25-2 이니로
알고리즘 멘토링

멘토 - 김수성

Traveling SCCC President / 28119

백준 28119 / https://www.acmicpc.net/problem/28119

대회를 개최하기 위해서는 N개의 건물에서 한 번씩 **차례대로** 회의를 진행해야 한다. 구체적으로, 회의를 진행해야 하는 건물의 순서 $A_1, A_2, \cdots, A_N$이 주어진다. i번째로 진행해야 하는 회의는 A_i번 건물에서 진행되며, 모든 A_i는 서로 다르다.

4차 산업혁명이 도래한 21세기에 매번 도로를 통해 이동하는 것은 비효율적이기 때문에, 찬솔이는 순간 이동 장치를 만들었다. 순간 이동 장치를 사용하면 지금까지 방문했던 건물 중 원하는 곳으로 순식간에 이동할 수 있다.

찬솔이는 S번 건물에서 출발해서 N개의 회의를 마친 다음 다시 S번 건물로 돌아와야 한다. 시간이 부족한 찬솔이를 위해 N개의 회의를 차례대로 모두 마치고 S번 건물로 돌아오는데 걸리는 최소 시간을 구해주자.

Traveling SCCC President / 28119

1 -> 2 -> 3 -> 4 순으로 방문하면 6의 비용이 들음

예제 입력 1 복사

```
4 6 1
1 2 1
2 3 2
3 4 3
1 4 4
2 4 5
1 3 2
1 2 3 4
```

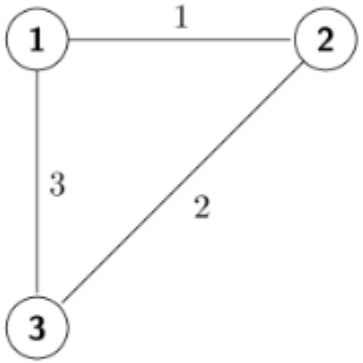
예제 출력 1 복사

```
6
```

Traveling SCCC President / 28119

차례대로 회의를 진행해야 함
하지만 미리 건물을 방문해 놓으면
회의를 진행할 순서가 되었을 때 순간 이동을 할 수 있음

예를 들어 숭실대학교 캠퍼스가 아래 그림과 같은 형태이고 1번 건물에서 출발하며, $1, 3, 2$번 건물에서 차례대로 회의를 진행해야 한다고 하자.



찬솔이는 먼저 1번 건물에서 회의를 진행한 뒤, 도로를 통해 2번 건물을 거쳐 3번 건물로 이동해서 회의를 진행한다. 이후 순간 이동 장치를 사용해 2번 건물로 이동해 회의를 진행한 뒤, 순간 이동 장치를 써서 1번 건물로 돌아올 수 있다. 이 과정에서 소요되는 총 시간은 3분이다.

Traveling SCCC President / 28119

우선 모든 정점을 방문해야 함

찬솔이는 S번 건물에서 출발해서 N개의 회의를 마친 다음 다시 S번 건물로 돌아와야 한다. 시간이 부족한 찬솔이를 위해 N개의 회의를 차례대로 모두 마치고 S번 건물로 돌아오는데 걸리는 최소 시간을 구해주자.

또한 한번 방문한 정점은 순간 이동(비용 0)으로 돌아올 수 있음
-> 그냥 방문 순서는 무시하고
모든 정점을 방문하는 최소 비용을 구해주면 됨

Traveling SCCC President / 28119

-> 그냥 방문 순서는 무시하고
모든 정점을 방문하는 최소 비용을 구해주면 됨

이는 MST로 구해 줄 수 있음

질문?

7주차 - DP

DP / 동적 계획법

DP

최적 부분 구조의 문제를 메모이제이션을 통해
중복 되는 하위 문제를 한번만 계산하는 최적화 기법

DP / 동적 계획법

최적 부분 구조

큰 문제를 작은 부분 문제의 답으로 구할 수 있는 문제

Ex) 피보나치 수열

$fib(N) = fib(N - 1) + fib(N - 2)$

Ex) 팩토리얼

$fac(N) = fac(N - 1) * N$

Ex) 이항 계수

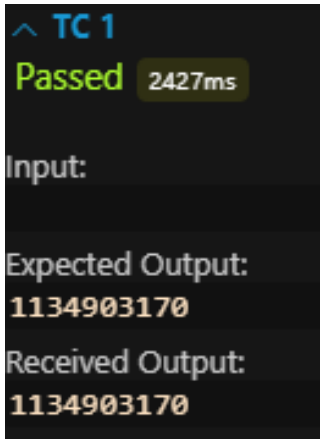
$comb(N, R) = comb(N - 1, R - 1) + comb(N, R - 1)$

DP / 동적 계획법

피보나치 수열

$$fib(N) = fib(N-1) + fib(N-2)$$

N이 45밖에 안되는데도 2.5초나 걸림
C++ 코드 -> 다른 언어는 더 걸림



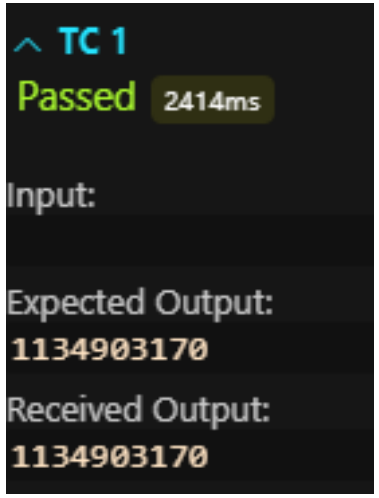

```cpp
#include <iostream>
using namespace std;

int fib(int num){
    if(num == 1 || num == 2) return 1;
    return fib(num - 1) + fib(num - 2);
}

int main(){
    cout << fib(45);
}
```

DP / 동적 계획법

피보나치 수열

이 정도로 시간이 오래 드는 이유?

Base Case 호출 횟수가 너무 많음



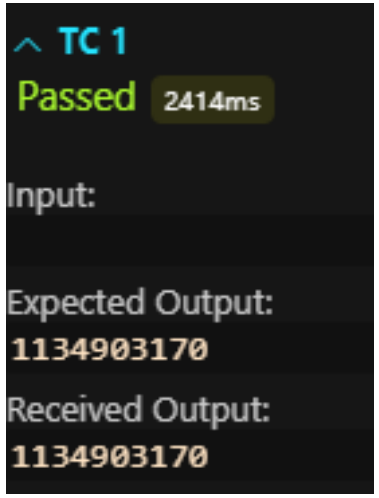

```cpp
#include <iostream>
using namespace std;
int cnt;

void fib(int num){
    if(num == 1 || num == 2){
        cnt++; return;
    }
    fib(num - 1); fib(num - 2);
}

int main(){
    fib(45);
    cout << cnt;
}
```

DP / 동적 계획법

피보나치 수열

fib(N) -> fib(N – 1), fib(N - 2)를 호출

시간 복잡도 O(2 ^ N)




```cpp
#include <iostream>
using namespace std;
int cnt;

void fib(int num){
    if(num == 1 || num == 2){
        cnt++; return;
    }
    fib(num - 1); fib(num - 2);
}

int main(){
    fib(45);
    cout << cnt;
}
```

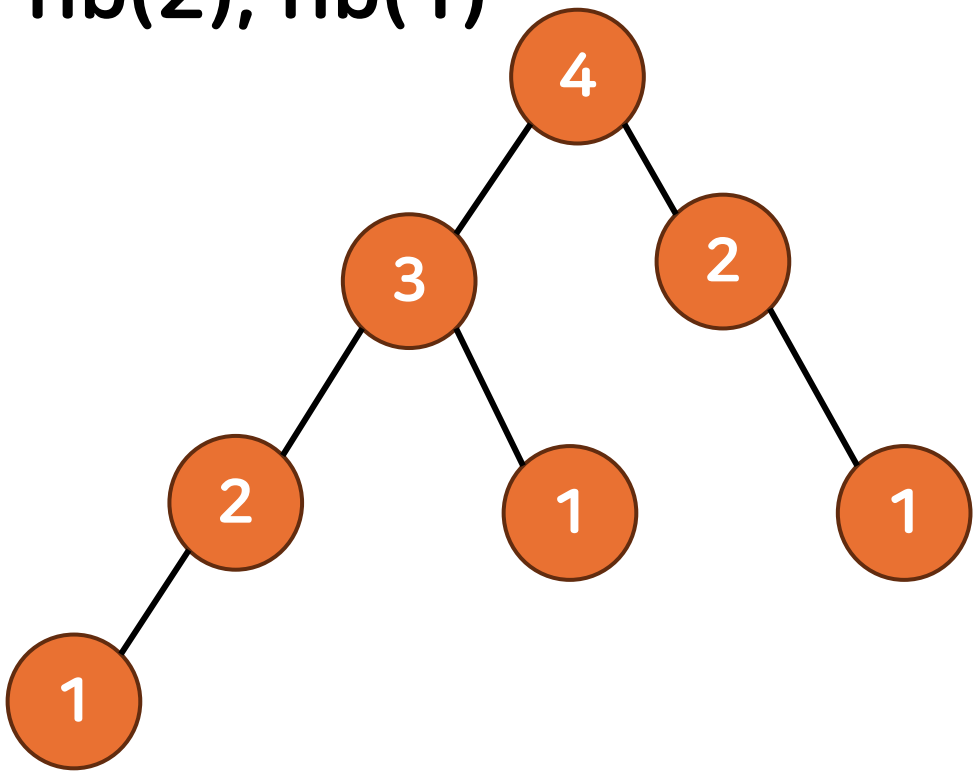
DP / 동적 계획법

피보나치 수열

Base Case 호출 횟수가 너무 많음

fib(4) -> fib(3), fib(2)

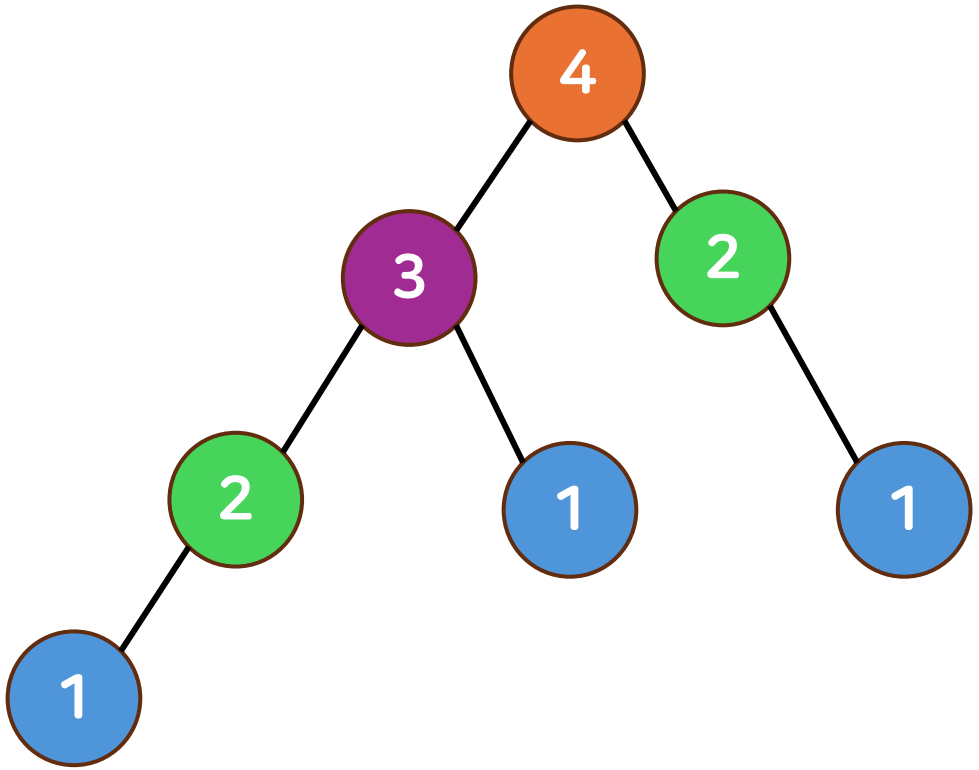
fib(3) -> fib(2), fib(1)



DP / 동적 계획법

피보나치 수열

중복되는 하위 문제가 생김

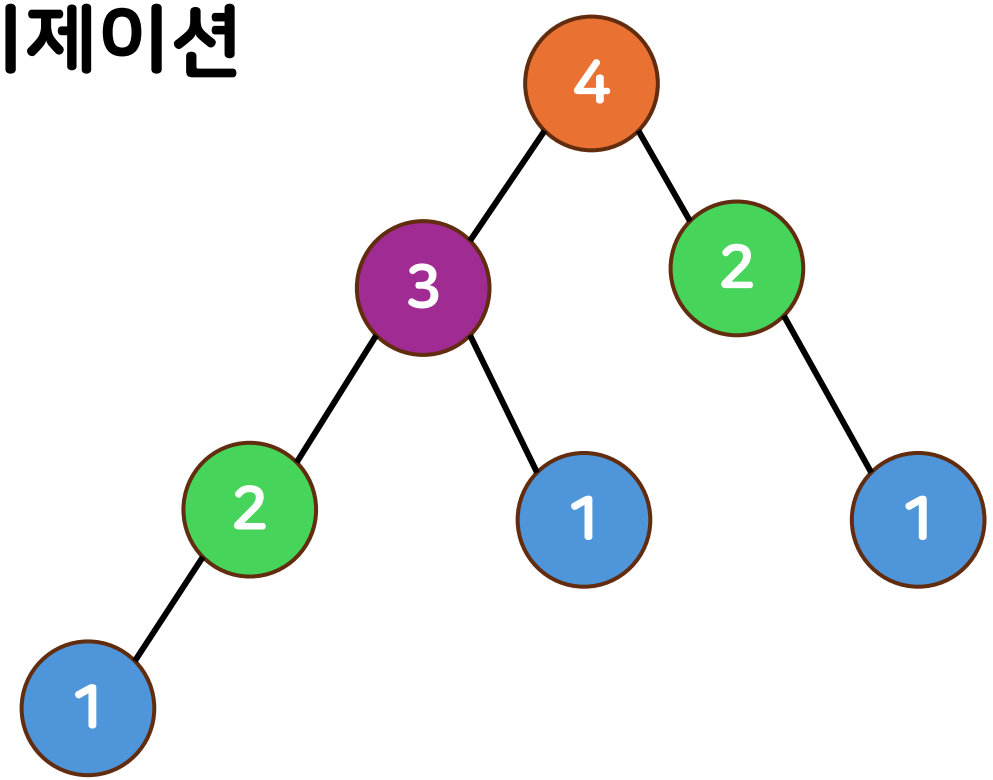


DP / 동적 계획법

피보나치 수열

중복되는 값들은 계산을 한번만 하자

-> 메모이제이션



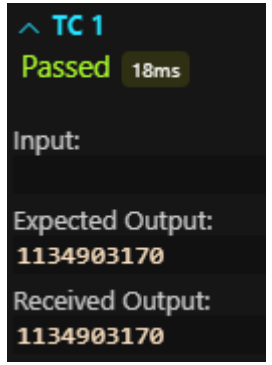
DP / 동적 계획법

메모이제이션

이미 계산한 값을 캐싱 함

값이 캐싱 되어 있으면 그 값을 그냥 가져다 쓰면 됨

2.5초 -> 0.02초



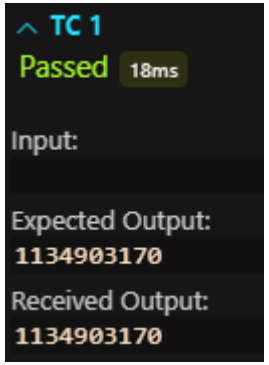

```cpp
#include <iostream>
using namespace std;
int cache[101];

int fib(int num){
    if(cache[num] != -1) return cache[num];
    if(num == 1 || num == 2) return cache[num] = 1;
    cache[num] = fib(num - 1) + fib(num - 2);

    return cache[num];
}

int main(){
    // 캐시 배열 초기화
    for(int i = 0;i <= 100;i++) cache[i] = -1;
    cout << fib(45);
}
```

DP / 동적 계획법

메모이제이션

각 함수는 한번만 방문, 계산 시간 O(1)

시간 복잡도 O(N)




```cpp
#include <iostream>
using namespace std;
int cache[101];

int fib(int num){
    if(cache[num] != -1) return cache[num];
    if(num == 1 || num == 2) return cache[num] = 1;
    cache[num] = fib(num - 1) + fib(num - 2);

    return cache[num];
}

int main(){
    // 캐시 배열 초기화
    for(int i = 0;i <= 100;i++) cache[i] = -1;
    cout << fib(45);
}
```

DP / 동적 계획법

DP 구현 방법

탑 다운

재귀로 구현 -> 메모리가 터질 수 있음

점화식이 직관적

바텀 업으로 짜는 게 최적화가 편한 문제가 있음

바텀 업

반복문으로 구현

점화식이 직관적이지 않음

Base Case가 애매한 경우 사용하기 힘듬

DP / 동적 계획법

탑 다운

```cpp
#include <iostream>
using namespace std;
int cache[101];

int fib(int num){
    if(cache[num] != -1) return cache[num];
    if(num == 1 || num == 2) return cache[num] = 1;
    cache[num] = fib(num - 1) + fib(num - 2);

    return cache[num];
}

int main(){
    // 캐시 배열 초기화
    for(int i = 0;i <= 100;i++) cache[i] = -1;
    cout << fib(45);
}
```

바텀 업

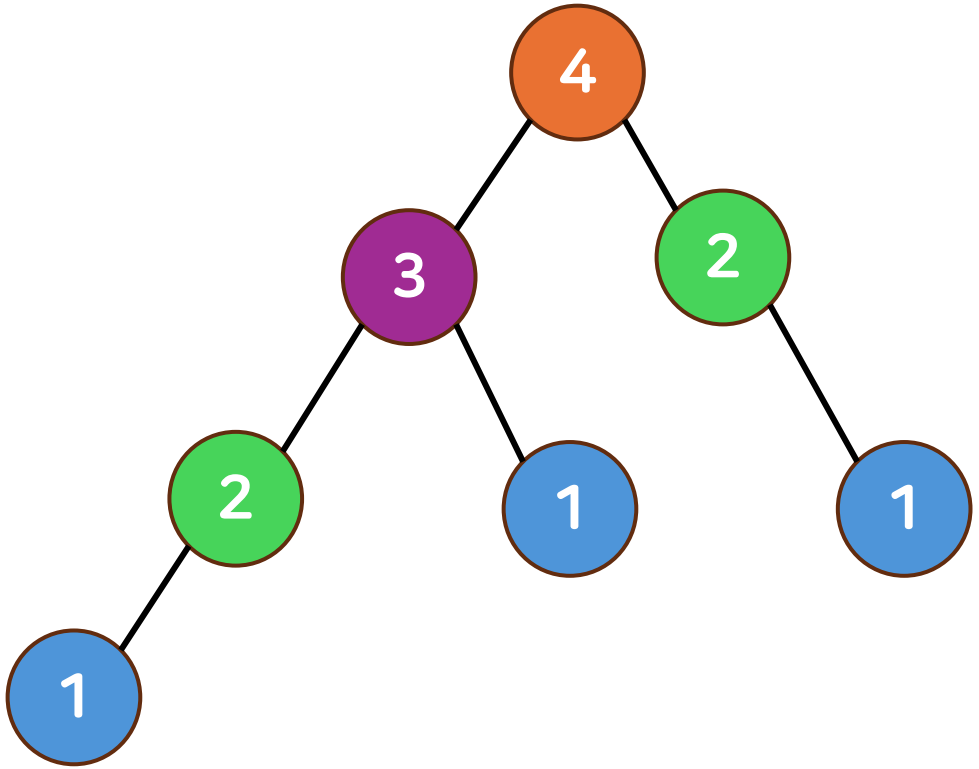
```cpp
#include <iostream>
using namespace std;
int cache[101];

int main(){
    // 캐시 배열 초기화
    for(int i = 0;i <= 100;i++) cache[i] = -1;
    cache[1] = cache[2] = 1;
    for(int i = 3;i <= 45;i++) cache[i] = cache[i - 1] + cache[i - 2];

    cout << cache[45];
}
```

DP / 동적 계획법

탑 다운

위의 식 부터 계산

재귀 함수가 알아서 다음에 필요한 함수를 호출 해 줌



DP / 동적 계획법

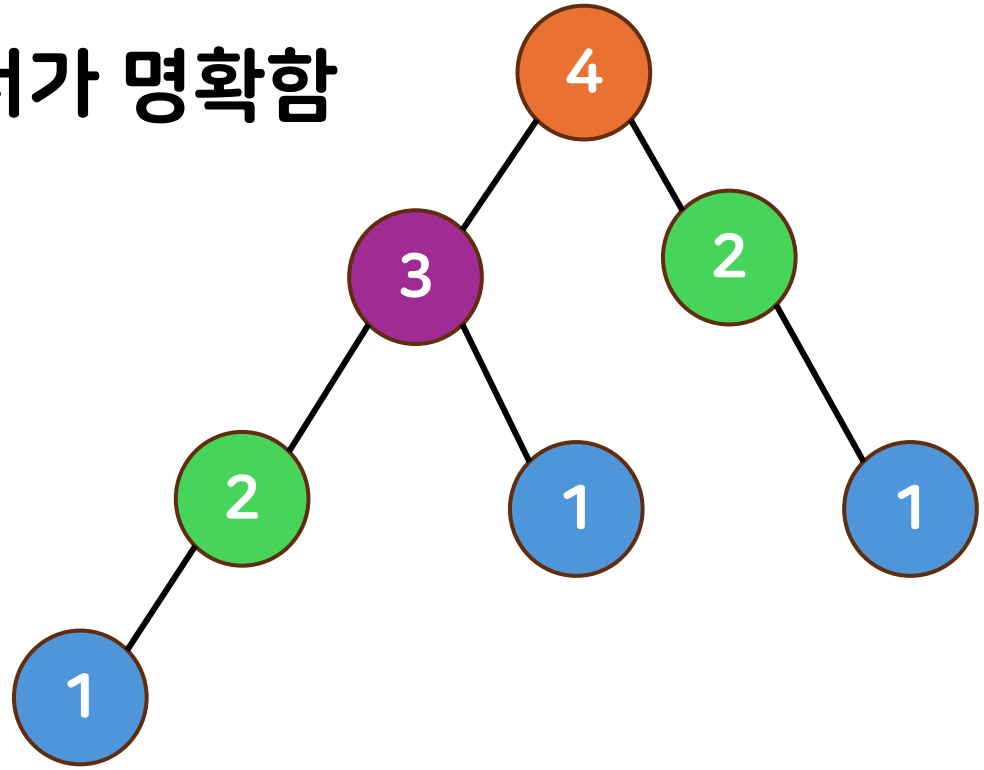
바텀 업

아래 식 부터 계산

점화 식 계산 순서를 알아야 함

피보나치 같이 1차원 DP는 순서가 명확함

-> 2차원, 3차원이 된다면?



질문?

2xn 타일링 / 11726

백준 11726 / https://www.acmicpc.net/problem/11726

문제

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

아래 그림은 2×5 크기의 직사각형을 채운 한 가지 방법의 예이다.

입력

첫째 줄에 n이 주어진다. (1 ≤ n ≤ 1,000)

출력

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

2xn 타일링 / 11726

1x2, 2x1 타일로 2xn 타일을 채우는 경우의 수 출력
경우의 수가 커질 수 있으므로 10007로 나눈 나머지 출력

예제 입력 1 복사

```
2
```

예제 출력 1 복사

```
2
```

예제 입력 2 복사

```
9
```

예제 출력 2 복사

```
55
```

2xn 타일링 / 11726

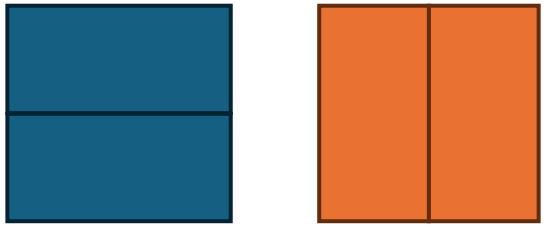
점화식부터 찾아봅시다

N = 1

1개

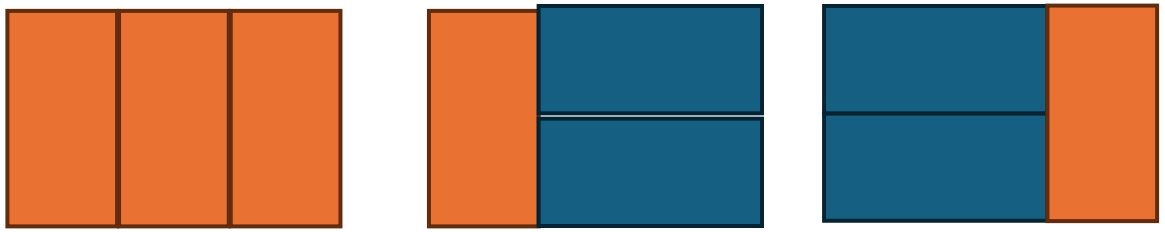
N = 2

2개



N = 3

3개

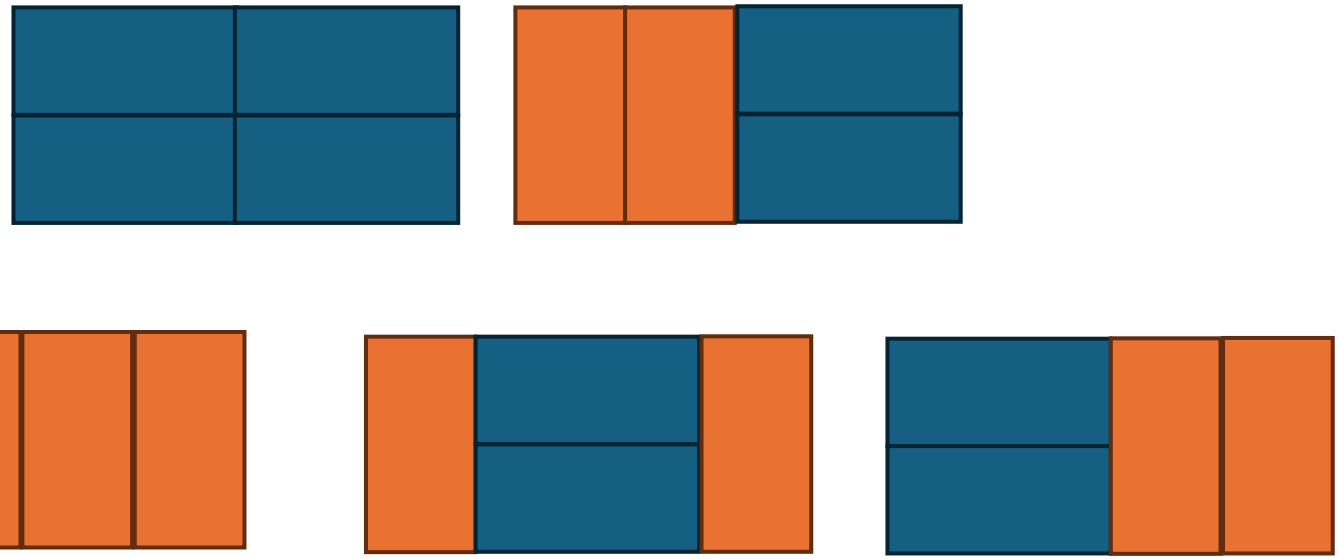


2xn 타일링 / 11726

점화식부터 찾아봅시다

N = 4

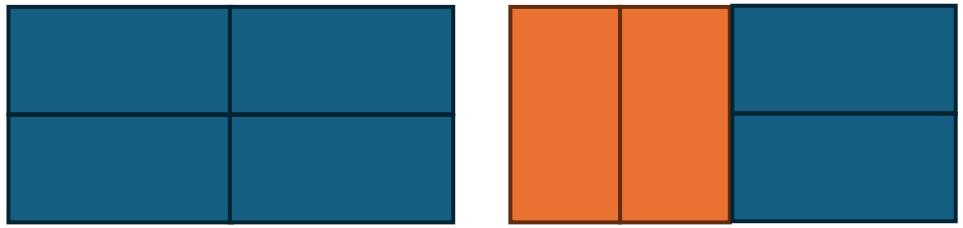
5개



2xn 타일링 / 11726

점화식부터 찾아봅시다

N = 4



N = 2

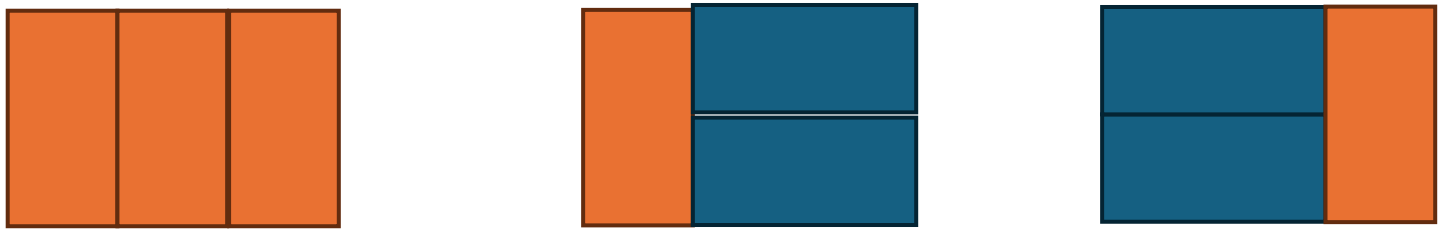
N – 2에서 누워 있는 타일 2개를 붙이면 N이 됨

2xn 타일링 / 11726

점화식부터 찾아봅시다

N = 4

N = 3



N – 1에서 서 있는 타일 1개를 붙이면 N이 됨

2xn 타일링 / 11726

점화식부터 찾아봅시다

N – 1에서 서 있는 타일 1개를 붙이면 N이 됨
N – 2에서 누워 있는 타일 2개를 붙이면 N이 됨

N = 1일 때는 경우의 수 1
N = 2 일 때는 경우의 수 2

점화식은 다음과 같음
f(N) = f(N – 1) + f(N – 2) (N >= 3)
f(1) = 1, f(2) = 2

2xn 타일링 / 11726

C++

```cpp
#include <iostream>
using namespace std;
using ll = long long;

const ll MAX = 1010;
const ll MOD = 10007;
ll n, dp[MAX];

ll solve(ll cur){
    ll& ret = dp[cur];
    // 현재 값이 이미 계산되었으면 반환
    if(ret != -1) return ret; ret = 0;
    // 점화식은 f(n - 1) + f(n - 2)
    ret += solve(cur - 1) + solve(cur - 2);

    return ret %= MOD;
}

int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n;
    // 캐시 배열 초기화
    for(int i = 0;i < MAX;i++) dp[i] = -1;
    // Base Case 설정
    dp[1] = 1; dp[2] = 2;

    cout << solve(n);

    return 0;
}
```

2xn 타일링 / 11726

Python

```python
import sys
input = sys.stdin.readline
sys.setrecursionlimit(10**6)

dp = [-1] * 1010
mod = 10007
dp[1] = 1
dp[2] = 2

def solve(cur):
    # 현재 값이 이미 계산되었으면 반환
    if dp[cur] != -1:
        return dp[cur]

    # 점화식은 f(n - 1) + f(n - 2)
    dp[cur] = (solve(cur - 1) + solve(cur - 2)) % mod
    return dp[cur]

n = int(input().rstrip())
print(solve(n))
```

질문?

계단 오르기 / 2579

백준 2579 / https://www.acmicpc.net/problem/2579

계단 오르는 데는 다음과 같은 규칙이 있다.

1. 계단은 한 번에 한 계단씩 또는 두 계단씩 오를 수 있다. 즉, 한 계단을 밟으면서 이어서 다음 계단이나, 다음 다음 계단으로 오를 수 있다.
2. 연속된 세 개의 계단을 모두 밟아서는 안 된다. 단, 시작점은 계단에 포함되지 않는다.
3. 마지막 도착 계단은 반드시 밟아야 한다.

따라서 첫 번째 계단을 밟고 이어 두 번째 계단이나, 세 번째 계단으로 오를 수 있다. 하지만, 첫 번째 계단을 밟고 이어 네 번째 계단으로 올라가거나, 첫 번째, 두 번째, 세 번째 계단을 연속해서 모두 밟을 수는 없다.

각 계단에 쓰여 있는 점수가 주어질 때 이 게임에서 얻을 수 있는 총 점수의 최댓값을 구하는 프로그램을 작성하시오.

입력

입력의 첫째 줄에 계단의 개수가 주어진다.

둘째 줄부터 한 줄에 하나씩 제일 아래에 놓인 계단부터 순서대로 각 계단에 쓰여 있는 점수가 주어진다. 계단의 개수는 300이하의 자연수이고, 계단에 쓰여 있는 점수는 10,000이하의 자연수이다.

출력

첫째 줄에 계단 오르기 게임에서 얻을 수 있는 총 점수의 최댓값을 출력한다.

계단 오르기 / 2579

각 칸을 밟으면 A[i]의 점수를 얻음
움직일 때는 1칸, 2칸 움직일 수 있고
연속한 3칸을 동시에 밟을 수 없음
점수를 최대화, 아래 예제는 1, 2, 4, 6 -> 10 + 20 + 25 + 20 = 75

예제 입력 1 복사

```
6
10
20
15
25
10
20
```

예제 출력 1 복사

```
75
```

계단 오르기 / 2579

DP식을 다음과 같이 정의 해 보자

f(i, j) = 현재 i번째 계단을 밟고 있고
j = 0이면 전의 계단을 밟지 않았고
j = 1이면 전의 계단을 밟음

Base Case
f(1, 0) = A[1]
f(2, 0) = A[1]
f(2, 1) = A[1] + A[2]

계단 오르기 / 2579

DP식을 다음과 같이 정의 해 보자

f(i, j) = 현재 i번째 계단을 밟고 있고
j = 0이면 전의 계단을 밟지 않았고
j = 1이면 전의 계단을 밟음

j = 0이면 전의 계단을 밟지 않음 -> 2칸 전의 계단을 밟아야 함
f(i, 0) = max(f(i - 2, 0) + f(i - 2, 1))+ A[i]

j = 1이면 전의 계단을 밟음
f(i, 1) = f(i - 1, 0) + A[i]

계단 오르기 / 2579

C++

```cpp
#include <iostream>
using namespace std;
using ll = long long;

const ll MAX = 303;
const ll INF = 1e9;
ll n, a[MAX], dp[MAX][2];

int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    cin >> n;
    for(int i = 1;i <= n;i++) cin >> a[i];

    // Base Case
    dp[1][0] = a[1];
    dp[2][0] = a[2];
    dp[2][1] = a[1] + a[2];

    for(int i = 3;i <= n;i++){
        // 전의 계단을 밟았으면 전의 계단에서 점화식 전이
        dp[i][1] = dp[i - 1][0] + a[i];

        // 전의 계단을 밟지 않았으면 2칸 전의 계단에서 점화식 전이
        dp[i][0] = max(dp[i - 2][0], dp[i - 2][1]) + a[i];
    }

    // 둘 중 큰 값이 정답
    cout << max(dp[n][0], dp[n][1]);

    return 0;
}
```

계단 오르기 / 2579

Python

```python
import sys
input = sys.stdin.readline

n = int(input().rstrip())
a = []
a.append(0)
for _ in range(n):
    num = int(input().rstrip())
    a.append(num)

dp = [[0, 0] for _ in range(303)]

# Base Case
dp[1][0] = a[1]
if(n >= 2):
    dp[2][0] = a[2]
    dp[2][1] = a[1] + a[2]

for i in range(3, n + 1):
    # 전의 계단을 밟았으면 전의 계단에서 점화식 전이
    dp[i][1] = dp[i - 1][0] + a[i]

    # 전의 계단을 밟지 않았으면 2칸 전의 계단에서 점화식 전이
    dp[i][0] = max(dp[i - 2][0], dp[i - 2][1]) + a[i]

# 둘 중 큰 값이 정답
print(max(dp[n][0], dp[n][1]))
```

질문?

1, 2, 3 더하기 3 / 15988

백준 15988 / https://www.acmicpc.net/problem/15988

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 7가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다.

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

정수 n이 주어졌을 때, n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 정수 n이 주어진다. n은 양수이며 1,000,000보다 작거나 같다.

출력

각 테스트 케이스마다, n을 1, 2, 3의 합으로 나타내는 방법의 수를 1,000,000,009로 나눈 나머지를 출력한다.

1, 2, 3 더하기 3 / 15988

T개의 테스트 케이스가 주어짐
정수 N을 1, 2, 3을 더해서 만들 수 있는 경우의 수 출력

EX) 4

3 + 1
2 + 2
2 + 1 + 1
1 + 3
1 + 1 + 2
1 + 2 + 1
1 + 1 + 1 + 1

1, 2, 3 더하기 3 / 15988

f(n) = n을 1, 2, 3의 덧셈으로 만들 수 있는 경우의 수

f(1) = 1 / 1

f(2) = 2 / 1 + 1, 2

f(3) = 4 / 1 + 1 + 1, 2 + 1, 1 + 2, 3

f(4) = 7 / 3 + 1, 2 + 1 + 1, 1 + 2 + 1, 1 + 1 + 1 + 1
/ 1 + 1 + 2, 2 + 2
/ 1 + 3

1, 2, 3 더하기 3 / 15988

f(n) = n을 1, 2, 3의 덧셈으로 만들 수 있는 경우의 수

f(1) = 1 / 1

f(2) = 2 / 1 + 1, 2

f(3) = 4 / 1 + 1 + 1, 2 + 1, 1 + 2, 3

f(4) = 7 / 1 + 1 + 1 + 1, 2 + 1 + 1, 1 + 2 + 1, 3 + 1
/ 1 + 1 + 2, 2 + 2
/ 1 + 3

1, 2, 3 더하기 3 / 15988

f(n) = n을 1, 2, 3의 덧셈으로 만들 수 있는 경우의 수

f(N – 1)의 식들에 1을 더하면 f(N)의 식이 됨
f(N – 2)의 식들에 2을 더하면 f(N)의 식이 됨
f(N – 2)의 식들에 3을 더하면 f(N)의 식이 됨

f(N) = f(N – 1) + f(N – 2) + F(N – 3) (N >= 4)
f(1) = 1, f(2) = 2
f(3) = 4, f(4) = 7

1, 2, 3 더하기 3 / 15988

f(N) = f(N – 1) + f(N – 2) + F(N – 3) (N >= 4)
f(1) = 1, f(2) = 2
f(3) = 4, f(4) = 7

1, 2, 3, 4를 Base Case로 두지 않고
f(0) = 1
f(N) = f(N – 1) + f(N – 2) + F(N – 3) (N > 0)
= 0 (N < 0)
로 두고 풀어도 됨

1, 2, 3 더하기 3 / 15988

C++

```cpp
#include <iostream>
using namespace std;
using ll = long long;

const ll MAX = 1010101;
const ll MOD = 1e9 + 9;
ll n, dp[MAX];

ll solve(ll cur){
    if(cur < 0) return 0;
    ll& ret = dp[cur];
    if(ret != -1) return ret; ret = 0;
    // 점화식은 f(n - 1) + f(n - 2) + f(n - 3)
    ret += solve(cur - 1) + solve(cur - 2) + solve(cur - 3);
    return ret %= MOD;
}

void run(){
    cin >> n;
    cout << solve(n) << "\n";
}

int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    ll t; cin >> t;
    for(int i = 0;i < MAX;i++) dp[i] = -1;
    // Base Case
    dp[0] = 1;
    while(t--) run();

    return 0;
}
```

1, 2, 3 더하기 3 / 15988

Python

```python
import sys
input = sys.stdin.readline

mod = 10 ** 9 + 9
dp = [-1] * 1010101

# Base Case
dp[0] = 1
dp[1] = 1
dp[2] = 2
dp[3] = 4

for i in range(3, 1010101):
    # 점화식은 f(n - 1) + f(n - 2) + f(n - 3)
    dp[i] = (dp[i - 1] + dp[i - 2] + dp[i - 3]) % mod

t = int(input().rstrip())
for _ in range(t):
    n = int(input().rstrip())
    print(dp[n])
```

질문?

기본 과제

2xn 타일링 – https://www.acmicpc.net/problem/11726

계단 오르기 - https://www.acmicpc.net/problem/2579

1, 2, 3 더하기 3 – https://www.acmicpc.net/problem/15988

돌 게임 4 - https://www.acmicpc.net/problem/9658

가장 긴 증가하는 부분 수열 - https://www.acmicpc.net/problem/11053

쉬운 계단 수 – https://www.acmicpc.net/problem/10844

고생하셨습니다