다이나믹 프로그래밍

최백준 choi@startlink.io

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

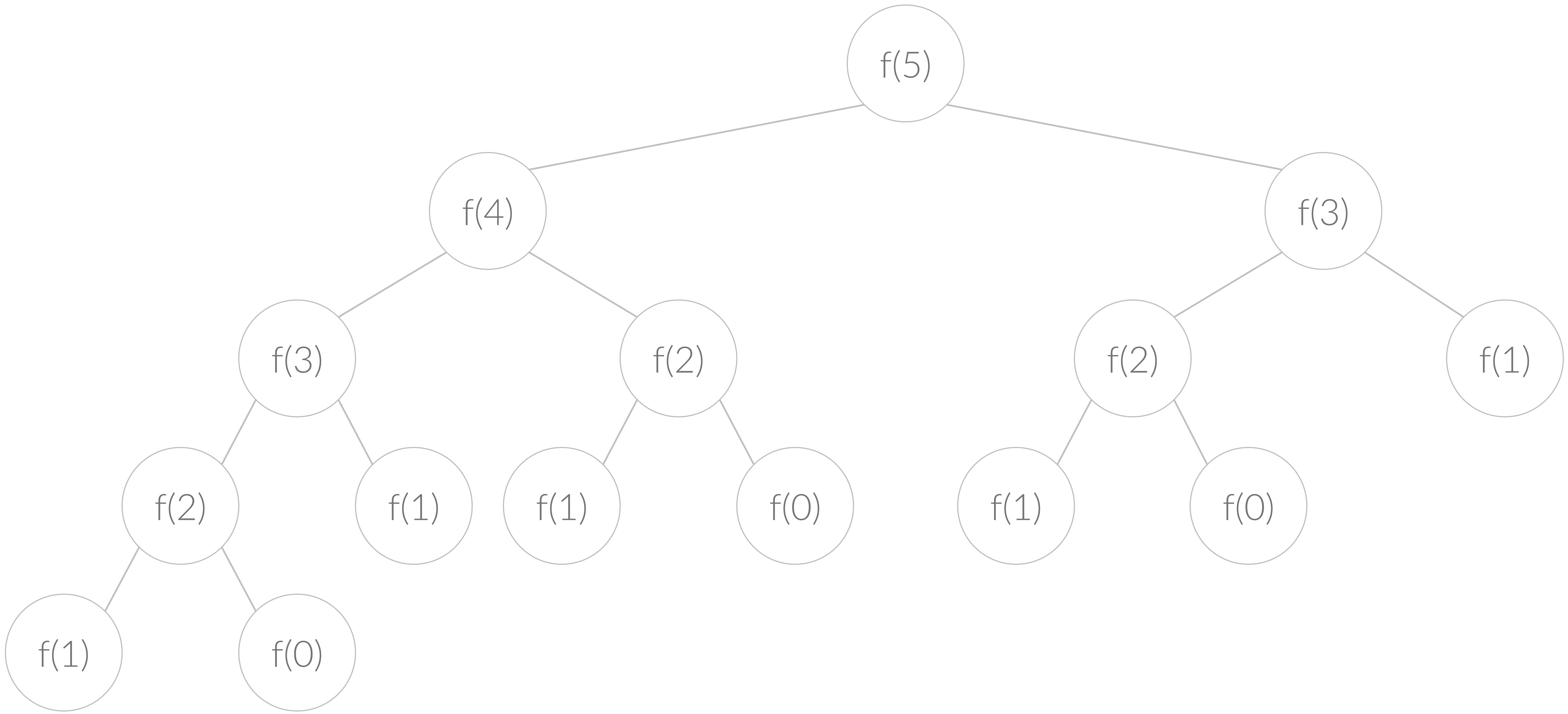
```
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

- 피보나치 수를 구하는 함수이다.

피보나치 수

Dynamic Programming

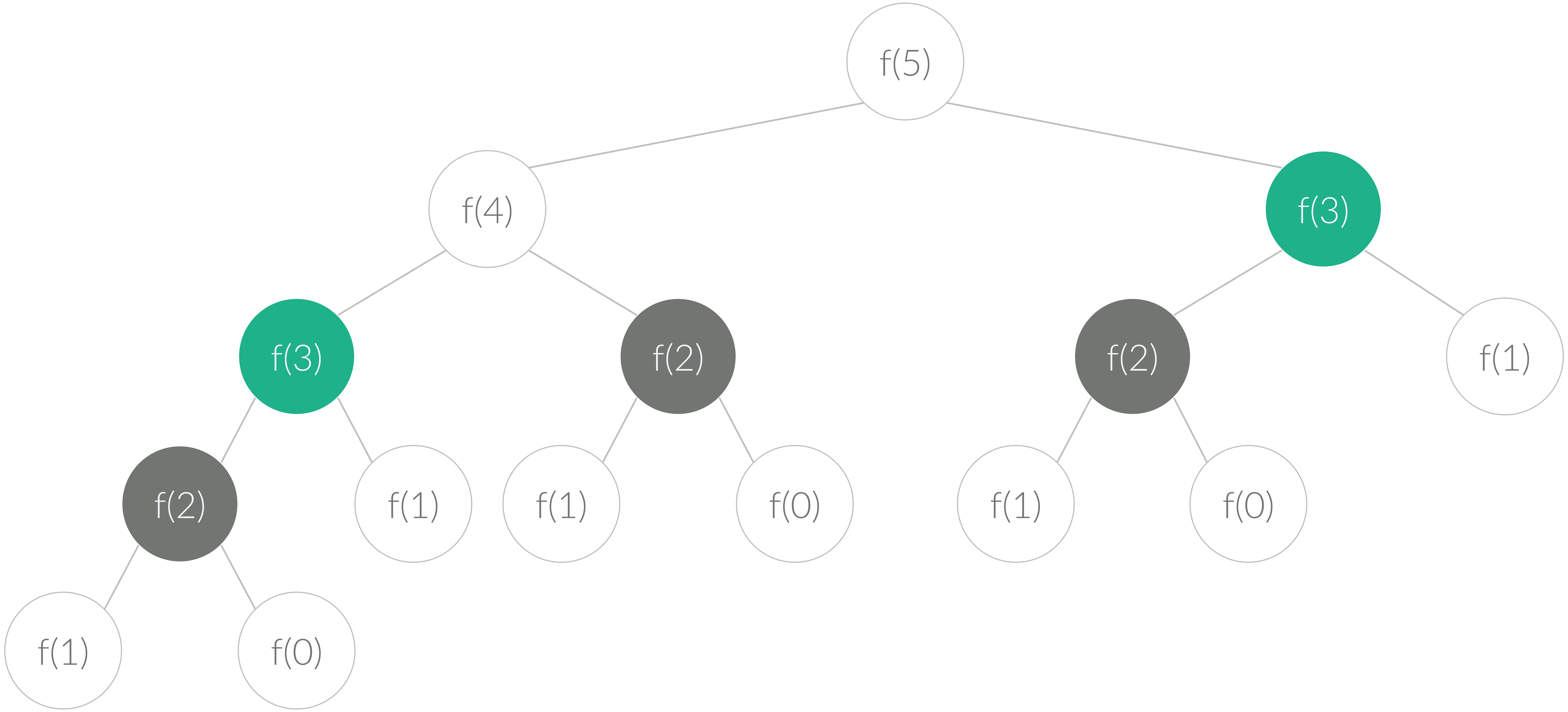
- fibonacci(5)를 호출한 경우 함수가 어떻게 호출되는지를 나타낸 그림



피보나치 수

Dynamic Programming

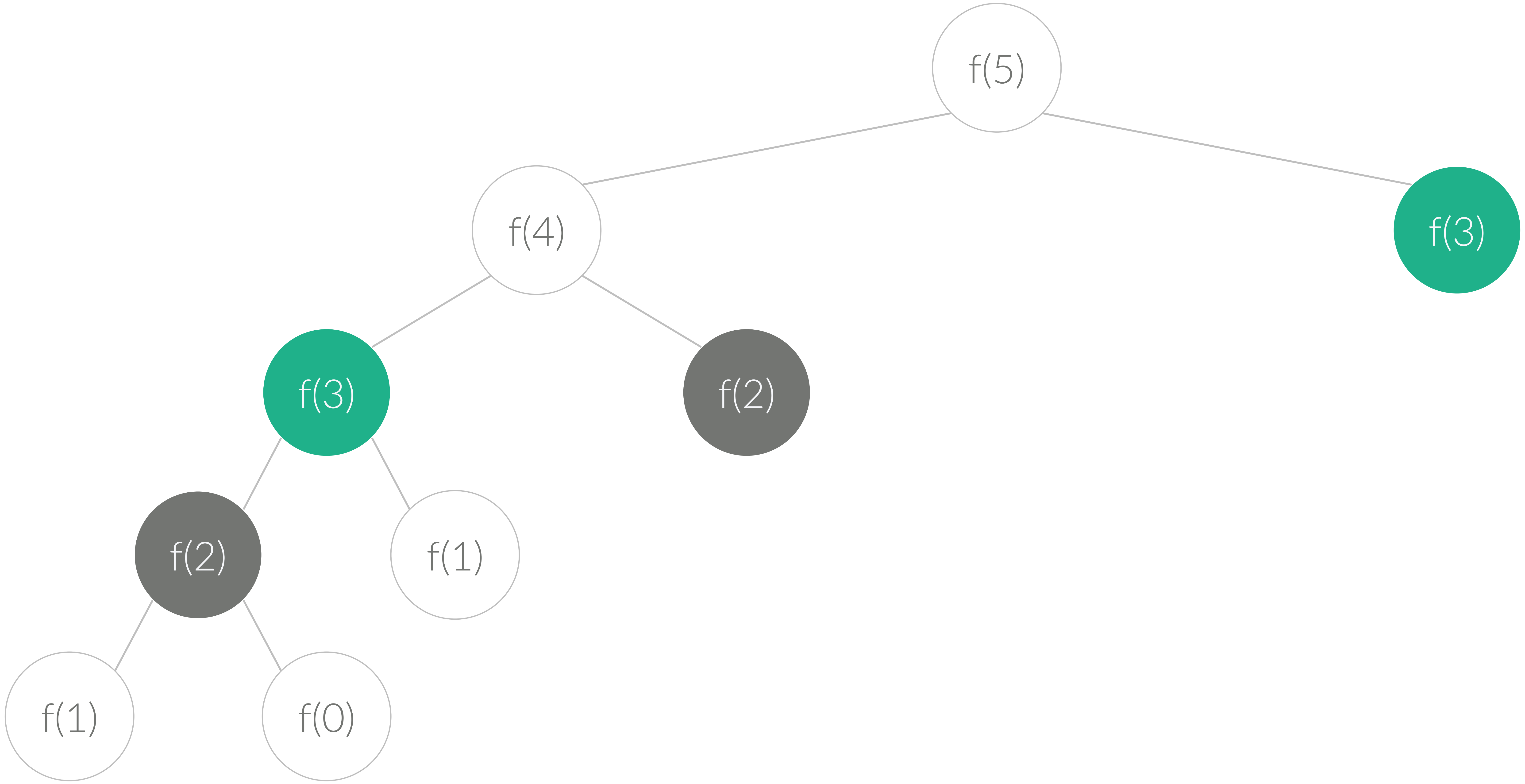
- 아래 그림과 같이 겹치는 호출이 생긴다.



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 {
        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] = 0;
    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

- 정수 X에 사용할 수 있는 연산은 다음과 같이 세 가지

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

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

1로 만들기

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

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

- N을 1로 만드려고 한다.
- N을 작게 만들어야 한다.
- 3으로 나누는 것이 수를 빠르게 작게 만든다.
- 3으로 나누는 것, 2로 나누는 것, 1을 빼는 우선 순위로 N을 1로 만들어 본다.

1로 만들기

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

- 3으로 나누는 것, 2로 나누는 것, 1을 빼는 우선 순위로 N을 1로 만들어 본다.
- 이 방법은 정답을 구할 수 없다.

1로 만들기

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

- 3으로 나누는 것, 2로 나누는 것, 1을 빼는 우선 순위로 N을 1로 만들어 본다.
- 이 방법은 정답을 구할 수 없다.
- 반례: 10

1로 만들기

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

- 정수 X에 사용할 수 있는 연산은 다음과 같이 세 가지

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

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

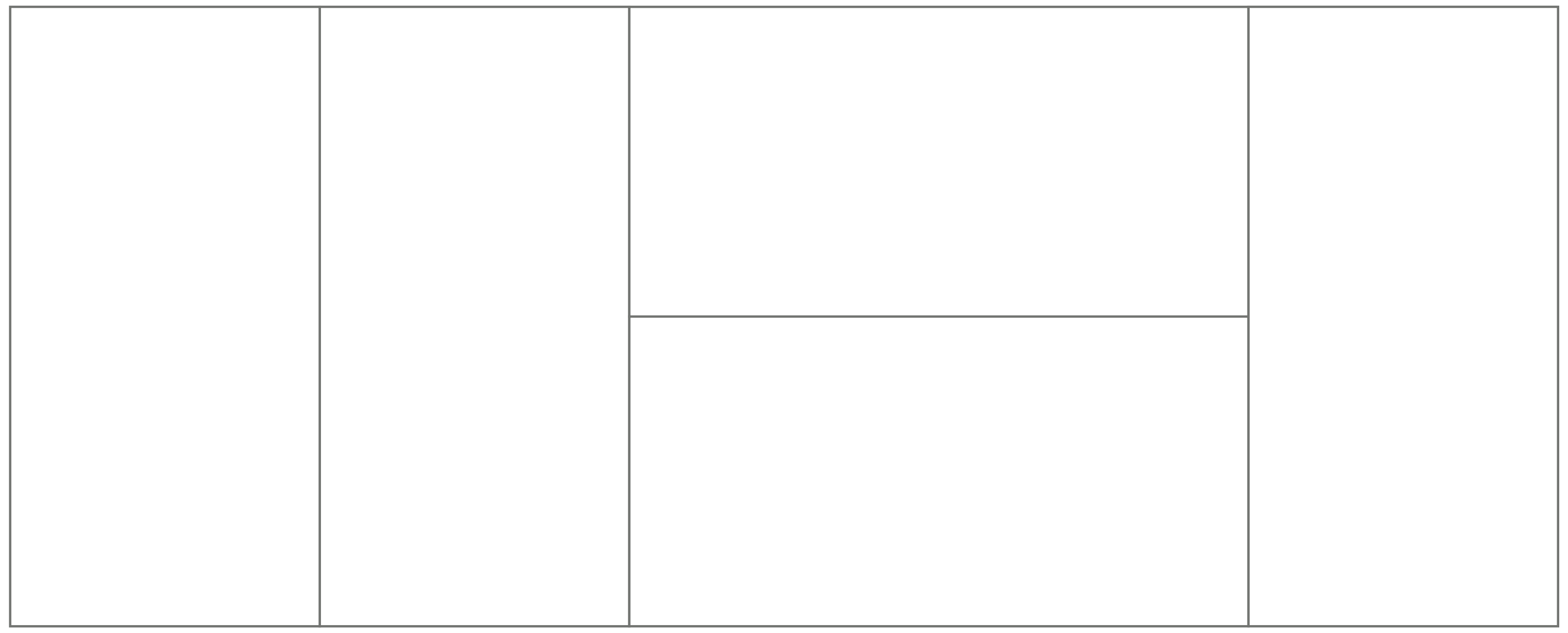
https://www.acmicpc.net/problem/1463

- Top-Down 방식 소스: http://codeplus.codes/237b841e0eb044f2876f08ac991ec2d7
- Bottom-up 방식 소스: http://codeplus.codes/c5f9e9a3a9c7436f9e8b46ad894b69b3

2×n 타일링

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

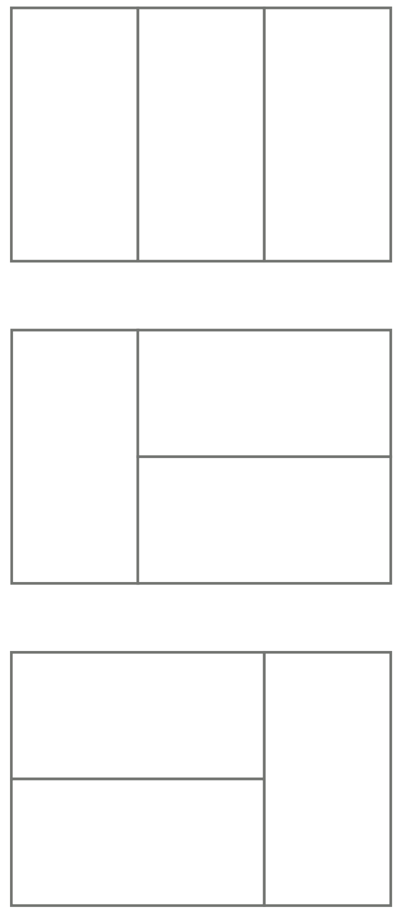
- 2×n 직사각형을 1×2, 2×1타일로 채우는 방법의 수
- 아래 그림은 2×5를 채우는 방법의 수
- D[i] = 2×i 직사각형을 채우는 방법의 수



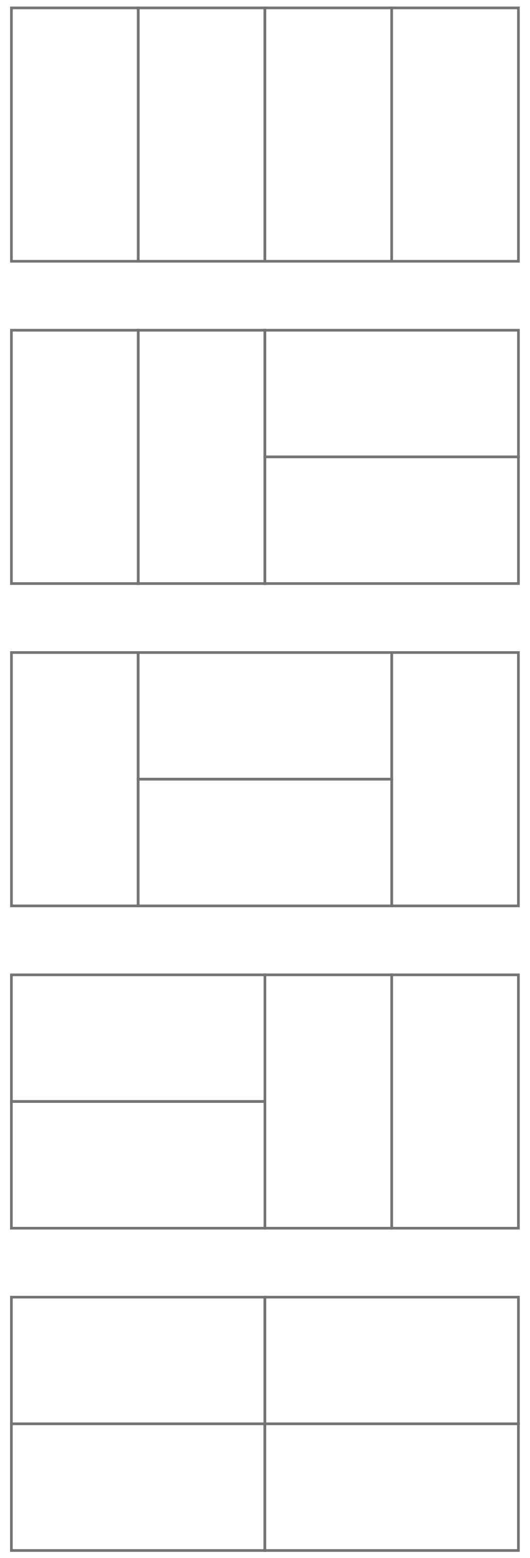
2×n 타일링

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

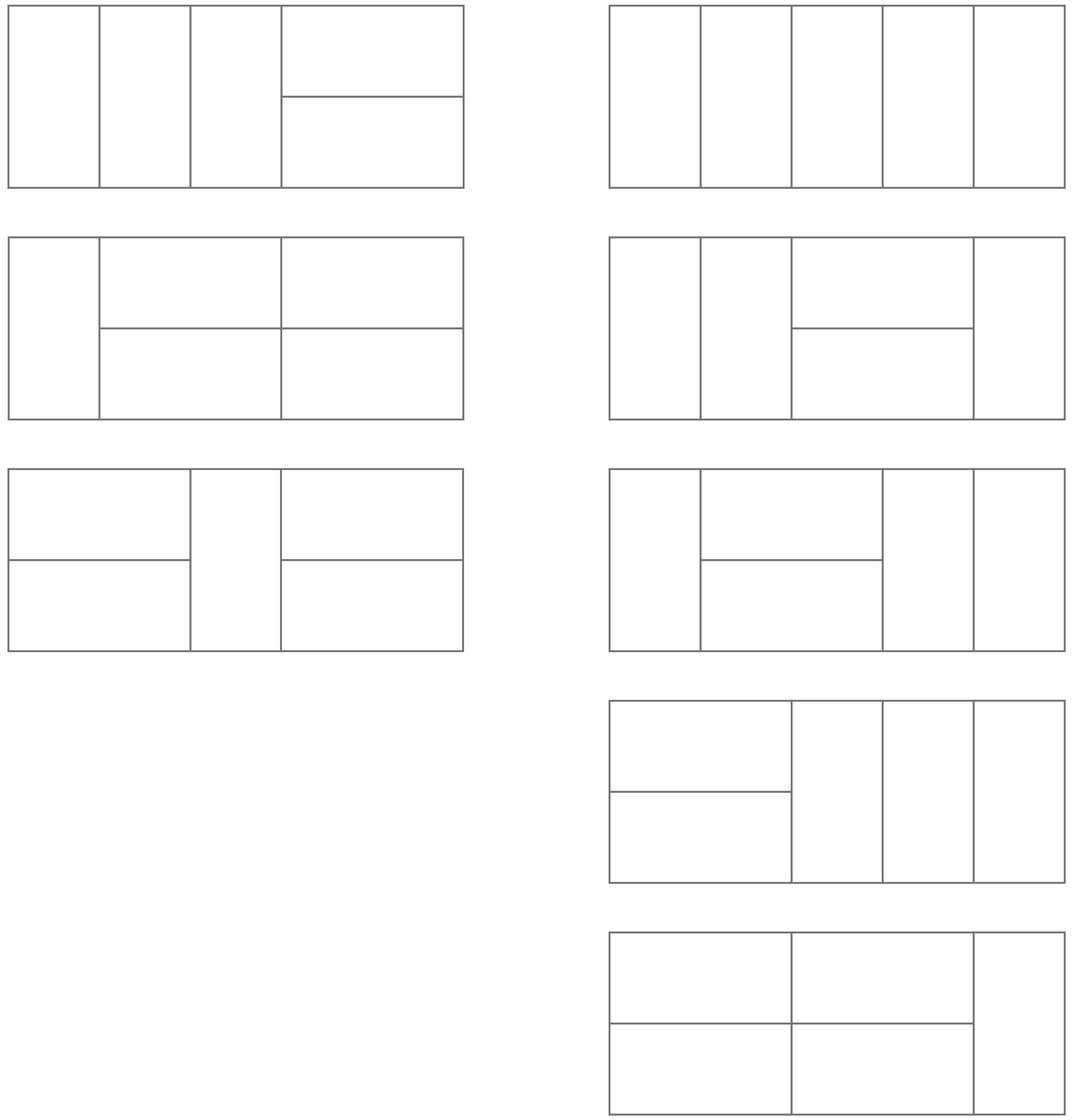
2×3



2×4



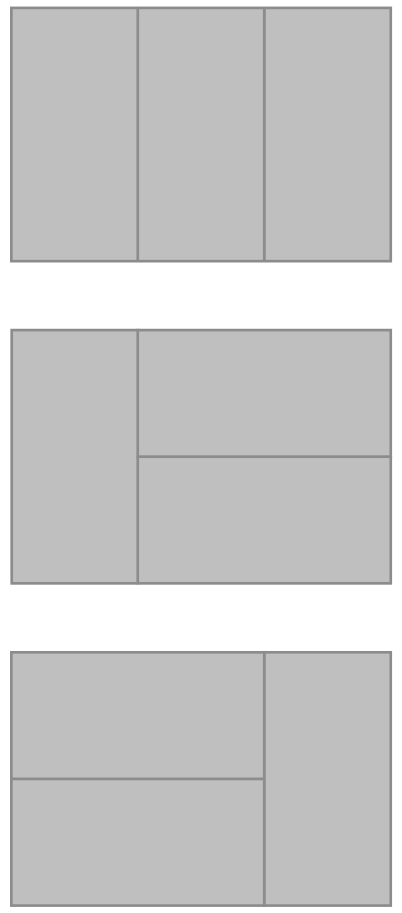
2×5



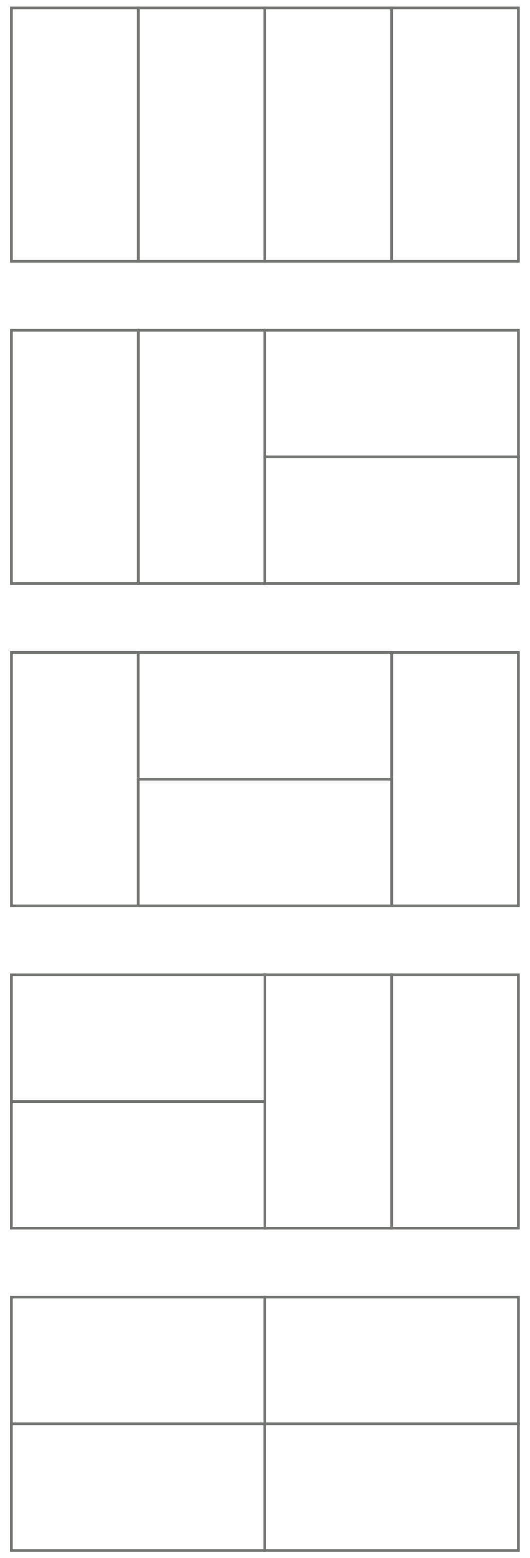
2×n 타일링

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

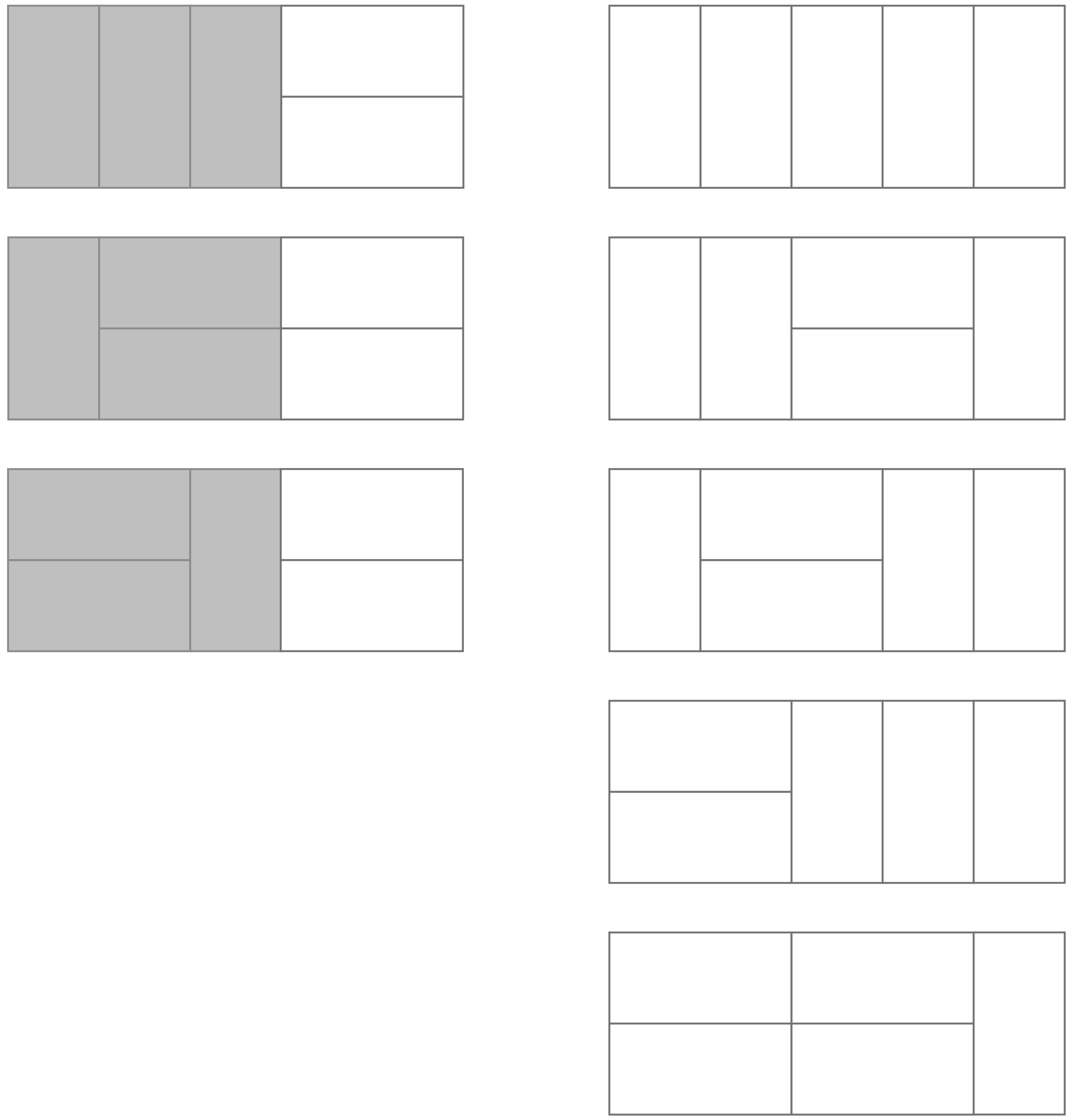
2×3



2×4



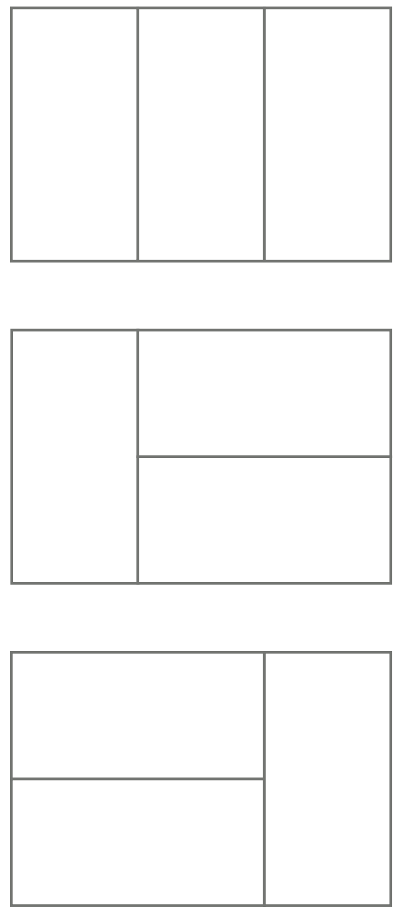
2×5



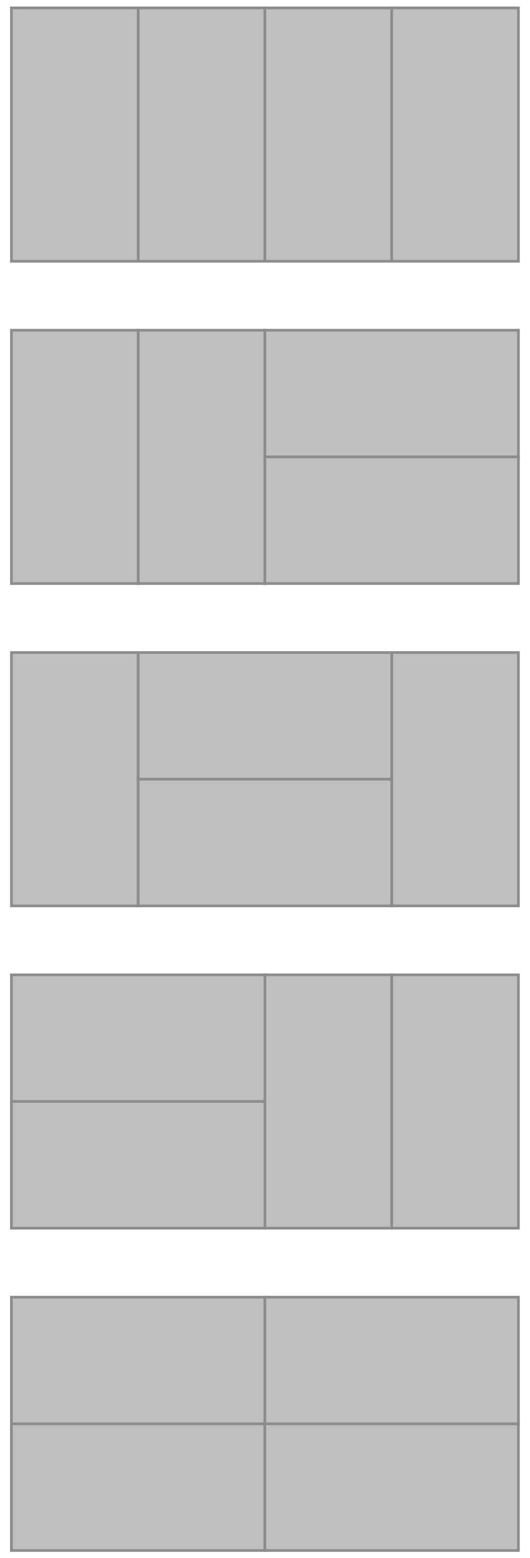
2×n 타일링

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

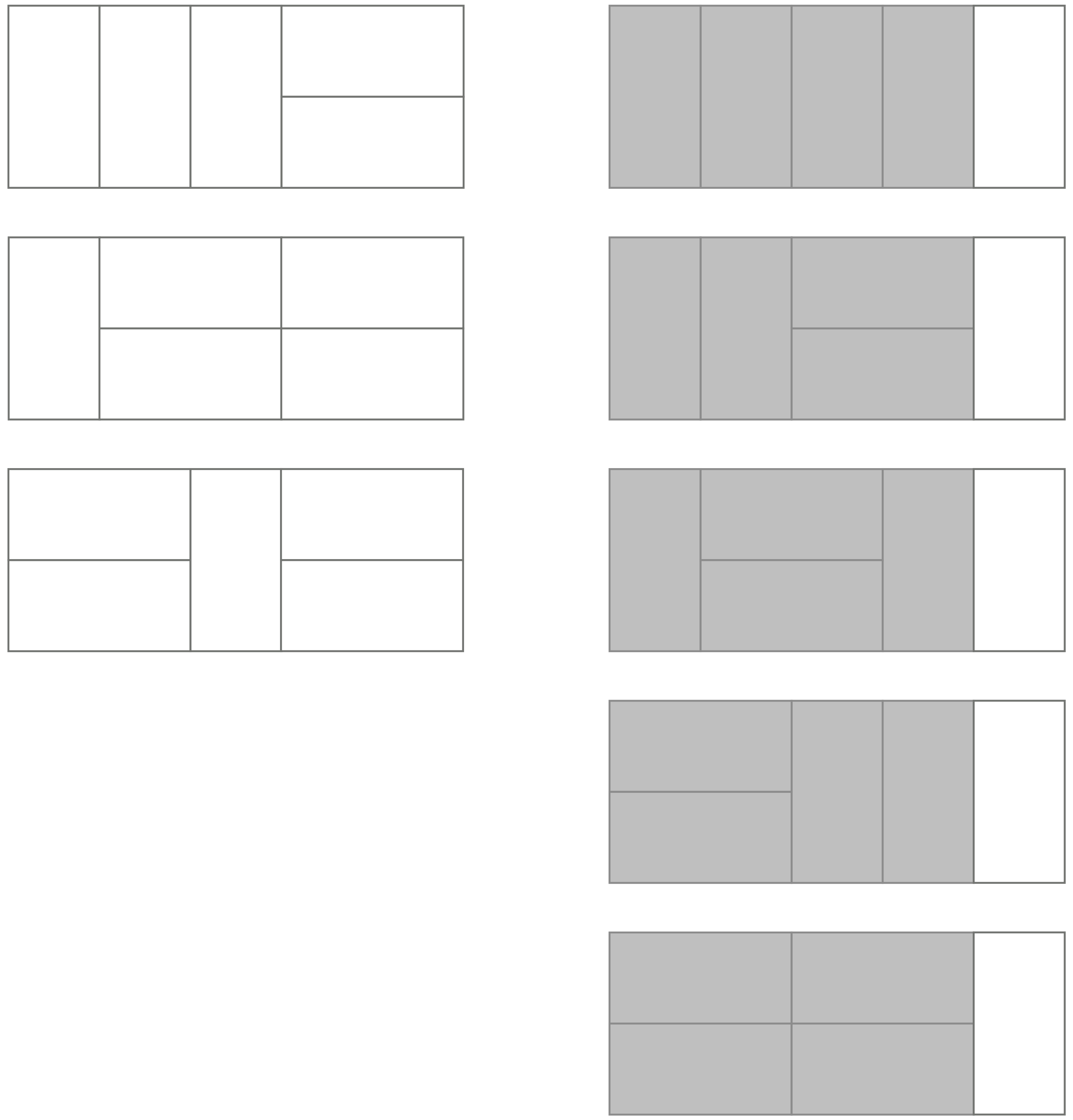
2×3



2×4



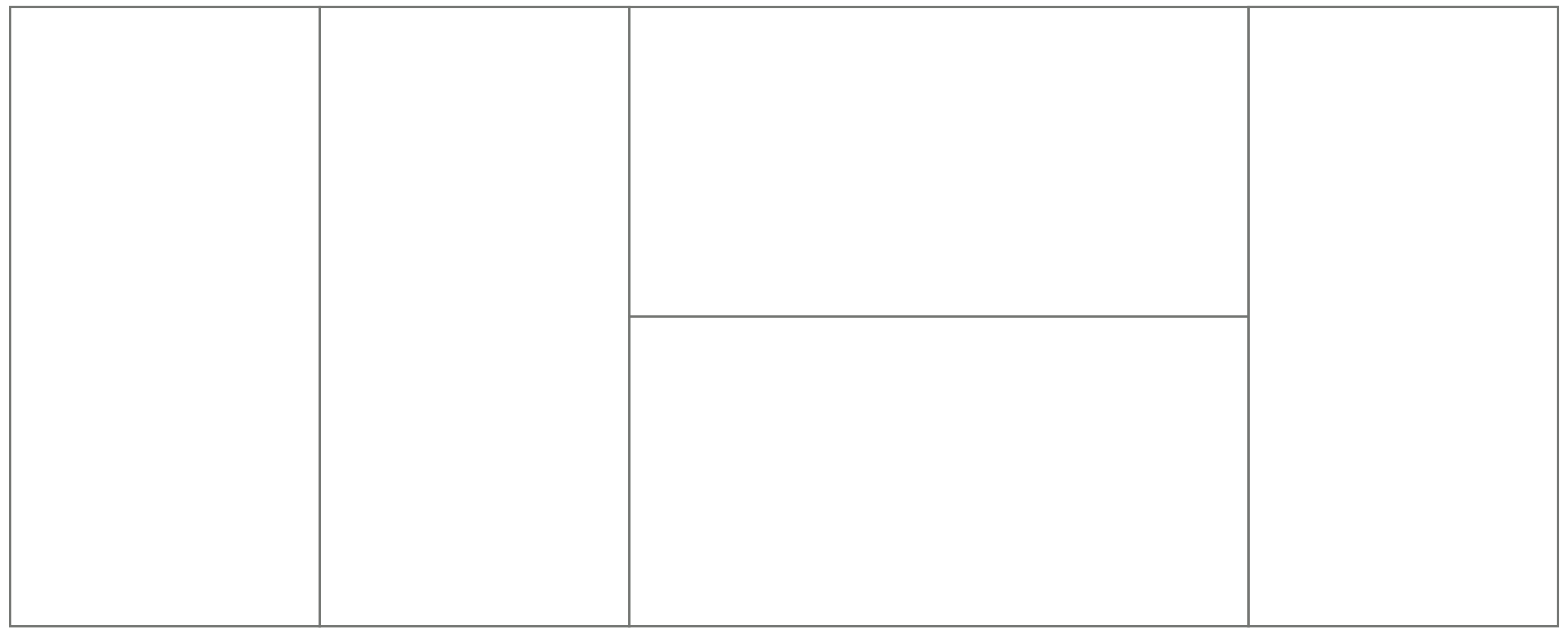
2×5



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

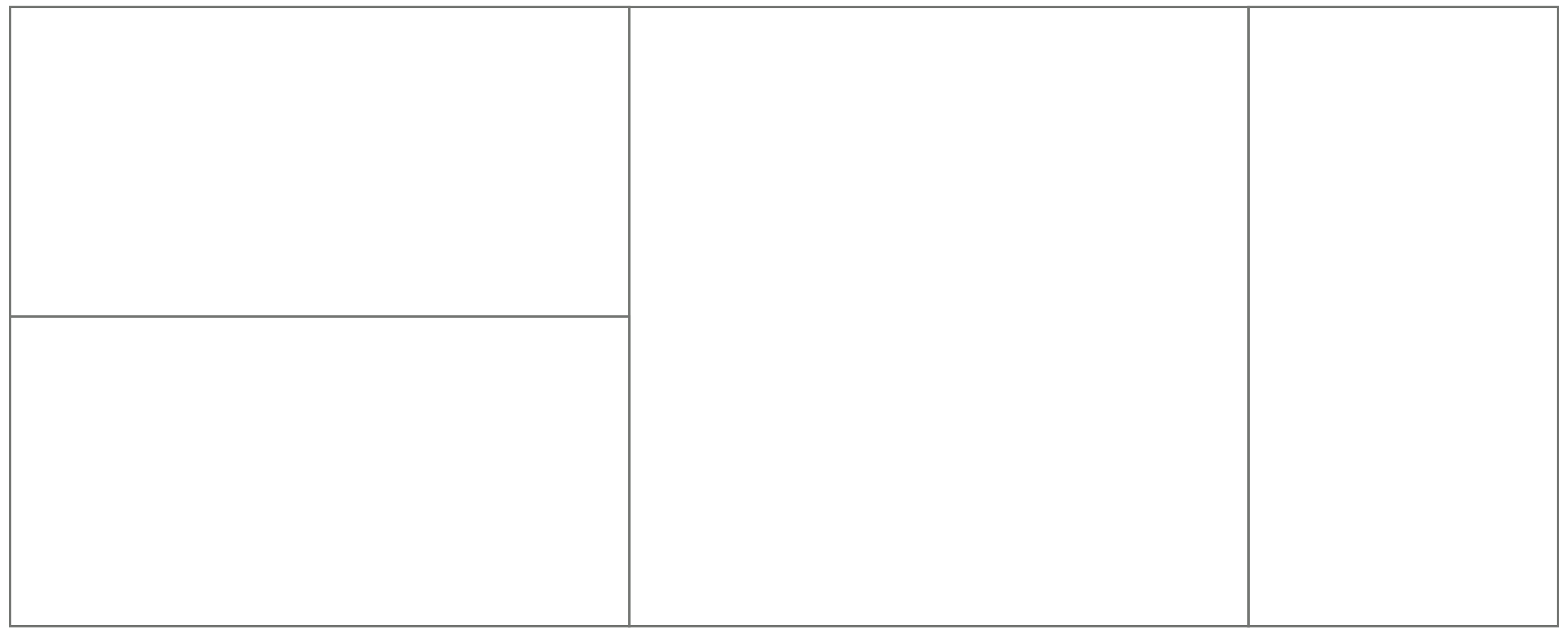
https://www.acmicpc.net/problem/11726

- 소스: http://codeplus.codes/32a53110c01746a393dc60dfc39d5ec4

2×n 타일링 2

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

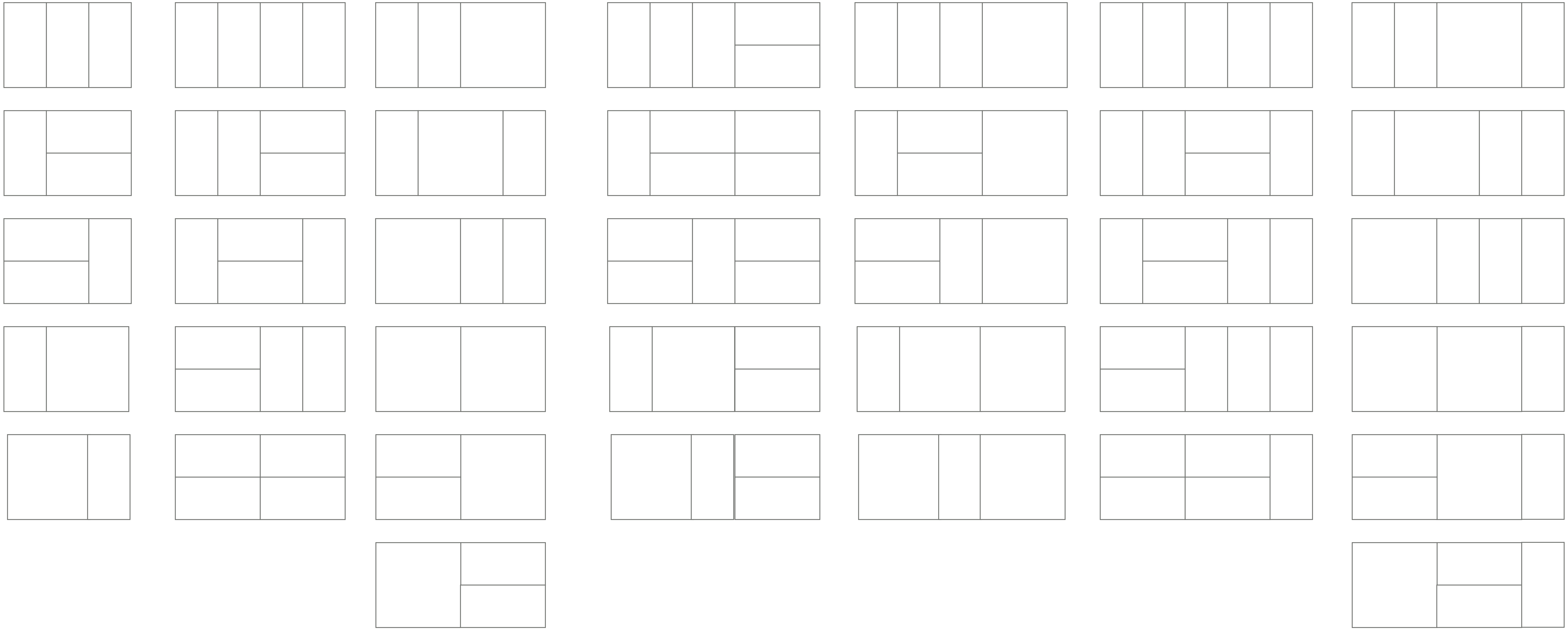
- 2×n 직사각형을 1×2, 2×1, 2×2타일로 채우는 방법의 수
- 아래 그림은 2×5를 채우는 방법의 수
- D[i] = 2×i 직사각형을 채우는 방법의 수



2×n 타일링 2

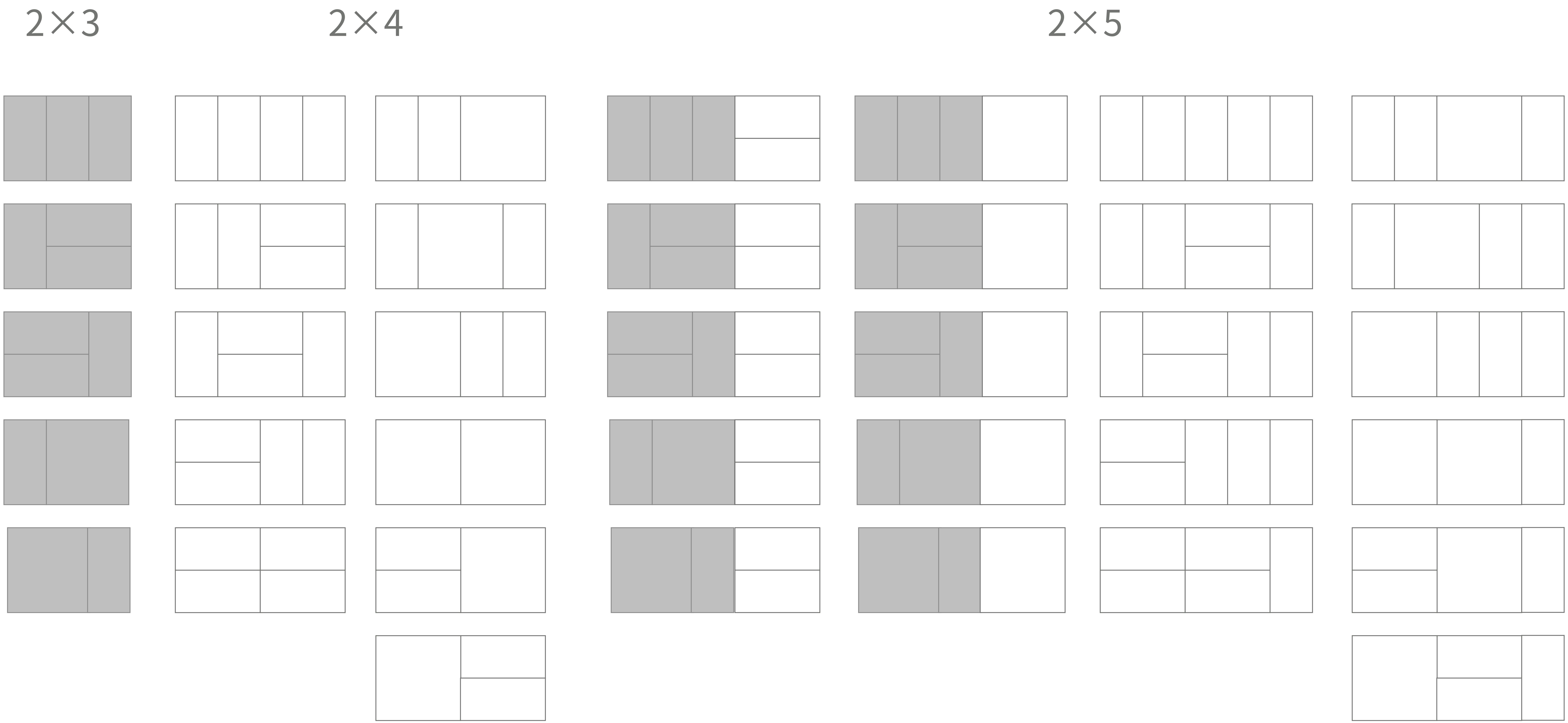
https://www.acmicpc.net/problem/11727

2×3 2×4 2×5



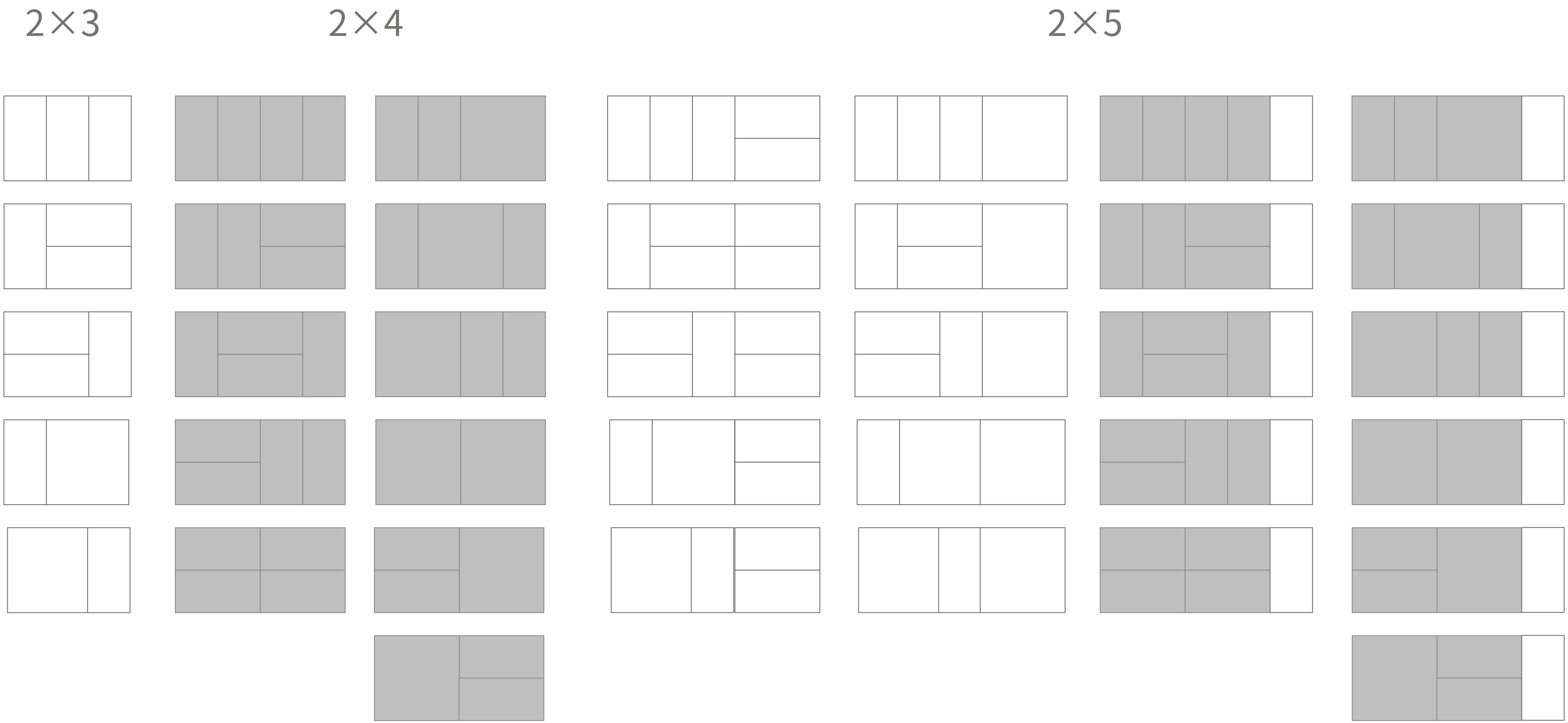
2×n 타일링 2

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



2×n 타일링 2

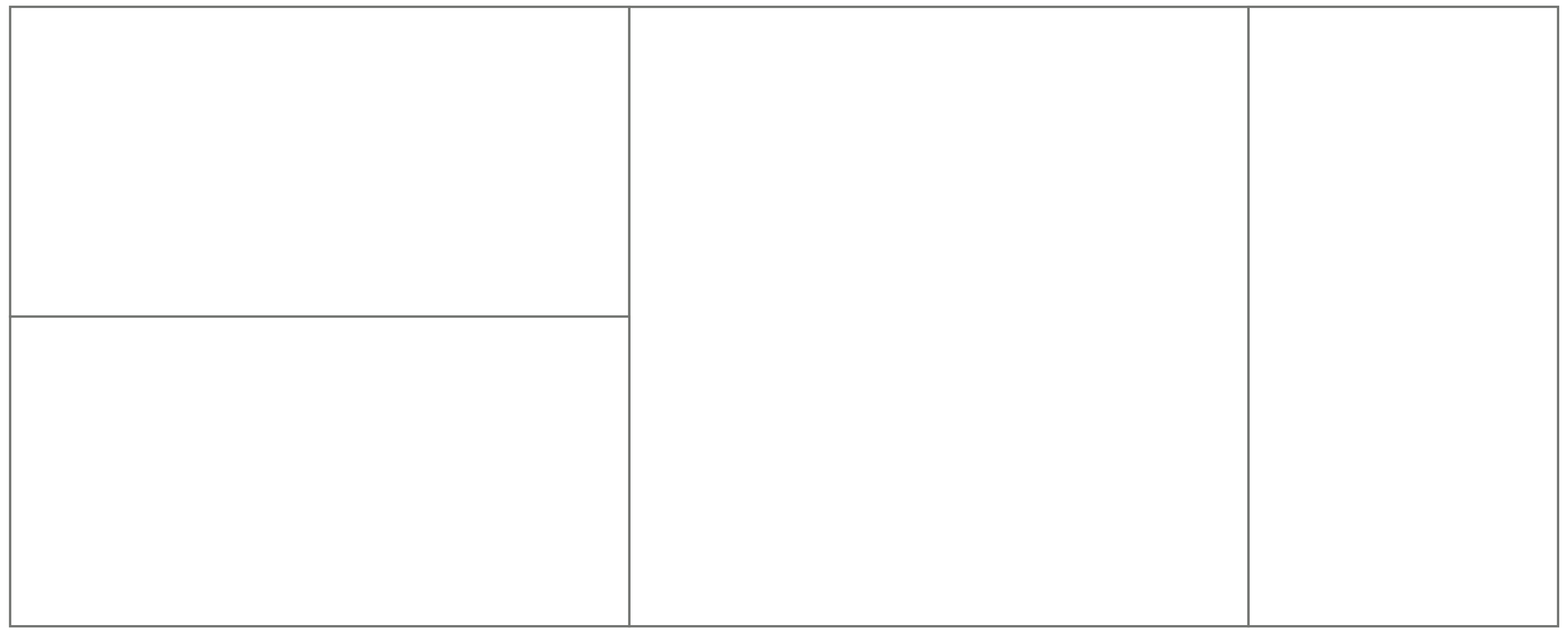
https://www.acmicpc.net/problem/11727



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

- 소스: http://codeplus.codes/b2ad578a5c874f88ac3893cd74e936f1

1, 2, 3 더하기

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

- 정수 n을 1, 2, 3의 합으로 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 합으로 나타내는 방법의 수

1, 2, 3 더하기

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

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

1, 2, 3 더하기

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

0

1

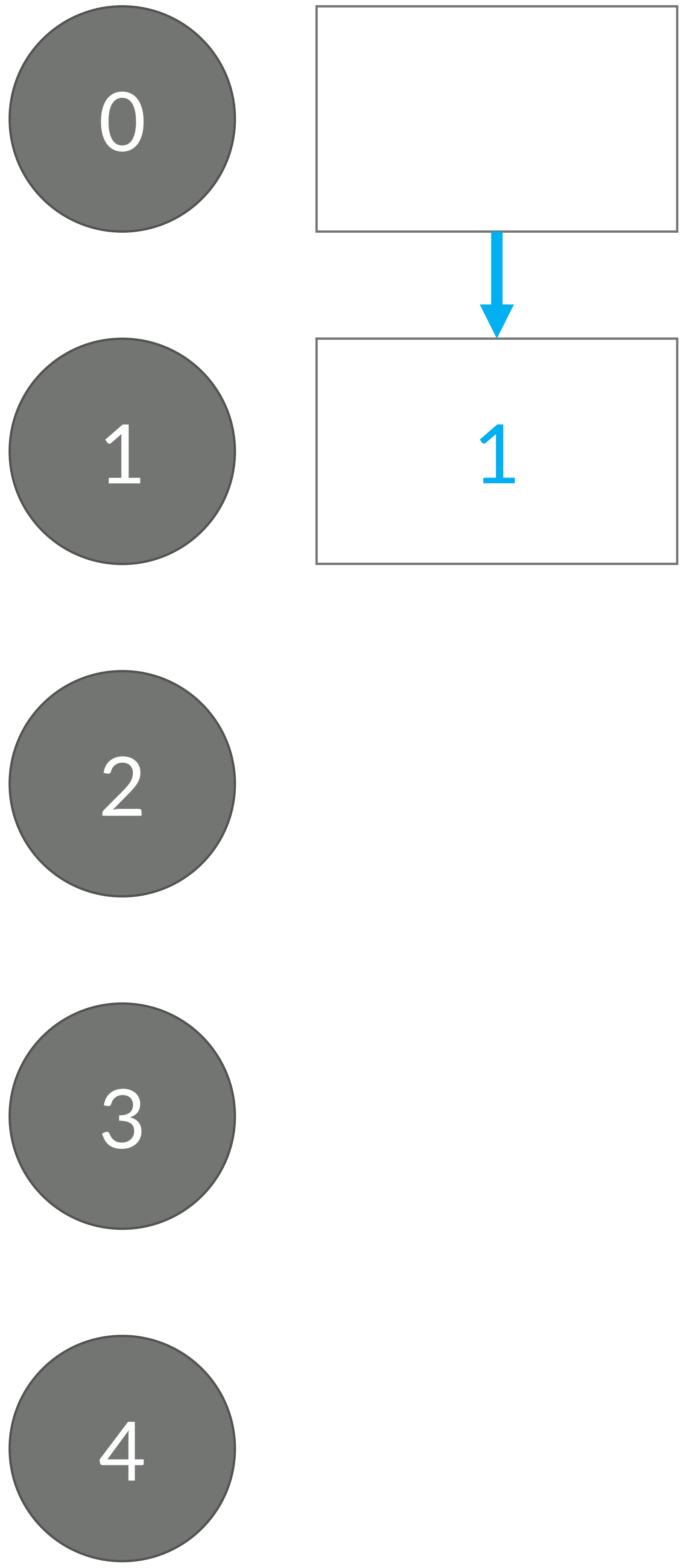
2

3

4

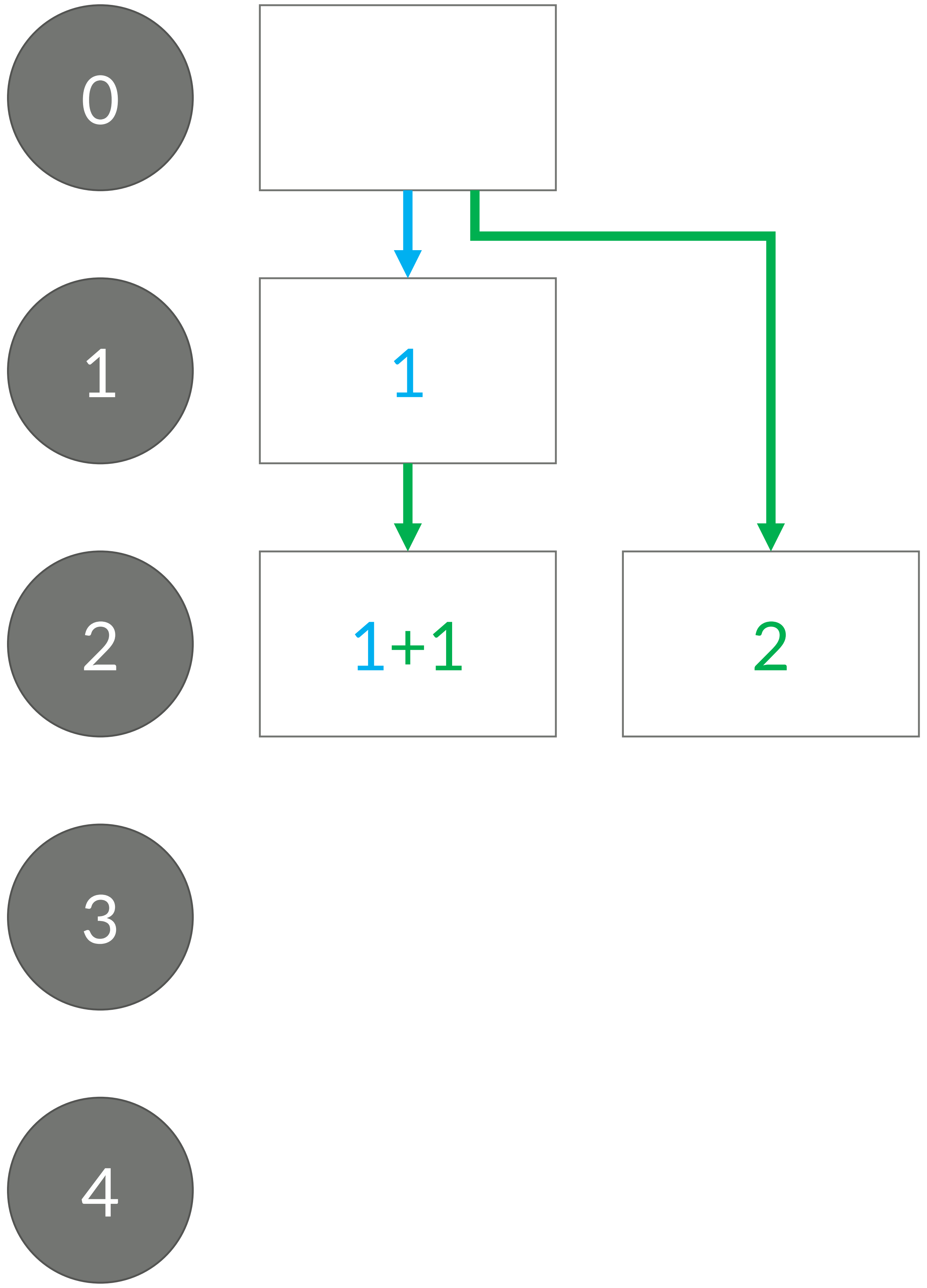
1, 2, 3 더하기

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



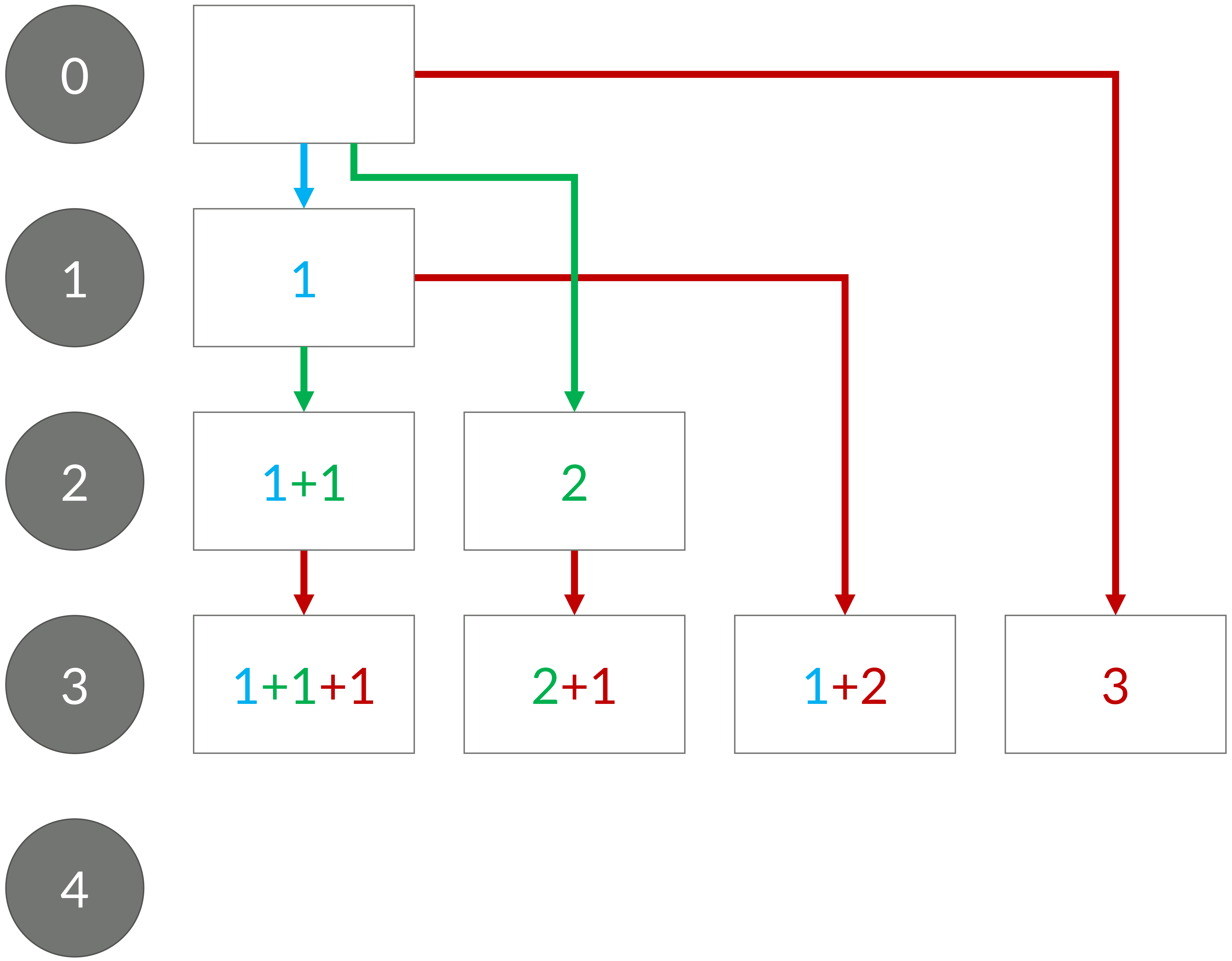
1, 2, 3 더하기

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



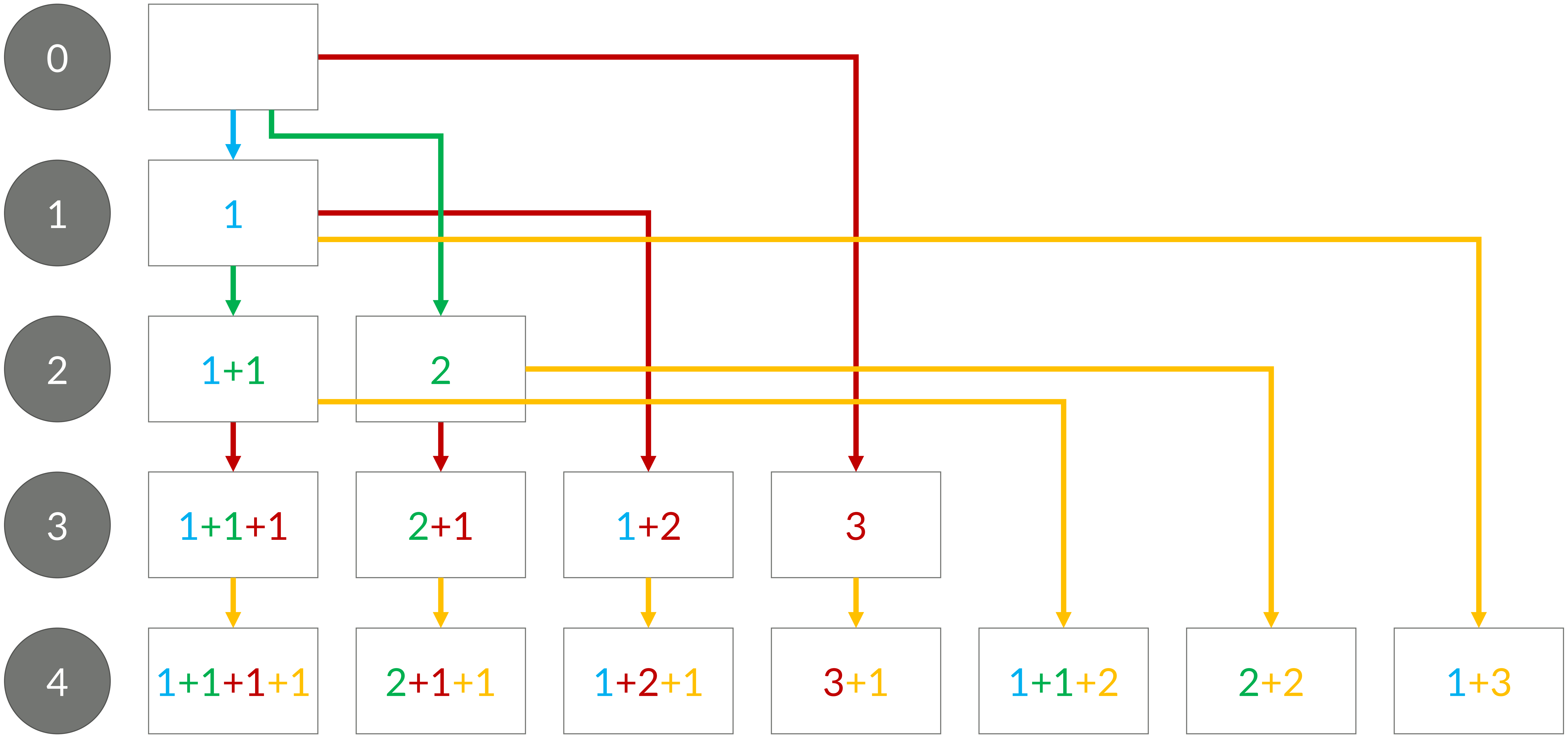
1, 2, 3 더하기

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



1, 2, 3 더하기

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



1, 2, 3 더하기

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

- 소스: http://codeplus.codes/0017890812084c1eb7732c9583a09a77

1, 2, 3 더하기 3

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

- 정수 n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 문제 ($n \le 1,000,000$)
- 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 3

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

- 소스: http://codeplus.codes/03fb341e7f4a493fbbb7f678dedaf665

카드 구매하기

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

- 카드 N개를 구매해야 한다.
- 카드팩은 총 N가지 종류가 존재한다.
- i번째 카드팩은 i개의 카드를 담고 있고, 가격은 P[i]원이다.
- 카드 N개를 구매하는 비용의 최대값을 구하는 문제

카드 구매하기

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

- D[i] = 카드 i개 구매하는 최대 비용
- 카드 i개를 구매하는 방법은?

카드 구매하기

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

- D[i] = 카드 i개 구매하는 최대 비용
- 카드 i개를 구매하는 방법은?
- 카드 1개가 들어있는 카드팩을 구매하고, 카드 i-1개를 구매
- 카드 2개가 들어있는 카드팩을 구매하고, 카드 i-2개를 구매
- …
- 카드 i-1개가 들어있는 카드팩을 구매하고, 카드 1개를 구매
- 카드 i개가 들어있는 카드팩을 구매하고, 카드 0개를 구매

카드 구매하기

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

- D[i] = 카드 i개 구매하는 최대 비용
- 카드 i개를 구매하는 방법은?
- 카드 1개가 들어있는 카드팩을 구매하고, 카드 i-1개를 구매
    - P[1] + D[i-1]
- 카드 2개가 들어있는 카드팩을 구매하고, 카드 i-2개를 구매
    - P[2] + D[i-2]
- …
- 카드 i-1개가 들어있는 카드팩을 구매하고, 카드 1개를 구매
    - P[i-1] + D[1]
- 카드 i개가 들어있는 카드팩을 구매하고, 카드 0개를 구매
    - P[i] + D[0]

카드 구매하기

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

- D[i] = 카드 i개 구매하는 최대 비용
- 카드 i개를 구매하는 방법은?
- 카드 j개가 들어있는 카드팩을 구매하고, 카드 i-j개를 구매
  - $D[i] = \max(P[j] + D[i-j])$ $(1 \le j \le i)$

카드 구매하기

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]);
    }
}
```

카드 구매하기

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

- 소스: http://codeplus.codes/cec76aec1f984fa4927feb941197a8cc

카드 구매하기 2

https://www.acmicpc.net/problem/16194

- 카드 N개를 구매해야 한다.
- 카드팩은 총 N가지 종류가 존재한다.
- i번째 카드팩은 i개의 카드를 담고 있고, 가격은 P[i]원이다.
- 카드 N개를 구매하는 비용의 최솟값을 구하는 문제

카드 구매하기 2

https://www.acmicpc.net/problem/16194

- D[i] = 카드 i개 구매하는 최소 비용
- 카드 i개를 구매하는 방법은?
- 카드 j개가 들어있는 카드팩을 구매하고, 카드 i-j개를 구매
    - D[i] = min(P[j] + D[i-j]) (1 ≤ j ≤ i)

카드 구매하기 2

https://www.acmicpc.net/problem/16194

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

- 이 방법은 배열 d에 항상 0이 들어간다.
- 카드를 구매하는 비용은 0보다 크기 때문에, min의 결과는 항상 0이다.
- 따라서, 배열의 초기값을 잘 설정해야 한다.

카드 구매하기 2

https://www.acmicpc.net/problem/16194

```
for (int i=1; i<=n; i++) {
    d[i] = 1000*10000;
}
d[0] = 0;
for (int i=1; i<=n; i++) {
    for (int j=1; j<=i; j++) {
        d[i] = min(d[i],d[i-j]+a[j]);
    }
}
```

- 카드의 개수 N ≤ 1,000, 카드팩의 가격 ≤ 10,000 이기 때문에
- 정답은 절대로 1000 * 10000을 넘지 않는다.

카드 구매하기 2

https://www.acmicpc.net/problem/16194

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

- d[i] = -1은 아직 정답을 구하지 않았다는 의미이다

카드 구매하기 2

https://www.acmicpc.net/problem/16194

- 소스: http://codeplus.codes/68d1cdd59da1480a9e00e5521ffaf3dd

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- 정수 n을 1, 2, 3의 합으로 나타내는 방법의 수를 구하는 문제
- 단, 같은 수를 두 번 이상 연속해서 사용하면 안된다.
- n = 4
- 1+2+1
- 1+3
- 3+1

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- D[i][j] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 j

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- D[i][j] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 j

- D[i][1] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 1
- D[i][2] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 2
- D[i][3] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 3

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- D[i][j] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 j

- D[i][1] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 1
  - 바로 전에 사용할 수 있는 수는 2, 3
- D[i][2] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 2
  - 바로 전에 사용할 수 있는 수는 1, 3
- D[i][3] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 3
  - 바로 전에 사용할 수 있는 수는 2, 3

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- D[i][j] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 j

- D[i][1] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 1
  - 바로 전에 사용할 수 있는 수는 2, 3
  - D[i][1] = D[i-1][2] + D[i-1][3]
- D[i][2] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 2
  - 바로 전에 사용할 수 있는 수는 1, 3
  - D[i][2] = D[i-1][1] + d[i-1][3]
- D[i][3] = i를 1, 2, 3의 합으로 나타내는 방법의 수, 마지막에 사용한 수는 3
  - 바로 전에 사용할 수 있는 수는 2, 3
  - D[i][3] = D[i-1][2] + D[i-1][3]

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- 1, 2, 3 더하기에서 한 것 처럼 D[0] = 1로 초기화하면 중복이 발생한다.
- D[0][1] = 1, D[0][2] = 1, D[0][3] = 1 로 초기화를 했다면
- D[1][1] = D[0][2] + D[0][3] = 2 (중복이 발생하게 된다)
- 따라서, 이 문제는 예외 처리를 해야 한다.

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- 1, 2, 3 더하기에서 한 것 처럼 D[0] = 1로 초기화하면 중복이 발생한다.
- D[0][1] = 1, D[0][2] = 1, D[0][3] = 1 로 초기화를 했다면
- D[1][1] = D[0][2] + D[0][3] = 2 (중복이 발생하게 된다)
- 따라서, 이 문제는 예외 처리를 해야 한다.
- D[i][1]
    - D[i-1][2] + D[i-1][3] (i > 1)
    - 1 (i == 1)
    - 0 (i < 1)

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- 1, 2, 3 더하기에서 한 것 처럼 D[0] = 1로 초기화하면 중복이 발생한다.
- D[0][1] = 1, D[0][2] = 1, D[0][3] = 1 로 초기화를 했다면
- D[1][1] = D[0][2] + D[0][3] = 2 (중복이 발생하게 된다)
- 따라서, 이 문제는 예외 처리를 해야 한다.
- D[i][2]
  - D[i-1][1] + D[i-1][3] (i > 2)
  - 1 (i == 2)
  - 0 (i < 2)

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- 1, 2, 3 더하기에서 한 것 처럼 D[0] = 1로 초기화하면 중복이 발생한다.
- D[0][1] = 1, D[0][2] = 1, D[0][3] = 1 로 초기화를 했다면
- D[1][1] = D[0][2] + D[0][3] = 2 (중복이 발생하게 된다)
- 따라서, 이 문제는 예외 처리를 해야 한다.
- D[i][3]
    - D[i-1][1] + D[i-1][2] (i > 3)
    - 1 (i == 3)
    - 0 (i < 3)

1, 2, 3 더하기 5

https://www.acmicpc.net/problem/15990

- 소스: http://codeplus.codes/037a62f2456d472a801b5988b16ef41f

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

- 소스: http://codeplus.codes/bbdb8973ee9249c289455f143a66ed09

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

- 소스: http://codeplus.codes/d3a328185fec4f4f9503fb8483f9590d

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

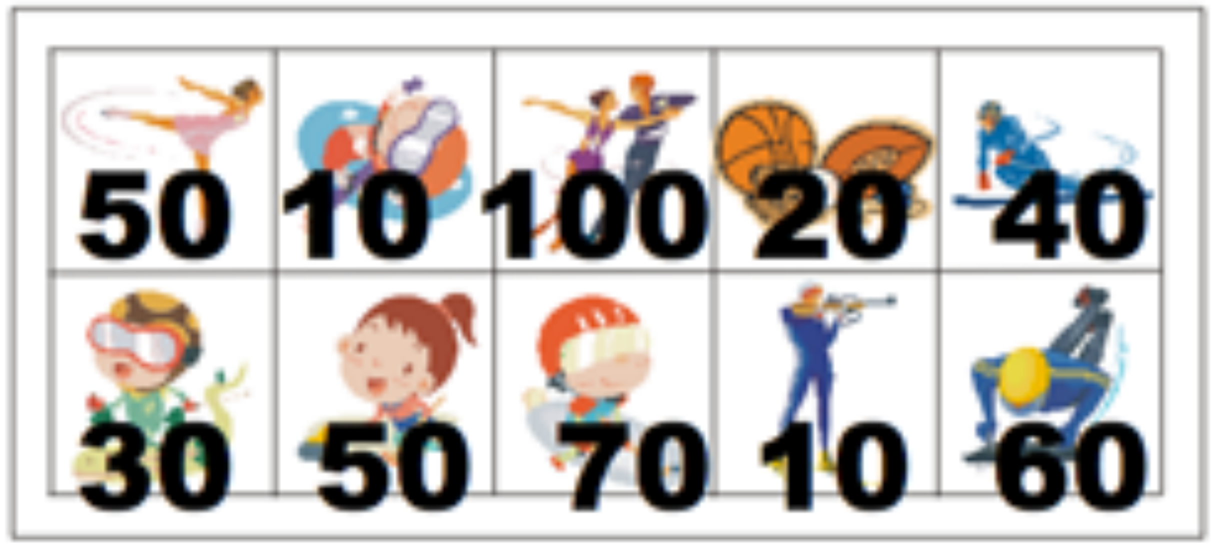
- 소스: http://codeplus.codes/b2f2c44d44f7462586a215815518a50a

스티커

https://www.acmicpc.net/problem/9465

- 스티커 2n개가 2×n 모양으로 배치되어 있다
- 스티커 한 장을 떼면 변을 공유하는 스티커는 모두 찢어져서 사용할 수 없다
- 점수의 합을 최대로 만드는 문제


(a)



(b)

스티커

https://www.acmicpc.net/problem/9465

- D[i][j] = 2×i 에서 얻을 수 있는 최대 점수, i번 열에서 뜯는 스티커는 j
- j = 0 -> 뜯지 않음
- j = 1 -> 위쪽 스티커를 뜯음
- j = 2 -> 아래쪽 스티커를 뜯음

스티커

https://www.acmicpc.net/problem/9465

- D[i][j] = 2×i 에서 얻을 수 있는 최대 점수, i번 열에서 뜯는 스티커는 j
- 뜯지 않음 (D[i][0])
  - i-1 열에서 스티커를 어떻게 뜯었는지 상관이 없다
  - max(D[i-1][0], D[i-1][1], D[i-1][2])
- 위쪽 스티커를 뜯음 (D[i][1])
  - i-1열에서 위쪽 스티커는 뜯으면 안된다
  - max(D[i-1][0], D[i-1][2]) + A[i][0]
- 아래쪽 스티커를 뜯음 (D[i][2])
  - i-1열에서 아래쪽 스티커는 뜯으면 안된다
  - max(D[i-1][0], D[i-1][1]) + A[i][1]

스티커

https://www.acmicpc.net/problem/9465

- 소스: http://codeplus.codes/8e9ca22f2ea64842ac0a2f2a21e86d38

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

- 소스: http://codeplus.codes/6316df75c74a4b6cb6252c93d59a214b

가장 긴 증가하는 부분 수열

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

- 수열 A가 주어졌을 때, 가장 긴 증가하는 부분 수열을 구하는 문제
- 예시
- 수열 A = {10, 20, 10, 30, 20, 50}
- 가장 긴 증가하는 부분 수열 A = {**10**, **20**, 10, **30**, 20, **50**}

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- D[i]은 A[i]이 반드시 포함되어야 한다.
- 가장 긴 부분 수열이 A[?], A[?], …, A[j], A[i] 라고 했을 때, 겹치는 부분 문제를 찾아보자.

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- D[i]은 A[i]이 반드시 포함되어야 한다.
- 가장 긴 부분 수열이 **A[?], A[?], …, A[j],** A[i] 라고 했을 때, 겹치는 부분 문제를 찾아보자.
- A[?], A[?], …, A[j]는 D[j]로 나타낼 수 있다. (A[j]을 마지막으로 하는 부분 수열이기 때문)
- 그럼 A[j]와 A[i]간의 관계를 생각해보자.

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], ⋯, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- D[i]은 A[i]이 반드시 포함되어야 한다.
- 가장 긴 부분 수열이 **A[?], A[?], ⋯, A[j],** A[i] 라고 했을 때, 겹치는 부분 문제를 찾아보자.
- A[?], A[?], ⋯, A[j]는 D[j]로 나타낼 수 있다. (A[j]을 마지막으로 하는 부분 수열이기 때문)
- 그럼 A[j]와 A[i]간의 관계를 생각해보자.
- A[j] < A[i]가 되어야 한다. (증가하는 부분 수열이 되어야 하기 때문)

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], ⋯, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- D[5]를 나타낸 그림

A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

A[5]를 마지막으로 하는 증가하는 부분 수열

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], ···, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

A[1]
10

A[1]	A[2]
10	20

A[1]	A[2]	A[3]
10	20	10

A[1]	A[2]	A[3]	A[4]
10	20	10	30

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], ⋯, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

A[1]
10

D[1] = 1

A[5]
20

10 < 20

A[1]	A[2]
10	20

D[2] = 2

A[5]
20

20 == 20

A[1]	A[2]	A[3]
10	20	10

D[3] = 1

A[5]
20

10 < 20

A[1]	A[2]	A[3]	A[4]
10	20	10	30

D[4] = 3

A[5]
20

30 > 20

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

D[5] = 2

A[1]
10

D[1] = 1

A[5]
20

10 < 20 O

A[1]	A[2]
10	20

D[2] = 2

A[5]
20

20 == 20 X

A[1]	A[2]	A[3]
10	20	10

D[3] = 1

A[5]
20

10 < 20 O

A[1]	A[2]	A[3]	A[4]
10	20	10	30

D[4] = 3

A[5]
20

30 > 20 X

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
10	20	10	30	20	50

A[1]
10

A[1]	A[2]
10	20

A[1]	A[2]	A[3]
10	20	10

A[1]	A[2]	A[3]	A[4]
10	20	10	30

A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
10	20	10	30	20	50

A[1]
10

D[1] = 1

A[1]	A[2]
10	20

D[2] = 2

A[1]	A[2]	A[3]
10	20	10

D[3] = 1

A[1]	A[2]	A[3]	A[4]
10	20	10	30

D[4] = 3

A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

D[5] = 2

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
10	20	10	30	20	50

A[1]
10

D[1] = 1

A[6]
50

10 < 50 ○

A[1]	A[2]
10	20

D[2] = 2

A[6]
50

20 < 50 ○

A[1]	A[2]	A[3]
10	20	10

D[3] = 1

A[6]
50

10 < 50 ○

A[1]	A[2]	A[3]	A[4]
10	20	10	30

D[4] = 3

A[6]
50

30 < 50 ○

A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

D[5] = 2

A[6]
50

20 < 50 ○

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], ⋯, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
10	20	10	30	20	50

D[6] = 4

A[1]
10

D[1] = 1

A[6]
50

10 < 50

A[1]	A[2]
10	20

D[2] = 2

A[6]
50

20 < 50

A[1]	A[2]	A[3]
10	20	10

D[3] = 1

A[6]
50

10 < 50

A[1]	A[2]	A[3]	A[4]
10	20	10	30

D[4] = 3

A[6]
50

30 < 50

A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

D[5] = 2

A[6]
50

20 < 50

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1					

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2				

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1			

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1	3		

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1	3	2	

가장 긴 증가하는 부분 수열

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

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1	3	2	4

가장 긴 증가하는 부분 수열

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

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

가장 긴 증가하는 부분 수열

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

- 정답은 D[1], …, D[N]중의 최대값이 된다.

가장 긴 증가하는 부분 수열

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

- 소스: http://codeplus.codes/4b32e342270c4b4a89a262d12391e970

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- 수열 A가 주어졌을 때, 가장 긴 증가하는 부분 수열을 구하는 문제
- 예시
- 수열 A = {10, 20, 10, 30, 20, 50}
- 가장 긴 증가하는 부분 수열 A = {**10**, **20**, 10, **30**, 20, **50**}

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- V[i] = A[i]의 앞에 와야 하는 수의 인덱스. 즉, A[i]의 앞에는 A[V[i]]가 와야 길이가 가장 길다

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1					
V[i]	0					

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- V[i] = A[i]의 앞에 와야 하는 수의 인덱스. 즉, A[i]의 앞에는 A[V[i]]가 와야 길이가 가장 길다

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2				
V[i]	0	1				

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- V[i] = A[i]의 앞에 와야 하는 수의 인덱스. 즉, A[i]의 앞에는 A[V[i]]가 와야 길이가 가장 길다

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1			
V[i]	0	1	0			

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- D[i] = A[1], ⋯, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- V[i] = A[i]의 앞에 와야 하는 수의 인덱스. 즉, A[i]의 앞에는 A[V[i]]가 와야 길이가 가장 길다

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1	3		
V[i]	0	1	0	2		

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- D[i] = A[1], ⋯, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- V[i] = A[i]의 앞에 와야 하는 수의 인덱스. 즉, A[i]의 앞에는 A[V[i]]가 와야 길이가 가장 길다

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1	3	2	
V[i]	0	1	0	2	3	

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- D[i] = A[1], ⋯, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- V[i] = A[i]의 앞에 와야 하는 수의 인덱스. 즉, A[i]의 앞에는 A[V[i]]가 와야 길이가 가장 길다

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1	3	2	4
V[i]	0	1	0	2	3	4

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- D[i] = A[1], ⋯, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- V[i] = A[i]의 앞에 와야 하는 수의 인덱스. 즉, A[i]의 앞에는 A[V[i]]가 와야 길이가 가장 길다

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1	3	2	4
V[i]	0	1	0	2	3	4

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 긴 증가하는 부분 수열의 길이
- V[i] = A[i]의 앞에 와야 하는 수의 인덱스. 즉, A[i]의 앞에는 A[V[i]]가 와야 길이가 가장 길다

i	1	2	3	4	5	6
A[i]	10	20	10	30	20	50
D[i]	1	2	1	3	2	4
V[i]	0	1	0	2	3	4

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

```
void go(int p) {
    // ? -> ? -> ... a[v[p]] -> a[p]
    // ---------------------
    //         go(v[p]);
    if (p == -1) {
        return ;
    }
    go(v[p]);
    cout << a[p] << ' ';
}
```

가장 긴 증가하는 부분 수열 4

https://www.acmicpc.net/problem/14002

- 소스: http://codeplus.codes/66d156460b644132b8afb647245960cf

가장 큰 증가하는 부분 수열

https://www.acmicpc.net/problem/11055

- 수열 A가 주어졌을 때, 그 수열의 증가 부분 수열 중에서 합이 가장 큰 것을 구하는 문제

가장 큰 증가하는 부분 수열

https://www.acmicpc.net/problem/11055

- D[i] = A[1], …, A[i] 까지 수열이 있을 때, A[i]을 마지막으로 하는 가장 큰 증가하는 부분 수열의 길이

가장 큰 증가하는 부분 수열

https://www.acmicpc.net/problem/11055

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

가장 큰 증가하는 부분 수열

https://www.acmicpc.net/problem/11055

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

가장 큰 증가하는 부분 수열

https://www.acmicpc.net/problem/11055

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

가장 큰 증가하는 부분 수열

https://www.acmicpc.net/problem/11055

- 소스: http://codeplus.codes/01991365d7e94cf3993a42e110154d6c

가장 긴 감소하는 부분 수열

https://www.acmicpc.net/problem/11722

- 수열 A가 주어졌을 때, 그 수열의 감소하는 부분 수열 중에서 가장 긴 것을 구하는 문제
- 두 가지 방법이 있다
- 입력으로 주어진 수열 A를 뒤집어서 가장 긴 증가하는 부분 수열을 구하는 방법
- 가장 긴 증가하는 부분 수열과 비슷하게 구하는 방법 (뒤에서부터 구해야 한다)

가장 긴 감소하는 부분 수열

https://www.acmicpc.net/problem/11722

- D[i] = A[i]에서 시작하는 가장 긴 감소하는 부분 수열의 길이

가장 긴 감소하는 부분 수열

https://www.acmicpc.net/problem/11722

- D[i] = A[i]에서 시작하는 가장 긴 감소하는 부분 수열의 길이
- D[i] = max(D[j]) + 1 (i < j && A[i] > A[j])

가장 긴 감소하는 부분 수열

https://www.acmicpc.net/problem/11722

- 소스: http://codeplus.codes/7620765b6efe4c0aa18383c700308d36

가장 긴 감소하는 부분 수열

https://www.acmicpc.net/problem/11722

- D[i] = A[i]에서 끝나는 가장 긴 감소하는 부분 수열의 길이
- D[i] = max(D[j]) + 1 (j < i && A[j] > A[i])

가장 긴 감소하는 부분 수열

https://www.acmicpc.net/problem/11722

- 소스: http://codeplus.codes/8f17ce1d8ab9427990d2ee4a7aee2dce

가장 긴 바이토닉 부분 수열

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

- 가장 긴 증가하는 부분 수열(D)과 가장 긴 감소하는 부분 수열(D2)를 구한 다음
- D[i] + D2[i] – 1이 가장 큰 값을 찾으면 된다

가장 긴 바이토닉 부분 수열

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

- 소스: http://codeplus.codes/ae99e012416747aab23985009bfbe7ff

연속합

https://www.acmicpc.net/problem/1912

- n개의 정수로 이루어진 임의의 수열이 주어진다.
- 우리는 이 중 연속된 몇 개의 숫자를 선택해서 구할 수 있는 합 중 가장 큰 합을 구하려고 한다.
- 단, 숫자는 한 개 이상 선택해야 한다.
- 예를 들어서 10, -4, 3, 1, 5, 6, -35, 12, 21, -1 이라는 수열이 주어졌다고 하자.
- 여기서 정답은 12+21인 33이 정답이 된다.

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- 이렇게 식을 구했으면, i번째 수에게 가능한 경우를 세야한다

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- 이렇게 식을 구했으면, i번째 수에게 가능한 경우를 세야한다
- i번째 수에게 가능한 경우
  1. i-1번째 수의 연속합에 포함되는 경우
  2. 새로운 연속합을 시작하는 경우

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- 이렇게 식을 구했으면, i번째 수에게 가능한 경우를 세야한다
- i번째 수에게 가능한 경우
  1. i-1번째 수의 연속합에 포함되는 경우
     - D[i-1] + A[i]
  2. 새로운 연속합을 시작하는 경우
     - A[i]
- 두 값 중에 어떤 값이 D[i]에 들어가야 할까? (최대값)
- D[i] = max(D[i-1]+A[i], A[i])

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10									

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = 10 + -4 = 6
- A[i] = -4

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6								

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = 6 + 3 = 9
- A[i] = 3

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6	9							

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = 9 + 1 = 10
- A[i] = 1

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6	9	10						

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = 10 + 5 = 15
- A[i] = 5

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6	9	10	15					

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = 15 + 6 = 21
- A[i] = 6

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6	9	10	15	21				

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = 21 + -35 = -14
- A[i] = -35

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6	9	10	15	21	-14			

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = -14 + 12 = -2
- A[i] = 12

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6	9	10	15	21	-14	12		

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = 12 + 21 = 33
- A[i] = 21

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6	9	10	15	21	-14	12	33	

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합
- D[i-1] + A[i] = 33 + -1 = 32
- A[i] = -1

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	3	1	5	6	-35	12	21	-1
D[i]	10	6	9	10	15	21	-14	12	33	32

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	-10	7	-35	12	21	-10	5	8
D[i]										

연속합

https://www.acmicpc.net/problem/1912

- D[i] = i번째 수로 끝나는 가장 큰 연속합

i	1	2	3	4	5	6	7	8	9	10
A[i]	10	-4	-10	7	-35	12	21	-10	5	8
D[i]	10	6	-4	7	-28	12	34	24	29	37

연속합

https://www.acmicpc.net/problem/1912

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

연속합

https://www.acmicpc.net/problem/1912

- 소스: http://codeplus.codes/05e491a5a27046bdaf66b4fb3b3db646

연속합 2

https://www.acmicpc.net/problem/13398

- dl[i] = 왼쪽에서부터 구한 연속합 정답
- dr[i] = 오른쪽에서부터 구한 연속합 정답

- 각각의 제거할 수 k에 대해서 dl[k-1] + dr[k+1] 의 최대값이 정답이 된다

연속합 2

https://www.acmicpc.net/problem/13398

- 소스: http://codeplus.codes/211c4a9dffdf4972bfee5e656476c682

제곱수의 합

https://www.acmicpc.net/problem/1699

- 주어진 자연수 N을 제곱수들의 합으로 표현할 때에 그 항의 최소개수를 구하는 문제
- $11=3^2+1^2+1^2$

제곱수의 합

https://www.acmicpc.net/problem/1699

- D[i] = i를 제곱수의 합으로 나타냈을 때, 필요한 항의 최소 개수
- i = ? + ? + ⋯ + ? + j
- 마지막 항이 중요하다.
- 마지막 항이 1인 경우
- 마지막 항이 4인 경우
- 마지막 항이 9인 경우
- 마지막 항이 16인 경우
- 마지막 항이 25인 경우
- …

제곱수의 합

https://www.acmicpc.net/problem/1699

- D[i] = i를 제곱수의 합으로 나타냈을 때, 필요한 항의 최소 개수
- i = ? + ? + ⋯ + ? + j
- 마지막 항이 중요하다.
- 마지막 항이 1인 경우 → ? + ? + ⋯ + ? = i-1
- 마지막 항이 4인 경우 → ? + ? + ⋯ + ? = i-4
- 마지막 항이 9인 경우 → ? + ? + ⋯ + ? = i-9
- 마지막 항이 16인 경우 → ? + ? + ⋯ + ? = i-16
- 마지막 항이 25인 경우 → ? + ? + ⋯ + ? = i-25
- ⋯

제곱수의 합

https://www.acmicpc.net/problem/1699

- D[i] = i를 제곱수의 합으로 나타냈을 때, 필요한 항의 최소 개수
- i = ? + ? + ⋯ + ? + j
- 마지막 항이 중요하다.
- 마지막 항이 1인 경우 → ? + ? + ⋯ + ? = i-1 → D[i-1] + 1
- 마지막 항이 4인 경우 → ? + ? + ⋯ + ? = i-4 → D[i-4] + 1
- 마지막 항이 9인 경우 → ? + ? + ⋯ + ? = i-9 → D[i-9] + 1
- 마지막 항이 16인 경우 → ? + ? + ⋯ + ? = i-16 → D[i-16] + 1
- 마지막 항이 25인 경우 → ? + ? + ⋯ + ? = i-25 → D[i-25] + 1
- ⋯

제곱수의 합

https://www.acmicpc.net/problem/1699

- D[i] = i를 제곱수의 합으로 나타냈을 때, 필요한 항의 최소 개수
- D[i] = min(D[i-j^2]+1) (1 ≤ i ≤ j^2)

제곱수의 합

https://www.acmicpc.net/problem/1699

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

제곱수의 합

https://www.acmicpc.net/problem/1699

- 소스: http://codeplus.codes/7c915207ae8d420da2057513e06a9176

합분해

https://www.acmicpc.net/problem/2225

- 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수

합분해

https://www.acmicpc.net/problem/2225

- 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수
- D[K][N] = 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수
- ? + ? + ? + ? + … + ? + L = N
- 위의 식이 나타내는 값: D[K][N]
- ? + ? + ? + ? + … + ? = N-L
- 위의 식이 나타내는 값: D[K-1][N-L]
- D[K][N] = ΣD[K-1][N-L] (0 ≤ L ≤ N)

합분해

https://www.acmicpc.net/problem/2225

```
d[0][0] = 1LL;
for (int i=1; i<=k; i++) {
    for (int j=0; j<=n; j++) {
        for (int l=0; l<=j; l++) {
            d[i][j] += d[i-1][j-l];
            d[i][j] %= mod;
        }
    }
}
```

합분해

https://www.acmicpc.net/problem/2225

- 소스: http://codeplus.codes/8aebbe6b2c284bf882c55404001eee08

합분해

https://www.acmicpc.net/problem/2225

- 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수
- D[K][N] = 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수
- $D[K][N] = \Sigma D[K-1][N-L]$ $(0 \leq L \leq N)$
- $D[K][N] = \Sigma D[K-1][N-L]$ $(0 \leq N-L \leq N)$
- $D[K][N] = \Sigma D[K-1][L]$ $(0 \leq L \leq N)$

합분해

https://www.acmicpc.net/problem/2225

- 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수
- D[K][N] = 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수
- D[K][N] = ΣD[K-1][L] (0 ≤ L ≤ N)
- D[K][N] = D[K-1][0] + D[K-1][1] + ... + D[K-1][N-1] + D[K-1][N]
- D[K][N-1] = D[K-1][0] + D[K-1][1] + ... + D[K-1][N-1]

합분해

https://www.acmicpc.net/problem/2225

- 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수
- D[K][N] = 0부터 N까지의 정수 K개를 더해서 그 합이 N이 되는 경우의 수
- D[K][N] = ΣD[K-1][L] (0 ≤ L ≤ N)
- D[K][N] = **D[K-1][0] + D[K-1][1] + … + D[K-1][N-1]** + D[K-1][N]
- D[K][N-1] = **D[K-1][0] + D[K-1][1] + … + D[K-1][N-1]**
- D[K][N] = D[K][N-1] + D[K-1][N]

합분해

https://www.acmicpc.net/problem/2225

- 소스: http://codeplus.codes/9887579548f34864a3c1e5148ae36a46

끝

코드 플러스

https://code.plus

- 슬라이드에 포함된 소스 코드를 보려면 "정보 수정 > 백준 온라인 저지 연동"을 통해 연동한 다음, "백준 온라인 저지"에 로그인해야 합니다.
- 강의 내용에 대한 질문은 코드 플러스의 "질문 게시판"에서 할 수 있습니다.
- 문제와 소스 코드는 슬라이드에 첨부된 링크를 통해서 볼 수 있으며, "백준 온라인 저지"에서 서비스됩니다.
- 슬라이드와 동영상 강의는 코드 플러스 사이트를 통해서만 볼 수 있으며, 동영상 강의의 녹화와 다운로드, 배포와 유통은 저작권법에 의해서 금지되어 있습니다.
- 다른 경로로 이 슬라이드나 동영상 강의를 본 경우에는 codeplus@startlink.io 로 이메일 보내주세요.

- 강의 내용, 동영상 강의, 슬라이드, 첨부되어 있는 소스 코드의 저작권은 스타트링크와 최백준에게 있습니다.