DP #1

김현정 Acka1357@gmail.com

Dynamic Programming

동적계획법

Dynamic Programming

동적계획법이란

- 문제를 여러 개의 작은 부분 문제(sub-problem)으로 나누어 해결하는 문제해결 기법

Dynamic Programming

동적계획법이란

- 문제를 여러 개의 작은 부분 문제(sub-problem)으로 나누어 해결하는 문제해결 기법

- Devide&Conquer와는 다르다.

Dynamic Programming

동적계획법이란

- 이미 계산된 부분 문제가 다시 발생하면
- 새롭게 계산하지 않고 이전의 계산값을 참조하여 이용한다.

Dynamic Programming

피보나치 함수

- fib(N) = fib(N – 1) + fib(N - 2)
- fib(0) = 1
- fib(1) = 1

Dynamic Programming

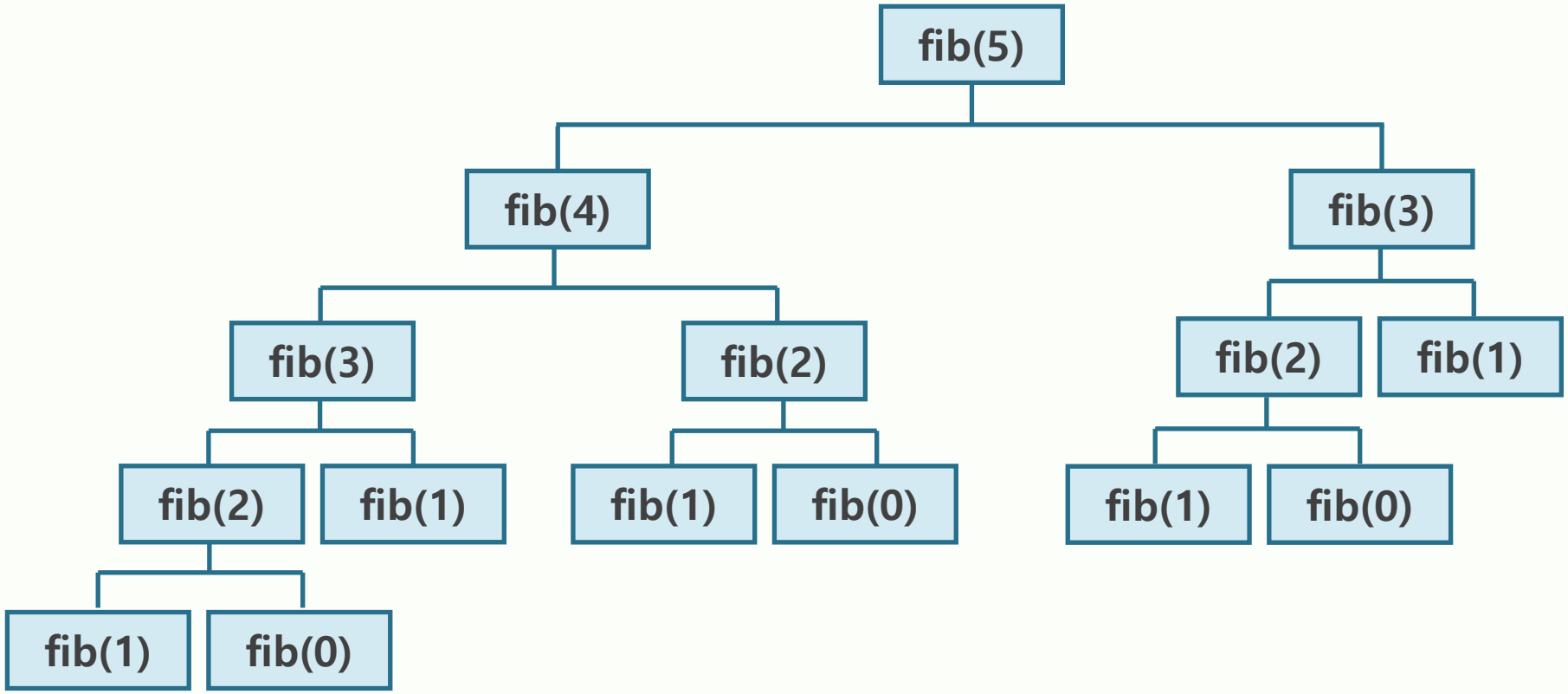
피보나치 함수

- fib(5)을 Devide&Conquer로 구한다면

Dynamic Programming

피보나치 함수

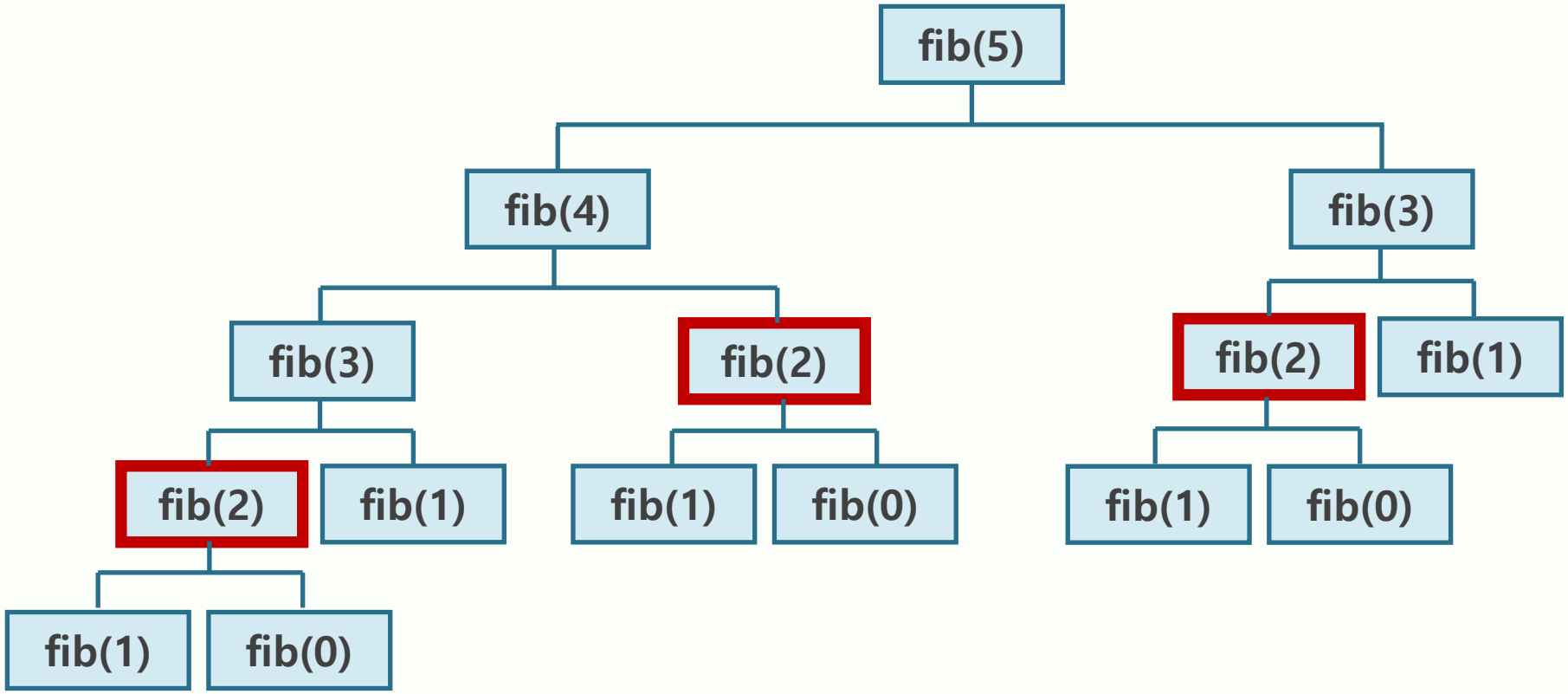
- fib(5)을 Devide&Conquer로 구한다면



Dynamic Programming

피보나치 함수

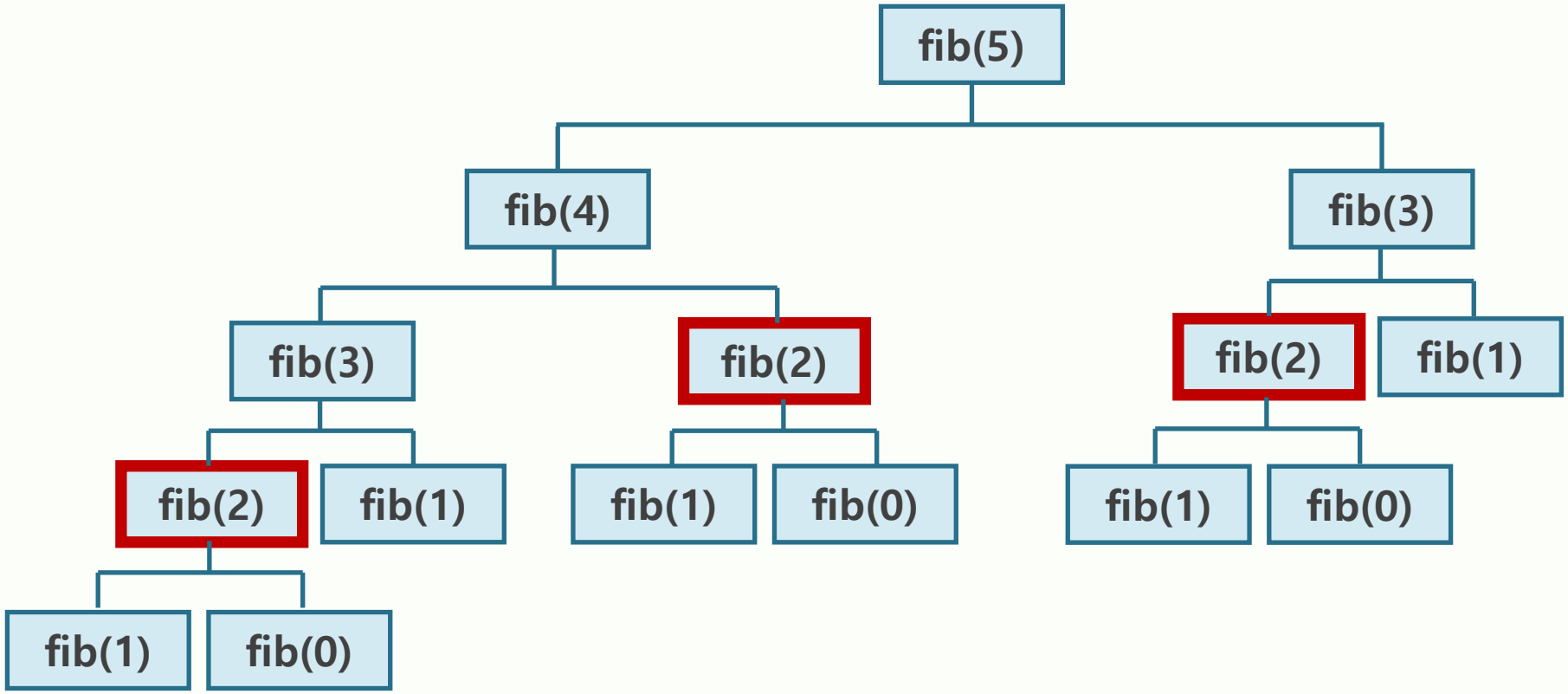
- fib(5)을 Devide&Conquer로 구한다면



Dynamic Programming

피보나치 함수

- fib(2)는 항상 2라는 값을 계산한다.



Dynamic Programming

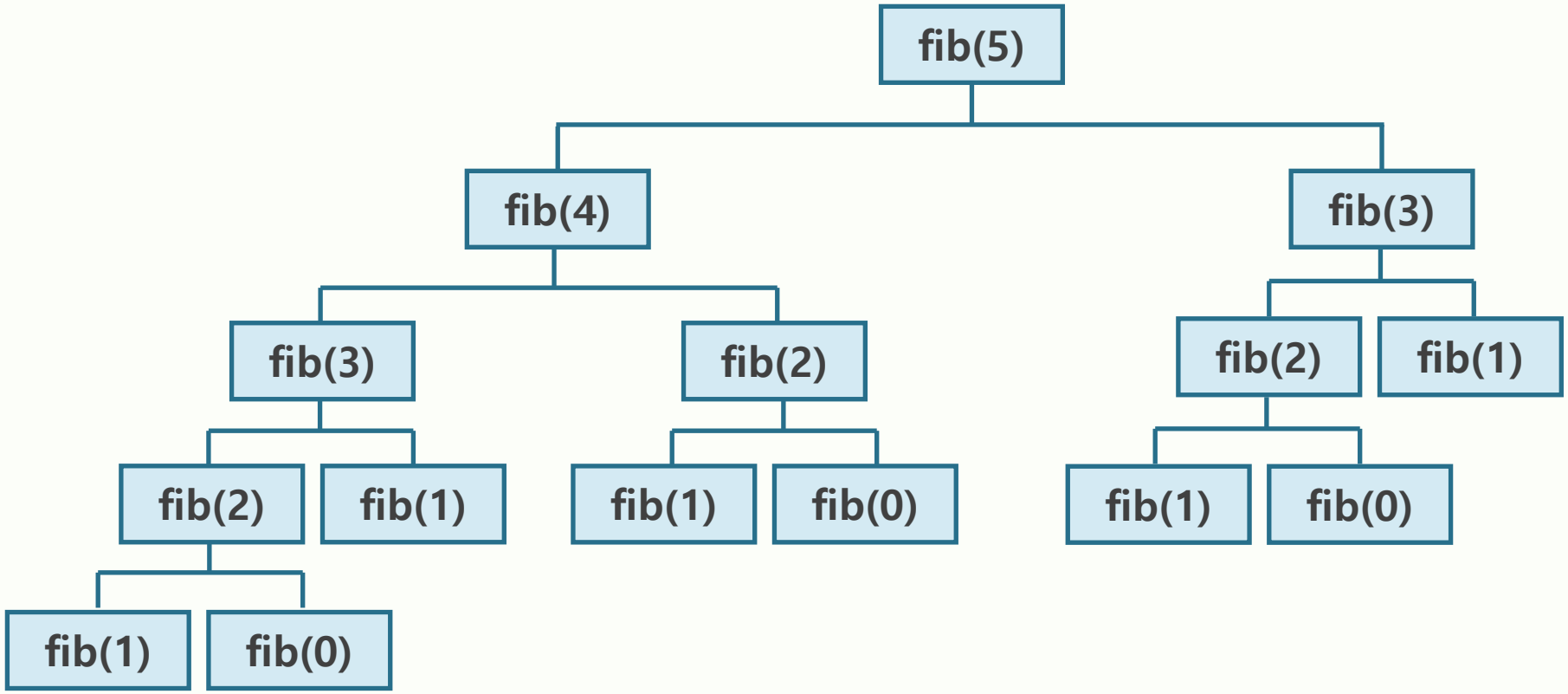
피보나치 함수

- Memoization
- 한 번 계산된 값을 기록해둔다.
- 중복 호출되었을때 새롭게 계산하지 않고
- 저장해둔 값을 가져와 사용한다.

Dynamic Programming

피보나치 함수

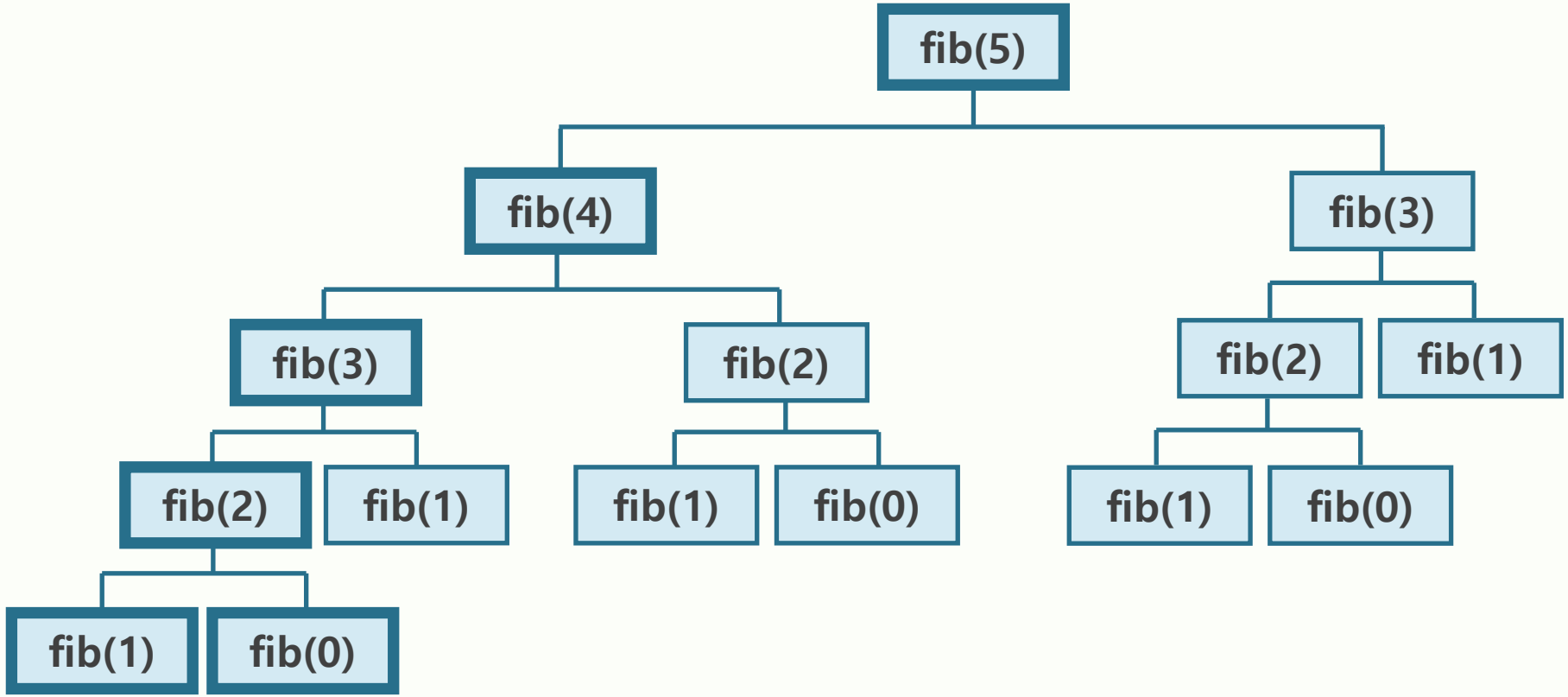
- F[i]: i번째 피보나치값



Dynamic Programming

피보나치 함수

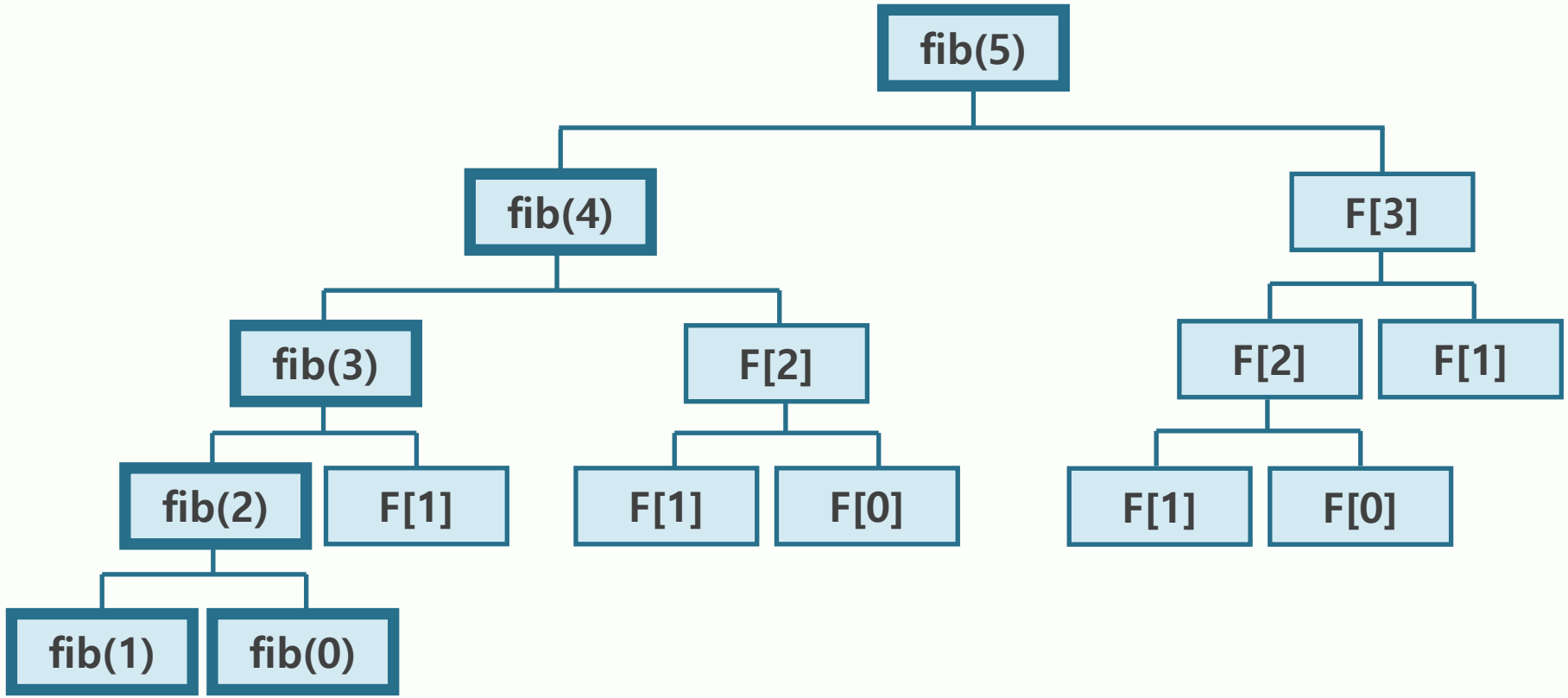
- F[i]: i번째 피보나치값



Dynamic Programming

피보나치 함수

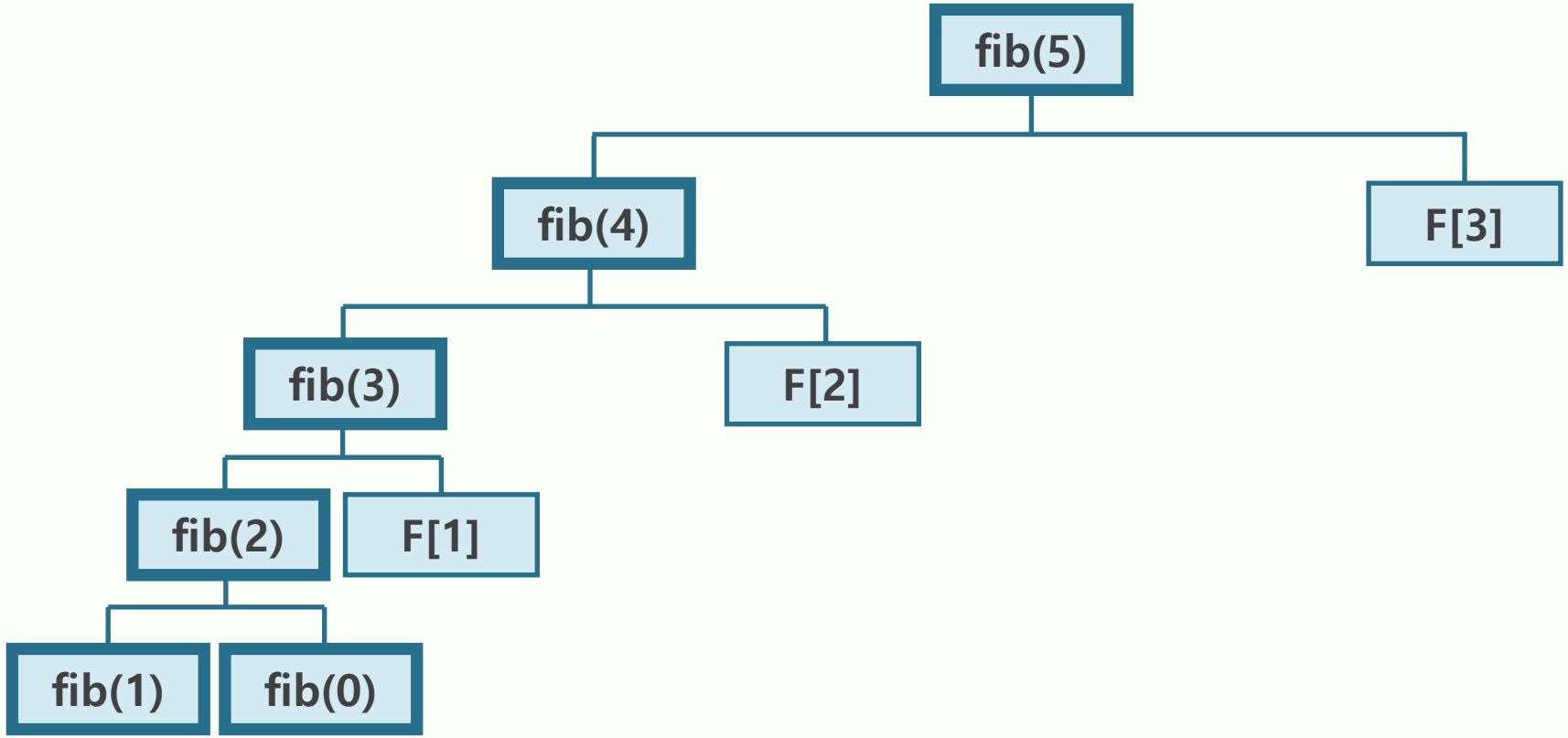
- F[i]: i번째 피보나치값



Dynamic Programming

피보나치 함수

- F[i]: i번째 피보나치값



Dynamic Programming

동적계획법이란

- 이미 계산된 부분 문제가 다시 발생하면
- 새롭게 계산하지 않고 이전의 계산값을 참조하여 이용한다.

Dynamic Programming

동적계획법이란

- 부분 문제를 다시 해결하는 데 필요한 시간을 절약
- 이전 계산값을 저장해둘 공간이 필요

Dynamic Programming

동적계획법이란

- 부분 문제를 다시 해결하는 데 필요한 시간을 절약
- 이전 계산값을 저장해둘 공간이 필요

- 시간과 메모리의 Trade-off

Dynamic Programming

Top-Down & Bottom-Up

- Top-Down과 Bottom-Up 형식으로 구현될 수 있다.

Dynamic Programming

Top-Down & Bottom-Up

- Top-Down
- 큰 문제에서 작은 부분문제를 재귀적으로 호출
- 계산된 값을 이용하여 큰 문제를 해결한다.

Dynamic Programming

Top-Down & Bottom-Up

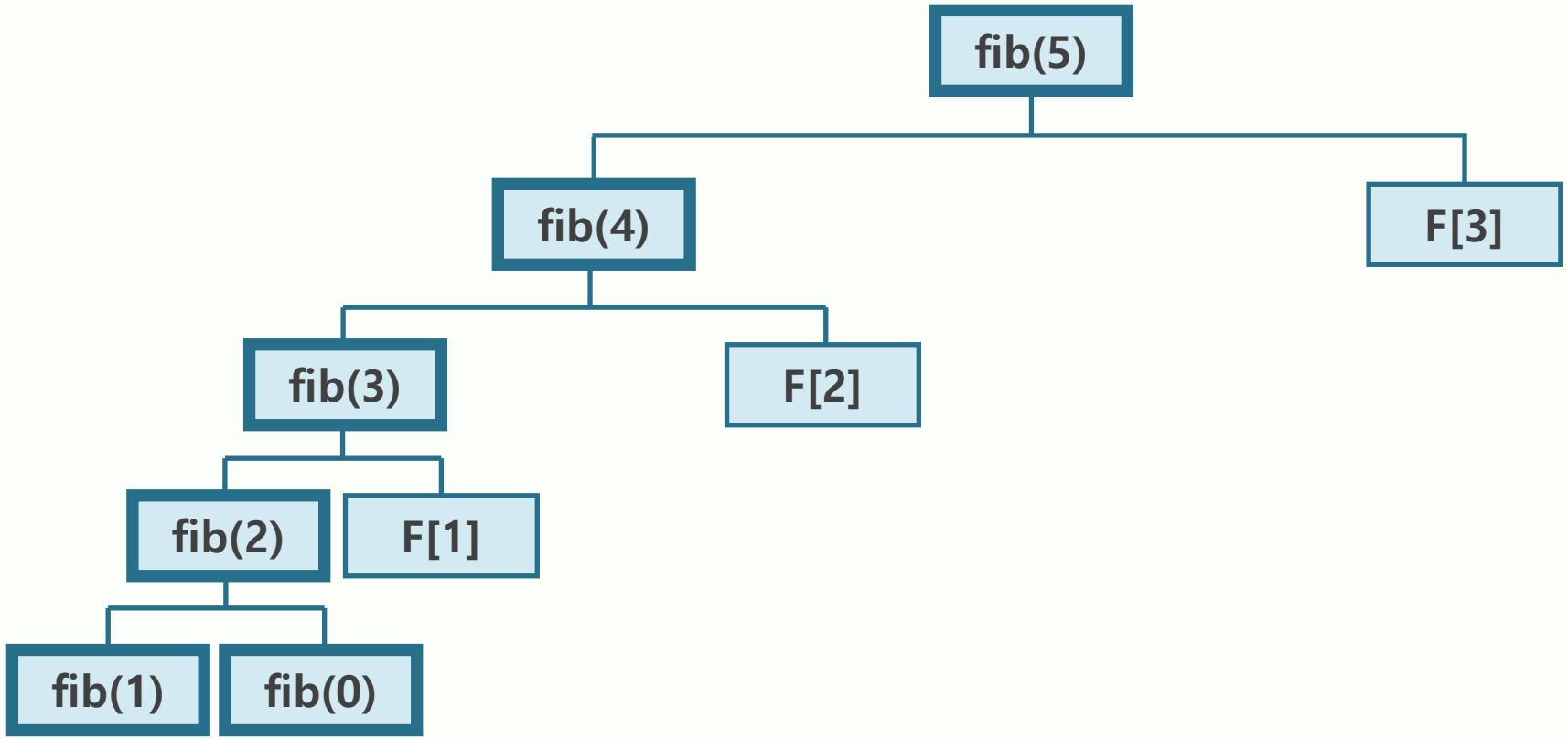
- Top-Down
- 큰 문제에서 작은 부분문제를 재귀적으로 호출
- 계산된 값을 이용하여 큰 문제를 해결한다.

- 부분문제의 중복을 피하기 위해 Memoizaion 기법을 함께 사용한다.

Dynamic Programming

Top-Down & Bottom-Up

- Top-Down



Dynamic Programming

Top-Down & Bottom-Up

- Top-Down

```
int F[MAX_FIB_NUM + 1] = {1, 1, };

int calc_fib(int x){
    if(F[x] != 0){
        return F[x];
    }
    return F[x] = calc_fib(x - 1) + calc_fib(x - 2);
}
```

Dynamic Programming

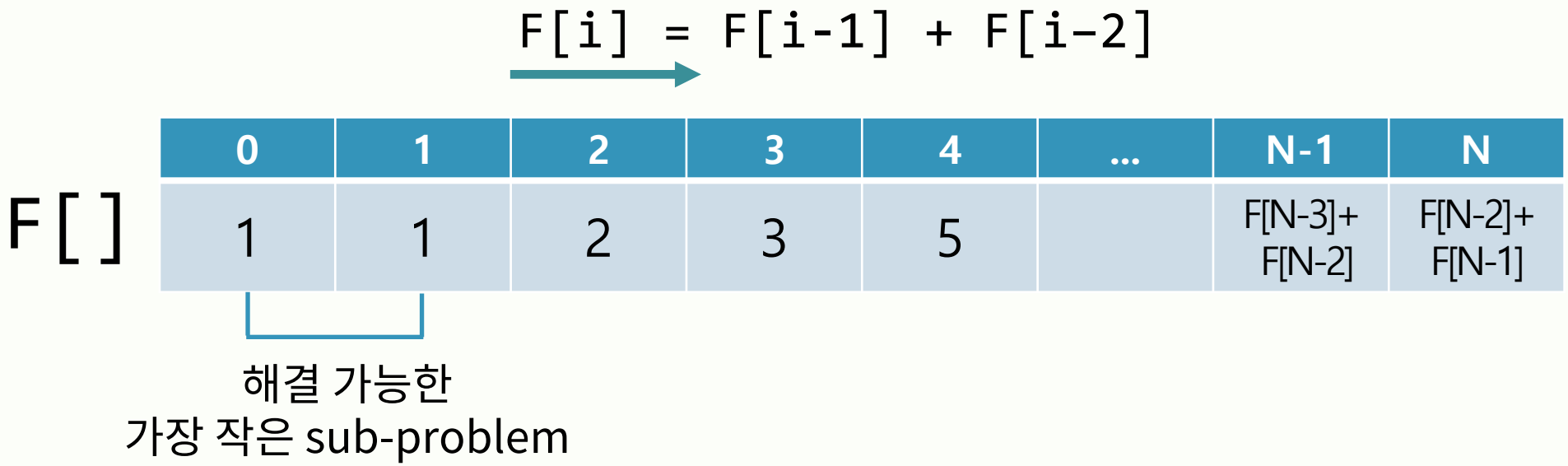
Top-Down & Bottom-Up

- Bottom-Up
- 작은 부분문제들을 미리 계산한다.
- 부분문제를 모아 큰 문제를 해결한다.
- 역시 배열을 이용해 값을 채워나간다.

Dynamic Programming

Top-Down & Bottom-Up

- Bottom-Up



Dynamic Programming

Top-Down & Bottom-Up

- Bottom-Up

```
int F[MAX_FIB_NUM + 1] = {1, 1, };

void calc_fib(){
    for(int i = 2; i <= MAX_FIB_NUM; i++){
        F[i] = F[i - 1] + F[i - 2];
    }
}
```

Dynamic Programming

Top-Down & Bottom-Up

- Top-Down
  - 일반적으로 재귀 함수를 통해 구현
  - 함수 호출에 대한 오버헤드

Dynamic Programming

Top-Down & Bottom-Up

- Top-Down
  - 일반적으로 재귀 함수를 통해 구현
  - 함수 호출에 대한 오버헤드

- Bottom-Up
  - 일반적으로 반복문을 통해 구현
  - 시간 및 메모리의 최적화

Dynamic Programming

Top-Down & Bottom-Up

- Bottom-Up
  - 큰 문제를 해결하기까지 어떤 sub-problem이 요구되는지 알 수 없음
  - 전체 문제를 계산하기 위해 모든 부분문제를 해결해야 한다.

Dynamic Programming

Top-Down & Bottom-Up

- Bottom-Up
  - 큰 문제를 해결하기까지 어떤 sub-problem이 요구되는지 알 수 없음
  - 전체 문제를 계산하기 위해 모든 부분문제를 해결해야 한다.

- Top-Down
  - 큰 문제에서 필요한 sub-problem만 호출
  - 필요한 부분만 계산하게 된다
  - 특정한 경우 Bottom-Up보다 빠르게 동작

동전 0

Problem: https://www.acmicpc.net/problem/11047

- N 종류의 동전
- 각 동전은 매우 많다.
- 동전을 적절히 사용해서 K원을 만들 때
- 필요한 동전 개수의 최소값

동전 0

Problem: https://www.acmicpc.net/problem/11047

- N 종류의 동전
- 각 동전은 매우 많다.
- 동전을 적절히 사용해서 K원을 만들 때
- 필요한 동전 개수의 최소값

- **$A_1 = 1$, $i \geq 2$인 경우에 A_i는 A_{i-1}의 배수**

동전 0

Problem: https://www.acmicpc.net/problem/11047

- $1 \leq N \leq 10$
- $1 \leq K \leq 100,000,000$

동전 0

Problem: https://www.acmicpc.net/problem/11047

- $A_1 = 1$, $i \geq 2$인 경우에 A_i는 A_{i-1}의 배수

동전 0

Problem: https://www.acmicpc.net/problem/11047

- $A_1 = 1$, $i \geq 2$인 경우에 A_i는 A_{i-1}의 배수
- A_i 원을 C개를 이용해 X원을 만든다면
- A_0 ~ A_{i-1}원을 이용해서는 C개보다 많은 동전이 필요하다.

동전 0

Problem: https://www.acmicpc.net/problem/11047

- $A_1 = 1$, $i \geq 2$인 경우에 A_i는 A_{i-1}의 배수
- A_i 원을 C개를 이용해 X원을 만든다면
- A_0 ~ A_{i-1}원을 이용해서는 C개보다 많은 동전이 필요하다.

- 큰 동전을 쓸 수 있는만큼 쓰는게 최적

동전 0

Problem: https://www.acmicpc.net/problem/11047

- C/C++: https://gist.github.com/Acka1357/e641ddcf96bf1848a0a14d773f7ef635

- JAVA: https://gist.github.com/Acka1357/d1b1dee94e467dc3555c27f530e00c98

동전 2

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

- N 종류의 동전
- 각 동전은 매우 많다.
- 동전을 적절히 사용해서 K원을 만들 때
- 필요한 동전 개수의 최소값

- K원을 만드는게 불가능하다면 -1을 출력한다.

동전 2

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

- $1 \le N \le 100$
- $1 \le K \le 10,000$

동전 2

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

- {1, 5, 12}원이 있을 때
- 15원을 만드는 경우를 생각해보자.

동전 2

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

- 동전 0과 같이 접근하면

동전 2

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

- 동전 0과 같이 접근하면

- 가장 큰 금액의 동전부터 먼저 사용한다.
- 동전을 적게 쓰기 위해서는 큰 동전을 쓰는 것이 유리하다.

동전 2

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

- 동전 0과 같이 접근하면

- 가장 큰 금액의 동전부터 먼저 사용한다.
- **동전을 적게 쓰기 위해서는 큰 동전을 쓰는 것이 유리하다?**

동전 2

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

- 가장 먼저 12원으로 만들 수 있는 최대 금액을 채운다.

- 12: (12 * 1)

동전 2

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

- 남은 금액을 그 다음으로 큰 5원을 이용해 채운다.

- 12: (12 * 1) + (5 * 0)

동전 2

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

- 남은 금액을 그 다음으로 큰 1원을 이용해 채운다.

- 15: (12 * 1) + (5 * 0) + (1 * 3)

동전 2

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

- 15: 12 + 1 + 1 + 1
- 동전 4개가 필요

동전 2

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

- 15: 5 + 5 + 5
- 동전 3개로도 가능하다.

동전 2

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

- 따라서 위와 같은 방법을 이용하려면
- 큰 동전을 하나 제거한 뒤, 남은 금액에 대해 같은 과정을 거친다.
- 위 동작을 반복한다.

동전 2

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

- 최적값이라는 것을 확신할 수 없기 때문에
- 결국 모든 조합을 다 해보게 된다.

동전 2

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

- 최적값이라는 것을 확신할 수 없기 때문에
- 결국 모든 조합을 다 해보게 된다.

- 100종류의 동전으로 K원을 만들 수 있는 총 경우의 수 …?

동전 2

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

- 동적 계획법을 적용해보자.

동전 2

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

- 준비물

동전 2

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

- 준비물

- 결과값을 저장해둘 테이블
- 점화식

동전 2

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

- D[i]: i원을 만들기 위해 필요한 동전의 최소개수

동전 2

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

- D[i]: i원을 만들기 위해 필요한 동전의 최소개수

- K원은 K보다 작은 금액에 새로운 동전을 추가해 만들 수 있다.
- 새로운 동전을 C[i]라고 하면
- D[K] = D[K – C[i]] + 1

동전 2

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

- $D[K] = Min_{i=0 \text{ to } N-1,\ K > C[i]}(D[K - C[i]]) + 1$

동전 2

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

- $D[K] = Min_{i=0 \text{ to } N-1,\ K > C[i]}(D[K - C[i]]) + 1$

- D[K]는 D[K – C[i]]라는 sub-problem으로 나누어 해결한다.
- D[0] = 0

동전 2

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

- 답: D[K]
  - K원을 만들기 위해 필요한 동전의 최소개수

동전 2

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

- C/C++: https://gist.github.com/Acka1357/50eeccfe772d36b4f8cab053b8348207

- JAVA: https://gist.github.com/Acka1357/24886962e241c79187b3855c72fdcf99

1, 2, 3 더하기

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

- 정수 N이 주어졌을 때
- N을 1, 2, 3의 합으로 나타내는 방법의 수
- 더하는 순서가 다르다면 다른 방법이 된다.

1, 2, 3 더하기

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

- $1 \le N \le 11$

1, 2, 3 더하기

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

- D[i] = i를 만드는는 방법의 수

1, 2, 3 더하기

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

- D[i] = i를 만드는는 방법의 수

- i를 만드는 방법:
- (i – 1을 만드는 모든 식) + 1
- (i – 2을 만드는 모든 식) + 2
- (i – 3을 만드는 모든 식) + 3

1, 2, 3 더하기

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

- D[N] = D[N – 1] + D[N – 2] + D[N – 3]

1, 2, 3 더하기

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

- D[N] = D[N – 1] + D[N – 2] + D[N – 3]
- D[N]: $\sum_{i=1}^{3} D[N-i]$

1, 2, 3 더하기

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

- D[0] = 1
- 동전을 하나도 안쓰는것도 하나의 경우가 된다.

1, 2, 3 더하기

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

- 답: D[N]
  - N원을 만드는 방법의 수

1, 2, 3 더하기

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

- C/C++: https://gist.github.com/Acka1357/f36930c2508160913814 91b730a77bdb

- JAVA: https://gist.github.com/Acka1357/ae01cd63c3ff2841ef024093ef0fa1cc

이친수

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

- 0과 1로 이루어진 이진수 중
- 0으로 시작하지 않으면서
- 1이 두 번 연속으로 나타나지 않는
- 수를 이친수라고 한다.

- N자리 이친수의 개수

이친수

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

- 0과 1로 이루어진 이진수 중
- 0으로 시작하지 않으면서
- 1이 두 번 연속으로 나타나지 않는
- 수를 이친수라고 한다.

- N자리 이친수의 개수

- $1 \leq N \leq 90$

이친수

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

- D[i]: 길이가 i인 이친수의 개수

이친수

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

- D[i]: 길이가 i인 이친수의 개수

- i번째 수를 0으로 채우는 경우
- D[i – 1]의 모든 경우에 대해서 가능하다

이친수

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

- D[i]: 길이가 i인 이친수의 개수

- i번째 수를 0으로 채우는 경우
- D[i – 1]의 모든 경우에 대해서 가능하다

- i번째 수를 1로 채우는 경우
- D[i – 1]의 중 끝이 0인 경우에만 가능하다.
- = D[i – 2]의 모든 경우에 i – 1번째 수를 0으로 채우는경우

이친수

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

- D[i] = D[i – 1] + D[i – 2]

이친수

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

- D[i] = D[i – 1] + D[i – 2]

- D[1] = 1: {1}
- D[2] = 1: {10}

이친수

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

- D[i] = D[i – 1] + D[i – 2]

- D[1] = D[2] = 1인 피보나치
- D[N]: (N – 1)번째 피보나치 수

이친수

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

- 답: D[N]
  - 길이가 N인 이친수의 개수

이친수

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

- D[i][j]: 길이가 i이면서 j로 끝나는 이친수의 개수

이친수

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

- D[i][0]: D[i – 1][0] + D[i – 1][1]
- D[i][1]: D[i – 1][0]

이친수

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

- 답: D[N][0] + D[N][1]
  - 길이가 N인 이친수 개수의 총합

이친수

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

- C/C++: https://gist.github.com/Acka1357/34efaec4aa29f5d3b39f4a8a30a38a25

- JAVA: https://gist.github.com/Acka1357/331176be7f8b1b07364c0fad862f31e3

쉬운 계단 수

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

- 계단 수: 인접한 모든 자리수의 차이가 1이 나는 수
- 길이가 N인 계단 수의 개수

쉬운 계단 수

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

- 계단 수: 인접한 모든 자리수의 차이가 1이 나는 수
- 길이가 N인 계단 수의 개수

- $1 \le N \le 100$

쉬운 계단 수

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

- D[i][j]: 길이가 i이면서 j로 끝나는 계단 수의 개수

쉬운 계단 수

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

- D[i][j]: 길이가 i이면서 j로 끝나는 계단 수의 개수

- D[i][0]: D[i – 1][1]
- D[i][1]: D[i – 1][0] + D[i – 1][2]
- D[i][2]: D[i – 1][1] + D[i – 1][3]
- …
- D[i][9]: D[i – 1][8]

쉬운 계단 수

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

- D[i][j]: D[i – 1][j – 1] + D[i – 1][j + 1]

쉬운 계단 수

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

- D[i][j]: D[i – 1][j – 1] + D[i – 1][j + 1]

- D[1][j] = 1 (j != 0)

쉬운 계단 수

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

- 답: $\sum_{i=0}^{9} D[N][i]$
  - 길이가 N인 계단수 개수의 총합

쉬운 계단 수

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

- C/C++: https://gist.github.com/Acka1357/a73747baed7ab897866a3d4a814cb123

- JAVA: https://gist.github.com/Acka1357/248b3650dbf82fdae30ccb0011a413b0

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- N개의 집을 빨강, 초록, 파랑 중 하나로 칠한다.
- 각 집을 어떤 색으로 칠하느냐에 따라 비용이 다르다.
- 이웃한 집은 색이 같아서는 안된다.

- 모든 집을 칠할 때 드는 최소 비용

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- N개의 집을 빨강, 초록, 파랑 중 하나로 칠한다.
- 각 집을 어떤 색으로 칠하느냐에 따라 비용이 다르다.
- 이웃한 집은 색이 같아서는 안된다.

- 모든 집을 칠할 때 드는 최소 비용

- $1 \le N \le 1,000$

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- i번 집을 칠할때 필요한 정보는
- i-1번 집을 어떤 색으로 칠했고
- 그 상황의 최소비용

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- 1 ~ i - 1번 집까지 칠해져 있을 때
- i번 집을 빨간색으로 칠하는 최소 비용
  - (i – 1)번 집은 빨간색이 아니어야 한다.
  - (i – 1)번 집을 초록/파란색으로 칠했을 때의 최소 비용
    + i번 집을 빨간색으로 칠할 때의 비용

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- 1 ~ i - 1번 집까지 칠해져 있을 때
- i번 집을 빨간색으로 칠하는 최소 비용
  - (i – 1)번 집은 빨간색이 아니어야 한다.
  - (i – 1)번 집을 초록/파란색으로 칠했을 때의 최소 비용 + i번 집을 빨간색으로 칠할 때의 비용

- i번 집을 초록색으로 칠하는 최소 비용
- i번 집을 파란색으로 칠하는 최소 비용

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- D[i][j]: 1 ~ i번 집이 칠해져있고,
  i번 집을 j색으로 칠했을 때의 최소비용
  - j = 0: 빨간색
  - j = 1: 초록색
  - j = 2: 파란색

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- D[i][j]: $\min_{k = 0 \text{ to } 2,\ k != j}(D[i - 1][k]) + cost[i][j]$

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- D[i][j]: $\min_{k = 0 \text{ to } 2,\ k \mathrel{!}= j}(D[i - 1][k]) + cost[i][j]$

- D[i][0] = min(D[i – 1][1], D[i – 1][2]) + cost[i][0]
- D[i][1] = min(D[i – 1][0], D[i – 1][2]) + cost[i][1]
- D[i][2] = min(D[i – 1][0], D[i – 1][1]) + cost[i][2]

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- 답: $\min_{i = 0 \text{ to } 2}(D[N][i])$
  - N번 집까지 모두 칠한 비용 중 최소비용

RGB거리

Problem: https://www.acmicpc.net/problem/1149

- C/C++: https://gist.github.com/Acka1357/a25f6baba2d5e298f8e8c0a39f777150

- JAVA: https://gist.github.com/Acka1357/4f32f683035675f736528e170275a89d

합분해

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

- 0부터 N까지의 정수를 더해서 N을 만드는 경우의 수
- 한 숫자를 여러번 사용 할 수 있다.
- K개 숫자를 사용해야한다.
- 덧셈의 순서가 다르면 다른 경우가 된다.

- 답을 1,000,000,000으로 나눈 나머지를 출력한다.

합분해

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

- 0부터 N까지의 정수를 더해서 N을 만드는 경우의 수
- 한 숫자를 여러번 사용 할 수 있다.
- K개 숫자를 사용해야한다.
- 덧셈의 순서가 다르면 다른 경우가 된다.

- $1 \leq N \leq 200$
- $1 \leq K \leq 200$

합분해

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

- 1, 2, 3 더하기와 비슷한 문제.
- K개 숫자를 사용하는 조건이 추가됨

합분해

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

- K개의 숫자를 사용한다
- 몇개의 숫자를 사용했는지에 대한 상태가 필요하다.

합분해

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

- D[i][j]: i개의 숫자를 더해서 j가 되는 경우의 수

합분해

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

- D[i][j]: i개의 숫자를 더해서 j가 되는 경우의 수

- D[0][0] = 1
- D[i][j] = $\sum_{k=0}^{j} D[i-1][j-k]$

합분해

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

- 답: D[K][N]:
  - K개의 숫자를 더해서 N이 되는 경우의 수
  - 연산과정에서 overflow가 일어나지 않도록 주의하며 나머지 연산

합분해

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

- C/C++: https://gist.github.com/Acka1357/4549422afc488a753e26e1887652c81f

- JAVA: https://gist.github.com/Acka1357/0edf22782939ad042677495de4817bad