리자.

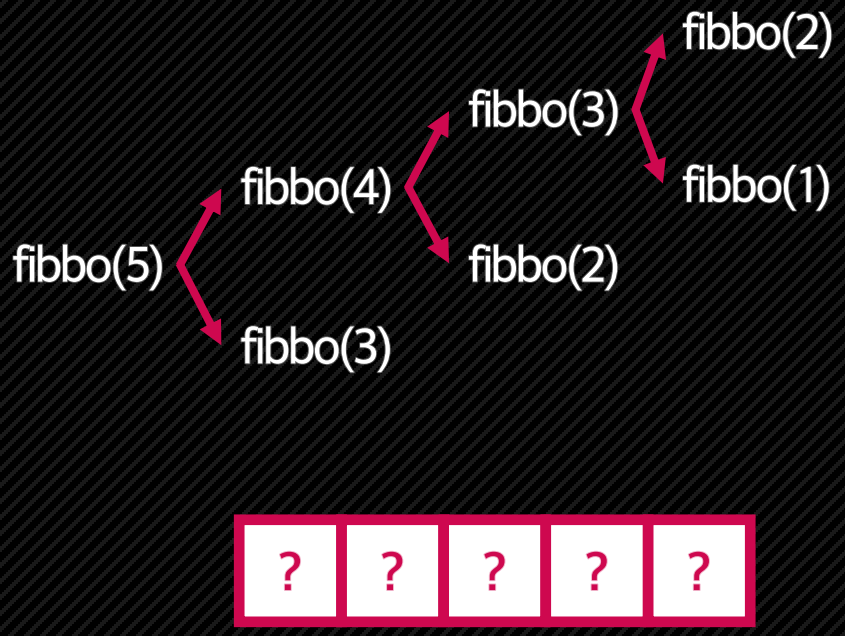
## Silver 3 - 피보나치 수 7 (#15624)


# 기본 아이디어를 떠올리자.

## Silver 3 - 피보나치 수 7 (#15624)

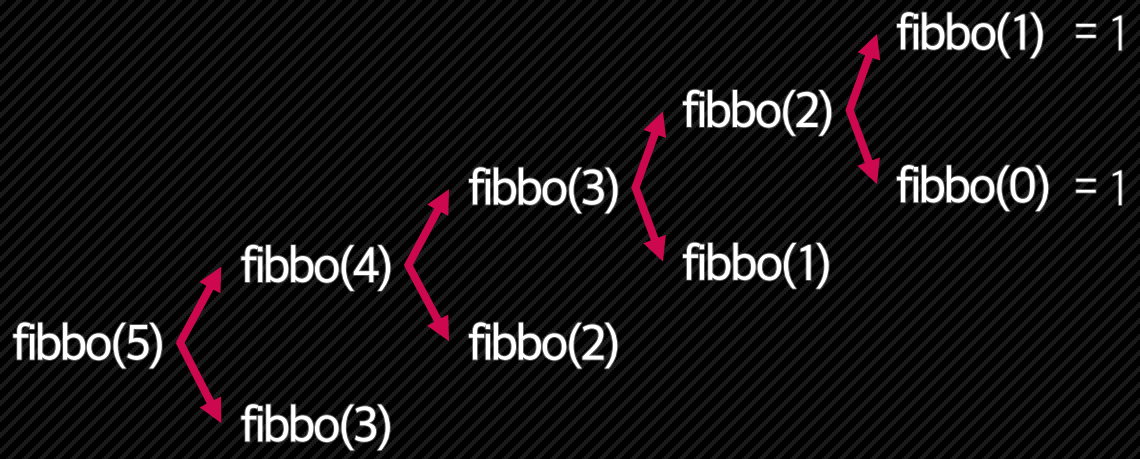

# 기본 아이디어를 떠올리자.

## Silver 3 - 피보나치 수 7 (#15624)



# 기본 아이디어를 떠올리자.

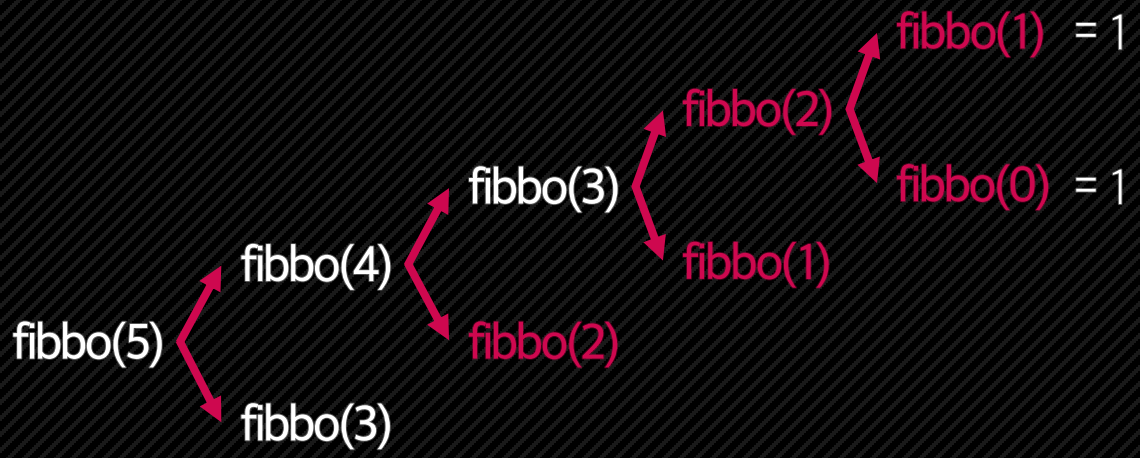
## Silver 3 - 피보나치 수 7 (#15624)




| 1 | ? | ? | ? | ? |
|---|---|---|---|---|

# 기본 아이디어를 떠올리자.

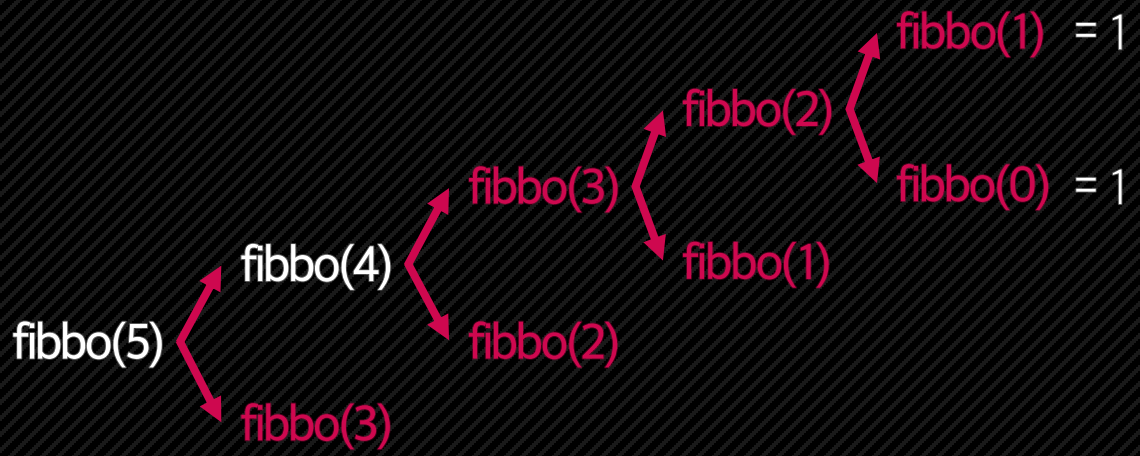
## Silver 3 - 피보나치 수 7 (#15624)




| 1 | 2 | ? | ? | ? |
|---|---|---|---|---|

# 기본 아이디어를 떠올리자.

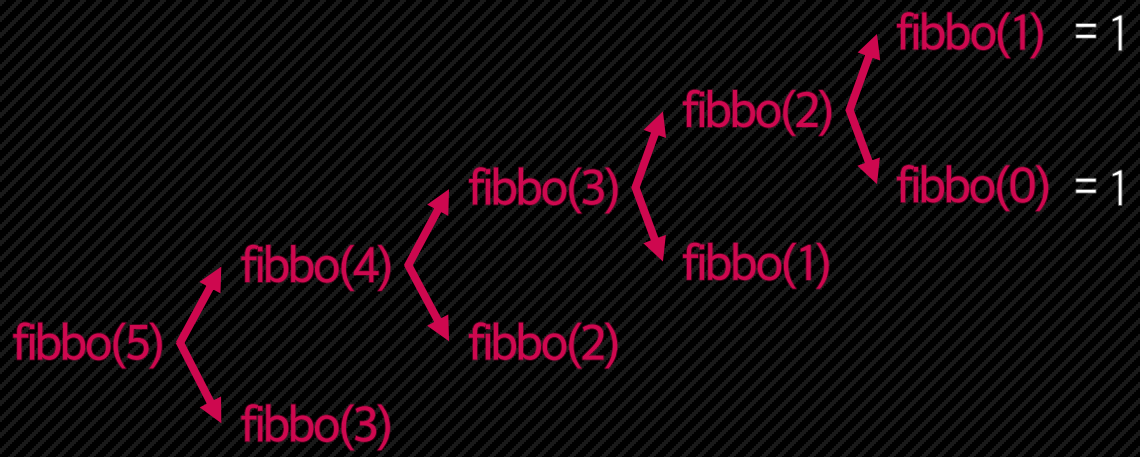
## Silver 3 - 피보나치 수 7 (#15624)




| 1 | 2 | 3 | ? | ? |
|---|---|---|---|---|

# 기본 아이디어를 떠올리자.

## Silver 3 - 피보나치 수 7 (#15624)




| 1 | 2 | 3 | 5 | 8 |
|---|---|---|---|---|

# 시간은 얼마나 줄어들까?

## Silver 3 - 피보나치 수 7 (#15624)

```
function fibbo(n)
    if n <= 1
        return 1
    if memo[n] != 0
        return memo[n]
    else return memo[n] = fibbo(n - 1) + fibbo(n - 2)
```

20: 10946 elapsed time: 9e-07s

...

50: 20365011074 elapsed time: 1.9e-06s

# 무슨 일이 생긴거죠?

## Silver 3 - 피보나치 수 7 (#15624)

```
function fibbo(n)
    if n <= 1
        return 1
    else return fibbo(n - 1) + fibbo(n - 2)
```

함수 호출 2,075,316,483회

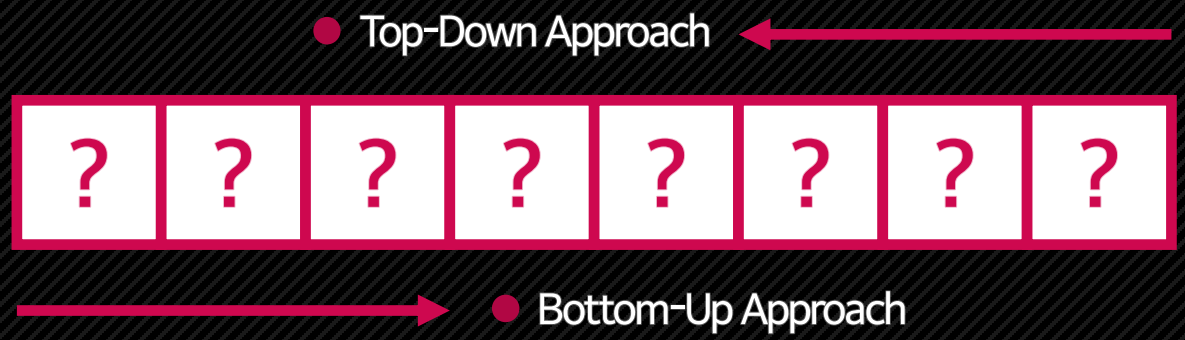
N = 50일 때

```
function fibbo(n)
    if n <= 1
        return 1
    if memo[n] != 0
        return memo[n]
    else return memo[n] = fibbo(n - 1) + fibbo(n - 2)
```

함수 호출 99회

# 다른 방식으로도 풀 수 있다?

## Silver 3 - 피보나치 수 7 (#15624)



# Bottom-Up

## Silver 3 - 피보나치 수 7 (#15624)

1, 1, 2, 3, 5, 8, 13, 21, 34, 55 …

$$P_n = P_{n-1} + P_{n-2}$$

그럼 그냥 미리 채워버리면 어떨까?

# Bottom-Up

## Silver 3 - 피보나치 수 7 (#15624)

```
function fibbo(n)
    set memo <- array with size n+1
    set memo[0] <- 1, memo[1] <- 1

    for (i = 2 ... n)
        memo[i] = memo[i - 1] + memo[i - 2]
    end
```

# Bottom-Up

## Silver 3 - 피보나치 수 7 (#15624)

```
function fibbo(n)
    set memo <- array with size n+1
    set memo[0] <- 1, memo[1] <- 1

    for (i = 2 ... n)
        memo[i] = memo[i - 1] + memo[i - 2]
    end
```

# Bottom-Up

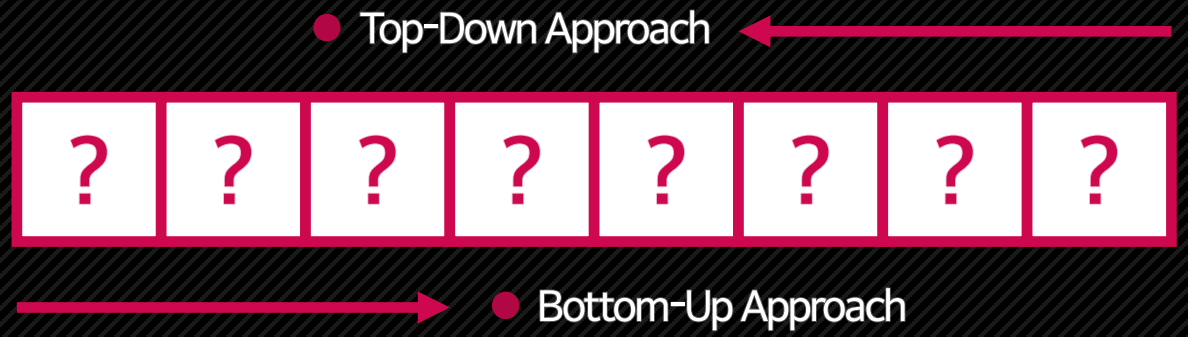
## Silver 3 - 피보나치 수 7 (#15624)

```
function fibbo(n)
    set memo <- array with size n+1
    set memo[0] <- 1, memo[1] <- 1

    for (i = 2 ... n)
        memo[i] = memo[i - 1] + memo[i - 2]
    end
```

# 뭘로 풀어야 해요?

## Silver 3 - 피보나치 수 7 (#15624)




- 이론상 시간복잡도는 두 기법 모두 동일하지만, Top-Down은 **재귀 오버헤드 때문에 실제 시간은 더 김.**
- 그러나 **충분히 무시할 수 있는 시간차**이기 때문에, 두 방법 모두 연습하는 것이 좋음!

# 언제 동적계획법을 쓸까?

- 큰 문제를 작은 문제로 쪼갰을 때, 작은 문제의 답을 통해 큰 문제의 답을 도출할 수 있는가?
- 큰 문제를 해결하기 위해 작은 문제의 답을 여러 번 구해야 하는가?

# 무슨 말인가요?

1, 1, 2, 3, 5, 8, 13, 21, 34, 55 …

$$P_n = P_{n-1} + P_{n-2}$$

- 큰 문제를 작은 문제로 쪼갰을 때, 작은 문제의 답을 통해 큰 문제의 답을 도출할 수 있는가?
  - → 피보나치 수열은 이전 항의 합으로 구해질 수 있음!
- 큰 문제를 해결하기 위해 작은 문제의 답을 여러 번 구해야 하는가?
  - → 앞에서 확인한 것 같이 같은 항을 여러 번 참고함!

# 쉬운 문제부터 차근차근…

## Silver 3 - 01타일 (#1904)

### 요약

- 00타일과 1타일이 있고, 이를 조합하여 숫자를 만들려고 한다.
- 숫자의 길이 N이 주어졌을 때, 만들 수 있는 모든 가짓수를 15746으로 나눈 나머지를 출력하는 프로그램을 작성하시오.

### 제약조건

- 숫자의 길이 N의 범위는 1 <= N <= 1,000,000 이다.

# 가능한 경우는?

## Silver 3 - 01타일 (#1904)

앞 상황 (고려 안함)


- 00을 붙이는 경우 (길이 +2)


- 1을 붙이는 경우 (길이 +1)

# 바꿔서 말하면…

## Silver 3 - 01타일 (#1904)

특정 항에 대하여…

| 앞 상황 (고려 안함) | 0 | 0 |
|---|---|---|

| 앞 상황 (고려 안함) | 1 |
|---|---|

## 바꿔서 말하면…

### Silver 3 - 01타일 (#1904)

특정 항에 대하여…

| 앞 상황 (고려 안함) | 0 | 0 |
|---|---|---|

| 앞 상황 (고려 안함) | 1 |
|---|---|

$$N_k = N_{k-1} + N_{k-2}$$

# 동적계획법의 기본?

## Silver 3 - 01타일 (#1904)

특정 항에 대하여…

| 앞 상황 (고려 안함) | 0 | 0 |
|---|---|---|

| 앞 상황 (고려 안함) | 1 |
|---|---|

$$N_k = N_{k-1} + N_{k-2}$$

**상황을 시뮬레이션 하고, 점화식을 세우는 연습을 하자!**

# 이건 해볼 수 있겠죠?

## Silver 3 - 1, 2, 3 더하기 (#9095)

**요약**

- 숫자 N이 주어졌을 때, 1, 2, 3의 합으로 나타낼 수 있는 경우의 수를 구하는 프로그램을 작성하시오.

**제약조건**

- 숫자의 N의 범위는 1 <= N <= 10 이다.

# 점화식을 연습해봅시다.

## Silver 3 - 2 × n 타일링 (#11726)

### 요약

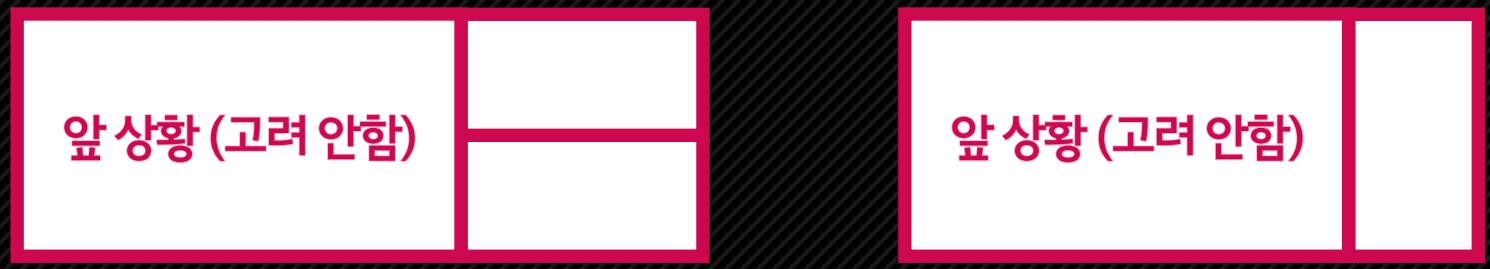
- 2 × n 크기의 직사각형을 1 × 2, 2 × 1 타일로 채우는 경우의 수를 구하는 프로그램을 작성하시오.

### 제약조건

- n 의 범위는 1 <= n <= 1,000 이다.

# 점화식을 연습해봅시다.

## Silver 3 - 2 × n 타일링 (#11726)




$$N_k = N_{k-1} + N_{k-2}$$

또?

# 다른걸 들고 왔어요.

## Silver 2 - 가장 긴 증가하는 부분수열 (#11053)

### 요약

- 수열 A가 주어졌을 때, 가장 긴 증가하는 부분 수열을 구하는 프로그램을 작성하시오.

### 제약조건

- 수열 A의 길이의 범위는 1 <= len(A) <= 1,000 이다.

# 다른걸 들고 왔어요.

## Silver 2 - 가장 긴 증가하는 부분수열 (#11053)

[10, 20, 10, 30, 20, 50]

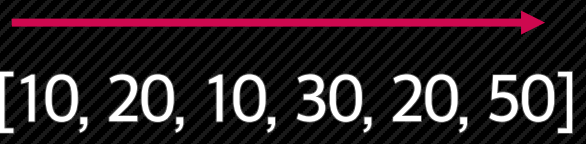
# 다른걸 들고 왔어요.

## Silver 2 - 가장 긴 증가하는 부분수열 (#11053)

[10, 20, 10, 30, 20, 50]

# 다른걸 들고 왔어요.

Silver 2 - 가장 긴 증가하는 부분수열 (#11053)



# 다른걸 들고 왔어요.

## Silver 2 - 가장 긴 증가하는 부분수열 (#11053)

# 다른걸 들고 왔어요.

## Silver 2 - 가장 긴 증가하는 부분수열 (#11053)

# 다른걸 들고 왔어요.

## Silver 2 - 가장 긴 증가하는 부분수열 (#11053)

# 다른걸 들고 왔어요.

## Silver 2 - 가장 긴 증가하는 부분수열 (#11053)


$$For\ all\ n+k\ s.t.\ Data_{n+k} > Data_n$$
$$DP_n = MAX(DP_{n+k}) + 1$$

## Silver 1 - 동전 2 (#2293)

### 요약

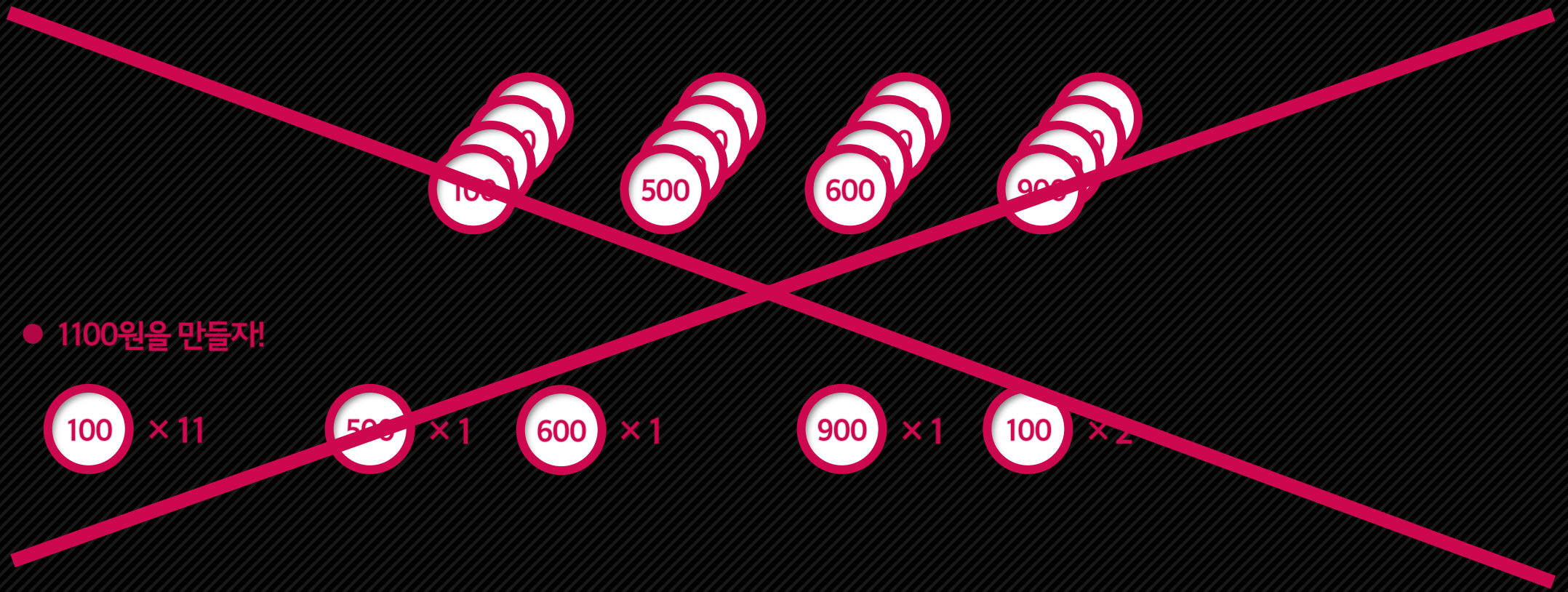
- N종류의 동전이 있고, K원을 만들려고 한다.
- 이때, K원을 만들기 위한 동전 개수의 최솟값을 구하는 프로그램을 작성하시오.

### 제약조건

- 만들려고 하는 금액 K의 범위는 $1 <= K <= 10,000$ 이다.
- 동전의 종류 N의 범위는 $1 <= M <= 100$ 이다.
- 동전의 가치 $A_i$의 범위는 $1 <= A_i <= 100,000$ 이다.

# 그리디로 못 푸는 건 알고…

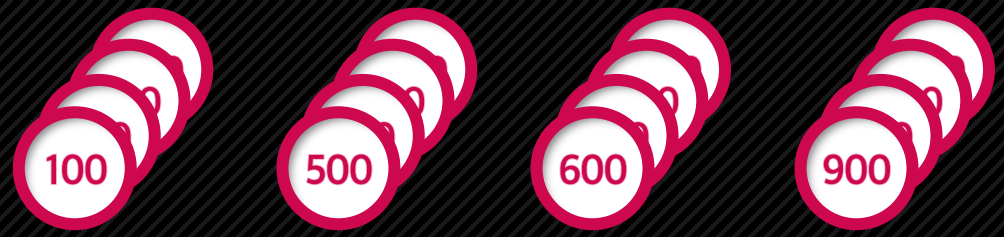
## Silver 1 - 동전 2




- 1100원을 만들자!

# 어떻게 생각해야 할까…

## Silver 1 - 동전 2 (#2293)




- 1100원을 만들자!

# 체계적으로 접근해보자.

## Silver 1 - 동전 2 (#2293)

- 우리가 구하려는 값이 뭘까?
- 특정 상황에서 선택 가능한 경우는 무엇이 있을까?
- 위에서 정리한 것을 기반으로, 현재 값은 어떻게 도출할 수 있을까?

# 체계적으로 접근해보자.

## Silver 1 - 동전 2 (#2293)

- 우리가 구하려는 값이 뭘까?

→ 현재 갖고 있는 동전을 활용해 N원을 만들 때, 필요한 동전의 최소 개수

→ DP 테이블을 '해당 금액을 만드는데 필요한 동전의 수'로 설정하자!

# 체계적으로 접근해보자.

## Silver 1 - 동전 2 (#2293)

- 우리가 구하려는 값이 뭘까?
  - → 현재 갖고 있는 동전을 활용해 N원을 만들 때, 필요한 동전의 최소 개수
  - → DP 테이블을 '해당 금액을 만드는데 필요한 동전의 수'로 설정하자!

- 특정 상황에서 선택 가능한 경우는 무엇이 있을까?
  - → 결국 각각의 동전을 하나씩 선택하는 경우 (단, 동전의 금액이 현재 금액보다 큰 경우 제외)

# 체계적으로 접근해보자.

## Silver 1 - 동전 2 (#2293)

- 우리가 구하려는 값이 뭘까?
  - → 현재 갖고 있는 동전을 활용해 N원을 만들 때, 필요한 동전의 최소 개수
  - → DP 테이블을 '해당 금액을 만드는데 필요한 동전의 수'로 설정하자!
- 특정 상황에서 선택 가능한 경우는 무엇이 있을까?
  - → 결국 각각의 동전을 하나씩 선택하는 경우 (단, 동전의 금액이 현재 금액보다 큰 경우 제외)
- 위에서 정리한 것을 기반으로, 현재 값은 어떻게 도출할 수 있을까?
  - → DP[N] = min(DP[N - coin[0]], DP[N - coin[1]] …) + 1

# 자, 그렇다면…

## Silver 1 - 동전 2 (#2293)


- 11원을 만들자!

# 자, 그렇다면…

## Silver 1 - 동전 2 (#2293)


- 11원을 만들자!

1 + 10원　　5 + 6원　　6 + 5원　　9 + 2원

# 많이 많이 연습합시다.

## Silver 1 - 카드 구매하기 (#11052)

### 요약

- 최대한 많은 돈을 들여 정확히 N개의 카드를 사려고 한다.
- i번째에 i개의 카드를 포함하는 카드팩 Pi의 가격이 주어진다.
- 최대의 비용을 구하는 프로그램을 작성하시오.

### 제약조건

- 카드의 개수 N의 범위는 $1 <= N <= 1,000$ 이다.
- 카드팩의 가격 $P_i$의 범위는 $1 <= P_i <= 10,000$ 이다.

# 2차원?

## Silver 1 - 쉬운 계단 수 (#10844)

### 요약

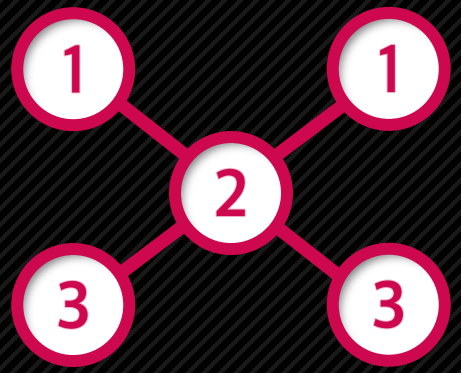
- 인접한 모든 수와 자리수가 정확히 1 차이나는 숫자를 계단 수라고 부른다.
- 숫자의 길이 N이 주어졌을 때, 만들 수 있는 계단 수의 갯수를 출력하는 프로그램을 작성하시오.

### 제약조건

- 숫자의 길이 N의 범위는 1 <= N <= 100 이다.

# 2차원?

## Silver 1 - 쉬운 계단 수 (#10844)




일반적으로 가능한 경우는 +1/-1

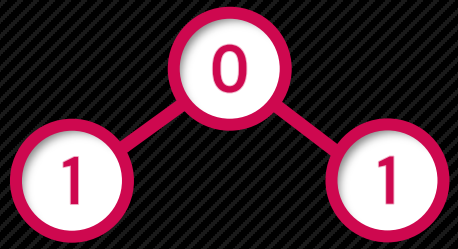
# 2차원?

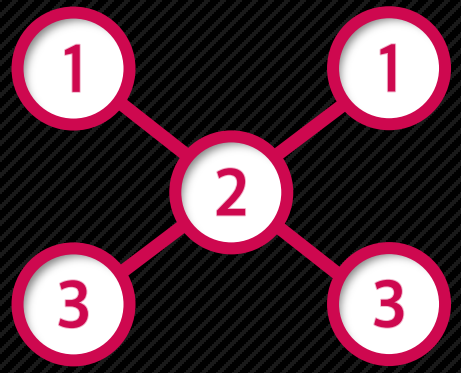
## Silver 1 - 쉬운 계단 수 (#10844)


어?

# 2차원?

## Silver 1 - 쉬운 계단 수 (#10844)




끝에 있는 0/9는 다른 숫자들과 달리, 가능한 경우가 한 가지 밖에 없음.

# 2차원?

## Silver 1 - 쉬운 계단 수 (#10844)

DP[10001][10]

↑ 몇 번째?　↑ 숫자는?

# 2차원?

## Silver 1 - 쉬운 계단 수 (#10844)

DP[10001][10]

몇 번째?　　숫자는?

(1 <= N <= 8) DP[X][N] = DP[X-1][N-1] + DP[X-1][N-2]

DP[X][0] = DP[X-1][1]

DP[X][9] = DP[X-1][8]

# 2차원도 연습해보자.

## Silver 1 - RGB 거리 (#1149)

### 요약

- N개의 집이 있고, 이를 빨강, 녹색, 파랑으로 칠하려고 한다. 단, 서로 이웃하고 있는 집은 다른 색으로 칠해져야 한다.
- 각각의 비용이 주어졌을 때, 모든 집을 칠하는 최솟값을 구하는 프로그램을 작성하시오.

### 제약조건

- 집의 수 N의 범위는 2 <= N <= 1,000 이다.
- 색을 칠하는 비용은 2 <= P <= 1,000 이다.

# 추가 추천 문제

## Silver 1 - 정수 삼각형

- 내려갈 방향은 몇 개죠?

## Silver 1 - 타일 채우기

- 점화식 세우기가 다소 까다롭습니다. 깊게 고민해보세요!

## Silver 1 - 전깃줄

- 앞에서 배웠던 기법을 그대로 사용합니다. 뭘까요?

## Level 2 - 가장 큰 정사각형 찾기

- 정사각형을 직접 그려보세요. 어디부터 확인해야 할까요?

“Any Question?”