다이나믹 프로그래밍 시작하기

최백준 choi@startlink.io

소개

Introduction

- 이름: 최백준 (choi@startlink.io)
- 코딩 교육 스타트업 Startlink (https://startlink.io) 공동 창업, 대표
- 온라인 저지 Baekjoon Online Judge (https://www.acmicpc.net) 운영자
- 알고리즘 커뮤니티 Algospot (https://algospot.com) 운영진
- ACM-ICPC 2010년 The World Finals 36등
- ACM-ICPC 2009년 인도 Amritapuri 리저널 대회 1등
- 2008년 Google Codejam Semifinalist
- 2007~2010년 정보올림피아드 학원 강사

5월 알고리즘 in 서울

강의 소개

- 알고리즘을 시작하는 사람이 듣기 가장 적합한 강의
- 4월 알고리즘 강의의 후속 강의 이지만, 4월 강의를 듣지 않아도 괜찮
- 4월 강의와 같은 내용은 6월에 열립니다.
- 4/25까지 할인중
- https://offline.startlink.help/hc/ko/articles/218618857

5월 고급 알고리즘

강의 소개

- ACM-ICPC와 같은 알고리즘 대회에 상위권을 목표로 하거나
- 회사 사내 시험을 대비하는데 있어서 최고의 강의!
- 4/25까지 할인중
- https://offline.startlink.help/hc/ko/articles/218491097

5월 다이나믹 프로그래밍!

강의 소개

- 다이나믹 프로그래밍 문제 200개 이상을 푸는 강의!
- 오늘 이 강의 내용수준부터 어려운 내용까지 모두 다루는 강의
- https://offline.startlink.help/hc/ko/articles/218375018

7월 BOJ 알고리즘 캠프

강의 소개

- 10일동안 아침 10시부터 오후 9시까지 알고리즘 문제를 푸는 캠프!
- 4/30까지 할인중!
- https://offline.startlink.help/hc/ko/articles/218592697

다이나믹 프로그래밍

다이나믹 프로그래밍

Dynamic Programming

- 큰 문제를 작은 문제로 나눠서 푸는 알고리즘
- Dynamic Programming의 다이나믹은 아무 의미가 없다.
- 이 용어를 처음 사용한 1940년 Richard Bellman은 멋있어보여서 사용했다고 한다
- https://en.wikipedia.org/wiki/Dynamic_programming#History

다이나믹 프로그래밍

Dynamic Programming

- 두 가지 속성을 만족해야 다이나믹 프로그래밍으로 문제를 풀 수 있다.

1. Overlapping Subproblem
2. Optimal Substructure

Overlapping Subproblem

Overlapping Subproblem

- 피보나치 수
- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ $(n \geq 2)$

Overlapping Subproblem

Overlapping Subproblem

- 피보나치 수
- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ $(n \geq 2)$
- 문제: N번째 피보나치 수를 구하는 문제
- 작은 문제: N-1번째 피보나치 수를 구하는 문제, N-2번째 피보나치 수를 구하는 문제

Overlapping Subproblem

Overlapping Subproblem

- 문제: N번째 피보나치 수를 구하는 문제
- 작은 문제: N-1번째 피보나치 수를 구하는 문제, N-2번째 피보나치 수를 구하는 문제

- 문제: N-1번째 피보나치 수를 구하는 문제
- 작은 문제: N-2번째 피보나치 수를 구하는 문제, N-3번째 피보나치 수를 구하는 문제

- 문제: N-2번째 피보나치 수를 구하는 문제
- 작은 문제: N-3번째 피보나치 수를 구하는 문제, N-4번째 피보나치 수를 구하는 문제

Overlapping Subproblem

Overlapping Subproblem

- 큰 문제와 작은 문제는 상대적이다.

- 문제: N번째 피보나치 수를 구하는 문제
- 작은 문제: N-1번째 피보나치 수를 구하는 문제, N-2번째 피보나치 수를 구하는 문제

- 문제: N-1번째 피보나치 수를 구하는 문제
- 작은 문제: N-2번째 피보나치 수를 구하는 문제, N-3번째 피보나치 수를 구하는 문제

Overlapping Subproblem

Overlapping Subproblem

- 큰 문제와 작은 문제를 같은 방법으로 풀 수 있다.
- 문제를 작은 문제로 쪼갤 수 있다.

Optimal Substructure

Optimal Substructure

- 문제의 정답을 작은 문제의 정답에서 구할 수 있다.

- 예시
- 서울에서 부산을 가는 가장 빠른 길이 대전과 대구를 순서대로 거쳐야 한다면
- 대전에서 부산을 가는 가장 빠른 길은 대구를 거쳐야 한다.

Optimal Substructure

Optimal Substructure

- 문제: N번째 피보나치 수를 구하는 문제
- 작은 문제: N-1번째 피보나치 수를 구하는 문제, N-2번째 피보나치 수를 구하는 문제
- **문제의 정답을 작은 문제의 정답을 합하는 것으로 구할 수 있다.**
- 문제: N-1번째 피보나치 수를 구하는 문제
- 작은 문제: N-2번째 피보나치 수를 구하는 문제, N-3번째 피보나치 수를 구하는 문제
- **문제의 정답을 작은 문제의 정답을 합하는 것으로 구할 수 있다.**
- 문제: N-2번째 피보나치 수를 구하는 문제
- 작은 문제: N-3번째 피보나치 수를 구하는 문제, N-4번째 피보나치 수를 구하는 문제
- **문제의 정답을 작은 문제의 정답을 합하는 것으로 구할 수 있다.**

Optimal Substructure

Optimal Substructure

- Optimal Substructure를 만족한다면, 문제의 크기에 상관없이 어떤 한 문제의 정답은 일정하다.
- 10번째 피보나치 수를 구하면서 구한 4번째 피보나치 수
- 9번째 피보나치 수를 구하면서 구한 4번째 피보나치 수
- …
- 5번째 피보나치 수를 구하면서 구한 4번째 피보나치 수
- 4번째 피보나치 수를 구하면서 구한 4번째 피보나치 수

- 4번째 피보나치 수는 항상 같다.

다이나믹 프로그래밍

Dynamic Programming

- 다이나믹 프로그래밍에서 각 문제는 한 번만 풀어야 한다.
- Optimal Substructure를 만족하기 때문에, 같은 문제는 구할 때마다 정답이 같다.
- 따라서, 정답을 한 번 구했으면, 정답을 어딘가에 메모해놓는다.
- 이런 메모하는 것을 코드의 구현에서는 배열에 저장하는 것으로 할 수 있다.
- 메모를 한다고 해서 영어로 Memoization이라고 한다.

피보나치 수

Dynamic Programming

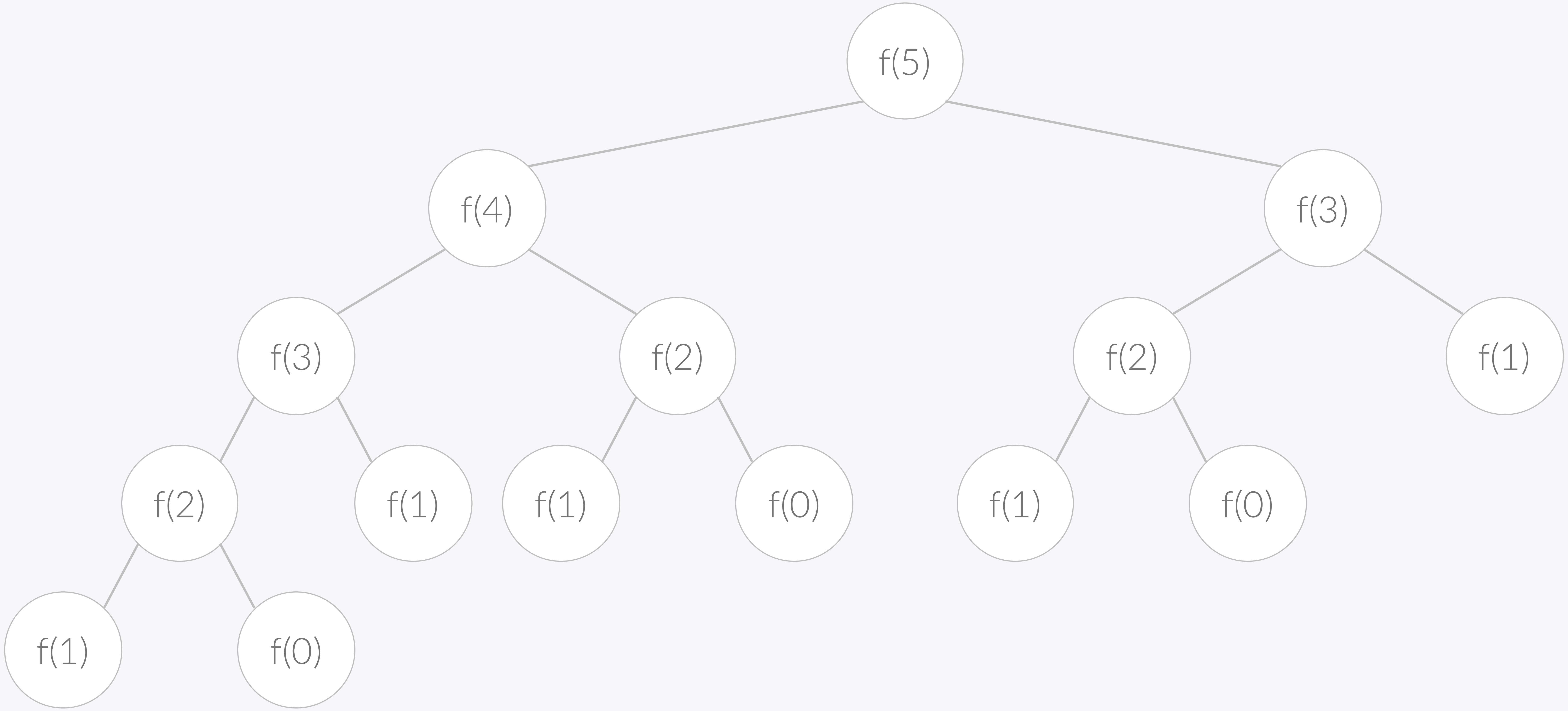
```
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

- 피보나치 수를 구하는 함수이다.

피보나치 수

Dynamic Programming

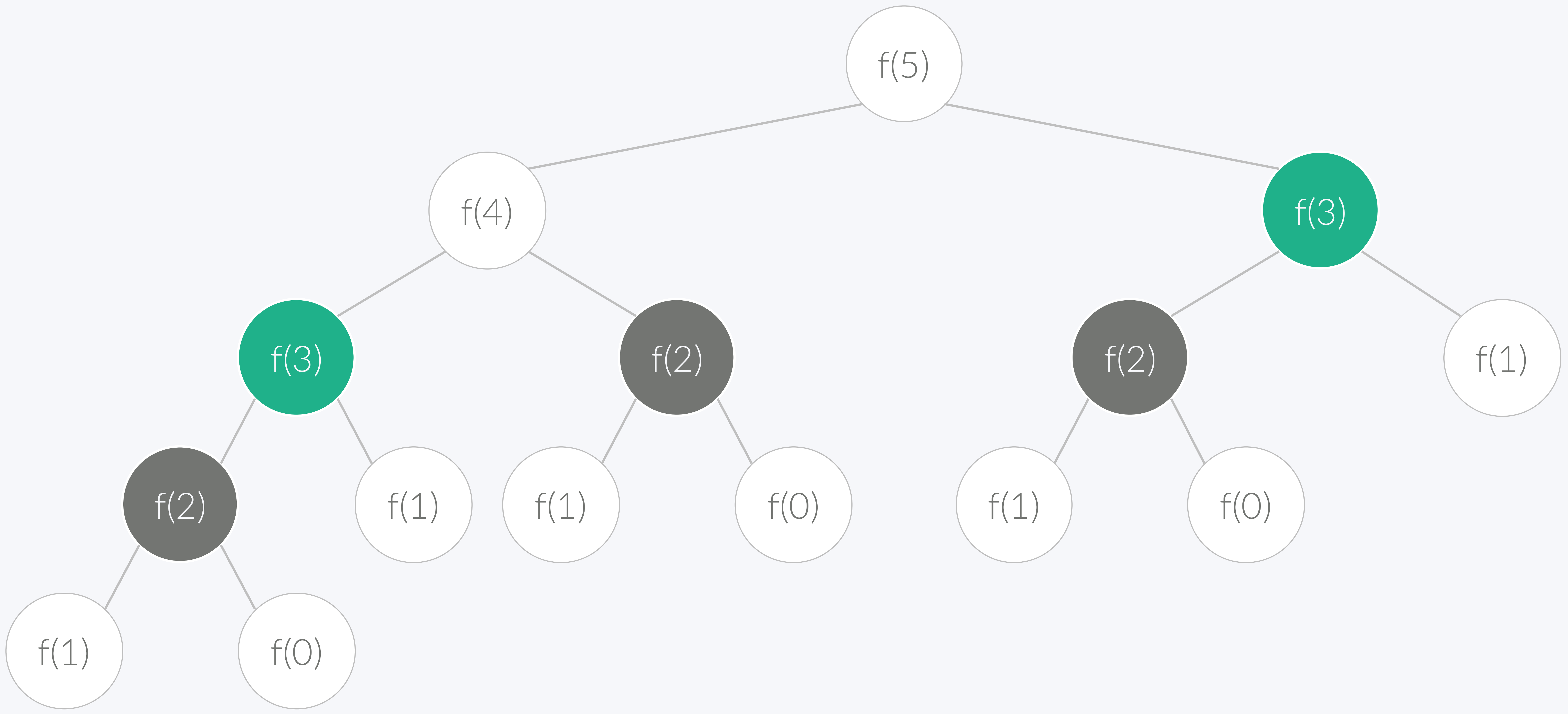
- fibonacci(5)를 호출한 경우 함수가 어떻게 호출되는지를 나타낸 그림



피보나치 수

Dynamic Programming

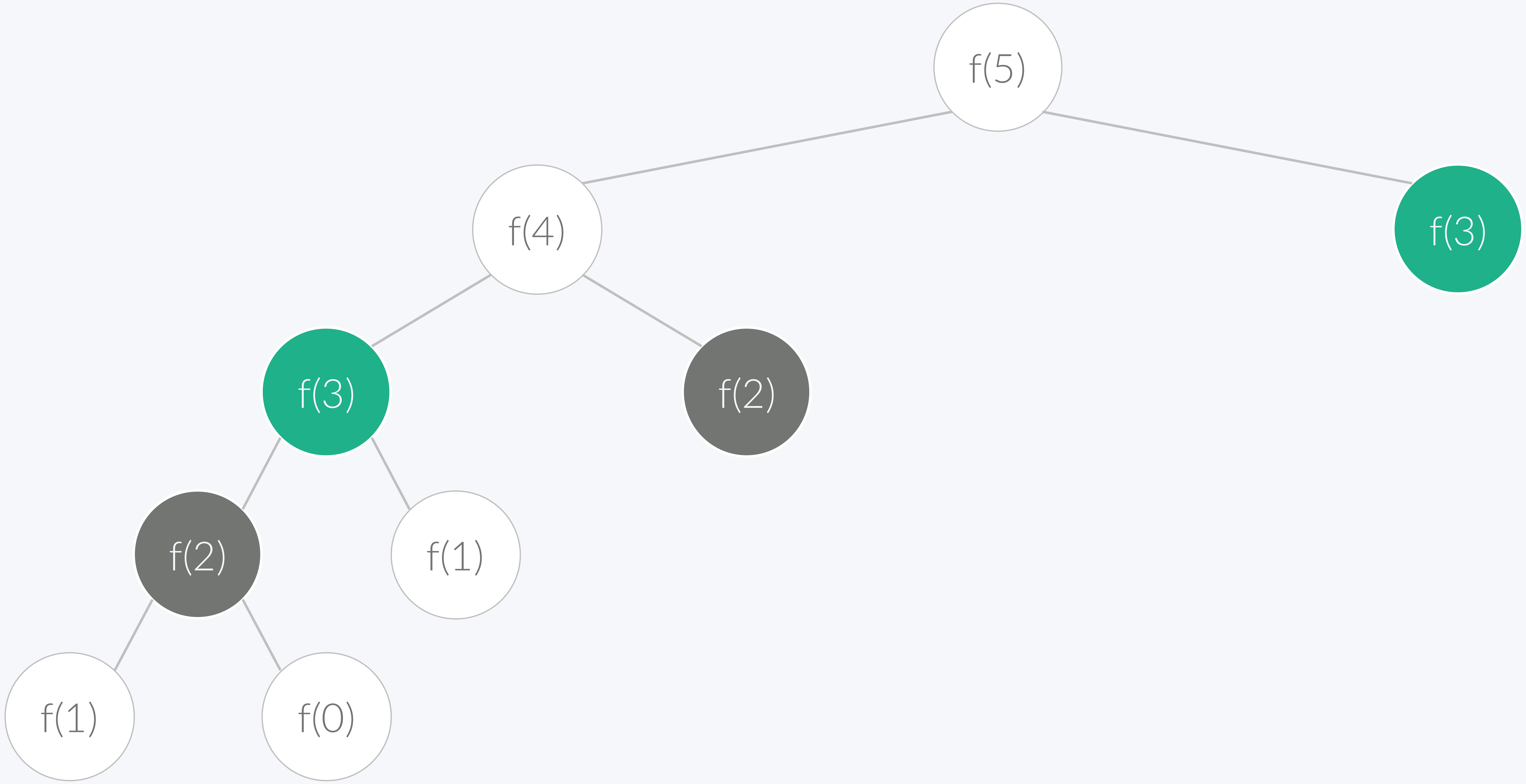
- 아래 그림과 같이 겹치는 호출이 생긴다.



피보나치 수

Dynamic Programming

- 한 번 답을 구할 때, 어딘가에 메모를 해놓고, 중복 호출이면 메모해놓은 값을 리턴한다.



피보나치 수

Dynamic Programming


```
int memo[100];
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        memo[n] = fibonacci(n-1) + fibonacci(n-2);
        return memo[n];
    }
}
```

피보나치 수

Dynamic Programming

```
int memo[100];
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        memo[n] = fibonacci(n-1) + fibonacci(n-2);
        return memo[n];
    }
}
```

피보나치 수

Dynamic Programming

```
int memo[100];
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        if (memo[n] > 0) {
            return memo[n];
        }
        memo[n] = fibonacci(n-1) + fibonacci(n-2);
        return memo[n];
    }
}
```

다이나믹 프로그래밍

Dynamic Programming

- 다이나믹을 푸는 두 가지 방법이 있다.

1. Top-down
2. Bottom-up

Top-down

Dynamic Programming

1. 문제를 작은 문제로 나눈다.
2. 작은 문제를 푼다.
3. 작은 문제를 풀었으니, 이제 문제를 푼다.

Top-down

Dynamic Programming

1. 문제를 풀어야 한다.
    - fibonacci(n)
2. 문제를 작은 문제로 나눈다.
    - fibonacci(n-1)과 fibonacci(n-2)로 문제를 나눈다.
3. 작은 문제를 푼다.
    - fibonacci(n-1)과 fibonacci(n-2)를 호출해 문제를 푼다.
4. 작은 문제를 풀었으니, 이제 문제를 푼다.
    - fibonacci(n-1)의 값과 fibonacci(n-2)의 값을 더해 문제를 푼다.

Top-down

Dynamic Programming

- Top-down은 재귀 호출을 이용해서 문제를 쉽게 풀 수 있다.

Bottom-up

Dynamic Programming

1. 문제를 크기가 작은 문제부터 차례대로 푼다.
2. 문제의 크기를 조금씩 크게 만들면서 문제를 점점 푼다.
3. 작은 문제를 풀면서 왔기 때문에, 큰 문제는 항상 풀 수 있다.
4. 그러다보면, 언젠간 풀어야 하는 문제를 풀 수 있다.

Bottom-up

Dynamic Programming

```
int d[100];
int fibonacci(int n) {
    d[0] = 1;
    d[1] = 1;
    for (int i=2; i<=n; i++) {
        d[i] = d[i-1] + d[i-2];
    }
    return d[n];
}
```

Bottom-up

Dynamic Programming

1. 문제를 크기가 작은 문제부터 차례대로 푼다.
   - for (int i=2; i<=n; i++)
2. 문제의 크기를 조금씩 크게 만들면서 문제를 점점 푼다.
   - for (int i=2; i<=n; **i++**)
3. 작은 문제를 풀면서 왔기 때문에, 큰 문제는 항상 풀 수 있다.
   - d[i] = d[i-1] + d[i-2];
4. 그러다보면, 언젠간 풀어야 하는 문제를 풀 수 있다.
   - d[n]을 구하게 된다.

문제 풀이 전략

다이나믹 문제 풀이 전략

Dynamic Programming

- 문제에서 구하려고 하는 답을 문장으로 나타낸다.
- 예: 피보나치 수를 구하는 문제
- N번째 피보나치 수
- 이제 그 문장에 나와있는 변수의 개수만큼 메모하는 배열을 만든다.
- Top-down인 경우에는 재귀 호출의 인자의 개수
- 문제를 작은 문제로 나누고, 수식을 이용해서 문제를 표현해야 한다.

문제 풀이

다이나믹 문제 풀이

Dynamic Programming

- 다이나믹은 문제를 많이 풀면서 감을 잡는 것이 중요하기 때문에
- 문제를 풀어 봅시다

1로 만들기

https://www.acmicpc.net/problem/1463

- 세준이는 어떤 정수 N에 다음과 같은 연산중 하나를 할 수 있다.

1. N이 3으로 나누어 떨어지면, 3으로 나눈다.
2. N이 2로 나누어 떨어지면, 2로 나눈다.
3. 1을 뺀다.

- 세준이는 어떤 정수 N에 위와 같은 연산을 선택해서 1을 만드려고 한다. 연산을 사용하는 횟수의 최소값을 출력하시오.

1로 만들기

https://www.acmicpc.net/problem/1463

- D[i]= i를 1로 만드는데 필요한 최소 연산 횟수
- i에게 가능한 경우를 생각해보자

1. i가 3으로 나누어 떨어졌을 때, 3으로 나누는 경우
2. i가 2로 나누어 떨어졌을 때, 2로 나누는 경우
3. i에서 1을 빼는 경우

1로 만들기

https://www.acmicpc.net/problem/1463

- D[i]= i를 1로 만드는데 필요한 최소 연산 횟수
- i에게 가능한 경우를 생각해보자

1. i가 3으로 나누어 떨어졌을 때, 3으로 나누는 경우
    - D[i/3] + 1
2. i가 2로 나누어 떨어졌을 때, 2로 나누는 경우
    - D[i/2] + 1
3. i에서 1을 빼는 경우
    - D[i-1] + 1

1로 만들기

https://www.acmicpc.net/problem/1463

- D[i]= i를 1로 만드는데 필요한 최소 연산 횟수
- i에게 가능한 경우를 생각해보자

1. i가 3으로 나누어 떨어졌을 때, 3으로 나누는 경우
   - D[i/3] + 1
2. i가 2로 나누어 떨어졌을 때, 2로 나누는 경우
   - D[i/2] + 1
3. i에서 1을 빼는 경우
   - D[i-1] + 1

- 세 값중의 최소값이 들어가게 된다.

1로 만들기

https://www.acmicpc.net/problem/1463

```
int go(int n) {
    if (n == 1) return 0;
    if (d[n] > 0) return d[n];
    d[n] = go(n-1) + 1;
    if (n%2 == 0) {
        int temp = go(n/2) + 1;
        if (d[n] > temp) d[n] = temp;
    }
    if (n%3 == 0) {
        int temp = go(n/3) + 1;
        if (d[n] > temp) d[n] = temp;
    }
    return d[n];
}
```

1로 만들기

https://www.acmicpc.net/problem/1463

```
d[1] = 0;
for (int i=2; i<=n; i++) {
    d[i] = d[i-1] + 1;
    if (i%2 == 0 && d[i] > d[i/2] + 1) {
        d[i] = d[i/2] + 1;
    }
    if (i%3 == 0 && d[i] > d[i/3] + 1) {
        d[i] = d[i/3] + 1;
    }
}
```

1로 만들기

https://www.acmicpc.net/problem/1463

- Top-Down 방식
- C
  - https://gist.github.com/Baekjoon/a53dc4861bd9d081682c
- C++
  - https://gist.github.com/Baekjoon/63b659f985beb8f64ca7
- Java
  - https://gist.github.com/Baekjoon/7b675fe68d3c2abfef40

1로 만들기

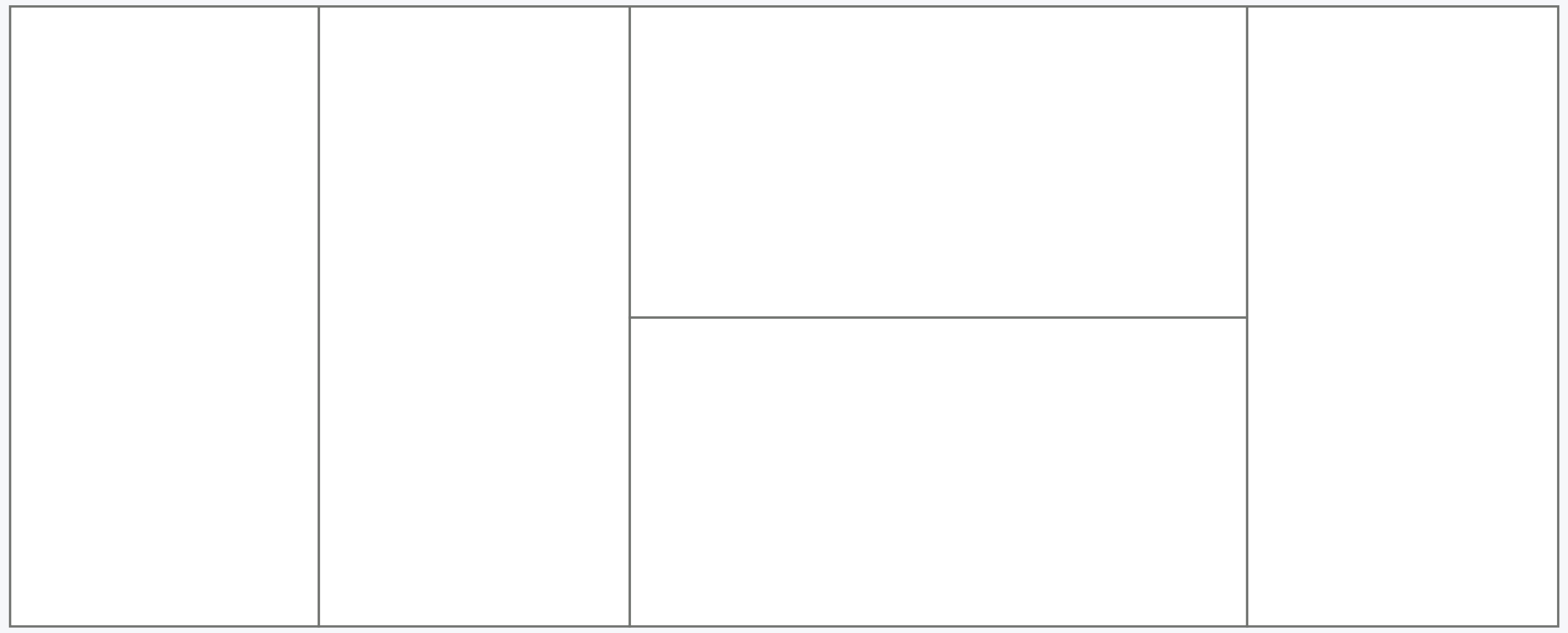
https://www.acmicpc.net/problem/1463

- Bottom-up 방식
- C
  - https://gist.github.com/Baekjoon/30f4bb39cdc66f7f16c1
- C++
  - https://gist.github.com/Baekjoon/31e553ab3b371fe06384
- Java
  - https://gist.github.com/Baekjoon/0813d3bc5db11b9bb72d

2×n 타일링

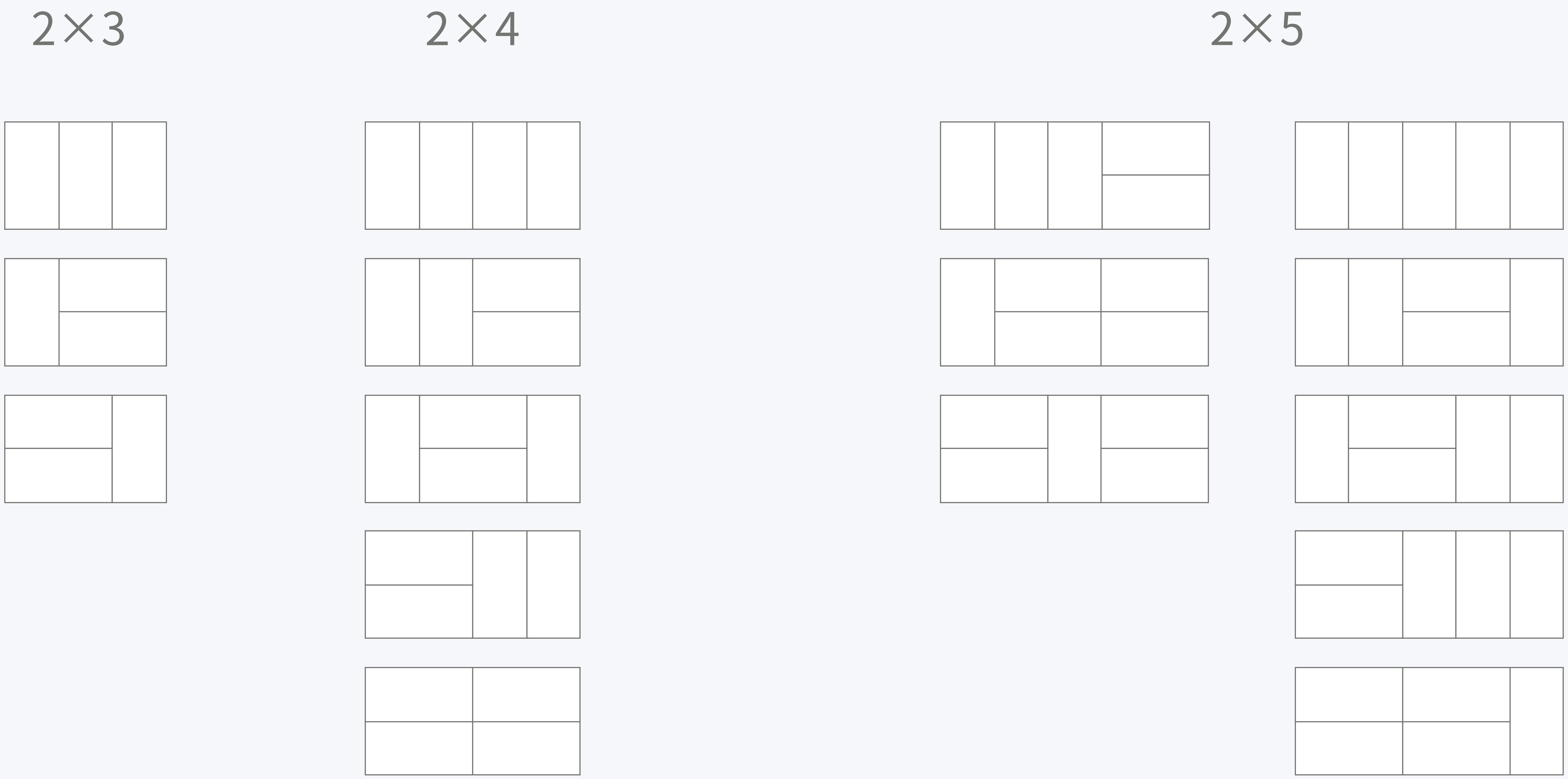
https://www.acmicpc.net/problem/11726

- 2×n 직사각형을 1×2, 2×1타일로 채우는 방법의 수
- 아래 그림은 2×5를 채우는 방법의 수
- D[i] = 2×i 직사각형을 채우는 방법의 수



2×n 타일링

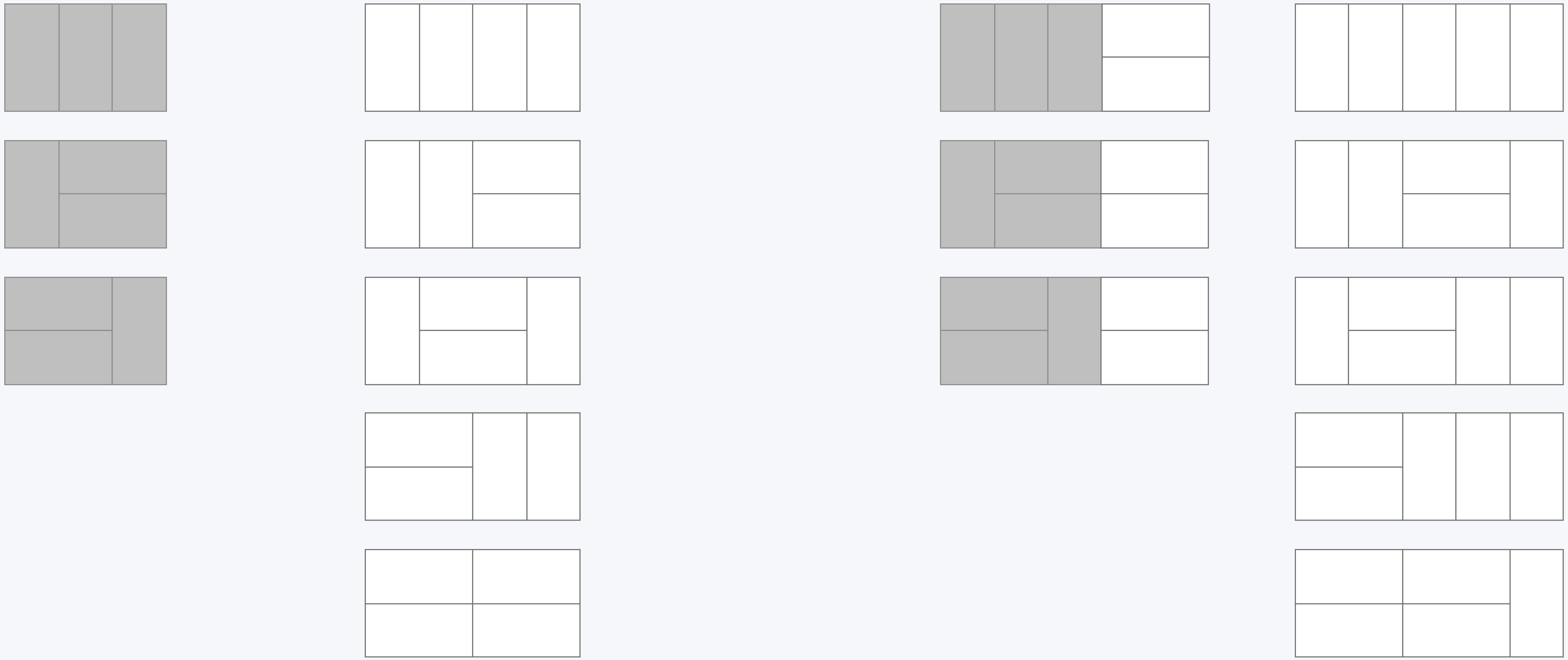
https://www.acmicpc.net/problem/11726



2×n 타일링

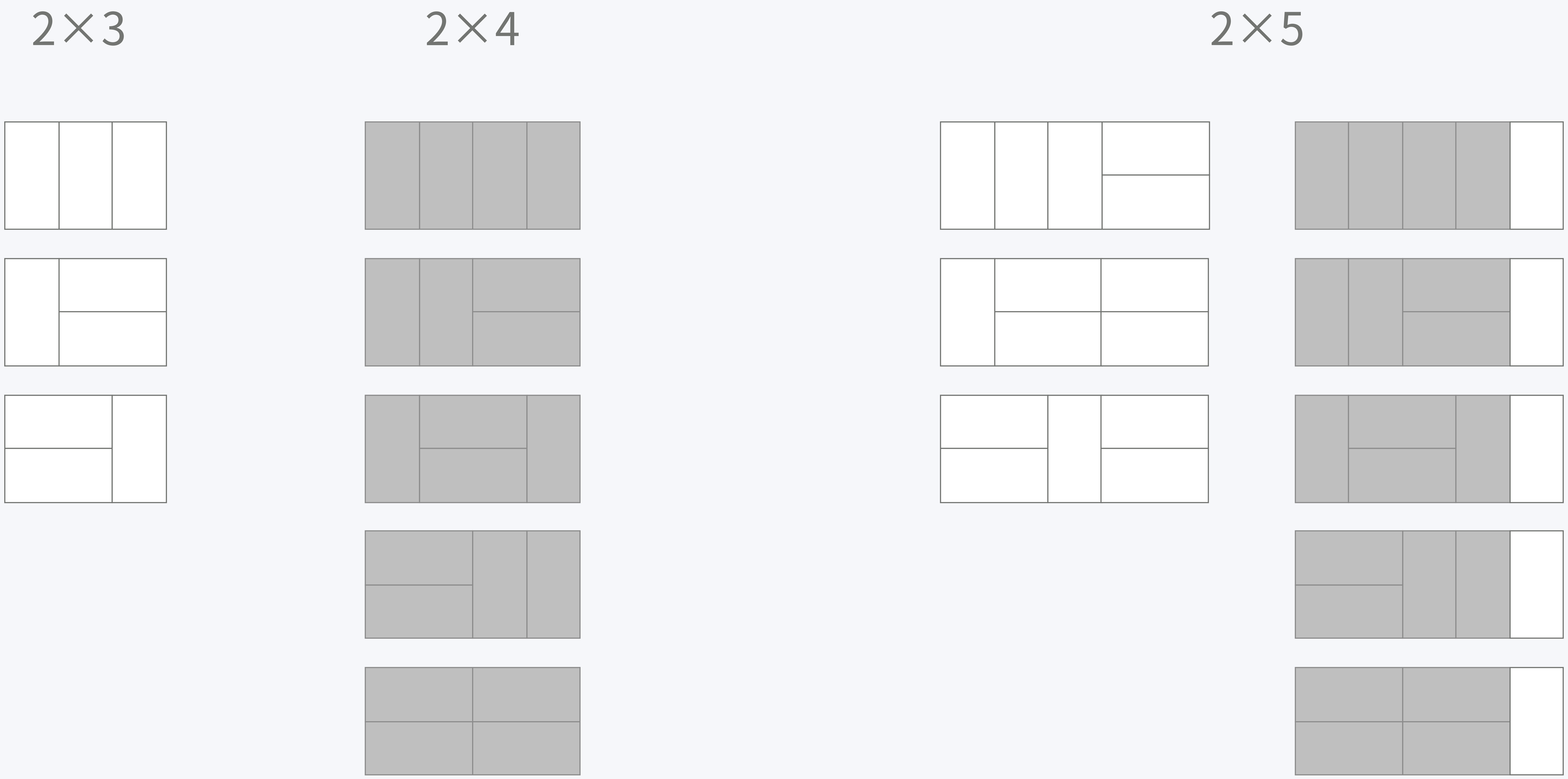
https://www.acmicpc.net/problem/11726

2×3 2×4 2×5



2×n 타일링

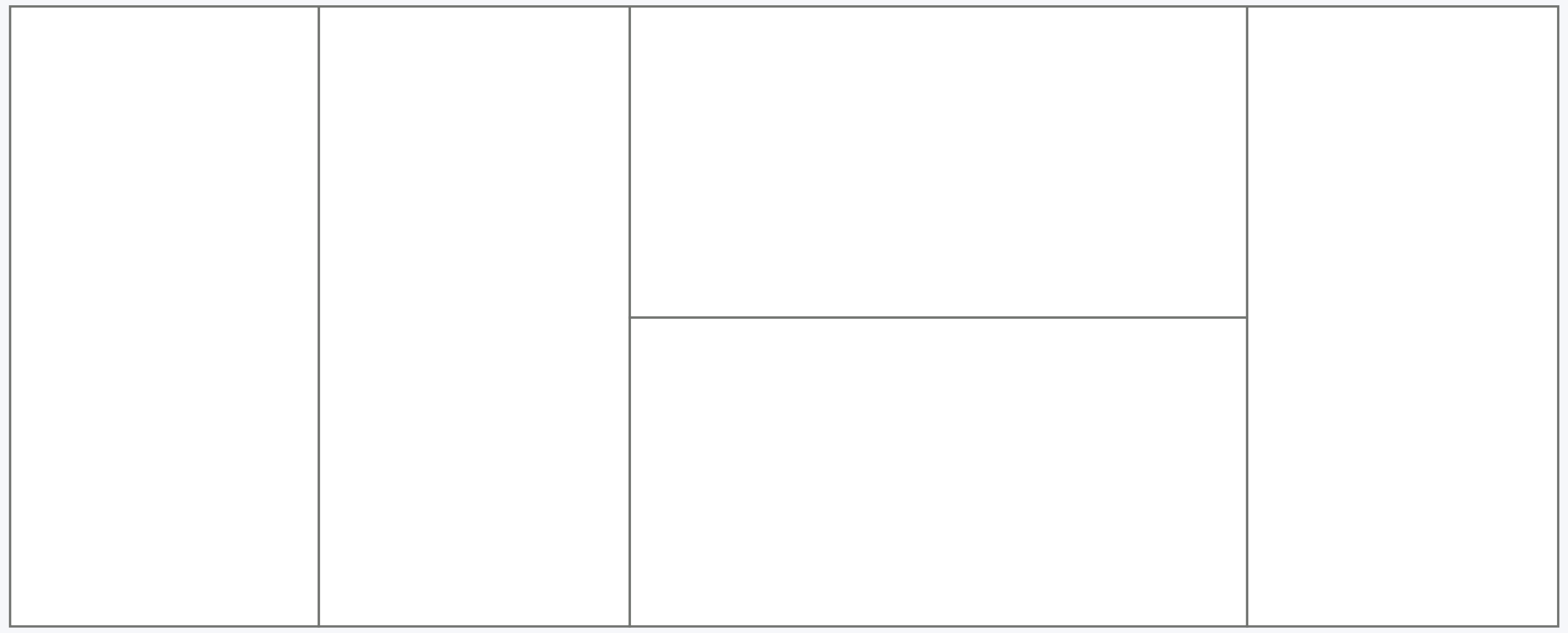
https://www.acmicpc.net/problem/11726



2×n 타일링

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

- 2×n 직사각형을 1×2, 2×1타일로 채우는 방법의 수
- D[i] = 2×i 직사각형을 채우는 방법의 수
- D[i] = D[i-1] + D[i-2]



2×n 타일링

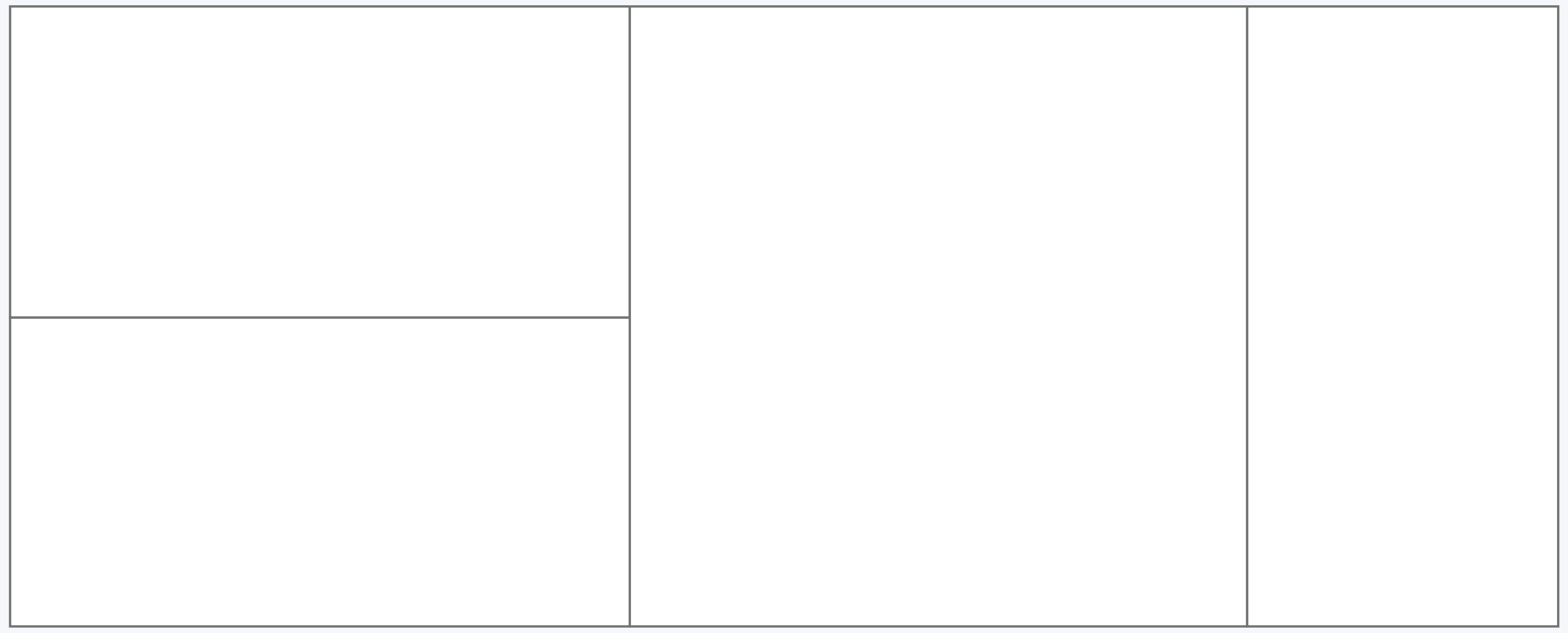
https://www.acmicpc.net/problem/11726

- C/C++
  - https://gist.github.com/Baekjoon/3527f6fdfd4771f8c3e1

2×n 타일링 2

https://www.acmicpc.net/problem/11727

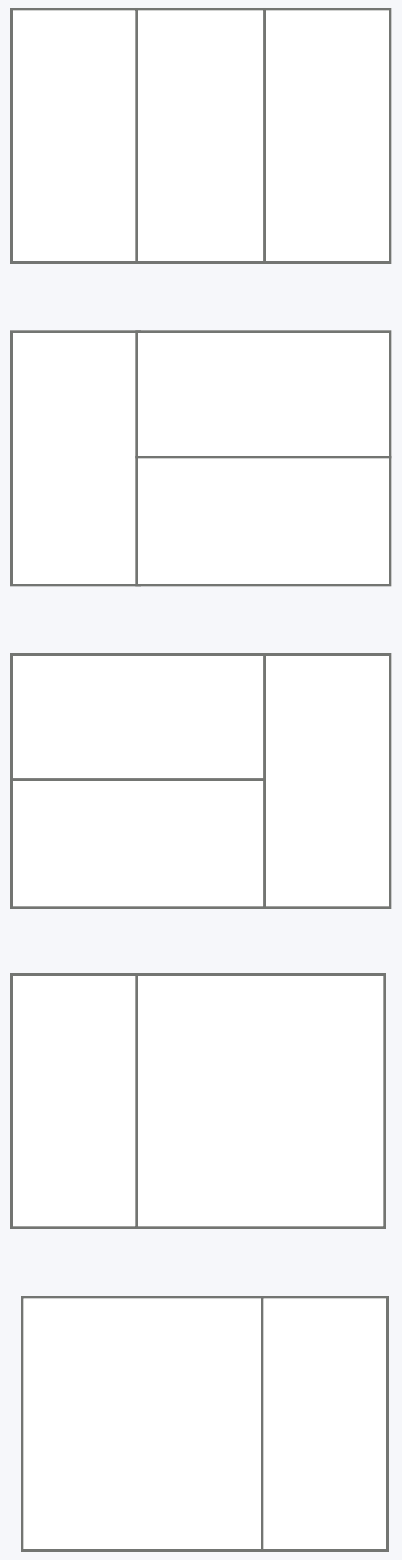
- 2×n 직사각형을 1×2, 2×1, 2×2타일로 채우는 방법의 수
- 아래 그림은 2×5를 채우는 방법의 수
- D[i] = 2×i 직사각형을 채우는 방법의 수



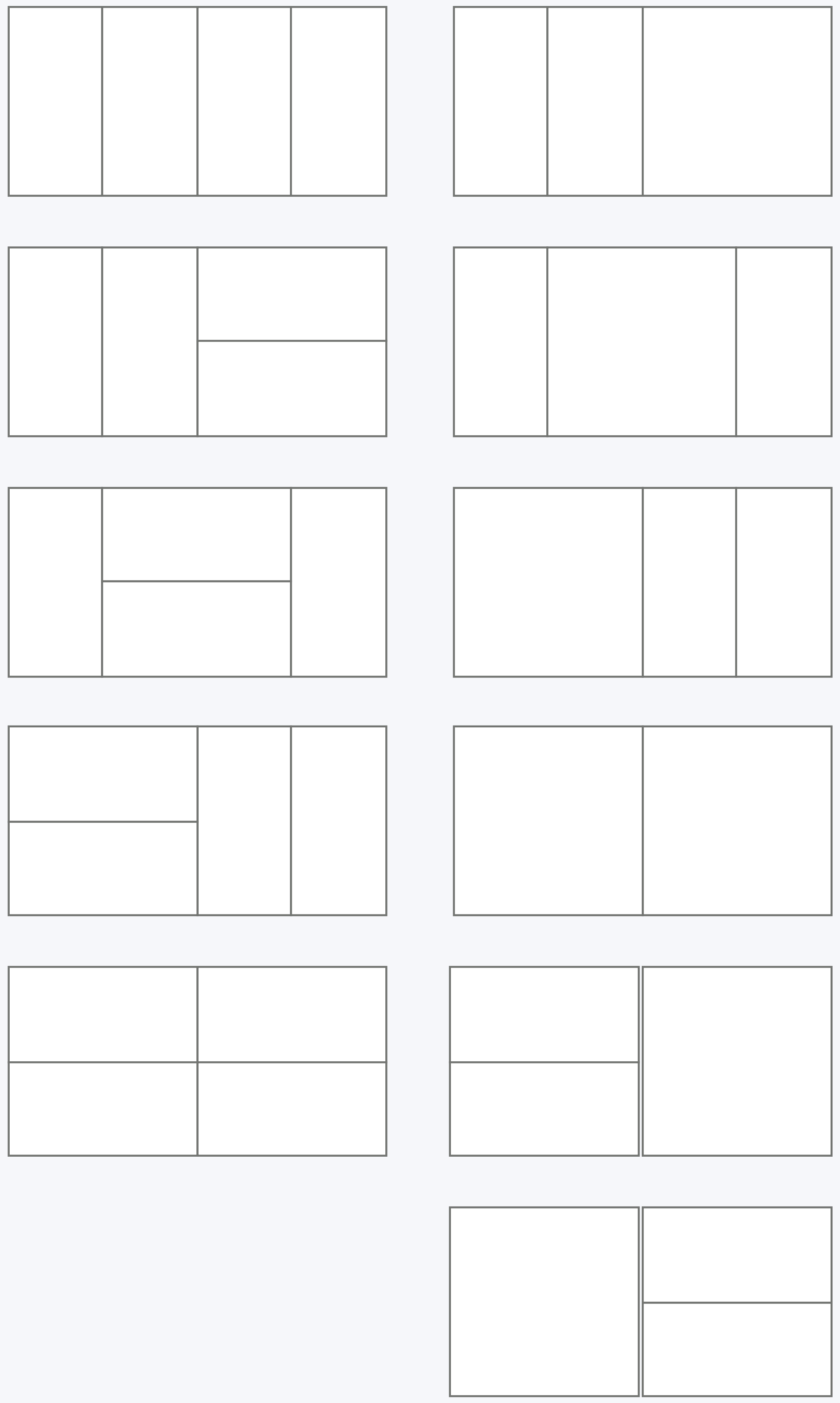
2×n 타일링 2

https://www.acmicpc.net/problem/11727

2×3



2×4

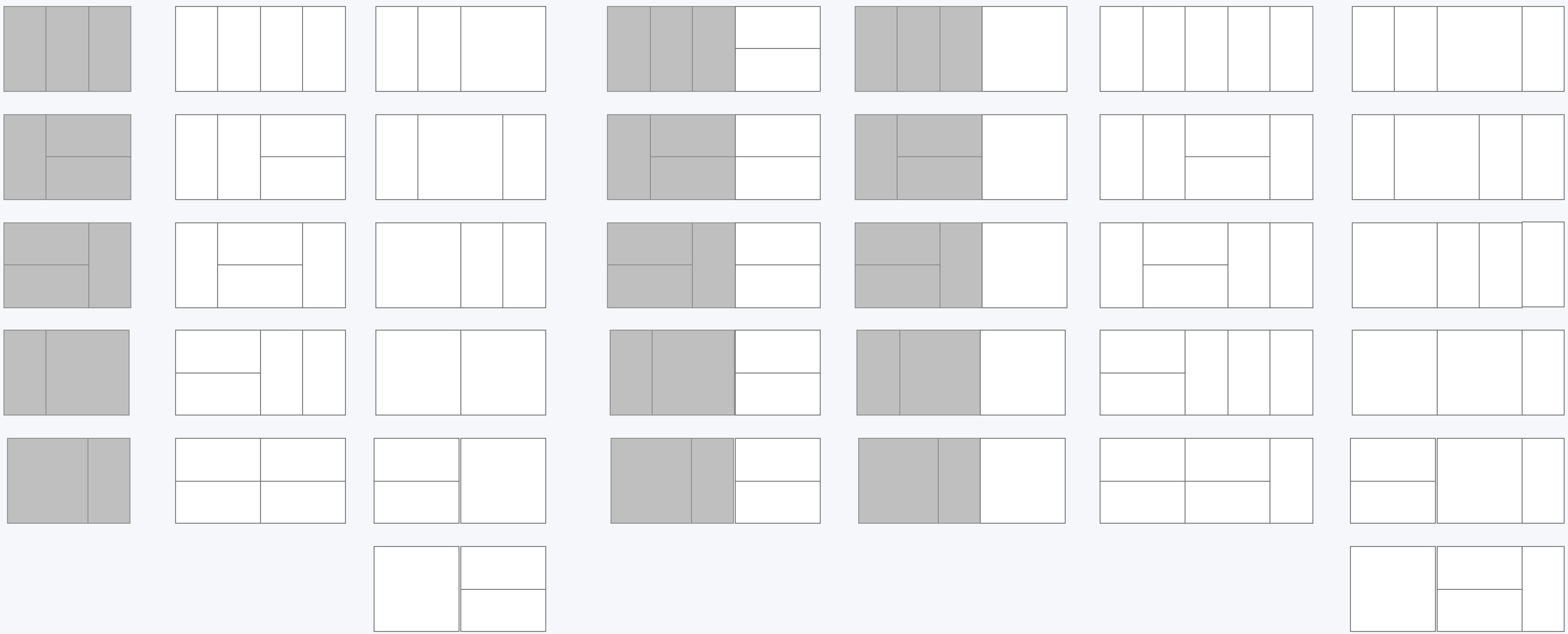


2×5

2×n 타일링 2

https://www.acmicpc.net/problem/11727

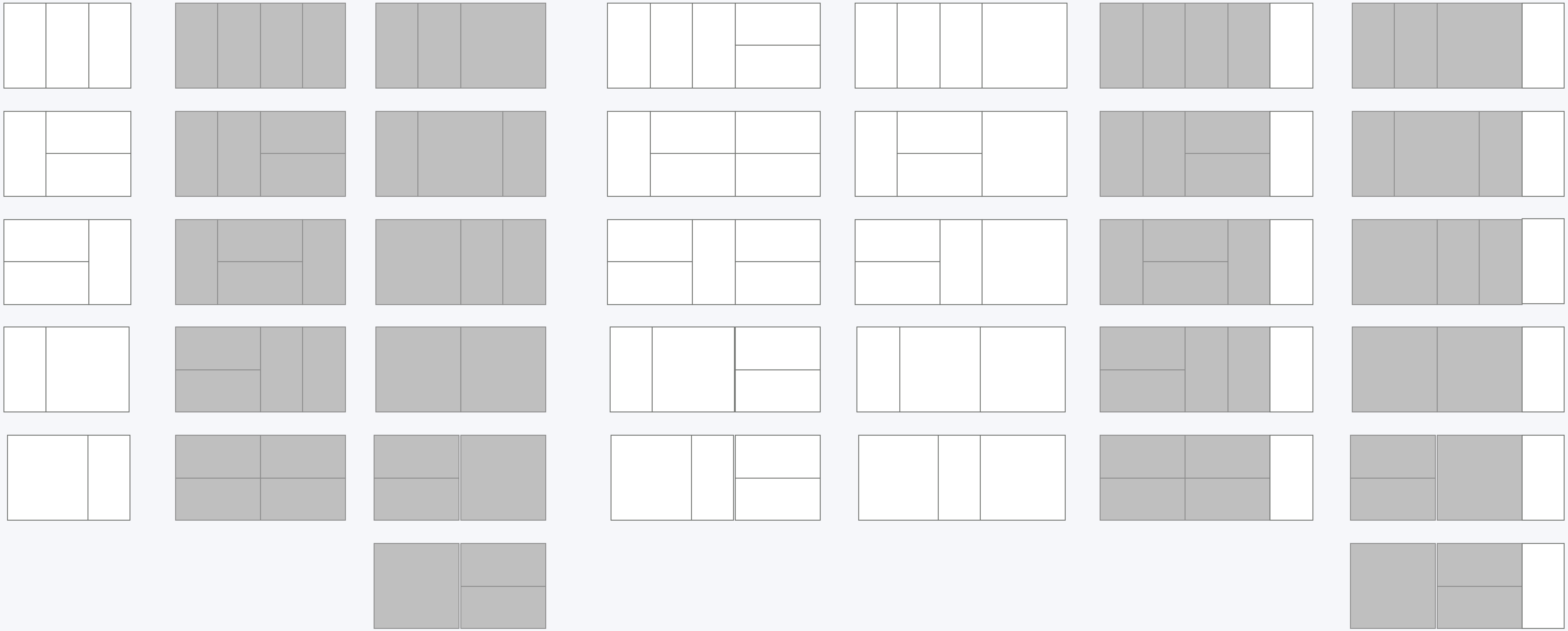
2×3 2×4 2×5



2×n 타일링 2

https://www.acmicpc.net/problem/11727

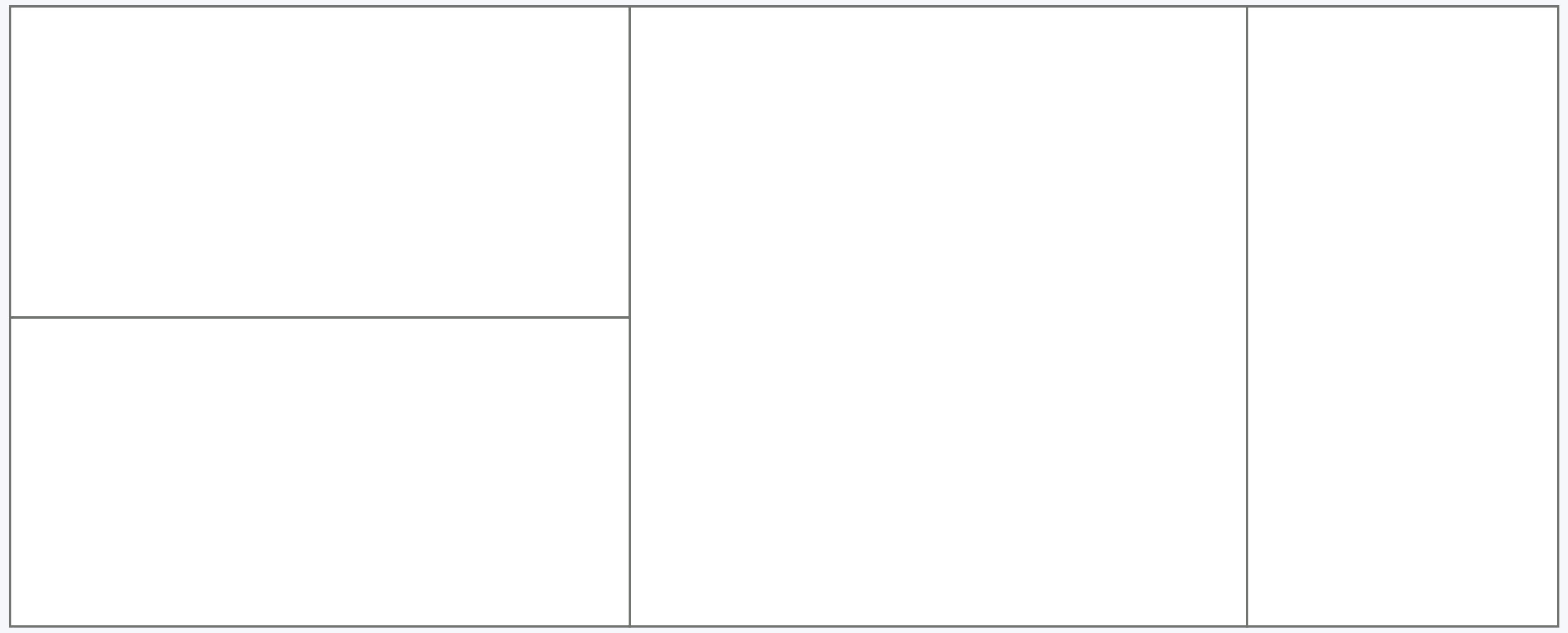
2×3 2×4 2×5



2×n 타일링 2

https://www.acmicpc.net/problem/11727

- 2×n 직사각형을 1×2, 2×1, 2×2타일로 채우는 방법의 수
- D[i] = 2×i 직사각형을 채우는 방법의 수
- D[i] = 2*D[i-2] + D[i-1]



2×n 타일링 2

https://www.acmicpc.net/problem/11727

- C/C++
  - https://gist.github.com/Baekjoon/2ac3e7f55b9f3799d02d

1, 2, 3 더하기

https://www.acmicpc.net/problem/9095

- 정수 n을 1, 2, 3의 조합으로 나타내는 방법의 수를 구하는 문제
- n = 4
- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

1, 2, 3 더하기

https://www.acmicpc.net/problem/9095

- D[i] = i를 1, 2, 3의 조합으로 나타내는 방법의 수

1, 2, 3 더하기

https://www.acmicpc.net/problem/9095

- D[i] = i를 1, 2, 3의 조합으로 나타내는 방법의 수
- D[i] = D[i-1] + D[i-2] + D[i-3]

1, 2, 3 더하기

https://www.acmicpc.net/problem/9095

- C/C++
    - https://gist.github.com/Baekjoon/6e4f9e363b3aaef733d1
- Java
    - https://gist.github.com/Baekjoon/e019984a7c7f1ac6bd32

붕어빵 판매하기

https://www.acmicpc.net/problem/11052

- 붕어빵 N개를 가지고 있다.
- 붕어빵 i개를 팔아서 얻을 수 있는 수익이 P[i]일 때, N개를 모두 판매해서 얻을 수 있는 최대 수익 구하기

붕어빵 판매하기

https://www.acmicpc.net/problem/11052

- D[i] = 붕어빵 i개를 팔아서 얻을 수 있는 최대 수익
- 가능한 경우 생각해보기

붕어빵 판매하기

https://www.acmicpc.net/problem/11052

- D[i] = 붕어빵 i개를 팔아서 얻을 수 있는 최대 수익
- 가능한 경우 생각해보기
- 붕어빵 1개를 P[1]에 팔기
- 붕어빵 2개를 P[2]에 팔기
- …
- 붕어빵 i-1개를 P[i-1]에 팔기
- 붕어빵 i개를 P[i]에 팔기

붕어빵 판매하기

https://www.acmicpc.net/problem/11052

- D[i] = 붕어빵 i개를 팔아서 얻을 수 있는 최대 수익
- 가능한 경우 생각해보기
- 붕어빵 1개를 P[1]에 팔기 → 남은 붕어빵의 개수: i-1
- 붕어빵 2개를 P[2]에 팔기 → 남은 붕어빵의 개수: i-2
- …
- 붕어빵 i-1개를 P[i-1]에 팔기 → 남은 붕어빵의 개수: 1
- 붕어빵 i개를 P[i]에 팔기 → 남은 붕어빵의 개수: 0

붕어빵 판매하기

https://www.acmicpc.net/problem/11052

- D[i] = 붕어빵 i개를 팔아서 얻을 수 있는 최대 수익
- 가능한 경우 생각해보기
- 붕어빵 1개를 P[1]에 팔기 → 남은 붕어빵의 개수: i-1 → P[1] + D[i-1]
- 붕어빵 2개를 P[2]에 팔기 → 남은 붕어빵의 개수: i-2 → P[2] + D[i-2]
- …
- 붕어빵 i-1개를 P[i-1]에 팔기 → 남은 붕어빵의 개수: 1 → P[i-1] + D[1]
- 붕어빵 i개를 P[i]에 팔기 → 남은 붕어빵의 개수: 0 → P[i] + D[0]

붕어빵 판매하기

https://www.acmicpc.net/problem/11052

- D[i] = 붕어빵 i개를 팔아서 얻을 수 있는 최대 수익
- 가능한 경우 생각해보기
- 붕어빵 1개를 P[1]에 팔기 → 남은 붕어빵의 개수: i-1 → P[1] + D[i-1]
- 붕어빵 2개를 P[2]에 팔기 → 남은 붕어빵의 개수: i-2 → P[2] + D[i-2]
- …
- 붕어빵 i-1개를 P[i-1]에 팔기 → 남은 붕어빵의 개수: 1 → P[i-1] + D[1]
- 붕어빵 i개를 P[i]에 팔기 → 남은 붕어빵의 개수: 0 → P[i] + D[0]
- D[i] = max(P[j] + D[i-j]) (1 ≤ j ≤ i)

붕어빵 판매하기

https://www.acmicpc.net/problem/11052

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

붕어빵 판매하기

https://www.acmicpc.net/problem/11052

- C
  - https://gist.github.com/Baekjoon/e8f8b7904a6395748246
- C++
  - https://gist.github.com/Baekjoon/d916898171846c9286aa
- Java
  - https://gist.github.com/Baekjoon/efaf5dee617b4ee9d305

쉬운 계단 수

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

- 인접한 자리의 차이가 1이 나는 수를 계단 수라고 한다
- 예: 45656
- 길이가 N인 계단 수의 개수를 구하는 문제

쉬운 계단 수

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

- D[i][j] = 길이가 i이가 마지막 숫자가 j인 계단 수의 개수
- D[i][j] = D[i-1][j-1] + D[i-1][j+1]

쉬운 계단 수

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

```
for (int i=1; i<=9; i++) d[1][i] = 1;
for (int i=2; i<=n; i++) {
    for (int j=0; j<=9; j++) {
        d[i][j] = 0;
        if (j-1 >= 0) d[i][j] += d[i-1][j-1];
        if (j+1 <= 9) d[i][j] += d[i-1][j+1];
        d[i][j] %= mod;
    }
}
long long ans = 0;
for (int i=0; i<=9; i++) ans += d[n][i];
ans %= mod;
```

쉬운 계단 수

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

- C/C++
  - https://gist.github.com/Baekjoon/4d98f519afbcdd5d3d0f
- Java
  - https://gist.github.com/Baekjoon/7e4e12ce1b0aa740d5d1

오르막 수

https://www.acmicpc.net/problem/11057

- 오르막 수는 수의 자리가 오름차순을 이루는 수를 말한다
- 인접한 수가 같아도 오름차순으로 친다
- 수의 길이 N이 주어졌을 때, 오르막 수의 개수를 구하는 문제
- 수는 0으로 시작할 수 있다
- 예: 1233345, 357, 8888888, 1555999

오르막 수

https://www.acmicpc.net/problem/11057

- D[i][j] = 길이가 i이고 마지막 숫자가 j인 오르막 수의 개수
- D[1][i] = 1
- D[i][j] += D[i-1][k] ($0 \le k \le j$)

오르막 수

https://www.acmicpc.net/problem/11057

```
for (int i=0; i<=9; i++) d[1][i] = 1;
for (int i=2; i<=n; i++) {
    for (int j=0; j<=9; j++) {
        for (int k=0; k<=j; k++) {
            d[i][j] += d[i-1][k];
            d[i][j] %= mod;
        }
    }
}
long long ans = 0;
for (int i=0; i<10; i++) ans  += d[n][i];
ans %= mod;
```

오르막 수

https://www.acmicpc.net/problem/11057

- C/C++
  - https://gist.github.com/Baekjoon/3d7ae9472aa843dc3a48
- Java
  - https://gist.github.com/Baekjoon/264f68b19e93cc9b46aa

이친수

https://www.acmicpc.net/problem/2193

- 0과 1로만 이루어진 수를 이진수라고 한다.
- 다음 조건을 만족하면 이친수라고 한다.

1. 이친수는 0으로 시작하지 않는다.
2. 이친수에서는 1이 두 번 연속으로 나타나지 않는다. 즉, 11을 부분 문자열로 갖지 않는다.

- N자리 이친수의 개수를 구하는 문제

이친수

https://www.acmicpc.net/problem/2193

- D[i][j] = i자리 이친수의 개수 중에서 j로 끝나는 것의 개수 (j=0, 1)

- 0으로 시작하지 않는다.
- D[1][0] = 0
- D[1][1] = 1

이친수

https://www.acmicpc.net/problem/2193

- D[i][j] = i자리 이친수의 개수 중에서 j로 끝나는 것의 개수 (j=0, 1)
- 가능한 경우
- 0으로 끝나는 경우
- 1로 끝나는 경우

이친수

https://www.acmicpc.net/problem/2193

- D[i][j] = i자리 이친수의 개수 중에서 j로 끝나는 것의 개수 (j=0, 1)
- 가능한 경우
- 0으로 끝나는 경우 (D[i][0])
  - 앞에 0과 1이 올 수 있다
  - D[i-1][0] + D[i-1][1]
- 1로 끝나는 경우 (D[i][1])
  - 앞에 1은 올 수 없다. 즉, 0만 올 수 있다.
  - D[i-1][0]

이친수

https://www.acmicpc.net/problem/2193

- D[i][j] = i자리 이친수의 개수 중에서 j로 끝나는 것의 개수 (j=0, 1)
- D[i][0] = D[i-1][0] + D[i-1][1]
- D[i][1] = D[i-1][0]

이친수

https://www.acmicpc.net/problem/2193

- D[i] = i자리 이친수의 개수
- 가능한 경우
- 0으로 끝나는 경우
- 1로 끝나는 경우

이친수

https://www.acmicpc.net/problem/2193

- D[i] = i자리 이친수의 개수
- 가능한 경우
- 0으로 끝나는 경우
  - 앞에 0과 1 모두 올 수 있다.
  - D[i-1]
- 1로 끝나는 경우
  - 앞에 0만 올 수 있다
  - 앞에 붙는 0을 세트로 생각해서 i-2자리에 01을 붙인다고 생각
  - D[i-2]

이친수

https://www.acmicpc.net/problem/2193

- D[i] = i자리 이친수의 개수
- D[i] = D[i-1] + D[i-2]

이친수

https://www.acmicpc.net/problem/2193

- C/C++
  - https://gist.github.com/Baekjoon/49b2bfd22be42707bb88
- Java
  - https://gist.github.com/Baekjoon/7fbfd8d0963139d638de

포도주 시식

https://www.acmicpc.net/problem/2156

- 포도주가 일렬로 놓여져 있고, 다음과 같은 2가지 규칙을 지키면서 포도주를 최대한 많이 마시려고 한다.

1. 포도주 잔을 선택하면 그 잔에 들어있는 포도주는 모두 마셔야 하고, 마신 후에는 원래 위치에 다시 놓아야 한다.
2. 연속으로 놓여 있는 3잔을 모두 마실 수는 없다.

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i] = A[1], …, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- i에게 가능한 경우

1. i번째 포도주를 마시는 경우
2. i번째 포도주를 마시지 않는 경우

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i] = A[1], ⋯, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- i에게 가능한 경우

1. i번째 포도주를 마시는 경우
   - D[i-1] + A[i]
2. i번째 포도주를 마시지 않는 경우
   - D[i-1]

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i] = A[1], ⋯, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- i에게 가능한 경우

1. i번째 포도주를 마시는 경우
   - D[i-1] + A[i]
2. i번째 포도주를 마시지 않는 경우
   - D[i-1]

- D[i] = max(D[i-1]+A[i], D[i-1])
- 위의 식은 포도주를 연속해서 3잔 마시면 안되는 경우를 처리하지 못한다.

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i][j] = A[1], …, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양, A[i]는 j번 연속해서 마신 포도주임
- D[i][0] = 0번 연속해서 마신 포도주 → A[i]를 마시지 않음
- D[i][1] = 1번 연속해서 마신 포도주 -> A[i-1]을 마시지 않았음
- D[i][2] = 2번 연속해서 마신 포도주 → A[i-1]을 마시고, A[i-2]는 마시지 않았어야 함

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i][j] = A[1], …, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양, A[i]는 j번 연속해서 마신 포도주임
- D[i][0] = 0번 연속해서 마신 포도주 → A[i]를 마시지 않음
    - max(D[i-1][0], D[i-1][1], D[i-1][2])
- D[i][1] = 1번 연속해서 마신 포도주 -> A[i-1]을 마시지 않았음
    - D[i-1][0] + A[i]
- D[i][2] = 2번 연속해서 마신 포도주 → A[i-1]을 마시고, A[i-2]는 마시지 않았어야 함
    - D[i-1][1] + A[i]

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i] = A[1], ⋯, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- 0번 연속해서 마신 포도주 → A[i]를 마시지 않음
- 1번 연속해서 마신 포도주 -> A[i-1]을 마시지 않았음
- 2번 연속해서 마신 포도주 → A[i-1]을 마시고, A[i-2]는 마시지 않았어야 함

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i] = A[1], ⋯, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- 0번 연속해서 마신 포도주 → A[i]를 마시지 않음
  - D[i-1]
- 1번 연속해서 마신 포도주 -> A[i-1]을 마시지 않았음
  - D[i-2] + A[i]
- 2번 연속해서 마신 포도주 → A[i-1]을 마시고, A[i-2]는 마시지 않았어야 함
  - D[i-3] + A[i-1] + A[i]

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i] = A[1], ⋯, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- 0번 연속해서 마신 포도주 → A[i]를 마시지 않음
  - D[i-1]
- 1번 연속해서 마신 포도주 -> A[i-1]을 마시지 않았음
  - D[i-2] + A[i]
- 2번 연속해서 마신 포도주 → A[i-1]을 마시고, A[i-2]는 마시지 않았어야 함
  - D[i-3] + A[i-1] + A[i]
- D[i] = max(D[i-1], D[i-2]+A[i], D[i-3] + A[i-1] + A[i])

포도주 시식

https://www.acmicpc.net/problem/2156

- D[i] = A[1], …, A[i] 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- D[i] = max(D[i-1], D[i-2]+A[i], D[i-3] + A[i-1] + A[i])
- i-2, i-3 때문에 예외 처리가 예상되기 때문에
- D[1] = A[1]
- D[2] = A[1] + A[2]
- 로 미리 처리를 해두고
- i = 3부터 문제를 푸는 것이 좋다.

포도주 시식

https://www.acmicpc.net/problem/2156

```
d[1] = a[1];
d[2] = a[1]+a[2];
for (int i=3; i<=n; i++) {
    d[i] = d[i-1];
    if (d[i] < d[i-2] + a[i]) {
        d[i] = d[i-2] + a[i];
    }
    if (d[i] < d[i-3] + a[i] + a[i-1]) {
        d[i] = d[i-3] + a[i] + a[i-1];
    }
}
```

포도주 시식

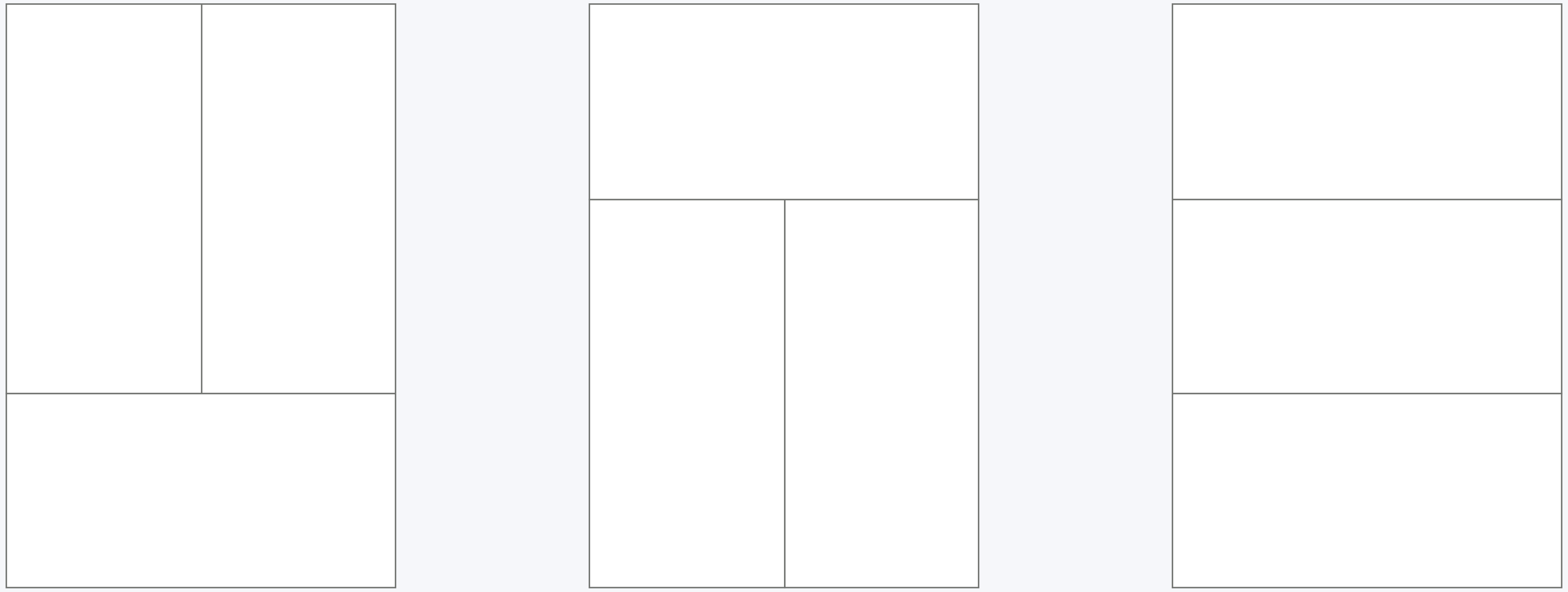
https://www.acmicpc.net/problem/2156

- C/C++
  - https://gist.github.com/Baekjoon/f042553b666185948f9f
- Java
  - https://gist.github.com/Baekjoon/5a8d8b46c4b2c608f3dd

타일 채우기

https://www.acmicpc.net/problem/2133

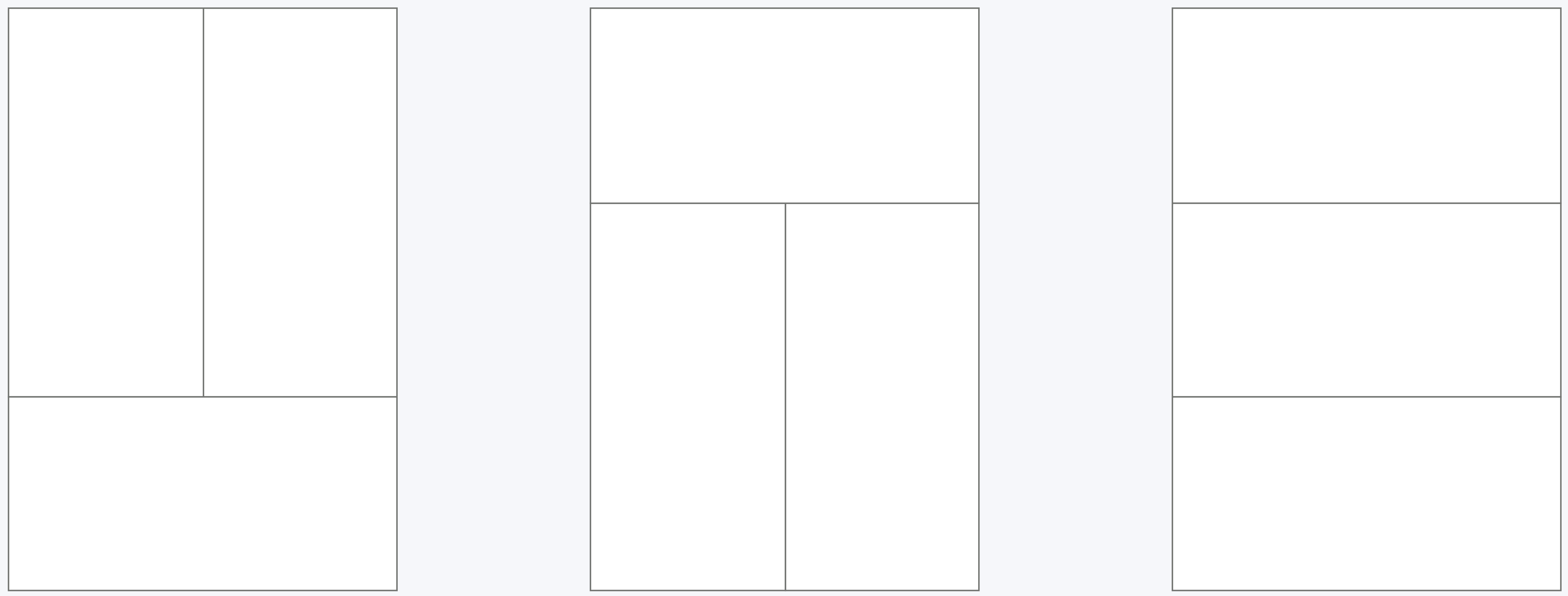
- 3×N을 1×2, 2×1로 채우는 방법의 수
- D[i] = 3×i를 채우는 방법의 수
- 마지막에 올 수 있는 가능한 경우의 수



타일 채우기

https://www.acmicpc.net/problem/2133

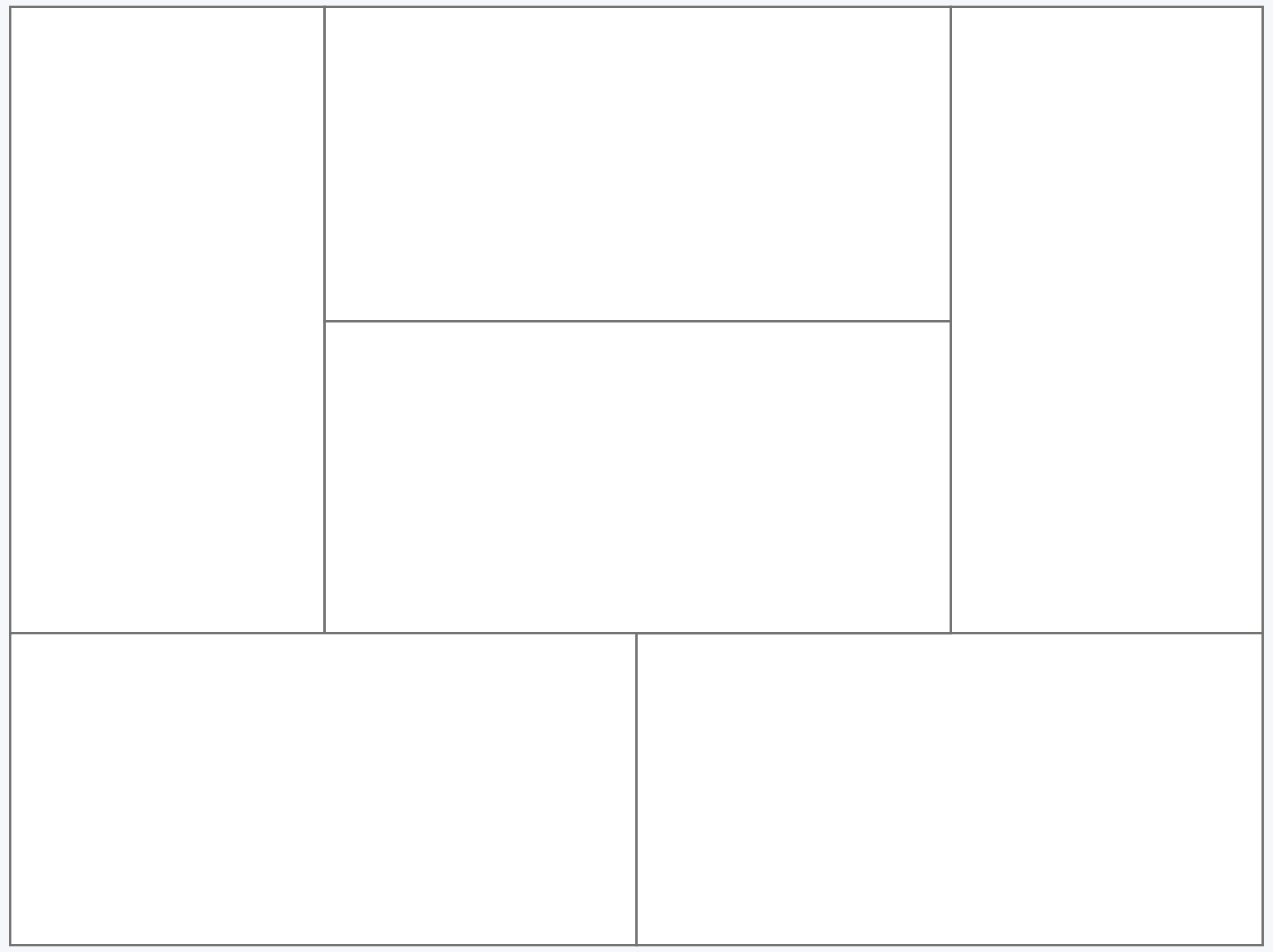
- 3×N을 1×2, 2×1로 채우는 방법의 수
- D[i] = 3×i를 채우는 방법의 수
- D[i] = 3 * D[i-2] (아니다)

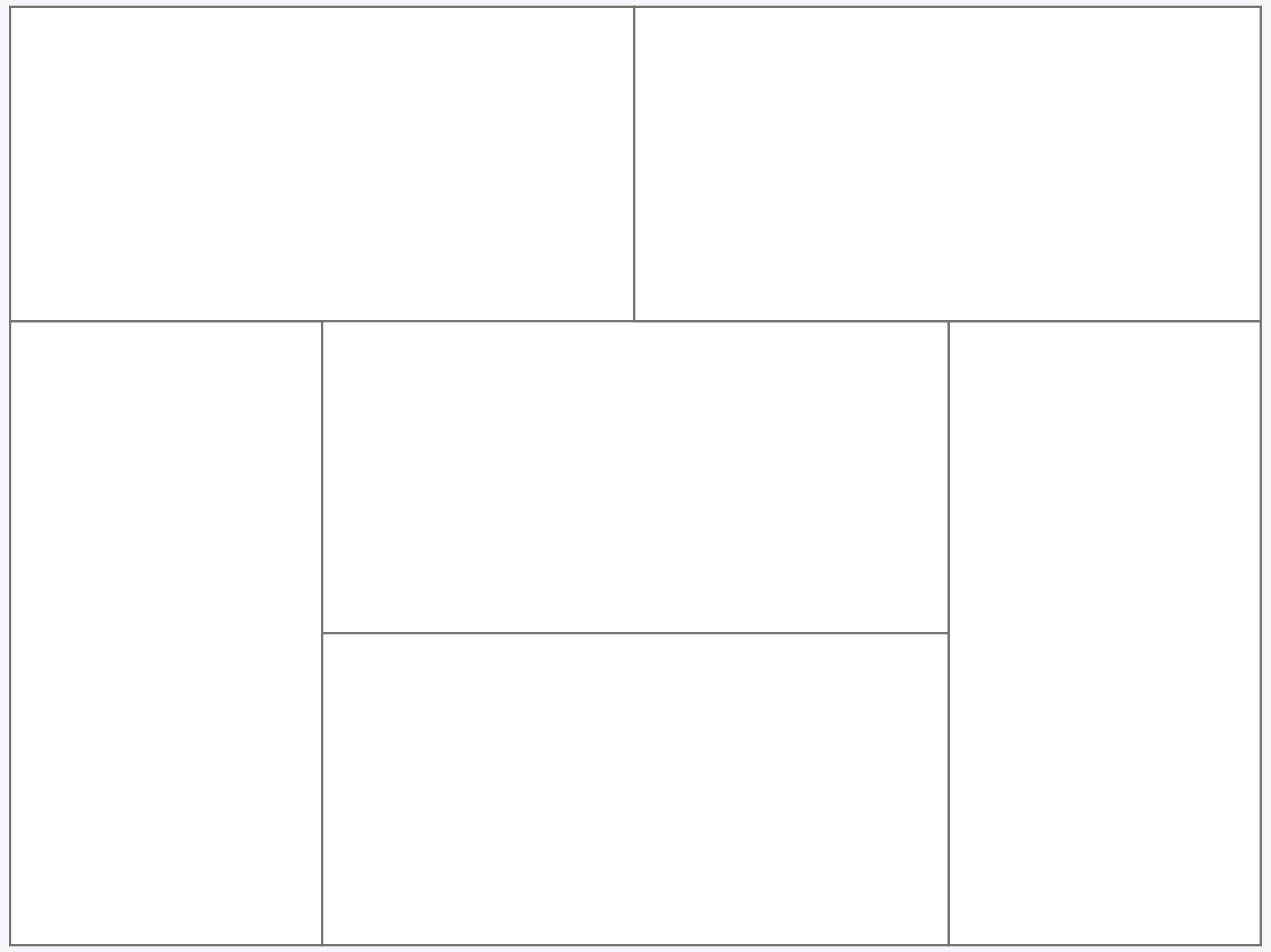


타일 채우기

https://www.acmicpc.net/problem/2133

- 3×N을 1×2, 2×1로 채우는 방법의 수
- D[i] = 3×i를 채우는 방법의 수
- 가능한 경우가 더 있다.

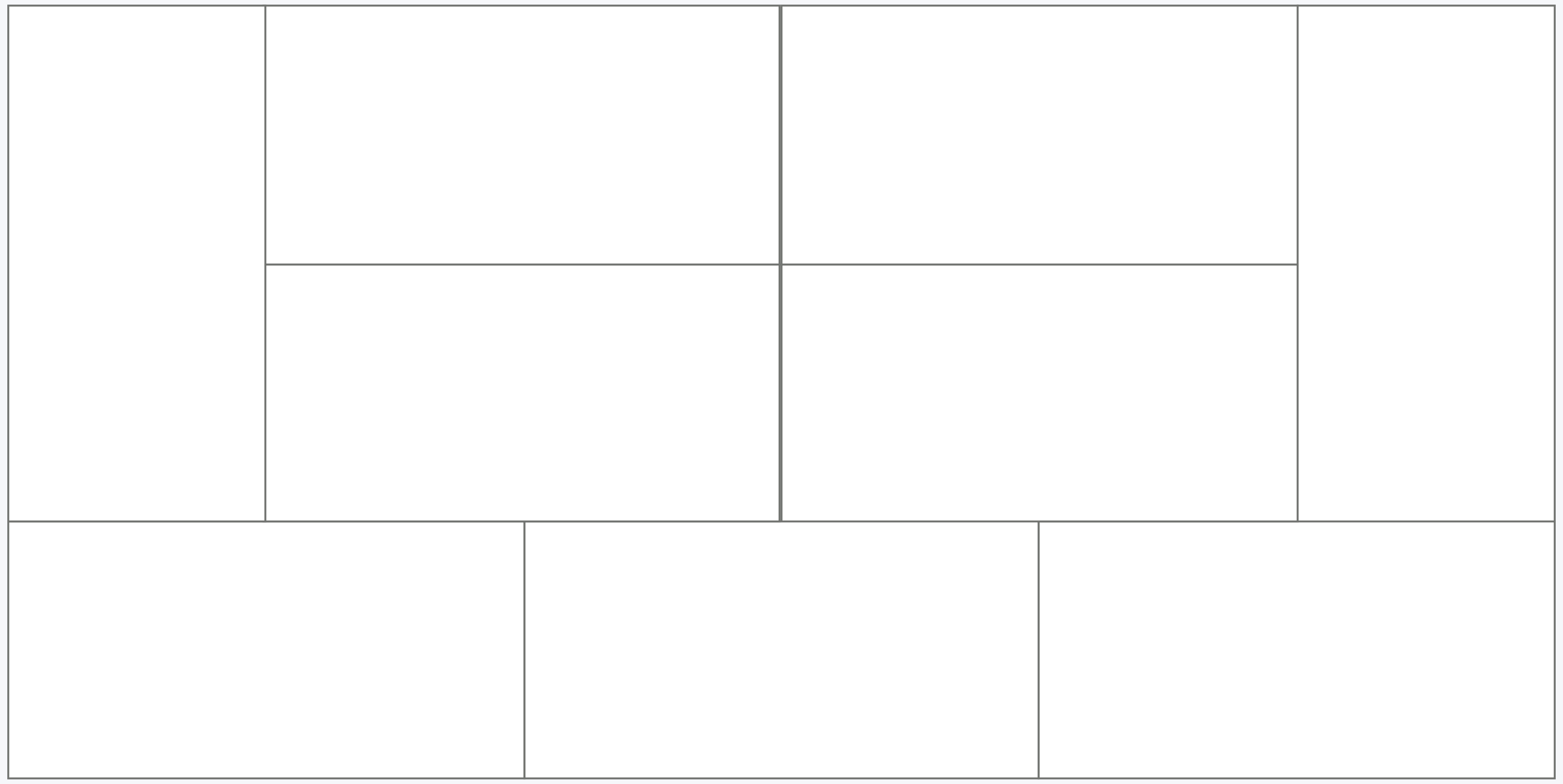


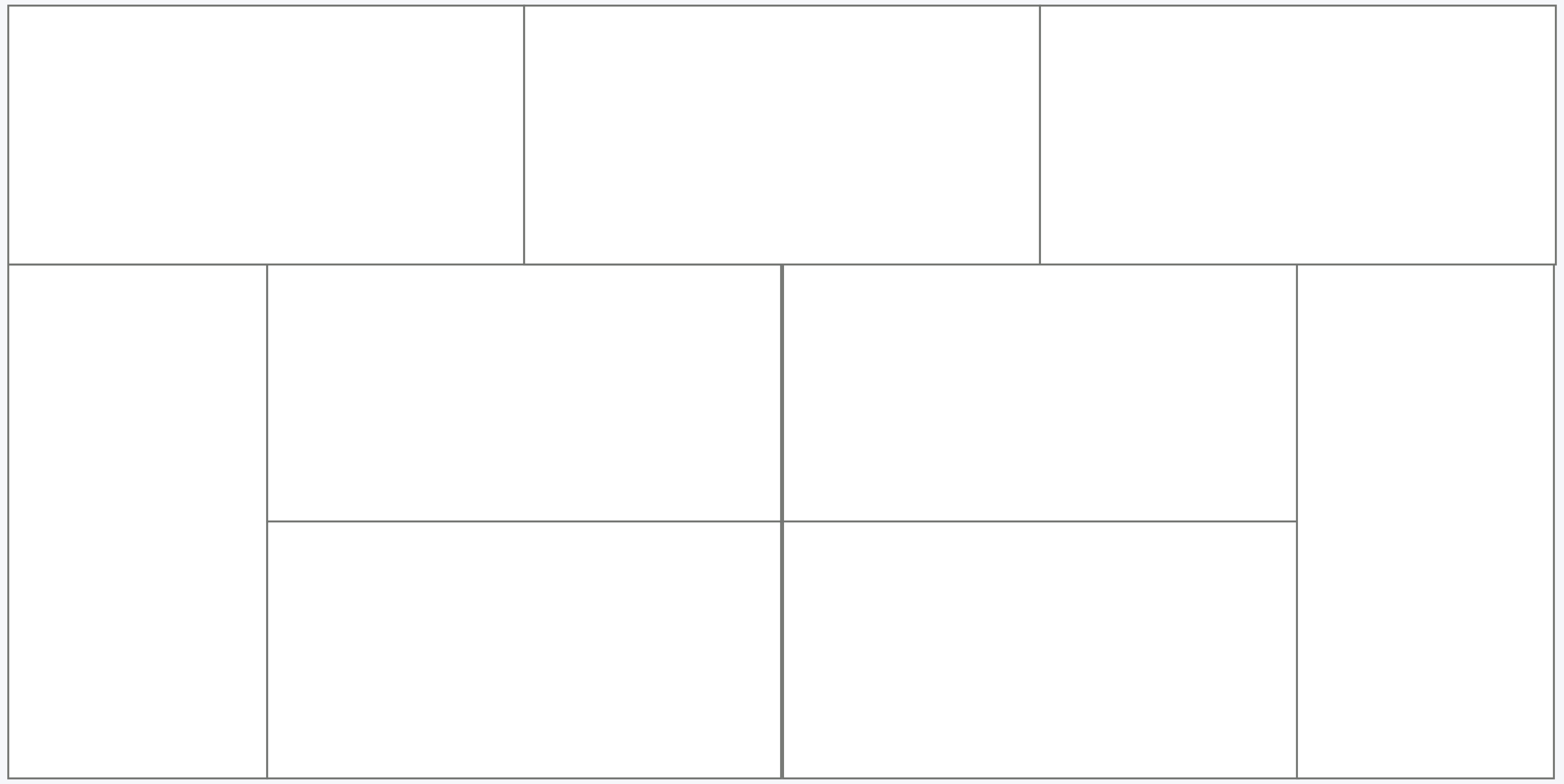


타일 채우기

https://www.acmicpc.net/problem/2133

- 3×N을 1×2, 2×1로 채우는 방법의 수
- D[i] = 3×i를 채우는 방법의 수
- 가능한 경우가 더 있다.





타일 채우기

https://www.acmicpc.net/problem/2133

- 3×N을 1×2, 2×1로 채우는 방법의 수
- D[i] = 3×i를 채우는 방법의 수
- D[i] = 3 * D[i-2] + 2*D[i-4] + 2*D[i-6] + …

타일 채우기

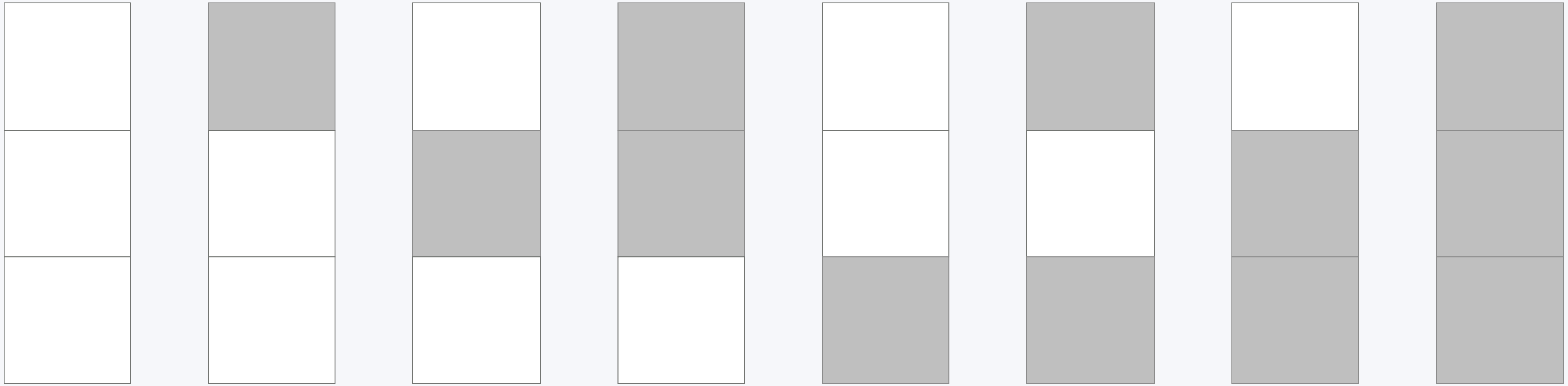
https://www.acmicpc.net/problem/2133

- C/C++
  - https://gist.github.com/Baekjoon/a27529894d77d469d252

타일 채우기

https://www.acmicpc.net/problem/2133

- 3×N을 1×2, 2×1로 채우는 방법의 수
- D[i][j] = 3×i를 채우는 방법의 수, i열의 상태는 j
- 마지막에 올 수 있는 가능한 경우의 수 (회색: 채워져 있는 칸)

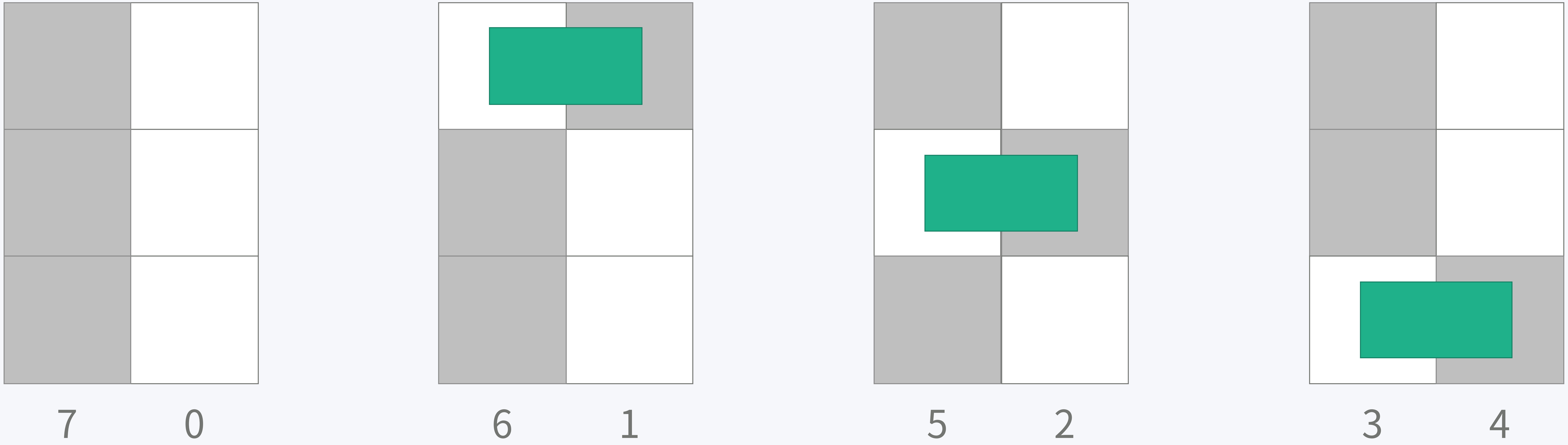


0 1 2 3 4 5 6 7

타일 채우기

https://www.acmicpc.net/problem/2133

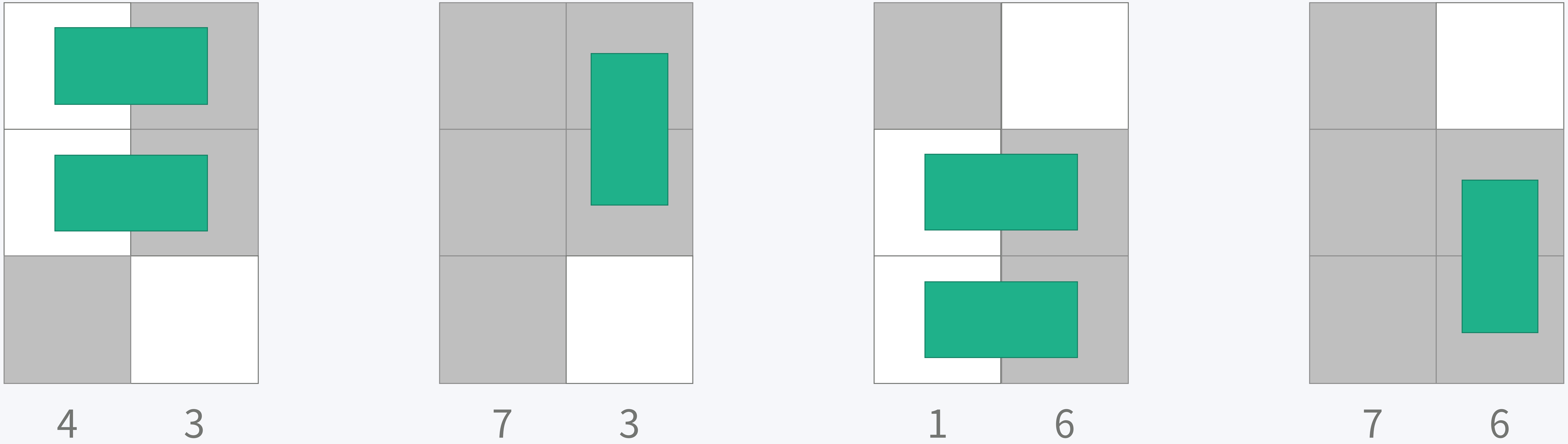
- i열을 채울 때, i-1에 빈 칸이 있으면 안된다



타일 채우기

https://www.acmicpc.net/problem/2133

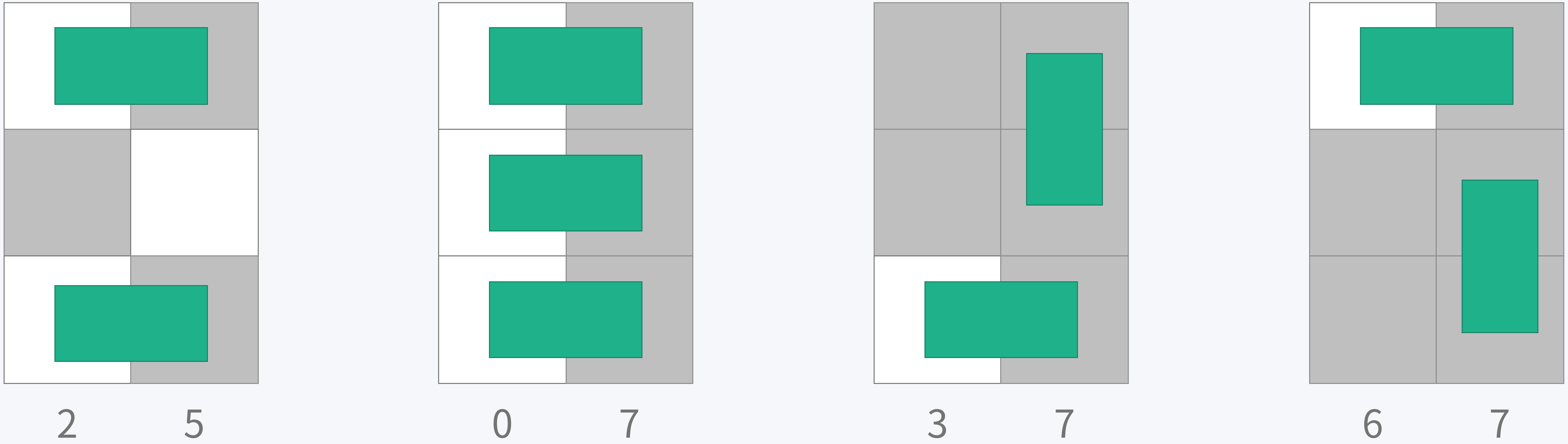
- i열을 채울 때, i-1에 빈 칸이 있으면 안된다



타일 채우기

https://www.acmicpc.net/problem/2133

- i열을 채울 때, i-1에 빈 칸이 있으면 안된다



타일 채우기

https://www.acmicpc.net/problem/2133

- D[i][0] = D[i-1][7]
- D[i][1] = D[i-1][6]
- D[i][2] = D[i-1][5]
- D[i][4] = D[i-1][3]
- D[i][3] = D[i-1][4] + D[i-1][7]
- D[i][6] = D[i-1][1] + D[i-1][7]
- D[i][5] = D[i-1][2]
- D[i][7] = D[i-1][0] + D[i-1][3] + D[i-1][6]

타일 채우기

https://www.acmicpc.net/problem/2133

```
D[0][7] = 1;
for (int i=1; i<=n; i++) {
    D[i][0] = D[i-1][7];
    D[i][1] = D[i-1][6];
    D[i][2] = D[i-1][5];
    D[i][4] = D[i-1][3];
    D[i][3] = D[i-1][4] + D[i-1][7];
    D[i][6] = D[i-1][1] + D[i-1][7];
    D[i][5] = D[i-1][2];
    D[i][7] = D[i-1][0] + D[i-1][3] + D[i-1][6];
}
```

타일 채우기

https://www.acmicpc.net/problem/2133

- C++
  - https://gist.github.com/Baekjoon/fefbc054071fa3fdb80b

파도반 수열

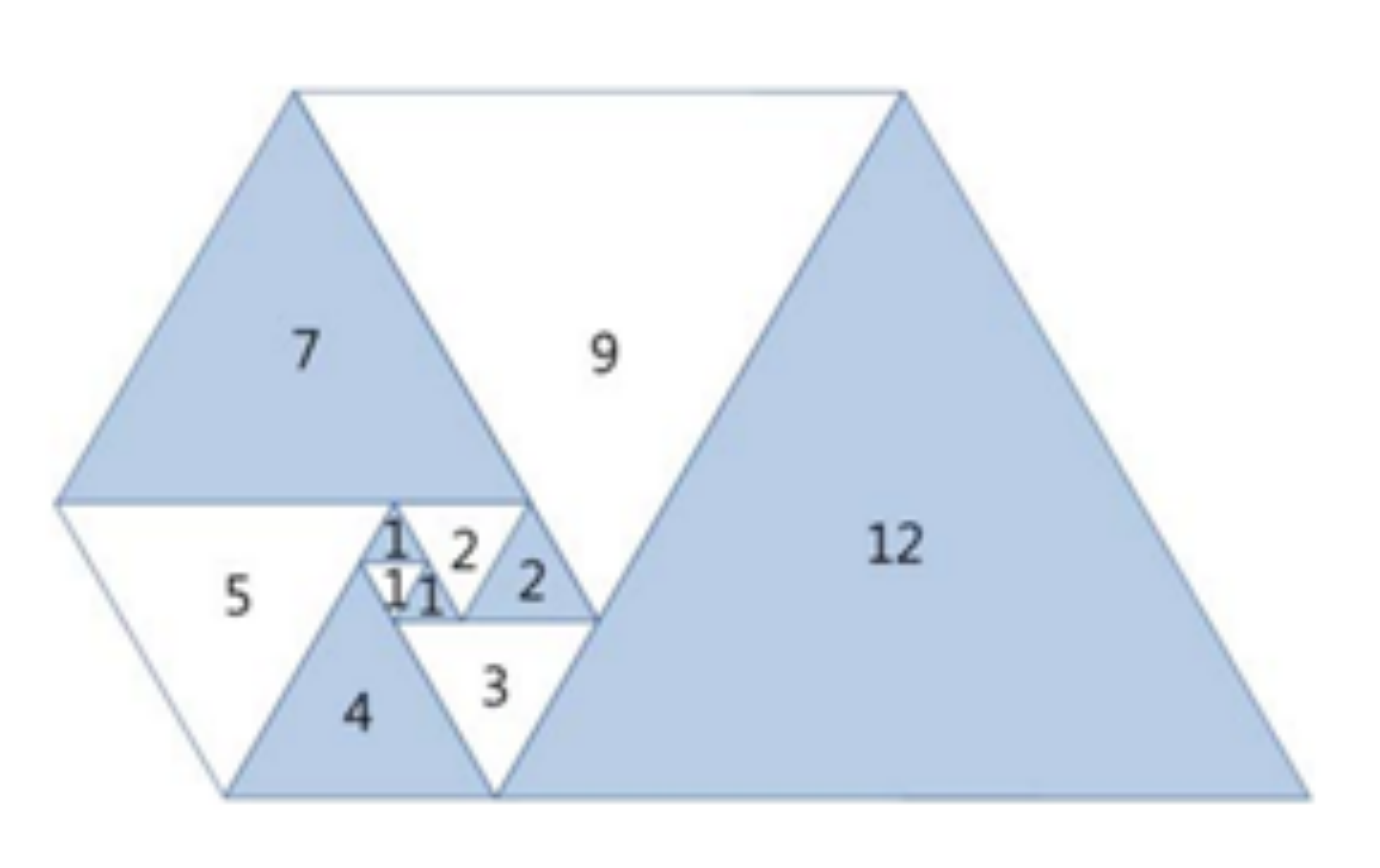
https://www.acmicpc.net/problem/9461

- 오른쪽 그림과 같이 삼각형이 나선 모양으로 놓여져 있다
- 첫 삼각형은 정삼각형으로 변의 길이는 1이다
- 그 다음에는 다음과 같은 과정으로 정삼각형을 계속 추가한다
- 나선에서 가장 긴 변의 길이를 k라 했을 때, 그 변에 길이가 k인 정삼각형을 추가한다
- 파도반 수열 P(N)은 나선에 있는 정삼각형의 변의 길이이다
- P(1)부터 P(10)까지 첫 10개 숫자는 1, 1, 1, 2, 2, 3, 4, 5, 7, 9이다
- N이 주어졌을 때, P(N)을 구하는 문제

파도반 수열

https://www.acmicpc.net/problem/9461

- 그림을 보고 유추할 수 있다.
- D[i] = D[i-1] + D[i-5]



파도반 수열

https://www.acmicpc.net/problem/9461

- C/C++
  - https://gist.github.com/Baekjoon/3ca4e70487835f651bae
- Java
  - https://gist.github.com/Baekjoon/e7826c911f92e1fb0369

이동하기

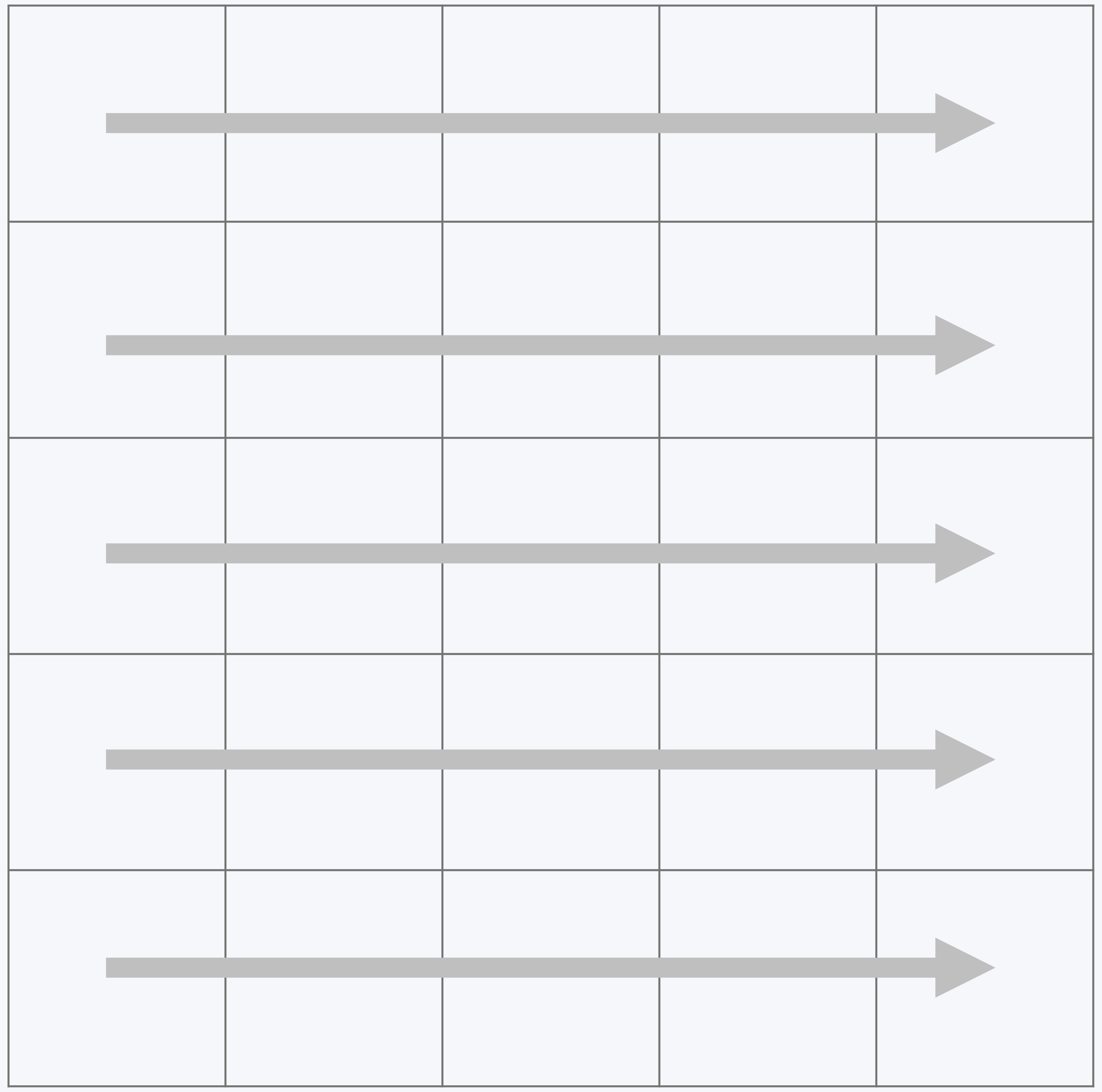
https://www.acmicpc.net/problem/11048

- 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)

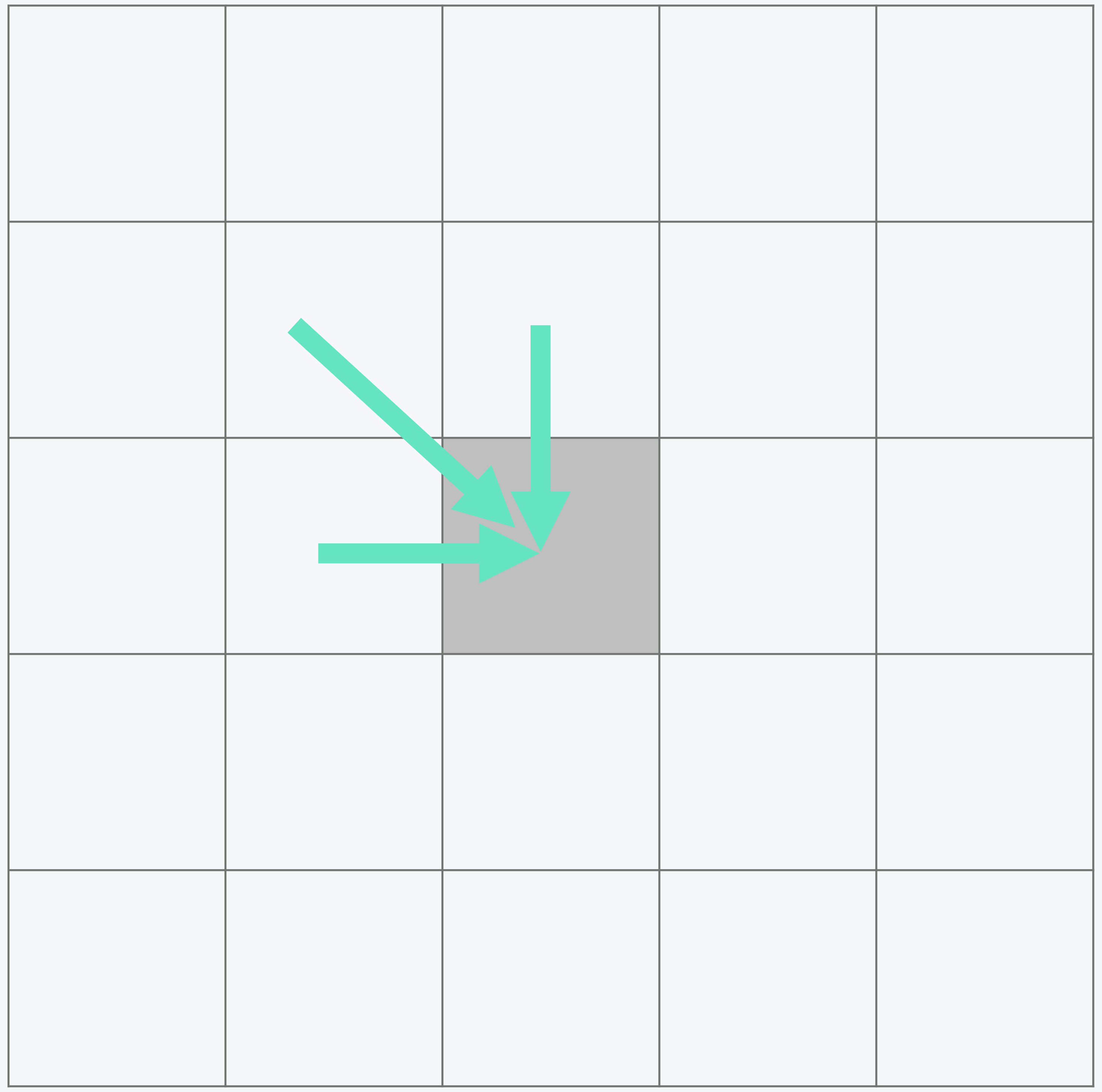


방법 1

이동하기

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

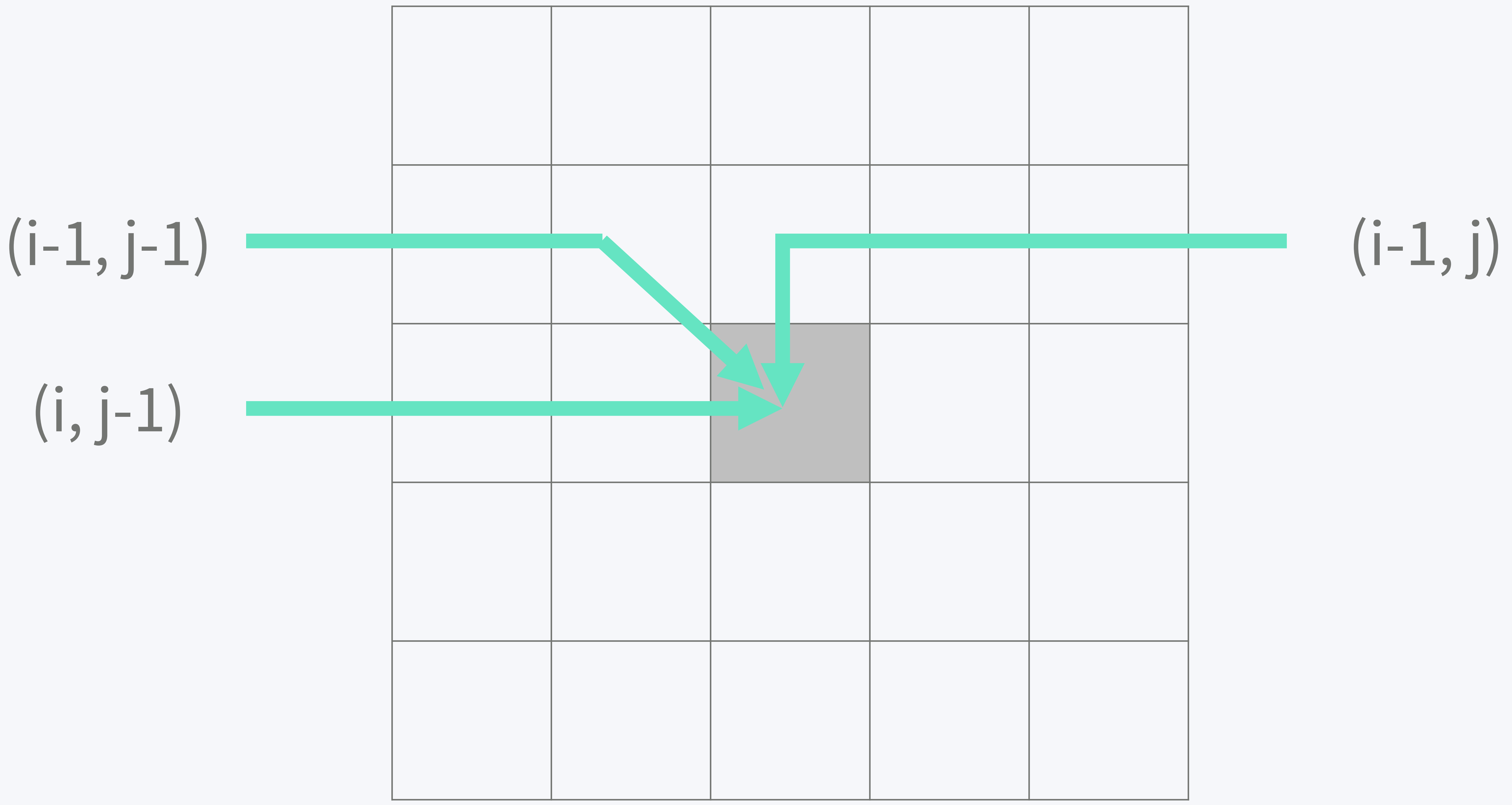
- 항상 아래와 오른쪽으로만 갈 수 있다.
- (i,j)에서 가능한 이동: (i+1, j), (i, j+1), (i+1, j+1)



이동하기

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

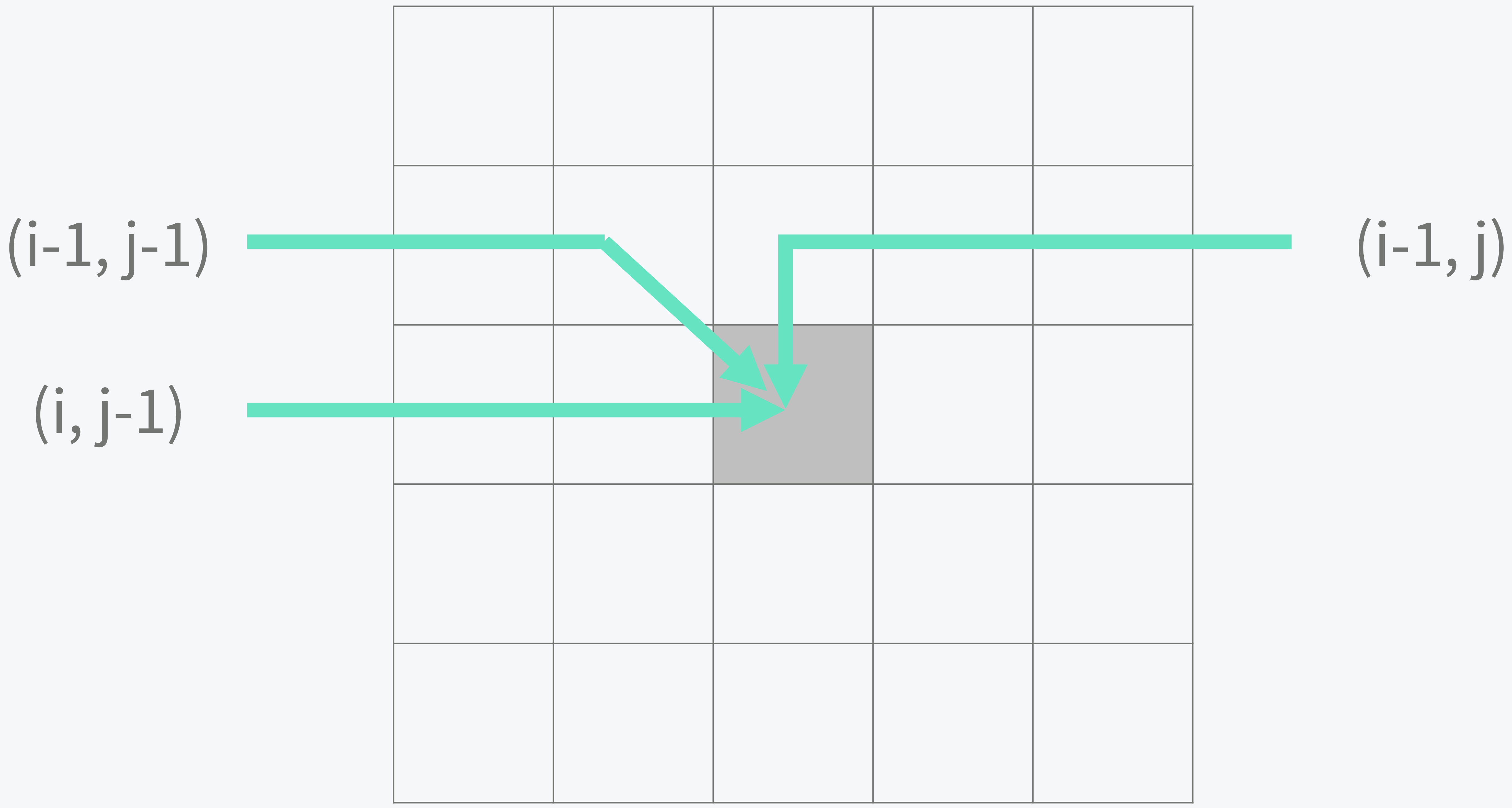
- 항상 아래와 오른쪽으로만 갈 수 있다.
- (i,j)에서 가능한 이동: (i+1, j), (i, j+1), (i+1, j+1)



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]



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];
    }
}
```

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
- 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

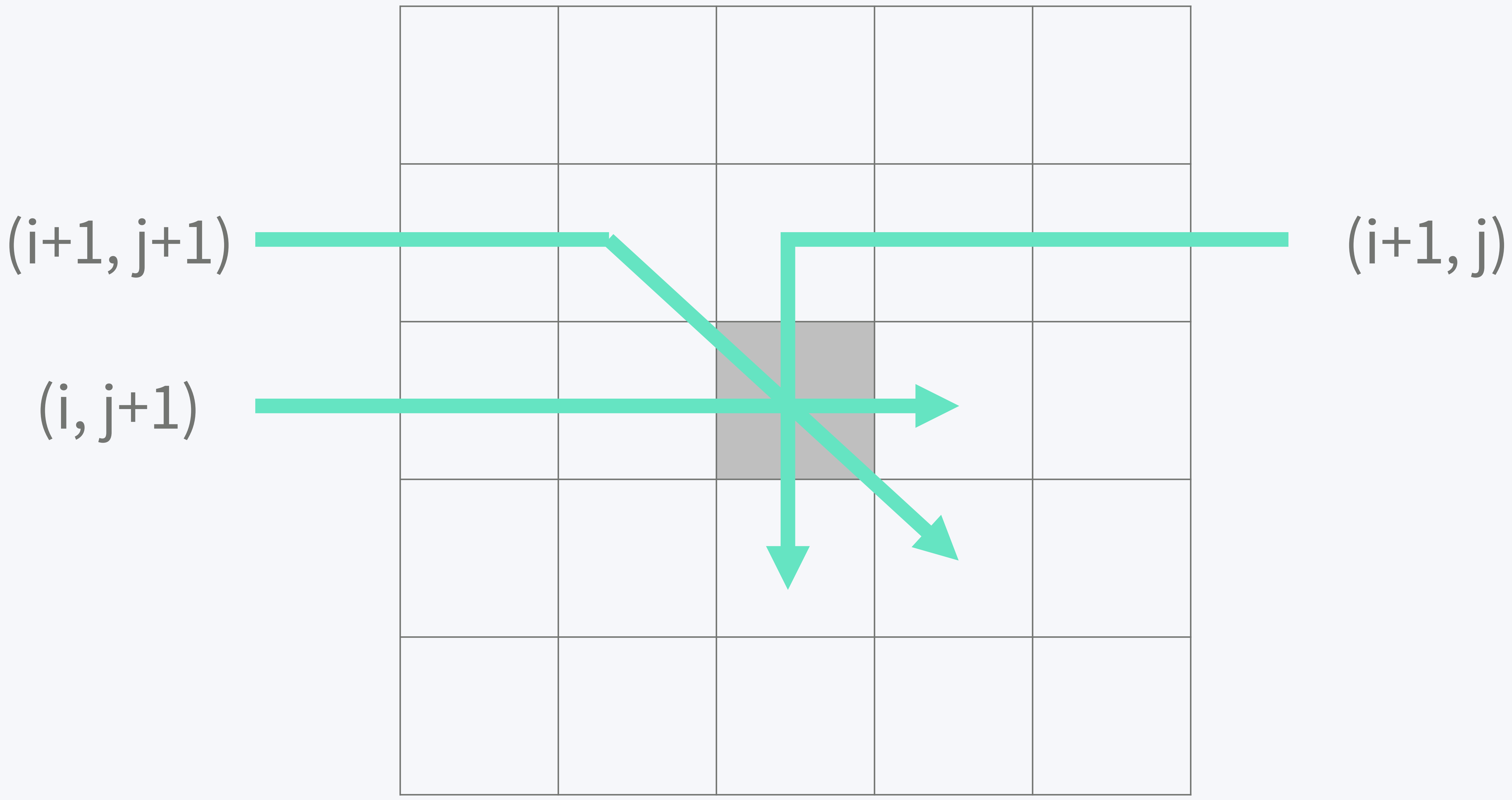
- C/C++
  - https://gist.github.com/Baekjoon/65f34d5cf5f329dde337
- Java
  - https://gist.github.com/Baekjoon/51fc0cd6a1b2db1a4d48

방법 2

이동하기

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

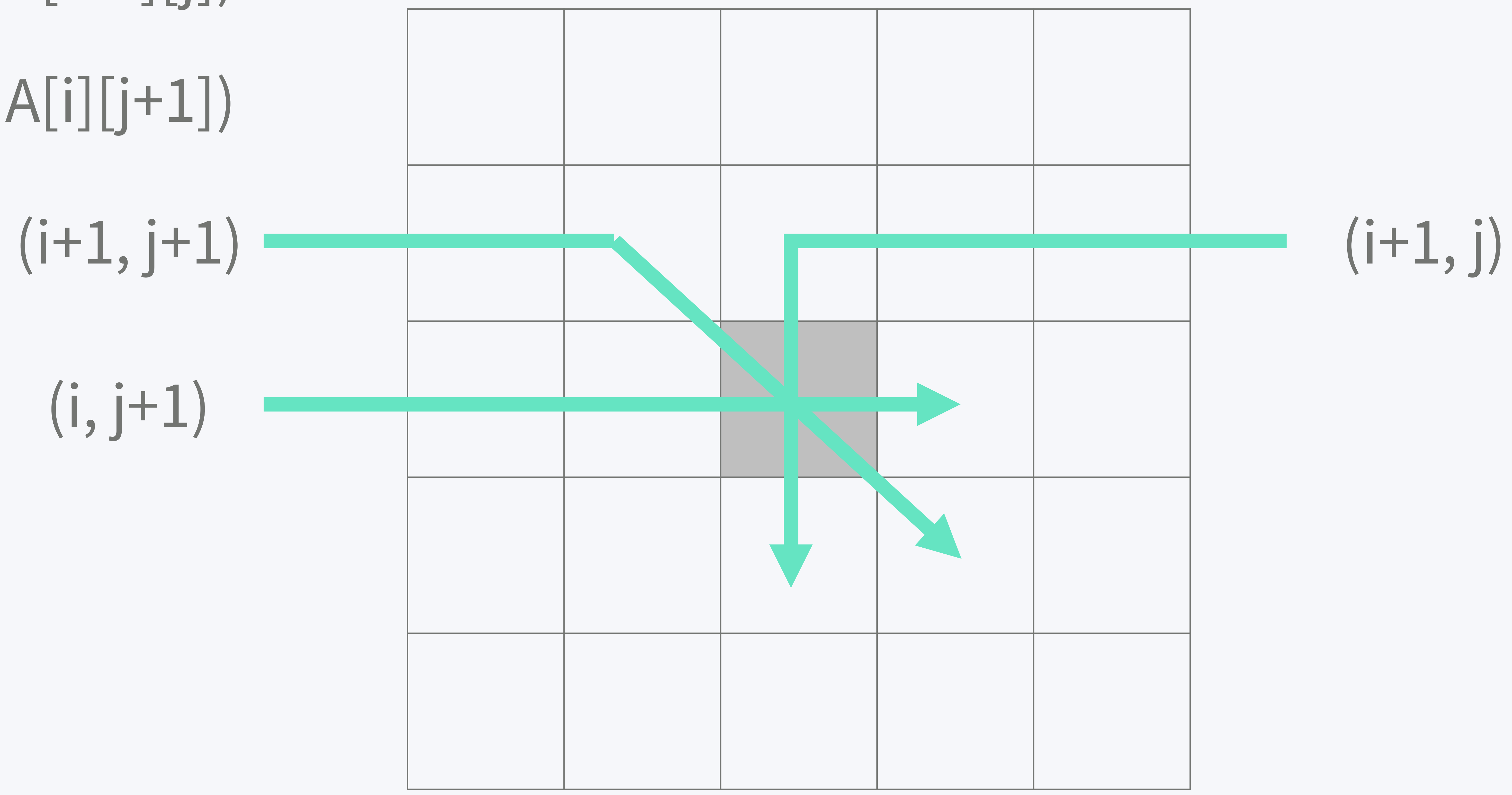
- 항상 아래와 오른쪽으로만 갈 수 있다.
- (i,j)에서 가능한 이동: (i+1, j), (i, j+1), (i+1, j+1)



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

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

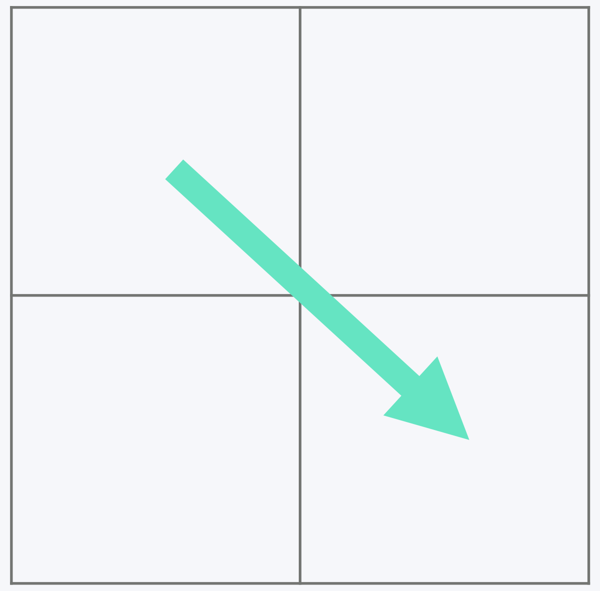
- C/C++
  - https://gist.github.com/Baekjoon/ce48db57373eabc2beeb
- Java
  - https://gist.github.com/Baekjoon/df34e7d5daf341c8b76a

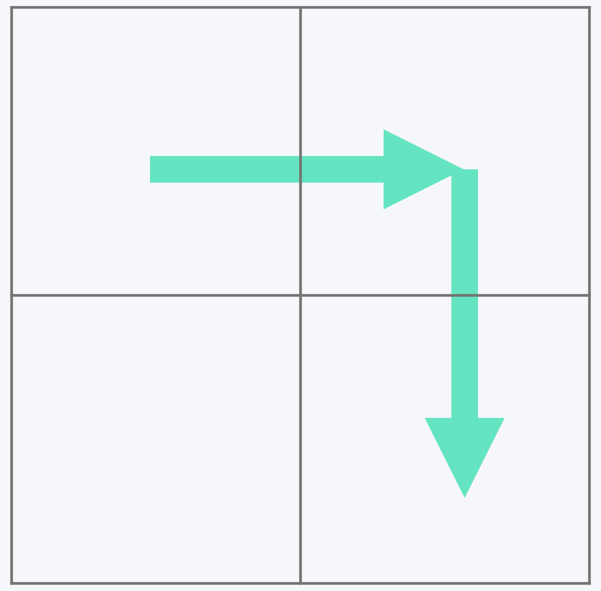
방법 3

이동하기

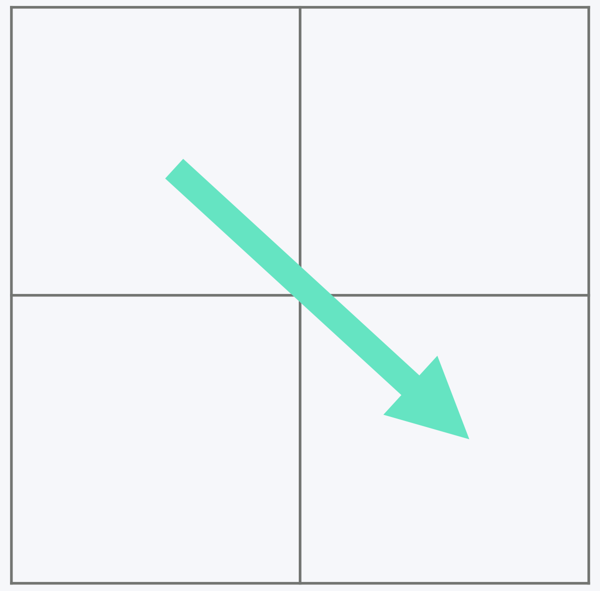
https://www.acmicpc.net/problem/11048

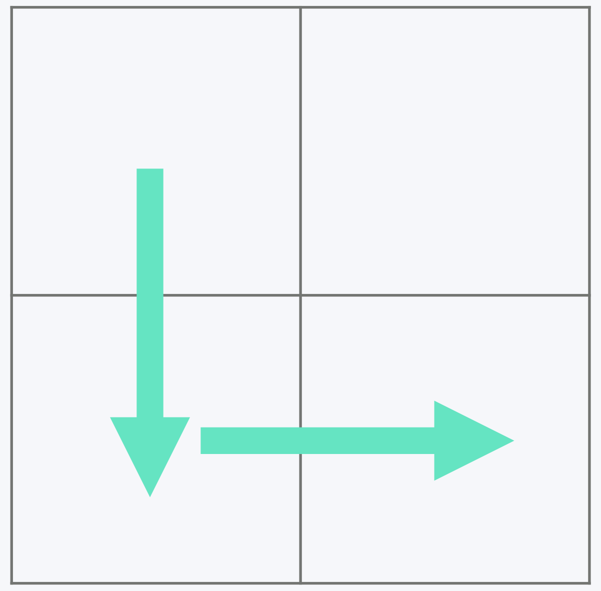
- 대각선 이동은 처리하지 않아도 된다
- 대각선 이동은 다른 2가지를 포함한 방법보다 항상 작거나 같다
- $A[i][j] + A[i+1][j+1] \leq A[i][j] + A[i][j+1] + A[i+1][j+1]$





- $A[i][j] + A[i+1][j+1] \leq A[i][j] + A[i+1][j] + A[i+1][j+1]$





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

방법 5

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
- 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

```
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

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, $0 \le k < j$)
- D[i][j] += D[k][j] (k+A[k][j] == i, $0 \le k < i$)

- 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