25-2 이니로
알고리즘 멘토링

멘토 - 김수성

돌 게임 4 / 9658

백준 9658 / https://www.acmicpc.net/problem/9658

문제

돌 게임은 두 명이서 즐기는 재밌는 게임이다.

탁자 위에 돌 N개가 있다. 상근이와 창영이는 턴을 번갈아가면서 돌을 가져가며, 돌은 1개, 3개 또는 4개 가져갈 수 있다. 마지막 돌을 가져가는 사람이 게임을 지게 된다.

두 사람이 완벽하게 게임을 했을 때, 이기는 사람을 구하는 프로그램을 작성하시오. 게임은 상근이가 먼저 시작한다.

입력

첫째 줄에 N이 주어진다. (1 ≤ N ≤ 1000)

출력

상근이가 게임을 이기면 SK를, 창영이가 게임을 이기면 CY을 출력한다.

돌 게임 4 / 9658

처음에 상근이가 3개를 가져 감→ 3개가 남음
두번째에 창영이는 무조건 1개를 가져가야 함 -> 2개가 남음
세번째에 상근이가 1개를 가져 감 -> 1개가 남음
마지막으로 창영이가 1개를 가져가서 상근이가 이기게 됨

출력

상근이가 게임을 이기면 SK를, 창영이가 게임을 이기면 CY을 출력한다.

예제 입력 1 복사

```
6
```

예제 출력 1 복사

```
SK
```

돌 게임 4 / 9658

DP[i][j] = 돌이 i개 남았고, j의 차례일 때 이기는 사람

j = 0이면 상근이의 차례, j = 1이면 창영이의 차례
또한 DP의 값이 0이면 상근이가 이기고,
DP의 값이 1이면 창영이가 이김

돌 게임 4 / 9658

j = 0이면 상근이의 차례, j = 1이면 창영이의 차례
또한 DP의 값이 0이면 상근이가 이기고,
DP의 값이 1이면 창영이가 이김

일단 Base Case를 확인해보자

돌 게임 4 / 9658

j = 0이면 상근이의 차례, j = 1이면 창영이의 차례
또한 DP의 값이 0이면 상근이가 이기고,
DP의 값이 1이면 창영이가 이김

DP[0][0] = 0
돌이 없는데 상근이의 차례면 창영이가 마지막 돌을 가져갔으므로
상근이가 이김

DP[0][1] = 1
돌이 없는데 창영이의 차례면 상근이가 마지막 돌을 가져갔으므로
창영이가 이김

돌 게임 4 / 9658

현재 상근이의 차례이면 돌을 가져갔을 때
본인이 이기는 경우의 수가 있으면 이길 수 있고 없으면 이길 수 없음
또한 돌을 가져가면 차례가 바뀌기 때문에 DP의 인덱스도 바뀜

DP[i][0] = MIN(DP[i – 1][1], DP[i – 3][1], DP[i – 4][1])

돌 게임 4 / 9658

창영이의 차례에도 마찬가지로
본인이 이기는 경우의 수가 있으면 이길 수 있고 없으면 이길 수 없음
또한 돌을 가져가면 차례가 바뀌기 때문에 DP의 인덱스도 바뀜

DP[i][1] = MAX(DP[i – 1][0], DP[i – 3][0], DP[i – 4][0])

돌 게임 4 / 9658

점화식은 다음과 같음

DP[0][0] = 0   DP[0][1] = 1
DP[i][0] = MIN(DP[I – 1][1], DP[I – 3][1], DP[I – 4][1])
DP[i][1] = MAX(DP[I – 1][0], DP[I – 3][0], DP[I – 4][0])

i < 0 일 때 예외 처리를 해줘야 함

돌 게임 4 / 9658

C++

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

const ll MAX = 1010;
ll n, dp[MAX][2];
ll diff[3] = {-1, -3, -4};

ll solve(ll cur, ll turn){
    ll& ret = dp[cur][turn];
    if(ret != -1) return ret; ret = turn ^ 1;

    for(int i = 0;i < 3;i++){
        ll nxt = cur + diff[i];
        if(nxt < 0) break; // 돌은 음수가 될 수 없음
        if(turn) ret = max(ret, solve(nxt, turn ^ 1));
        else ret = min(ret, solve(nxt, turn ^ 1));
    }

    return ret;
}

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

    cin >> n;
    for(int i = 0;i < MAX;i++){
        for(int j = 0;j < 2;j++) dp[i][j] = -1;
    }

    dp[0][0] = 0; dp[0][1] = 1;
    cout << (solve(n, 0) ? "CY" : "SK");

    return 0;
}
```

돌 게임 4 / 9658

Python

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

dp = [[-1] * 2 for _ in range(1010)]
n = int(input().rstrip())

dp[0][0] = 0
dp[0][1] = 1

def solve(cur, turn):
    if dp[cur][turn] != -1:
        return dp[cur][turn]
    dp[cur][turn] = turn ^ 1

    for i in [-1, -3, -4]:
        nxt = cur + i
        # 돌은 음수가 될 수 없음
        if nxt < 0:
            break

        if turn == 1:
            dp[cur][turn] = max(dp[cur][turn], solve(nxt, turn ^ 1))
        else:
            dp[cur][turn] = min(dp[cur][turn], solve(nxt, turn ^ 1))

    return dp[cur][turn]

print("CY" if solve(n, 0) == 1 else "SK")
```

질문?

가장 긴 증가하는 부분 수열 / 11053

백준 11053 / https://www.acmicpc.net/problem/11053

문제

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

예를 들어, 수열 A = {10, 20, 10, 30, 20, 50} 인 경우에 가장 긴 증가하는 부분 수열은 A = {**10**, **20**, 10, **30**, 20, **50**} 이고, 길이는 4이다.

입력

첫째 줄에 수열 A의 크기 N (1 ≤ N ≤ 1,000)이 주어진다.

둘째 줄에는 수열 A를 이루고 있는 A_i가 주어진다. (1 ≤ A_i ≤ 1,000)

출력

첫째 줄에 수열 A의 가장 긴 증가하는 부분 수열의 길이를 출력한다.

가장 긴 증가하는 부분 수열 / 11053

가장 긴 증가하는 부분 수열의 길이를 출력하는 문제
아래에서는 (10 20 10 30 20 50) 으로 정답이 4

출력

첫째 줄에 수열 A의 가장 긴 증가하는 부분 수열의 길이를 출력한다.

예제 입력 1 복사

```
6
10 20 10 30 20 50
```

예제 출력 1 복사

```
4
```

가장 긴 증가하는 부분 수열 / 11053

인덱스 i를 마지막으로 포함하는 수열을 생각해보자

그러면 i 이전의 인덱스의 값이 A[i]보다 작은 어떤 j를
마지막으로 포함하는 수열에 A[i]를 추가해서 만들 수 있음

EX) 10 20 10 30 20 50
-> 10 50
-> 20 50
-> 30 50

가장 긴 증가하는 부분 수열 / 11053

인덱스 i를 마지막으로 포함하는 수열을 생각해보자

그러면 i 이전의 인덱스의 값이 A[i]보다 작은 어떤 j를
마지막으로 포함하는 수열에 A[i]를 추가해서 만들 수 있음

DP[i] = 수열의 마지막이 A[i]인 수열 중 최대 길이

j < i && A[j] < A[i] 인 j에 대해 A[i]를 붙여서 수열을 만들 수 있음
-> DP[i] = DP[j] + 1

가장 긴 증가하는 부분 수열 / 11053

DP[i] = 수열의 마지막이 A[i]인 수열 중 최대 길이

j < i && A[j] < A[i] 인 j에 대해 A[i]를 붙여서 수열을 만들 수 있음
-> DP[i] = DP[j] + 1

Base Case는 DP[0] = 0으로 설정 하면 됨

가장 긴 증가하는 부분 수열 / 11053

C++

```cpp
const ll MAX = 1010;
ll n, a[MAX], dp[MAX];

ll solve(ll cur){
    ll& ret = dp[cur];
    if(ret != -1) return ret; ret = 0;

    for(int i = 0;i < cur;i++){
        // 이전의 값이 크거나 같으면 증가하는 부분 수열이 아님
        if(a[i] >= a[cur]) continue;

        // 이전의 수열 길이에 +1
        ret = max(ret, solve(i) + 1);
    }

    return ret;
}

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];
    for(int i = 0;i < MAX;i++) dp[i] = -1;

    ll result = 0; dp[0] = 0;
    //1 ~ n의 DP값 중 최댓값이 정답
    for(int i = 1;i <= n;i++) result = max(result, solve(i));
    cout << result;

    return 0;
}
```

가장 긴 증가하는 부분 수열 / 11053

Python

```python
MAX = 1010
n = int(input())
a = [0] + list(map(int, input().split()))
dp = [-1] * MAX

def solve(cur):
    if dp[cur] != -1:
        return dp[cur]
    dp[cur] = 0

    for i in range(cur):
        # 이전의 값이 크거나 같으면 증가하는 부분 수열이 아님
        if a[i] >= a[cur]:
            continue
        # 이전의 수열 길이에 +1
        dp[cur] = max(dp[cur], solve(i) + 1)

    return dp[cur]

result = 0
dp[0] = 0

# 1 ~ n의 DP값 중 최댓값이 정답
for i in range(1, n + 1):
    result = max(result, solve(i))

print(result)
```

질문?

쉬운 계단 수 / 10844

백준 10844 / https://www.acmicpc.net/problem/10844

문제

45656이란 수를 보자.

이 수는 인접한 모든 자리의 차이가 1이다. 이런 수를 계단 수라고 한다.

N이 주어질 때, 길이가 N인 계단 수가 총 몇 개 있는지 구해보자. 0으로 시작하는 수는 계단수가 아니다.

입력

첫째 줄에 N이 주어진다. N은 1보다 크거나 같고, 100보다 작거나 같은 자연수이다.

출력

첫째 줄에 정답을 1,000,000,000으로 나눈 나머지를 출력한다.

쉬운 계단 수 / 10844

길이가 1 인 계단 수 1 2 3 4 5 6 7 8 9
길이가 2 인 계단 수 10 21 32 43 54 65 76 87 98
12 23 34 45 56 67 78

예제 입력 1 복사

```
1
```

예제 출력 1 복사

```
9
```

예제 입력 2 복사

```
2
```

예제 출력 2 복사

```
17
```

쉬운 계단 수 / 10844

길이가 1 인 계단 수 1 2 3 4 5 6 7 8 9

길이가 2 인 계단 수 10 21 32 43 54 65 76 87 98
12 23 34 45 56 67 78

계단 수의 마지막 수만 고려해보자
계단 수의 현재 수는 이전의 수와 차이가 1이여야 함

쉬운 계단 수 / 10844

계단 수의 마지막 수만 고려해보자
계단 수의 현재 수는 이전의 수와 차이가 1이여야 함

DP[i][j] = 길이가 i이고, 마지막 수가 j일 때 계단 수의 개수
DP[i][j]는 DP[i – 1][j – 1], DP[i – 1][j + 1]에서 전이 함

쉬운 계단 수 / 10844

DP[i][j] = 길이가 i이고, 마지막 수가 j일 때 계단 수의 개수
DP[i][j]는 DP[i – 1][j – 1], DP[i – 1][j + 1]에서 전이 함

Base Case

i가 1일 때 계단 수는 0으로 시작 할 수 없음
DP[1][j] = 0 (j = 0)
DP[1][j] = 1 (j > 0 && j < 9)

쉬운 계단 수 / 10844

DP[i][j] = 길이가 i이고, 마지막 수가 j일 때 계단 수의 개수
DP[i][j]는 DP[i – 1][j – 1], DP[i – 1][j + 1]에서 전이 함

DP[i][j]는 DP[i – 1][j – 1], DP[i – 1][j + 1]의 계단 수에서
j를 붙여 만들 수 있음
-> DP[i][j] = DP[i – 1][j – 1] + DP[i – 1][j + 1]

j < 0인 경우와 j > 9인 경우는 0으로 처리 해줘야 함
값이 커질 수 있으므로 1e9로 나누기 처리도 해줘야 함

쉬운 계단 수 / 10844

C++

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

    ll n; cin >> n;
    for(int i = 0;i < MAX;i++){
        for(int j = 0;j <= 9;j++) dp[i][j] = -1;
    }

    // 0 ~ 9의 합이 정답
    for(int i = 0;i <= 9;i++){
        result += solve(n, i);
        result %= MOD;
    }

    cout << result;
    return 0;
}
```

```cpp
const ll MAX = 101;
const ll MOD = 1e9;
ll n, dp[MAX][10], result;

ll solve(ll cur, ll num){
    // 수가 음수거나 한 자리 수가 아니면 0 반환
    if(num > 9 || num < 0) return 0;
    ll& ret = dp[cur][num];
    if(ret != -1) return ret; ret = 0;

    // 0으로 시작하는 계단 수는 계단 수가 아님
    if(cur == 1) return ret = num ? 1 : 0;

    // 이전의 수와 현재 수가 한개 차이가 나야 함
    ret += solve(cur - 1, num + 1) % MOD;
    ret += solve(cur - 1, num - 1) % MOD;

    return ret %= MOD;
}
```

쉬운 계단 수 / 10844

Python

```python
MOD = 10**9
dp = [[-1] * 10 for _ in range(1010)]

def solve(cur, num):
    # 수가 음수거나 한 자리 수가 아니면 0 반환
    if num < 0 or num > 9:
        return 0

    if dp[cur][num] != -1:
        return dp[cur][num]

    # 0으로 시작하는 계단 수는 계단 수가 아님
    if cur == 1:
        return 1 if num != 0 else 0

    # 이전의 수와 현재 수가 한개 차이가 나야 함
    dp[cur][num] = solve(cur - 1, num - 1) + solve(cur - 1, num + 1)
    dp[cur][num] %= MOD

    return dp[cur][num]

n = int(input().rstrip())

result = 0
# 0 ~ 9의 합이 정답
for i in range(10):
    result += solve(n, i)
    result %= MOD

print(result)
```

질문?

8주차 - DP

1, 2, 3 더하기 4 / 15989

백준 15989 / https://www.acmicpc.net/problem/15989

문제

정수 4를 1, 2, 3의 합으로 나타내는 방법은 총 4가지가 있다. 합을 나타낼 때는 수를 1개 이상 사용해야 한다. 합을 이루고 있는 수의 순서만 다른 것은 같은 것으로 친다.

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+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 10,000보다 작거나 같다.

출력

각 테스트 케이스마다, n을 1, 2, 3의 합으로 나타내는 방법의 수를 출력한다.

1, 2, 3 더하기 4 / 15989

저번주에 풀었던 문제와 비슷함
저번주에 풀었던 문제는 순서를 고려함

이 문제는 순서를 고려하지 않음

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

예제 입력 1 복사

```
3
4
7
10
```

예제 출력 1 복사

```
4
8
14
```

1, 2, 3 더하기 4 / 15989

이 문제는 순서를 고려하지 않음
2 + 1 + 1, 1 + 1 + 2, 1 + 2 + 1은 같은 것으로 취급

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

순서를 고려하지 않는 법

-> 순서에 제한을 둬 보자

제한을 두는 법

-> DP 상태에 추가

1, 2, 3 더하기 4 / 15989

제한을 두는 법

-> DP 상태에 추가

순서를 고려할 때 문제를 푸는 법

DP[i] = DP[i – 1] + DP[i – 2] + DP[i – 3]

순서를 고려하지 않으니 제한을 둬보자

-> 숫자는 항상 오름차순으로 나와야 함

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

1, 2, 3 더하기 4 / 15989

순서를 고려하지 않으니 제한을 둬보자
-> 숫자는 항상 오름차순으로 나와야 함

EX) 1 + 1 + 2
1 + 1 + 2 + 2 + 3
1 + 3 + 3 + 3

1 + 2 + 1 처럼 오름차순이 아니면
올바르지 않은 것으로 간주함

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

1, 2, 3 더하기 4 / 15989

EX) 1 + 1 + 2
1 + 1 + 2 + 2 + 3
1 + 3 + 3 + 3

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

숫자를 오름차순으로 배치해보자

숫자 2가 나오면 더 이상 뒤에 + 1 을 붙일 수 없음
숫자 3이 나오면 더 이상 뒤에 + 1, +2 를 붙일 수 없음

1, 2, 3 더하기 4 / 15989

숫자 2가 나오면 더 이상 뒤에 + 1 을 붙일 수 없음
숫자 3이 나오면 더 이상 뒤에 + 1, +2 를 붙일 수 없음

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

숫자를 몇 까지 사용했는지 알아야 함
-> DP 상태에 추가함

1, 2, 3 더하기 4 / 15989

숫자를 몇 까지 사용했는지 알아야 함
-> DP 상태에 추가함

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

원래 점화식
DP[i] = i를 1, 2, 3의 합으로 표현한 경우의 수

바뀐 점화식
DP[i][j] = 마지막 숫자가 j일 때 i를 1, 2, 3의 합으로 표현한
오름차순 경우의 수

1, 2, 3 더하기 4 / 15989

바뀐 점화식

DP[i][j] = 마지막 숫자가 j일 때 i를 1, 2, 3의 합으로 표현한
오름차순 경우의 수

DP[i][1] = 마지막 숫자가 1

DP[i][1] = DP[i – 1][1]

DP[i][2] = 마지막 숫자가 2

DP[i][2] = DP[i – 2][1] + DP[i – 2][2]

- 1+1+1+1
- 2+1+1 (1+1+2, 1+2+1)
- 2+2
- 1+3 (3+1)

1, 2, 3 더하기 4 / 15989

바뀐 점화식
DP[i][j] = 마지막 숫자가 j일 때 i를 1, 2, 3의 합으로 표현한
오름차순 경우의 수

DP[i][3] = 마지막 숫자가 3
DP[i][3] = DP[i – 3][1] + DP[i – 3][2] + DP[i – 3][3]

1, 2, 3 더하기 4 / 15989

DP[i][j] = 마지막 숫자가 j일 때 i를 1, 2, 3의 합으로 표현한
오름차순 경우의 수

DP[i][1] = DP[i – 1][1]
DP[i][2] = DP[i – 2][1] + DP[i – 2][2]
DP[i][3] = DP[i – 3][1] + DP[i – 3][2] + DP[i – 3][3]

Base Case

DP[0][1] = 1
DP[0][2] = DP[0][3] = 0

1, 2, 3 더하기 4 / 15989

Base Case

DP[0][1] = 1

DP[0][2] = DP[0][3] = 0

DP[3][3]을 구할 때

DP[3][3] = DP[0][1] + DP[0][2] + DP[0][3] 으로 식이 나옴

DP[0]의 값이 모두 1이면 3이 나와서 틀림

1, 2, 3 더하기 4 / 15989

C++

```cpp
const ll MAX = 10101;
ll dp[MAX][4];

ll solve(ll cur, ll mx){
    if(cur < 0) return 0;
    ll& ret = dp[cur][mx];
    if(ret != -1) return ret; ret = 0;
    if(!cur) return ret = (mx == 1);

    for(int i = 1;i <= mx;i++) ret += solve(cur - mx, i);
    return ret;
}

void run(){
    ll n; cin >> n;
    ll result = 0;
    for(int i = 1;i <= 3;i++) result += solve(n, i);
    cout << result << "\n";
}

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

    for(int i = 0;i < MAX;i++){
        for(int j = 0;j < 4;j++) dp[i][j] = -1;
    }

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

    return 0;
}
```

1, 2, 3 더하기 4 / 15989

Python

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

MAX = 10101
dp = [[-1] * 4 for _ in range(MAX)]
```

```python
def run():
    n = int(input().strip())
    result = 0
    for i in range(1, 4):
        result += solve(n, i)
    print(result)

t = int(input().strip())
for _ in range(t):
    run()
```

```python
def solve(cur, mx):
    if cur < 0:
        return 0

    ret = dp[cur][mx]
    if ret != -1:
        return ret

    ret = 0
    if cur == 0:
        dp[cur][mx] = 1 if mx == 1 else 0
        return dp[cur][mx]

    for i in range(1, mx + 1):
        ret += solve(cur - mx, i)

    dp[cur][mx] = ret
    return ret
```

질문?

동전 2 / 2294

백준 2294 / https://www.acmicpc.net/problem/2294

문제

n가지 종류의 동전이 있다. 이 동전들을 적당히 사용해서, 그 가치의 합이 k원이 되도록 하고 싶다. 그러면서 동전의 개수가 최소가 되도록 하려고 한다. 각각의 동전은 몇 개라도 사용할 수 있다.

입력

첫째 줄에 n, k가 주어진다. (1 ≤ n ≤ 100, 1 ≤ k ≤ 10,000) 다음 n개의 줄에는 각각의 동전의 가치가 주어진다. 동전의 가치는 100,000보다 작거나 같은 자연수이다. 가치가 같은 동전이 여러 번 주어질 수도 있다.

출력

첫째 줄에 사용한 동전의 최소 개수를 출력한다. 불가능한 경우에는 -1을 출력한다.

동전 2 / 2294

N개의 동전이 주어지고, 동전을 사용해서 M원을 만들 때 사용하는 동전의 최소 개수를 구하는 문제

5 + 5 + 5 = 15

예제 입력 1 복사

```
3 15
1
5
12
```

예제 출력 1 복사

```
3
```

동전 2 / 2294

각 동전은 여러 번 사용 할 수 있음
-> 냅색 문제가 아님

어차피 한 동전을 여러 번 사용 할 수 있으니
동전에 대한 정보는 dp에 필요가 없음

DP[i][j] = i 번째 동전까지 사용 했을 때 j 원을 만들기 위한 최소 동전
-> DP[j] = j 원을 만들기 위한 최소 동전

동전 2 / 2294

DP[j] = j 원을 만들기 위한 최소 동전

각 배열 A의 값 A[i]에 대해서 각 동전을 j원을 만들기 위해서
쓴다고 생각해보자

예제에서는 A = {1, 5, 12}
A[1]을 사용해서 j원을 만들기 위해서는 j – A[1]에서 A[1]을 더함
A[2]를 사용해서 j원을 만들기 위해서는 j – A[1]에서 A[2]를 더함
…

동전 2 / 2294

DP[j] = j 원을 만들기 위한 최소 동전

예제에서는 A = {1, 5, 12}
A[1]을 사용해서 j원을 만들기 위해서는 j – A[1]에서 A[1]을 더함
A[2]를 사용해서 j원을 만들기 위해서는 j – A[1]에서 A[2]를 더함

DP[j] = MIN(DP[j], DP[j – A[i]] + 1)
j – A[i]에서 동전을 1개 더 사용 했으므로 +1 을 해줘야 함
DP의 식에서 MIN값을 사용하므로 INF값으로 DP 초기화

동전 2 / 2294

C++

```cpp
const ll MAX = 10101;
const ll INF = 1e12;
ll dp[MAX], n, m, a[101];

ll solve(ll cur){
    if(cur < 0) return INF;
    ll& ret = dp[cur];
    if(ret != -1) return ret;
    // min값을 구하기 위해서 큰 수로 초기화
    ret = INF;

    // Base Case
    if(!cur) return ret = 0;

    // 각 동전을 1개 씩 사용해서 cur을 만들 때 최솟값
    for(int i = 1;i <= n;i++){
        ret = min(ret, solve(cur - a[i]) + 1);
    }

    return ret;
}
```

```cpp
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];
    for(int i = 0;i < MAX;i++) dp[i] = -1;

    // 값이 INF면 M원을 만들 수 없음
    cout << (solve(m) == INF ? -1 : solve(m));

    return 0;
}
```

동전 2 / 2294

Python

```
n, m = map(int, input().split())
a = [0] * (n+1)
for i in range(1, n+1):
    a[i] = int(input())

dp = [-1] * (m+1)

result = solve(m)
# 값이 INF면 M원을 만들 수 없음
print(-1 if result == INF else result)
```

```
import sys
sys.setrecursionlimit(10**7)
input = sys.stdin.readline

INF = 10**12
def solve(cur):
    if cur < 0:
        return INF

    if dp[cur] != -1:
        return dp[cur]

    # Base case
    if cur == 0:
        dp[cur] = 0
        return 0

    # min값을 구하기 위해서 큰 수로 초기화
    dp[cur] = INF

    # 각 동전을 1개 씩 사용해서 cur을 만들 때 최솟값
    for i in range(1, n+1):
        dp[cur] = min(dp[cur], solve(cur - a[i]) + 1)

    return dp[cur]
```

질문?

가장 긴 바이토닉 부분 수열 / 11054

백준 11054 / https://www.acmicpc.net/problem/11054

문제

수열 S가 어떤 수 S_k를 기준으로 $S_1 < S_2 < \ldots S_{k-1} < S_k > S_{k+1} > \ldots S_{N-1} > S_N$을 만족한다면, 그 수열을 바이토닉 수열이라고 한다.

예를 들어, {10, 20, **30**, 25, 20}과 {10, 20, 30, **40**}, {**50**, 40, 25, 10} 은 바이토닉 수열이지만, {1, 2, 3, 2, 1, 2, 3, 2, 1}과 {10, 20, 30, 40, 20, 30} 은 바이토닉 수열이 아니다.

수열 A가 주어졌을 때, 그 수열의 부분 수열 중 바이토닉 수열이면서 가장 긴 수열의 길이를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 수열 A의 크기 N이 주어지고, 둘째 줄에는 수열 A를 이루고 있는 A_i가 주어진다. ($1 \le N \le 1{,}000$, $1 \le A_i \le 1{,}000$)

출력

첫째 줄에 수열 A의 부분 수열 중에서 가장 긴 바이토닉 수열의 길이를 출력한다.

가장 긴 바이토닉 부분 수열 / 11054

가장 긴 바이토닉 부분 수열의 길이를 구하는 문제

수열 S가 어떤 수 S_k를 기준으로 $S_1 < S_2 < \ldots S_{k-1} < S_k > S_{k+1} > \ldots S_{N-1} > S_N$을 만족한다면, 그 수열을 바이토닉 수열이라고 한다.

1 5 2 1 4 3 4 5 2 1

증가하다가 어느 순간부터 감소하는 수열

예제 입력 1 복사

```
10
1 5 2 1 4 3 4 5 2 1
```

예제 출력 1 복사

```
7
```

가장 긴 바이토닉 부분 수열 / 11054

1 5 2 1 4 3 4 5 2 1
증가하다가 어느 순간부터 감소하는 수열

저번에 푼 가장 긴 증가하는 부분 수열 문제와 비슷함
DP[i] = DP[j] + 1 (j < i && A[j] < A[i])

증가하는 부분 수열에 감소하는 부분 수열을 붙인 것으로 생각

가장 긴 바이토닉 부분 수열 / 11054

증가하는 부분 수열에 감소하는 부분 수열을 붙인 것으로 생각

현재 수열이 증가하고 있으면 계속 증가하거나, 감소하는 상태로 바뀜
현재 수열이 감소하고 있으면 계속 감소해야 함

-> 현재 수열이 증가하고 있는지, 감소하고 있는지 알아야 함

가장 긴 바이토닉 부분 수열 / 11054

현재 수열이 증가하고 있으면 계속 증가하거나, 감소하는 상태로 바뀜
현재 수열이 감소하고 있으면 계속 감소해야 함

-> 현재 수열이 증가하고 있는지, 감소하고 있는지 알아야 함

DP[i][0] = 현재 수열이 증가하고 i 번째 인덱스가 수열의 마지막 일 때 최대 길이

DP[i][1] = 현재 수열이 감소하고 i 번째 인덱스가 수열의 마지막 일 때 최대 길이

가장 긴 바이토닉 부분 수열 / 11054

DP[i][0] = 현재 수열이 증가하고 i 번째 인덱스가 수열의 마지막 일 때 최대 길이

DP[i][1] = 현재 수열이 감소하고 i 번째 인덱스가 수열의 마지막 일 때 최대 길이

DP[i][0] = DP[j][0] + 1 (j < i && A[j] < A[i])
DP[i][1] = MAX(DP[j][0], DP[j][1]) + 1 (j < i && A[j] > A[i])

가장 긴 바이토닉 부분 수열 / 11054

DP[i][0] = DP[j][0] + 1 (j < i && A[j] < A[i])
DP[i][1] = MAX(DP[j][0], DP[j][1]) + 1 (j < i && A[j] > A[i])

Base Case

DP[1][0] = DP[1][1] = 1

{10, 20, **30**, 25, 20}과 {10, 20, 30, **40**}, {**50**, 40, 25, 10} 은 바이토닉 수열이지만,

문제의 조건에 따라서 DP[i][0]도 정답이 될 수 있음

가장 긴 바이토닉 부분 수열 / 11054

C++

```cpp
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];
    for(int i = 0;i < MAX;i++){
        for(int j = 0;j <= 1;j++) dp[i][j] = -1;
    }

    ll result = 0;
    for(int i = 1;i <= n;i++){
        // DP[i][0], DP[i][1] 중 최댓값이 정답
        result = max(result, max(solve(i, 0), solve(i, 1)));
    }
    cout << result;
    return 0;
}
```

```cpp
const ll MAX = 1010;
ll dp[MAX][2], a[MAX], n;

ll solve(ll cur, ll cnt){
    ll& ret = dp[cur][cnt];
    if(ret != -1) return ret; ret = 1;

    // 감소하는 상태
    if(cnt) for(int i = 1;i < cur;i++){
        // 이전 값이 현재 값보다 작으면 건너 뜀
        if(a[i] <= a[cur]) continue;
        ret = max(ret, solve(i, 0) + 1);
        ret = max(ret, solve(i, 1) + 1);
    }
    // 증가하는 상태
    else for(int i = 1;i < cur;i++){
        // 이전 값이 현재 값보다 크면 건너 뜀
        if(a[i] >= a[cur]) continue;
        ret = max(ret, solve(i, 0) + 1);
    }

    return ret;
}
```

가장 긴 바이토닉 부분 수열 / 11054

Python

```python
n = int(input())
A = list(map(int, input().split()))
a = [0] + A

dp = [[-1] * 2 for _ in range(n + 1)]

ans = 0
for i in range(1, n + 1):
    # DP[i][0], DP[i][1]중 최댓값이 정답
    ans = max(ans, solve(i, 0), solve(i, 1))

print(ans)
```

```python
import sys
sys.setrecursionlimit(10**7)
input = sys.stdin.readline

def solve(cur, cnt):
    if dp[cur][cnt] != -1:
        return dp[cur][cnt]

    ret = 1
    # 감소하는 상태
    if cnt == 1:
        for i in range(1, cur):
            # 이전 값이 현재 값보다 작으면 건너 뜀
            if a[i] <= a[cur]:
                continue
            ret = max(ret, solve(i, 0) + 1, solve(i, 1) + 1)
    # 증가하는 상태
    else:
        for i in range(1, cur):
            # 이전 값이 현재 값보다 크면 건너 뜀
            if a[i] >= a[cur]:
                continue
            ret = max(ret, solve(i, 0) + 1)

    dp[cur][cnt] = ret
    return ret
```

내리막 길 / 1520

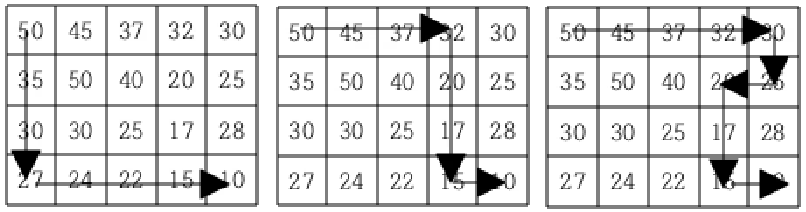
백준 1520 / https://www.acmicpc.net/problem/1520

문제

여행을 떠난 세준이는 지도를 하나 구하였다. 이 지도는 아래 그림과 같이 직사각형 모양이며 여러 칸으로 나뉘어져 있다. 한 칸은 한 지점을 나타내는데 각 칸에는 그 지점의 높이가 쓰여 있으며, 각 지점 사이의 이동은 지도에서 상하좌우 이웃한 곳끼리만 가능하다.

50	45	37	32	30
35	50	40	20	25
30	30	25	17	28
27	24	22	15	10

현재 제일 왼쪽 위 칸이 나타내는 지점에 있는 세준이는 제일 오른쪽 아래 칸이 나타내는 지점으로 가려고 한다. 그런데 가능한 힘을 적게 들이고 싶어 항상 높이가 더 낮은 지점으로만 이동하여 목표 지점까지 가고자 한다. 위와 같은 지도에서는 다음과 같은 세 가지 경로가 가능하다.



지도가 주어질 때 이와 같이 제일 왼쪽 위 지점에서 출발하여 제일 오른쪽 아래 지점까지 항상 내리막길로만 이동하는 경로의 개수를 구하는 프로그램을 작성하시오.

내리막 길 / 1520

(1,1)에서 출발해서 항상 감소하는 숫자로 이동 할 때 (n,m)에 도착하는 경우의 수를 구하는 문제

예제 입력 1 복사

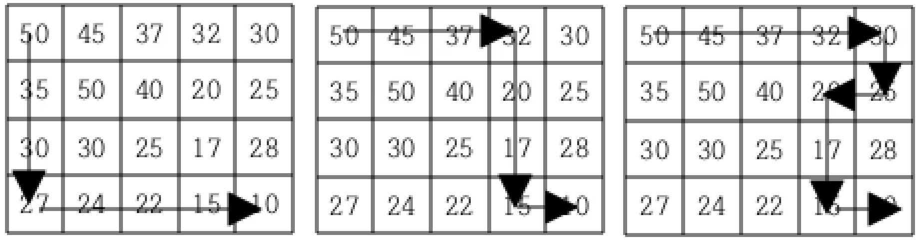
```
4 5
50 45 37 32 30
35 50 40 20 25
30 30 25 17 28
27 24 22 15 10
```

예제 출력 1 복사

```
3
```

내리막 길 / 1520

각 칸을 정점, 각 칸에서 내리막으로 가는 길을 간선으로 가진
그래프를 생각해보자



각 칸에 도달 할 수 있는 경우의 수는 각 칸에 연결되어 있는
이전의 칸의 경우의 수를 더해서 구할 수 있음

내리막 길 / 1520

각 칸에 도달 할 수 있는 경우의 수는 각 칸에 연결되어 있는
이전의 칸의 경우의 수를 더해서 구할 수 있음

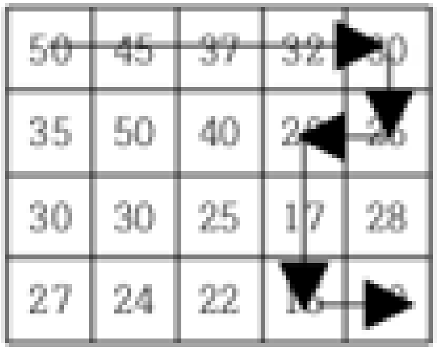
DP[i][j] = (0,0)에서 내리막 길만 사용해서 (i,j)에 도달하는 경우의 수
DP[i][j] = DP[i – 1][j] + DP[i + 1][j] + DP[i][j – 1] + DP[i][j + 1]

물론 이전의 값이 더 클 때만 더해야 함

내리막 길 / 1520

DP[i][j] = (0,0)에서 내리막 길만 사용해서 (i,j)에 도달하는 경우의 수
DP[i][j] = DP[i – 1][j] + DP[i + 1][j] + DP[i][j – 1] + DP[i][j + 1]

또한 이 문제는 바텀 업으로 풀기 어려움
바텀 업으로 DP 테이블을 채우는 순서가
일정하지 않음



DP 테이블을 채우는 순서를 구해야 함
-> 위상정렬
2주 뒤에 배움

내리막 길 / 1520

C++

```cpp
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++){
        for(int j = 1;j <= m;j++){
            cin >> a[i][j];
            dp[i][j] = -1;
        }
    }
    cout << solve(n, m);

    return 0;
}
```

```cpp
const ll MAX = 505;
ll n, m, dp[MAX][MAX], a[MAX][MAX];
ll dx[4] = {0, 0, 1, -1}, dy[4] = {1, -1, 0, 0};

bool outrange(ll cy, ll cx){
    return cx <= 0 || cy <= 0 || cx > m || cy > n;
}

ll solve(ll cy, ll cx){
    ll& ret = dp[cy][cx];
    if(ret != -1) return ret; ret = 0;
    // Base Case
    if(cy == 1 && cx == 1) return ret = 1;

    for(int i = 0;i < 4;i++){
        ll ny = cy + dy[i], nx = cx + dx[i];
        // 격자 밖이면 건너 뜀
        if(outrange(ny, nx)) continue;
        // 내리막 길이 아니면 건너 뜀
        if(a[ny][nx] <= a[cy][cx]) continue;
        ret += solve(ny, nx);
    }

    return ret;
}
```

내리막 길 / 1520

Python

```python
n, m = map(int, input().split())
a  = [[0] * (m+1) for _ in range(n+1)]
dp = [[-1] * (m+1) for _ in range(n+1)]

for i in range(1, n+1):
    row = list(map(int, input().split()))
    for j in range(1, m+1):
        a[i][j] = row[j-1]

print(solve(n, m))
```

```python
import sys
sys.setrecursionlimit(10**7)
input = sys.stdin.readline

dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

def solve(cy, cx):
    if dp[cy][cx] != -1:
        return dp[cy][cx]

    # Base Case
    if cy == 1 and cx == 1:
        dp[cy][cx] = 1
        return 1

    ret = 0
    for i in range(4):
        ny = cy + dy[i]
        nx = cx + dx[i]
        # 격자 밖이면 건너 뜀
        if not (1 <= ny <= n and 1 <= nx <= m):
            continue

        # 내리막 길이 아니면 건너 뜀
        if a[ny][nx] <= a[cy][cx]:
            continue
        ret += solve(ny, nx)

    dp[cy][cx] = ret
    return ret
```

질문?

고생하셨습니다