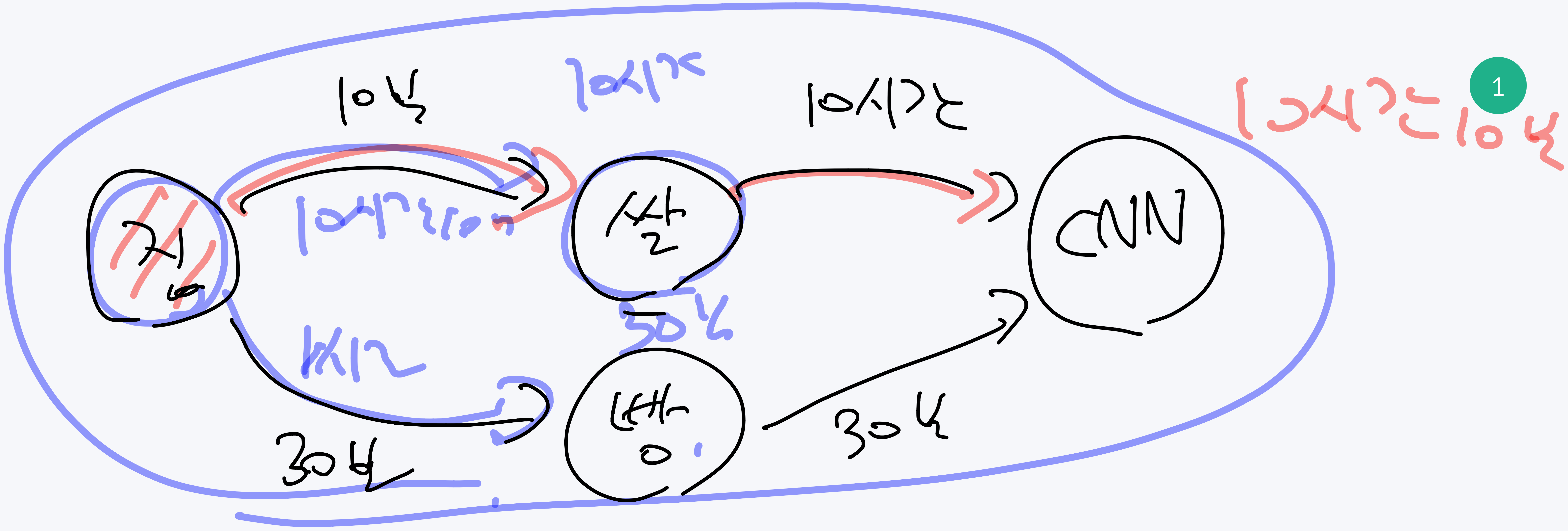
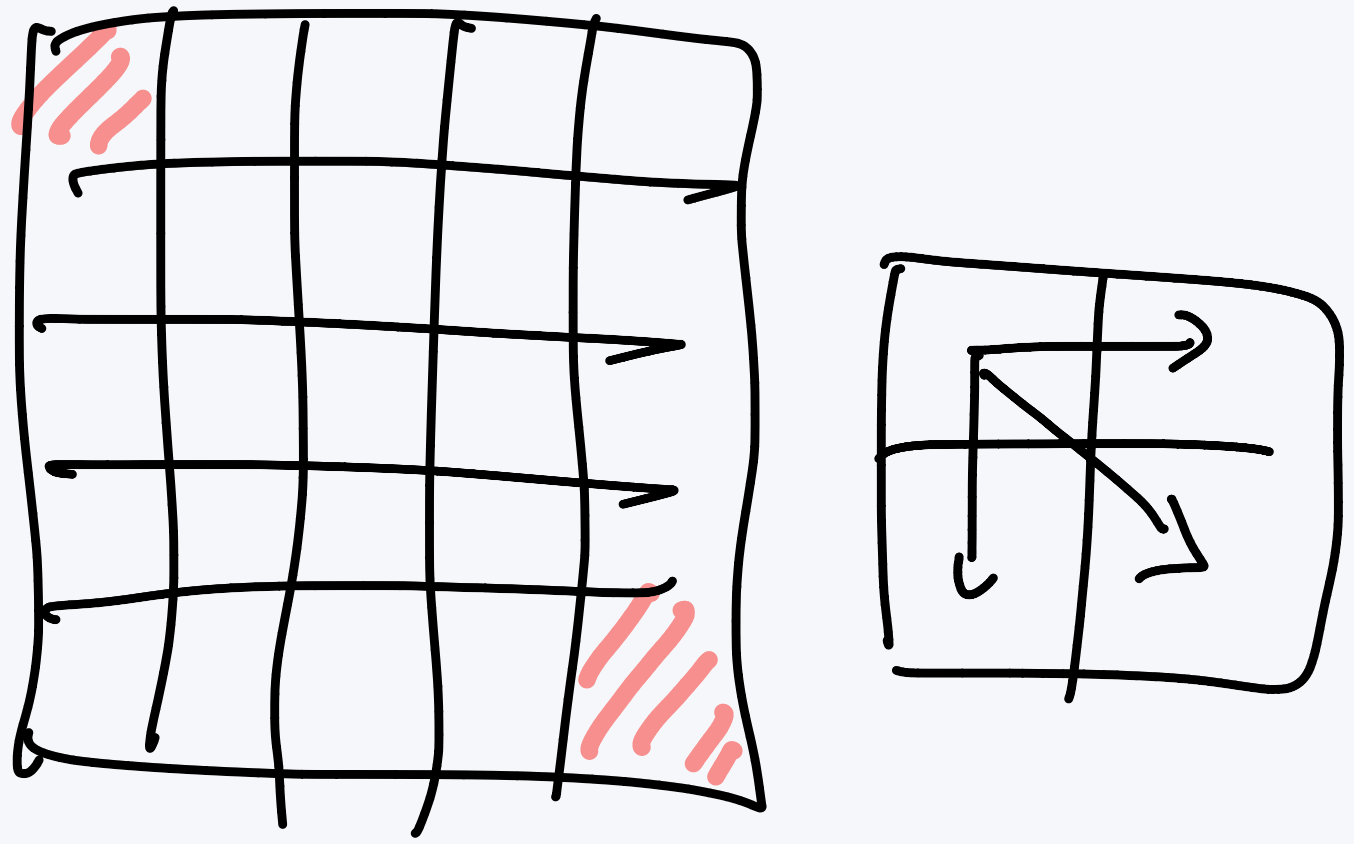


다이나믹 프로그래밍 2

최백준 choi@startlink.io

이동하기

https://www.acmicpc.net/problem/11048

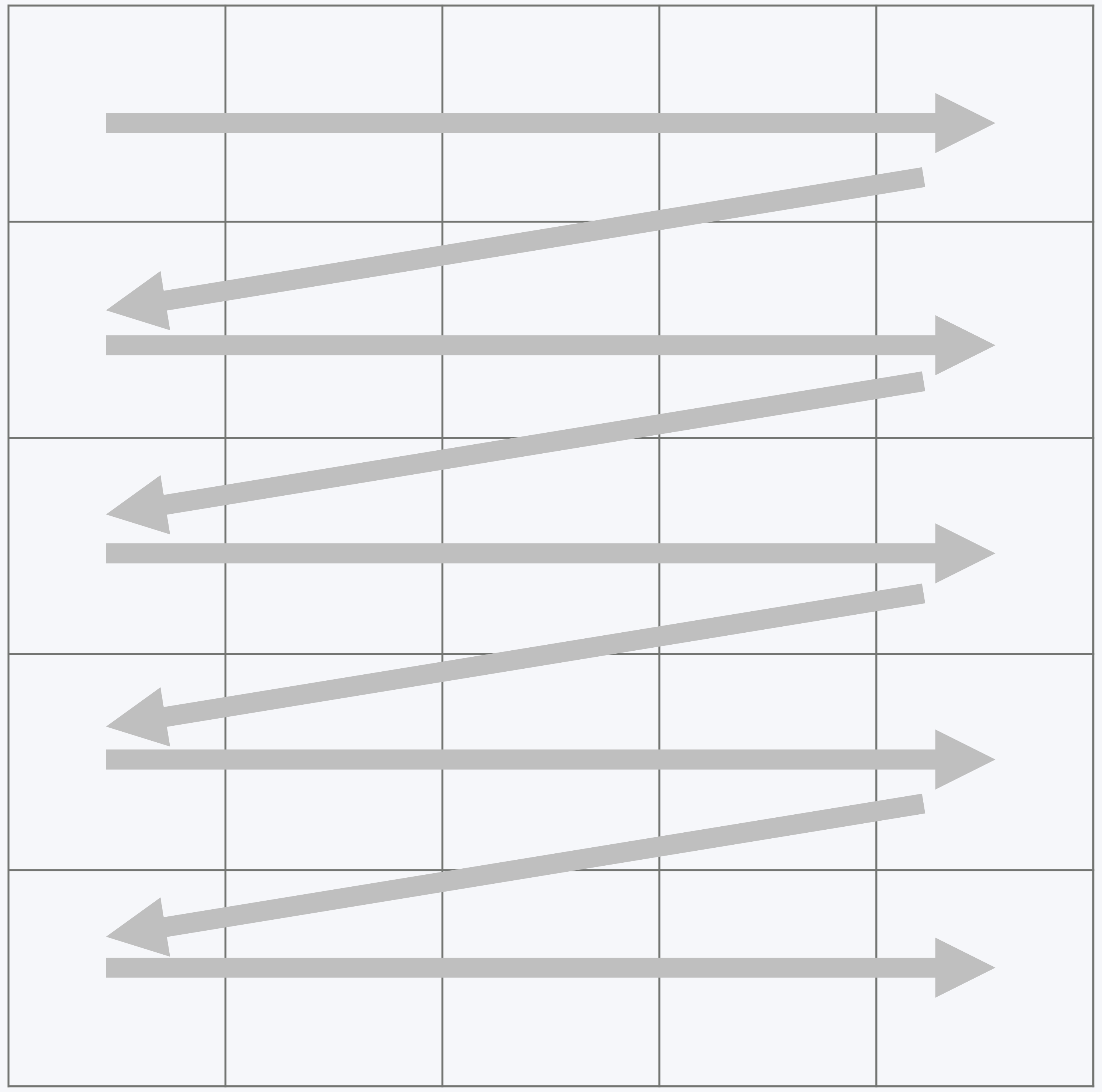


- 준규는 N×M 크기의 미로에 갇혀있다
- 미로는 1×1크기의 방으로 나누어져 있고, 각 방에는 사탕이 놓여져 있다
- 미로의 가장 왼쪽 윗 방은 (1, 1)이고, 가장 오른쪽 아랫 방은 (N, M)이다
- 준규는 현재 (1, 1)에 있고, (N, M)으로 이동하려고 한다
- 준규가 (i, j)에 있으면, (i+1, j), (i, j+1), (i+1, j+1)로 이동할 수 있고, 각 방을 방문할 때마다 방에 놓여져있는 사탕을 모두 가져갈 수 있다
- 또, 미로 밖으로 나갈 수는 없다
- 준규가 (N, M)으로 이동할 때, 가져올 수 있는 사탕 개수의 최대값

이동하기

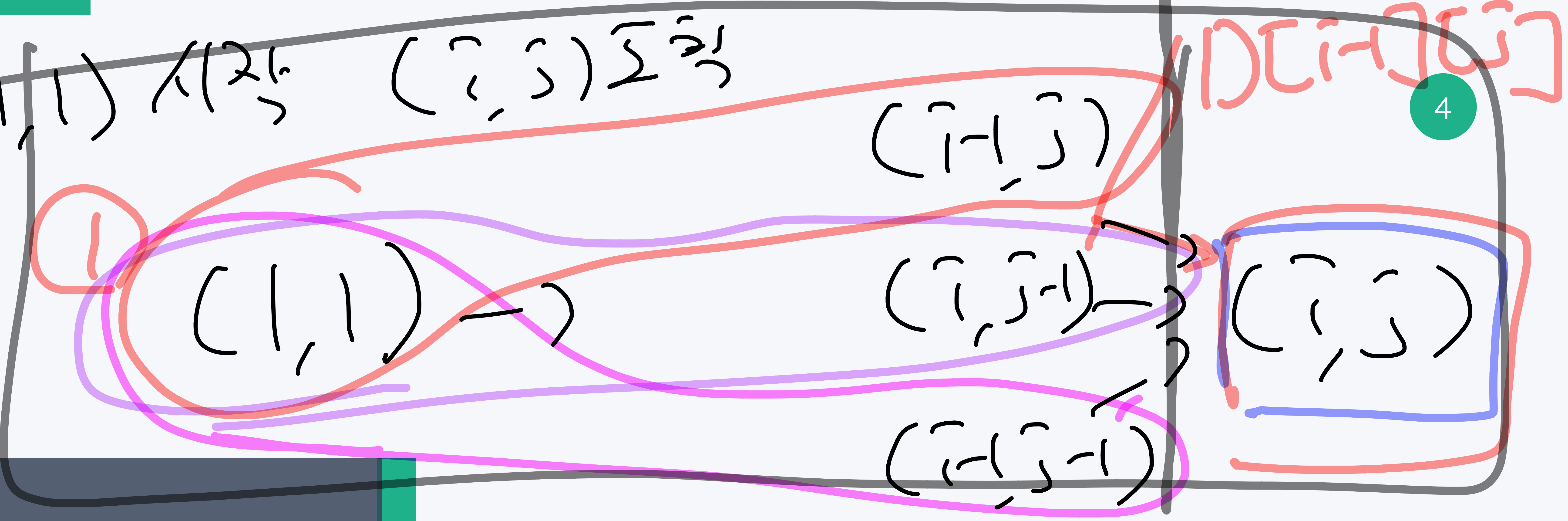
https://www.acmicpc.net/problem/11048

- 항상 아래와 오른쪽으로만 갈 수 있다.
- (i,j)에서 가능한 이동: (i+1, j), (i, j+1), (i+1, j+1)



D[i][j] = (1,1) 시작 (i,j) 도착

= max(①, ②, ③)

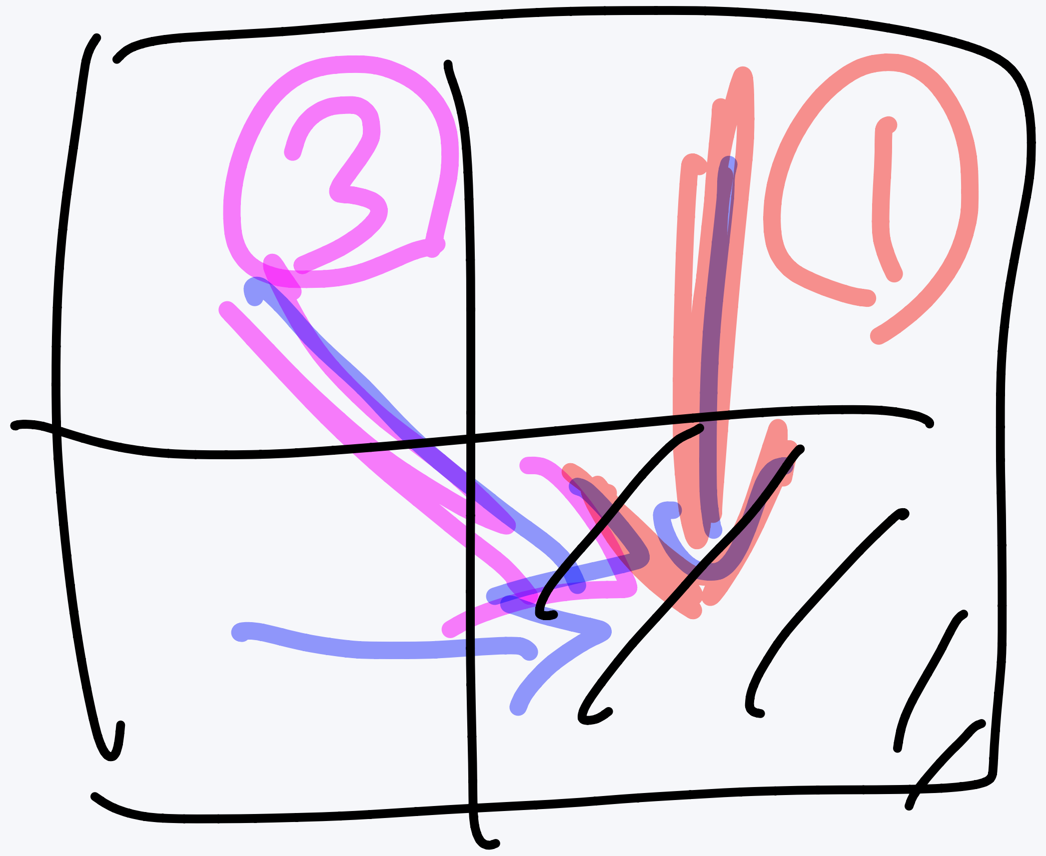


방법 1

① D[i-1][j] + A[i][j]

② D[i][j-1] + A[i][j]

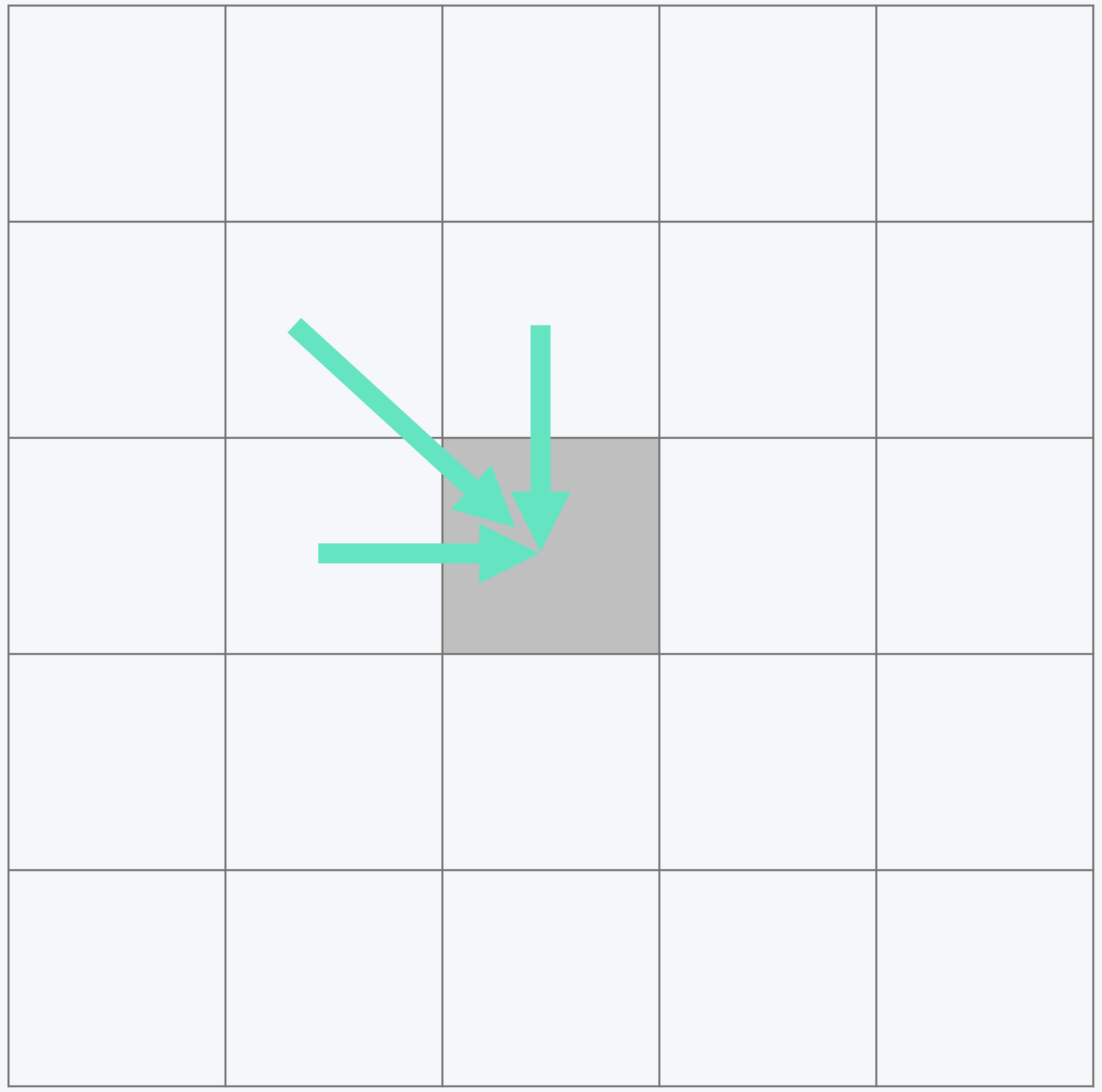
③ D[i-1][j-1] + A[i][j]



이동하기

https://www.acmicpc.net/problem/11048

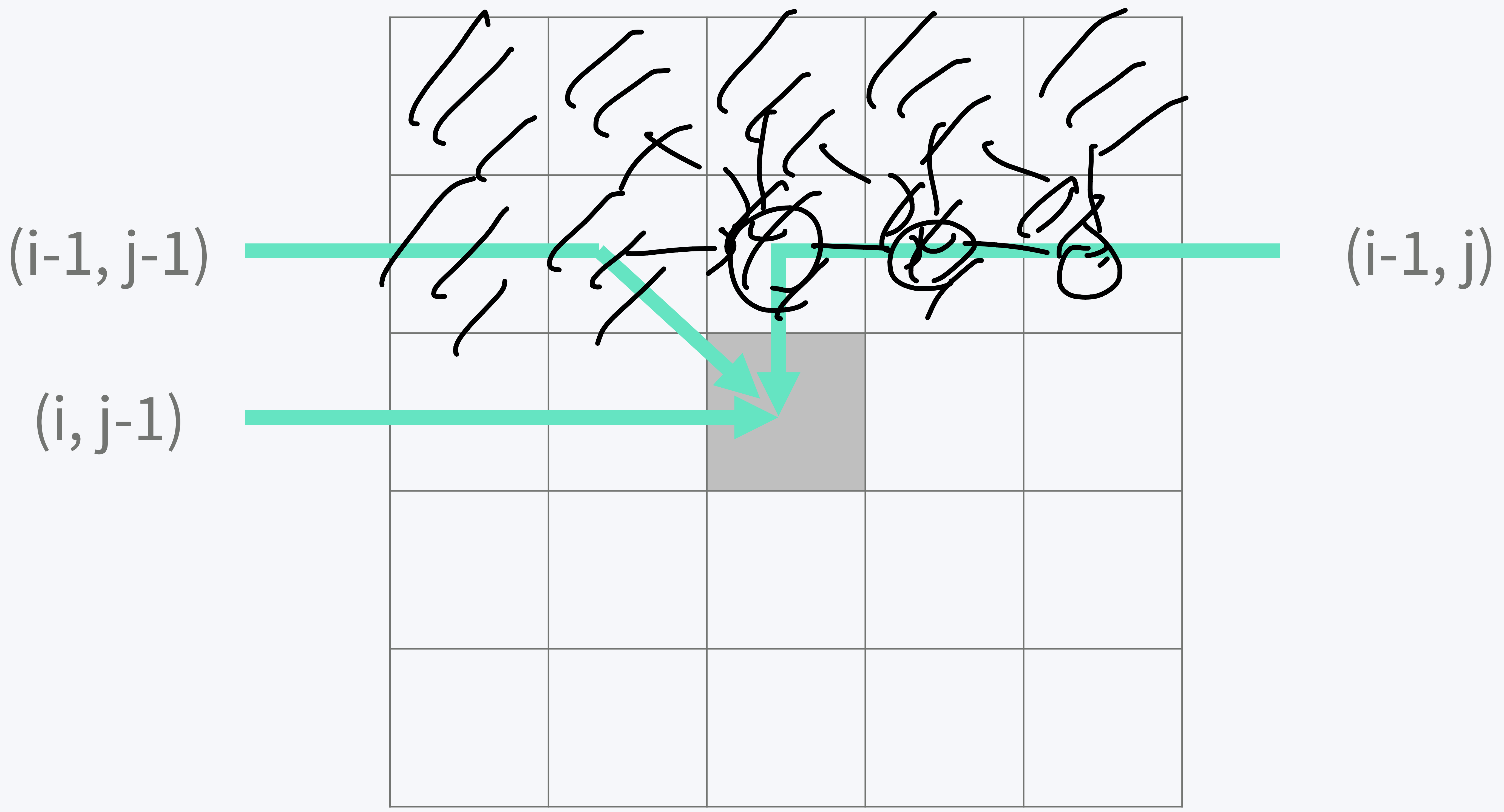
- 항상 아래와 오른쪽으로만 갈 수 있다.
- (i,j)에서 가능한 이동: (i+1, j), (i, j+1), (i+1, j+1)



이동하기

https://www.acmicpc.net/problem/11048

- 항상 아래와 오른쪽으로만 갈 수 있다.
- (i,j)에서 가능한 이동: (i+1, j), (i, j+1), (i+1, j+1)

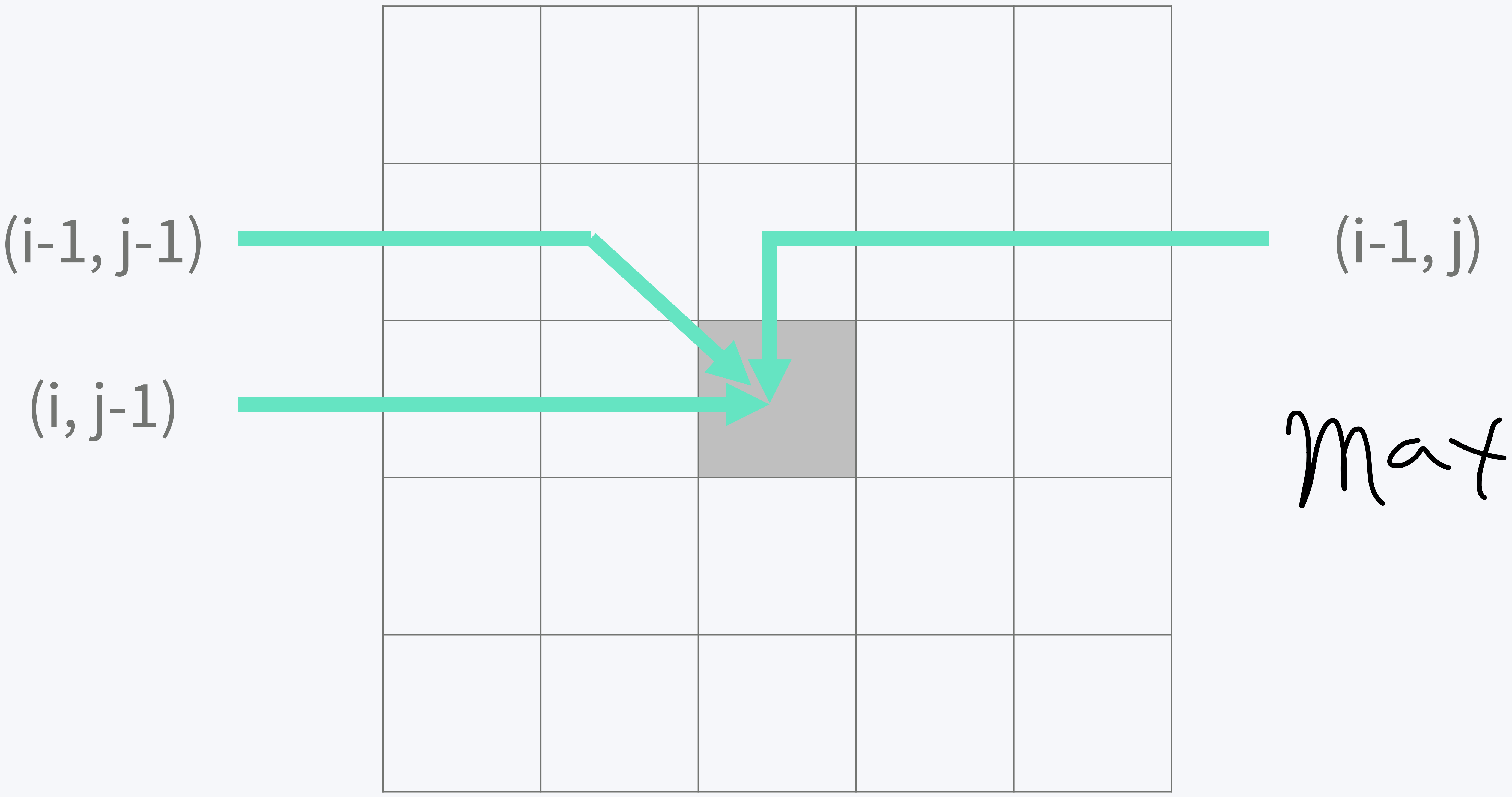


이동하기

https://www.acmicpc.net/problem/11048

- D[i][j] = (i, j)로 이동할 때 가져올 수 있는 최대 사탕 개수
- D[i][j] = Max(D[i-1][j-1], D[i][j-1], D[i-1][j]) + A[i][j]

```
#define max(a,b)
(((a)>(b))?(a):(b))
```



max(go(1), go(2))

이동하기

https://www.acmicpc.net/problem/11048

D[i][j] = (1,1) 시작
(i,j) 도착

```
for (int i=1; i<=n; i++) {
    for (int j=1; j<=m; j++) {
        d[i][j] = max3(d[i-1][j],d[i][j-1],d[i-1][j-1])+a[i][j];
    }
}
```

max(max(a,b),c)

max(a,b)
if(a>b) return(a)
else return b

if(a>b && a>c) return a;

max({a,b,c});

C++11

이동하기

https://www.acmicpc.net/problem/11048

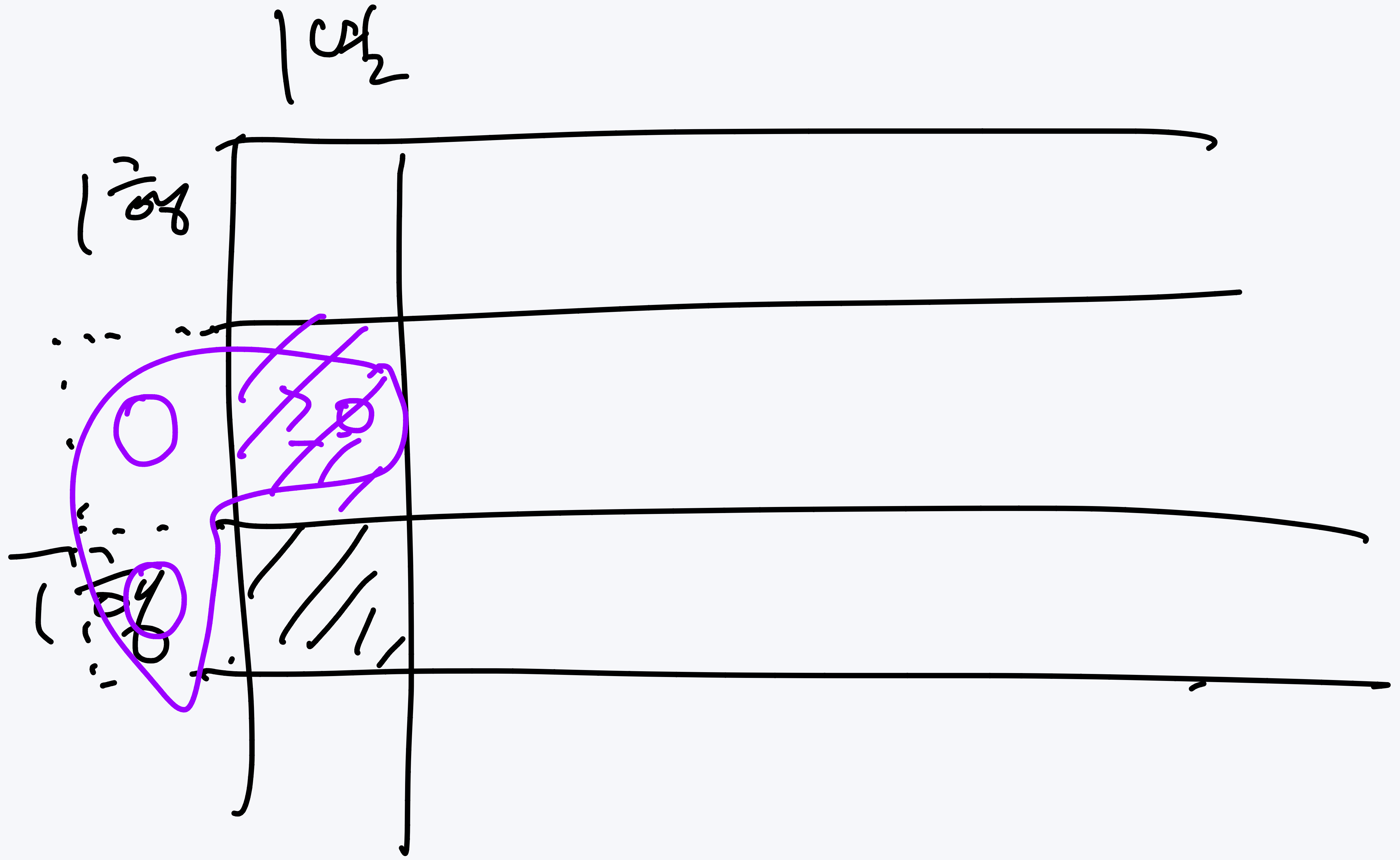
$i-1 \geq 1$

0~n-1

1~n

```
for (int i=1; i<=n; i++) {
    for (int j=1; j<=m; j++) {
        d[i][j] = max3(d[i-1][j],d[i][j-1],d[i-1][j-1])+a[i][j];
    }
}
```

- i-1, j-1 범위 검사를 하지 않은 이유
- i = 1, j =1인 경우
- i = 1인 경우
- j = 1인 경우



이동하기

https://www.acmicpc.net/problem/11048

```
for (int i=1; i<=n; i++) {
    for (int j=1; j<=m; j++) {
        d[i][j] = max3(d[i-1][j],d[i][j-1],d[i-1][j-1])+a[i][j];
    }
}
```

- i-1, j-1 범위 검사를 하지 않은 이유
- i = 1, j =1인 경우: d[i-1][j], d[i][j-1], d[i-1][j-1]은 0이기 때문
- i = 1인 경우: d[i-1][j] = 0 < d[i][j-1] 이기 때문
- j = 1인 경우: d[i][j-1] = 0 < d[i-1][j] 이기 때문

이동하기

https://www.acmicpc.net/problem/11048

- C/C++
  - https://gist.github.com/Baekjoon/65f34d5cf5f329dde337
- Java
  - https://gist.github.com/Baekjoon/51fc0cd6a1b2db1a4d48

$D[i][j]$ = (1,1) 시작 (i,j) 도착

방법 2

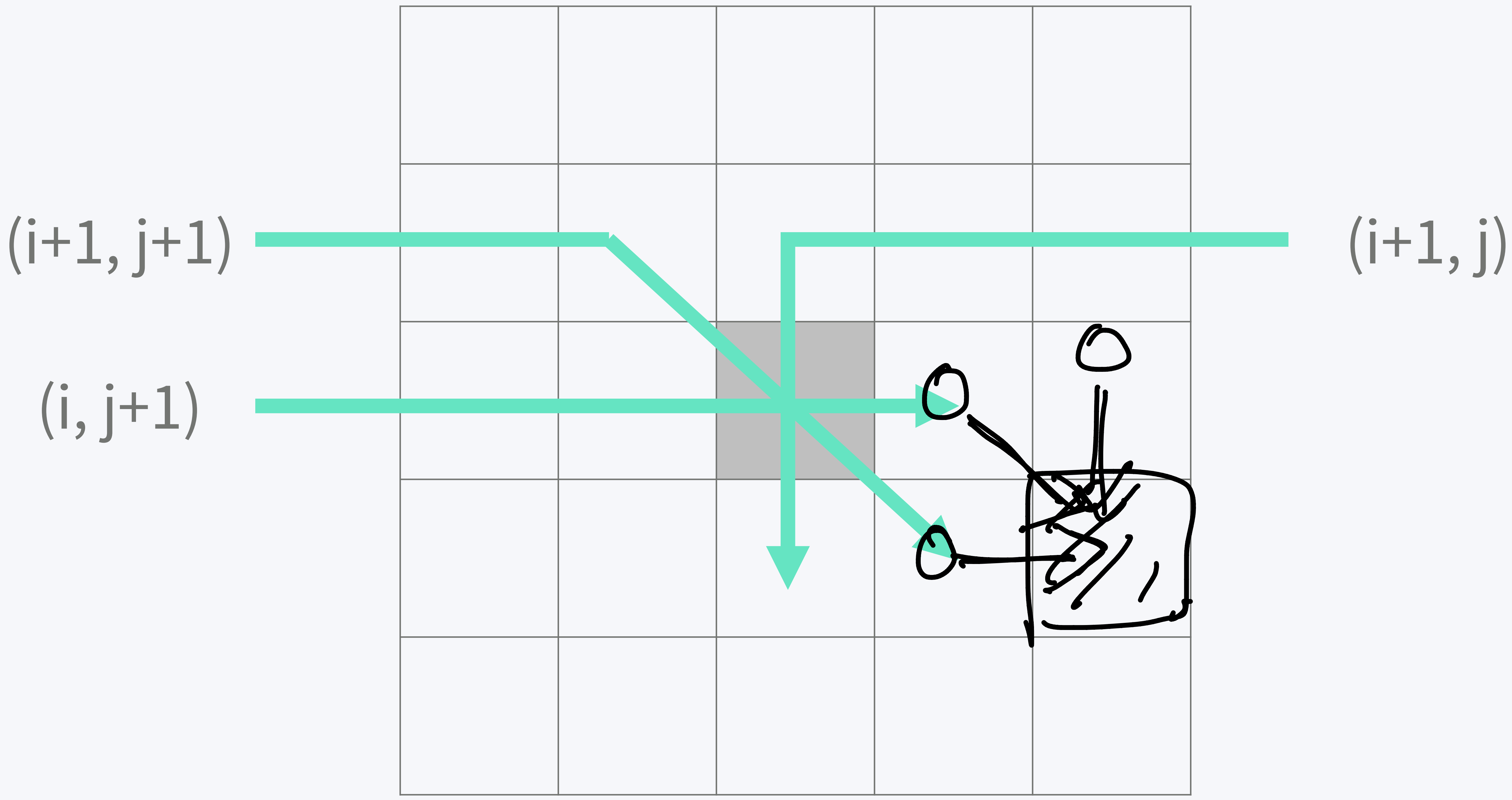
$$(1,1) \rightarrow (i,j) \rightarrow \begin{cases} (i+1,j) \\ (i,j+1) \\ (i+1,j+1) \end{cases}$$

$$D[i+1][j] = D[i][j] + A[i+1][j]$$

이동하기

https://www.acmicpc.net/problem/11048

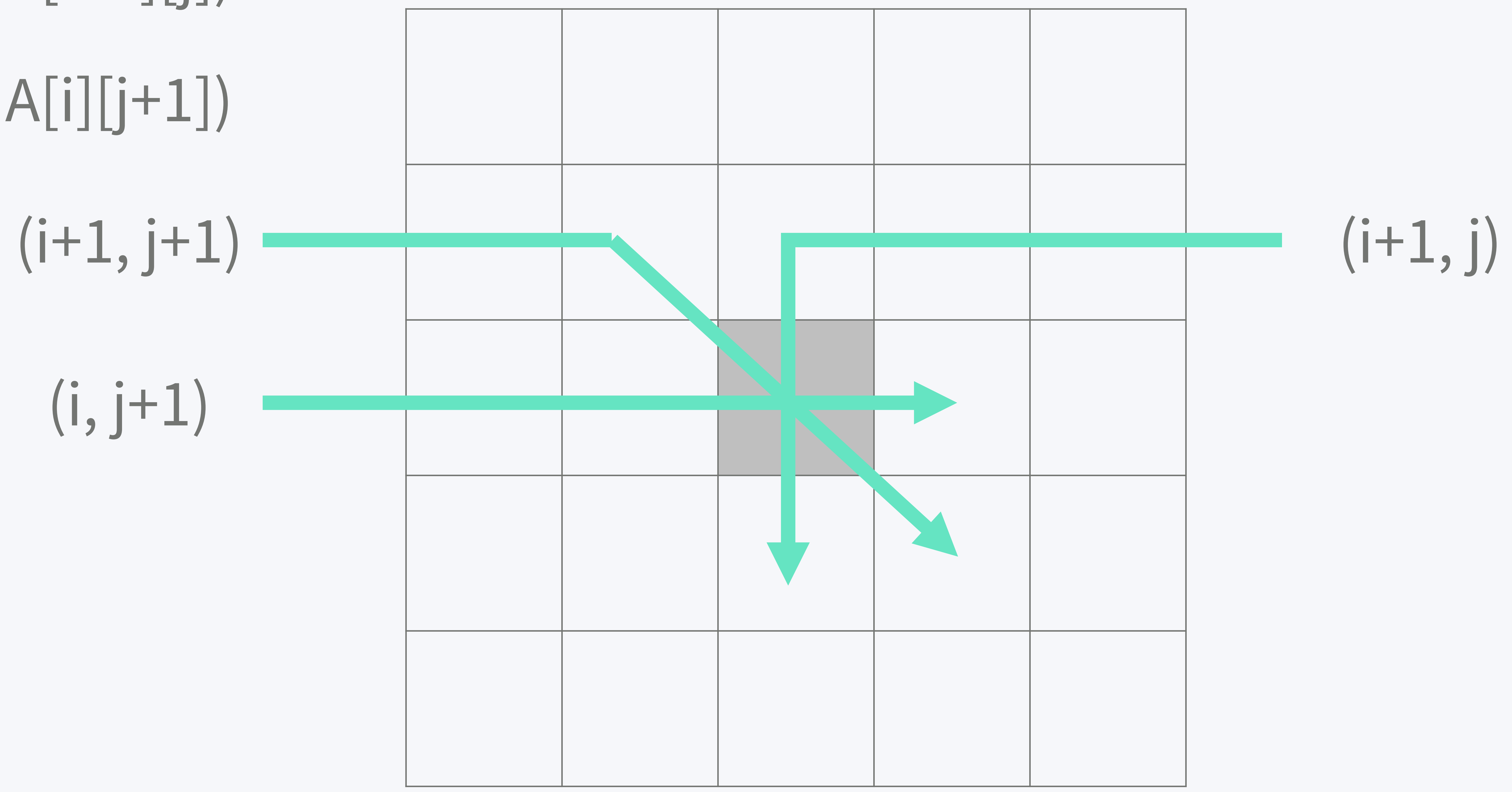
- 항상 아래와 오른쪽으로만 갈 수 있다.
- (i,j)에서 가능한 이동: (i+1, j), (i, j+1), (i+1, j+1)



이동하기

https://www.acmicpc.net/problem/11048

- D[i][j] = (i, j)로 이동할 때 가져올 수 있는 최대 사탕 개수
- D[i+1][j+1] = max(D[i+1][j+1], D[i][j] + A[i+1][j+1])
- D[i+1][j] = max(D[i+1][j], D[i][j] + A[i+1][j])
- D[i][j+1] = max(D[i][j+1], D[i][j] + A[i][j+1])



이동하기

D 배열에 들어있는 값

https://www.acmicpc.net/problem/11048

```
for (int i=1; i<=n; i++) {
    for (int j=1; j<=m; j++) {
        if (d[i][j+1] < d[i][j] + a[i][j+1]) {
            d[i][j+1] = d[i][j] + a[i][j+1];
        }
        if (d[i+1][j] < d[i][j] + a[i+1][j]) {
            d[i+1][j] = d[i][j] + a[i+1][j];
        }
        if (d[i+1][j+1] < d[i][j] + a[i+1][j+1]) {
            d[i+1][j+1] = d[i][j] + a[i+1][j+1];
        }
    }
}
```

이동하기

https://www.acmicpc.net/problem/11048

- C/C++
  - https://gist.github.com/Baekjoon/ce48db57373eabc2beeb
- Java
  - https://gist.github.com/Baekjoon/df34e7d5daf341c8b76a

방법 3

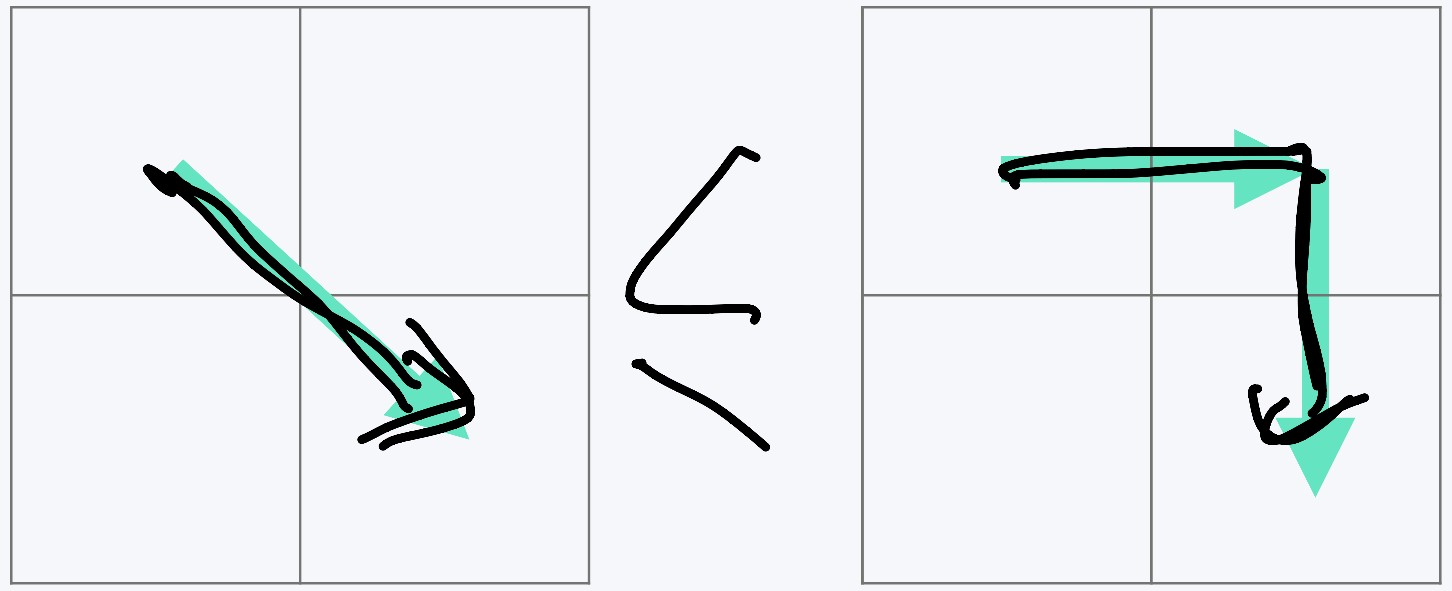
이동하기

https://www.acmicpc.net/problem/11048

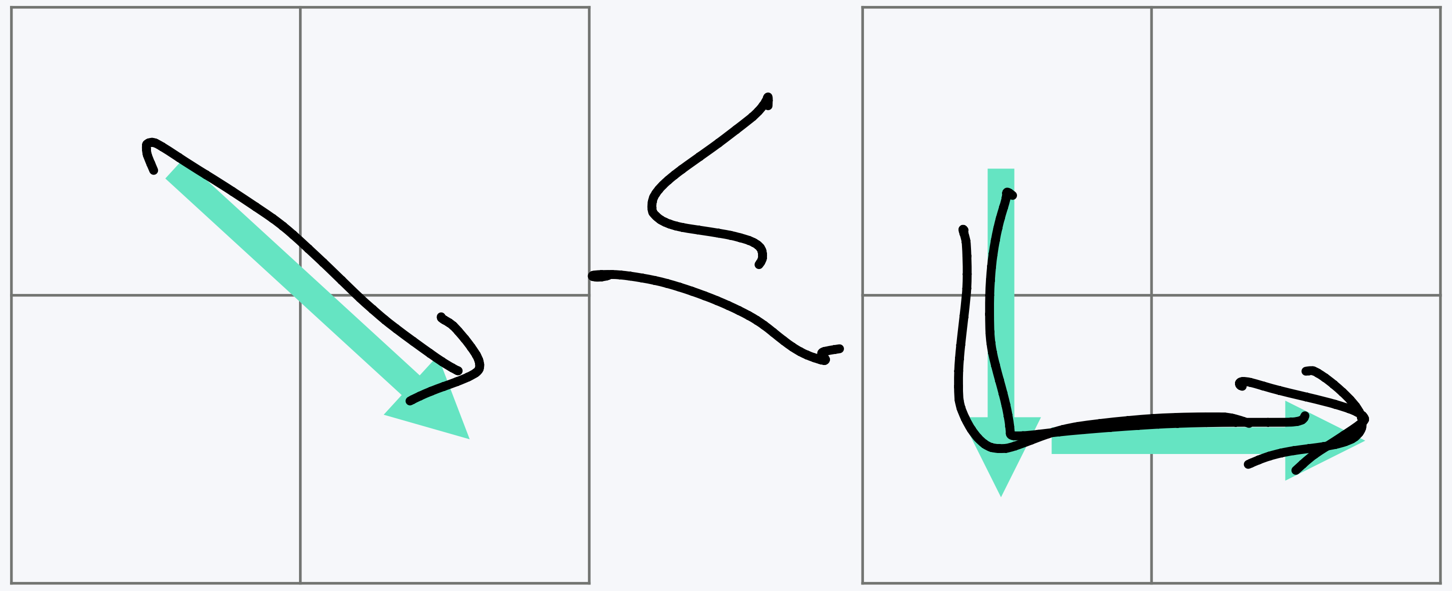
$\begin{matrix} 3 & -1 \\ -1 & 3 \end{matrix}$

이유: $\boxed{20}$

- 대각선 이동은 처리하지 않아도 된다
- 대각선 이동은 다른 2가지를 포함한 방법보다 항상 작거나 같다
- A[i][j] + A[i+1][j+1] ≤ A[i][j] + A[i][j+1] + A[i+1][j+1]



- A[i][j] + A[i+1][j+1] ≤ A[i][j] + A[i+1][j] + A[i+1][j+1]



이동하기

https://www.acmicpc.net/problem/11048

```
for (int i=1; i<=n; i++) {
    for (int j=1; j<=m; j++) {
        d[i][j] = max(d[i-1][j],d[i][j-1])+a[i][j];
    }
}
```

이동하기

https://www.acmicpc.net/problem/11048

- C/C++
  - https://gist.github.com/Baekjoon/9abbafed45a936cd5c67
- Java
  - https://gist.github.com/Baekjoon/6b580f31de918530a7e9

방법 4

이동하기

https://www.acmicpc.net/problem/11048

- 재귀 함수를 이용해서도 구현할 수 있다
- D[i][j] = (i, j)로 이동할 때 가져올 수 있는 최대 사탕 개수
- D[i][j] = max(D[i][j-1], D[i-1][j]) + A[i][j]
- 식이 달라지는 것이 아니고 구현 방식이 달라지는 것이다

이동하기

https://www.acmicpc.net/problem/11048

```
int go(int i, int j) {
    if (i == 1 && j == 1) return a[1][1];
    if (i < 1 || j < 1) return 0;
    if (d[i][j] >= 0) {
        return d[i][j];
    }
    d[i][j] = go(i-1, j) + a[i][j];
    int temp = go(i, j-1) + a[i][j];
    if (d[i][j] < temp) {
        d[i][j] = temp;
    }
    return d[i][j];
}
```

이동하기

https://www.acmicpc.net/problem/11048

- 방법 1~4의 점화식은 모두 같았는데
- 구현 방식, 식을 채우는 순서만 조금씩 달랐다

이동하기

https://www.acmicpc.net/problem/11048

- C/C++
  - https://gist.github.com/Baekjoon/3833c76ff9cf5a8a0f2c
- Java
  - https://gist.github.com/Baekjoon/9cda32c0258dab2ce563

$D[i][j]$ = (1,1) 시작 (i,j) 도착
= (i,j) 시작 (N,M) 도착

$D[i][j+1]$
$D[i+1][j]$

방법 5

(i,j) → (i+1,j) → (N,M)
(i,j) → (i,j+1) → (N,M)

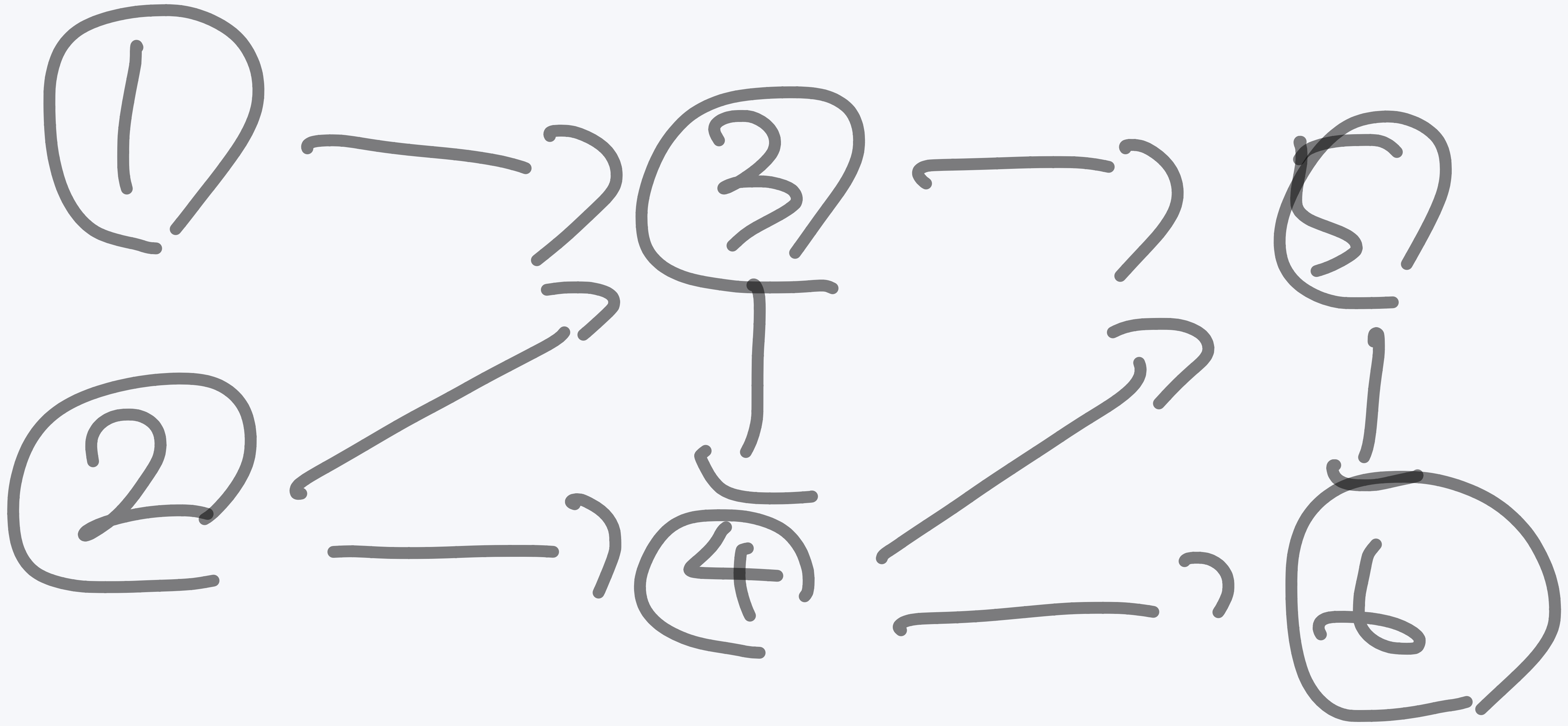
이동하기

https://www.acmicpc.net/problem/11048

DAG

$F_i = F_{i-1} + F_{i-2}$

- 점화식을 조금 바꿔서 세워보자
- D[i][j] = (i, j)에서 이동을 시작했을 때, 가져올 수 있는 최대 사탕 개수
- 지금까지의 점화식
- D[i][j] = (i, j)로 이동했을 때, 가져올 수 있는 최대 사탕 개수



이동하기

https://www.acmicpc.net/problem/11048

- 점화식을 조금 바꿔서 세워보자
- D[i][j] = (i, j)에서 이동을 시작했을 때, 가져올 수 있는 최대 사탕 개수
- 도착(N, M)으로 정해져 있는데, 시작(i, j)을 이동시키는 방식
- 지금까지의 점화식
- D[i][j] = (i, j)로 이동했을 때, 가져올 수 있는 최대 사탕 개수
- 시작은 (1, 1)로 정해져 있고, 도착 (i, j)을 이동시 키는 방식

이동하기

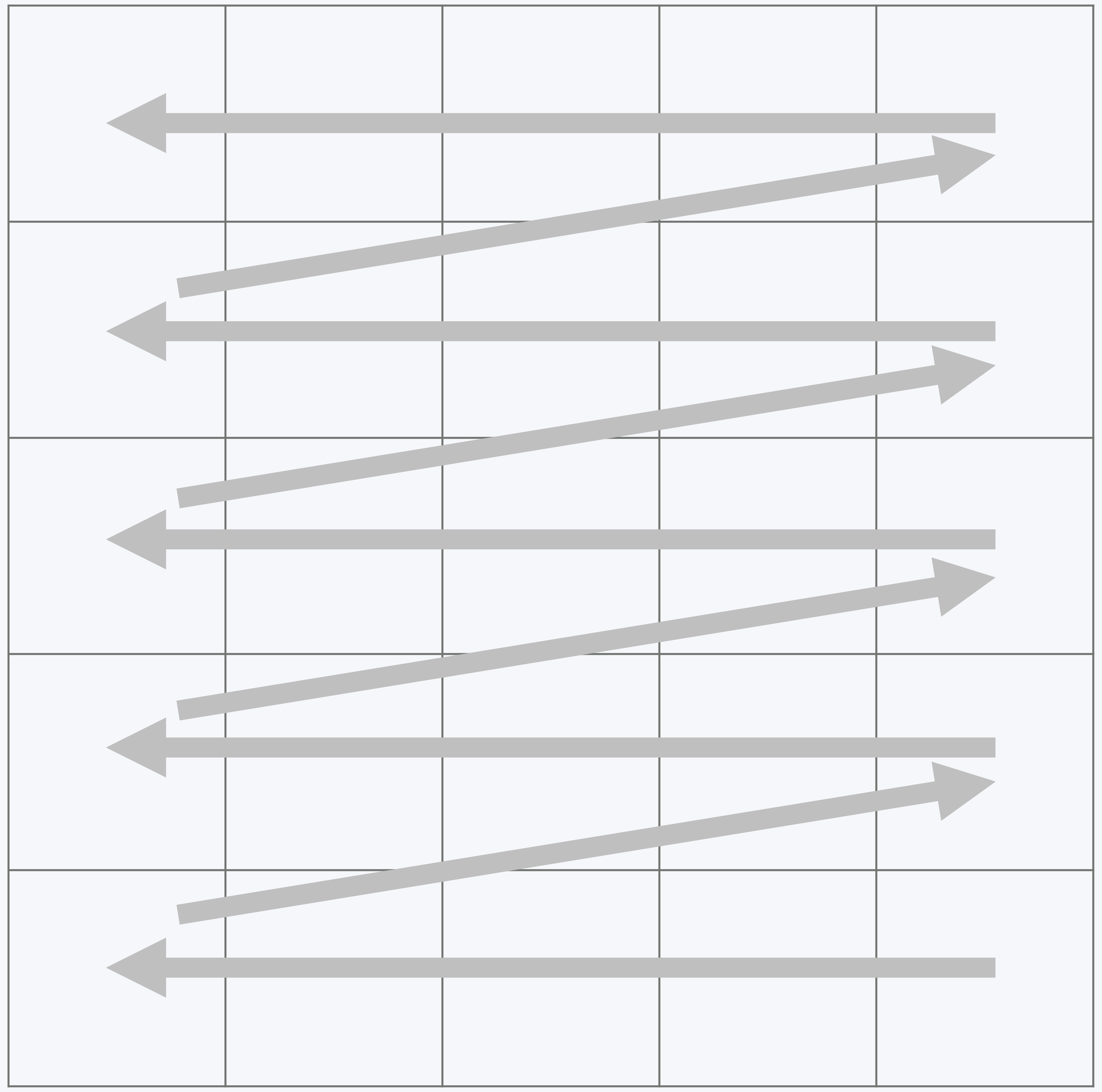
https://www.acmicpc.net/problem/11048

- D[i][j] = (i, j)에서 이동을 시작했을 때, 가져올 수 있는 최대 사탕 개수
- D[i][j] = max(D[i+1][j], D[i][j+1]) + A[i][j]

이동하기

https://www.acmicpc.net/problem/11048

- 항상 아래와 오른쪽으로만 갈 수 있다.
- (i,j)에서 가능한 이동: (i+1, j), (i, j+1), (i+1, j+1)



이동하기

https://www.acmicpc.net/problem/11048

```
int go(int x, int y) {
    if (x > n || y > m) return 0;
    if (d[x][y] >= 0) return d[x][y];
    d[x][y] = go(x+1,y) + a[x][y];
    int temp = go(x,y+1) + a[x][y];
    if (d[x][y] < temp) {
        d[x][y] = temp;
    }
    return d[x][y];
}
```

이동하기

https://www.acmicpc.net/problem/11048

- D[i][j] = (i, j)에서 이동을 시작했을 때, 가져올 수 있는 최대 사탕 개수
- D[i][j] = max(D[i+1][j], D[i][j+1]) + A[i][j]
- 정답은 D[1][1]에 있다.
- 즉, go(1, 1)을 호출해서 답을 구해야 한다.

이동하기

https://www.acmicpc.net/problem/11048

- C/C++
  - https://gist.github.com/Baekjoon/6ce67c14be4576103dec
  - https://gist.github.com/Baekjoon/0c8d0f4d28eb497063d9db0cb092b805
- Java
  - https://gist.github.com/Baekjoon/172da179fea2419ca3e7

문제 풀기

점프

https://www.acmicpc.net/problem/1890

- N×N 게임판에 수가 적혀져 있음
- 게임의 목표는 가장 왼쪽 위 칸에서 가장 오른쪽 아래 칸으로 규칙에 맞게 점프를 해서 가는 것
- 각 칸에 적혀있는 수는 현재 칸에서 갈 수 있는 거리를 의미
- 반드시 오른쪽이나 아래쪽으로만 이동해야 함
- 0은 더 이상 진행을 막는 종착점이며, 항상 현재 칸에 적혀있는 수만큼 오른쪽이나 아래로 가야 함
- 가장 왼쪽 위 칸에서 가장 오른쪽 아래 칸으로 규칙에 맞게 이동할 수 있는 경로의 개수를 구하는 문제

점프

https://www.acmicpc.net/problem/1890

- D[i][j] = (i, j)칸에 갈 수 있는 경로의 개수
- (i, j)칸에 올 수 있는 칸을 찾아야 한다.

점프

https://www.acmicpc.net/problem/1890

- D[i][j] = (i, j)칸에 갈 수 있는 경로의 개수
- (i, j)칸에 올 수 있는 칸을 찾아야 한다.
- D[i][j] += D[i][k] (k+a[i][k] == j, $0 \le k < j$)
- D[i][j] += D[k][j] (k+a[k][j] == i, $0 \le k < i$)

점프

https://www.acmicpc.net/problem/1890

- D[i][j] = (i, j)칸에 갈 수 있는 경로의 개수
- (i, j)칸에 올 수 있는 칸을 찾아야 한다.
- D[i][j] += D[i][k] (k+A[i][k] == j, 0 ≤ k < j)
- D[i][j] += D[k][j] (k+A[k][j] == i, 0 ≤ k < i)

- 한 칸을 채우는데 필요한 복잡도: O(N)
- 총 시간 복잡도 : O(N^3)

점프

https://www.acmicpc.net/problem/1890

- C/C++: https://gist.github.com/Baekjoon/b5f8bae461a2e43a35b25d515a6b5946

점프

https://www.acmicpc.net/problem/1890

- D[i][j] = (i, j)칸에 갈 수 있는 경로의 개수
- (i, j)에서 갈 수 있는 칸을 찾아야 한다.
- D[i][j+A[i][j]] += D[i][j];
- D[i+A[i][j]][j] += D[i][j];

- 한 칸을 채우는데 필요한 복잡도: O(1)
- 총 시간 복잡도 : O(N^2)

점프

https://www.acmicpc.net/problem/1890

- C/C++: https://gist.github.com/Baekjoon/cdbfbfb7b4de890d766de1579529bee2

팰린드롬?

https://www.acmicpc.net/problem/10942

- 어떤 수열의 부분 수열이 팰린드롬인지 확인하는 문제
- 팰린드롬인지 확인하는데 걸리는 시간 : O(N)
- 질문이 M개면 O(MN)이라는 시간이 걸림
- $1 \leq M \leq 1{,}000{,}000$, $1 \leq N \leq 2{,}000$

팰린드롬?

https://www.acmicpc.net/problem/10942

- D[i][j] = A[i] ~ A[j]가 팰린드롬이면 1, 아니면 0
- 길이가 1인 부분 수열은 반드시 팰린드롬이다
  - D[i][i] = 1
- 길이가 2인 부분 수열은 두 수가 같을 때만 팰린드롬이다
  - D[i][i+1] = 1 (A[i] == A[i+1])
  - D[i][i+1] = 0 (A[i] != A[i+1])

팰린드롬?

https://www.acmicpc.net/problem/10942

- D[i][j] = A[i] ~ A[j]가 팰린드롬이면 1, 아니면 0
- 길이가 1인 부분 수열은 반드시 팰린드롬이다
  - D[i][i] = 1
- 길이가 2인 부분 수열은 두 수가 같을 때만 팰린드롬이다
  - D[i][i+1] = 1 (A[i] == A[i+1])
  - D[i][i+1] = 0 (A[i] != A[i+1])
- A[i] ~ A[j]가 팰린드롬이 되려면, A[i] == A[j] 이어야 하고, A[i+1] ~ A[j-1]이 팰린드롬이어야 한다
  - D[i][j] = 1 (A[i] == A[j] && D[i+1][j-1] == 1)

팰린드롬?

https://www.acmicpc.net/problem/10942

- 일반적인 방식으로 배열을 채우지 않기 때문에
- 재귀 호출을 사용하는 것이 편하다
- 다음 페이지의 소스에서
- -1: 아직 채우지 않음
- 0: 팰린드롬이 아님
- 1: 팰린드롬
- 이라는 뜻이다

팰린드롬?

https://www.acmicpc.net/problem/10942

```
int go(int i, int j) {
    if (i == j) {
        return 1;
    } else if (i+1 == j) {
        if (a[i] == a[j]) return 1;
        else return 0;
    }
    if (d[i][j] > 0) return d[i][j];
    if (a[i] != a[j]) return d[i][j] = 0;
    else return d[i][j] = go(i+1,j-1);
}
```

팰린드롬?

https://www.acmicpc.net/problem/10942

- 재귀 호출을 사용하지 않고도 풀 수 있다
- 길이가 1인 D[i][j]를 채우고
- 2인 것을 채우고
- 3인 것을 채우고
- …
- N-1인 것을 채우는 방식을 이용하면
- for문으로도 채울 수 있다

팰린드롬?

https://www.acmicpc.net/problem/10942

```
for (int i=1; i<=n; i++) d[i][i] = true;
for (int i=1; i<=n-1; i++) {
    if (a[i] == a[i+1]) d[i][i+1] = true;
}
for (int k=3; k<=n; k++) {
    for (int i=1; i<=n-k+1; i++) {
        int j = i+k-1;
        if (a[i] == a[j] && d[i+1][j-1]) {
            d[i][j] = true;
        }
    }
}
```

팰린드롬?

https://www.acmicpc.net/problem/10942

- Bottom-up
  - C/C++
    - https://gist.github.com/Baekjoon/81223456ca57de96091f
  - Java
    - https://gist.github.com/Baekjoon/f1003fbe8651b961de45
- Top-down
  - C/C++
    - https://gist.github.com/Baekjoon/46d75f285e3d2312e8a3
  - Java
    - https://gist.github.com/Baekjoon/d4c90eb580674a898972

팰린드롬 분할

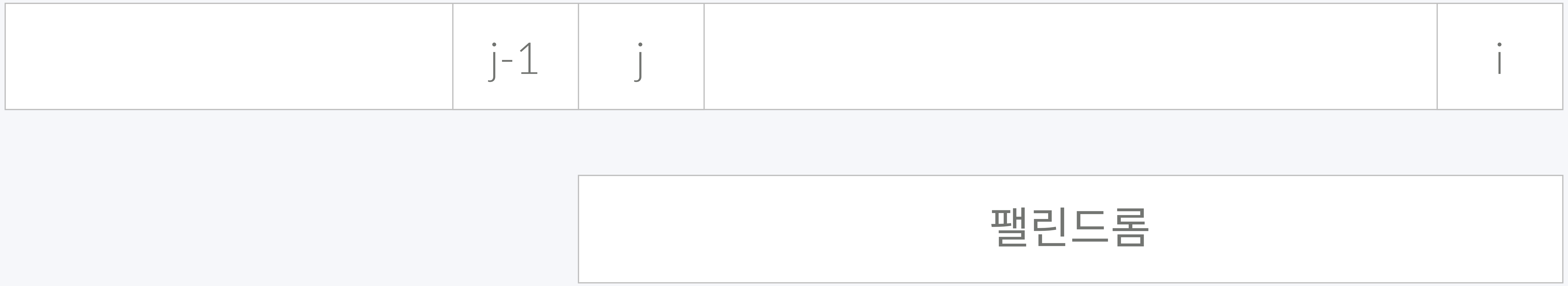
https://www.acmicpc.net/problem/1509

- 어떤 문자열을 팰린드롬으로 분할하는데 분할 개수의 최소값
- 예: ABACABA
- A, B, A, C, A, B, A
- A, BACAB, A
- ABA, C, ABA
- ABACABA

팰린드롬 분할

https://www.acmicpc.net/problem/1509

- D[i] = i번째 문자열까지를 팰린드롬 분할 했을 때, 분할의 최소 개수
- D[i] = min(D[j-1]) + 1 (i ~ j는 팰린드롬)



팰린드롬 분할

https://www.acmicpc.net/problem/1509

```
d[0] = 0;
for (int i=1; i<=n; i++) {
    d[i] = -1;
    for (int j=1; j<=i; j++) {
        if (c[j][i]) {
            if (d[i] == -1 || d[i] > d[j-1]+1) {
                d[i] = d[j-1]+1;
            }
        }
    }
}
```

팰린드롬 분할

https://www.acmicpc.net/problem/1509

- C
- https://gist.github.com/Baekjoon/10c6d1aec73e44b9a670
- C++
- https://gist.github.com/Baekjoon/608140e408836170a0ed
- Java
- https://gist.github.com/Baekjoon/5fd1bfa24a7a4c8de393

동전 1

https://www.acmicpc.net/problem/2293

- n가지 종류의 동전이 있다
- 각각의 동전이 나타내는 가치는 다르다
- 이 동전들을 적당히 사용해서, 그 가치의 합이 k원이 되도록 하고 싶다
- 그 경우의 수를 구하시오.
- 각각의 동전은 몇 개라도 사용할 수 있다.

동전 1

https://www.acmicpc.net/problem/2293

- D[i] = 동전을 적절히 사용해서 i원을 만드는 경우의 수
- 사용할 수 있는 동전: N가지 (A[1], A[2], …, A[N])
- D[i] += D[i-A[j]] $(1 \leq j \leq N)$

동전 1

https://www.acmicpc.net/problem/2293

- D[i] = 동전을 적절히 사용해서 i원을 만드는 경우의 수
- 사용할 수 있는 동전: N가지 (A[1], A[2], …, A[N])
- D[i] += D[i-A[j]] ($1 \le j \le N$)
- 위와 같은 경우에는
- 1+1+2, 1+2+1, 2+1+1을 모두 다른 경우로 처리하게 된다.

동전 1

https://www.acmicpc.net/problem/2293

- D[i][j] = A[1] ~ A[j]까지 동전을 적절히 사용해서 i원을 만드는 경우의 수
    - 현재 사용가능한 동전은 A[j]
- i원을 만드는 가능한 경우
- A[j]를 사용하는 경우
- A[j]를 사용하지 않는 경우

동전 1

https://www.acmicpc.net/problem/2293

- D[i][j] = A[1] ~ A[i]까지 동전을 적절히 사용해서 j원을 만드는 경우의 수
  - 현재 사용가능한 동전은 A[i]
- j원을 만드는 가능한 경우
- A[i]를 사용하는 경우
  - A[i]를 사용하면, A[1]~A[i-1]까지 동전을 사용해서 j-A[i]원을 만들어야 한다
  - D[i-1][j-A[i]]
- A[j]를 사용하지 않는 경우
  - 사용하지 않았기 때문에, A[1]~A[i-1]까지 동전을 사용해서 j원을 만들어야 한다
  - D[i-1][j]
- D[i][j] = D[i-1][j-A[i]] + D[i-1][j]

동전 1

https://www.acmicpc.net/problem/2293

```
d[0][0] = 1;
for (int i=1; i<=n; i++) {
    for (int j=0; j<=m; j++) {
        d[i][j] = d[i-1][j]; // 동전 사용하지 않음
        if (j-a[i] >= 0) {
            d[i][j] += d[i][j-a[i]]; // 동전 사용
        }
    }
}
```

동전 1

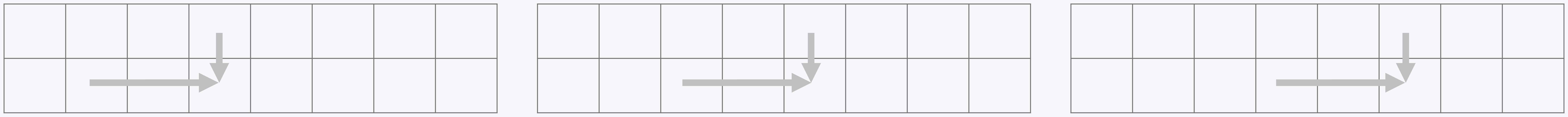
https://www.acmicpc.net/problem/2293

- D[i] = 동전을 적절히 사용해서 i원을 만드는 경우의 수
- 사용할 수 있는 동전: N가지 (A[1], A[2], …, A[N])
- D[i] += D[i-A[j]] $(1 \le j \le N)$
- 위와 같은 경우에는
- 1+1+2, 1+2+1, 2+1+1을 모두 같은 경우로 처리하게 된다.
- 이런 경우를 처리하기 위해서
- A[1]로만 i원을 만들고, A[1]과 A[2]로만 만들고, …, A[1]~A[N]으로만 만들고 하는 방식으로 문제를 풀 수 있다.
- 즉, 앞의 2차원 식을 1차원으로 바꿔야 한다.

동전 1

https://www.acmicpc.net/problem/2293

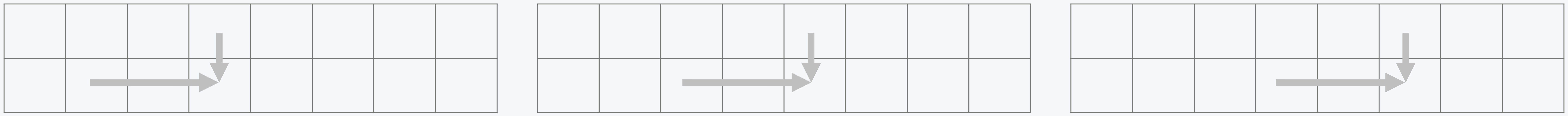
- D[i] = 동전을 적절히 사용해서 i원을 만드는 경우의 수
- 사용할 수 있는 동전: N가지 (A[1], A[2], ⋯, A[N])
- D[i] += D[i-A[j]] ($1 \le j \le N$)
- D[i][j] = A[1] ~ A[j]까지 동전을 적절히 사용해서 i원을 만드는 경우의 수
    - 현재 사용가능한 동전은 A[j]
- D[i][j] = D[i-1][j-A[i]] + D[i-1][j]
- 2차원 -> 1차원



동전 1

https://www.acmicpc.net/problem/2293

- D[i] = 동전을 적절히 사용해서 i원을 만드는 경우의 수
- 사용할 수 있는 동전: N가지 (A[1], A[2], …, A[N])
- D[i] += D[i-A[j]] ($1 \le j \le N$)
- D[i][j] = A[1] ~ A[j]까지 동전을 적절히 사용해서 i원을 만드는 경우의 수
  - 현재 사용가능한 동전은 A[j]
- D[i][j] = D[i-1][j-A[i]] + D[i-1][j]
- 2차원 -> 1차원 (아래 그림을 보면 위쪽 배열 D[i-1][j] 은 D[i][j]에 추가되는 역할만 한다)



동전 1

https://www.acmicpc.net/problem/2293

```
d[0] = 1;
for (int i=1; i<=n; i++) {
    for (int j=0; j<=m; j++) {
        if (j-a[i] >= 0) {
            d[j] += d[j-a[i]];
        }
    }
}
```

동전 1

https://www.acmicpc.net/problem/2293

- C/C++
  - https://gist.github.com/Baekjoon/64855ce0c5f17cf40901
- Java
  - https://gist.github.com/Baekjoon/0bfa53f6fee1ef1b6b9d

동전 2

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

- n가지 종류의 동전이 있다
- 각각의 동전이 나타내는 가치는 다르다
- 이 동전들을 적당히 사용해서, 그 가치의 합이 k원이 되도록 하고 싶다
- 그러면서 동전의 개수가 최소가 되도록 하려고 한다
- 각각의 동전은 몇개라도 사용할 수 있다

동전 2

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

- 동전 1과 비슷한 방법으로 풀 수 있다
- D[i] = i원을 만드는데 필요한 동전의 최소 개수

동전 2

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

```
for (int i=0; i<=m; i++) {
    d[i] = -1;
}
d[0] = 0;
for (int i=1; i<=n; i++) {
    for (int j=0; j<=m; j++) {
        if (j-a[i] >= 0 && d[j-a[i]] != -1) {
            if (d[j] == -1 || d[j] > d[j-a[i]]+1) {
                d[j] = d[j-a[i]] + 1;
            }
        }
    }
}
```

동전 2

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

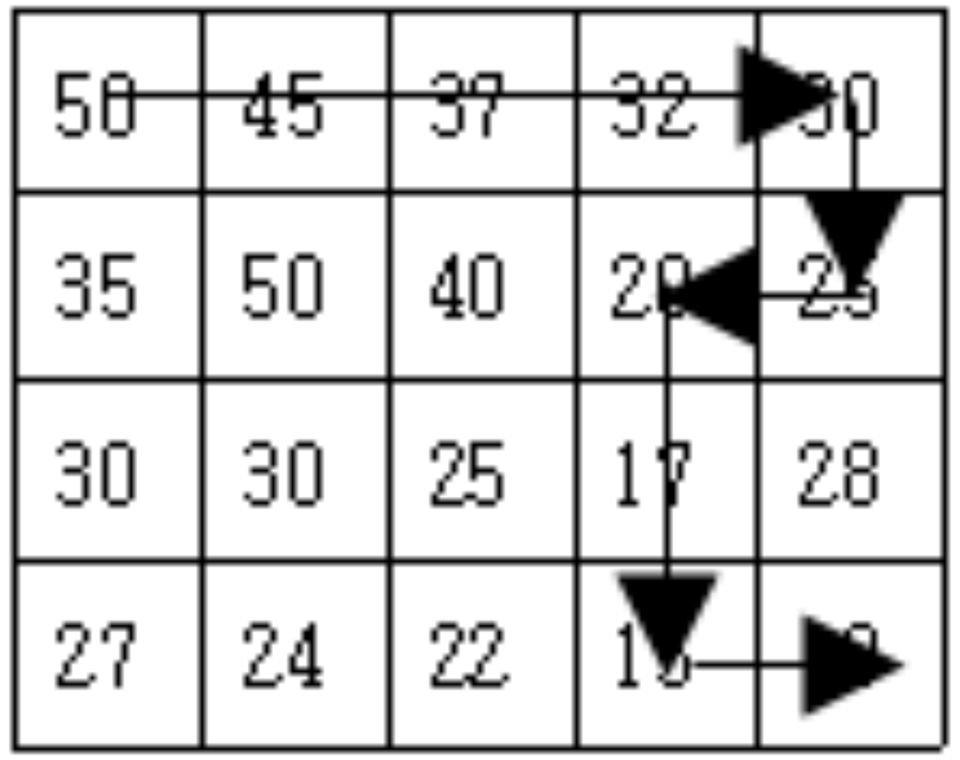
- C/C++
  - https://gist.github.com/Baekjoon/4a581d812b2892f5d85f
- Java
  - https://gist.github.com/Baekjoon/3db9cc732a3234efe680

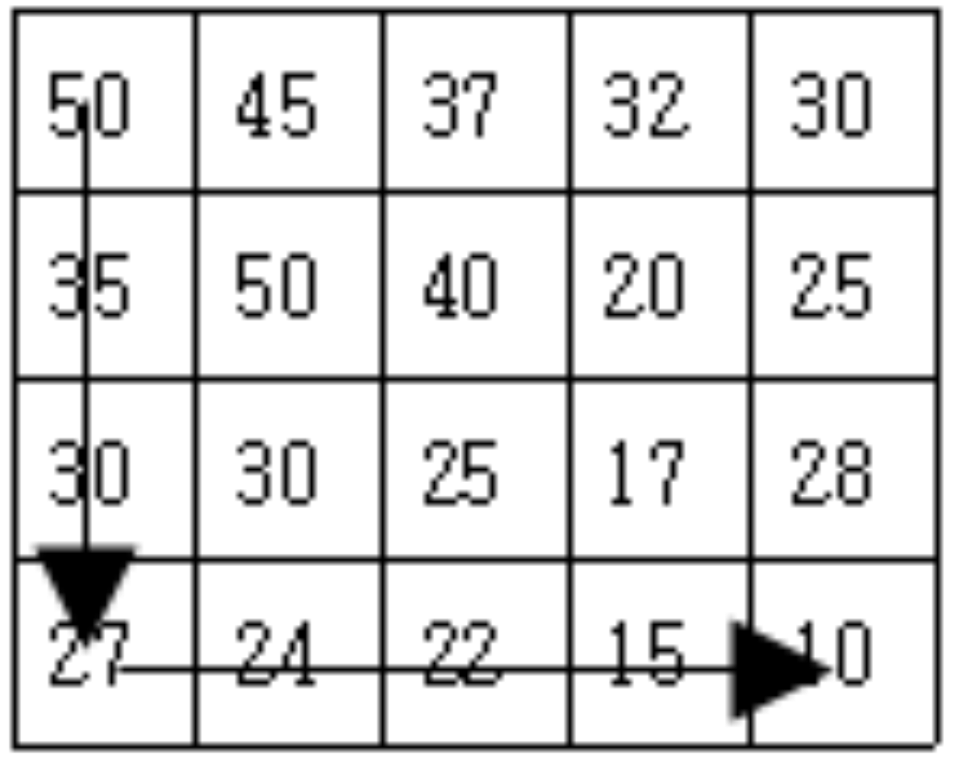
내리막 길

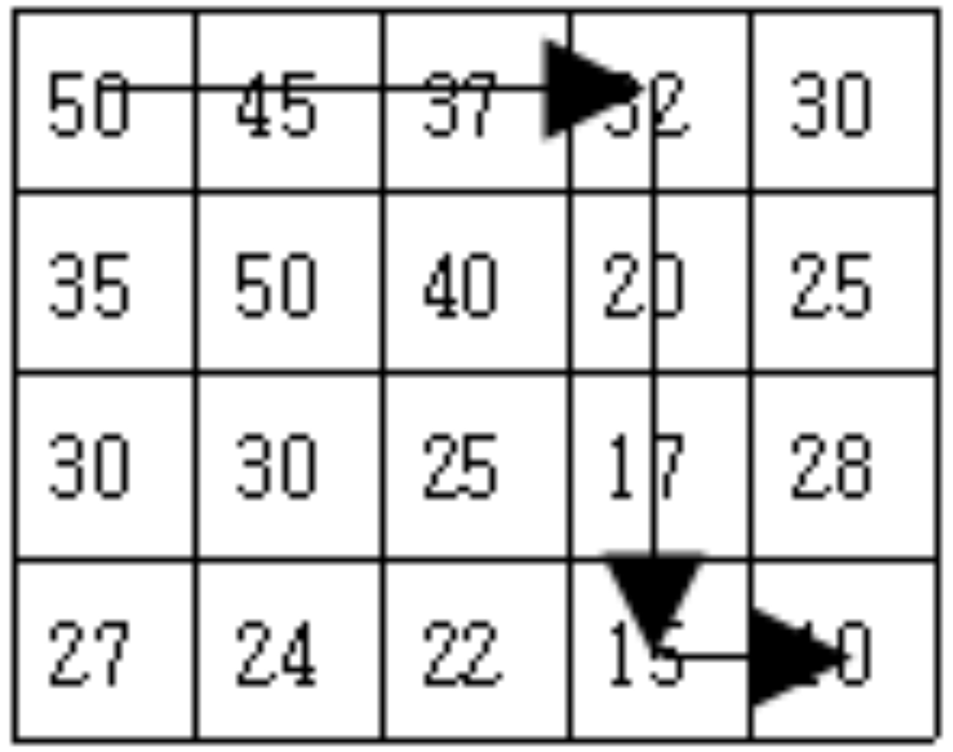
https://www.acmicpc.net/problem/1520

- N*M 크기의 지도
- (1,1)에서 시작해서 (N,M)로 가는 내리막 길의 개수

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







내리막 길

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

- D[i][j] = (i,j)에서 시작해서 (N,M)로 가는 내리막 길의 개수
- D[N][M] = 1
- 이동하는 방향이 4방향이다.
- 이동하기와 다르게 문제의 크기가 줄어들지 않는다
- 하지만, 수가 감소하는 방향으로만 이동할 수 있기 때문에, 사이클은 생기지 않는다
- D[i][j] += D[x][y]
- (i,j) -> (x,y)로 이동할 수 있어야 함

내리막 길

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

```
int go(int x, int y) {
    if (x == n-1 && y == m-1) return 1;
    if (d[x][y]) return d[x][y];
    for (int k=0; k<4; k++) {
        int nx = x+dx[k];
        int ny = y+dy[k];
        if (0 <= nx && nx < n && 0 <= ny && ny < m) {
            if (a[x][y] > a[nx][ny]) d[x][y] += go(nx,ny);
        }
    }
    return d[x][y];
}
```

내리막 길

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

- Top-down
  - C/C++
    - https://gist.github.com/Baekjoon/1a2832a35769675b6727
  - Java
    - https://gist.github.com/Baekjoon/a8466ac399855aec2258
- Bottom-up
  - C/C++
    - https://gist.github.com/Baekjoon/6d6e66747da68ced6b2b

파일 합치기

https://www.acmicpc.net/problem/11066

- 각 장이 쓰여진 파일을 합쳐서 최종적으로 소설의 완성본이 들어있는 한 개의 파일을 만든
- 이 과정에서 두 개의 파일을 합쳐서 하나의 임시파일을 만들고, 이 임시파일이나 원래의 파일을 계속 두 개씩 합쳐서 소설의 여러 장들이 연속이 되도록 파일을 합쳐나가고, 최종적으로는 하나의 파일로 합친다
- 두 개의 파일을 합칠 때 필요한 비용(시간 등)이 두 파일 크기의 합이라고 가정할 때, 최종적인 한 개의 파일을 완성하는데 필요한 비용의 총 합

파일 합치기

https://www.acmicpc.net/problem/11066

- 연속된 파일만 합칠 수 있다
- 파일은 2개의 연속된 파일을 합치는 것이다
- N^3 다이나믹을 생각해볼 수 있다.

파일 합치기

https://www.acmicpc.net/problem/11066

- D[i][j] = i번째 장부터 j번째 장까지 합쳤을 때, 필요한 최소 비용

- i번째 장부터 k번째 장까지 합친 파일과 k+1번째 장부터 j번째 장까지 합치면 된다
- D[i][j] = D[i][k] + D[k+1][j] + 합치는 비용

파일 합치기

https://www.acmicpc.net/problem/11066

- C/C++: https://gist.github.com/Baekjoon/bb044e22a3ef51475bfe57da1cf0dfe0
- Java: https://gist.github.com/Baekjoon/af09054dd38302f80916ed2b042572d6

행렬 곱셈 순서

https://www.acmicpc.net/problem/11049

- 크기가 N×M인 행렬 A와 M×K인 B를 곱할 때 필요한 곱셈 연산의 수는 총 N×M×K번
- 행렬 N개를 곱하는데 필요한 곱셈 연산의 수는 행렬을 곱하는 순서에 따라 다르다
- A의 크기가 5×3이고, B의 크기가 3×2, C의 크기가 2×6인 경우
- (AB)C = 5×3×2 + 5×2×6 = 30 + 60 = 90
- A(BC) = 3×2×6 + 5×3×6 = 36 + 90 = 126

행렬 곱셈 순서

https://www.acmicpc.net/problem/11049

- D[i][j] = i번째 행렬부터 j번째 행렬까지 곱했을 때, 곱셈 연산의 최소값
- 행렬의 순서를 바꿀 수 없다

i		k		j

- i와 j 사이의 어딘가(k)에서 행렬을 나눠서 곱셈을 해야 한다
- (i~k까지 곱한 행렬) × (k+1~j까지 곱한 행렬)
- D[i][k] + D[k+1][j] + 행렬 곱셈에서 필요한 연산 횟수

행렬 곱셈 순서

https://www.acmicpc.net/problem/11049

- D[i][j] = i번째 행렬부터 j번째 행렬까지 곱했을 때, 곱셈 연산의 최소값
- A[i] = i번째 행렬의 크기 (A[i][0] x A[i][1])
- D[i][j] = Min(D[i][k]+D[k+1][j]+A[i][0]*A[k][1]*A[j][1])

행렬 곱셈 순서

https://www.acmicpc.net/problem/11049

```
int go(int x, int y) {
    if (d[x][y]) return d[x][y];
    if (x == y) return 0;
    if (x+1 == y) {
        return a[x][0]*a[x][1]*a[y][1];
    }
    int &ans = d[x][y];
    ans = -1;
```

행렬 곱셈 순서

https://www.acmicpc.net/problem/11049

```
    for (int k=x; k<=y-1; k++) {
        int t1 = go(x,k);
        int t2 = go(k+1,y);
        if (ans == -1 || ans > t1+t2+a[x][0]*a[k][1]*a[y][1]) {
            ans = t1+t2+a[x][0]*a[k][1]*a[y][1];
        }
    }
    return ans;
}
```

행렬 곱셈 순서

https://www.acmicpc.net/problem/11049

- C/C++
  - https://gist.github.com/Baekjoon/45c10837dadd4a2c29eb
- Java
  - https://gist.github.com/Baekjoon/f9995c7e91eea4b300b6

구간 나누기

https://www.acmicpc.net/problem/2228

- N(1≤N≤100)개의 수로 이루어진 1차원 배열이 있다
- 이 배열을 M(1≤M≤N/2 올림)개의 구간으로 나눠서 구간에 속한 수들의 총 합이 최대가 되도록 하려 한다
- 단, 다음의 조건들이 만족되어야 한다
  1. 하나의 구간은 하나 이상의 연속된 수들로 이루어진다.
  2. 두 개의 구간이 서로 겹치거나 붙어 있어서는 안 된다.
  3. M개의 구간이 모두 있어야 한다. M개 이하가 아니다.

구간 나누기

https://www.acmicpc.net/problem/2228

- D[i][j] = i개의 수를 j개의 그룹으로 나누었을 때, 합의 최대값
- i번째 수에게 가능한 경우
- i번째 수를 그룹에 추가하는 경우
- i번째 수를 그룹에 추가하지 않는 경우

구간 나누기

https://www.acmicpc.net/problem/2228

- D[i][j] = i개의 수를 j개의 그룹으로 나누었을 때, 합의 최대값
- i번째 수에게 가능한 경우
- i번째 수를 그룹에 추가하는 경우
  - 그룹의 수: 변하지 않음 M
  - i-1개의 수를 M개의 그룹으로 나누어야 함
  - D[i-1][j]
- i번째 수를 그룹에 추가하지 않는 경우
  - i번째 수를 그룹에 추가해야 함
  - 어디서 부터 그룹에 추가해야 할지 결정해야 함. (k번째 수 부터 그룹에 추가)
  - D[k-2][j-1] + (A[k] + … + A[i]) (k-2인 이유는 붙어있으면 안되기 때문)

구간 나누기

https://www.acmicpc.net/problem/2228

```
int go(int n, int m) {
    if (m == 0) return 0;
    if (n <= 0) return min;
    if (c[n][m]) return d[n][m];
    c[n][m] = true;
    int &ans = d[n][m];
    ans = go(n-1,m);
    for (int i=1; i<=n; i++) {
        int temp = go(i-2, m-1) + s[n]-s[i-1];
        if (ans < temp) ans = temp;
    }
    return ans;
}
```

구간 나누기

https://www.acmicpc.net/problem/2228

- C/C++
  - https://gist.github.com/Baekjoon/a751e2551ca4577dc1a5
- Java
  - https://gist.github.com/Baekjoon/30a479acead102c97c28

자두나무

https://www.acmicpc.net/problem/2240

- 매 초마다, 두 개의 나무 중 하나의 나무에서 열매가 떨어지게 된다
- 만약 열매가 떨어지는 순간, 자두가 그 나무의 아래에 서 있으면 자두는 그 열매를 받을 수 있다
- 열매는 T($1 \leq T \leq 1,000$)초 동안 떨어지게 된다
- 자두는 최대 W($1 \leq W \leq 30$)번만 움직이고 싶어 한다
- 매 초마다 어느 나무에서 열매가 떨어질지에 대한 정보가 주어졌을 때, 자두가 받을 수 있는 열매 개수 최대값

자두나무

https://www.acmicpc.net/problem/2240

- D[sec][turn] = sec에 turn번 움직여서 받을 수 있는 열매의 최대 개수
- $1 \leq sec \leq T$
- $0 \leq turn \leq W$
- 지금 위치 = turn % 2 + 1번 나무

자두나무

https://www.acmicpc.net/problem/2240

- D[sec][turn] = sec에 turn번 움직여서 받을 수 있는 열매의 최대 개수
- 움직이지 않는 경우
- 움직이는 경우
- 움직이는 경우와 상관없이 위치만 같으면 열매를 받을 수 있다

자두나무

https://www.acmicpc.net/problem/2240

- D[sec][turn] = sec에 turn번 움직여서 받을 수 있는 열매의 최대 개수
- 움직이지 않는 경우
  - D[sec+1][turn]
- 움직이는 경우
  - D[sec+1][turn+1]

자두나무

https://www.acmicpc.net/problem/2240

- D[sec][turn] = sec에 turn번 움직여서 받을 수 있는 자두의 최대 개수
- 움직이지 않는 경우
  - D[sec+1][turn]
- 움직이는 경우
  - D[sec+1][turn+1]

- where = turn % 2 + 1
- 열매를 받을 수 있는 경우는 a[pos] == where

자두나무

https://www.acmicpc.net/problem/2240

```
int go(int pos, int turn) {
    if (pos == n+1 && turn <= m) return 0;
    if (turn > m) return 0;
    if (d[pos][turn] != -1) {
        return d[pos][turn];
    }
    int where = turn % 2 + 1;
    d[pos][turn] = max(go(pos+1, turn), go(pos+1, turn+1)) + (where
== a[pos] ? 1 : 0);
    return d[pos][turn];
}
```

자두나무

https://www.acmicpc.net/problem/2240

- 가장 처음에 호출해야 하는 값은 2개이다.
- 1에서 시작하는 경우
    - go(1, 0)
- 2에서 시작하는 경우
    - go(1, 1)

```
memset(d,-1,sizeof(d));
printf("%d\n",max(go(1,0), go(1,1)));
```

자두나무

https://www.acmicpc.net/problem/2240

- C/C++
  - https://gist.github.com/Baekjoon/81d5dc3a75e0e242dcff
- Java
  - https://gist.github.com/Baekjoon/abd0254f31b3e036bb3e

고층 빌딩

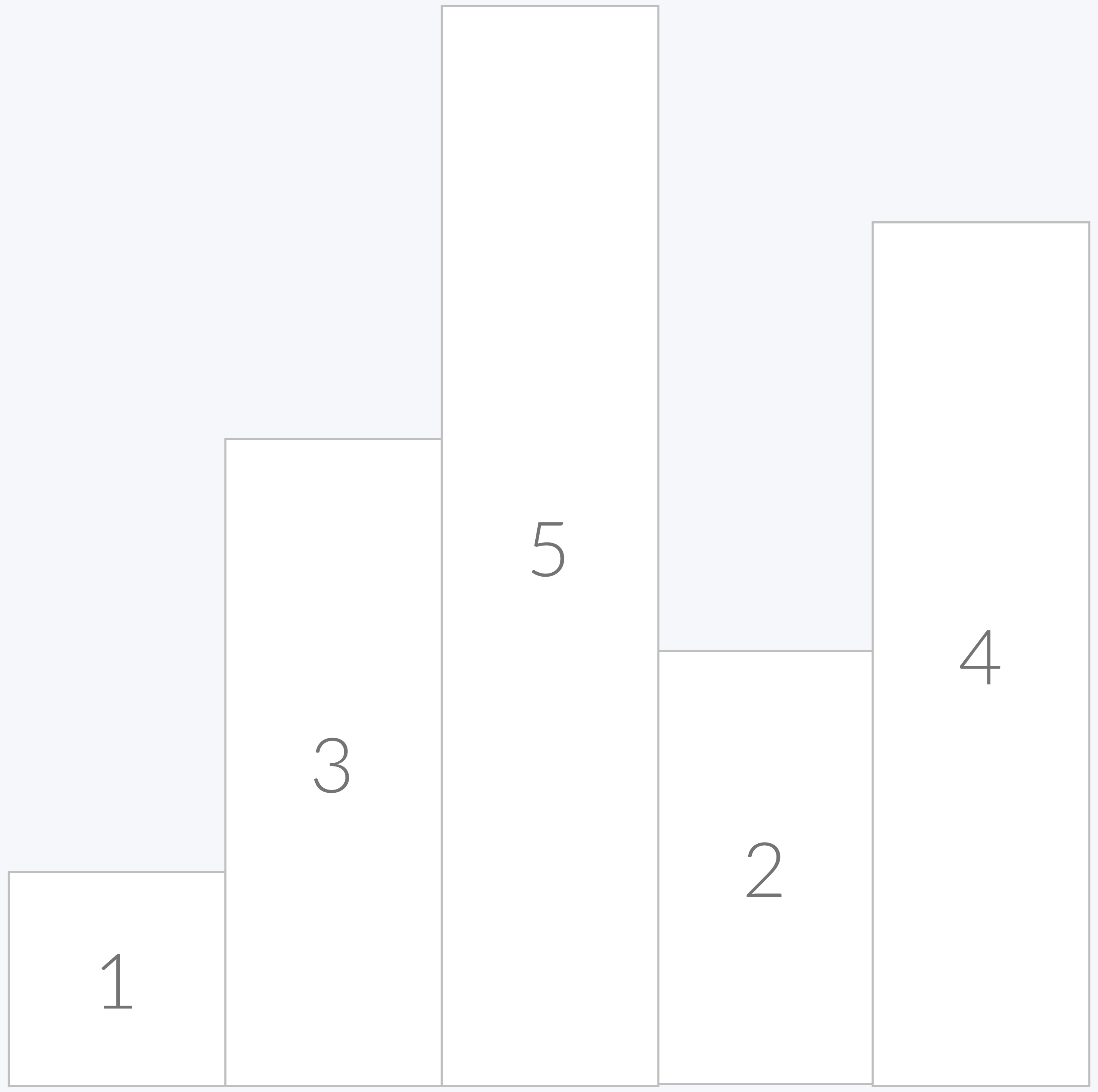
https://www.acmicpc.net/problem/1328

- 빌딩: N개, 높이 1~N
- 빌딩의 개수 N
- 가장 왼쪽에서 봤을 때 보이는 빌딩의 수 L
- 가장 오른쪽에서 봤을 때 보이는 빌딩의 수 R이 주어졌을 때
- 가능한 빌딩 순서의 경우의 수

고층 빌딩

https://www.acmicpc.net/problem/1328

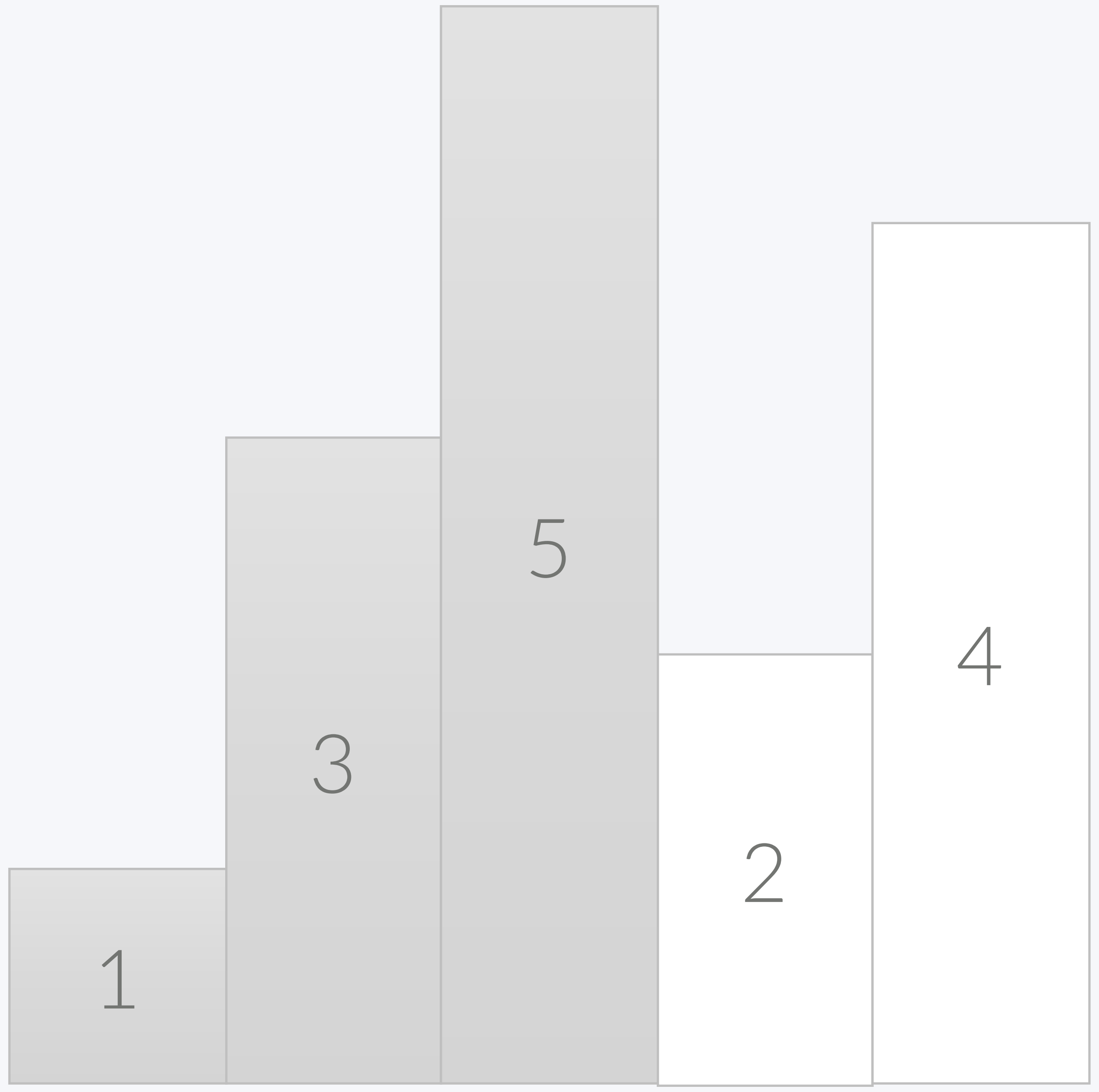
- N = 5, L = 3, R = 2인 경우 가능한 배치



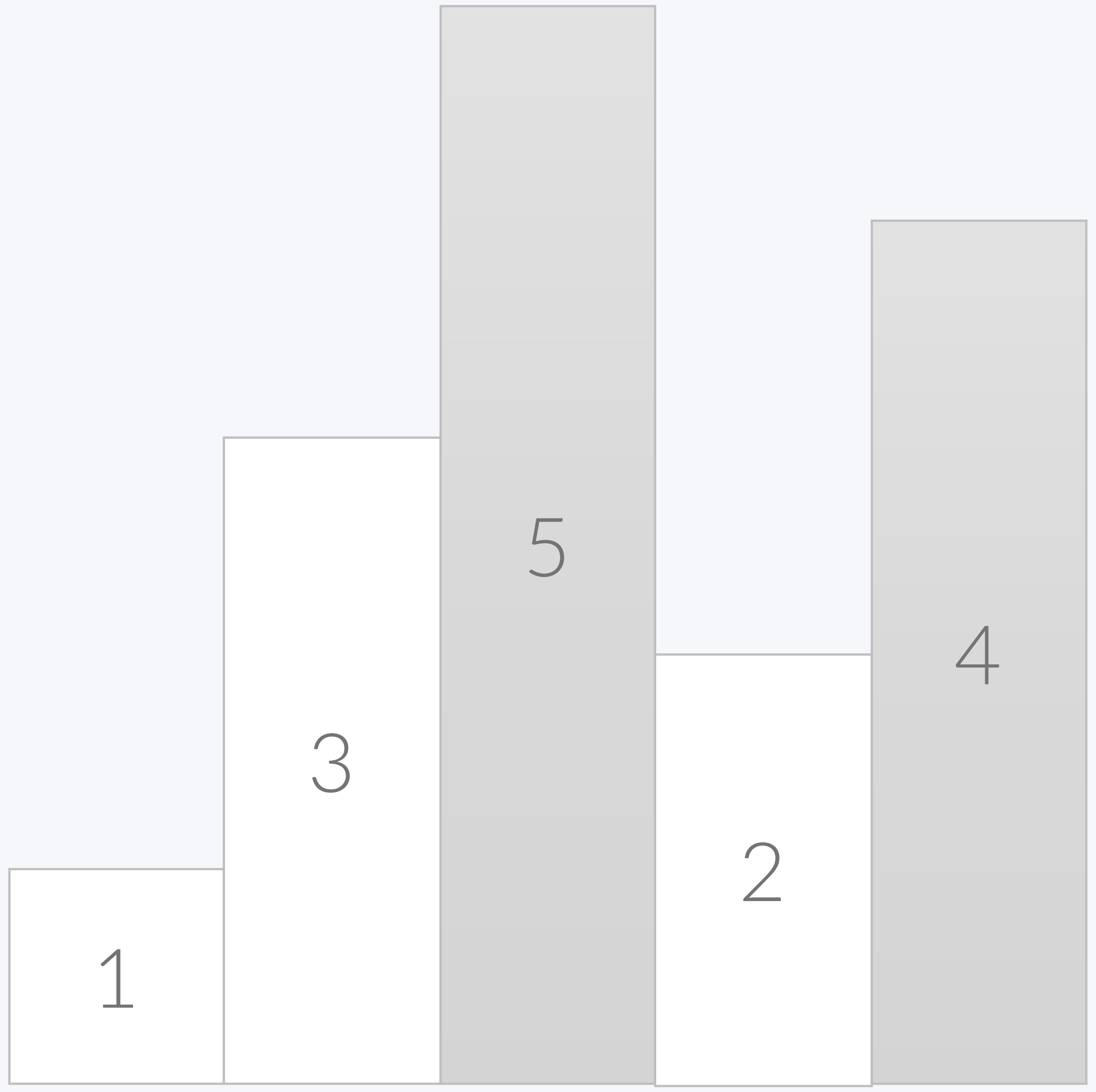
고층 빌딩

https://www.acmicpc.net/problem/1328

- N = 5, L = 3, R = 2인 경우 가능한 배치



왼쪽에서 봤을 때 3개



오른쪽에서 봤을 때 2개

고층 빌딩

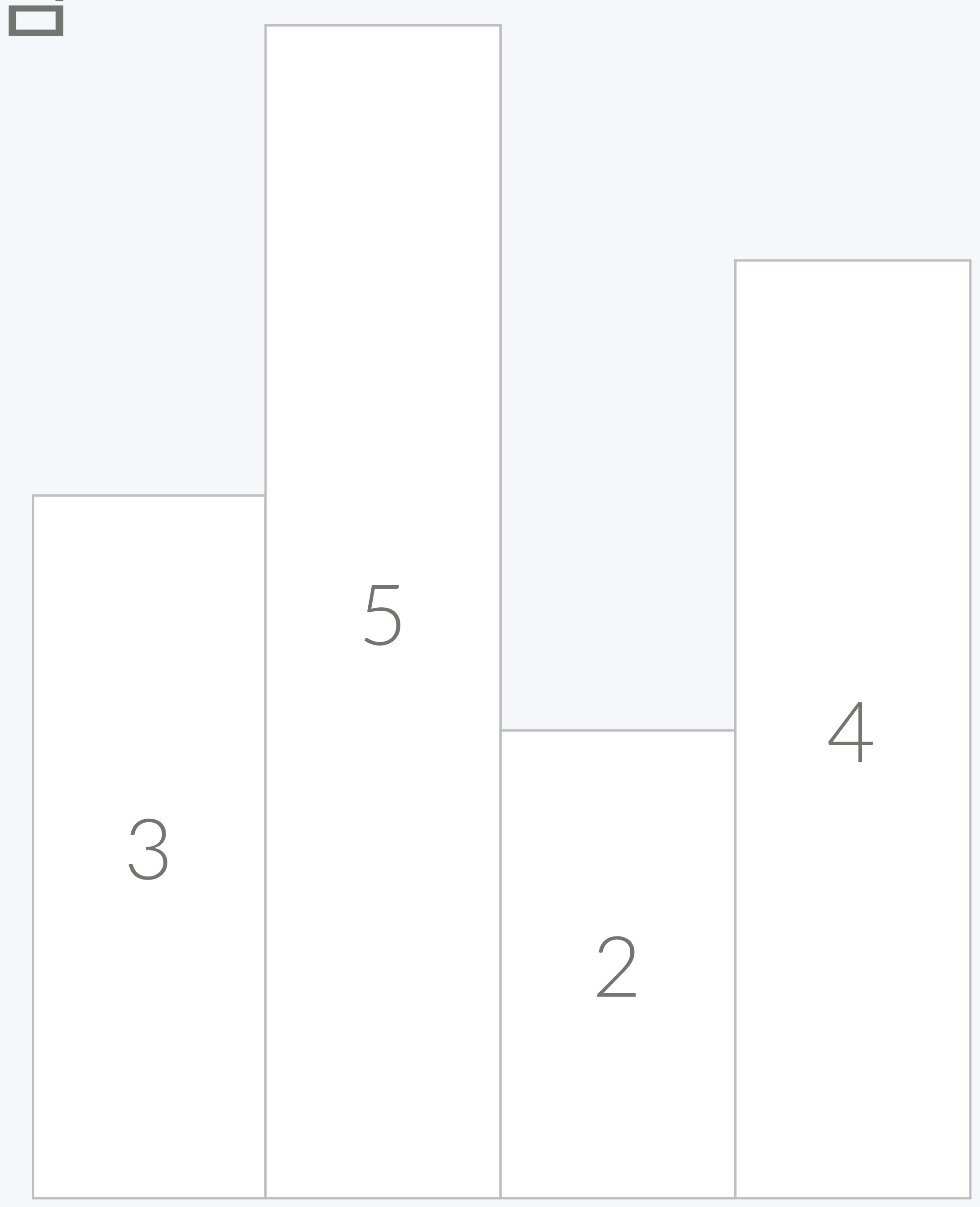
https://www.acmicpc.net/problem/1328

- D[N][L][R] = 높이가 1~N인 빌딩 N개, 왼쪽에서 L개 보임, 오른쪽에서 R개 보이는 빌딩 배치의 개수
- 빌딩 2~N까지 이미 세워져있고, 여기에 높이가 1인 빌딩을 추가하는 방식으로 문제를 풀 수 있다.
- 빌딩 2~N까지 모두 세워져 있다.
- 여기에 높이가 1인 빌딩을 추가한다
- 2~N까지 모두 세워져 있을 때,빌딩을 추가하는 방법의 수 N개

고층 빌딩

https://www.acmicpc.net/problem/1328

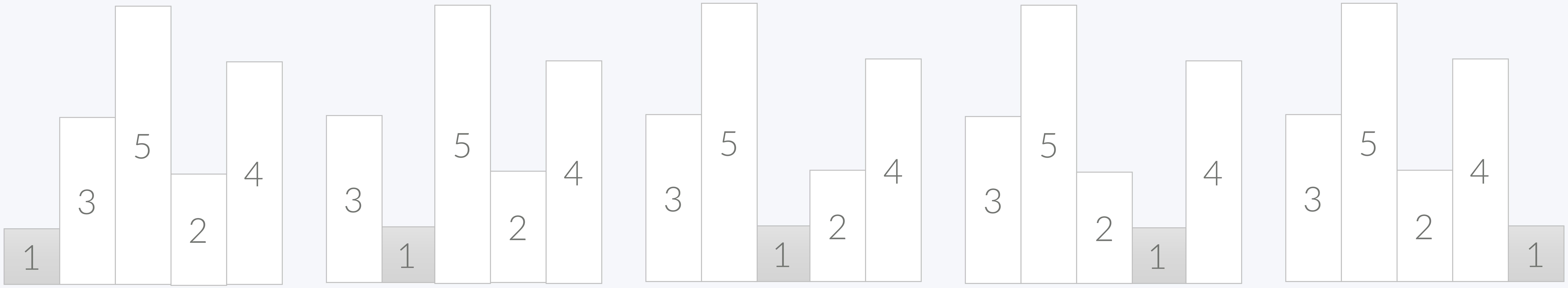
- 2~5까지 빌딩이 모두 있을 때, 높이가 1인 빌딩을 추가하는 방법
- 빌딩은 3, 5, 2, 4로 세워져 있다고 가정
- 왼쪽에서 2개, 오른쪽에서 2개 보임



고층 빌딩

https://www.acmicpc.net/problem/1328

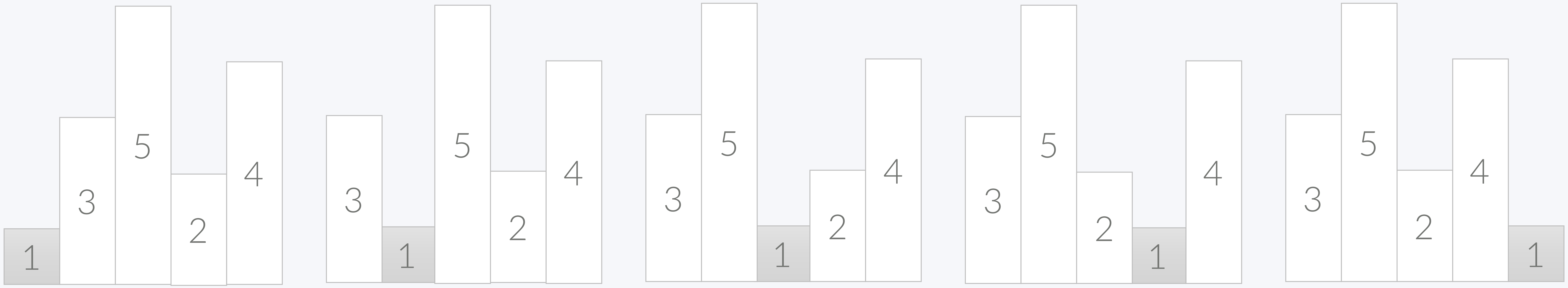
- 2~5까지 빌딩이 모두 있을 때, 높이가 1인 빌딩을 추가하는 방법
- 빌딩은 3, 5, 2, 4로 세워져 있다고 가정
- 왼쪽에서 2개, 오른쪽에서 2개 보임



고층 빌딩

https://www.acmicpc.net/problem/1328

- 2~5까지 빌딩이 모두 있을 때, 높이가 1인 빌딩을 추가하는 방법
- 빌딩은 3, 5, 2, 4로 세워져 있다고 가정
- 왼쪽에서 2개, 오른쪽에서 2개 보임
- N=4, L=2, R=2



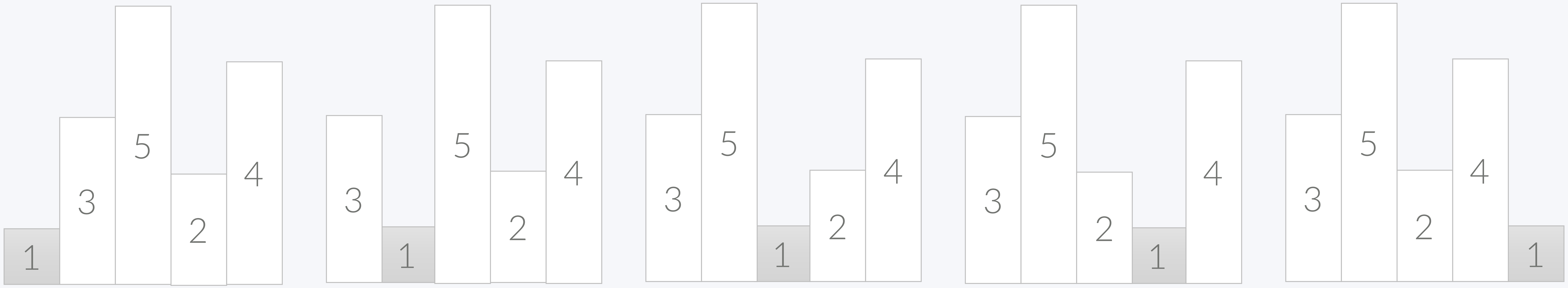
N=5, L=3, R=2 | N=5, L=2, R=2 | N=5, L=2, R=2 | N=5, L=2, R=2 | N=5, L=2, R=3

N+=1, L+=1 | N+=1 | N+=1 | N+=1 | N+=1, R+=1

고층 빌딩

https://www.acmicpc.net/problem/1328

- 가운데 끼워넣는 경우는 L과 R이 변하지 않는다는 사실을 알 수 있다
- 가장 앞에 넣는 경우는 왼쪽에서 보이는 것이 하나 증가한다
- 가장 뒤는 오른쪽에서 보이는 것이 하나 증가한다

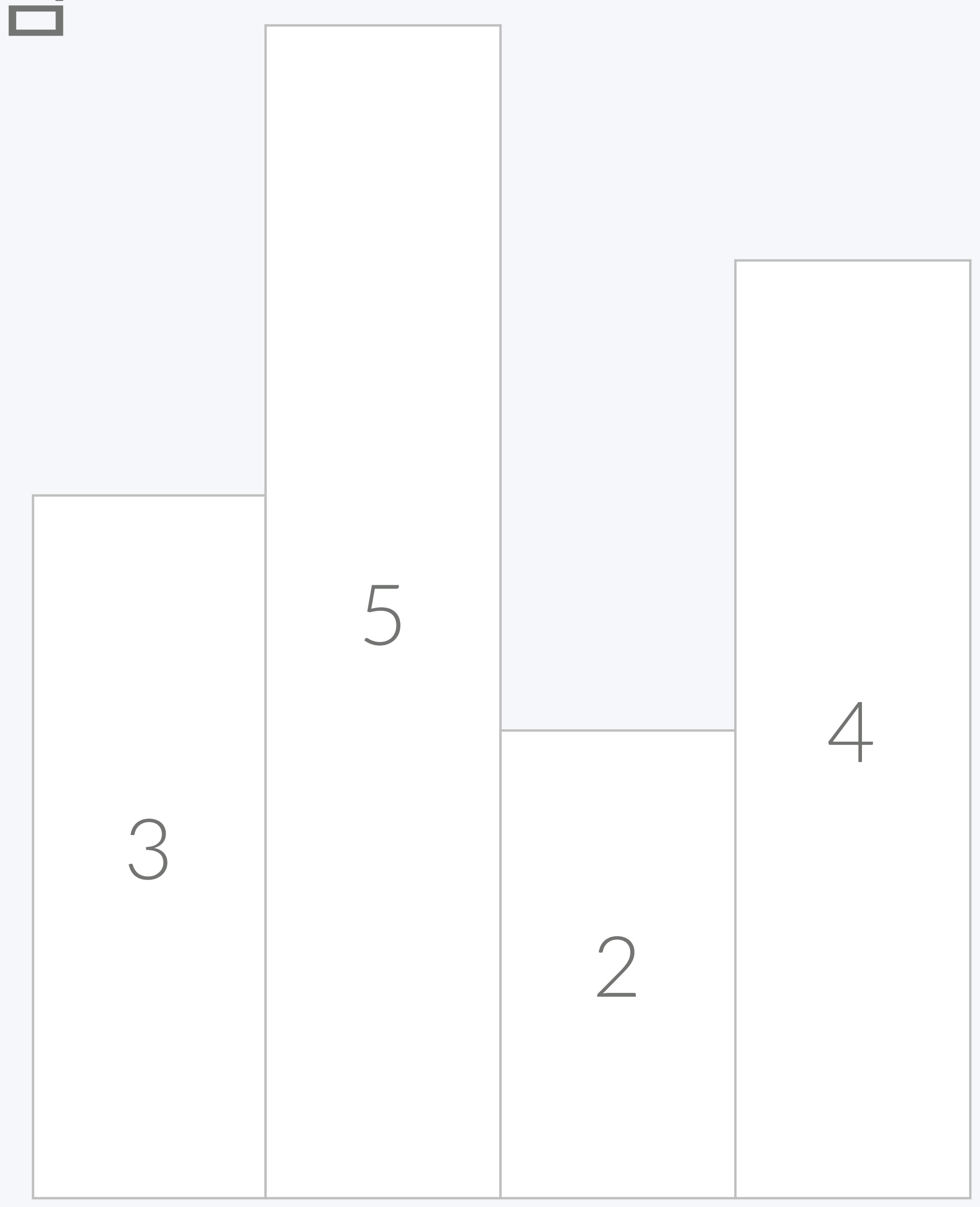


N=5, L=3, R=2	N=5, L=2, R=2	N=5, L=2, R=2	N=5, L=2, R=2	N=5, L=2, R=3
N+=1, L+=1	N+=1	N+=1	N+=1	N+=1, R+=1

고층 빌딩

https://www.acmicpc.net/problem/1328

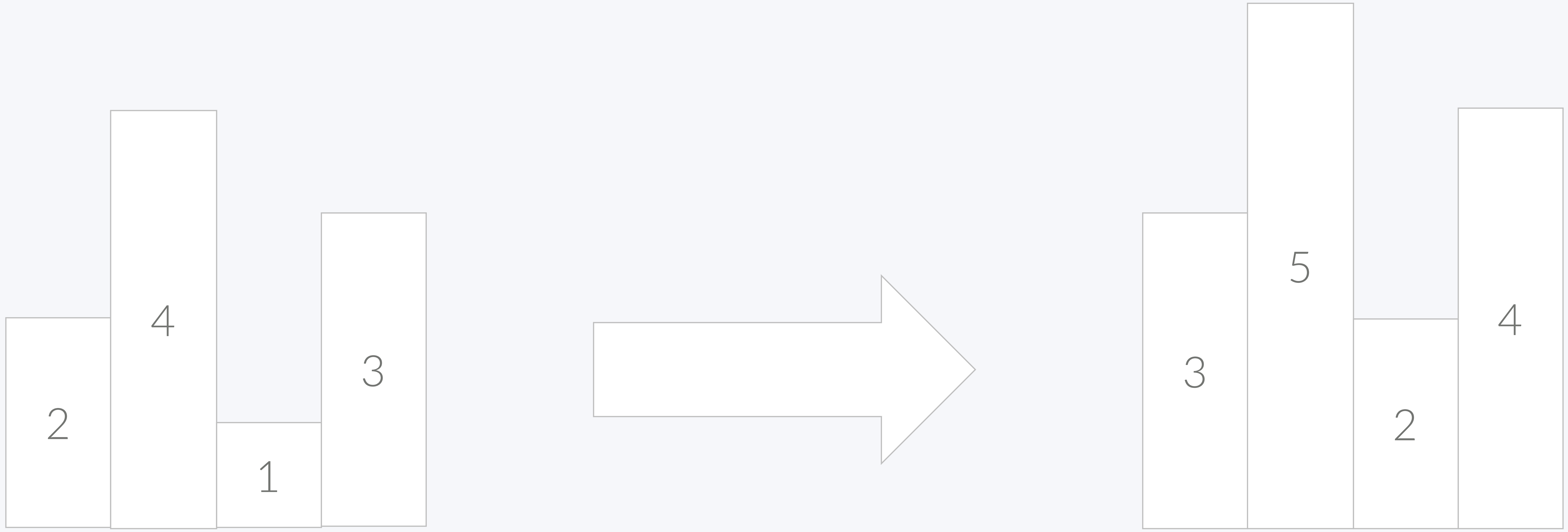
- 2~5까지 빌딩이 모두 있을 때, 높이가 1인 빌딩을 추가하는 방법
- 빌딩은 3, 5, 2, 4로 세워져 있다고 가정
- 왼쪽에서 2개, 오른쪽에서 2개 보임
- 이건 사실



고층 빌딩

https://www.acmicpc.net/problem/1328

- 2~5까지 빌딩이 모두 있을 때, 높이가 1인 빌딩을 추가하는 방법
- 빌딩은 3, 5, 2, 4로 세워져 있다고 가정
- 1~4까지 빌딩이 있고, 빌딩이 2, 4, 1, 3으로 세워져 있는 경우와 같다
- 이 경우에 모든 빌딩에 높이를 1씩 증가시키면, 같은 경우가 된다.



고층 빌딩

https://www.acmicpc.net/problem/1328

- D[N][L][R] = 빌딩 N개, 왼쪽에서 L개 보임, 오른쪽에서 R개 보이는 빌딩 배치의 개수
- 가장 왼쪽에 빌딩 1을 추가하는 경우
  - L이 하나 증가해야 하기 때문에
  - D[N+1][L+1][R] += D[N][L][R]
- 가장 오른쪽에 빌딩 1을 추가하는 경우
  - R이 하나 증가해야 하기 때문에
  - D[N+1][L][R+1] += D[N][L][R]
- 사이에 빌딩 1을 추가하는 경우
  - D[N+1][L][R] += D[N][L][R] * (N-1)
  - 추가할 수 있는 경우가 N-1개 존재

고층 빌딩

https://www.acmicpc.net/problem/1328

- D[N][L][R] = 빌딩 N개, 왼쪽에서 L개 보임, 오른쪽에서 R개 보이는 빌딩 배치의 개수
- D[N+1][L+1][R] += D[N][L][R]
- D[N+1][L][R+1] += D[N][L][R]
- D[N+1][L][R] += D[N][L][R] * (N-1)

고층 빌딩

https://www.acmicpc.net/problem/1328

- C/C++
  - https://gist.github.com/Baekjoon/19863841d081bcc33991abd874d6dab8

고층 빌딩

https://www.acmicpc.net/problem/1328

- D[N][L][R] = 빌딩 N개, 왼쪽에서 L개 보임, 오른쪽에서 R개 보이는 빌딩 배치의 개수
- 가장 왼쪽에 빌딩 1이 있는 경우
  - L이 하나 증가해야 하기 때문에
  - D[N-1][L-1][R]
- 가장 오른쪽에 빌딩 1이 있는 경우
  - R이 하나 증가해야 하기 때문에
  - D[N-1][L][R-1]
- 사이에 빌딩 1이 있는 경우
  - D[N-1][L][R] * (N-2)
  - 추가할 수 있는 경우가 N-2개 존재

고층 빌딩

https://www.acmicpc.net/problem/1328

- D[N][L][R] = 빌딩 N개, 왼쪽에서 L개 보임, 오른쪽에서 R개 보이는 빌딩 배치의 개수
- D[N][L][R] = D[N-1][L-1][R] + D[N-1][L][R-1] + D[N-1][L][R]*(N-2)

고층 빌딩

https://www.acmicpc.net/problem/1328

```
d[1][1][1] = 1LL;
for (int i=2; i<=n; i++) {
    for (int j=1; j<=l; j++) {
        for (int k=1; k<=r; k++) {
            d[i][j][k] = d[i-1][j-1][k] + d[i-1][j][k-1] + d[i-
1][j][k] * (i-2);
            d[i][j][k] %= mod;
        }
    }
}
```

고층 빌딩

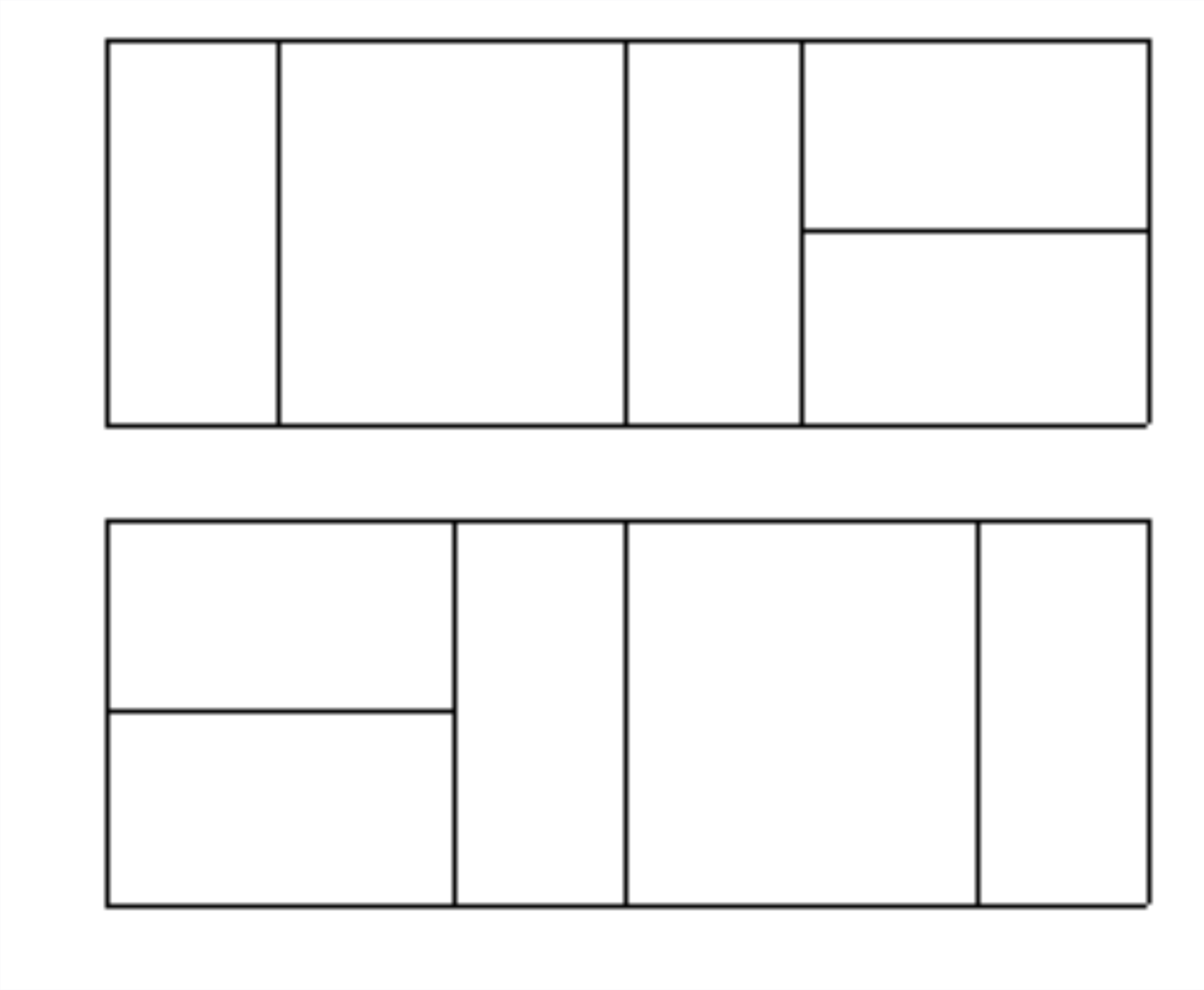
https://www.acmicpc.net/problem/1328

- C/C++
  - https://gist.github.com/Baekjoon/69a74b9d3fe008b7ce6c
- Java
  - https://gist.github.com/Baekjoon/eb36df31ebaa8dc28ed

타일 코드

https://www.acmicpc.net/problem/1720

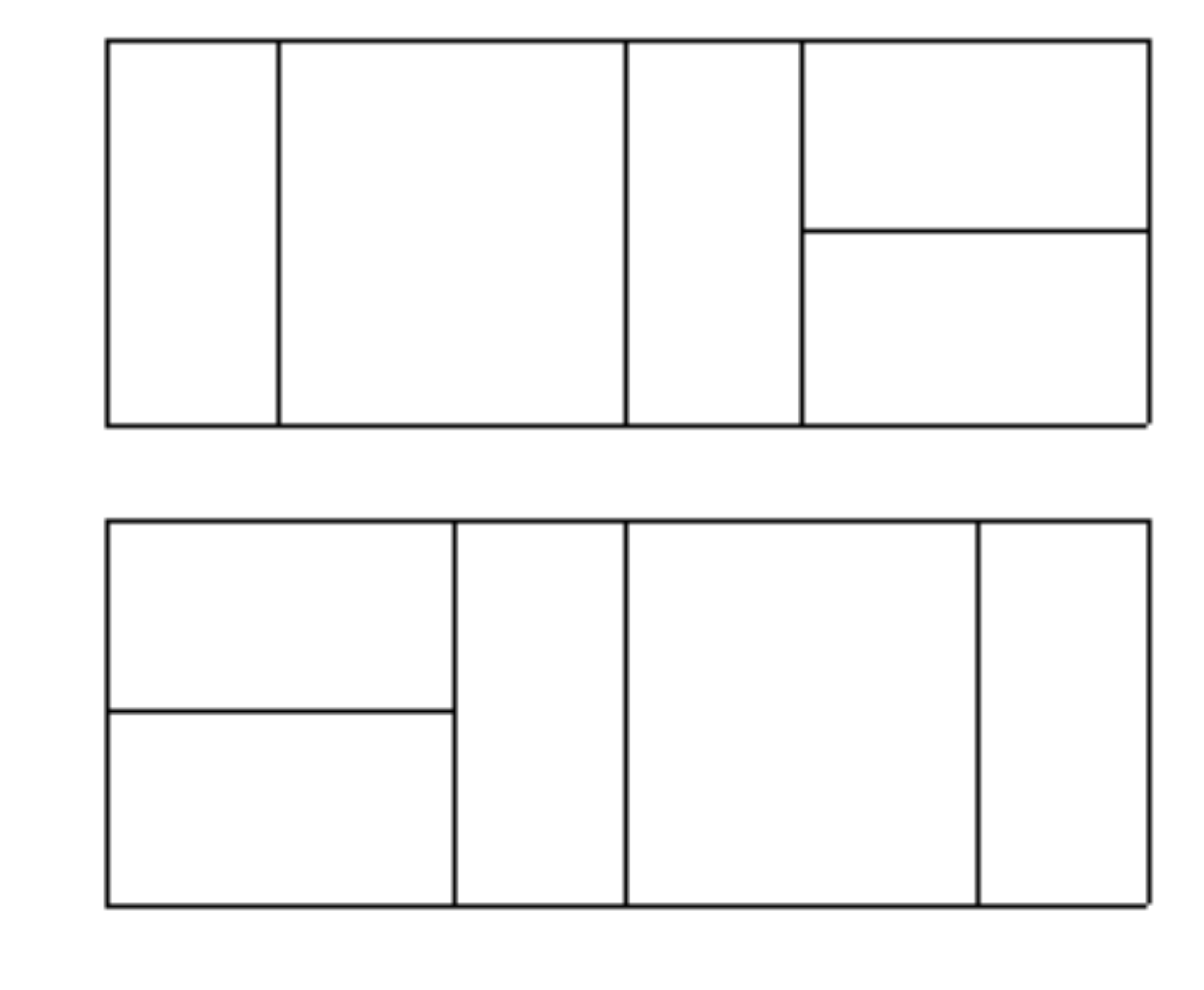
- 2×N 크기의 넓은 판을 1×2 (또는 2×1) 크기와 2×2 크기의 타일로 채우려고 한다.
- N이 주어지면, 서로 좌우 대칭을 이루는 경우를 제외한 타일 코드의 개수를 구하는 프로그램을 작성하시오.



타일 코드

https://www.acmicpc.net/problem/1720

- 2×N 크기의 넓은 판을 1×2 (또는 2×1) 크기와 2×2 크기의 타일로 채우려고 한다.
- N이 주어지면, 서로 좌우 대칭을 이루는 경우를 제외한 타일 코드의 개수를 구하는 프로그램을 작성하시오.
- 정답 = 전체 – 좌우 대칭



타일 코드

https://www.acmicpc.net/problem/1720

- 좌우 대칭

- 홀수인 경우
- D[(i-1)/2]

(i-1)/2	1	(i-1)/2

타일 코드

https://www.acmicpc.net/problem/1720

- 좌우 대칭

- 짝수인 경우
- D[(i-2)/2] * 2

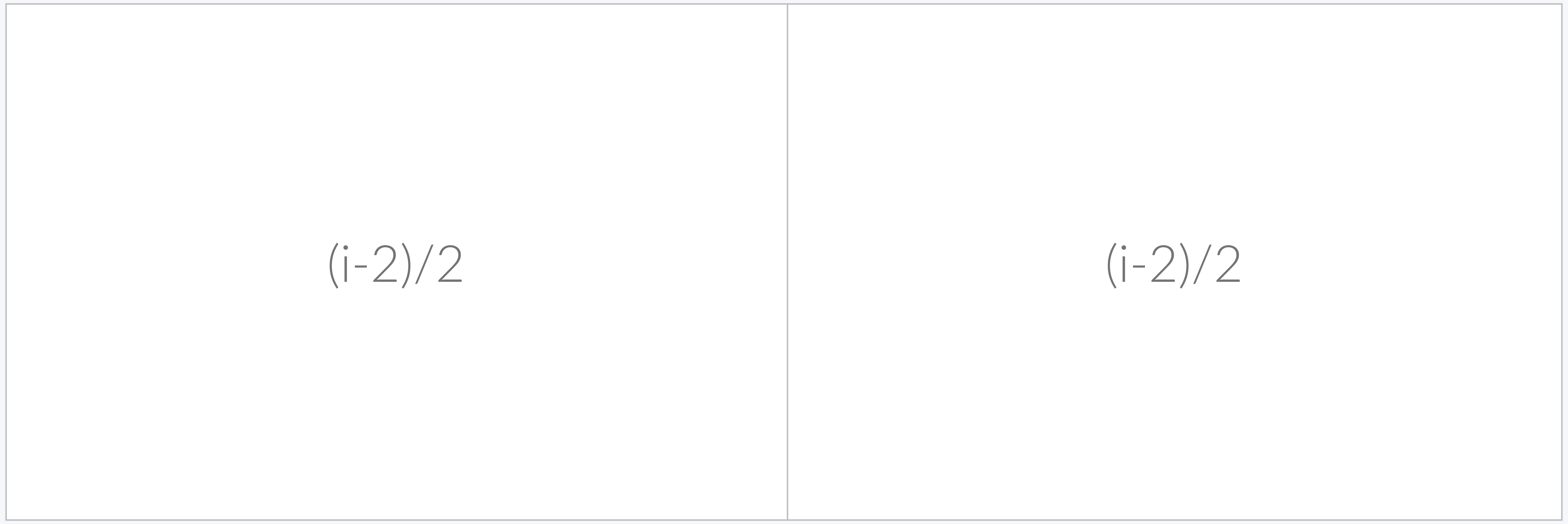
(i-2)/2	2	(i-2)/2

타일 코드

https://www.acmicpc.net/problem/1720

- 좌우 대칭

- 짝수인 경우
- D[i/2] * 2



타일 코드

https://www.acmicpc.net/problem/1720

- 좌우 대칭
- 홀수인 경우
  - B = D[(i-1)/2]
- 짝수인 경우
  - B = D[i/2] + 2*D[(i-2)/2]
- 정답
  - (D[i] + B)/2

타일 코드

https://www.acmicpc.net/problem/1720

```
a[1] = 1; // 대칭 포함
a[2] = 3;
for (int i=3; i<=30; i++) {
    a[i] = a[i-1] + a[i-2] * 2LL;
}
d[1] = 1; // 대칭 없음
d[2] = 3;
for (int i=3; i<=30; i++) {
    long long b = 0;
    if (i%2 == 1) b = a[(i-1)/2];
    else b = a[i/2] + 2*a[(i-2)/2];
    d[i] = (a[i]+b)/2;
}
```

타일 코드

https://www.acmicpc.net/problem/1720

- C/C++
  - https://gist.github.com/Baekjoon/dc863540a85773f6d8fb
- Java
  - https://gist.github.com/Baekjoon/0f4066bcb221f786b6a0

기타리스트

https://www.acmicpc.net/problem/1495

- 첫 볼륨: S
- 연주해야 하는 곡의 개수 N개
- 가능한 볼륨의 범위: 0 ~ M

- i번 곡을 연주하기 전에 볼륨을 V[i]만큼 바꿔야 한다

- i번 곡을 연주하기 직전 볼륨이 P라면
- i번 곡은 P+V[i] 또는 P-V[i] 로 연주해야 한다
- 마지막 곡을 연주할 수 있는 볼륨 중 최대값

기타리스트

https://www.acmicpc.net/problem/1495

- D[i][j] = i번 곡을 볼륨 j로 연주할 수 있으면 1 없으면 0
- N = 3, S = 5, M = 10
- V[1] = 5, V[2] = 3, V[3] = 7

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

기타리스트

https://www.acmicpc.net/problem/1495

- D[i][j] = i번 곡을 볼륨 j로 연주할 수 있으면 1 없으면 0
- N = 3, S = 5, M = 10
- V[1] = 5, V[2] = 3, V[3] = 7

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

기타리스트

https://www.acmicpc.net/problem/1495

- D[i][j] = i번 곡을 볼륨 j로 연주할 수 있으면 1 없으면 0
- N = 3, S = 5, M = 10
- V[1] = 5, V[2] = 3, V[3] = 7

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

기타리스트

https://www.acmicpc.net/problem/1495

- D[i][j] = i번 곡을 볼륨 j로 연주할 수 있으면 1 없으면 0
- N = 3, S = 5, M = 10
- V[1] = 5, V[2] = 3, V[3] = 7

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

기타리스트

https://www.acmicpc.net/problem/1495

- D[i][j] = i번 곡을 볼륨 j로 연주할 수 있으면 1 없으면 0
- N = 3, S = 5, M = 10
- V[1] = 5, V[2] = 3, V[3] = 7

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

기타리스트

https://www.acmicpc.net/problem/1495

- D[i][j] = i번 곡을 볼륨 j로 연주할 수 있으면 1 없으면 0
- D[i][j] 가 1이면
- D[i+1][j+V[i+1]]와 D[i+1][j-V[i+1]] 을 1로 만들 수 있다.

기타리스트

https://www.acmicpc.net/problem/1495

- C++
  - https://gist.github.com/Baekjoon/307d2dd070857278d8f6

1학년

https://www.acmicpc.net/problem/5557

- 마지막 두 숫자 사이에 '='을 넣고, 나머지 숫자 사이에는 '+' 또는 '-'를 넣어 등식을 만든다
- 예를 들어, "8 3 2 4 8 7 2 4 0 8 8"에서 등식 "8+3-2-4+8-7-2-4-0+8=8"을 만들 수 있다

1학년

https://www.acmicpc.net/problem/5557

- D[i][j] = i까지 수를 사용해서 j를 만드는 방법의 수

1학년

https://www.acmicpc.net/problem/5557

- $D[i][j] = D[i-1][j-A[i]] + D[i-1][j+A[i]]$

1학년

https://www.acmicpc.net/problem/5557

- C/C++: https://gist.github.com/Baekjoon/75773bafdba8a717afc6c560596ad449

올바른 괄호 문자열

https://www.acmicpc.net/problem/3012

- 만들 수 있는 괄호 문자열의 개수를 구하는 문제

- `(?([?)]?}?` -> 3개

올바른 괄호 문자열

https://www.acmicpc.net/problem/3012

- D[i][j] = i~j까지 문자열을 이용해서 만들 수 있는 올바른 괄호 문자열의 개수

올바른 괄호 문자열

https://www.acmicpc.net/problem/3012

- D[i][j] = i~j까지 문자열을 이용해서 만들 수 있는 올바른 괄호 문자열의 개수
- i번째에 있는 왼쪽 괄호 와 짝이 맞는 오른쪽 괄호의 위치를 k 라고 했을 때,
- (i+1, k-1)와 (k+1, j)로 나눌 수 있다.

올바른 괄호 문자열

https://www.acmicpc.net/problem/3012

- D[i][j] = i~j까지 문자열을 이용해서 만들 수 있는 올바른 괄호 문자열의 개수

- D[i][j] += D[i+1][k-1] * D[k+1][j]

올바른 괄호 문자열

https://www.acmicpc.net/problem/3012

- C/C++: https://gist.github.com/Baekjoon/8ba00d0064e8ca9c65f9ce8d64be5626

같은 탑

https://www.acmicpc.net/problem/1126

- N개의 조각이 주어졌을 때, 두 개의 탑을 만든다
- 이 때, 두 탑의 높이를 같게 만드려고 한다.
- 가능한 탑의 높이 중 최대값을 구하는 문제

같은 탑

https://www.acmicpc.net/problem/1126

- 모든 조각의 높이의 합은 500,000을 넘지 않는다
- 즉, 두 탑의 최대 높이는 500,000/2 = 250,000 이다.

같은 탑

https://www.acmicpc.net/problem/1126

- 각각의 조각에 대해서 다음과 같은 세 가지를 결정할 수 있다.

- 첫 번째 탑에 조각을 올려놓는다
- 두 번째 탑에 조각을 올려놓는다
- 조각을 탑 위에 올려놓지 않는다

같은 탑

https://www.acmicpc.net/problem/1126

- 문제를 일반화 할 수 있다

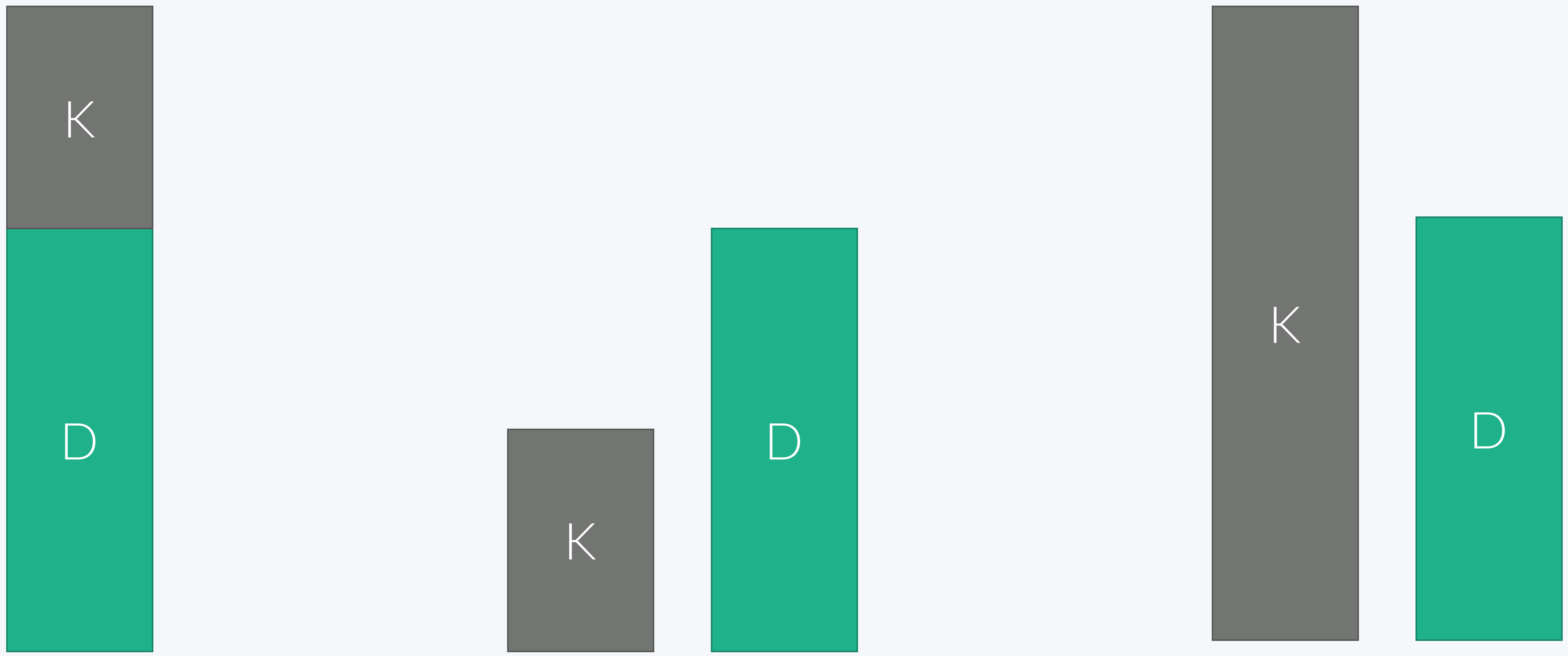
- 탑 하나의 높이는 D이고, 또 다른 탑의 높이는 0이다.
- 여기서, 조각을 적절히 놓아서 만들 수 있는 가장 큰 두 탑의 높이
- 이 때, 두 탑의 높이는 같아야 한다.

- D[N][D] = 조각이 N개 남았고, 높은 탑의 높이가 D

같은 탑

https://www.acmicpc.net/problem/1126

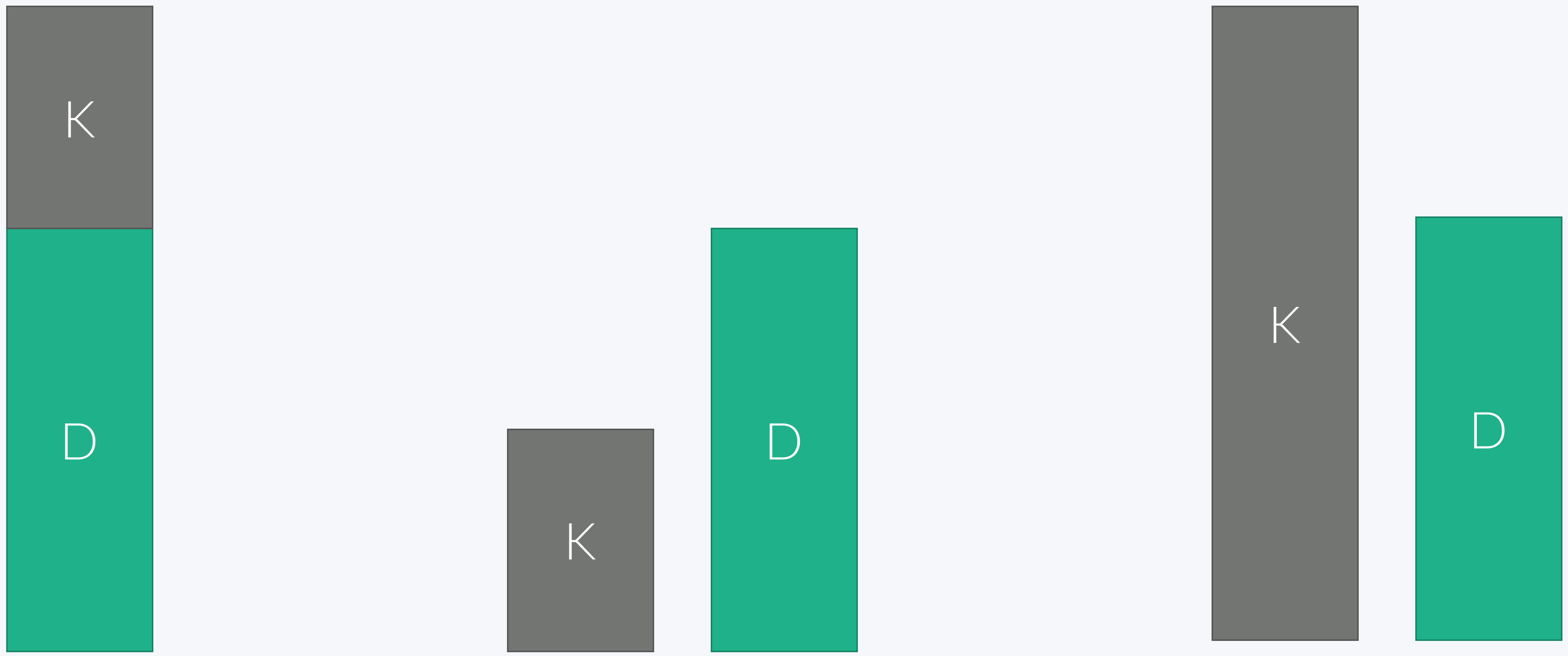
- 탑 하나의 높이는 D이고, 또 다른 탑의 높이는 0이다.
- 조각의 높이는 K이다.



같은 탑

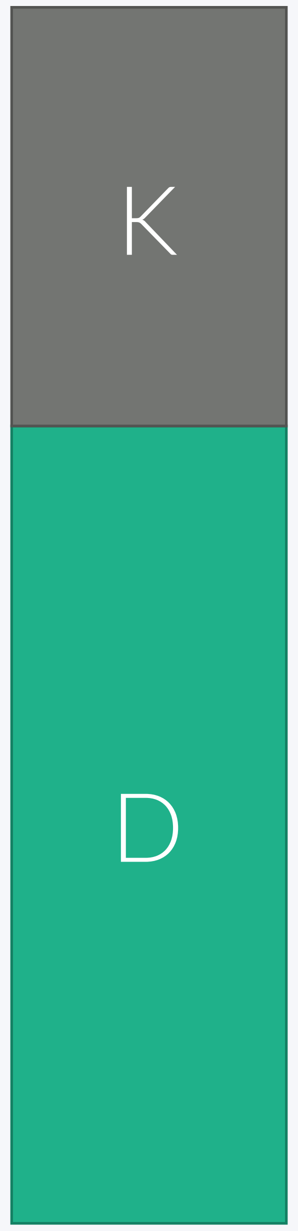
https://www.acmicpc.net/problem/1126

- 블럭을 D인 탑에 놓는 경우
- 블럭을 0인 탑에 놓는 경우

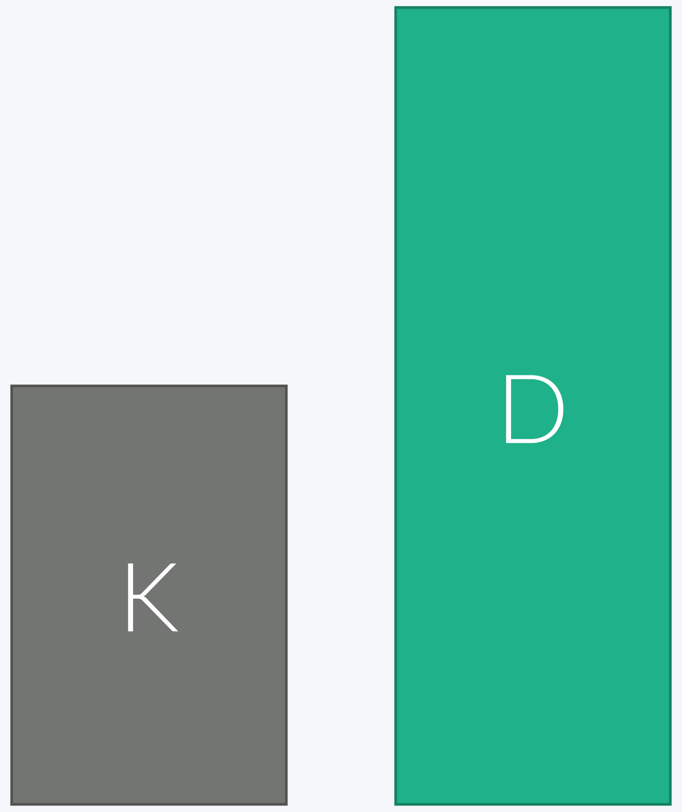


같은 탑

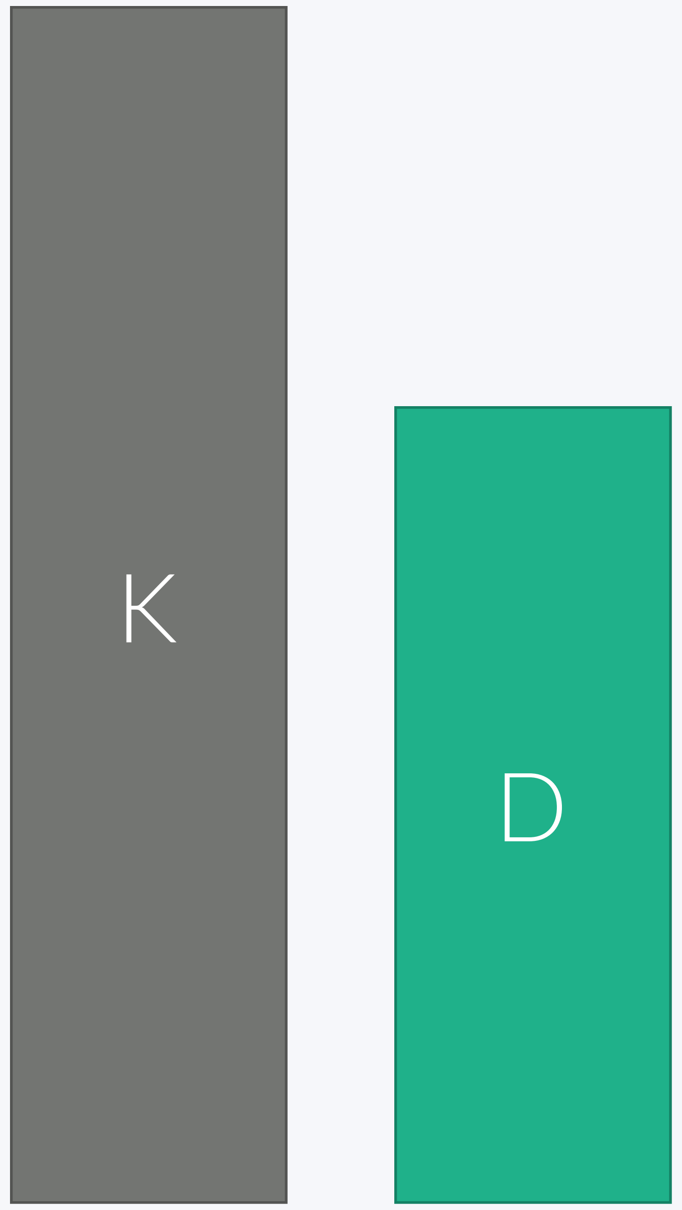
https://www.acmicpc.net/problem/1126



D[K+1][diff+A[K]]



A[K] + D[K+1][diff-A[K]]



diff+ D[K+1][A[K]-diff]

같은 탑

https://www.acmicpc.net/problem/1126

- C/C++: https://gist.github.com/Baekjoon/bfc5bacbe4fae6558cba43eb4da8ec5e