

D.com



# Algorithm - 스터디

#Beginner 편

---

## 4강. 디피디피뿔



# 오늘의 학습 목표!

---

DP 문제 풀이 전략에 대해 알아보시다!

# 알고리즘

Algorithm

어떤 문제를 해결하는 방법

# 자료구조

## Data Structure

데이터를 표현하고 저장하는 방법

**동적 계획법**

Dynamic Programming

# 동적 계획법이 뭐죠?

---

# | 동적 계획법이란?

## 동적 계획법

“큰 문제를 작은 문제로 나눠서 푸는 기법!”

이 기법은 Richard bellman(리차드 벨먼)이 고안했으며,  
Dynamic이라는 단어가 멋있어 보여 선택했다고 합니다!  
#즉 Dynamic Programming라는 용어에 큰 의미는 없습니다!

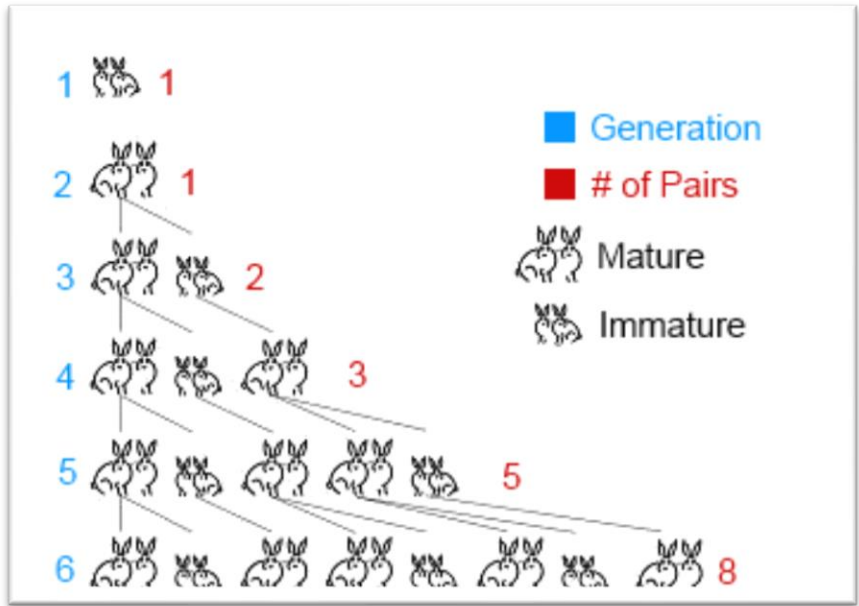
# | 동적 계획법이란?

동적 계획법  
“큰 문제를 작은 문제로 나눠서 푼다?”

작은 문제의 정답을 구하다 보면,  
큰 문제의 정답을 구할 수 있습니다!



# | 동적 계획법이란?



고등학교 때 한번씩 보셨을  
피보나치 수열입니다!  
우리는 이 문제를 풀기 위해  
‘점화식’이라는 개념을 도입했습니다!

$$A_n = A_{n-1} + A_{n-2}$$

# 동적 계획법이란?

## ‘점화식’ 인접한 항들 사이의 관계식

N번째 항을 구하려면, N-1번째 항과 N-2번째 항을 알아야 하고,  
N-1번째 항을 구하려면 N-2번째 항과 N-3번째 항도 알아야 하고, ...  
결국 N번째 항은 그보다 작은 항들을 품으로써 알 수 있다!  
즉, 피보나치 N 문제를 구하기 위해서는, 그보다 작은 피보나치 문제들을 풀어야 한다!  
이것이 바로 다이나믹 프로그래밍의 기초 아이디어입니다!  
#구하기 쉬운 작은 문제들을 풀어나가면 결국은 구하기 어려운 큰 문제를 풀 수 있다!

# 동적 계획법이란?

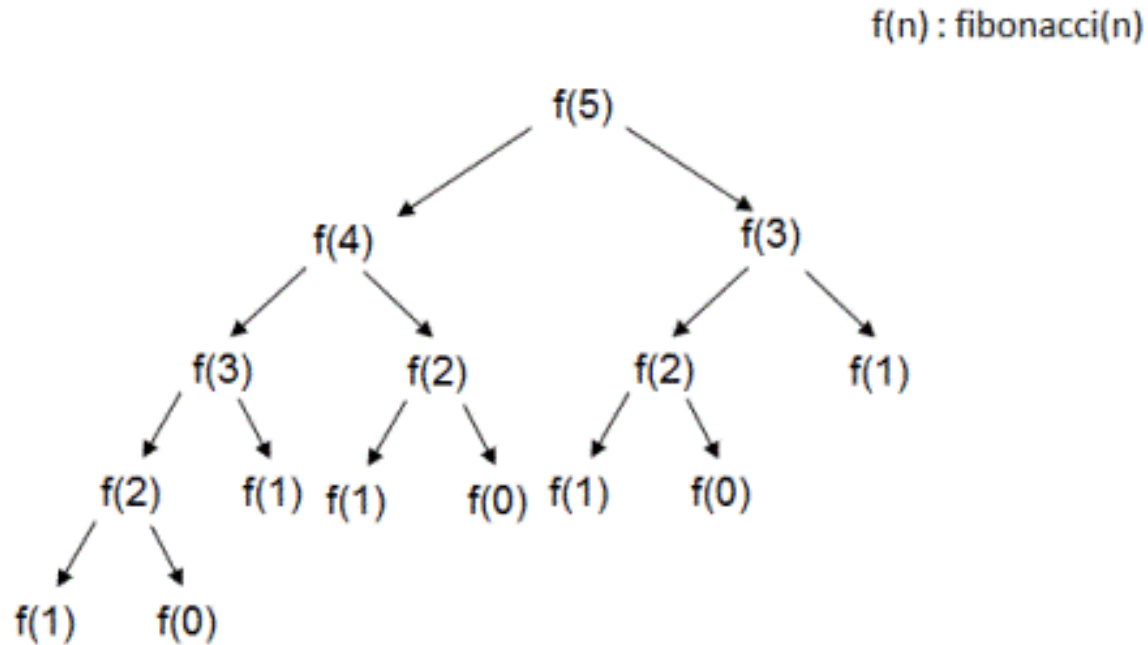
```
int Fibo(int N)
{
    if (N == 1)
    {
        return 0;
    }
    else if (N == 2)
    {
        return 1;
    }
    else
    {
        return Fibo(N - 1) + Fibo(N - 2);
    }
}
```

앞에서 알아본 원리를 구현하면  
다음과 같습니다!(재귀 이용)

하지만 앞의 코드로 피보나치 50을 빠르게 구할 수 있을까요?  
원하는 시간안에 구할 수 없다면, 그 이유는 무엇일까요?

# 동적 계획법이란?

피보나치 재귀로 풀이하면 다음과 같습니다!

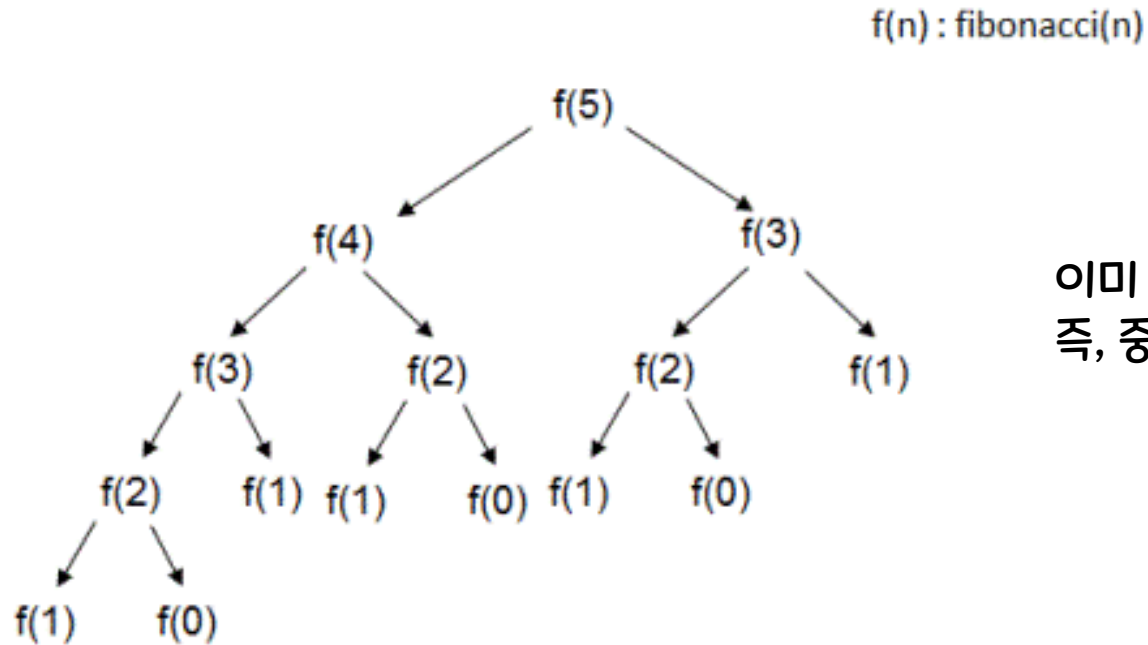


Recursion tree generated for computing 5<sup>th</sup> number of fibonacci sequence

피보나치 5를 구하는  
동작 순서를 나타내면 다음과 같습니다!

# 동적 계획법이란?

피보나치 재귀로 풀이하면 다음과 같습니다!



이미 호출했던 함수를 다시 호출하는 것을 볼 수 있습니다!  
즉, 중복이 발생하고 있습니다!

Recursion tree generated for computing 5<sup>th</sup> number of fibonacci sequence

**중복을 피해라!**

# Memoization

## 메모이제이션

:반복되는 결과를 메모해두는 것!

#메모'리'제이션 아닙니다!

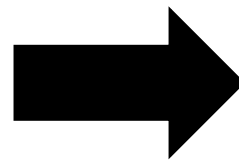


**중복을 피하는 일은  
매우 중요합니다!**

```

int Fibo(int N)
{
    if (N == 1)
    {
        return 0;
    }
    else if (N == 2)
    {
        return 1;
    }
    else
    {
        return Fibo(N - 1) + Fibo(N - 2);
    }
}

```



```

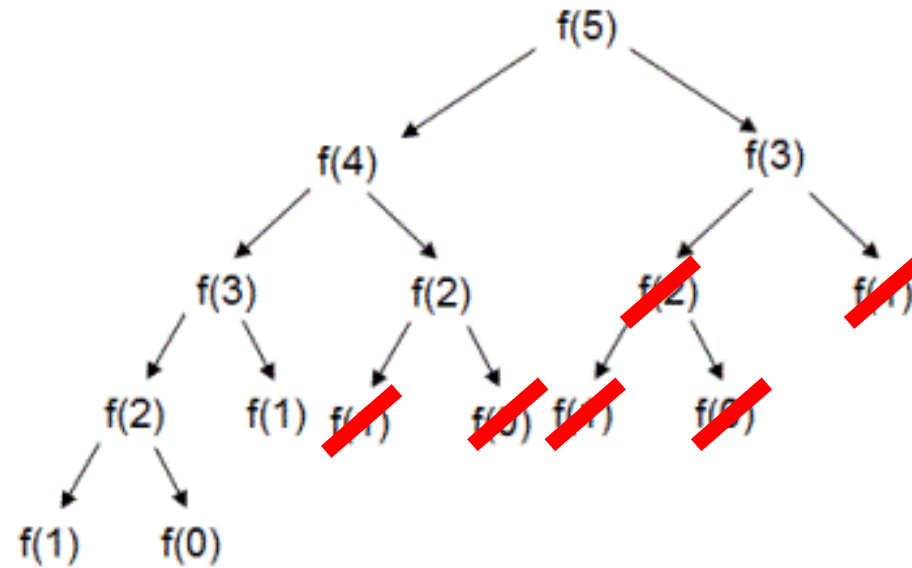
int memo[100];
int Fibo(int N)
{
    if (N == 1)
    {
        return 0;
    }
    else if (N == 2)
    {
        return 1;
    }
    else
    {
        if (memo[N] > 0)
        {
            return memo[N];
        }
        memo[N] = Fibo(N - 1) + Fibo(N - 2);
        return memo[N];
    }
}

```

**메모이제이션을 적용한 코드입니다!**  
 구한 값을 메모해두고 나중에 필요할 때 사용할 수 있습니다.

## 메모의 힘!

$f(n) : \text{fibonacci}(n)$



Recursion tree generated for computing 5<sup>th</sup> number of fibonacci sequence

**다시, 다이나믹 프로그래밍을 정  
의하면 다음과 같습니다.**

**큰 문제는 작은 문제들을 통해 푼다!**  
**단, 작은 문제의 정답을 구했으면, 어딘가에 메모해두자!**

**DP를 구현하는 방식에는 두가지가 있습니다!**

**탑다운(Top-down)  
바텀업(Bottom-Up)**

## **탐다운(Top-down)**

**가장 큰 문제에서부터 점점 작은 문제를 풀어나가는 방식입니다.  
주로 재귀를 이용하여 구현합니다.**

```
int memo[100];
int Fibo(int N)
{
    if (N == 1)
    {
        return 0;
    }
    else if (N == 2)
    {
        return 1;
    }
    else
    {
        if (memo[N] > 0)
        {
            return memo[N];
        }
        memo[N] = Fibo(N - 1) + Fibo(N - 2);
        return memo[N];
    }
}
```

## 탑다운(Top-down)

Fibo(N)을 구하기 위해, Fibo(N-1)과 Fibo(N-2)라는 작은 문제로 나눠 푼다.  
이렇게 큰 문제에서 점점 작은 문제를 풀어나가다 보면 결국 큰 문제를 풀 수 있다.

## **바텀업(Bottom-up)**

**탑다운과 반대로, 작은 문제에서 점점 큰 문제를 풀어나가는 방식입니다.  
주로 반복문을 이용하여 구현합니다.**





```
int dp[100];
int Fibo(int N)
{
    dp[1] = 0;
    dp[2] = 1;
    for (int i = 3; i <= N; i++)
    {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    return dp[N];
}
```

## 바텀업(Bottom-Up)

Fibo(N)을 구하기 위해, Fibo(1)부터 풀어나간다.  
이렇게 작은 문제에서 점점 큰 문제를 풀어나가다 보면  
결국 큰 문제를 풀 수 있다.

# 탭다운 VS 바텀업

**“탭다운 방식은 점화식을 표현하기 쉽다는 장점이 있고,  
바텀업 방식은 재귀 함수를 사용하지 않기 때문에  
메모리와 처리시간을 절약할 수 있다는 장점이 있습니다.”**

**각자 자신에게 잘 맞는 방법을 사용하시면 됩니다!  
간혹 두 방법 간의 구현 난이도 편차가 크게 나타나는 문제도 있기에,  
두 방법 모두 알아두셔야 합니다!**

# DP 문제를 접근하는 방법

1. dp문제인지 판단하기!(작은 문제로 쪼갤 수 있는가? 메모이제이션이 필요한가?)
  2. 메모이제이션을 위한 배열 혹은 벡터를 만든다.
  3. 큰 문제를 작은 문제로 표현하고, 점화식을 세운다.

<https://www.acmicpc.net/problem/2747>

<https://www.acmicpc.net/problem/2748>

**피보나치 문제를 DP로 풀어봅시다!**

**1로 만들기 문제를 같이 풀어봅시다!**

<https://www.acmicpc.net/problem/1463>

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	90080	29522	18731	31.943%

## 문제

정수  $X$ 에 사용할 수 있는 연산은 다음과 같이 세 가지 이다.

1.  $X$ 가 3으로 나누어 떨어지면, 3으로 나눈다.
2.  $X$ 가 2로 나누어 떨어지면, 2로 나눈다.
3. 1을 뺀다.

정수  $N$ 이 주어졌을 때, 위와 같은 연산 세 개를 적절히 사용해서 1을 만들려고 한다. 연산을 사용하는 횟수의 최솟값을 출력하시오.

## 입력

첫째 줄에 1보다 크거나 같고,  $10^6$ 보다 작거나 같은 정수  $N$ 이 주어진다.

## 출력

첫째 줄에 연산을 하는 횟수의 최솟값을 출력한다.

# 1단계 dp문제인지 확인하기!

**N을 1로 만드는 문제는  
N/3, N/2, N-1을 1로 만드는 문제로 생각할 수 있지 않을까?(작은 문제로 쪼개기)**


**2단계 메모이제이션을 위한 배열을 만들자!**

**문제에서 정수 N은 1000000이하가 조건이므로,  
Int dp[1000001]로 배열을 선언하자!**



## 3단계 점화식을 세워보자!

$dp[i] \Rightarrow i$ 를 1로 바꾸는데 필요한 최소 연산 횟수라고 해보자!  
그렇다면  $dp[i]$ 는  $dp[i/3] + 1$ ,  $dp[i/2] + 1$ ,  $dp[i-1] + 1$  중에서의  
최소값이라 할 수 있지 않을까?



```
int dp[1000001];
int go(int N)
{
    for (int i = 2; i <= N; i++)
    {
        dp[i] = dp[i - 1] + 1;
        if (i % 2 == 0)
        {
            dp[i] = min(dp[i], dp[i / 2] + 1);
        }
        if (i % 3 == 0)
        {
            dp[i] = min(dp[i], dp[i / 3] + 1);
        }
    }
    return dp[N];
}
```

**구현해보자! (바텀업)**

```
int d[1000001];
int go(int n) {
    if (n == 1) {
        return 0;
    }
    if (d[n] > 0) {
        return d[n];
    }
    d[n] = go(n-1) + 1;
    if (n%2 == 0) {
        int temp = go(n/2) + 1;
        if (d[n] > temp) {
            d[n] = temp;
        }
    }
    if (n%3 == 0) {
        int temp = go(n/3) + 1;
        if (d[n] > temp) {
            d[n] = temp;
        }
    }
    return d[n];
}
```

**구현해보자! (탐다운)**

**한 문제만 더 같이 풀어봅시다!**

<https://www.acmicpc.net/problem/11726>

## 문제

---

$2 \times n$  크기의 직사각형을  $1 \times 2$ ,  $2 \times 1$  타일로 채우는 방법의 수를 구하는 프로그램을 작성하시오.

아래 그림은  $2 \times 5$  크기의 직사각형을 채운 한 가지 방법의 예이다.



## 입력

---

첫째 줄에  $n$ 이 주어진다. ( $1 \leq n \leq 1,000$ )

## 출력

---

첫째 줄에  $2 \times n$  크기의 직사각형을 채우는 방법의 수를 10,007로 나눈 나머지를 출력한다.

# 1 단계 dp문제인지 확인하기!

2\*N 타일 문제는  
그보다 작은 타일에 타일을 추가하는 방법으로 풀 수 있지 않을까?  
(문제 분류가 dp니까)

**2단계 메모이제이션을 위한 배열을 만들자!**

**문제에서 정수 N은 1000까지가 조건이므로,  
Int dp[1001]로 배열을 선언하자!**

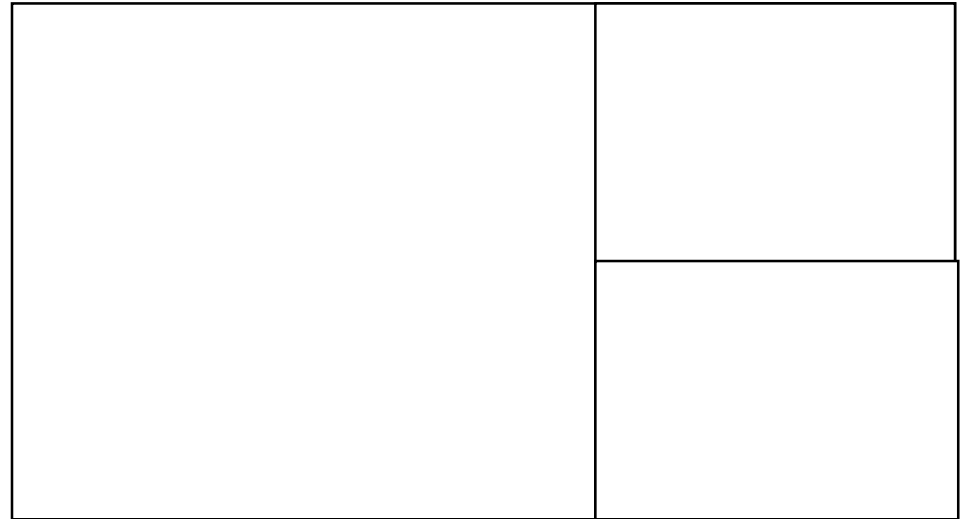
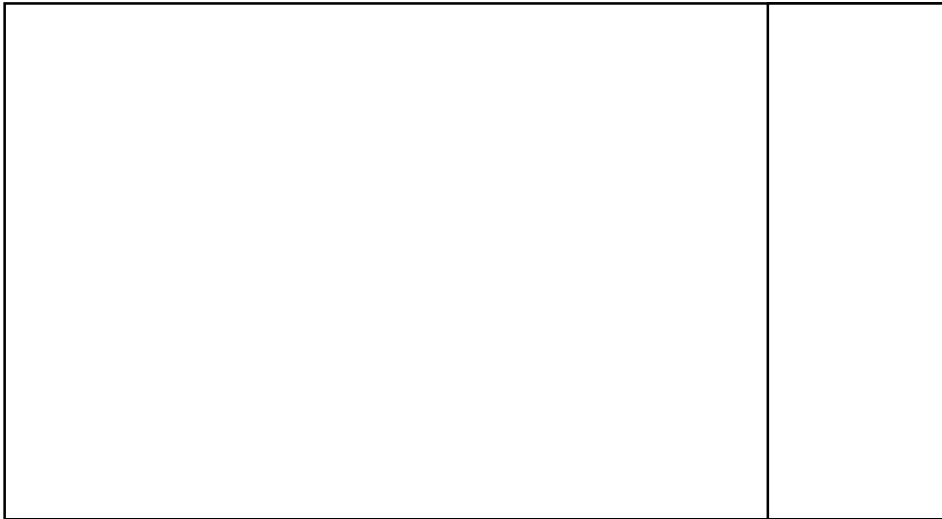
## **3단계 점화식을 세워보자!**

**$dp[i] \Rightarrow 2 \times i$  직사각형을 채우는 방법의 수라 해보자!**



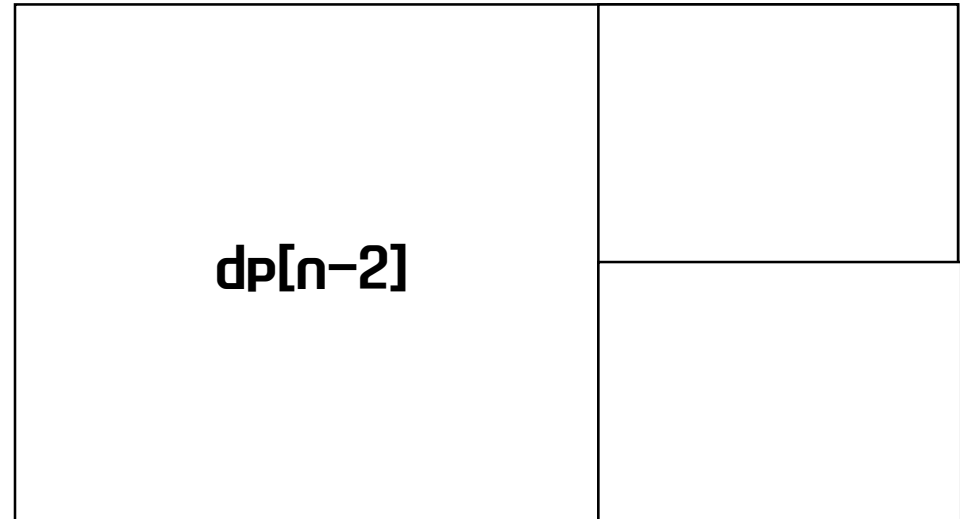
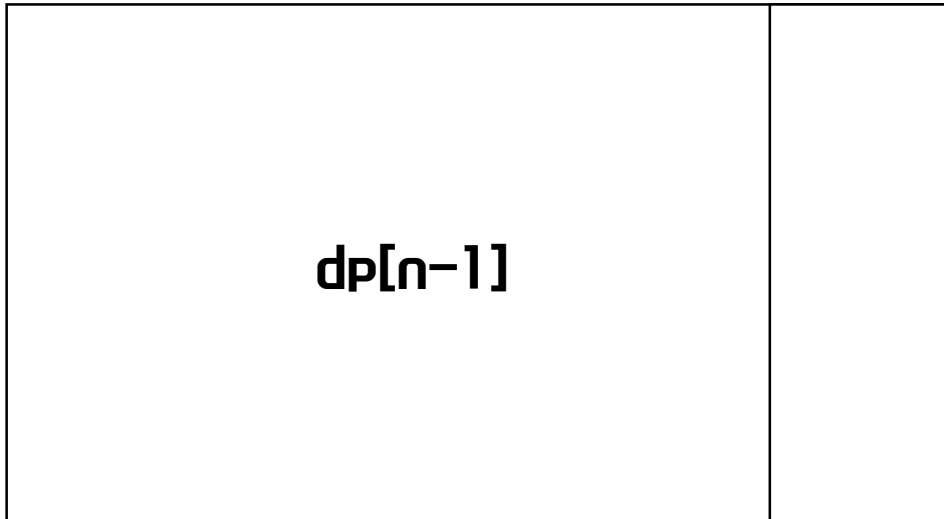
## 3단계 점화식을 세워보자!

가장 오른쪽에 타일을 놓는 방법은 아래와 같이 두가지가 있다!



## 3단계 점화식을 세워보자!

가장 오른쪽에 타일을 놓는 방법은 아래와 같이 두가지가 있다!



## 3단계 점화식을 세워보자!

즉,  $dp[n]$ 은  $dp[n-1]$ 과  $dp[n-2]$ 의 합으로 표현할 수 있다!

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

#결국 피보나치 점화식과 같습니다!

# 3단계 점화식을 세워보자!

한번 직접 타일을 그려보며 세운 점화식이 맞는지  
확인해봅시다!  
(파워포인트로 그리기 힘들어요 ㅜㅜ)

## 꿀팁!

```
for(int i=3;i<=data;i++){  
    d[i] = d[i-1] + d[i-2];  
    d[i] %= 10007;  
}
```

앞으로도 문제에 N으로 나눈 나머지를 출력하라는 문제를 많이 볼 수 있습니다.

대부분 문제의 정답의 숫자가 매우 커 오버플로우가 발생할 때 조건으로 붙습니다!

**이때, 결과를 구하고 마지막에 나눌 필요 없이, 구하는 중간에 계속 나눠줘도 상관없습니다!**

#가능한 이유는 다음을 참고해주세요

<https://www.acmicpc.net/problem/10430>

# 다음시간!

---

그래프와 트리라는 자료구조에 대해 알아보시다!



# The end.

“네 시작은 미약하였으나 네 나중은 심히 창대하리라”  
-욥기 8장 7절