ALOHA 중급반 보충자료

# 수학, 기초 DP

# 목차

# 0. Problem Solving 기초

- 왜 틀렸을까?
- PS 수학
    - 최대공약수
    - 소수

# 1. Dynamic Prgramming 기초

- DP란?
- DP 접근법
- 문제 풀어보기

# 0. Problem Solving 기초

- 왜 틀렸을까?
- PS 수학
  - 최대공약수
  - 소수

"일단 워밍업부터 합시다."

## 왜 틀렸을까?

# 틀리는 유형

- 컴파일에러 ( Compile Error, CE )
- 런타임에러 ( Runtime Error, RTE )
- 시간초과 ( Time Limit Exceeded, TLE )
- 메모리 초과 ( Memory Limit Exceeded, MLE )
- 틀렸습니다 ( Wrong Answer, WA )

## 왜 틀렸을까?

# 컴파일에러

- 컴파일하다가 터진 경우
- 예제는 맞춰보고 제출하자. ~~( 조교가 싫어해요;; )~~
- 아래에 뜨는 에러 메시지를 읽어보자.

## 왜 틀렸을까?

# 런타임 에러

- Segmentation Fault
  - 재귀함수가 너무 깊어졌다
  - 시스템 메모리를 건드렸다 ~~( 조교에게 문의하세요 )~~
  - 배열 범위 밖을 참조했다 ( Out of Index, OoI )

- Divide by 0 ( div zero, div0 )

- 재귀함수가 너무 깊어졌다. ( Over depth )

왜 틀렸을까?

# 런타임 에러

- Segmentation Fault

주로 정의되지 않은 메모리 영역에 접근해서 생기는 오류
ex) int arr[100];
　　이때, arr은 0~99번의 칸을 가진다.
　　즉,-1번이나 100번 칸 등에 접근을 시도하면 이 오류를 만날 수 있다.

**배열 크기를 처음부터 넉넉하게 선언하자.**
**DFS같은 탐색 문제를 풀 때 예외처리를 잘 하자.**

## 왜 틀렸을까?

# 시간 초과

- 내가 짠 알고리즘이 출제자가 원하는 답보다 비효율적인 경우 (최악의 경우 반복문이 몇 번 도는지 확인해보자)

- 컴퓨터는 1초에 단순연산(+,-)을 10억 번 정도 할 수 있다.

- 계산을 덜 할 수 있는 다른 풀이를 생각해보자.

왜 틀렸을까?

# 메모리 초과

- OoM ( Out of Memory ) 라고도 한다.

- 배열 크기가 너무 클 때
  보통 메모리 제한은 128MB이다.
  128 MB == 131072 KB
  == 134217728 Bytes
  == sizeof(int) * 33554432
  == sizeof(long long) * 16777216

- STL(queue, stack 등)에 너무 많은 데이터가 들어가 있을 때

- 스택 영역에 너무 많은 함수가 쌓인 경우
  보통 재귀함수에서 이러한 경우가 생긴다.

## 왜 틀렸을까?

# 틀렸습니다

- 예외 처리
  (내가 생각하지 못한 경우는 무엇일까?)

- 오버 플로우
  int의 범위는 $-2^{32} \sim 2^{32} - 1$ 까지이다. ( 약 21억 )
  경우에 따라 long long을 써야 할 수도 있다. ( 약 20자리 )

- 잘못된 풀이 ~~(ㅠㅠ)~~

# 최대 공약수 / 최소 공배수

- 유클리드 호제법

## 큰 수에서 작은 수를 빼도
## 최대공약수는 같다.

# 최대 공약수 / 최소 공배수

```
int GCD( int a, int b )
{
    if ( a == 0 )
        return b;
    if ( a > b )
        return GCD( b, a );
    return GCD( a, b - a );
}
```

문제점?

문제: a랑 b의 차이가 크면?

PS 수학

# 최대 공약수 / 최소 공배수

```
int GCD( int a, int b )
{
	if ( a % b == 0 )
	{
		return b;
	}
	return GCD( b, a % b );
}
```

그럼, 최소 공배수는?

이미 다 알고 있겠지만…….

문제: b==0이라면?

ALOHA The algorithm club.

# 최대 공약수 / 최소 공배수

```
int GCD( int a, int b )
{
	if ( a == 0 )
	{
		return b;
	}
	return GCD(a % b,  b);
}
```

그럼, 최소 공배수는?

이미 다 알고 있겠지만…….

# 최대 공약수 / 최소 공배수

```
int LCM( int a, int b )
{
    return a * b / GCD( a, b );
}
```

이렇게 하면 된다!

# 최대 공약수 / 최소 공배수

```
int LCM( int a, int b )
{
    return a / GCD( a, b ) * b;
}
```

~~이렇게 하면 된다?~~
아니다. 이렇게 해야한다!

# 최대 공약수 / 최소 공배수

```
int LCM( int a, int b )
{
    if ( GCD(a, b) == 0 )
        return 0;
    return a / GCD( a, b ) * b;
}
```

~~이렇게 하면 된다?~~
~~아니다. 이렇게 해야한다?~~
아니다. 이렇게 해야한다!

PS 수학

# 9613

# GCD 합

ALOHA
The algorithm club.

# 2436

# 공약수

2436

# 공약수

크게 나누면 두가지 경우가 있다.

1. GCD == LCM
   답이 되는 두 자연수 a, b가 같은 걸 의미한다.

2. 그 외의 경우
   a = a′ * GCD, b = b′ * GCD (a′ 와 b′ 는 서로소)
   a′ * b′ = LCM / GCD
   를 의미한다.
   즉 LCM / GCD 의 약수 중 서로소이고, 두 합이 최소인 값(a′ , b′)을 찾은 다음,
   이 값을 GCD에 곱하면 된다.

   서로소인 건, GCD가 1인지 확인하면 된다.

ALOHA
The algorithm club.

# 소수 구하기

- 에라토스테네스의 체
  소인수 or 인수를 써야 하는 문제 같은 경우, 소수를 미리 구해 놓으면 편하다.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

# 소수 구하기

- 에라토스테네스의 체
  소인수 or 인수를 써야 하는 문제 같은 경우, 소수를 미리 구해 놓으면 편하다.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

# 소수 구하기

- 에라토스테네스의 체
  소인수 or 인수를 써야 하는 문제 같은 경우, 소수를 미리 구해 놓으면 편하다.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

…… 이런 식으로 합성수를 하나하나 지워 나가면 됩니다.

# 소수 구하기

- 에라토스테네스의 체
  소인수 or 인수를 써야 하는 문제 같은 경우, 소수를 미리 구해 놓으면 편하다.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

완성되면 이런 모양새가 된다.

# 소수 구하기

- 에라토스테네스의 체
  그럼 이걸 코드로는 어떻게 구현할까요?

```
for ( int i = 2; i <= n; i++ )
{
    for ( int j = 2; i*j <=n; j++ )
        chk[i * j] = 1;
}
```

# 소수 구하기

- 에라토스테네스의 체
  그럼 이걸 코드로는 어떻게 구현할까요?

```
for ( int i = 2; i <= n; i++ )
{
    for ( int j = i * i; j <=n; j+=i )
        chk[j] = 1;
}
```

이렇게 체크를 i*i부터 하면
훨씬 적은 반복으로도
소수를 구할 수 있다.

n = 1000 기준
전자는 5070회
후자는 2550회

# 소수 구하기

- 에라토스테네스의 체
  그럼 이걸 코드로는 어떻게 구현할까요?

```
for ( int i = 2; i <= n; i++ )
{
    if ( chk[i] ) // 소수가 아니다
        continue;
    for ( int j = i * i; j <=n; j+=i )
        chk[j] = 1;
}
```

# 1929

# 소수 구하기

6588

# 골드바흐의 추측

6588

# 골드바흐의 추측

일단, 에라토스테네스의 체와, 그로 인해 만들어진 소수 배열(벡터)이 필요하다.

N의 범위가 1,000,000 까지이기 때문에 $O(N^2)$로 소수를 구하면 시간 초과가 난다.

3부터 체크하기 시작해서, (주어진 수 – 현재 소수) 가 소수인지 에라토스테네스의 체 배열을 이용해 체크하고, 소수일 경우, 현재 소수와 (주어진 수 – 현재 소수) 를 출력

3부터 체크하는 이유는 2를 제외한 모든 소수는 홀수이며, 테스트케이스는 4보다 큰 수이기 때문이다.

만약 주어진 수보다 작거나 같은 최대의 소수에 도달했을 때까지 값을 구하지 못했을 경우, “Goldbach's conjecture is wrong.” 을 출력

# 조합 (Combination) 구하기

- C( n, r ) = n! / ( r! * ( n – r )! )
  물론 이 값을 factorial 함수를 짜서 직접 구할 수도 있다.

  BUT 그렇게 할 경우 팩토리얼을 구하는 과정에서 오버플로우가 날 수 있고,
  조합을 구할 때마다 새로 계산을 해야 한다는 단점이 있다.

- 굳이 팩토리얼을 구하지 않더라도, 우리는 점화식을 이용해서 조합을 구할 수 있다.

# 조합 (Combination) 구하기

- C( n, r ) = C( n-1, r ) + C( n-1, r-1 )

$$\boxed{{}_{n}C_{r} + {}_{n}C_{r+1} = {}_{n+1}C_{r+1}}$$

$$
\begin{array}{c}
{}_{1}C_{0} \quad {}_{1}C_{1} \\
{}_{2}C_{0} \quad {}_{2}C_{1} \quad {}_{2}C_{2} \\
{}_{3}C_{0} \quad {}_{3}C_{1} \quad {}_{3}C_{2} \quad {}_{3}C_{3} \\
\vdots \\
{}_{n}C_{0} \quad {}_{n}C_{1} \quad {}_{n}C_{2} \cdots {}_{n}C_{r} + {}_{n}C_{r+1} \cdots {}_{n}C_{n-2} \quad {}_{n}C_{n-1} \quad {}_{n}C_{n} \\
{}_{n+1}C_{r+1}
\end{array}
$$

➔ 위 줄 둘의 합을 아래줄 가운데에 쓴다.

➔ 한줄 내려왔으니 앞번호는 n+1,

➔ 사선 ↙ 방향으로 뒷번호가 같으므로 뒷번호는 r+1

11051

# 이항 계수 2

11051

# 이항 계수 2

파스칼의 삼각형을 이차원 배열로 구현한다.

DP[n][m] = C(n,m)

DP[1][0] = DP[1][1] = 1 로 초기화하고, 나머지는 아래의 점화식을 통해 DP로 구하자

DP[n][m] = DP[n-1][m-1] + DP[n-1][m]

BUT, DP[n][0]은 예외로 1로 초기화하는 걸 권장한다. 위의 점화식을 그대로 쓰게 되면 배열의 -1번 칸에 접근할 수 있어 Runtime Error가 날 수 있다.

# 1. Dynamic Programming 기초

- DP 도입부
- LIS (N^2)

" 걱정하지 말아요. 정말 기초…일 테니까. "

ALOHA
The algorithm club.

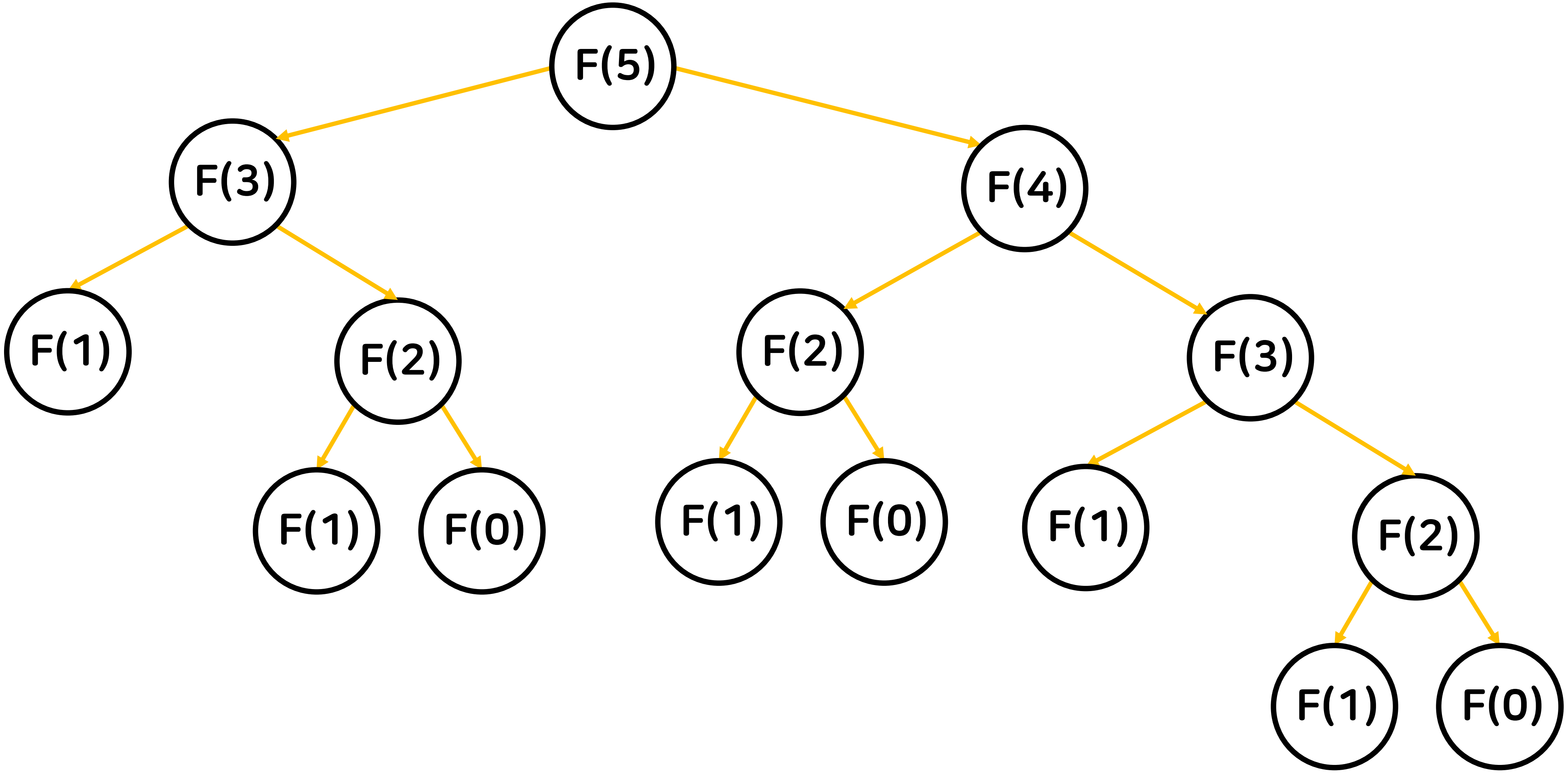
# DP란?

- 복잡한 문제를 간단한 여러 개의 문제로 나누어 푸는 방법

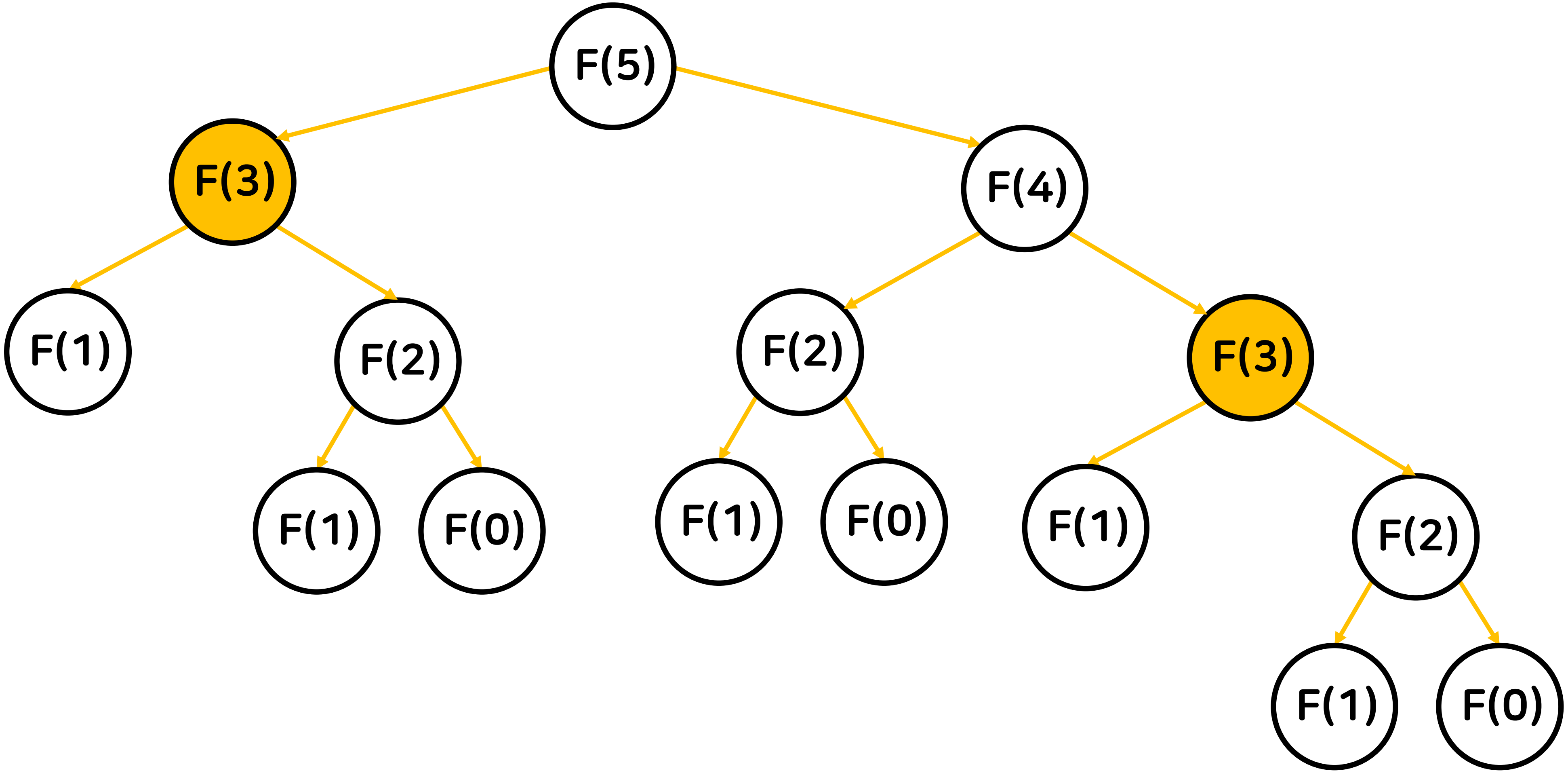
- 하위 문제의 답을 메모리에 저장하는 것으로, 연산 횟수를 줄인다.

- 그리디 알고리즘과의 차이
  그리디는 매 순간의 최적해를 찾기 때문에, 결과값이 꼭 최적해가 아닐 수도 있다.
  그러나 DP는 부분 문제들 중에서 최적의 값을 고르며 진행하기 때문에
  결과값이 최적해가 됨을 보장한다.

# DP 도입부

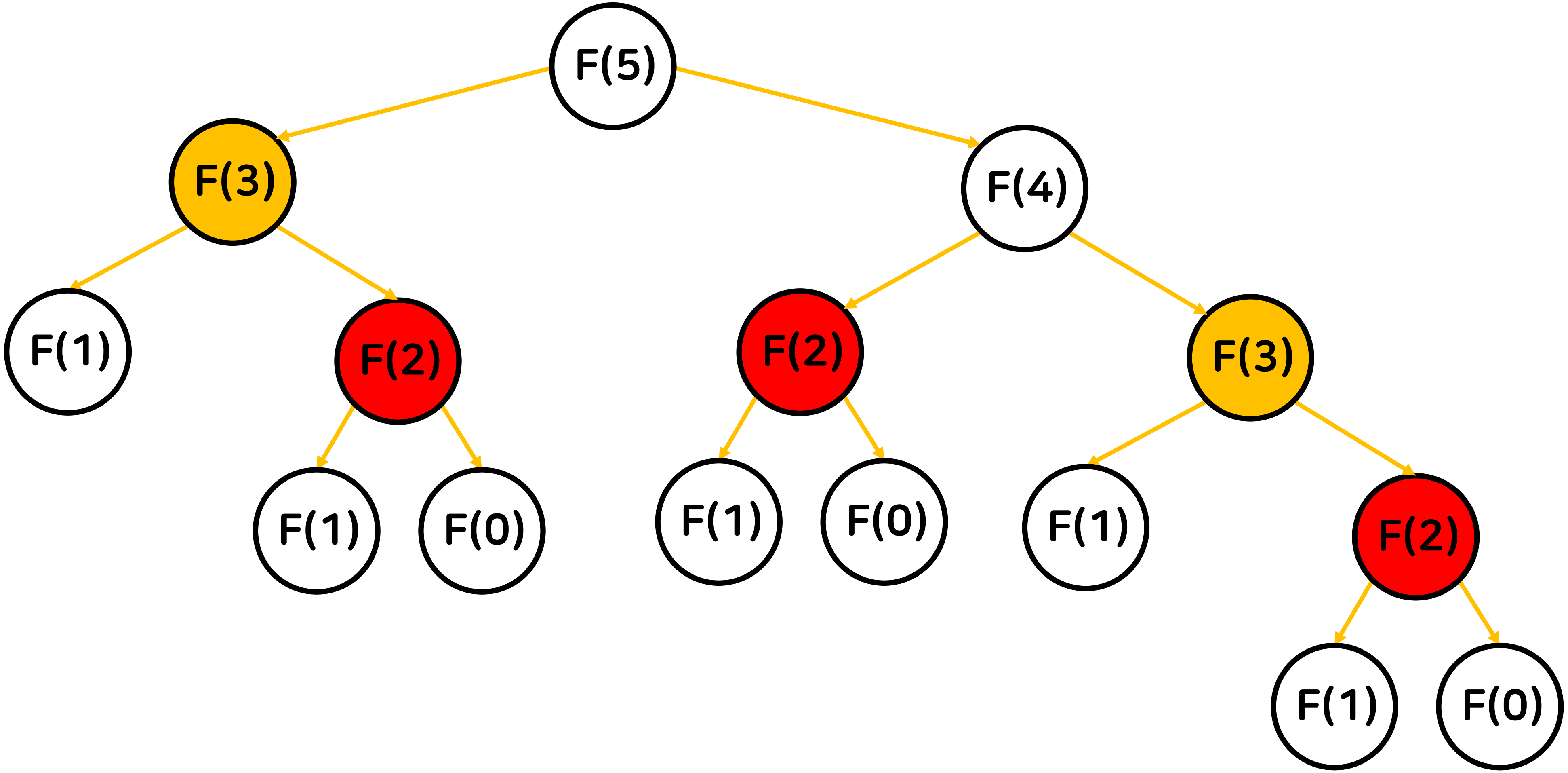


# DP 도입부




ALOHA The algorithm club.

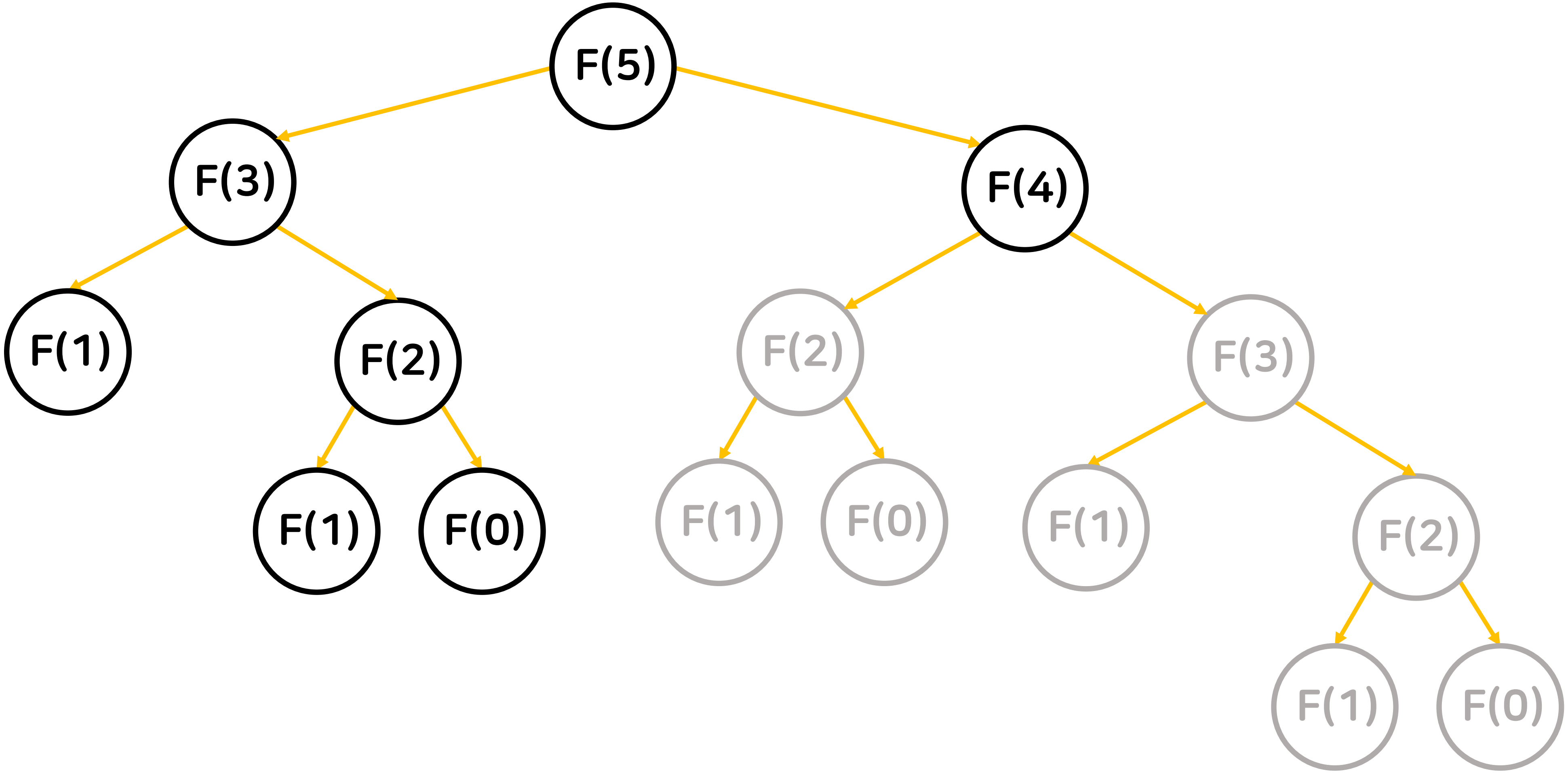
# DP 도입부




ALOHA
The algorithm club.

# 같은 연산을 불필요하게 반복한다!
# 한 번 연산한 결과는 메모리에 저장하자!
# →Memoization

# DP 도입부



# 구현 방식 : Top-Down

- F(N)을 구하기 위해 F(N-1)과 F(N-2)를 구한 뒤 더하는 방식
- 재귀함수를 이용해 구현

```
int fibo( int idx )
{
	if ( idx == 0 ) return 0;
	else if( idx == 1 ) return 1;

	return fibo( idx - 1 ) + fibo( idx – 2 );
}
```

# 구현 방식 : Bottom-Up

- F(N-1)과 F(N-2)를 먼저 구하고, 두 값을 더해서 F(N)을 구하려는 방식
- 반복문을 이용해 구현

```
int fibo[100] = { 0, 1 };

for ( int i = 2; i <= n; i++ )
    {
        fibo[ i ] = fibo[ i - 1 ]+fibo[ i - 2 ];
    }
}
```

# Top-Down vs Bottom-Up

Top-Down
- 직관적인 구현 가능

- 부분 문제 간의 의존 관계 고려 X

Bottom-Up
- 슬라이딩 윈도우를 통한
  메모리 절약 가능

- 부분 문제 간의 의존 관계 고려 O
  → 비직관적인 구현

# Sliding Window

- Window : 내가 지금 보고 있는 범위
  DP 배열로써 저장될 영역이다.

- 실질적으로 연산에 필요한 데이터만 저장하고 있으면, 메모리를 절약할 수 있다.
  메모리 제한이 빡센 문제의 경우, 이 방식이 문제 해결의 Key Point가 된다.

- 피보나치 수의 경우, N번째 수를 구하기 위해서는 N-1, N-2번째의 수가 필요하므로, 실제로 연산에 필요한 배열의 사이즈는 2이다.

# Sliding Window

- 피보나치 수의 경우, N번째 수를 구하기 위해서는 N-1, N-2번째의 수가 필요하므로, 실제로 연산에 필요한 배열의 사이즈는 2이다.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | - | - | - | - | - | - | - | - | - |

# Sliding Window

- 피보나치 수의 경우, N번째 수를 구하기 위해서는 N-1, N-2번째의 수가 필요하므로, 실제로 연산에 필요한 배열의 사이즈는 2이다.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | - | - | - | - | - | - | - | - |

# Sliding Window

- 피보나치 수의 경우, N번째 수를 구하기 위해서는 N-1, N-2번째의 수가 필요하므로, 실제로 연산에 필요한 배열의 사이즈는 2이다.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | - | - | - | - | - | - | - |

# Sliding Window

- 피보나치 수의 경우, N번째 수를 구하기 위해서는 N-1, N-2번째의 수가 필요하므로, 실제로 연산에 필요한 배열의 사이즈는 2이다.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 |

# Sliding Window

- 피보나치 수의 경우, N번째 수를 구하기 위해서는 N-1, N-2번째의 수가 필요하므로, 실제로 연산에 필요한 배열의 사이즈는 2이다.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 |

DP 도입부

# 2096

# 내려가기

ALOHA
The algorithm club.

2096

# 내려가기

일단 DP 문제이니 배열부터 만들어보자

DP[N][i] = N번째 줄에서 i번째 칸을 지나갈 때 얻을 수 있는 최대/최소값

이때는 i = 0,1,2 일 때 모두 따로 점화식을 만들어야 한다.

DP[N][0] = a + max(DP[N-1][0], DP[N-1][1])
DP[N][1] = b + max(DP[N-1][0], DP[N-1][1], DP[N-1][2])
DP[N][2] = c + max(DP[N-1][1], DP[N-1][2])

위의 점화식은 최댓값의 경우로, 최솟값은 max를 min으로 바꾸면 된다.

그러나……

2096

# 내려가기

메모리 제한이 4MB이다. ~~세상에…~~
4MB = sizeof(int) * 1048576
심지어 주어지는 입력의 수가 최대 100,000이고, 최대, 최솟값을 각각 DP로 구해야 하기 때문에, Sliding Window가 절대적으로 필요하다.

그럼 우리는 얼마나 메모리를 아낄 수 있을까?

사실 현재 줄 바로 전의 점수합만 가지고 있으면, 현재 줄에서의 값을 구할 수 있다.
또한, 현재 줄의 숫자는 다음 줄에선 사용되지 않으므로 굳이 배열로 저장하지 않아도 된다.

말로만 하면 어려우니, 직접 해 보자.

2096

# 내려가기

현재 줄 : 1

현재 줄을 지나가면 받을 수 있는 점수

| a | b | c |
|---|---|---|
| 1 | 2 | 3 |

최댓값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| 줄 | a | b | c |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 2 | 3 |

최솟값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| 줄 | a | b | c |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 2 | 3 |

2096

# 내려가기

현재 줄 : 2

현재 줄을 지나가면 받을 수 있는 점수

| a | b | c |
|---|---|---|
| 4 | 5 | 6 |

최댓값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| 줄 | a | b | c |
| 1 | 1 | 2 | 3 |
| 2 | 6 | 8 | 9 |

최솟값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| 줄 | a | b | c |
| 1 | 1 | 2 | 3 |
| 2 | 5 | 6 | 8 |

2096

# 내려가기

현재 줄 : 3

현재 줄을 지나가면 받을 수 있는 점수

| a | b | c |
|---|---|---|
| 4 | 9 | 0 |

최댓값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| 줄 | a | b | c |
| 2 | 6 | 8 | 9 |
| 3 | 14 | 18 | 9 |

최솟값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| 줄 | a | b | c |
| 2 | 5 | 6 | 8 |
| 3 | 9 | 14 | 6 |

**최댓값 = 18, 최솟값 = 6**

2096

# 내려가기

이걸 코드로 어떻게 구현할 수 있을까?
Mod 연산 (나머지 연산)을 사용하면 Sliding Window를 쉽게 구현할 수 있다.

DP 배열을 2*3으로 만들고, 현재 보는 줄 번호를 N이라 하면

DP[N % 2][0] = a + max(DP[(N-1) % 2][0], DP[(N-1) % 2][1])
DP[N % 2][1] = b + max(DP[(N-1) % 2][0], DP[(N-1) % 2][1], DP[(N-1) % 2][2])
DP[N % 2][2] = c + max(DP[(N-1) % 2][1], DP[(N-1) % 2][2])

이런 식으로 점화식을 바꾸기만 하면 된다.
(2로 나눈 나머지는 0 아니면 1이기 때문에)

2096

# 내려가기

현재 줄 : 1

현재 줄을 지나가면 받을 수 있는 점수

| a | b | c |
|---|---|---|
| 1 | 2 | 3 |

최댓값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| idx | a | b | c |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 2 | 3 |

현재 줄

최솟값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| idx | a | b | c |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 2 | 3 |

2096

# 내려가기

현재 줄 : 2

현재 줄을 지나가면 받을 수 있는 점수

| a | b | c |
|---|---|---|
| 4 | 5 | 6 |

최댓값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| idx | a | b | c |
| 0 | 6 | 8 | 9 |
| 1 | 1 | 2 | 3 |

현재 줄

최솟값

| | 마지막으로 지나간 자리 | | |
|---|---|---|---|
| idx | a | b | c |
| 0 | 5 | 6 | 8 |
| 1 | 1 | 2 | 3 |

2096

# 내려가기

현재 줄 : 3

현재 줄을 지나가면 받을 수 있는 점수

| a | b | c |
| --- | --- | --- |
| 4 | 9 | 0 |

최댓값

| | 마지막으로 지나간 자리 | | |
| --- | --- | --- | --- |
| idx | a | b | c |
| 0 | 6 | 8 | 9 |
| 1 | 12 | 18 | 9 |

현재 줄

최솟값

| | 마지막으로 지나간 자리 | | |
| --- | --- | --- | --- |
| idx | a | b | c |
| 0 | 5 | 6 | 8 |
| 1 | 9 | 14 | 6 |

**최댓값 = 18, 최솟값 = 6**

# DP에서 배열이 갖는 의미

- 원하는 답을 얻기 위해 구한 부분 문제들의 답을 저장하는 영역
  ex) 기존의 피보나치 수를 저장하는 배열

- 문제에 따라 배열에 어떤 기준으로 값을 저장할지 달라진다.
  ex) 피보나치 수 : 1차원 배열 (DP[n] = n 번째 피보나치 수)
  하노이 탑 : 1차원 배열 (DP[n] = n 개의 원판을 이동시키는 최소 횟수)
  문제에 따라서 2차원이 될 수도 있다.

DP 도입부

# 1149

# RGB 거리

ALOHA
The algorithm club.

1149

# RGB 거리

앞에서 푼 2096번 내려가기와 상당히 유사한 문제이다.
심지어 메모리도 넉넉해서 슬라이딩 윈도우도 사용하지 않아도 된다.

DP 배열은
DP[N][i] = N번째 집에 i색을 칠했을 때의 최소 금액 (0 = R, 1 = G, 2 = B)

점화식은

DP[N][0] = a + min(DP[N-1][1], DP[N-1][1])
DP[N][1] = a + min(DP[N-1][0], DP[N-1][2])
DP[N][2] = a + min(DP[N-1][1], DP[N-1][0])

그전 집에 i색 이외의 색을 칠했을 때의 금액 중 작은 값과,
현재 i색을 칠하는데 드는 값의 합

…역시 직접 해보자

1149

# RGB 거리

현재 집 번호 : 1

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 26 | 40 | 83 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | - | - | - |
| 2 | - | - | - |
| 3 | - | - | - |

1149

# RGB 거리

현재 집 번호 : 1

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 26 | 40 | 83 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | - | - | - |
| 3 | - | - | - |

ALOHA
The algorithm club.

1149

# RGB 거리

현재 집 번호 : 2

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 49 | 60 | 57 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | - | - | - |
| 3 | - | - | - |

ALOHA
The algorithm club.

# RGB 거리

현재 집 번호 : 2

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 49 | 60 | 57 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | 89 | - | - |
| 3 | - | - | - |

1149

# RGB 거리

현재 집 번호 : 2

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 49 | 60 | 57 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | 89 | 86 | - |
| 3 | - | - | - |

1149

# RGB 거리

현재 집 번호 : 2

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 49 | 60 | 57 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | 89 | 86 | 83 |
| 3 | - | - | - |

1149

# RGB 거리

현재 집 번호 : 3

현재 집을 칠하는 비용

| R | G | B |
| --- | --- | --- |
| 13 | 89 | 99 |

| | 마지막으로 칠한 색 | | |
| --- | --- | --- | --- |
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | 89 | 86 | 83 |
| 3 | - | - | - |

1149

# RGB 거리

현재 집 번호 : 3

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 13 | 89 | 99 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | 89 | 86 | 83 |
| 3 | 96 | - | - |

1149

# RGB 거리

현재 집 번호 : 3

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 13 | 89 | 99 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | 89 | 86 | 83 |
| 3 | 96 | 172 | - |

1149

# RGB 거리

현재 집 번호 : 3

현재 집을 칠하는 비용

| R | G | B |
|---|---|---|
| 13 | 89 | 99 |

| | 마지막으로 칠한 색 | | |
|---|---|---|---|
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | 89 | 86 | 83 |
| 3 | 96 | 172 | 185 |

1149

# RGB 거리

현재 집 번호 : 3

현재 집을 칠하는 비용

| R | G | B |
| --- | --- | --- |
| 13 | 89 | 99 |

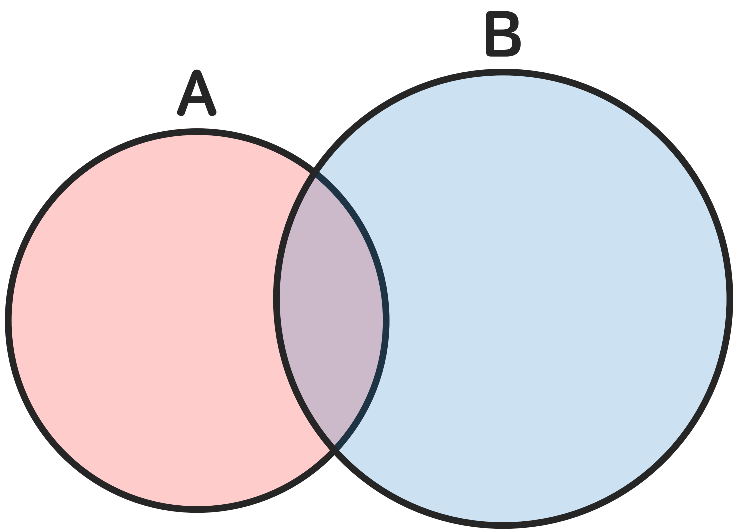
| | 마지막으로 칠한 색 | | |
| --- | --- | --- | --- |
| N | R | G | B |
| 0 | 0 | 0 | 0 |
| 1 | 26 | 40 | 83 |
| 2 | 89 | 86 | 83 |
| 3 | 96 | 172 | 185 |

**최솟값 : 96**

# 포함 배제의 원리

- 집합 A, B가 있고, |A| = 3, |B| = 4, |A ∩ B| = 2 일 때,
  |A ∪ B| = ?

- |A ∪ B| = |A| + |B| - |A ∩ B|



11659

# 구간 합 구하기 4

11659

# 구간 합 구하기 4

N과 M의 범위가 각각 100,000이다.
즉 최악의 경우 $100{,}000^2$ 번 덧셈을 해야 해서 단순 덧셈만 하면 시간 초과가 난다.

그럼 어떻게 해야 문제를 풀 수 있을까?
풀이 공개 전 먼저 생각해 보자.

# 구간 합 구하기 4

N과 M의 범위가 각각 100,000이다.
즉 최악의 경우 $100{,}000^2$ 번 덧셈을 해야 해서 단순 덧셈만 하면 시간 초과가 난다.

그럼 어떻게 해야 문제를 풀 수 있을까?
풀이 공개 전 먼저 생각해 보자.

DP[N] = 첫번째부터 N번째 값까지의 합

i = 2, j = 4 일 때

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 5 | 4 | 3 | 2 | 1 |

11659

# 구간 합 구하기 4

점화식

DP[j] – DP[i-1] → 문제의 답

1에서 j번째 값까지의 합에서 1에서 i번째 값 전까지의 합을 뺀 것!

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 5 | 4 | 3 | 2 | 1 |

# LIS란?

- Longest Incresing Subsequence :
  가장 긴 증가하는 부분수열

- Subsequence :
  어떤 수열의 원소 일부를 뽑아내 만든 수열,
  꼭 연속한 원소를 뽑아야 할 필요는 없으나, 뽑아낸 원소의 순서는 기존 수열과 같아야 한다.
  ex )
  A = {1, 2, 3, 4, 5, 6, 7, 8, 9}
  {1,3,5,9} ⊂ Sub(A);
  {1,5,3,9} ⊄ Sub(A);

# LIS란?

- Longest Incresing Subsequence :
  가장 긴 증가하는 부분수열

- Incresing Subsequence :
  어떤 수열의 부분 수열 중, 원소가 증가하는 순서로 된 부분 수열
  ex)
  A = {3, 1, 4, 2, 6, 5, 8, 9, 7}
  {3, 4, 6, 8, 9} ⊂ LIS(A)
  {1, 2, 5, 8, 9} ⊂ LIS(A)
  {2, 6, 8, 9} ⊄ LIS(A)

# LIS란?

- Longest Incresing Subsequence :
  가장 긴 증가하는 부분수열
  어떤 수열의 IS 중에서, 가장 길이가 긴 것
  A = {3, 1, 4, 2, 6, 5, 8, 9, 7}
  {3, 4, 6, 8, 9} ⊂ IS(A)
  {2, 6, 5, 8, 9} ⊄ IS(A)

# $O(N^2)$ LIS

- 전체 문제
  길이 N의 수열 안에서 LIS의 길이를 구하는 것

- 부분 문제
  길이 K(1 <= K < N )의 수열 안에서의 LIS 길이를 구하는 것

- DP 배열
  DP[N] = DP[1]~DP[N]을 포함하는 부분 수열 안에서, DP[N]을 마지막 원소로 갖는 LIS의 길이

# $O(N^2)$ LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8, 10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | - |

# O($N^2$) LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8 ,10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | - | - | - | - | - | - | - | - | - |

# O(N²) LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8 ,10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | - | - | - | - | - | - | - | - |

# $O(N^2)$ LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8 ,10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | - | - | - | - | - | - | - |

# $O(N^2)$ LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8 , 10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2 | - | - | - | - | - | - |

# $O(N^2)$ LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8 , 10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 3 | - | - | - | - | - |

# O($N^2$) LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8 , 10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 3 | 4 | - | - | - | - |

# O($N^2$) LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8 , 10, 3}

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 2 | 3 | 2 | 3 | 4 | 1 | - | - | - |

# $O(N^2)$ LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8, 10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 3 | 4 | 1 | 5 | - | - |

LIS

# O($N^2$) LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8, 10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 3 | 4 | 1 | 5 | 6 | - |

# O($N^2$) LIS

- DP 배열
  DP[N] = A[1]~A[N]을 포함하는 부분 수열 안에서, A[N]을 마지막 원소로 갖는 LIS의 길이

  A = {2, 6, 9, 4, 5, 7, 1, 8, 10, 3}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 3 | 4 | 1 | 5 | 6 | 2 |

# O($N^2$) LIS

- 이중 for 문을 사용해서 LIS를 구현한다.

- 바깥 for 문은 DP[1]~DP[N]을 구한다.
  안쪽 for 문은 DP[1]~DP[K-1]까지 보면서
  A[K]보다 작은 원소를 끝값으로 가지는 LIS 중 최대 길이를 찾는다.

- DP[K] = 찾은 최댓값 + 1 이 된다.

- DP[1]~DP[N] 중 최댓값이 수열 A의 LIS의 길이이다.

LIS

11053

# 가장 긴 증가하는 부분 수열

11053

# 가장 긴 증가하는 부분 수열

앞에서 설명한 LIS 알고리즘을 그대로 짠 다음 값을 출력하면 된다.

진짜 LIS 응용은 바로 다음 장에

LIS

2565

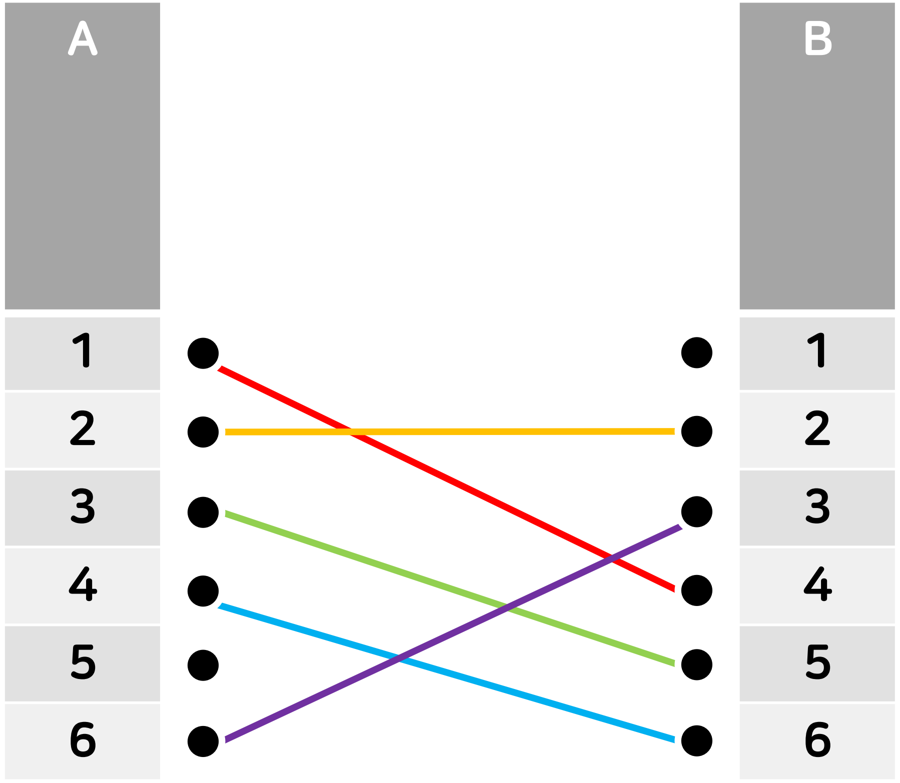
# 전깃줄

ALOHA
The algorithm club.

2565

# 전깃줄

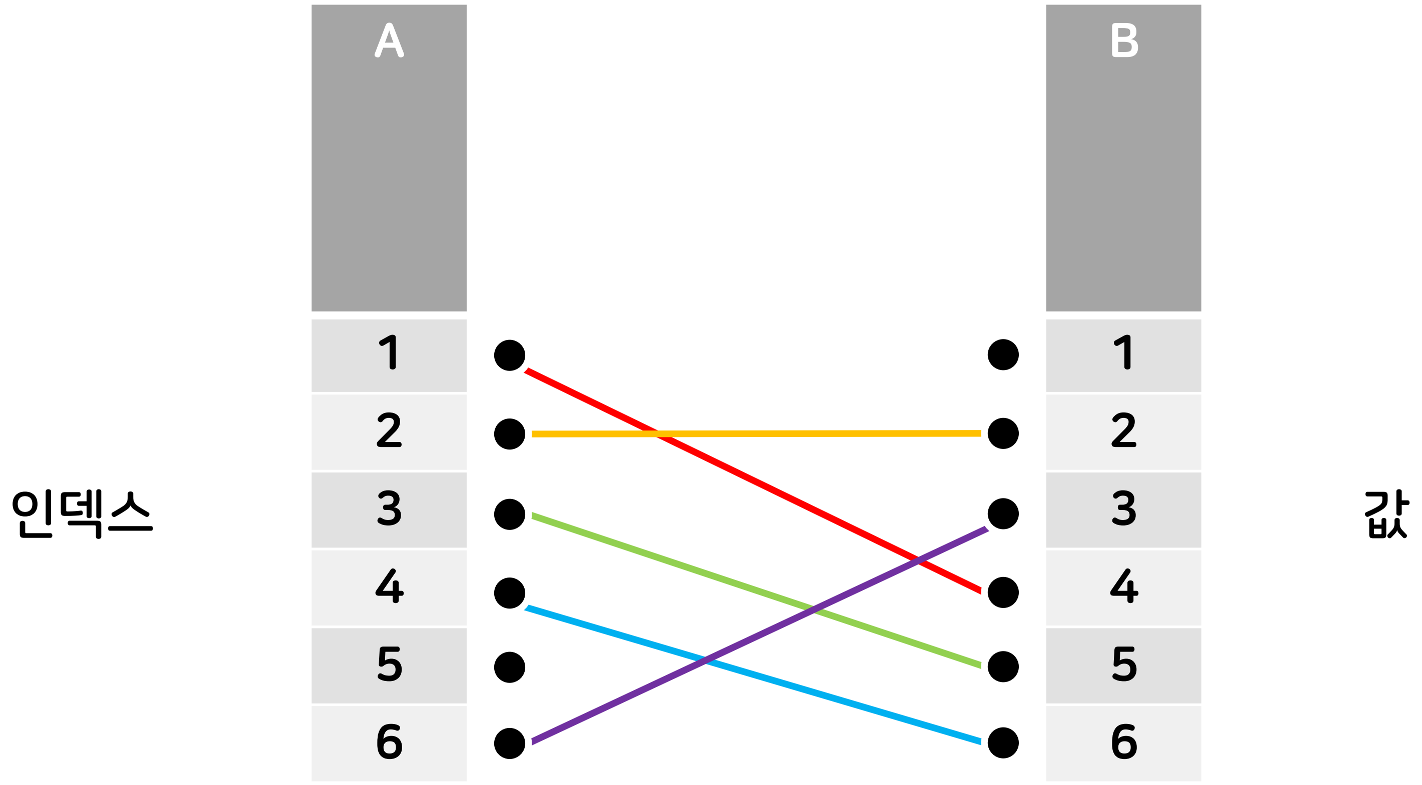
전깃줄이 꼬이지 않으려면 어떤 조건이 필요할까?



2565

# 전깃줄

전깃줄이 꼬이지 않으려면 어떤 조건이 필요할까?



2565

# 전깃줄

전깃줄이 꼬이지 않으려면 어떤 조건이 필요할까?

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 4 | 2 | 5 | 6 | 0 | 3 |

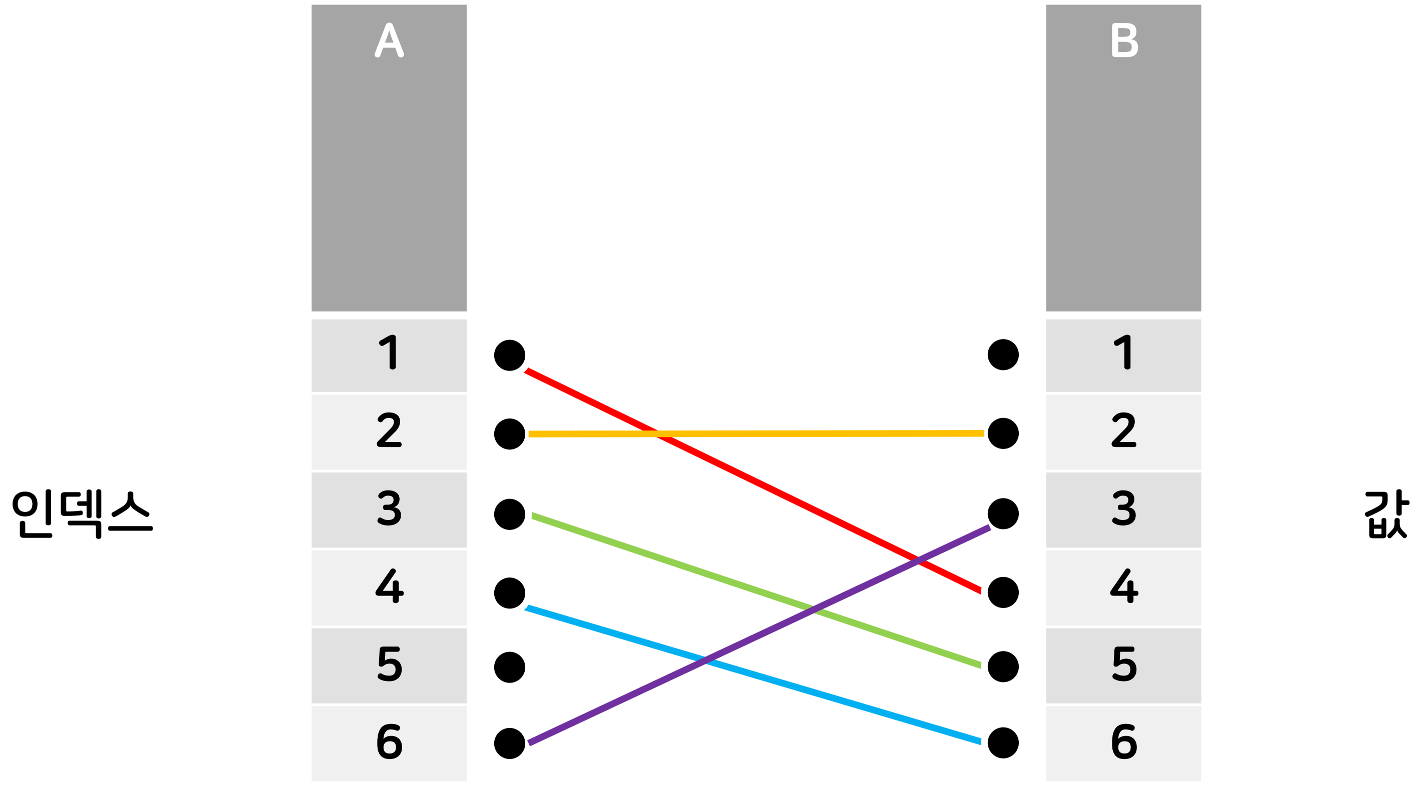
| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 0 | 2 |

LIS의 길이 = 3

2565

# 전깃줄

전깃줄이 꼬이지 않으려면 어떤 조건이 필요할까?



2565

# 전깃줄

A 전봇대의 위치를 인덱스로 잡고,
전깃줄로 연결된 B 전봇대의 위치를 값으로 한 다음,
배열 안에서의 LIS를 구하면,
겹치지 않고 놓을 수 있는 가장 많은 전깃줄의 수가 된다.

여기서 없앤 전깃줄의 개수를 구하려면,
(원래 전깃줄의 개수) - (LIS의 길이)를 구하면 된다.

"수고하셨어요!!!"

"나도 수고했어…….."