

다이나믹 프로그래밍 1

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

DC
DC
DP

DP

.

가

.

.

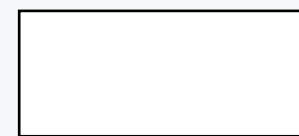
다이나믹 프로그래밍

Dynamic Programming

4

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

1. Overlapping Subproblem



2. Optimal Substructure



Overlapping Subproblem

5

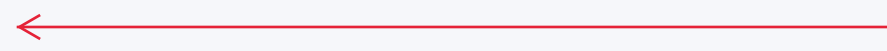
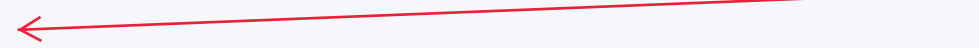
Overlapping Subproblem

- 피보나치 수

- $F_0 = 0$

- $F_1 = 1$

- $F_n = F_{n-1} + F_{n-2} \ (n \geq 2)$



Overlapping Subproblem

Overlapping Subproblem

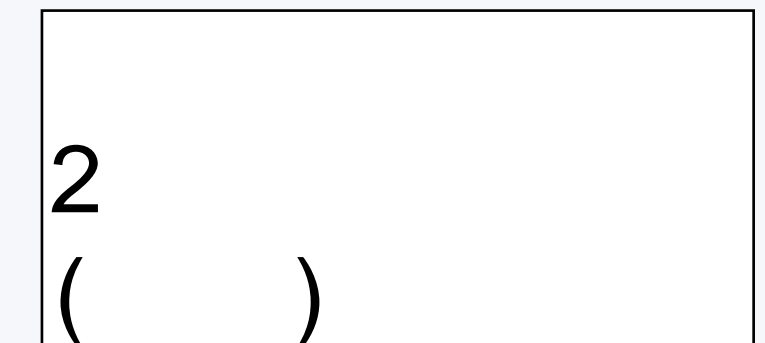
- 피보나치 수
- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2} \ (n \geq 2)$
- 문제: N번째 피보나치 수를 구하는 문제
- 작은 문제: N-1번째 피보나치 수를 구하는 문제, N-2번째 피보나치 수를 구하는 문제

Overlapping Subproblem

7

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

10

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

Optimal Substructure

12

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

$$\left(\begin{array}{c} \text{ } \end{array} \right) : O\left(\frac{2^N}{2}\right)$$

14

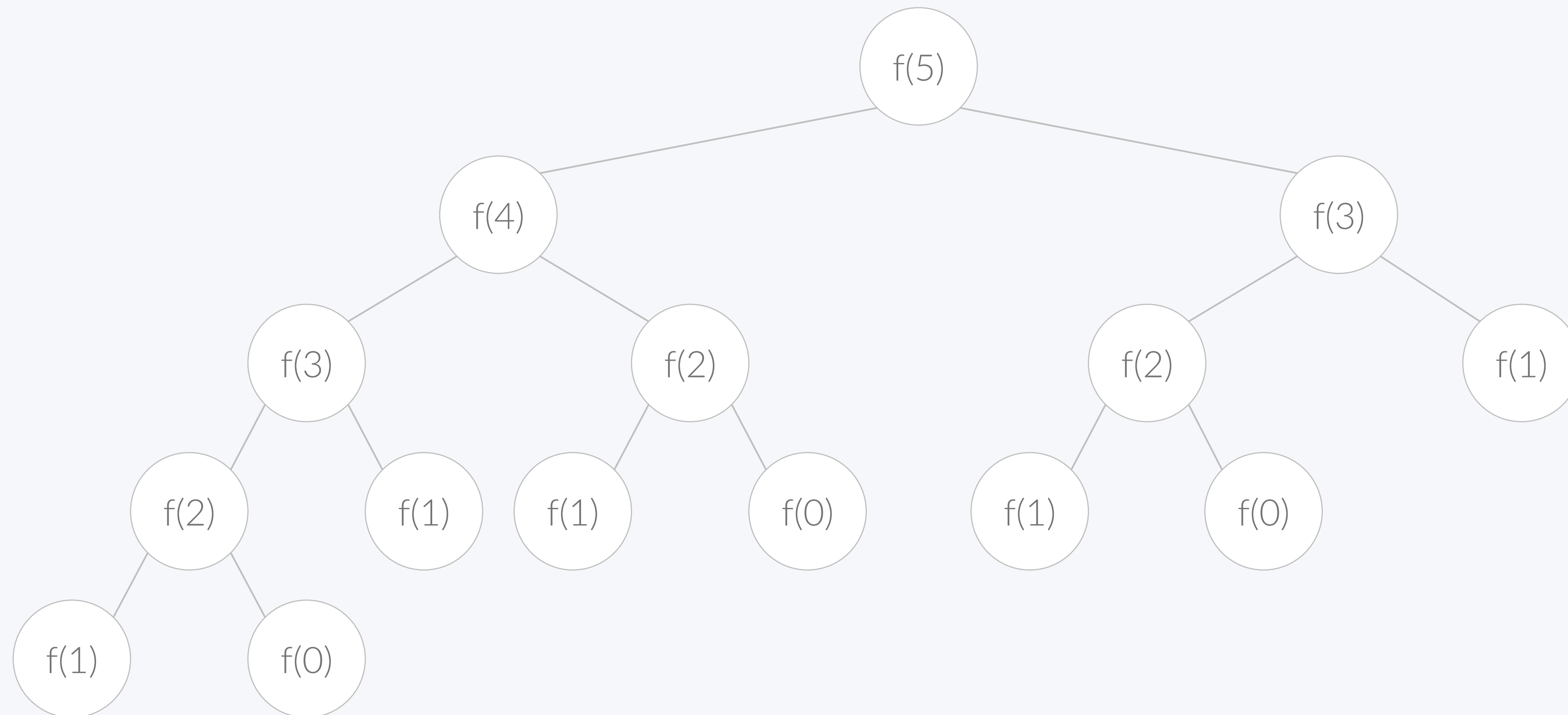
```
int fibonacci(int n) {  
    if (n <= 1) {  
        return n;  
    } else {  
        return fibonacci(n-1) + fibonacci(n-2);  
    }  
}
```

- 피보나치 수를 구하는 함수이다.

피보나치 수

Dynamic Programming

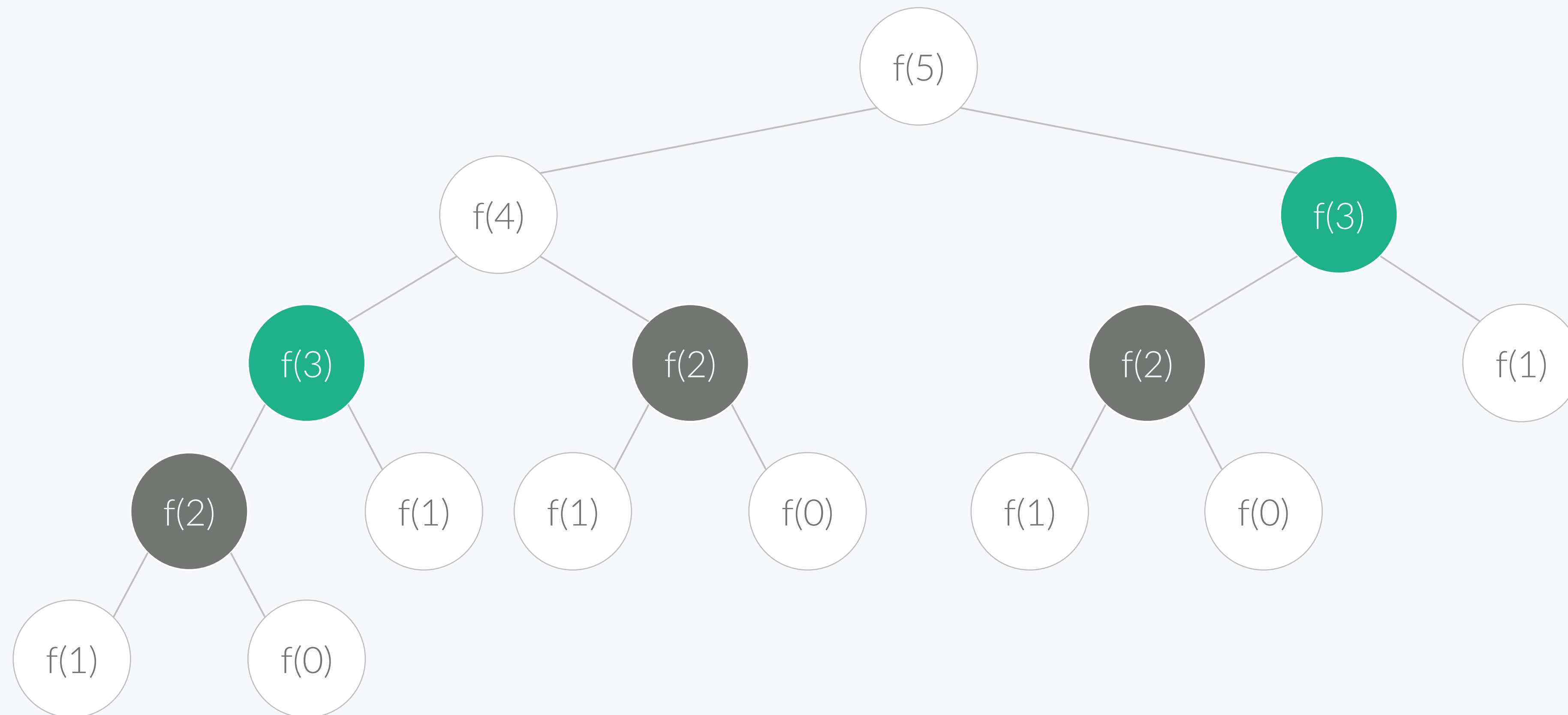
- fibonacci(5)를 호출한 경우 함수가 어떻게 호출되는지를 나타낸 그림



피보나치 수

Dynamic Programming

- 아래 그림과 같이 겹치는 호출이 생긴다.

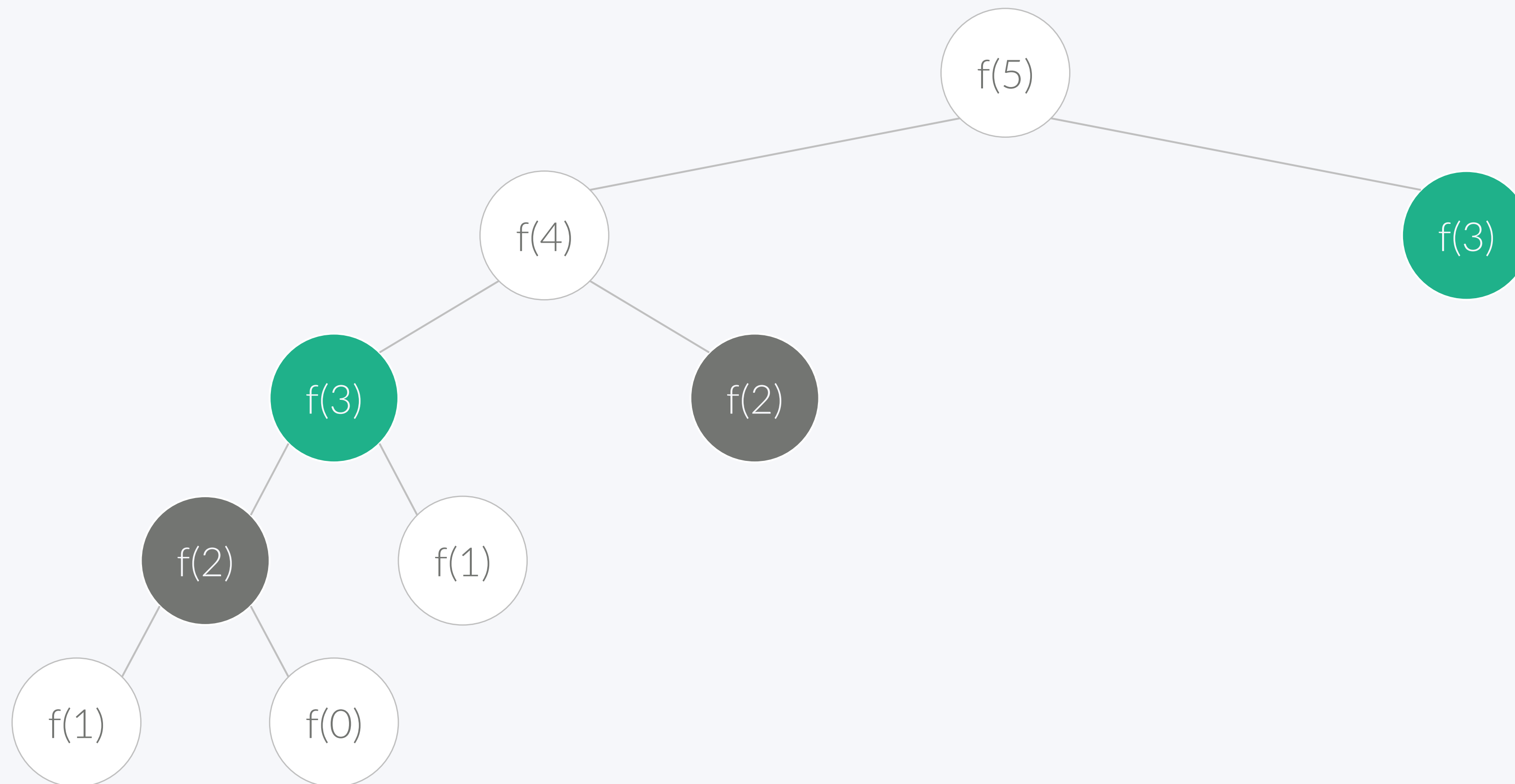


피보나치 수

Dynamic Programming

17

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



피보나치 수

Dynamic Programming

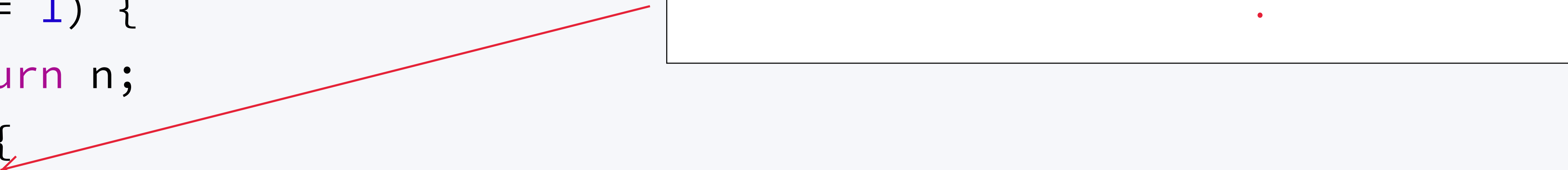
```
int memo[100];
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        memo[n] = fibonacci(n-1) + fibonacci(n-2);
        return memo[n];
    }
}
```

피보나치 수

19

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];
    }
}
```

Why? : $O(N)$
1
가
가
DP
X

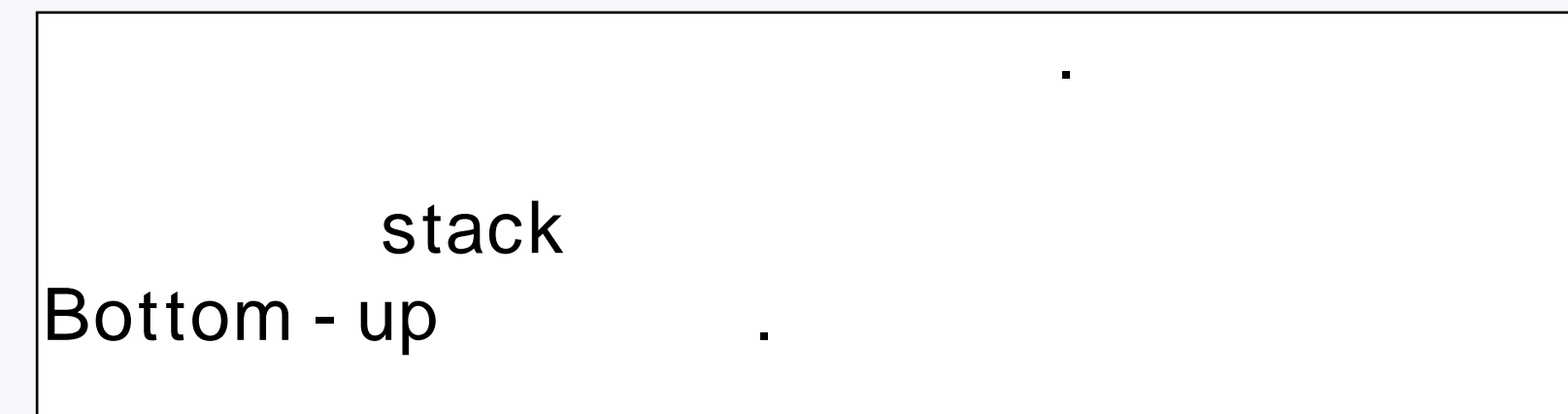
가
> 0

다이나믹 프로그래밍

21

Dynamic Programming

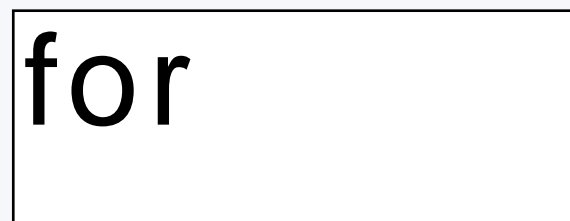
- 다이나믹을 푸는 두 가지 방법이 있다.



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 \longrightarrow $D[N] = N$

- 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

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

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

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

N	가	가?
---	---	----

1로 만들기

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

- $D[i] = i$ 를 1로 만드는데 필요한 최소 연산 횟수
- i 에게 가능한 경우를 생각해보자
 1. i 가 3으로 나누어 떨어졌을 때, 3으로 나누는 경우
 2. i 가 2로 나누어 떨어졌을 때, 2로 나누는 경우
 3. i 에서 1을 빼는 경우

1로 만들기

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

- $D[i]$ = i 를 1로 만드는데 필요한 최소 연산 횟수
- i 에게 가능한 경우를 생각해보자
 1. i 가 3으로 나누어 떨어졌을 때, 3으로 나누는 경우
 - $D[i/3] + 1$
 2. i 가 2로 나누어 떨어졌을 때, 2로 나누는 경우
 - $D[i/2] + 1$
 3. i 에서 1을 빼는 경우
 - $D[i-1] + 1$

1로 만들기

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

- $D[i]$ = i 를 1로 만드는데 필요한 최소 연산 횟수
- i 에게 가능한 경우를 생각해보자
 1. i 가 3으로 나누어 떨어졌을 때, 3으로 나누는 경우
 - $D[i/3] + 1$
 2. i 가 2로 나누어 떨어졌을 때, 2로 나누는 경우
 - $D[i/2] + 1$
 3. i 에서 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>

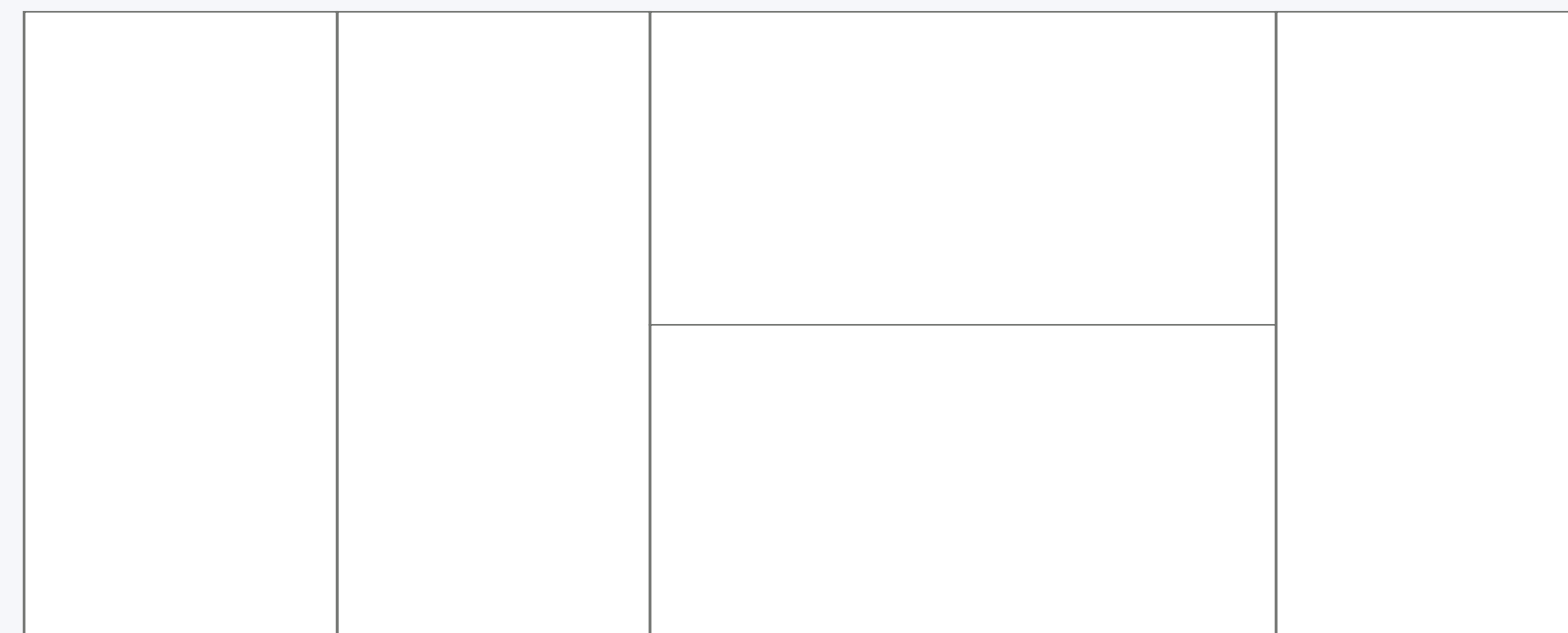
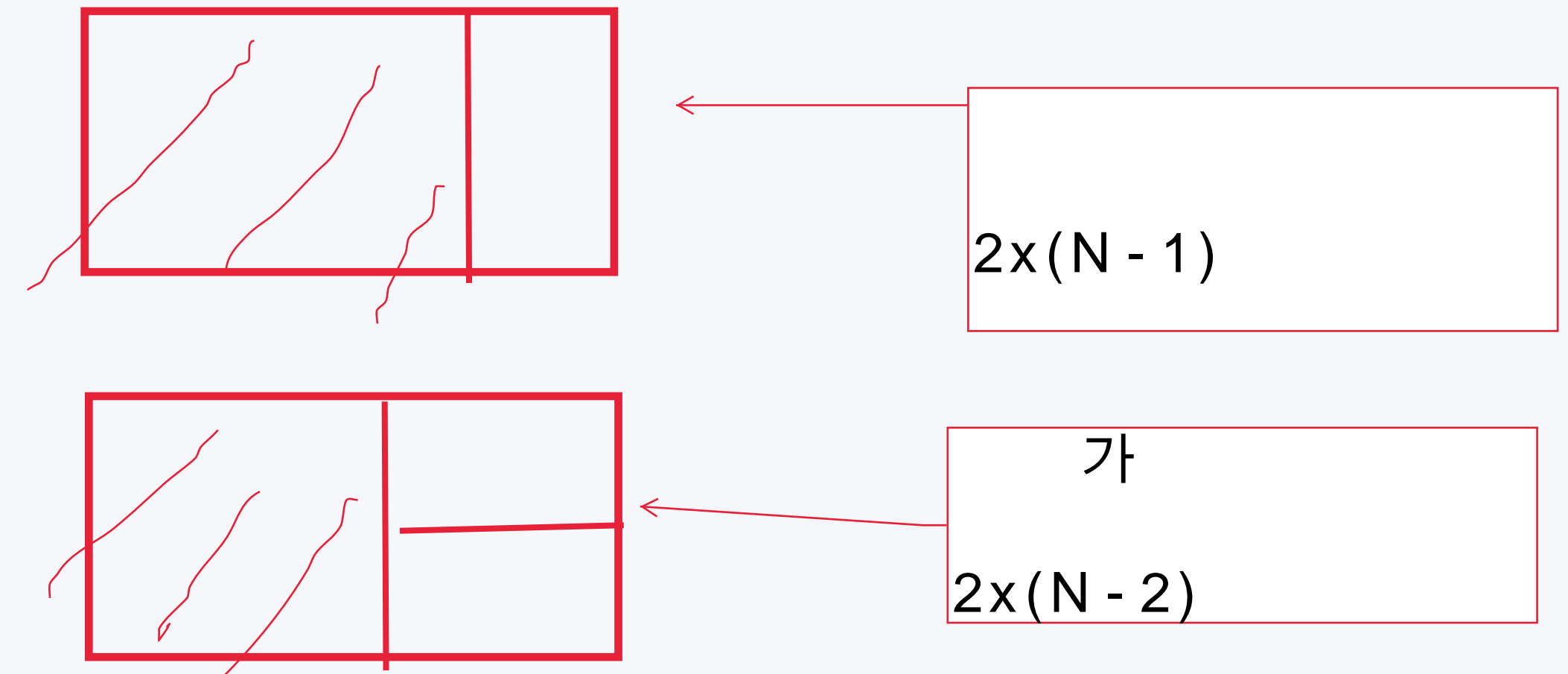
- Bottom-up 방식
- C: <https://gist.github.com/Baekjoon/30f4bb39cdc66f7f16c1>
- C++: <https://gist.github.com/Baekjoon/31e553ab3b371fe06384>
- Java: <https://gist.github.com/Baekjoon/0813d3bc5db11b9bb72d>

2×n 타일링

-> 1 i 가 .

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

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

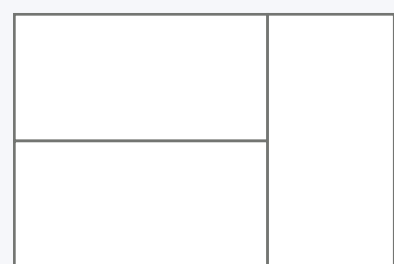
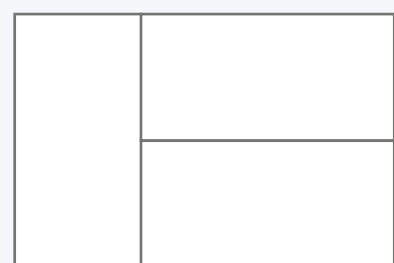
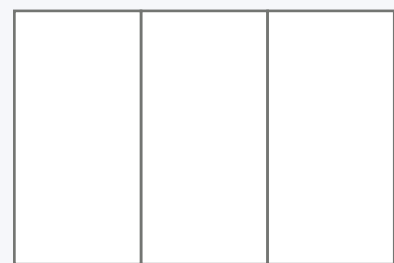


2×n 타일링

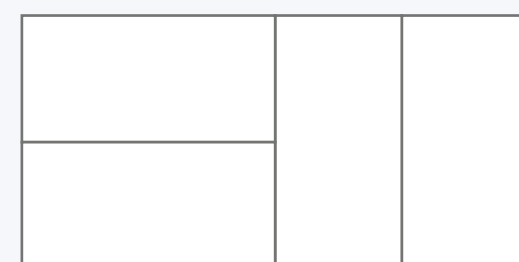
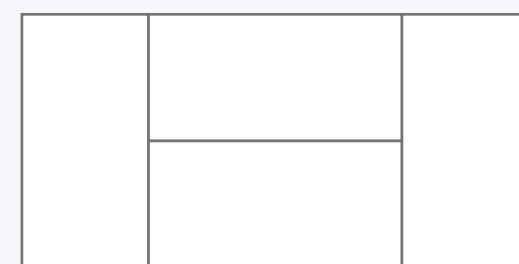
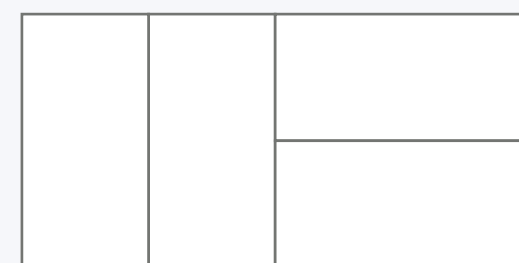
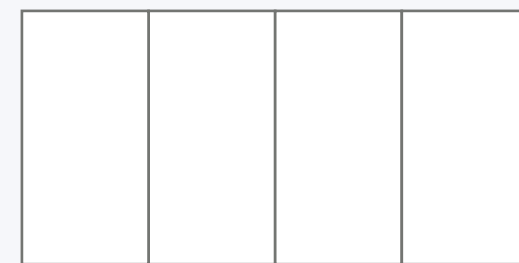
<https://www.acmicpc.net/problem/11726>

41

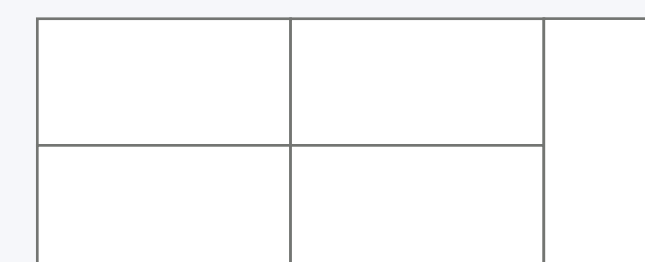
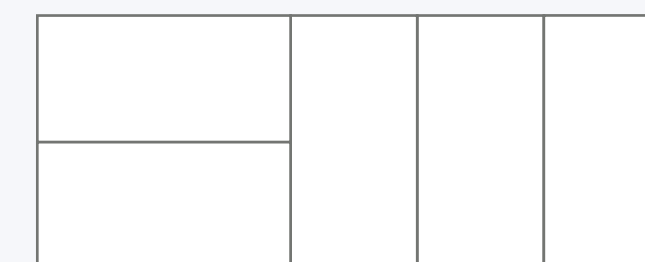
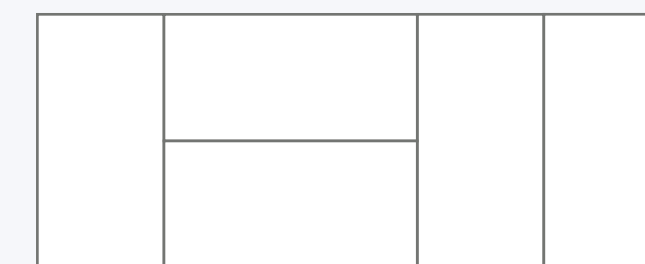
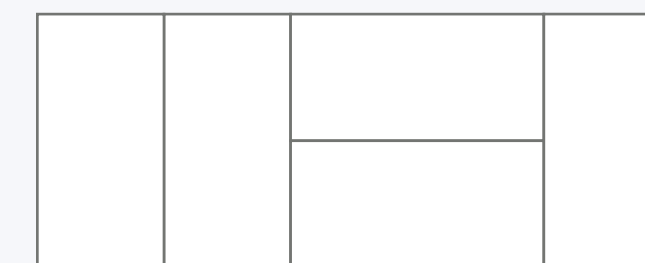
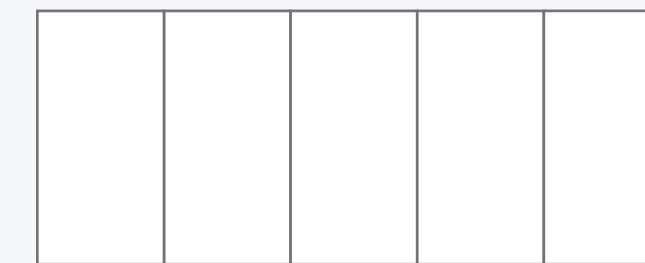
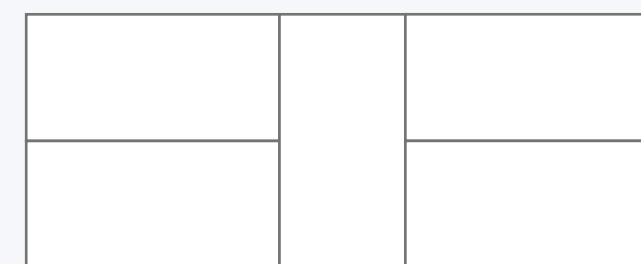
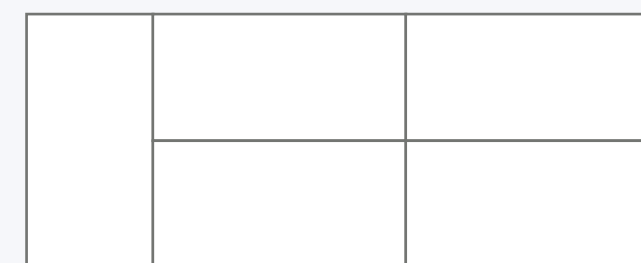
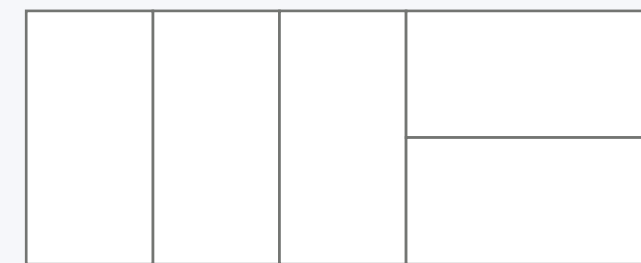
2×3



2×4



2×5

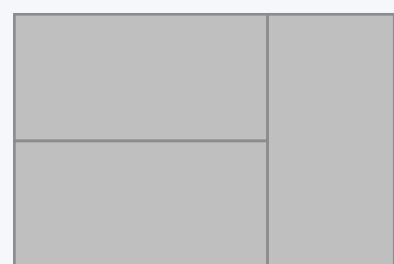
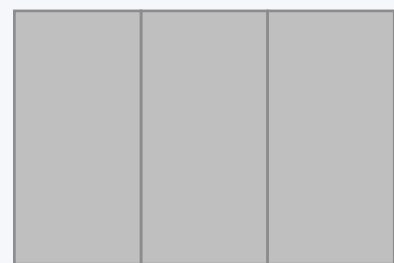


2×n 타일링

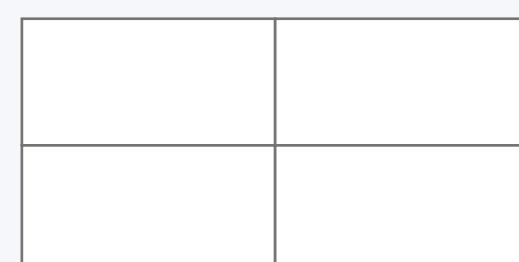
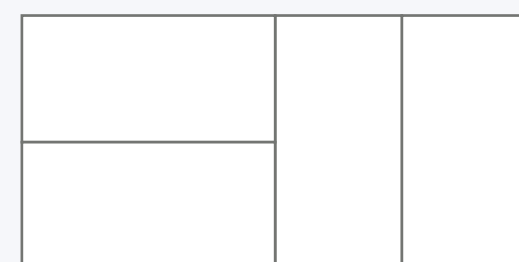
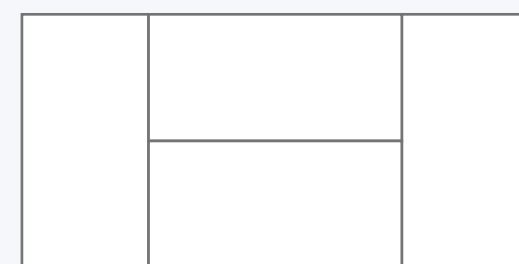
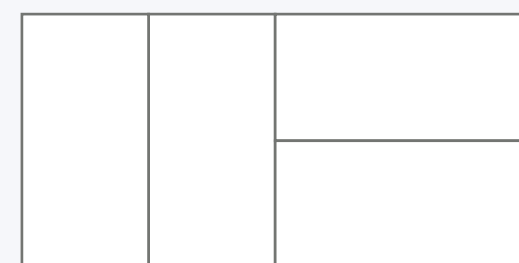
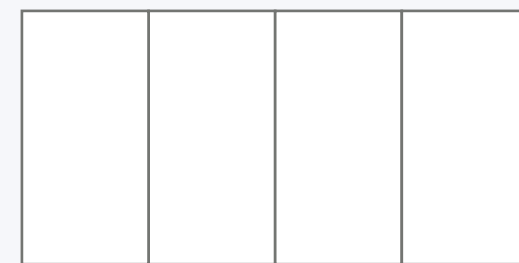
<https://www.acmicpc.net/problem/11726>

42

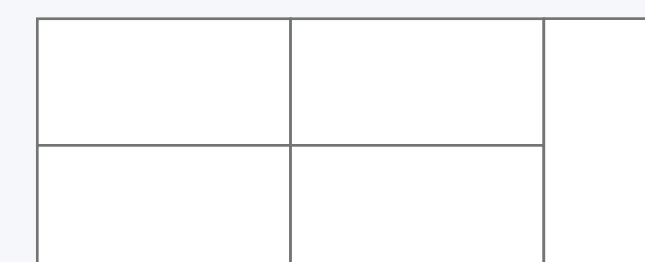
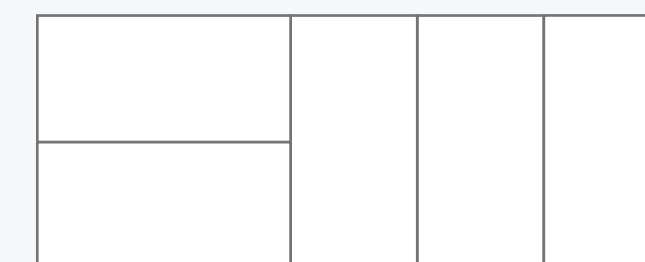
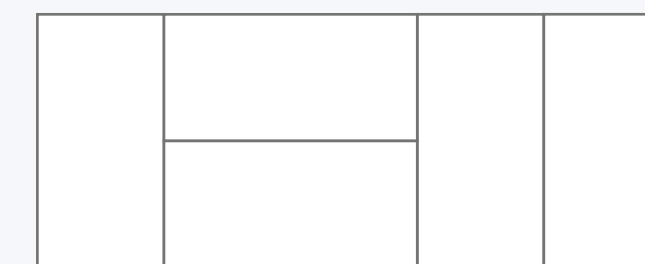
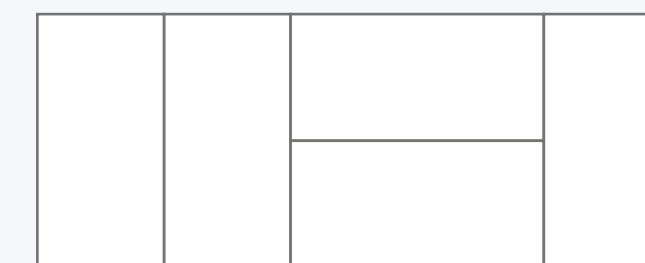
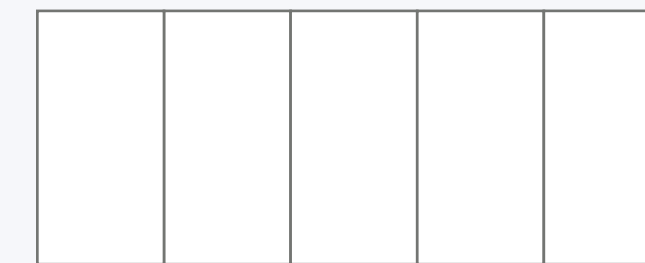
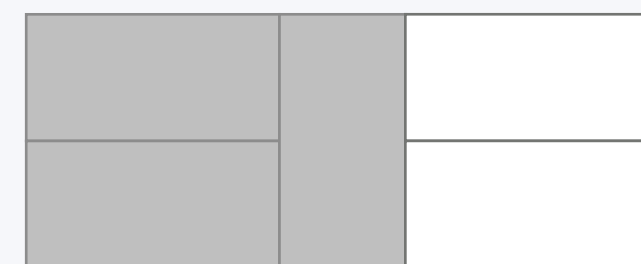
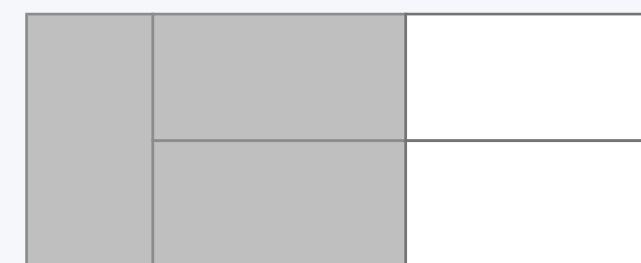
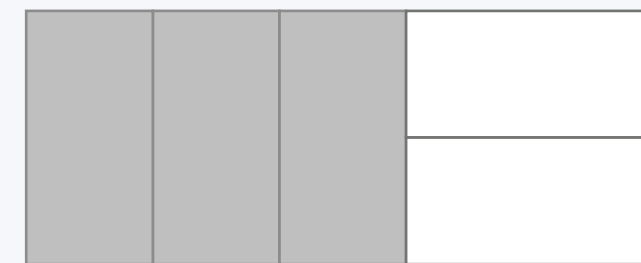
2×3



2×4



2×5

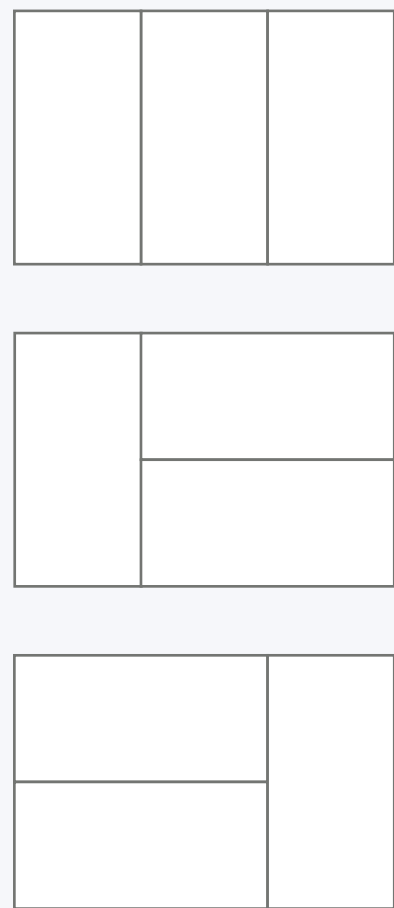


2×n 타일링

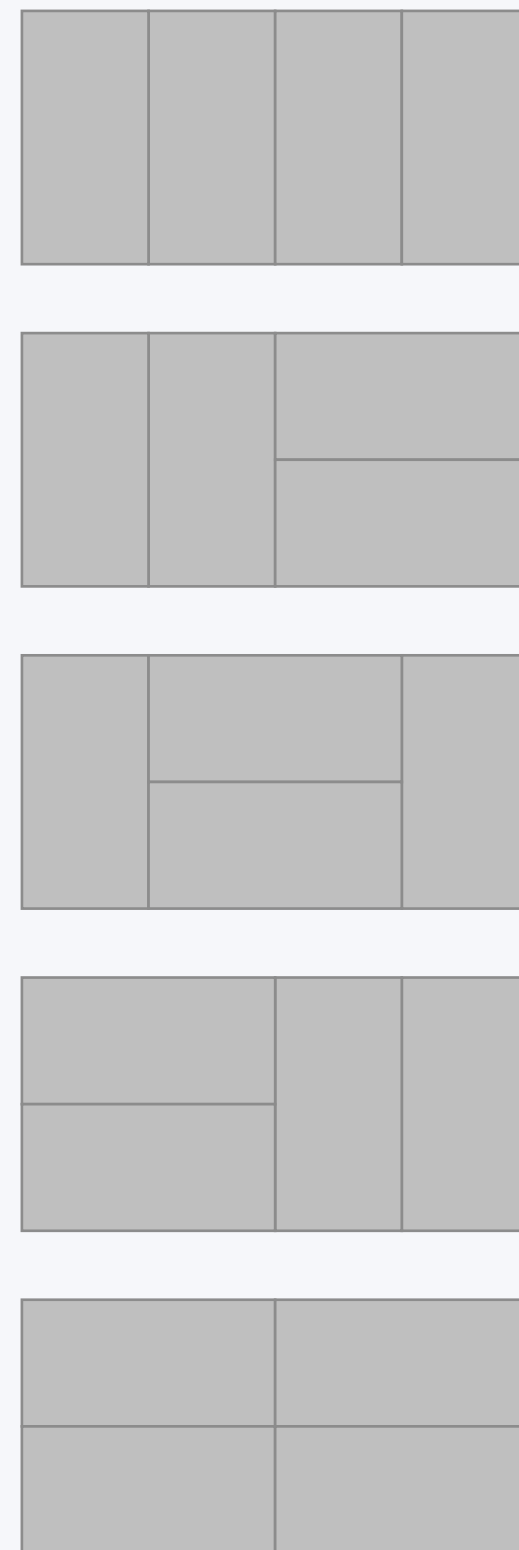
<https://www.acmicpc.net/problem/11726>

43

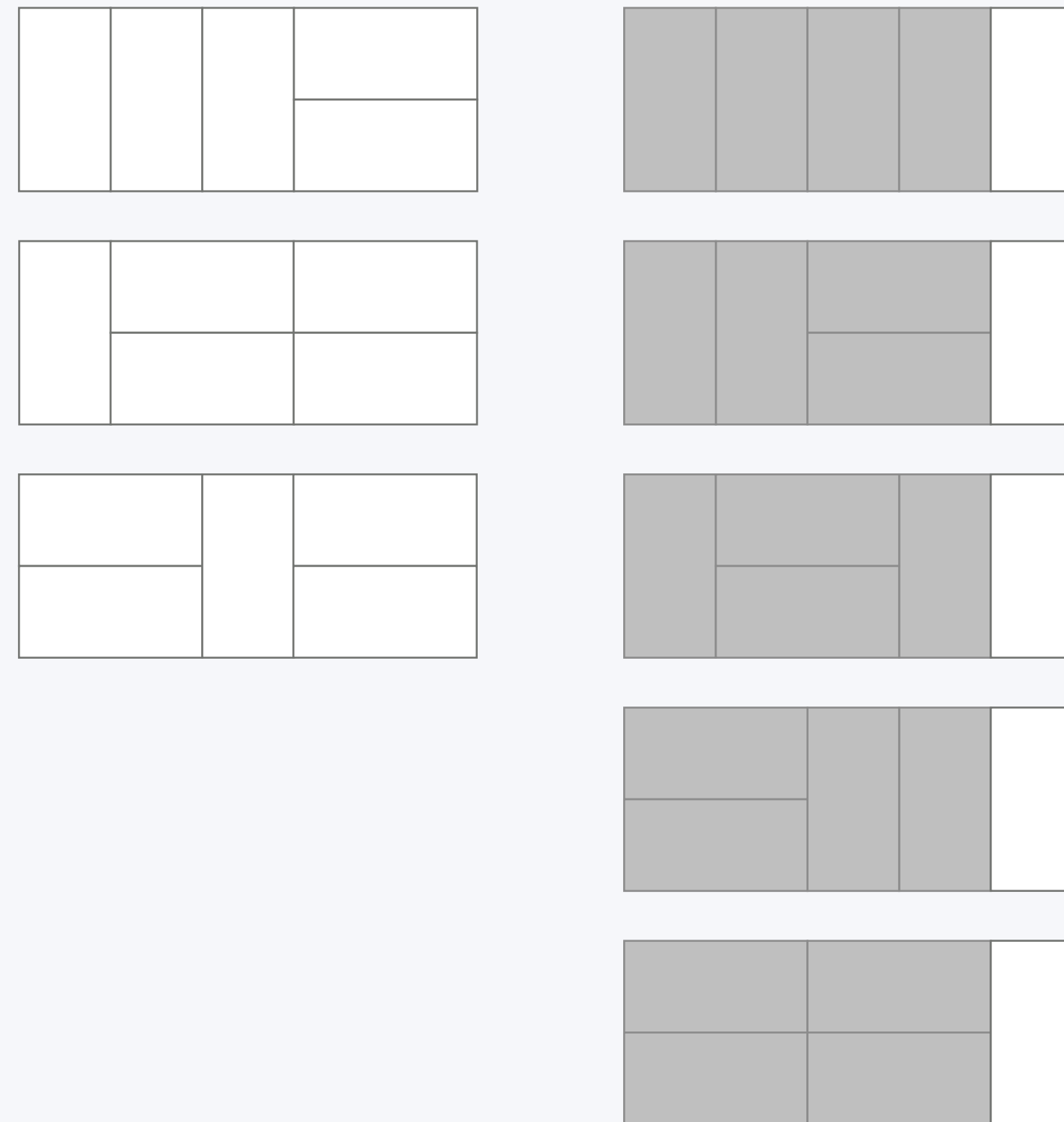
2×3



2×4



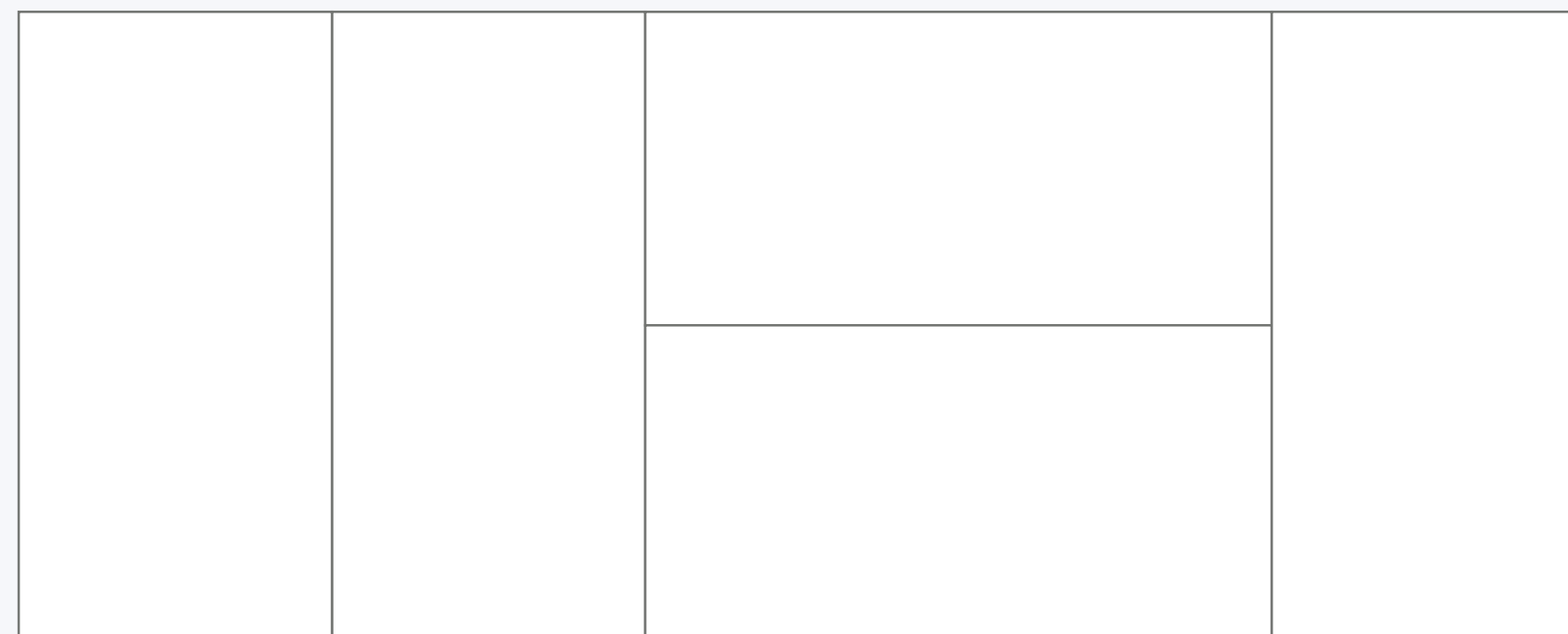
2×5



2×n 타일링

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

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



2 × n 타일링

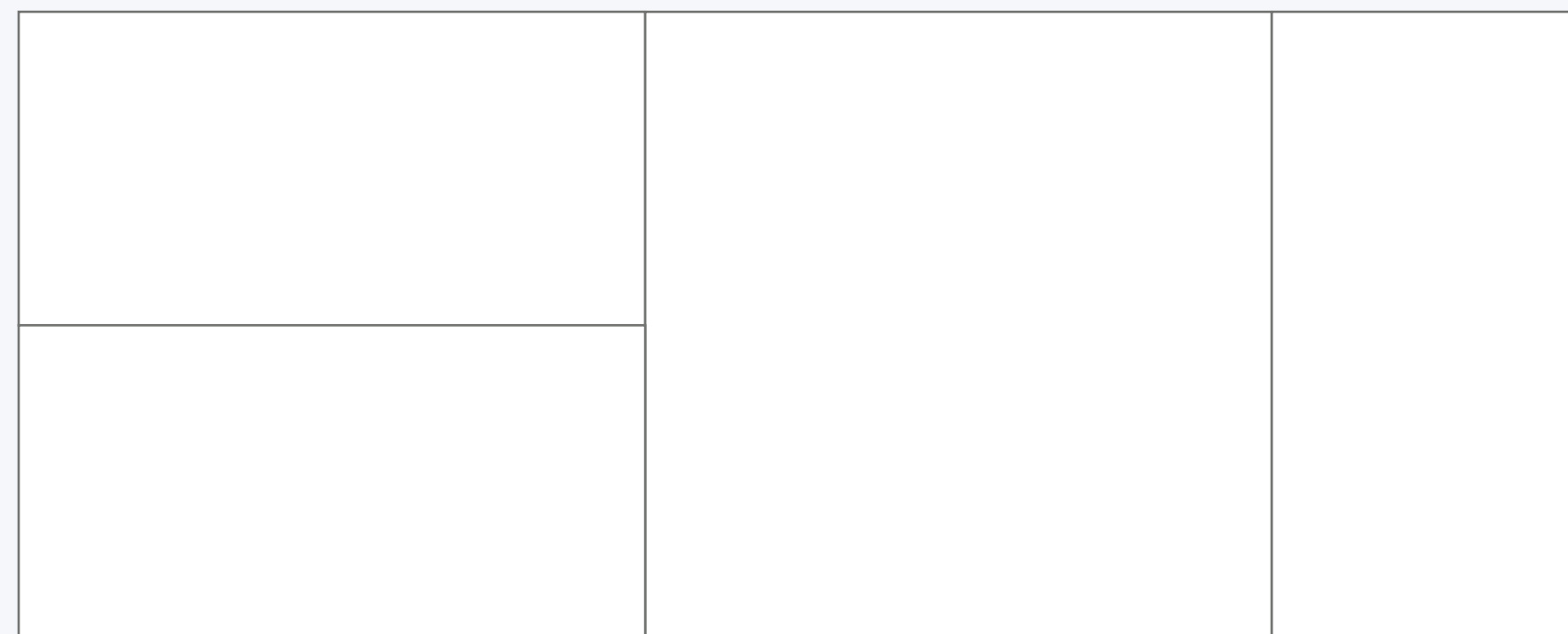
<https://www.acmicpc.net/problem/11726>

- C/C++: <https://gist.github.com/Baekjoon/3527f6fd4771f8c3e1>
- Java: <https://gist.github.com/Baekjoon/53f6e5ec06bfbafad977150df382cf55>

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
- $D[i] = 2 \times i$ 직사각형을 채우는 방법의 수



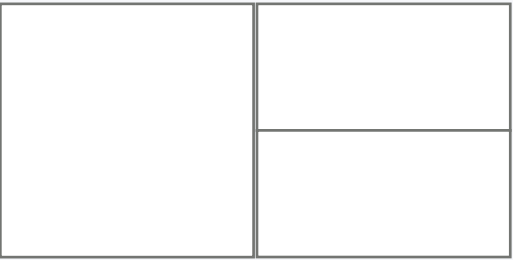
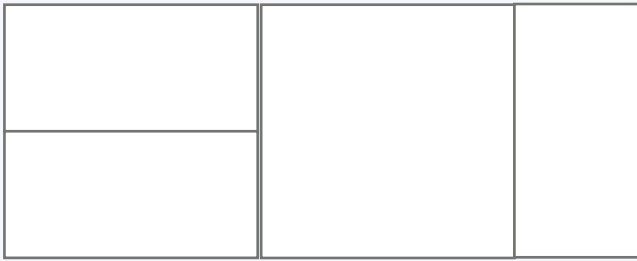
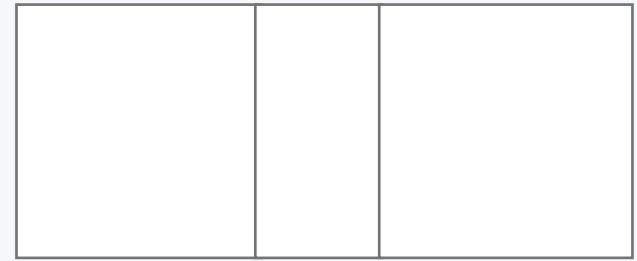
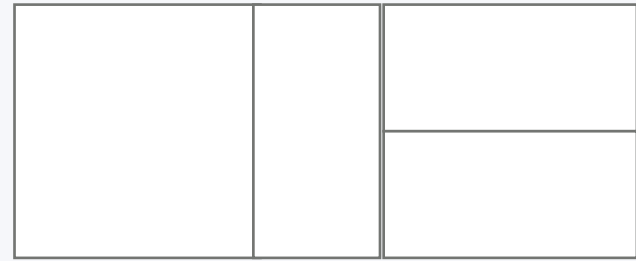
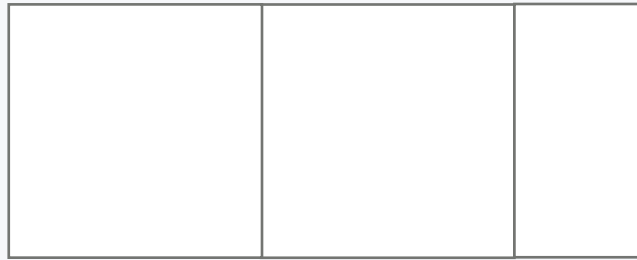
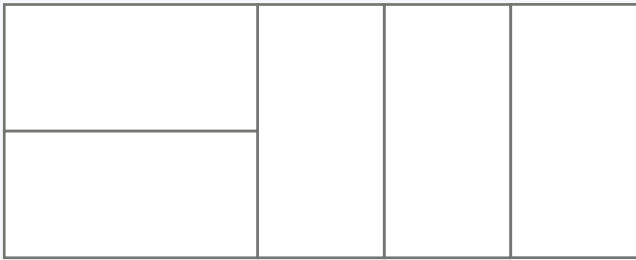
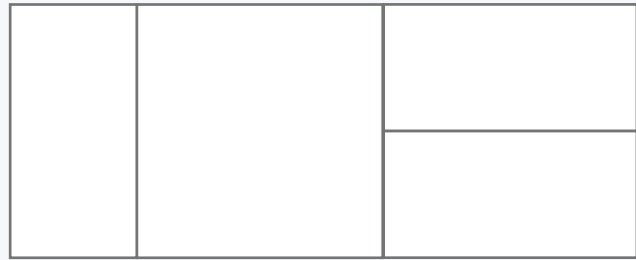
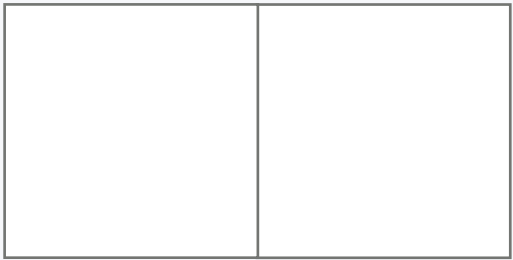
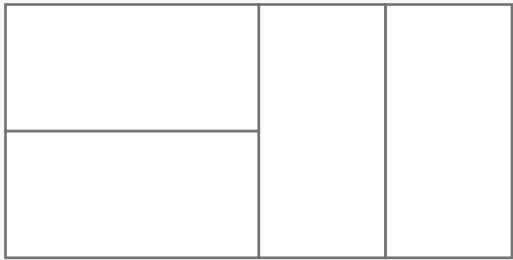
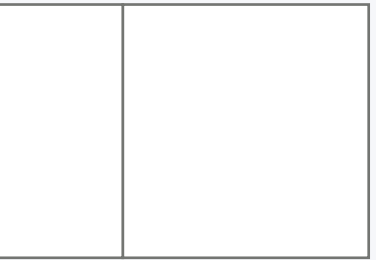
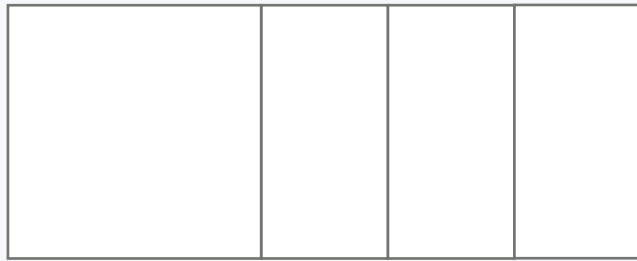
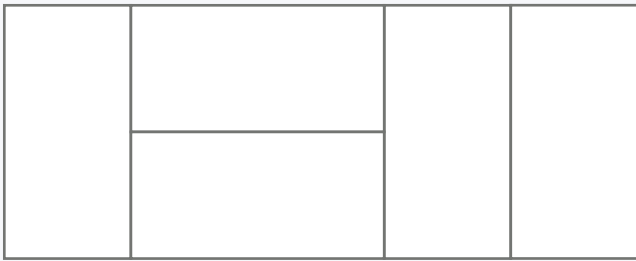
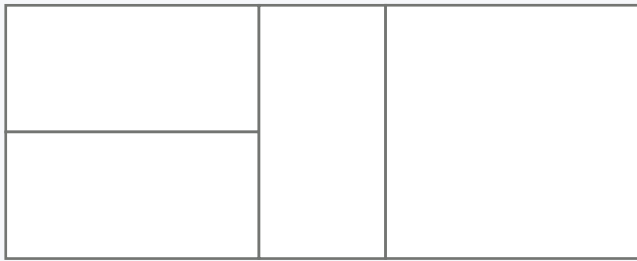
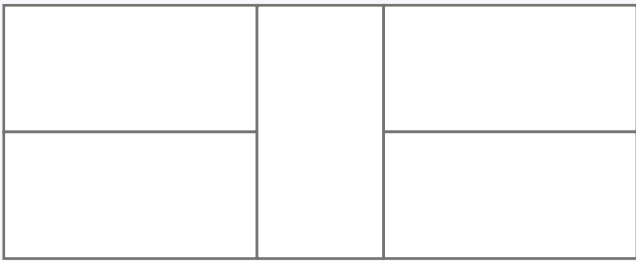
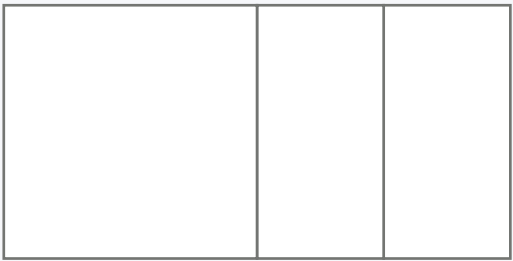
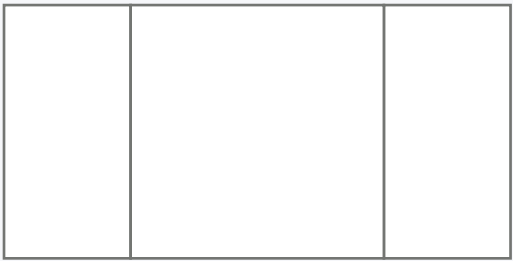
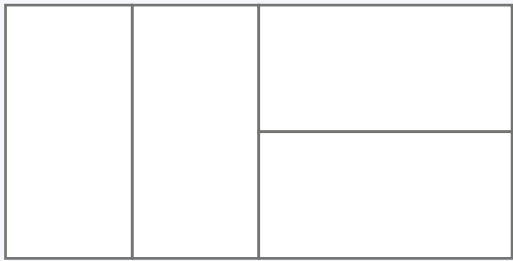
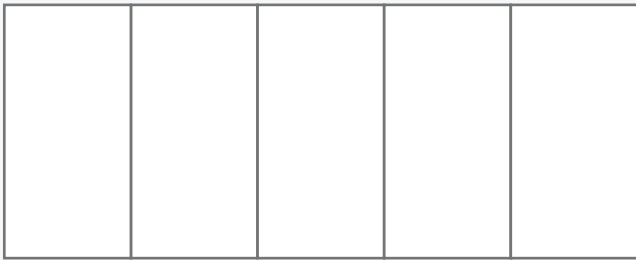
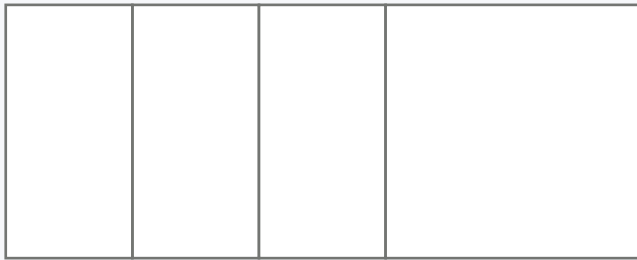
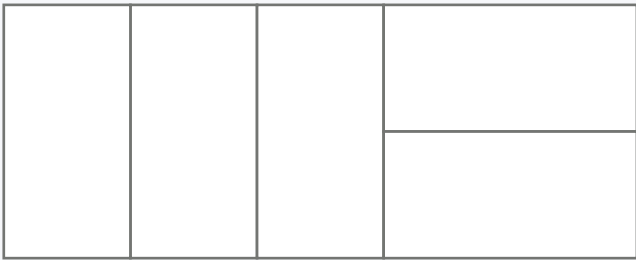
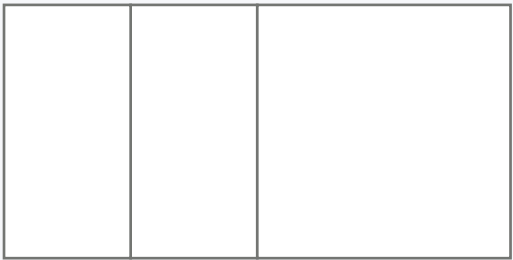
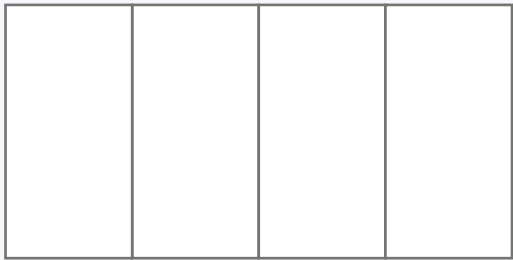
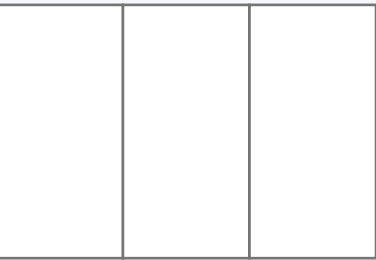
2×n 타일링 2

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

2×3

2×4

2×5



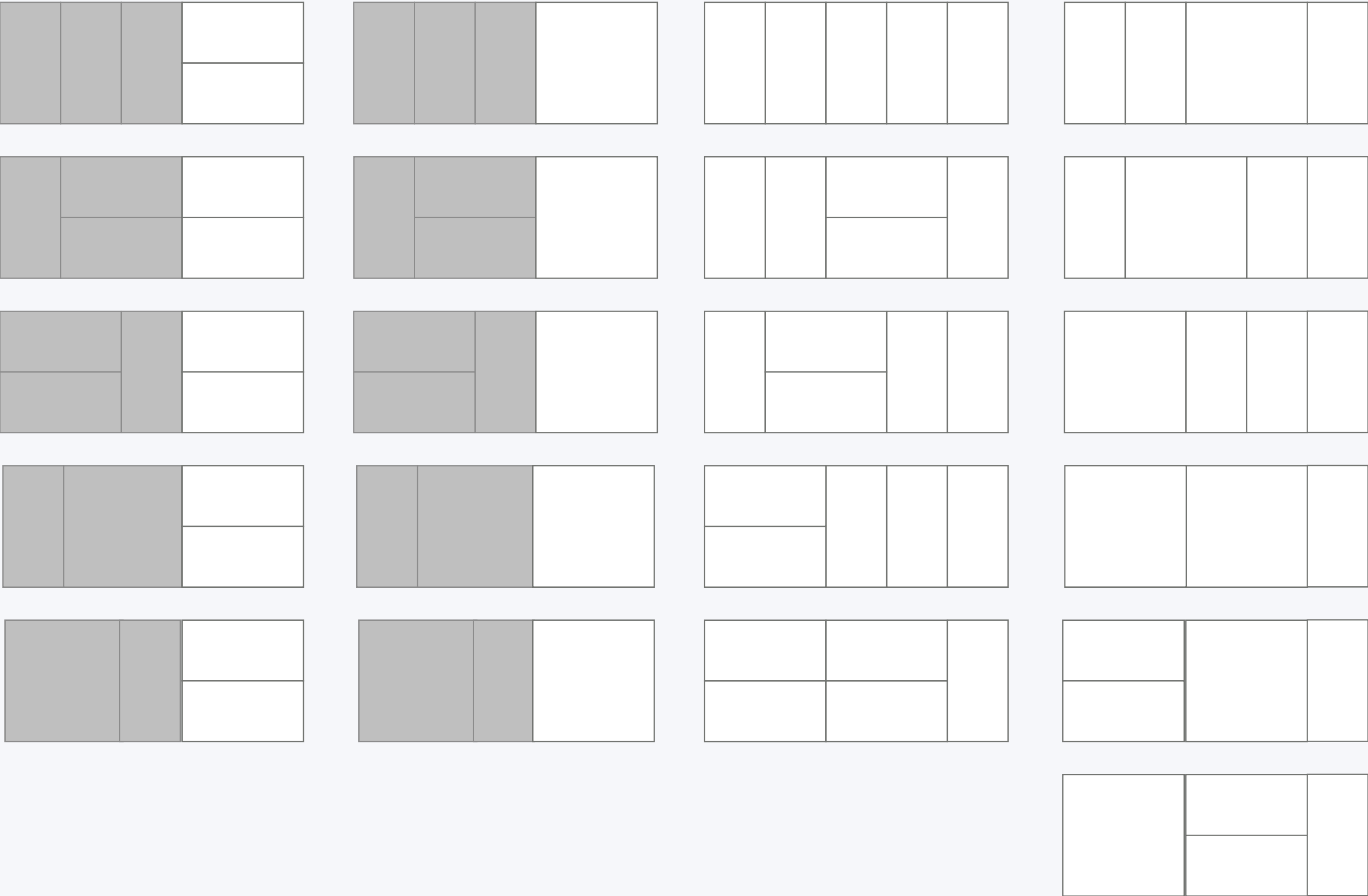
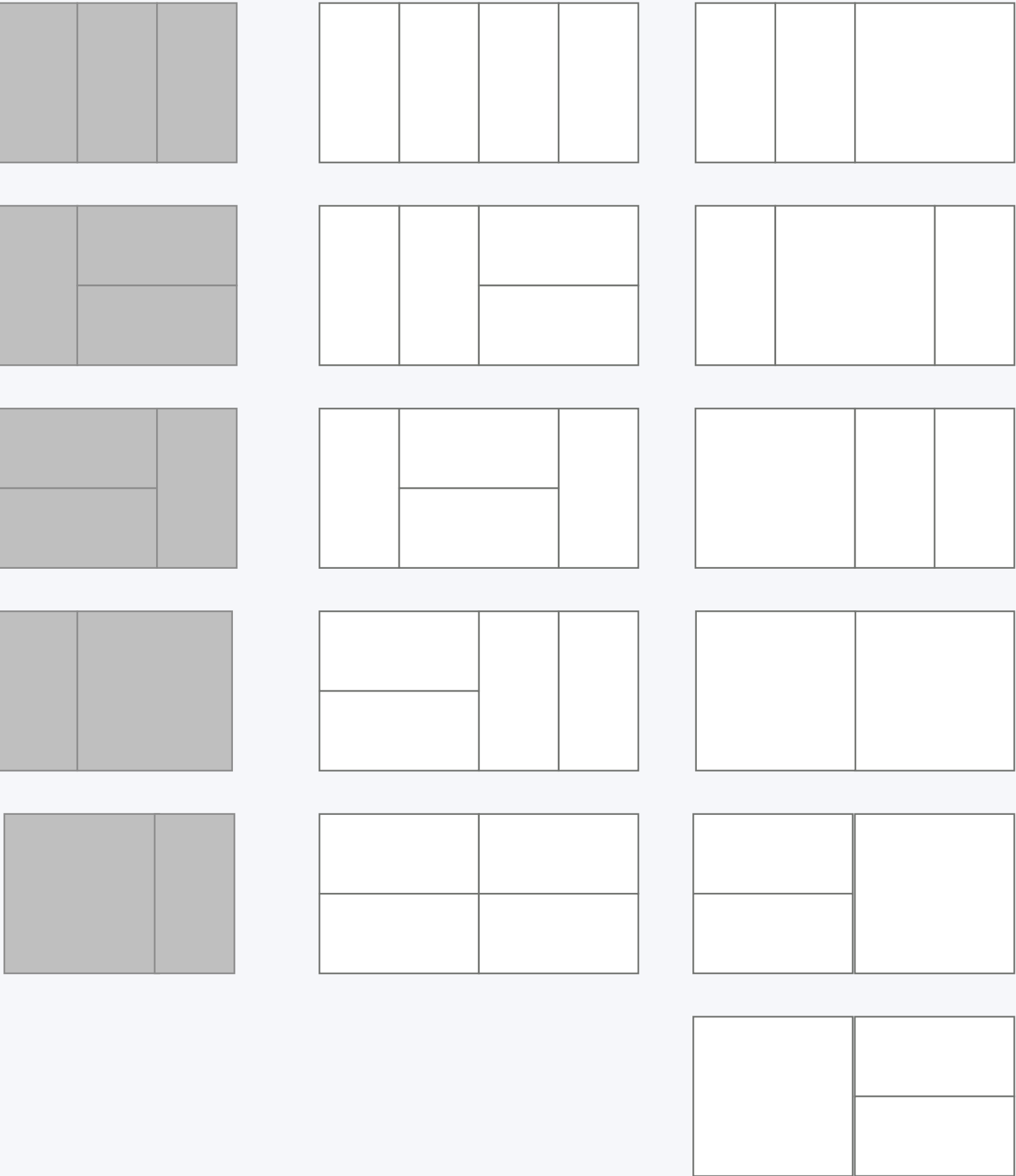
2×n 타일링 2

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

2×3

2×4

2×5



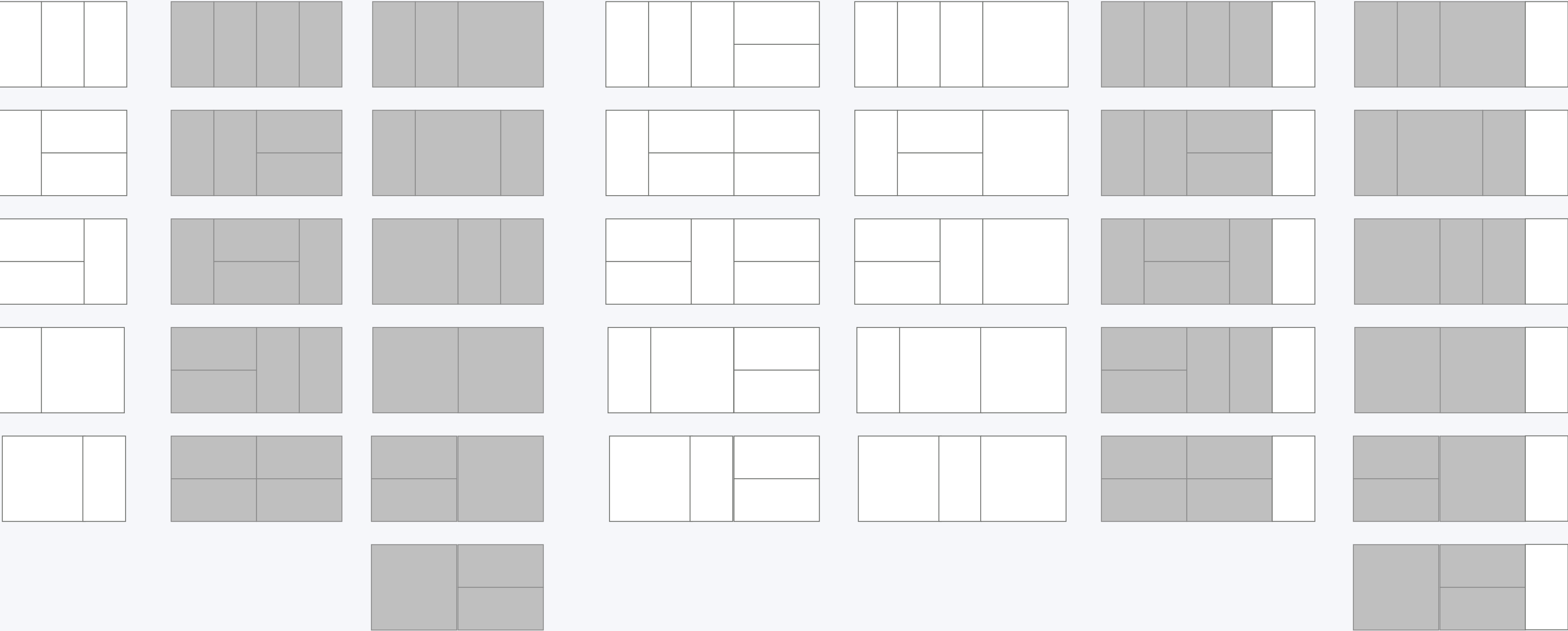
2×n 타일링 2

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

2×3

2×4

2×5

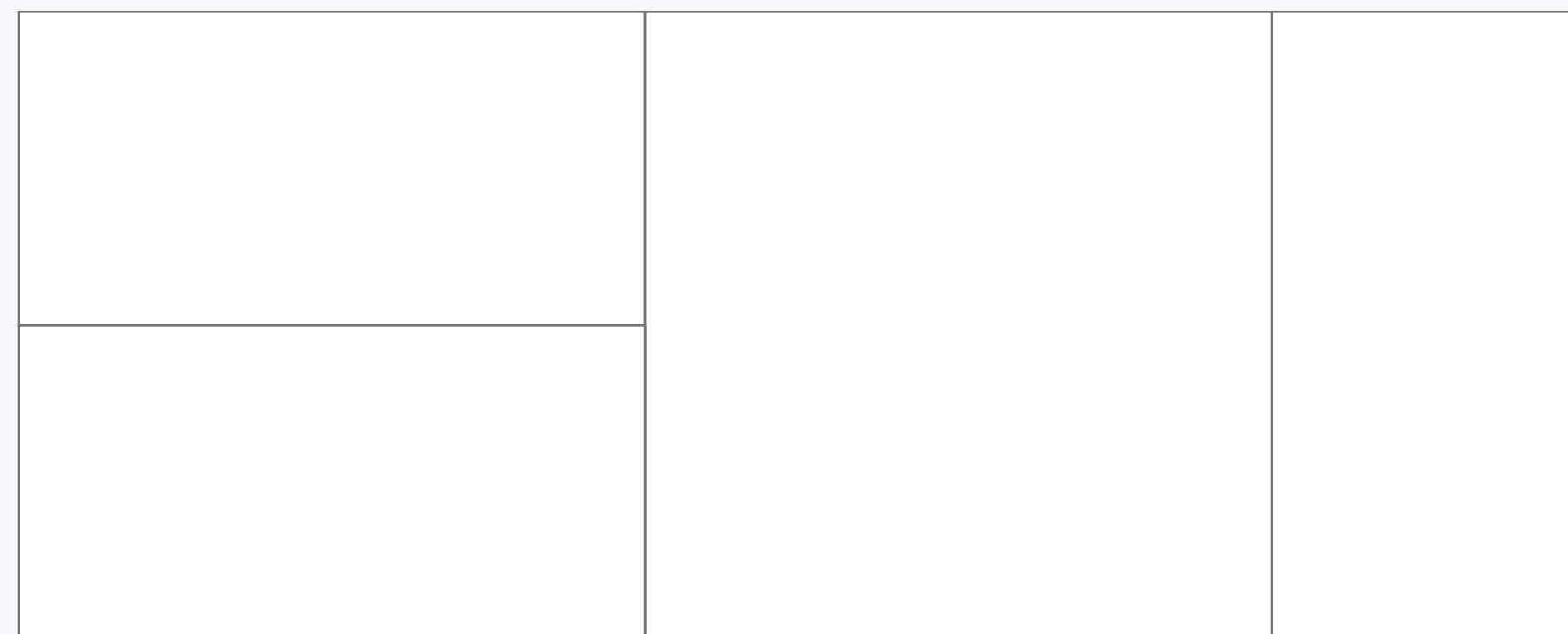


2×n 타일링 2

50

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

- 2×n 직사각형을 1×2, 2×1, 2×2타일로 채우는 방법의 수
- $D[i] = 2 \times i$ 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>
- Java: <https://gist.github.com/Baekjoon/a6dc09520f1581d9b0f0cacb7057b0a6>

1, 2, 3 더하기

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

52

< >
X
: n
O(n) O(1)

- 정수 n 을 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

Case 1: 1
n - 1

Case 2: 2가
n - 2

Case 3: 3
n - 3

1, 2, 3 더하기

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

- $D[i] = i$ 를 1, 2, 3의 조합으로 나타내는 방법의 수

1, 2, 3 더하기

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

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

1, 2, 3 더하기

55

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

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

붕어빵 판매하기

56

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

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

붕어빵 판매하기

57

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

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

붕어빵 판매하기

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

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

붕어빵 판매하기

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

- $D[i]$ = 붕어빵 i 개를 팔아서 얻을 수 있는 최대 수익
- 가능한 경우 생각해보기
- 붕어빵 1개를 $P[1]$ 에 팔기 \rightarrow 남은 붕어빵의 개수: $i-1$
- 붕어빵 2개를 $P[2]$ 에 팔기 \rightarrow 남은 붕어빵의 개수: $i-2$
- ...
- 붕어빵 $i-1$ 개를 $P[i-1]$ 에 팔기 \rightarrow 남은 붕어빵의 개수: 1
- 붕어빵 i 개를 $P[i]$ 에 팔기 \rightarrow 남은 붕어빵의 개수: 0

붕어빵 판매하기

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

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

붕어빵 판매하기

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

- $D[i]$ = 붕어빵 i 개를 팔아서 얻을 수 있는 최대 수익
- 가능한 경우 생각해보기
- 붕어빵 1개를 $P[1]$ 에 팔기 \rightarrow 남은 붕어빵의 개수: $i-1 \rightarrow P[1] + D[i-1]$
- 붕어빵 2개를 $P[2]$ 에 팔기 \rightarrow 남은 붕어빵의 개수: $i-2 \rightarrow P[2] + D[i-2]$
- ...
- 붕어빵 $i-1$ 개를 $P[i-1]$ 에 팔기 \rightarrow 남은 붕어빵의 개수: $1 \rightarrow P[i-1] + D[1]$
- 붕어빵 i 개를 $P[i]$ 에 팔기 \rightarrow 남은 붕어빵의 개수: $0 \rightarrow P[i] + D[0]$
- $D[i] = \max(P[j] + D[i-j]) \ (1 \leq j \leq i)$

붕어빵 판매하기

62

<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

64

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

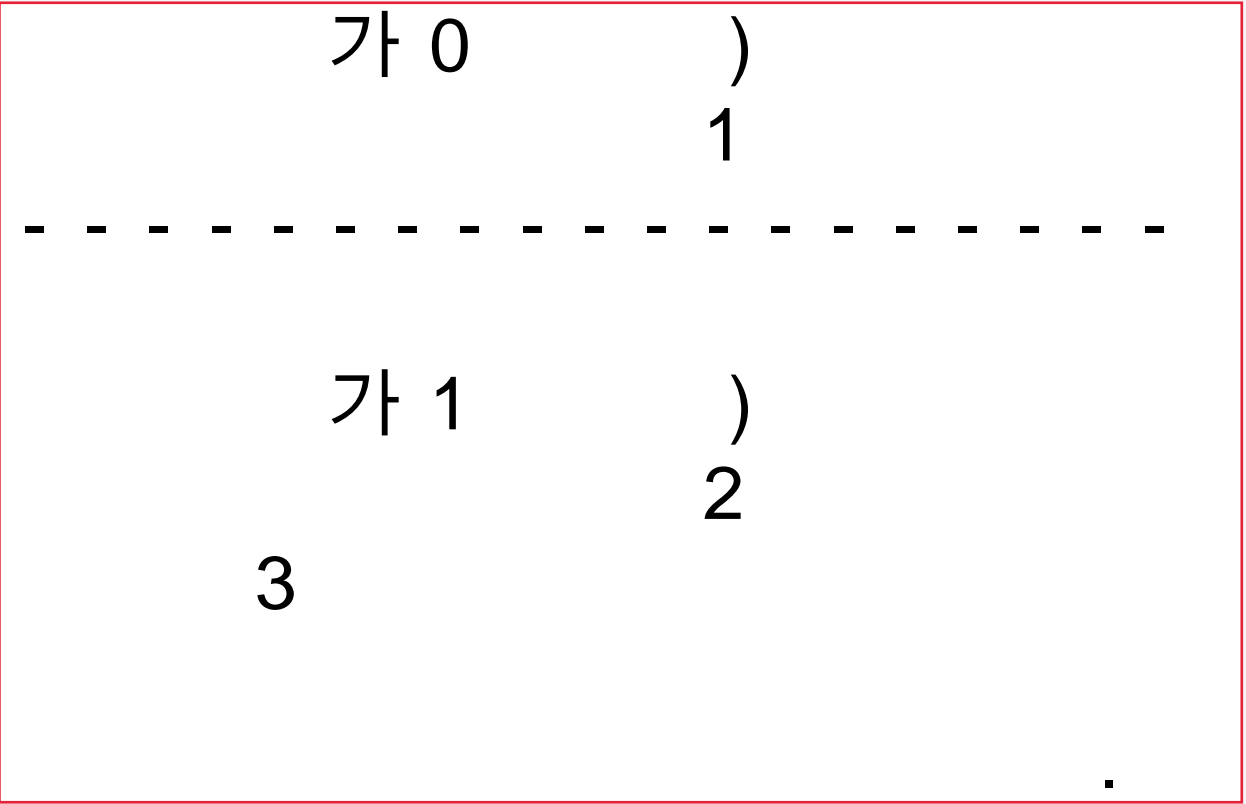
- 인접한 자리의 차이가 1이 나는 수를 계단 수라고 한다
- 예: 45656
- 길이가 N인 계단 수의 개수를 구하는 문제

가 1

쉬운 계단 수

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

- $D[i][j]$ = 길이가 i 이가 마지막 숫자가 j 인 계단 수의 개수
- $D[i][j] = D[i-1][j-1] + D[i-1][j+1]$ ←



```
Cf)            9901
- >            int    long long
                .

Why?)
(A+B)%C = (A%C + B%C)%C가      (      )
                .
```

쉬운 계단 수

66

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

```
for (int i=1; i<=9; i++) d[1][i] = 1; //      :      가1
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]; //      가9      8
        if (j+1 <= 9) d[i][j] += d[i-1][j+1]; //      가0      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 i 이고 마지막 숫자가 j 인 오르막 수의 개수
- $D[1][i] = 1$
- $D[i][j] += D[i-1][k] \ (0 \leq k \leq j)$

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. $\longrightarrow D[1][0] = 0, D[1][1] = 1$
 2. 이친수에서는 1이 두 번 연속으로 나타나지 않는다. 즉, 11을 부분 문자열로 갖지 않는다.
- N자리 이친수의 개수를 구하는 문제

```
D[N][0] = N, 0
D[N][1] = N, 1

D[N][0] = D[N - 1][0] + D[N - 1][1]
D[N][1] = D[N - 1][0]
```


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>

- $D[i][j]$ = i 자리 이친수의 개수 중에서 j 로 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

1

.

77

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

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

$D[N]$ = 가 N

Case 1) 가 0 : $D[N - 1]$

Case 2) 가 1 : $D[N - 2]$

)
 $D[1] = 1$

이친수

78

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

- $D[i]$ = i 자리 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 $i-2$ 자리에 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

80

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

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

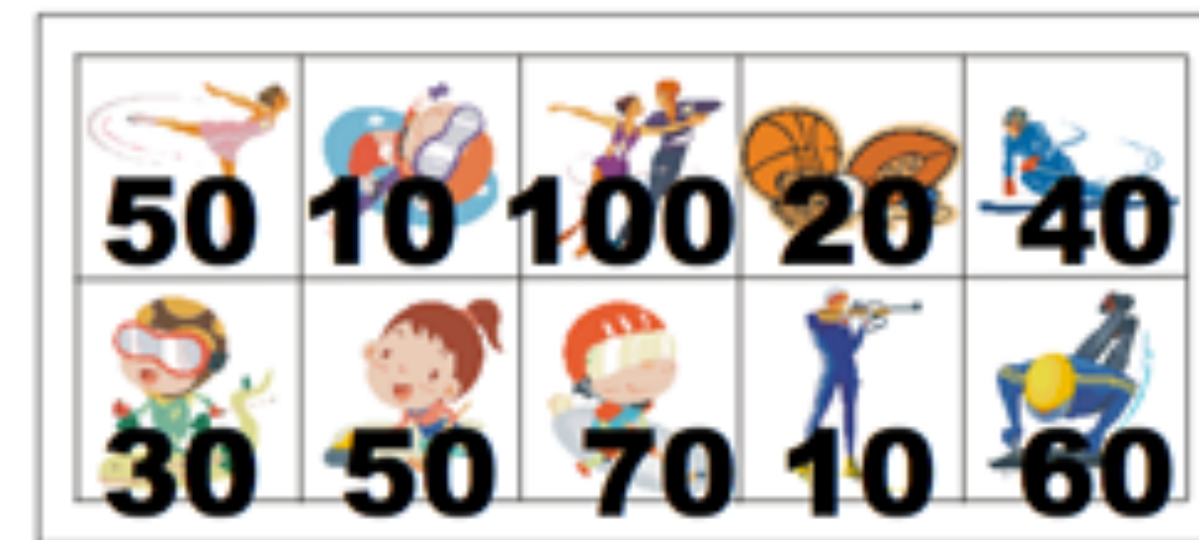
스티커

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

- 스티커 $2n$ 개가 $2 \times n$ 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>

- $D[i][j] = 2 \times i$ 에서 얻을 수 있는 최대 점수, i 번 열에서 뜯는 스티커는 j

 $j = 0 \rightarrow$ 뜯지 않음 // 가

- $j = 1 \rightarrow$ 위쪽 스티커를 뜯음
- $j = 2 \rightarrow$ 아래쪽 스티커를 뜯음

스티커

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

- $D[i][j] = 2 \times i$ 에서 얻을 수 있는 최대 점수, i 번 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 ($D[i][1]$)
 - $i-1$ 열에서 위쪽 스티커는 뜯으면 안된다
 - $\max(D[i-1][0], D[i-1][2]) + A[i][0]$
- 아래쪽 스티커를 뜯음 ($D[i][2]$)
 - $i-1$ 열에서 아래쪽 스티커는 뜯으면 안된다
 - $\max(D[i-1][0], D[i-1][1]) + A[i][1]$

스티커

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

- C/C++: <https://gist.github.com/Baekjoon/3fec4c0ef968b943041a>
- Java: <https://gist.github.com/Baekjoon/a05b5a7ef6e5cd8f7951>

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>

- $D[i] = A[1], \dots, A[i]$ 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- i 에게 가능한 경우
 1. i 번째 포도주를 마시는 경우
 2. i 번째 포도주를 마시지 않는 경우

포도주 시식

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

- $D[i] = A[1], \dots, A[i]$ 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- i 에게 가능한 경우
 1. i 번째 포도주를 마시는 경우
 - $D[i-1] + A[i]$
 2. i 번째 포도주를 마시지 않는 경우
 - $D[i-1]$

포도주 시식

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

- $D[i] = A[1], \dots, A[i]$ 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- i 에게 가능한 경우
 1. i 번째 포도주를 마시는 경우
 - $D[i-1] + A[i]$
 2. i 번째 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>

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

포도주 시식

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

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

포도주 시식

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

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

포도주 시식

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

- $D[i] = A[1], \dots, A[i]$ 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- 0번 연속해서 마신 포도주 $\rightarrow A[i]$ 를 마시지 않음
 - $D[i-1]$
- 1번 연속해서 마신 포도주 $\rightarrow A[i-1]$ 을 마시지 않았음
 - $D[i-2] + A[i]$
- 2번 연속해서 마신 포도주 $\rightarrow A[i-1]$ 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>

- $D[i] = A[1], \dots, A[i]$ 까지 포도주를 마셨을 때, 마실 수 있는 포도주의 최대 양
- 0번 연속해서 마신 포도주 $\rightarrow A[i]$ 를 마시지 않음
 - $D[i-1]$
- 1번 연속해서 마신 포도주 $\rightarrow A[i-1]$ 을 마시지 않았음
 - $D[i-2] + A[i]$
- 2번 연속해서 마신 포도주 $\rightarrow A[i-1]$ 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>

- $D[i] = A[1], \dots, A[i]$ 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
- $i = 3$ 부터 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>

가장 긴 증가하는 부분 수열

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

LIS
(Longest Increasing
Subsequence)

97

- 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>

- $D[i] = A[1], \dots, A[i]$ 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 $A[?], A[?], \dots, A[j], A[i]$ 라고 했을 때, 겹치는 부분 문제를 찾아보자.

가장 긴 증가하는 부분 수열

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

- $D[i] = A[1], \dots, A[i]$ 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 $A[?], A[?], \dots, A[j], A[i]$ 라고 했을 때, 겹치는 부분 문제를 찾아보자.
- $A[?], A[?], \dots, A[j]$ 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

100

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

- $D[i] = A[1], \dots, A[i]$ 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 $A[?], A[?], \dots, A[j], A[i]$ 라고 했을 때, 겹치는 부분 문제를 찾아보자.
- $A[?], A[?], \dots, A[j]$ 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

101

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

- $D[i] = A[1], \dots, A[i]$ 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

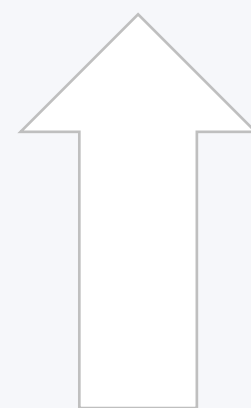
가장 긴 증가하는 부분 수열

102

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

- $D[i] = A[1], \dots, A[i]$ 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

103

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

- $D[i] = A[1], \dots, A[i]$ 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

104

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

- $D[i] = A[1], \dots, A[i]$ 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]	A[5]
10	20

$D[1] = 1$ $10 < 20$ ○

A[1]	A[2]	A[5]
10	20	20

$D[2] = 2$ $20 == 20$ ✗

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

$D[3] = 1$ $10 < 20$ ○

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

$D[4] = 3$ $30 > 20$ ✗

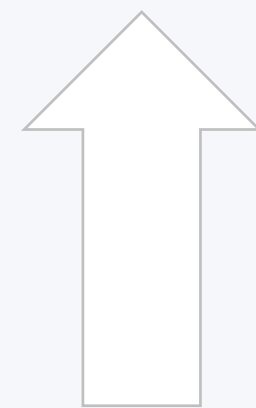
가장 긴 증가하는 부분 수열

105

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

- $D[i] = A[1], \dots, A[i]$ 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

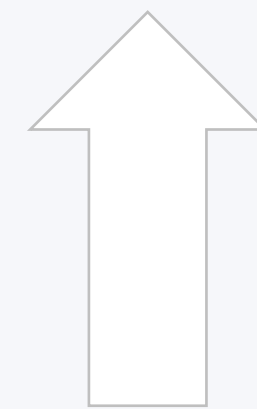
A[1]	A[2]	A[3]	A[4]	A[5]
10	20	10	30	20

가장 긴 증가하는 부분 수열

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

- $D[i] = A[1], \dots, A[i]$ 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

 $D[5] = 2$

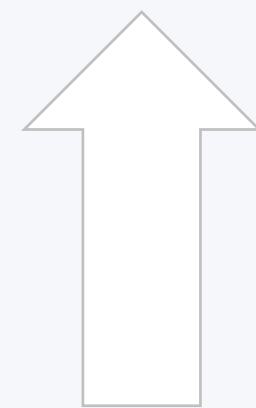
가장 긴 증가하는 부분 수열

107

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

- $D[i] = A[1], \dots, A[i]$ 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]		A[6]	
10	$D[1] = 1$	50	$10 < 50$

A[1]	A[2]		A[6]	
10	20	$D[2] = 2$	50	$20 < 50$

A[1]	A[2]	A[3]		A[6]	
10	20	10	$D[3] = 1$	50	$10 < 50$

A[1]	A[2]	A[3]	A[4]		A[6]	
10	20	10	30	$D[4] = 3$	50	$30 < 50$

A[1]	A[2]	A[3]	A[4]	A[5]		A[6]	
10	20	10	30	20	$D[5] = 2$	50	$20 < 50$

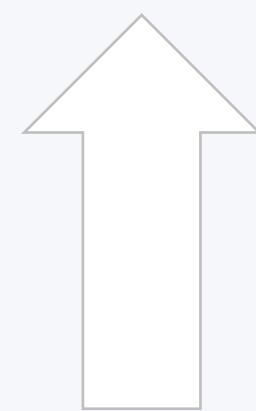
가장 긴 증가하는 부분 수열

108

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

- $D[i] = A[1], \dots, A[i]$ 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]	A[6]
10	50

$D[1] = 1$ $10 < 50$ ○

A[1]	A[2]	A[6]
10	20	50

$D[2] = 2$ $20 < 50$ ○

A[1]	A[2]	A[3]	A[6]
10	20	10	50

$D[3] = 1$ $10 < 50$ ○

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

$D[4] = 3$ $30 < 50$ ○

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

$D[5] = 2$ $20 < 50$ ○

가장 긴 증가하는 부분 수열

109

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

- $D[i] = A[1], \dots, A[i]$ 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

110

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

- $D[i] = A[1], \dots, A[i]$ 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

111

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

- $D[i] = A[1], \dots, A[i]$ 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

112

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

- $D[i] = A[1], \dots, A[i]$ 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		

D[4]		
30		1
Case1)		
D[3]	10	
D[3] + 1 = 2		
Case2)		
D[2]	20	
D[2] + 1 = 3		
Case3)		
D[1]	10	
D[1] + 1 = 2		
-> Case2	가	

가장 긴 증가하는 부분 수열

113

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

- $D[i] = A[1], \dots, A[i]$ 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

114

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

- $D[i] = A[1], \dots, A[i]$ 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

115

<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 $D[1], \dots, D[N]$ 중의 최대값이 된다.

```

        )
        : N
        : O(N)
why) D[1]~D[N-1]
        : O(N^2)

```

가장 긴 증가하는 부분 수열

117

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

- C: <https://gist.github.com/Baekjoon/1602b252bc8f1a1ee044>
- C++: <https://gist.github.com/Baekjoon/667ca791c2b9d5b1d2d2>
- Java: <https://gist.github.com/Baekjoon/9fc6b7ea58a9ea9968b2>

가장 큰 증가하는 부분 수열

118

<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

119

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

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

가장 큰 증가하는 부분 수열

120

<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

121

<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

122

<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

123

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

- C: <https://gist.github.com/Baekjoon/6b9e8f26a82320643d2b>
- C++: <https://gist.github.com/Baekjoon/9a6855e1f942d771ec83>
- Java: <https://gist.github.com/Baekjoon/85045323277de97683d9>

가장 긴 감소하는 부분 수열

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

- 수열 A가 주어졌을 때, 그 수열의 감소하는 부분 수열 중에서 합이 가장 큰 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

125

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

- C/C++: <https://gist.github.com/Baekjoon/4e6c6bbbbe38dc526a75>
- Java: <https://gist.github.com/Baekjoon/7f21125d09f6573f23e755a3df2d83d6>

가장 긴 바이토닉 부분 수열

126

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

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

가장 긴 바이토닉 부분 수열

127

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

- C: <https://gist.github.com/Baekjoon/719292258386a75e0329>
- C++: <https://gist.github.com/Baekjoon/77f2005a22665817527b>
- Java: <https://gist.github.com/Baekjoon/8f3324770000c87f9185>

연속합

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

- n 개의 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 $12+21$ 인 33이 정답이 된다.

	가	.
Ex) 3 -1 3		
->	5	

연속합

129

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

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

연속합

130

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

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

연속합

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

- $D[i]$ = i 번째 수로 끝나는 가장 큰 연속합
- 이렇게 식을 구했으면, i 번째 수에게 가능한 경우를 세야한다
- i 번째 수에게 가능한 경우
 1. $i-1$ 번째 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>

- $D[i]$ = i 번째 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>

- $D[i]$ = i 번째 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

135

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

- $D[i] = i$ 번째 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>

- $D[i]$ = i 번째 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

137

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

- $D[i]$ = i 번째 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>

- $D[i] = i$ 번째 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>

- $D[i] = i$ 번째 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

140

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

- $D[i] = i$ 번째 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>

- $D[i] = i$ 번째 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

- $D[i] = i$ 번째 수로 끝나는 가장 큰 연속합

[illegible]

연속합

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

- $D[i]$ = i 번째 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

144

<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

145

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

- C: <https://gist.github.com/Baekjoon/6577cbbc4ebeda00604c>
- C++: <https://gist.github.com/Baekjoon/dc368ddfc7c138a6411f>
- Java: <https://gist.github.com/Baekjoon/885882aa75ec74535c45>

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

-> ans ans = 0

why)

-1,000

```
int ans = -1000;
for (int i=0; i<n; i++) {
    if (ans < d[i]) {
        ans = d[i];
    }
}
```

-> for i=0
ans 가 -1000

연속합 2

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

- $dl[i]$ = 왼쪽에서부터 구한 연속합 정답
- $dr[i]$ = 오른쪽에서부터 구한 연속합 정답
- 각각의 제거할 수 k 에 대해서 $dl[k-1] + dr[k+1]$ 의 최대값이 정답이 된다

연속합 2

147

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

- C++: <https://gist.github.com/Baekjoon/1988b8af8d857305c3f093d25ccdfd82>
- Java: <https://gist.github.com/Baekjoon/b8f063b0edfff1fdf56587213c94b720>

계단 오르기 //

148

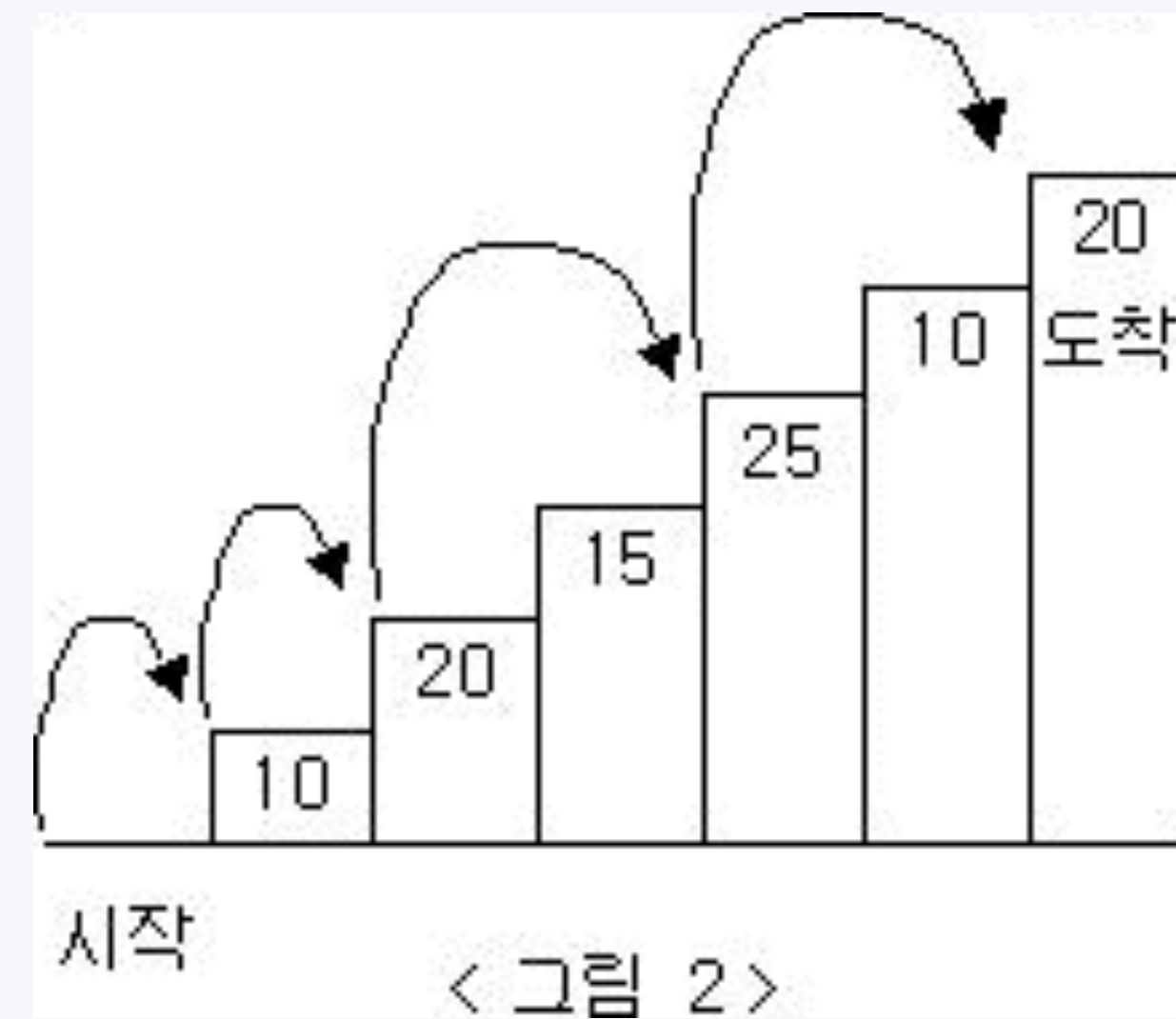
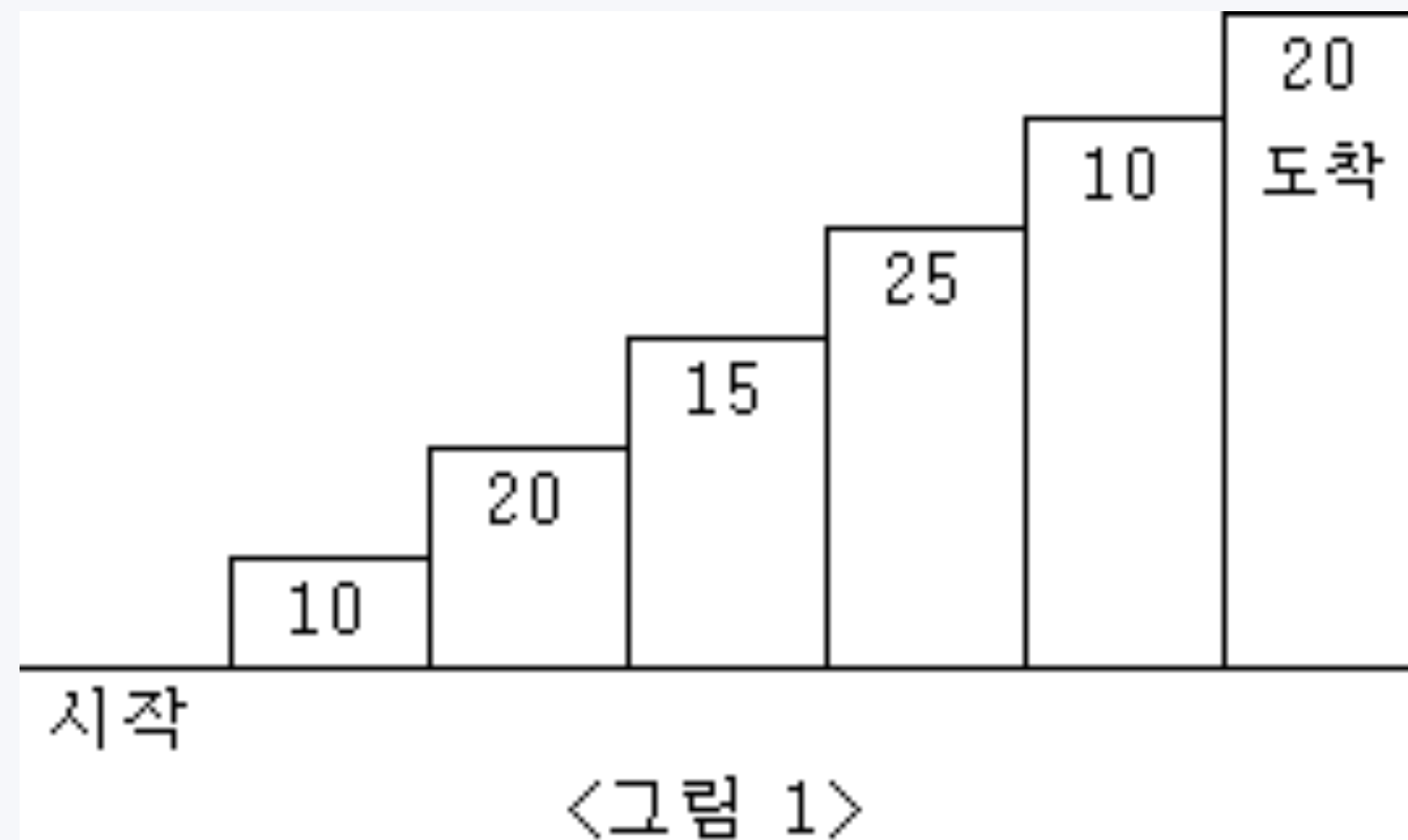
<https://www.acmicpc.net/problem/2579>

- 계단 오르는 데는 다음과 같은 규칙이 있다.
 1. 계단은 한 번에 한 계단씩 또는 두 계단씩 오를 수 있다. 즉, 한 계단을 밟으면서 이어서 다음 계단이나, 다음 다음 계단으로 오를 수 있다.
 2. 연속된 세 개의 계단을 모두 밟아서는 안된다. 단, 시작점은 계단에 포함되지 않는다.
 3. 마지막 도착 계단은 반드시 밟아야 한다.
- 각 계단에 쓰여 있는 점수가 주어질 때 이 게임에서 얻을 수 있는 총 점수의 최대값을 구하는 프로그램을 작성하시오.

계단 오르기

<https://www.acmicpc.net/problem/2579>

149



계단 오르기

150

<https://www.acmicpc.net/problem/2579>

- $D[i][j]$ = i 번째 계단에 올라갔을 때, 최대 점수. i 번째 계단은 j 개 연속해서 올라온 계단임
- $D[i][0] = 0$ 개 연속 $\rightarrow i$ 번째 계단에 올라가야 하기 때문에 불가능한 경우
- $D[i][1] = 1$ 개 연속 $\rightarrow i-1$ 번째 계단은 밟으면 안됨
- $D[i][2] = 2$ 개 연속 $\rightarrow i$ 번째 계단은 밟아야 하고, 반드시 1개 연속해서 올라온 계단이어야 함

계단 오르기

151

<https://www.acmicpc.net/problem/2579>

- $D[i][j]$ = i 번째 계단에 올라갔을 때, 최대 점수. i 번째 계단은 j 개 연속해서 올라온 계단임
- $D[i][1]$ = 1개 연속 \rightarrow $i-1$ 번째 계단은 밟으면 안됨
 - $i-2$ 번째 계단은 1개 연속이거나 2개 연속이거나 상관이 없음
 - $\max(D[i-2][1], D[i-2][2]) + A[i]$
- $D[i][2]$ = 2개 연속 \rightarrow i 번째 계단은 밟아야 하고, 반드시 1개 연속해서 올라온 계단이어야 함
 - $i-1$ 번째 계단은 반드시 1개 연속 계단이어야 함
 - $D[i-1][1] + A[i]$

계단 오르기

<https://www.acmicpc.net/problem/2579>

- $D[i][j]$ = i 번째 계단에 올라갔을 때, 최대 점수. i 번째 계단은 j 개 연속해서 올라온 계단임
- $D[i][1] = 1$ 개 연속 $\rightarrow i-1$ 번째 계단은 밟으면 안됨
 - $i-2$ 번째 계단은 1개 연속이거나 2개 연속이거나 상관없음
 - $\max(D[i-2][1], D[i-2][2]) + A[i]$
- $D[i][2] = 2$ 개 연속 $\rightarrow i$ 번째 계단은 밟아야 하고, 반드시 1개 연속해서 올라온 계단이어야 함
 - $i-1$ 번째 계단은 반드시 1개 연속 계단이어야 함
 - $D[i-1][1] + A[i]$
- 정답은 $D[n][1]$ 과 $D[n][2]$ 중의 최대값이 된다.

계단 오르기

153

<https://www.acmicpc.net/problem/2579>

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

계단 오르기

<https://www.acmicpc.net/problem/2579>

- C/C++: <https://gist.github.com/Baekjoon/ae33086d692514edb61a>
- Java: <https://gist.github.com/Baekjoon/5823d80d59a9ff24ef80>

계단 오르기

<https://www.acmicpc.net/problem/2579>

- $D[i]$ = i 번째 계단에 올라갔을 때, 최대 점수.
- 1개 연속 $\rightarrow i-1$ 번째 계단은 밟으면 안됨
 - $i-2$ 번째 계단은 반드시 밟았어야 함
 - $D[i-2] + A[i]$
- 2개 연속 $\rightarrow i-1$ 번째 계단을 밟고, $i-2$ 번째 계단은 밟으면 안됨
 - $i-3$ 번째 계단은 반드시 밟았어야 함
 - $D[i-3] + A[i-1] + A[i]$

계단 오르기

156

<https://www.acmicpc.net/problem/2579>

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

계단 오르기

157

<https://www.acmicpc.net/problem/2579>

- C/C++: <https://gist.github.com/Baekjoon/616d81f1806273c88266>
- Java: <https://gist.github.com/Baekjoon/d844d321f439b78839b2>

제공수의 합

// 1 2 3

158

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

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

제곱수의 합

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

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

제공수의 합

160

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

- $D[i] = i$ 를 제공수의 합으로 나타냈을 때, 필요한 항의 최소 개수
- $i = ? + ? + \dots + ? + j$
- 마지막 항이 중요하다.
- 마지막 항이 1인 경우 $\rightarrow ? + ? + \dots + ? = i-1$
- 마지막 항이 4인 경우 $\rightarrow ? + ? + \dots + ? = i-4$
- 마지막 항이 9인 경우 $\rightarrow ? + ? + \dots + ? = i-9$
- 마지막 항이 16인 경우 $\rightarrow ? + ? + \dots + ? = i-16$
- 마지막 항이 25인 경우 $\rightarrow ? + ? + \dots + ? = i-25$
- ...

제곱수의 합

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

- $D[i] = i$ 를 제곱수의 합으로 나타냈을 때, 필요한 항의 최소 개수
- $i = ? + ? + \dots + ? + j$
- 마지막 항이 중요하다.
- 마지막 항이 1인 경우 $\rightarrow ? + ? + \dots + ? = i-1 \rightarrow D[i-1] + 1$
- 마지막 항이 4인 경우 $\rightarrow ? + ? + \dots + ? = i-4 \rightarrow D[i-4] + 1$
- 마지막 항이 9인 경우 $\rightarrow ? + ? + \dots + ? = i-9 \rightarrow D[i-9] + 1$
- 마지막 항이 16인 경우 $\rightarrow ? + ? + \dots + ? = i-16 \rightarrow D[i-16] + 1$
- 마지막 항이 25인 경우 $\rightarrow ? + ? + \dots + ? = i-25 \rightarrow D[i-25] + 1$
- ...

제곱수의 합

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

- $D[i]$ = i 를 제곱수의 합으로 나타냈을 때, 필요한 항의 최소 개수
- $D[i] = \min(D[i-j^2]+1)$ (~~$1 \leq i \leq j^2$~~)

$$\begin{aligned} i-j^2 &\geq 0 \\ j^2 &\leq i \end{aligned}$$

$D[i-j^2]$:

1 :

?

제곱수의 합

163

<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

164

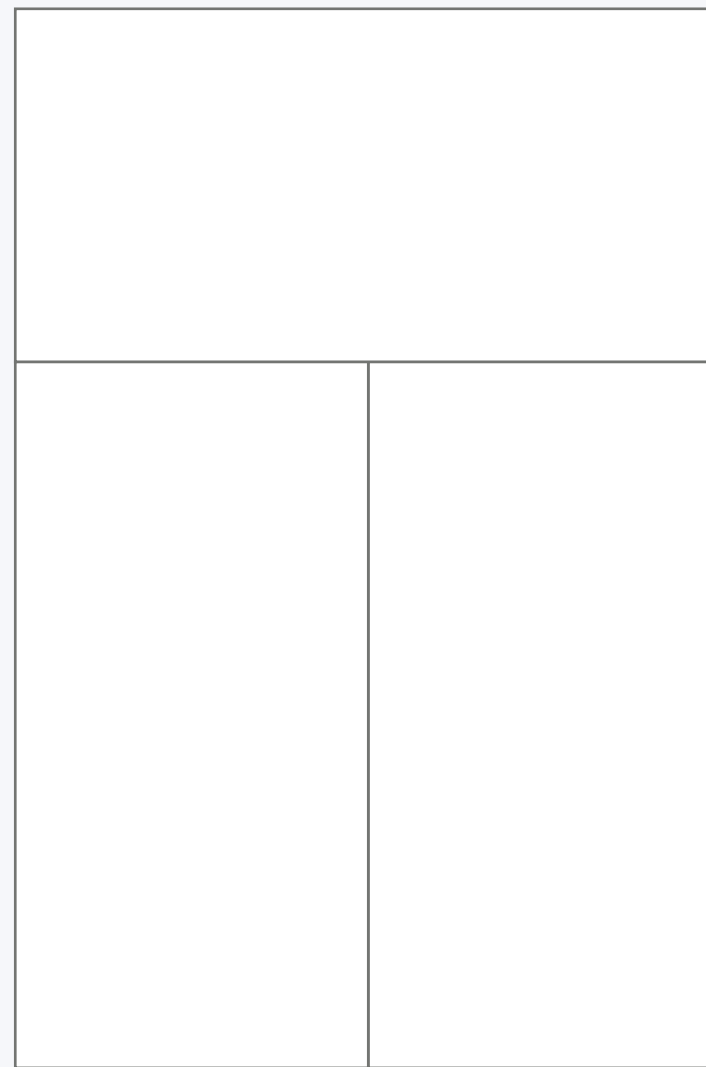
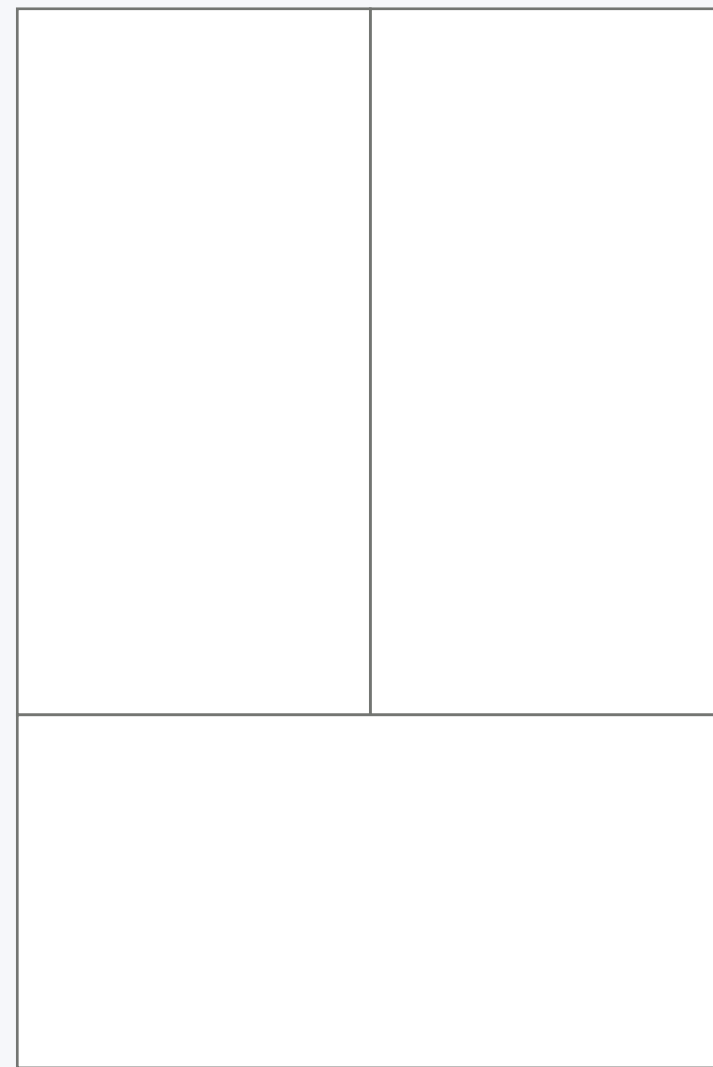
<https://www.acmicpc.net/problem/1699>

- C/C++: <https://gist.github.com/Baekjoon/66c23e0a64ae7924aa19>
- Java: <https://gist.github.com/Baekjoon/73d0186fef9bbd633f95>

타일 채우기

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

- $3 \times N$ 을 1×2 , 2×1 로 채우는 방법의 수
- $D[i] = 3 \times i$ 를 채우는 방법의 수
- 마지막에 올 수 있는 가능한 경우의 수

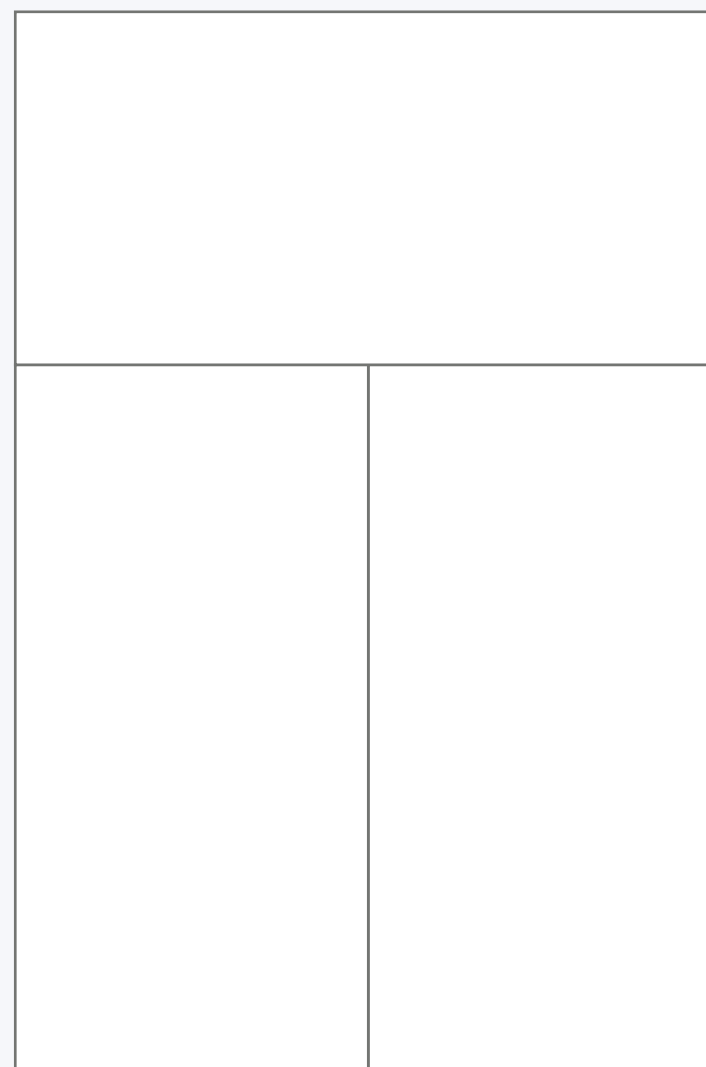
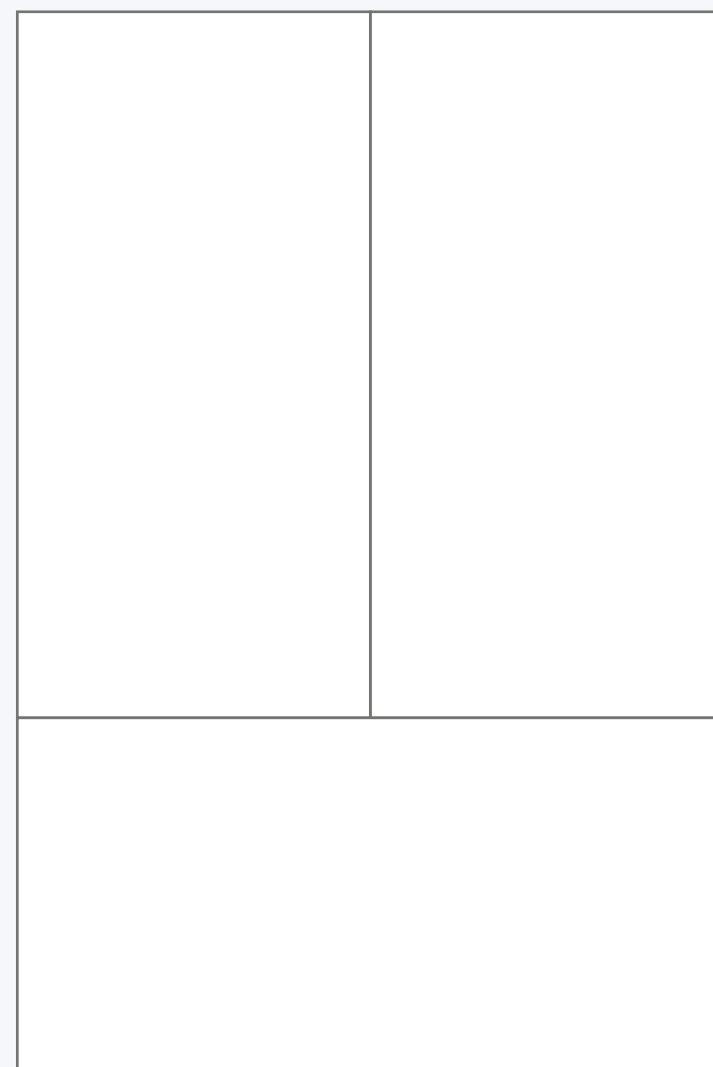


타일 채우기

166

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

- $3 \times N$ 을 1×2 , 2×1 로 채우는 방법의 수
- $D[i] = 3 \times i$ 를 채우는 방법의 수
- $D[i] = 3 * D[i-2]$ (아니다)

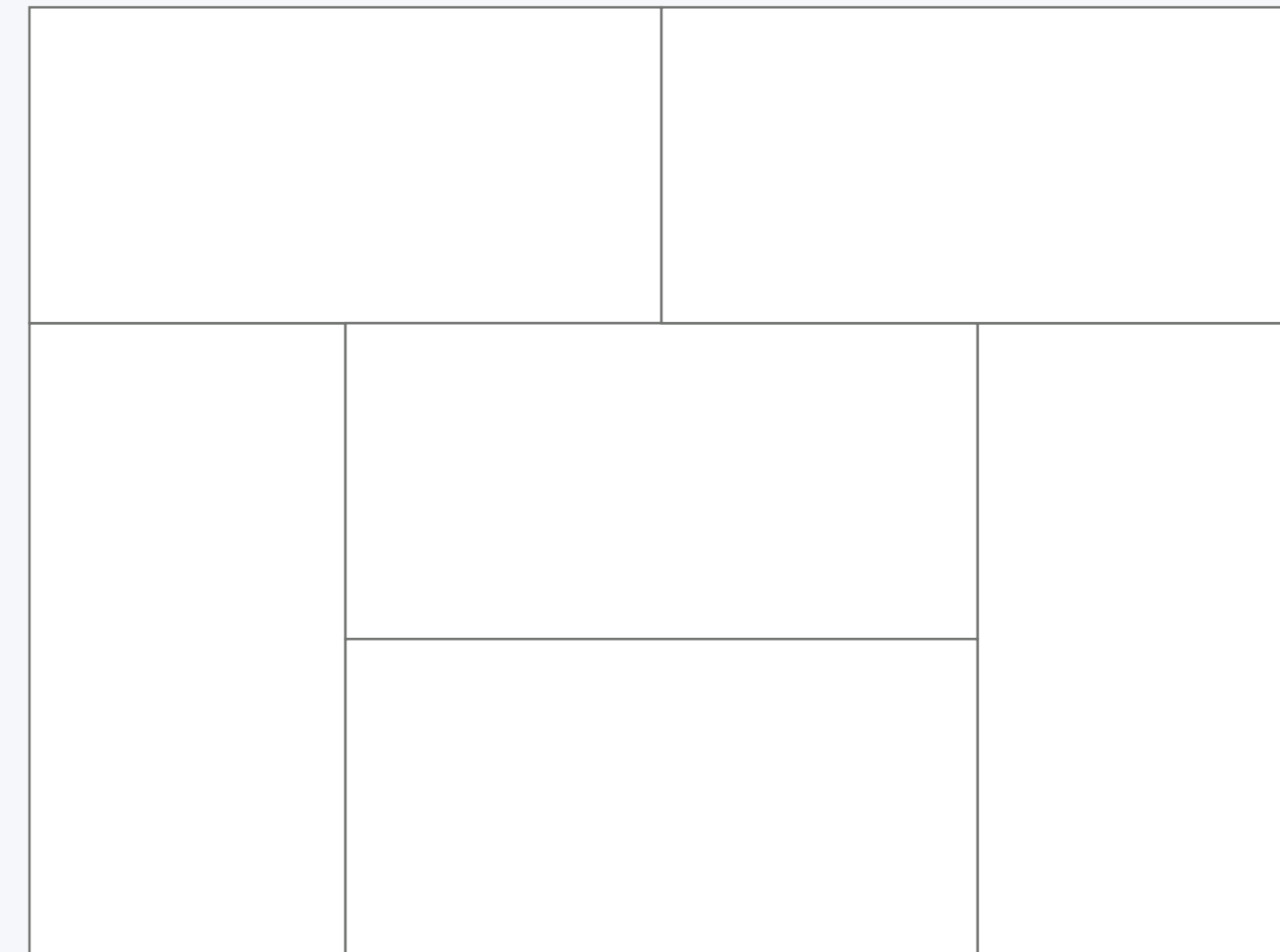
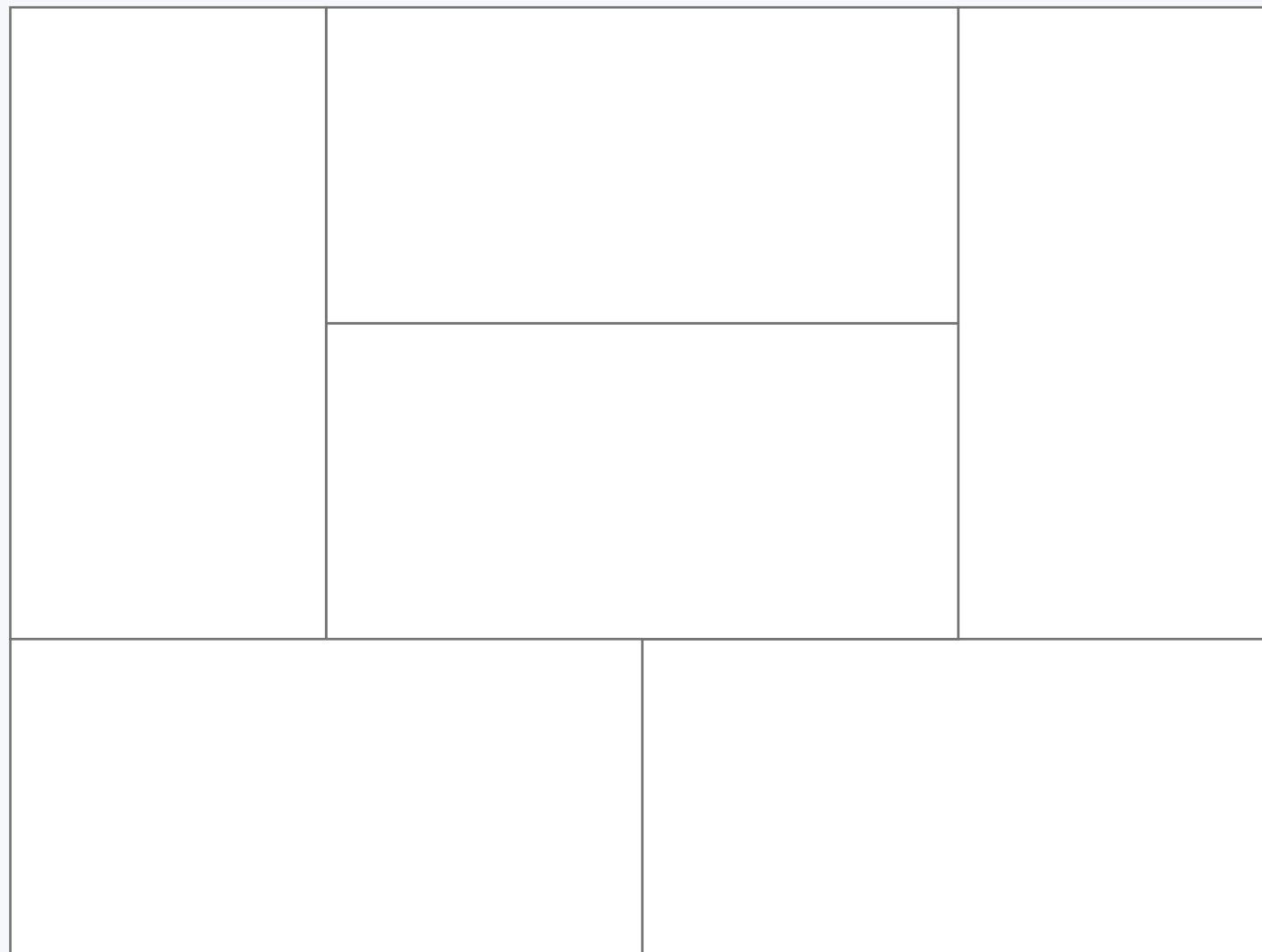


타일 채우기

167

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

- $3 \times N$ 을 1×2 , 2×1 로 채우는 방법의 수
- $D[i] = 3 \times i$ 를 채우는 방법의 수
- 가능한 경우가 더 있다.

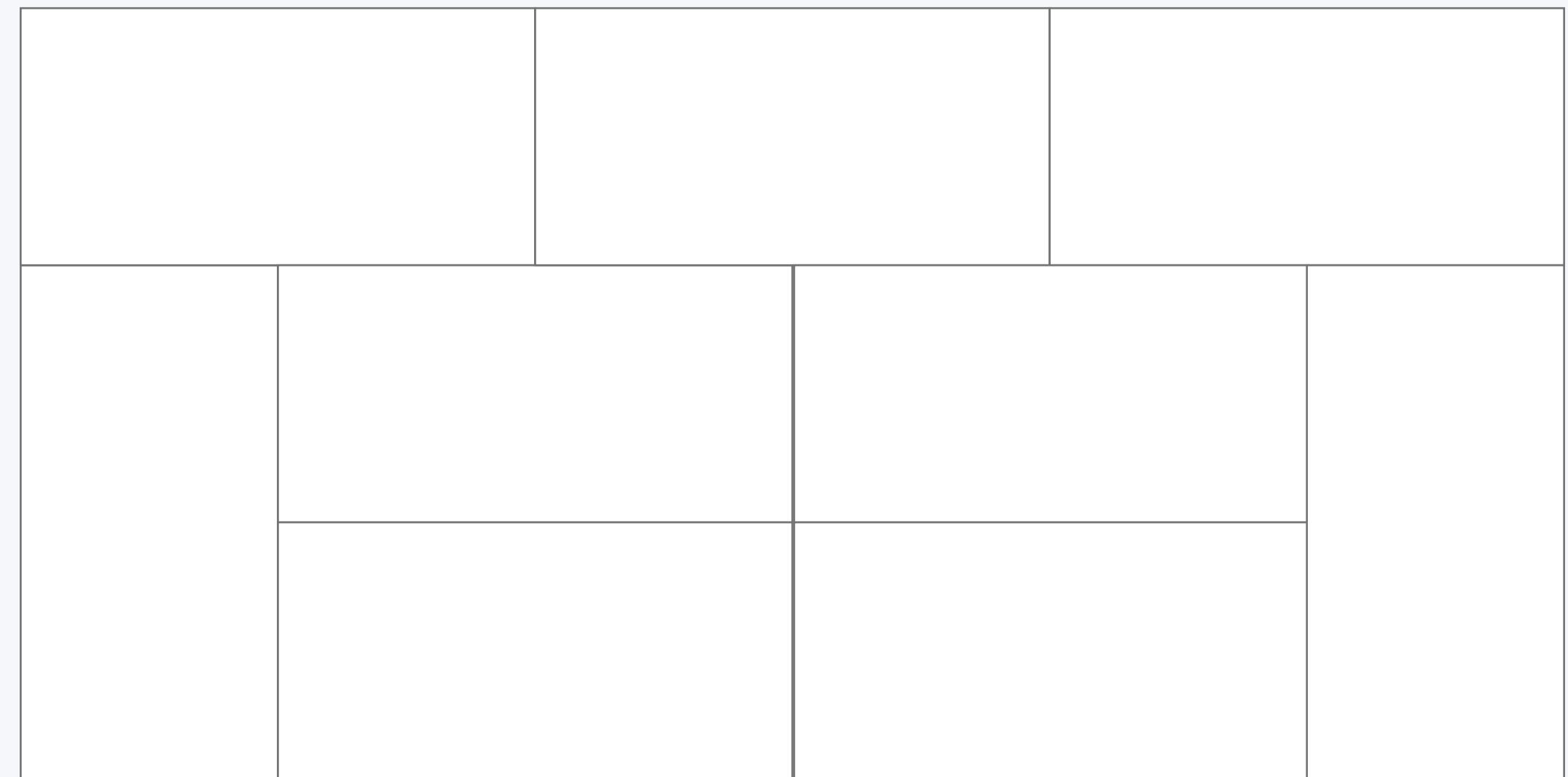
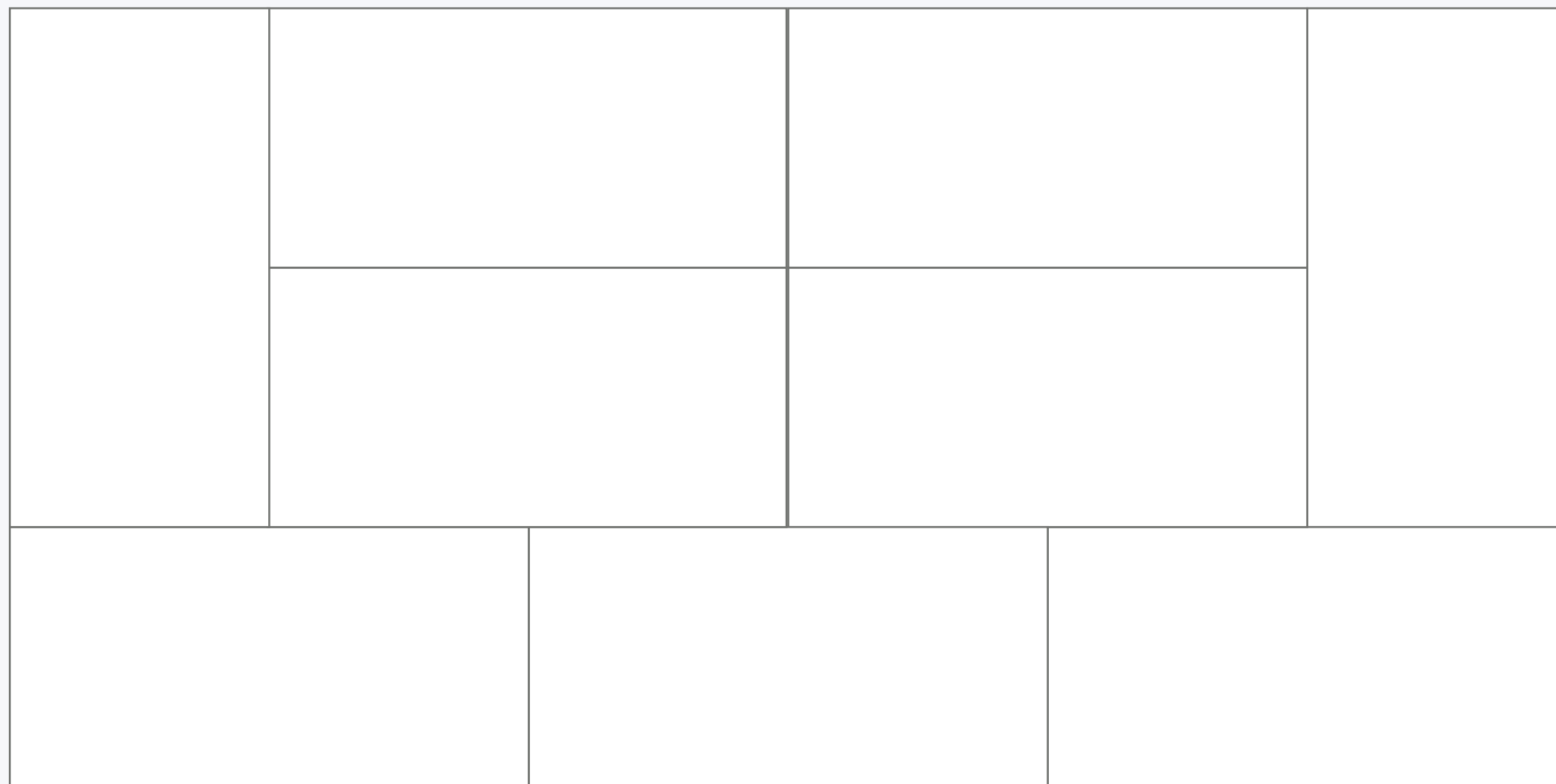


타일 채우기

168

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

- $3 \times N$ 을 1×2 , 2×1 로 채우는 방법의 수
- $D[i] = 3 \times i$ 를 채우는 방법의 수
- 가능한 경우가 더 있다.



타일 채우기

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

- $3 \times N$ 을 1×2 , 2×1 로 채우는 방법의 수
- $D[i] = 3 \times i$ 를 채우는 방법의 수
- $D[i] = 3 * D[i-2] + 2 * D[i-4] + 2 * D[i-6] + \dots$

타일 채우기

170

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

- C/C++: <https://gist.github.com/Baekjoon/a27529894d77d469d252>
- Java: <https://gist.github.com/Baekjoon/fe102cef6f7b7ebecdedef00ec811a7c>

파도반 수열

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

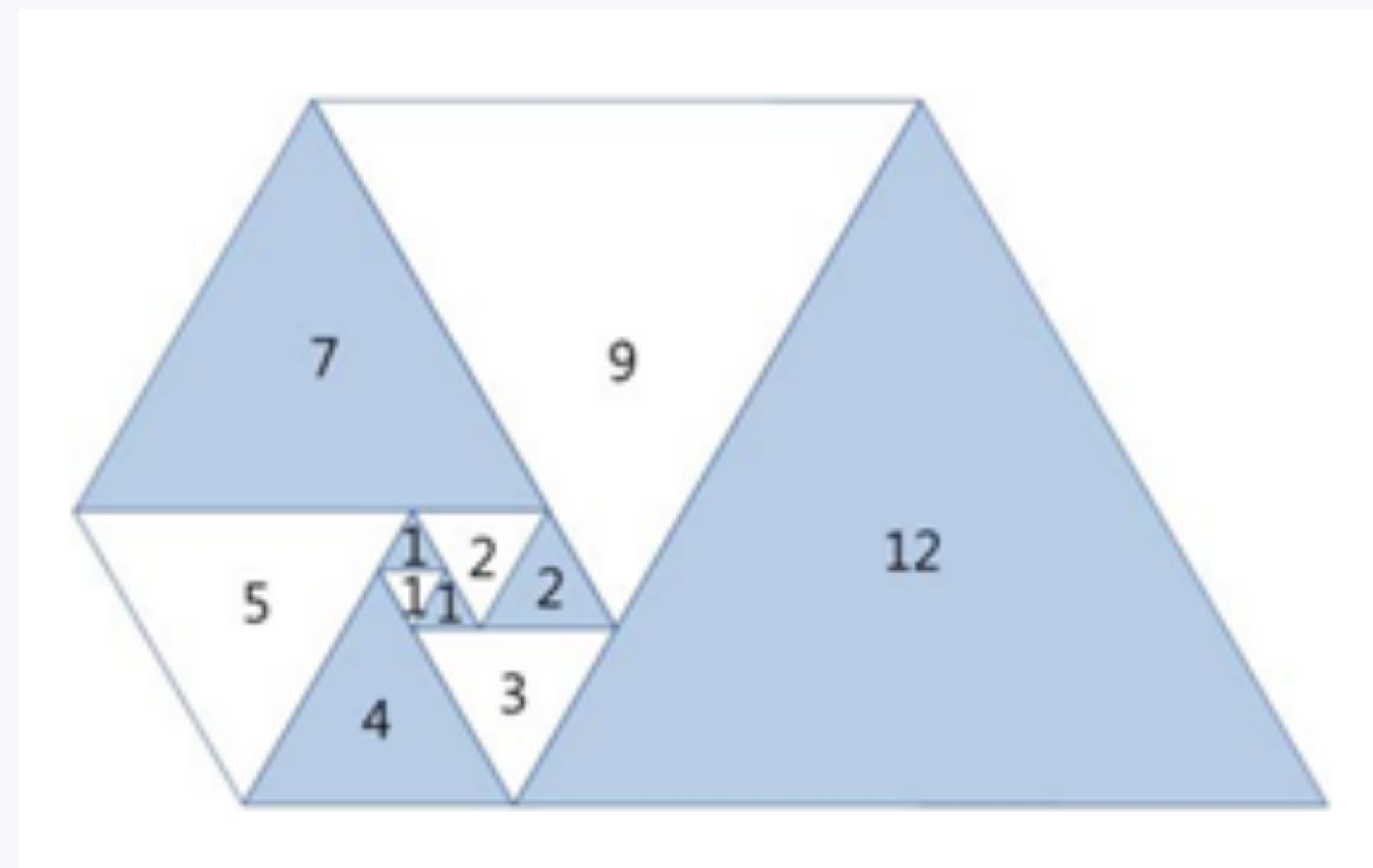
- 오른쪽 그림과 같이 삼각형이 나선 모양으로 놓여져 있다
- 첫 삼각형은 정삼각형으로 변의 길이는 1이다
- 그 다음에는 다음과 같은 과정으로 정삼각형을 계속 추가한다
- 나선에서 가장 긴 변의 길이를 k 라 했을 때, 그 변에 길이가 k 인 정삼각형을 추가한다
- 파도반 수열 $P(N)$ 은 나선에 있는 정삼각형의 변의 길이이다
- $P(1)$ 부터 $P(10)$ 까지 첫 10개 숫자는 1, 1, 1, 2, 2, 3, 4, 5, 7, 9이다
- N 이 주어졌을 때, $P(N)$ 을 구하는 문제

파도반 수열

172

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

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



파도반 수열

173

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

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

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

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

(K * N) :
N :
-> $O(K * N^2)$

합분해

175

<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
- $? + ? + ? + ? + \dots + ? + L = N$
- 위의 식이 나타내는 값: $D[K][N]$
- $? + ? + ? + ? + \dots + ? = N - L$
- 위의 식이 나타내는 값: $D[K-1][N-L]$
- $D[K][N] = \sum D[K-1][N-L] \ (0 \leq L \leq N)$

합분해

176

<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>

- C/C++: <https://gist.github.com/Baekjoon/a334580d1729037f5fb1>
- Java: <https://gist.github.com/Baekjoon/354ed0a3657ecbf00c67>

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
- $D[K][N] = \sum D[K-1][N-L] \ (0 \leq L \leq N)$
- $D[K][N] = \sum D[K-1][N-L] \ (0 \leq N-L \leq N)$
- $D[K][N] = \sum D[K-1][L] \ (0 \leq L \leq N)$

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
- $D[K][N] = \sum D[K-1][L] \ (0 \leq L \leq N)$
- $D[K][N] = D[K-1][0] + D[K-1][1] + \dots + D[K-1][N-1] + D[K-1][N]$
- $D[K][N-1] = D[K-1][0] + D[K-1][1] + \dots + D[K-1][N-1]$

합분해

180

<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
- $D[K][N] = \sum D[K-1][L] \ (0 \leq L \leq N)$
- $D[K][N] = D[K-1][0] + D[K-1][1] + \dots + D[K-1][N-1] + D[K-1][N]$
- $D[K][N-1] = D[K-1][0] + D[K-1][1] + \dots + D[K-1][N-1]$
- $D[K][N] = D[K][N-1] + D[K-1][N]$

합분해

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

- C++: <https://gist.github.com/Baekjoon/307eeb5e498f68a5f0fab64f6ae0bd05>
- Java: <https://gist.github.com/Baekjoon/3b53e9288ac9d652d8a6e34688c272c8>

암호 코드

182

<https://www.acmicpc.net/problem/2011>

- 어떤 암호가 주어졌을 때, 그 암호의 해석이 몇 가지가 나올 수 있는지 구하는 문제
- A는 1, B는 2, ..., Z는 26
- BEAN -> 25114
- 25114 -> "BEAAD", "YAAD", "YAN", "YKD", "BEKD", "BEAN"

암호 코드

183

<https://www.acmicpc.net/problem/2011>

- $D[i]$ = i 번째 문자까지 해석했을 때, 나올 수 있는 해석의 가짓수
- i 번째 문자에게 가능한 경우
- 1자리 암호
- 2자리 암호

암호 코드

<https://www.acmicpc.net/problem/2011>

- $D[i]$ = i 번째 문자까지 해석했을 때, 나올 수 있는 해석의 가짓수
- i 번째 문자에게 가능한 경우
- 1자리 암호
 - 0을 제외
- 2자리 암호
 - $10 \leq x \leq 26$

암호 코드

185

<https://www.acmicpc.net/problem/2011>

```
d[0] = 1;
for (int i=1; i<=n; i++) {
    int x = s[i] - '0';
    if (1 <= x && x <= 9) {
        d[i] = (d[i] + d[i-1]) % mod;
    }
    if (i==1) continue;
    if (s[i-1] == '0') continue;
    x = (s[i-1] - '0') * 10 + (s[i] - '0');
    if (10 <= x && x <= 26) {
        d[i] = (d[i] + d[i-2]) % mod;
    }
}
```

암호 코드

<https://www.acmicpc.net/problem/2011>

- C: <https://gist.github.com/Baekjoon/33126f8bcbfa55ebc9b1>
- C++: <https://gist.github.com/Baekjoon/dc071a89a81cb88fb887>
- Java: <https://gist.github.com/Baekjoon/725d19f289d7dda4a191>