25-1 이니로
알고리즘 멘토링

멘토 - 김수성

이분 그래프 / 1707

백준 1707 / https://www.acmicpc.net/problem/1707

문제

그래프의 정점의 집합을 둘로 분할하여, 각 집합에 속한 정점끼리는 서로 인접하지 않도록 분할할 수 있을 때, 그러한 그래프를 특별히 이분 그래프 (Bipartite Graph) 라 부른다.

그래프가 입력으로 주어졌을 때, 이 그래프가 이분 그래프인지 아닌지 판별하는 프로그램을 작성하시오.

입력

입력은 여러 개의 테스트 케이스로 구성되어 있는데, 첫째 줄에 테스트 케이스의 개수 K가 주어진다. 각 테스트 케이스의 첫째 줄에는 그래프의 정점의 개수 V와 간선의 개수 E가 빈 칸을 사이에 두고 순서대로 주어진다. 각 정점에는 1부터 V까지 차례로 번호가 붙어 있다. 이어서 둘째 줄부터 E개의 줄에 걸쳐 간선에 대한 정보가 주어지는데, 각 줄에 인접한 두 정점의 번호 u, v (u ≠ v)가 빈 칸을 사이에 두고 주어진다.

출력

K개의 줄에 걸쳐 입력으로 주어진 그래프가 이분 그래프이면 YES, 아니면 NO를 순서대로 출력한다.

이분 그래프 / 1707

T개의 테스트 케이스가 주어지고
각 테스트 케이스에서 그래프가 주어짐
이 그래프가 이분 그래프인지 판단하는 문제

예제 입력 1 복사

```
2
3 2
1 3
2 3
4 4
1 2
2 3
3 4
4 2
```

예제 출력 1 복사

```
YES
NO
```

이분 그래프 / 1707

각 정점과 인접한 다음 정점들은
현재 정점과 번호가 달라야 함

DFS를 돌면서 현재 정점이 0이면 다음 정점은 1
현재 정점이 1이면 다음 정점은 0으로 배치하면 됨

이분 그래프 / 1707

DFS를 돌면서 현재 정점이 0이면 다음 정점은 1
현재 정점이 1이면 다음 정점은 0으로 배치하면 됨

만약 다음 정점에 번호를 배치할 때
이미 번호가 배치되어 있고 현재 배치할 번호와 다르면
이분 그래프가 아님

이분 그래프 / 1707

또한 항상 그래프의 연결 요소가 1개라는 보장이 없음
반복문을 돌면서 현재 정점이 방문 하지 않은 상태면
그 정점에 대해서 DFS를 돌아야 함

이분 그래프 / 1707

C++

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

    ll t; cin >> t;
    while(t--) solve();

    return 0;
}
```

```cpp
void solve(){
    cin >> n >> m;

    result = 1;
    for(int i = 1;i <= n;i++) {
        // 간선 리스트와 방문 배열 초기화
        a[i].clear(); v[i] = -1;
    }

    while(m--){
        // 간선 추가
        ll s, e; cin >> s >> e;
        add(s, e);
    }

    for(int i = 1;i <= n;i++){
        // 방문을 했으면 넘어 감
        if(v[i] != -1) continue;
        dfs(i, 0);
    }

    cout << (result ? "YES" : "NO") << "\n";
}
```

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

const ll MAX = 20101;
vector <ll> a[MAX];
ll v[MAX], result, n, m;

void dfs(ll cur, ll num){
    // 현재 정점의 번호 설정
    v[cur] = num;

    for(auto& nxt : a[cur]){
        // 다음 정점을 방문 했으면
        if(v[nxt] != -1){
            // 배치할 번호와 이미 배치 된 번호가 다르면 NO
            if(v[nxt] != num ^ 1) result = 0;
            continue;
        }

        // 번호를 바꿔서 탐색
        dfs(nxt, num ^ 1);
    }
}

void add(int s, int e){
    // 양방향 간선 추가
    a[s].push_back(e);
    a[e].push_back(s);
}
```

이분 그래프 / 1707

Python

```python
t = int(input().rstrip())
for _ in range(t):
    # 방문 배열과 간선 리스트 초기화
    v = [-1] * 20101
    a = [[] for _ in range(20101)]
    result = 1

    n, m = list(map(int, input().rstrip().split()))
    # 간선 추가
    for _ in range(m):
        s, e = list(map(int, input().rstrip().split()))
        add(s, e)

    for i in range(1, n + 1):
        # 방문을 했으면 넘어 감
        if v[i] != -1:
            continue
        dfs(i, 0)

    print("YES" if result == 1 else "NO")
```

```python
def dfs(cur, num):
    global result
    # 현재 정점의 번호 설정
    v[cur] = num

    for nxt in a[cur]:
        # 다음 정점을 방문 했으면
        if v[nxt] != -1:
            # 배치할 번호와 이미 배치 된 번호가 다르면 NO
            if v[nxt] != num ^ 1:
                result = 0
            continue
        # 번호를 바꿔서 탐색
        dfs(nxt, num ^ 1)
```

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

result = 0

def add(s, e):
    # 양방향 간선 추가
    a[s].append(e)
    a[e].append(s)
```

질문?

숨바꼭질 2 / 12851

백준 12851 / https://www.acmicpc.net/problem/12851

문제

수빈이는 동생과 숨바꼭질을 하고 있다. 수빈이는 현재 점 N(0 ≤ N ≤ 100,000)에 있고, 동생은 점 K(0 ≤ K ≤ 100,000)에 있다. 수빈이는 걷거나 순간이동을 할 수 있다. 만약, 수빈이의 위치가 X일 때 걷는다면 1초 후에 X-1 또는 X+1로 이동하게 된다. 순간이동을 하는 경우에는 1초 후에 2*X의 위치로 이동하게 된다.

수빈이와 동생의 위치가 주어졌을 때, 수빈이가 동생을 찾을 수 있는 가장 빠른 시간이 몇 초 후인지 그리고, 가장 빠른 시간으로 찾는 방법이 몇 가지 인지 구하는 프로그램을 작성하시오.

입력

첫 번째 줄에 수빈이가 있는 위치 N과 동생이 있는 위치 K가 주어진다. N과 K는 정수이다.

출력

첫째 줄에 수빈이가 동생을 찾는 가장 빠른 시간을 출력한다.

둘째 줄에는 가장 빠른 시간으로 수빈이가 동생을 찾는 방법의 수를 출력한다.

숨바꼭질 2 / 12851

정점 N에서 M까지 가는 최단거리와 경우의 수 출력
한 번의 연산마다 +1 -1 *2 연산을 할 수 있음

예제 입력 1 복사

```
5 17
```

예제 출력 1 복사

```
4
2
```

숨바꼭질 2 / 12851

아래 예제에서는
5 -> 10 -> 9 -> 18 -> 17
5 -> 4 -> 8 -> 16 -> 17

예제 입력 1 복사

```
5 17
```

예제 출력 1 복사

```
4
2
```

숨바꼭질 2 / 12851

각 연산이 +1 -1 *2이므로
i번 째 정점이 i + 1, i – 1, 2 * i와 연결된 그래프

최단거리를 구하는 건 쉬움
-> BFS 사용

경우의 수를 구하는 법?

숨바꼭질 2 / 12851

경우의 수를 구하는 법?
일단 처음 시작점은 경우의 수가 1

그리고 다음에 방문 하는 정점을 처음 방문하거나
경로의 길이가 같으면 현재 상황이 최단거리
-> 현재 경우의 수를 더하면 됨

숨바꼭질 2 / 12851

C++

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

const ll MAX = 101010;
ll n, m, d[MAX], cnt[MAX];
queue <ll> q;
```

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

    cin >> n >> m;
    // 방문 배열 초기화
    for(int i = 0;i < MAX;i++) d[i] = -1;
    bfs(n);

    cout << d[m] << "\n" << cnt[m];

    return 0;
}
```

```cpp
void bfs(ll st){
    q.push(st);
    d[st] = 0; cnt[st] = 1;

    while(!q.empty()){
        ll cur = q.front(); q.pop();
        for(auto& nxt : {cur + 1, cur - 1, 2 * cur}){
            // 다음 정점이 점 밖에 있으면 건너 뜀
            if(nxt < 0 || nxt >= MAX) continue;
            // 다음 정점이 방문하지 않은 정점이면
            if(d[nxt] == -1){
                // 큐에 삽입하고 거리 배열 +1
                q.push(nxt);
                d[nxt] = d[cur] + 1;
            }

            // 다음 정점과의 거리가 최단거리이면
            // 현재 정점까지 가는 경우의 수를 더함
            if(d[nxt] == d[cur] + 1) cnt[nxt] += cnt[cur];
        }
    }
}
```

숨바꼭질 2 / 12851

Python

```python
import sys
from collections import deque
input = sys.stdin.readline

n, m = list(map(int, input().rstrip().split()))
d = [-1] * 101010
cnt = [0] * 101010

# 방문 배열 초기화
for i in range(101010):
    d[i] = -1
```

```python
bfs(n)
print(d[m])
print(cnt[m])
```

```python
q = deque()
def bfs(st):
    q.append(st)
    d[st] = 0
    cnt[st] = 1

    while len(q) != 0:
        cur = q.popleft()
        for nxt in (cur + 1, cur - 1, 2 * cur):
            # 다음 정점이 점 밖에 있으면 건너 뜀
            if nxt < 0 or nxt >= 101010:
                continue

            # 다음 정점이 방문하지 않은 정점이면
            if d[nxt] == -1:
                # 큐에 삽입하고 거리 배열 +1
                q.append(nxt)
                d[nxt] = d[cur] + 1

            # 다음 정점과의 거리가 최단거리이면
            # 현재 정점까지 가는 경우의 수를 더함
            if d[nxt] == d[cur] + 1:
                cnt[nxt] += cnt[cur]
```

질문?

5주차 - 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

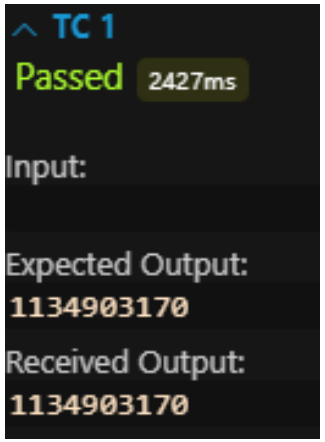
$comb(N, R) = comb(N - 1, R - 1) + comb(N, R - 1)$

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



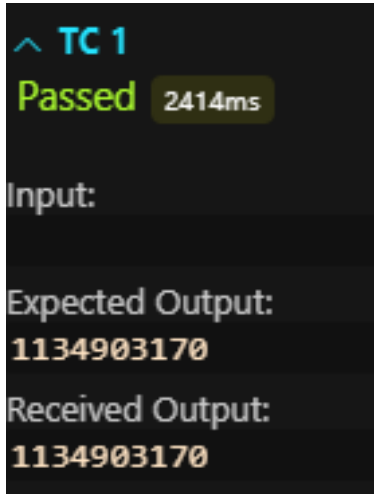

```cpp
#include <iostream>
using namespace std;

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



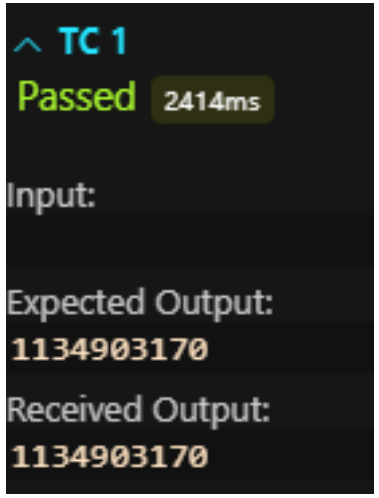

```cpp
#include <iostream>
using namespace std;
int cnt;

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;
}
```

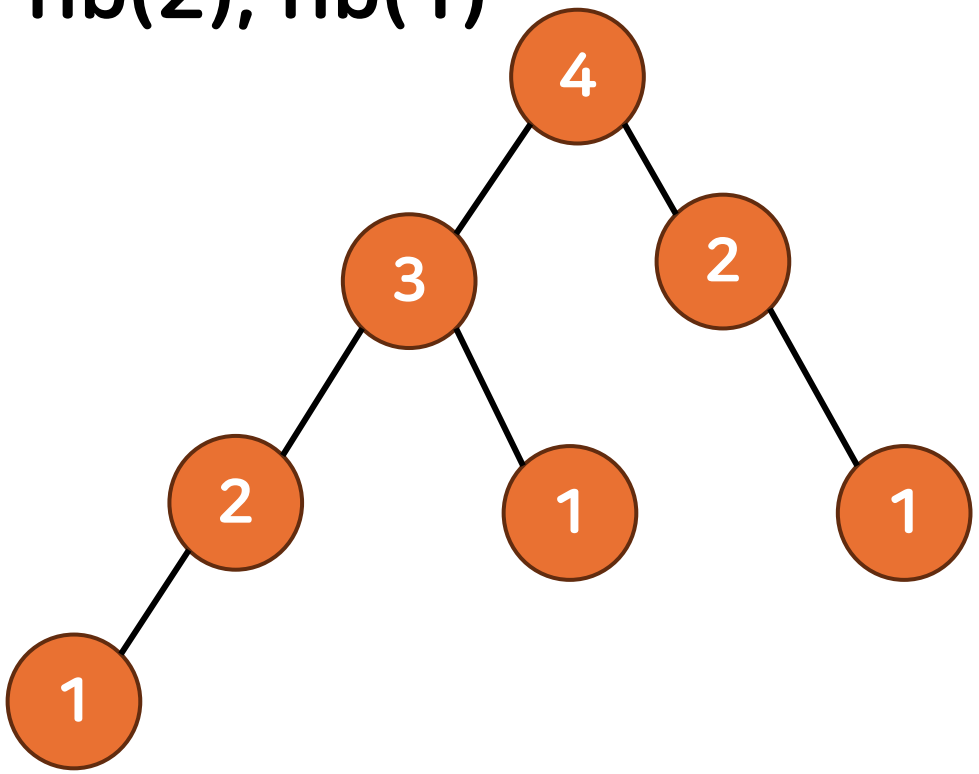
DP / 동적 계획법

피보나치 수열

Base Case 호출 횟수가 너무 많음

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

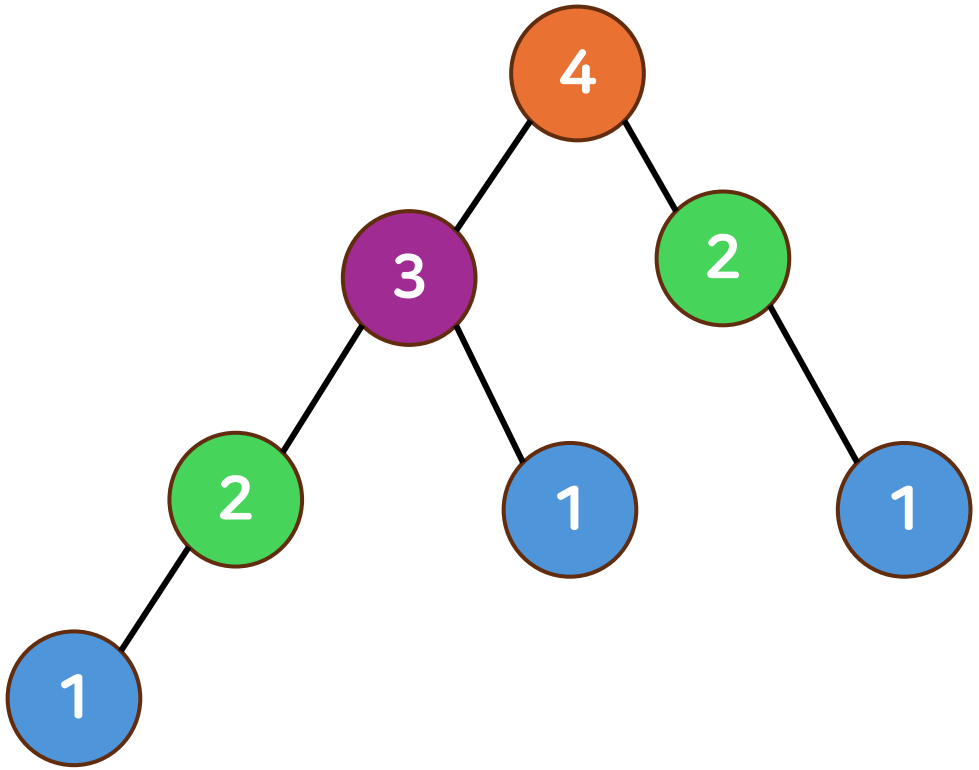
fib(3) -> fib(2), fib(1)



DP / 동적 계획법

피보나치 수열

중복되는 하위 문제가 생김

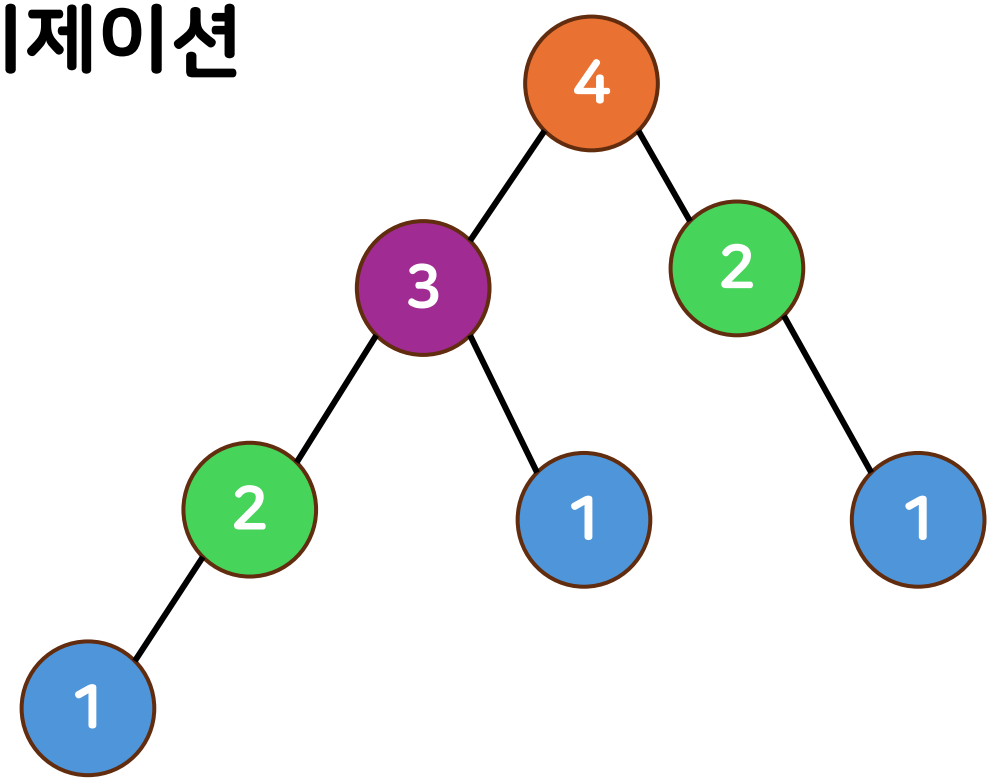


DP / 동적 계획법

피보나치 수열

중복되는 값들은 계산을 한번만 하자

-> 메모이제이션



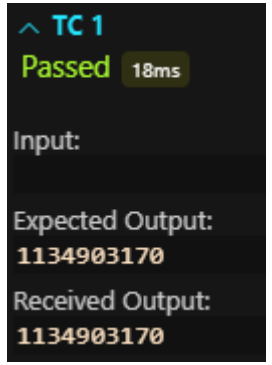
DP / 동적 계획법

메모이제이션

이미 계산한 값을 캐싱 함

값이 캐싱 되어 있으면 그 값을 그냥 가져다 쓰면 됨

2.5초 -> 0.02초



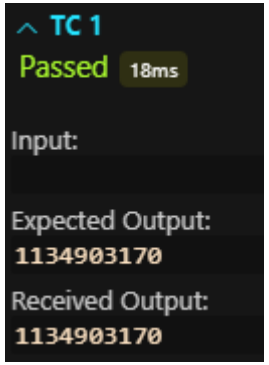

```cpp
#include <iostream>
using namespace std;
int cache[101];

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

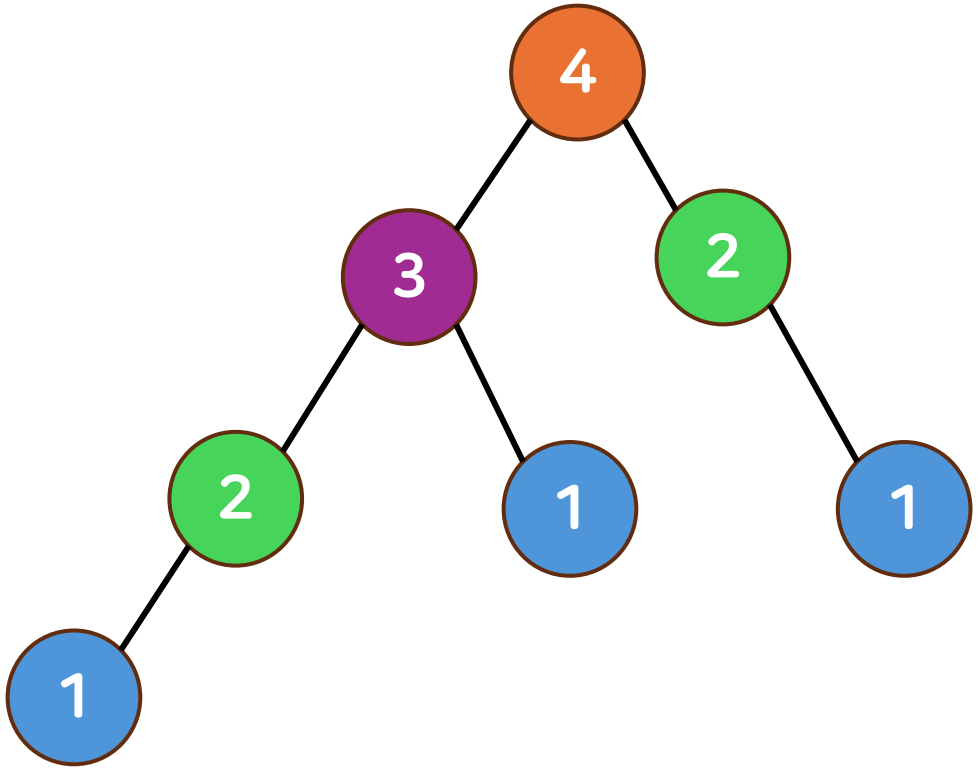
```cpp
#include <iostream>
using namespace std;
int cache[101];

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

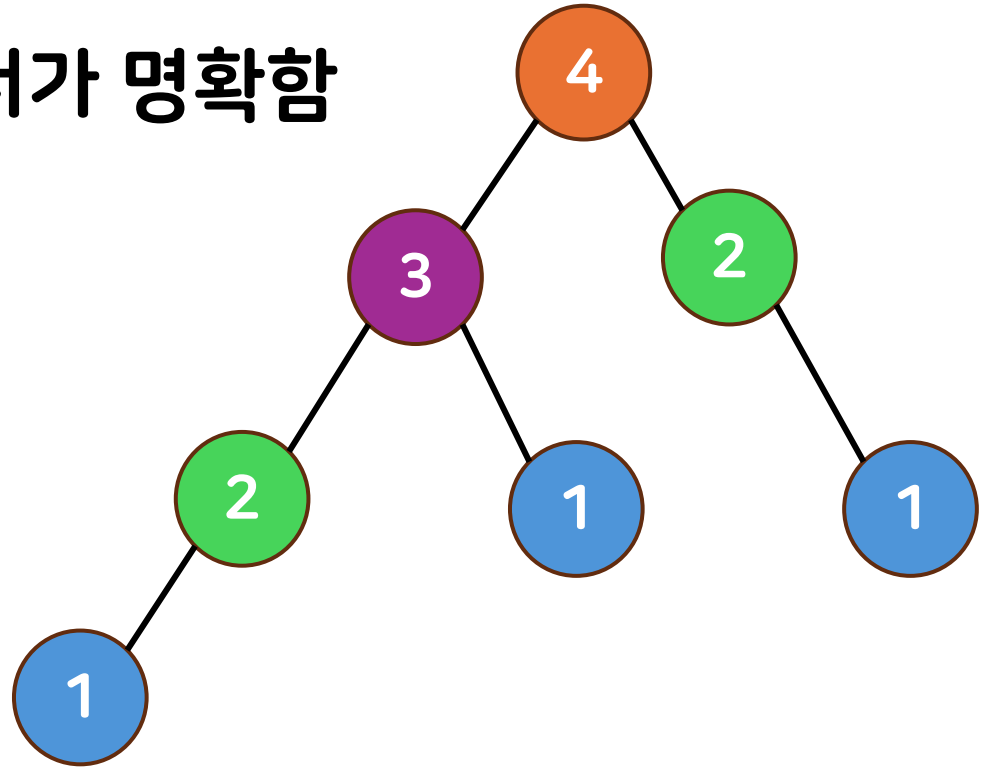
바텀 업

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

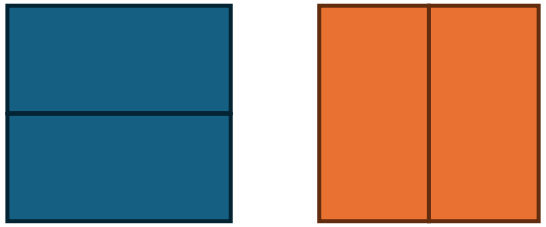
점화식부터 찾아봅시다

N = 1

1개

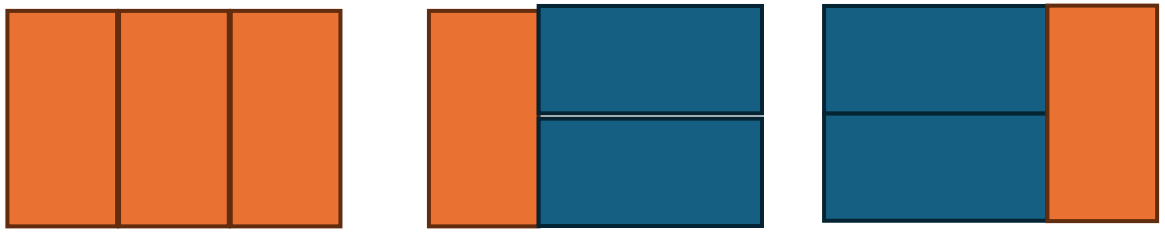
N = 2

2개



N = 3

3개

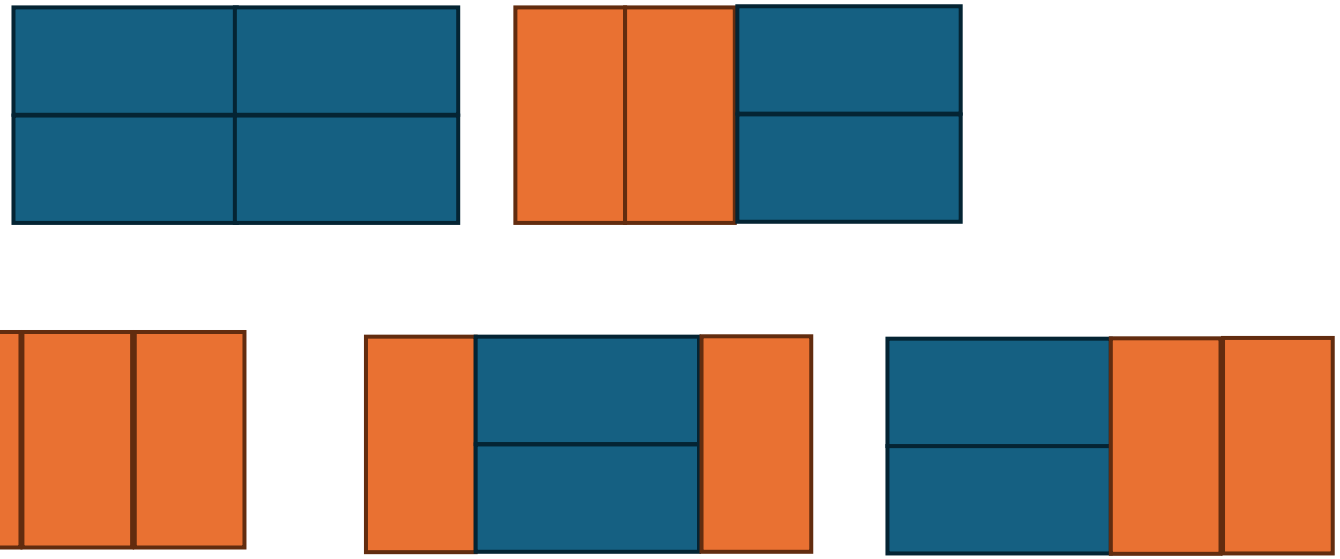


2xn 타일링 / 11726

점화식부터 찾아봅시다

N = 4

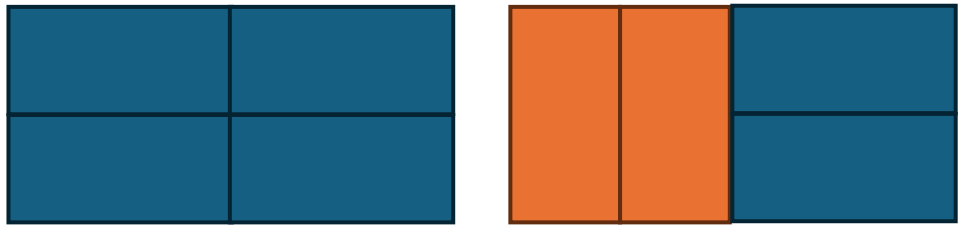
5개



2xn 타일링 / 11726

점화식부터 찾아봅시다

N = 4



N = 2

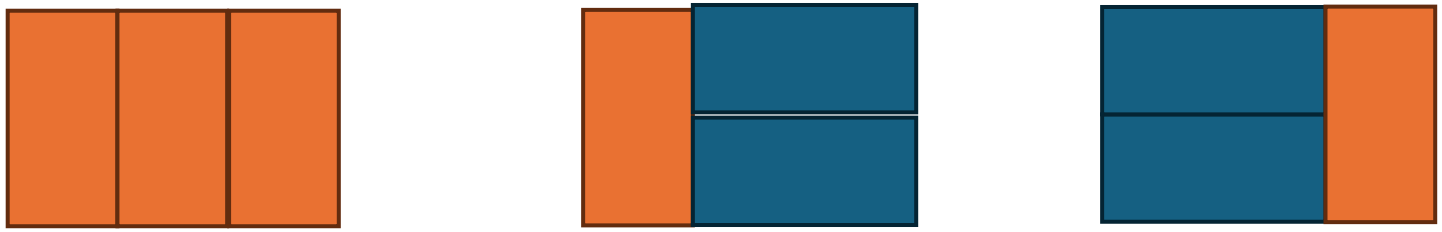
N – 2에서 누워 있는 타일 2개를 붙이면 N이 됨

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

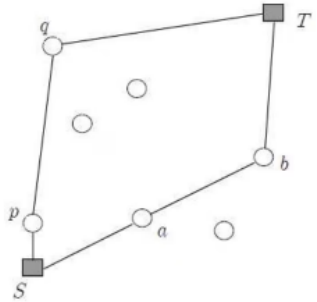
고생하셨습니다

경비행기 / 2585

백준 2585 / https://www.acmicpc.net/problem/2585

문제

경비행기 독수리호가 출발지 S에서 목적지 T로 가능한 빠른 속도로 안전하게 이동하고자 한다. 이때, 경비행기의 연료통의 크기를 정하는 것이 중요한 문제가 된다. 큰 연료통을 장착하면 중간에 내려서 급유를 받는 횟수가 적은 장점이 있지만 연료통의 무게로 인하여 속도가 느려지고, 안정성에도 문제가 있을 수 있다. 한편 작은 연료통을 장착하면 비행기의 속도가 빨라지는 장점이 있지만 중간에 내려서 급유를 받아야 하는 횟수가 많아지는 단점이 있다. 문제는 중간에 내려서 급유를 받는 횟수가 k이하 일 때 연료통의 최소용량을 구하는 것이다. 아래 예를 보자.



위 그림은 S, T와 7개의 중간 비행장의 위치를 나타내고 있는 그림이다. 위 예제에서 중간급유를 위한 착륙 허용 최대횟수 k=2라면 S-a-b-T로 가는 항로가 S-p-q-T로 가는 항로보다 연료통이 작게 된다. 왜냐하면, S-p-q-T항로에서 q-T의 길이가 매우 길어서 이 구간을 위해서 상당히 큰 연료통이 필요하기 때문이다. 문제는 이와 같이 중간에 최대 K번 내려서 갈 수 있을 때 최소 연료통의 크기가 얼마인지를 결정하여 출력하면 된다. 참고사항은 다음과 같다.

a. 모든 비행기는 두 지점 사이를 반드시 직선으로 날아간다. 거리의 단위는 ㎞이고 연료의 단위는 ℓ(리터)이다. 1ℓ당 비행거리는 10㎞이고 연료주입은 ℓ단위로 한다.
b. 두 위치간의 거리는 평면상의 거리이다. 예를 들면, 두 점 g=(2,1)와 h=(37,43)간의 거리 d(g,h)는 $\sqrt{(2-37)^2+(1-43)^2}$ = 54.671... 이고 50<d(g,h) ≤ 60이므로 필요한 연료는 6ℓ가 된다.
c. 출발지 S의 좌표는 항상 (0,0)이고 목적지 T의 좌표는 (10000,10000)으로 모든 입력 데이터에서 고정되어 있다.
d. 출발지와 목적지를 제외한 비행장의 수 n은 3 ≤ n ≤ 1000이고 그 좌표 값 (x,y)의 범위는 0<x,y<10000의 정수이다. 그리고 최대 허용 중간급유 횟수 k는 0 ≤ k ≤ 1000이다.

경비행기 / 2585

정점을 M번 까지 방문 가능할 때
(0, 0) -> (10000, 10000) 까지 가는데 드는 최소 연료

입력

첫 줄에는 n과 k가 하나의 공백을 사이에 두고 주어진다. 그 다음 n개의 줄에는 각 비행장 (급유지)의 정수좌표가 x y 형식으로 주어진다.

출력

S에서 T까지 k번 이하로 중간급유 하여 갈 수 있는 항로에서의 최소 연료통 용량에 해당되는 정수를 출력한다.

예제 입력 1 복사

```
10 1
10 1000
20 1000
30 1000
40 1000
5000 5000
1000 60
1000 70
1000 80
1000 90
7000 7000
```

예제 출력 1 복사

```
708
```

경비행기 / 2585

한 정점에서 다른 정점까지 가는데
정점 사이의 직선 거리 만큼 비용이 들음

또한 연료통만 된다면 한 정점에서
모든 정점으로 이동 할 수 있음

문제에서는 M번 이하로 중간 급유해서 마지막 정점까지 가는데
필요한 최소 연료통 용량을 구해야 함

경비행기 / 2585

문제에서는 M번 이하로 중간 급유해서 마지막 정점까지 가는데 필요한 최소 연료통 용량을 구해야 함

정점을 M번 이하로 방문할 때 마지막 정점까지 가는데 필요한 최소 연료통 용량을 구하면 됨

-> 어차피 한 정점에서 모든 정점으로 이동 가능 함

-> 두 번에 걸쳐 가는 것보다 한번에 가는 게 이득

경비행기 / 2585

정점을 M번 이하로 방문할 때 마지막 정점까지 가는데
필요한 최소 연료통 용량을 구하면 됨

어디서 많이 본 것 같은 글
-> 최적화 문제
결정 문제로 바꿔보자

경비행기 / 2585

최적화 문제

정점을 M번 이하로 방문할 때 마지막 정점까지
가는데 필요한 최소 연료통

결정 문제

연료통의 용량이 K일 때 마지막 정점까지 가는데
정점을 M번 이하로 방문 할 수 있는가

경비행기 / 2585

결정 문제

연료통의 용량이 K일 때 마지막 정점까지 가는데
정점을 M번 이하로 방문 할 수 있는가

최단거리가 M 이하인가

-> BFS

연료통의 용량이 K

-> 간선의 길이가 K 초과면

-> 그 간선은 이용 할 수 없음

경비행기 / 2585

C++

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

const ll MAX = 1010;
const ll INF = 1e9;
ll n, m, d[MAX];
pair <ll, ll> a[MAX];

ll dist(pair <ll, ll> a, pair <ll, ll> b){
    ll dx = a.first - b.first;
    ll dy = a.second - b.second;
    return dx * dx + dy * dy;
}
```

```cpp
ll minimization(){
    ll lo = 0, hi = 1e5;
    while(lo < hi){
        ll mid = (lo + hi) / 2;
        if(decision(mid)) hi = mid;
        else lo = mid + 1;
    }

    return lo;
}

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

    cin >> n >> m;
    for(int i = 1;i <= n;i++) cin >> a[i].first >> a[i].second;
    a[n + 1] = { 10000, 10000 };

    cout << minimization();

    return 0;
}
```

```cpp
bool decision(ll num){
    num *= 10;
    queue <ll> q;
    for(int i = 1;i <= n + 1;i++) d[i] = INF;
    d[0] = 0; q.push(0);

    while(!q.empty()){
        ll cur = q.front(); q.pop();
        for(auto nxt = 1;nxt <= n + 1;nxt++){
            // 현재 정점과 다음 정점의 거리가 num보다 크면
            // 다음 정점으로 갈 수 없음
            if(dist(a[nxt], a[cur]) > num * num) continue;

            // 이미 방문 했으면 건너 뜀
            if(d[nxt] != INF) continue;

            q.push(nxt);
            d[nxt] = d[cur] + 1;
        }
    }

    // 끝점과의 최단거리가 M 이하이면 도착 가능
    return d[n + 1] - 1 <= m;
}
```

경비행기 / 2585

Python

```python
import sys
from collections import deque
input = sys.stdin.readline

inf = 1000000000
n, m = list(map(int, input().rstrip().split()))

a = []
a.append((0, 0))
for _ in range(n):
    x, y = list(map(int, input().rstrip().split()))
    a.append((x, y))
a.append((10000, 10000))

def dist(a, b):
    dx = a[0] - b[0]
    dy = a[1] - b[1]
    return dx * dx + dy * dy
```

```python
def minimization():
    lo = 0
    hi = 10000
    while lo < hi:
        mid = (lo + hi) // 2
        if decision(mid):
            hi = mid
        else:
            lo = mid + 1
    return lo

print(minimization())
```

```python
def decision(num):
    num *= 10
    q = deque()
    d = [inf] * (n + 2)
    d[0] = 0
    q.append(0)

    while len(q):
        cur = q.popleft()
        for nxt in range(1, n + 2):
            # 현재 정점과 다음 정점의 거리가 num보다 크면
            # 다음 정점으로 갈 수 없음
            if dist(a[nxt], a[cur]) > num * num:
                continue

            # 이미 방문 했으면 건너 뜀
            if d[nxt] != inf:
                continue

            q.append(nxt)
            d[nxt] = d[cur] + 1
    # 끝점과의 최단거리가 M 이하이면 도착 가능
    return d[n + 1] - 1 <= m
```

질문?

트리의 높이와 너비 / 2250

백준 2250 / https://www.acmicpc.net/problem/2250

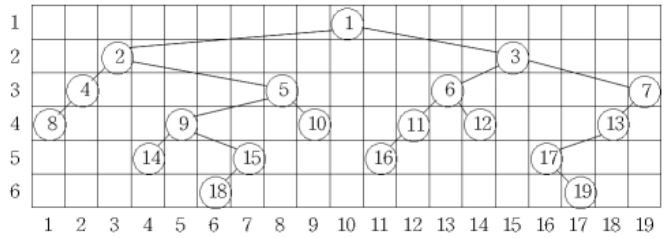
문제

이진트리를 다음의 규칙에 따라 행과 열에 번호가 붙어있는 격자 모양의 틀 속에 그리려고 한다. 이때 다음의 규칙에 따라 그리려고 한다.

1. 이진트리에서 같은 레벨(level)에 있는 노드는 같은 행에 위치한다.
2. 한 열에는 한 노드만 존재한다.
3. 임의의 노드의 왼쪽 부트리(left subtree)에 있는 노드들은 해당 노드보다 왼쪽의 열에 위치하고, 오른쪽 부트리(right subtree)에 있는 노드들은 해당 노드보다 오른쪽의 열에 위치한다.
4. 노드가 배치된 가장 왼쪽 열과 오른쪽 열 사이엔 아무 노드도 없이 비어있는 열은 없다.

이와 같은 규칙에 따라 이진트리를 그릴 때 각 레벨의 너비는 그 레벨에 할당된 노드 중 가장 오른쪽에 위치한 노드의 열 번호에서 가장 왼쪽에 위치한 노드의 열 번호를 뺀 값 더하기 1로 정의한다. 트리의 레벨은 가장 위쪽에 있는 루트 노드가 1이고 아래로 1씩 증가한다.

아래 그림은 어떤 이진트리를 위의 규칙에 따라 그려 본 것이다. 첫 번째 레벨의 너비는 1, 두 번째 레벨의 너비는 13, 3번째, 4번째 레벨의 너비는 각각 18이고, 5번째 레벨의 너비는 13이며, 그리고 6번째 레벨의 너비는 12이다.

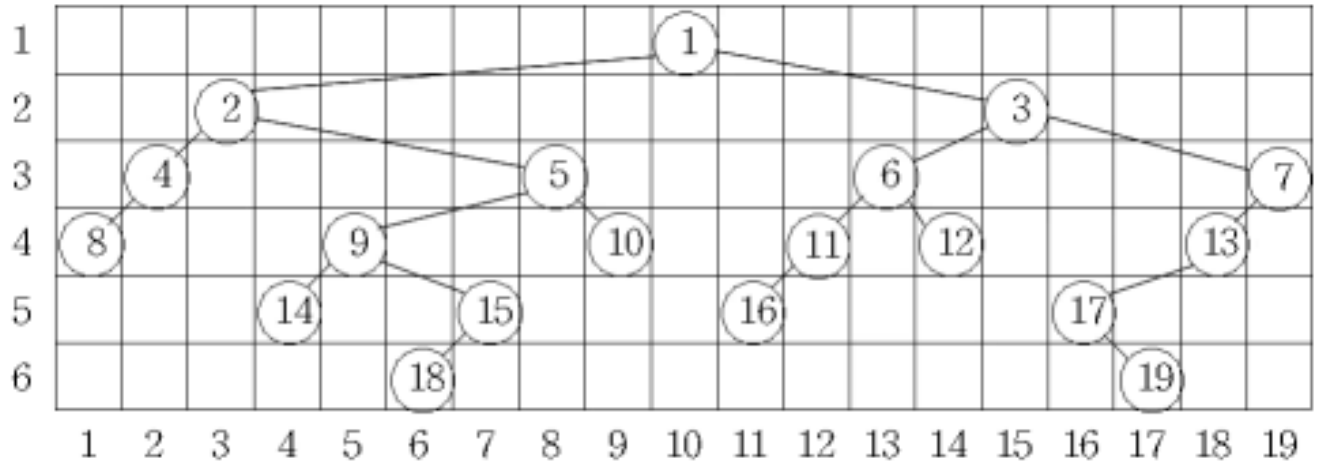


우리는 주어진 이진트리를 위의 규칙에 따라 그릴 때에 너비가 가장 넓은 레벨과 그 레벨의 너비를 계산하려고 한다. 위의 그림의 예에서 너비가 가장 넓은 레벨은 3번째와 4번째로 그 너비는 18이다. 너비가 가장 넓은 레벨이 두 개 이상 있을 때는 번호가 작은 레벨을 답으로 한다. 그러므로 이 예에 대한 답은 레벨은 3이고, 너비는 18이다.

임의의 이진트리가 입력으로 주어질 때 너비가 가장 넓은 레벨과 그 레벨의 너비를 출력하는 프로그램을 작성하시오

트리의 높이와 너비 / 2250

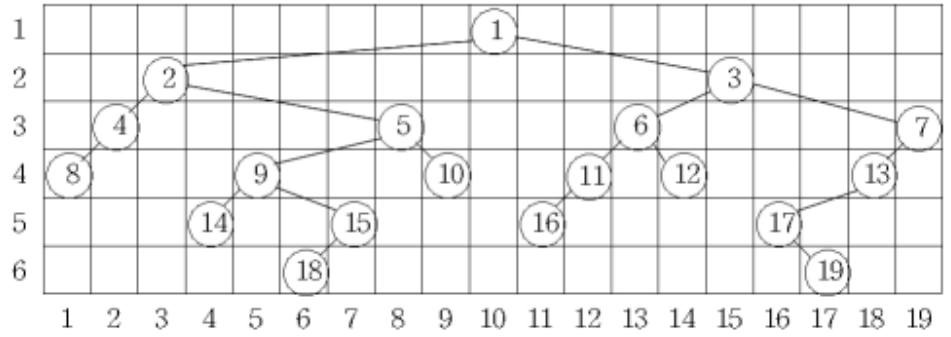
정점을 조건에 따라 배치했을 때
너비가 가장 넓은 레벨과 그 레벨의 너비 출력
아래 그림에선 레벨 3이 너비 18(2 ~ 19)로 가장 넓음



트리의 높이와 너비 / 2250

높이는 전의 정점의 높이에 +1을 해주면 구할 수 있음
아래의 조건에 따르면 한 정점의 왼쪽 서브 트리는 항상 왼쪽
오른쪽 서브 트리는 항상 오른쪽에 있음

1. 이진트리에서 같은 레벨(level)에 있는 노드는 같은 행에 위치한다.
2. 한 열에는 한 노드만 존재한다.
3. 임의의 노드의 왼쪽 부트리(left subtree)에 있는 노드들은 해당 노드보다 왼쪽의 열에 위치하고, 오른쪽 부트리(right subtree)에 있는 노드들은 해당 노드보다 오른쪽의 열에 위치한다.
4. 노드가 배치된 가장 왼쪽 열과 오른쪽 열 사이엔 아무 노드도 없이 비어있는 열은 없다.

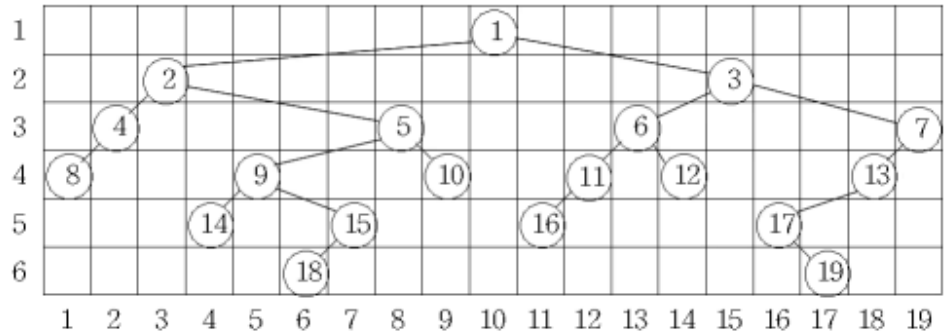


트리의 높이와 너비 / 2250

아래의 조건에 따르면 한 정점의 왼쪽 서브 트리는 항상 왼쪽
오른쪽 서브 트리는 항상 오른쪽에 있음

왼쪽에서부터 정점을 할당하면 왼쪽 자식이 없을 때까지
DFS로 방문을 하고 왼쪽 자식이 없으면 그때 숫자를 할당해주면 됨

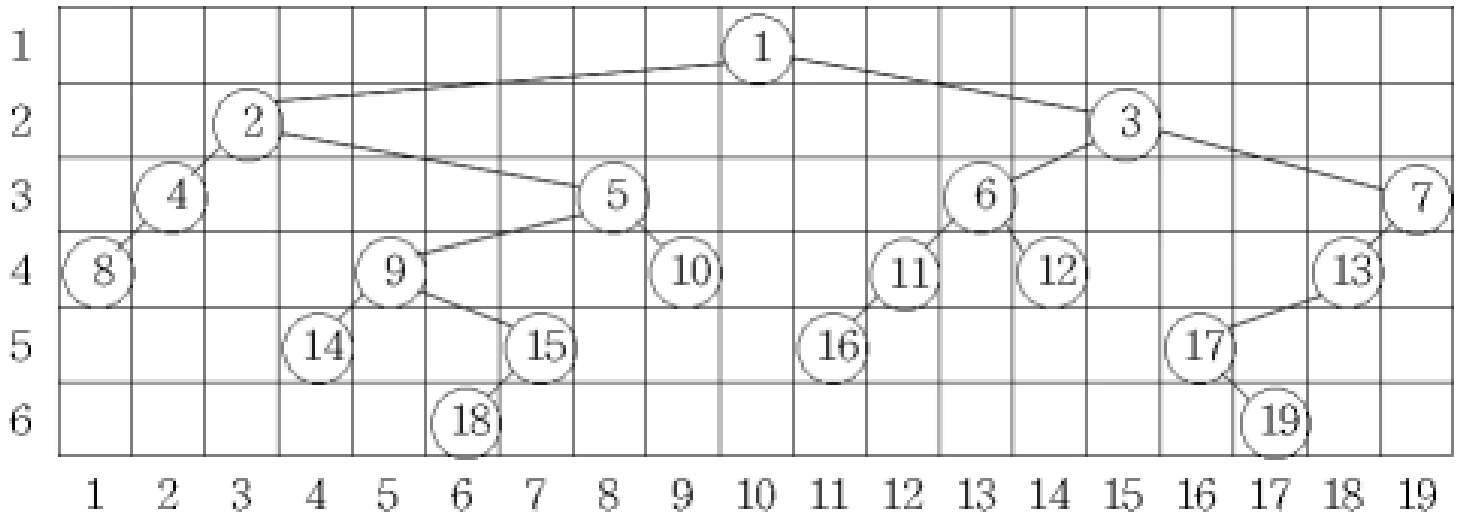
그러면 항상 왼쪽 자식이 현재 정점보다 먼저 숫자를 할당 받음



트리의 높이와 너비 / 2250

모든 왼쪽 자식에 숫자 할당이 끝났으면 자신에게 숫자를 할당
그리고 오른쪽 자식에게 이 과정을 반복 하면 됨

-> DFS로 구현



트리의 높이와 너비 / 2250

C++

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

const ll MAX = 10101;
const ll INF = 1e9;
ll n, m, l[MAX], r[MAX], idx[MAX];
bool root[MAX];
ll cnt, dep[MAX], mn[MAX], mx[MAX];
```

```cpp
void dfs(ll cur, ll dep = 1){
    // 왼쪽 자식이 있으면 DFS
    if(l[cur] != -1) dfs(l[cur], dep + 1);

    // 왼쪽 자식 탐색이 끝났으면
    // 현재 정점에 번호 부여
    ll num = ++cnt;
    mn[dep] = min(mn[dep], num);
    mx[dep] = max(mx[dep], num);

    // 오른쪽 자식이 있으면 DFS
    if(r[cur] != -1) dfs(r[cur], dep + 1);
}
```

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

    cin >> n;
    // 루트 값을 찾아야 함
    // min 값 INF로 초기화
    for(int i = 1;i <= n;i++) root[i] = 1, mn[i] = INF;
    for(int i = 1;i <= n;i++){
        cin >> idx[i] >> l[idx[i]] >> r[idx[i]];

        // 누군가의 자식 정점이면 루트 노드가 아님
        if(l[idx[i]] != -1) root[l[idx[i]]] = 0;
        if(r[idx[i]] != -1) root[r[idx[i]]] = 0;
    }

    for(int i = 1;i <= n;i++){
        // 루트 노드가 아니면 건너 뜀
        if(!root[i]) continue;
        dfs(i);
    }

    ll diff = 0, result = 0;
    for(int i = 1;i <= n;i++){
        // 현재 깊이의 너비가 원래 너비보다 크면
        if(mx[i] - mn[i] + 1 > diff){
            // 값 갱신
            result = i;
            diff = mx[i] - mn[i] + 1;
        }
    }

    cout << result << " " << diff;

    return 0;
}
```

트리의 높이와 너비 / 2250

Python

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

inf = 1000000000
n = int(input().rstrip())
l = [0] * (n + 1)
r = [0] * (n + 1)
idx = [0] * (n + 1)
root = [1] * (n + 1)
mx = [0] * (n + 1)
mn = [inf] * (n + 1)
num = 0
```

```python
def dfs(cur, dep):
    global num
    # 왼쪽 자식이 있으면 DFS
    if l[cur] != -1:
        dfs(l[cur], dep + 1)

    # 왼쪽 자식 탐색이 끝났으면
    # 현재 정점에 번호 부여
    num += 1
    mn[dep] = min(mn[dep], num)
    mx[dep] = max(mx[dep], num)

    # 오른쪽 자식이 있으면 DFS
    if r[cur] != -1:
        dfs(r[cur], dep + 1)

for i in range(1, n + 1):
    # 루트 노드가 아니면 건너 뜀
    if root[i] == 0:
        continue
    dfs(i, 1)
```

```python
for _ in range(n):
    arr = list(map(int, input().rstrip().split()))
    idx = arr[0]
    l[idx], r[idx] = arr[1], arr[2]

    # 누군가의 자식 정점이면 루트 노드가 아님
    if l[idx] != -1:
        root[l[idx]] = 0
    if r[idx] != -1:
        root[r[idx]] = 0

for i in range(1, n + 1):
    # 루트 노드가 아니면 건너 뜀
    if root[i] == 0:
        continue
    dfs(i, 1)

result = 0
diff = 0

for i in range(1, n + 1):
    # 현재 깊이의 너비가 원래 너비보다 크면
    if mx[i] - mn[i] + 1 > diff:
        # 값 갱신
        diff = mx[i] - mn[i] + 1
        result = i

print(f"{result} {diff}")
```

질문?

고생하셨습니다