DP1

1/21

SCCC 정주영 @greimul

출석체크 겸 간단한 문제

2747번
피보나치 수

DP?

~~mic Programming ->~~

~~• Dyna~~

~~적인 프로~~

단어의 뜻은 실제 DP하고는 아무 상관이 없다.

DP?

어떠한 문제를 풀 때, 작은 문제로 쪼개서, 작은 문제를 이용하여 큰문제를 푸는 방법.

한번 계산했던 결과는 저장을 해서, 계산의 반복을 피한다.

-> 처리속도 up!

DP 문제가 가지는 두 가지 특성

1. Overlapping Subproblems

- 문제를 작은 문제(Subproblem)로 쪼갤 수 있다.
- 작은 문제들의 해를 반복적으로 요구한다.

2. Optimal Structure

- 작은 문제의 솔루션을 가지고 큰 문제의 솔루션을 구할 수 있다.

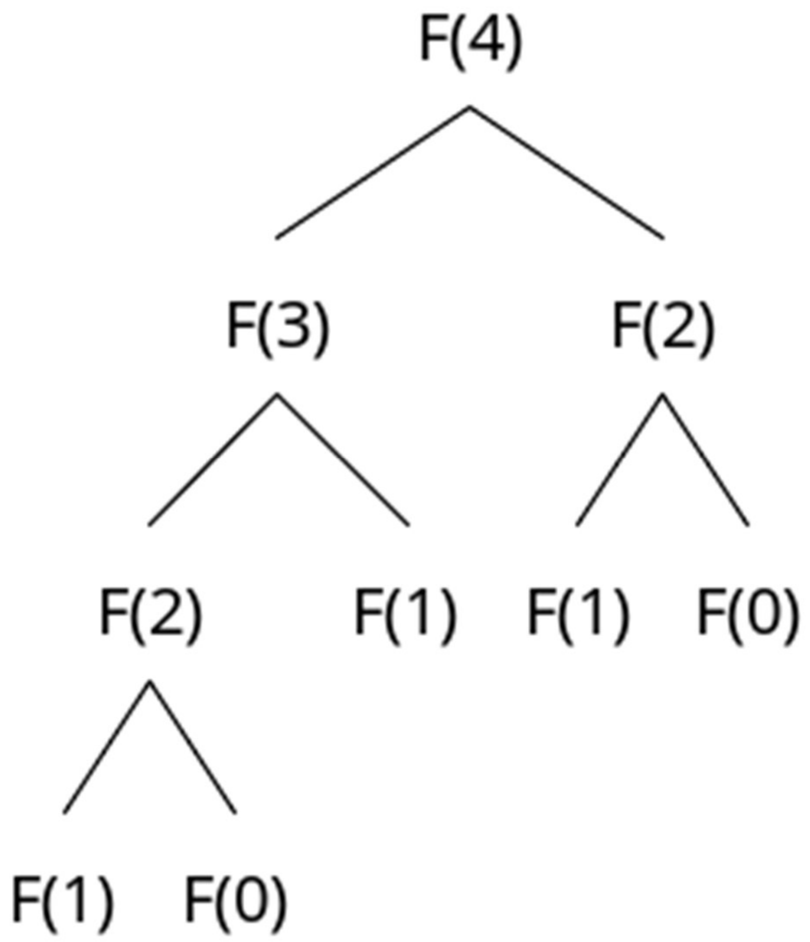
Fibonacci Number

$F(0) = 0$

$F(1) = 1$

$F(n) = F(n-1)+F(n-2)$

F(4)를
구하자



DP 구현 방법

1. Top-Down

2. Bottom-Up

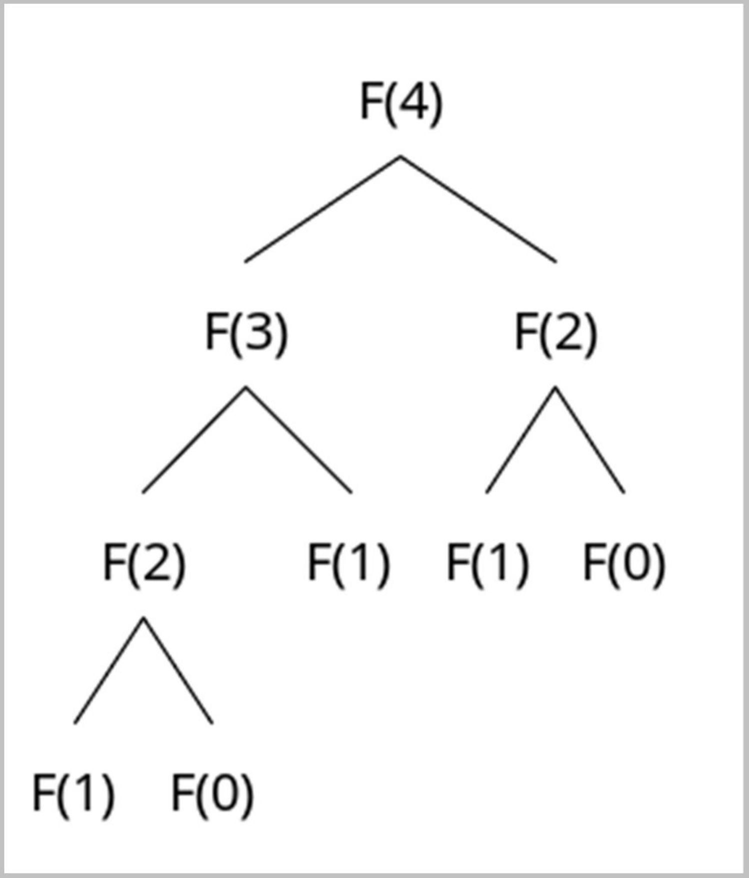
+ 메모이제이션 (Memoization)

반복 계산이 필요한 경우, 계산한 값을 저장 해 두어, 반복횟수를 줄인다.

Top-Down

- 재귀함수를 사용
- 필요한 작은 문제들을 그때그때 불러준다.
- 이미 계산되어 있는 값들은 바로 리턴.

```
int FTD(int n) {
    if (fibo[n] == -1) {
        fibo[n] = FTD(n - 1) + FTD(n - 2);
    }
    return fibo[n];
}
```



메모이제이션을 안했다면?

- 중복된 계산 때문에

지수시간 시간 복잡도가 되어서 시간초과!

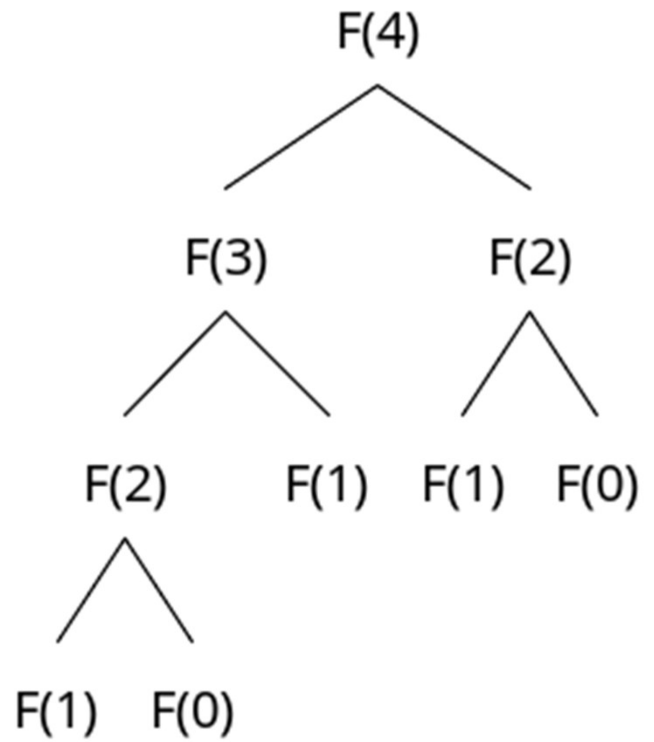
```
int FTD(int n) {
    if (n == 1) {
        return 1;
    }
    if (n == 0) {
        return 0;
    }
    return FTD(n - 1) + FTD(n - 2);
}
```

문제 번호	결과
2747	시간 초과

Bottom-Up

- 반복문을 사용
- 작은 문제부터 차근차근 해를 구해준다.

```
int FBU(int n) {
    for (int i = 2; i <= n; i++) {
        fibo[i] = fibo[i - 1] + fibo[i - 2];
    }
    return  fibo[n];
}
```



9461번
파도반 수열

파도반 수열

- 2747번 피보나치 수 문제와 점화식만 다르다.
- Top-Down, Bottom-Up 중 편한 것을 선택하여 구현하면 된다.

- F(0) = 0, F(1) = 1, F(2) =1, F(3) = 1
- F(n) = F(n-1) + F(n-5)

Top-Down vs Bottom-Up

Top-Down 방식은 재귀함수를 통해 구현되므로, 함수 호출에 대한 메모리 소모가 생긴다. Bottom-Up은 반복문으로 구현 되므로, 비교적 쉽게 시간 및 메모리 최적화를 할 수 있다.

Bottom-Up 방식은 큰 문제 해결까지 어떤 부분문제가 필요한지 알 수 없으므로, 모든 부분문제를 해결해야 한다. 하지만 Top-Down 방식은 필요한 부분문제만 호출하므로, 특정한 경우에는, Top-Down 이 더 빠르게 작동할 수 있다.

하지만 (DP1 수준의)일반적인 경우에는 유의미한 차이는 없기때문에 생각하기 쉬운쪽으로 구현을 하자.

결론

- DP를 풀기위해서는?

- 1. DP테이블 정의 (문제 정의)
- 2. 점화식을 세운다. (작은 문제 -> 큰 문제)
- 3. 구현 (Top-Down or Bottom-Up)
- Accept!

- 설명은 간단하지만 말처럼 쉽지는 않다.
- 답은 “다다익선”. 실력 상승을 위해서는 많이 풀어보면 된다.

1463번
1로 만들기

1로 만들기

- DP 테이블 정의
- D[n] = n를 1로 만들기 위한 최소 횟수

- 점화식
- $\mathrm{D[n]} = \min(D[n-1],\ D[\mathrm{n/2}], D[\mathrm{n/3}]) + 1$

9095번 1,2,3 더하기

1,2,3 더하기

- DP 테이블 정의
- D[n] = n을 나타낼 수 있는 방법의 수

- 점화식
- $D[n] = D[n-1] + D[n-2] + D[n-3]$

참고: 테스트 케이스가 있는 문제

- 테스트 케이스가 존재하는 문제의 경우 한번의 실행에서 여러 케이스를 답 해야 한다.
- 만약 매 테스트마다 F(n) 을 계산하게 되면 시간이 너무 오래 걸리게 된다.

- 그러므로 DP배열을 최대 제한N까지 미리 채워 둔 후 답을 하면 좀 더 시간을 단축 할 수 있다.

문제 번호	결과	메모리	시간
15988	맞았습니다!!	9800 KB	20 ms
15988	맞았습니다!!	9800 KB	792 ms

참고: MOD 연산

- DP 문제를 풀때

  방법의 수를 1,000,000,009로 나눈 나머지를 출력한다.
- 와 같은 조건이 붙어있는 경우가 많다.

- 모든 계산을 마친 후에 나머지를 계산해준다면 편하겠지만, 이런 문제들은 계산도중에 값이 너무 커져 오버플로우가 나는 경우 이므로, 중간 중간에 나머지를 계산 해야 한다.

- 그래서 이런 문제를 풀기위해서 MOD 연산을 알아야 한다.

MOD 연산

- A+B 를 X로 나눈 나머지 = $(A+B)\%X$= $\big((A\%X)+(B\%X)\big)\%X$
- A-B를 X로 나눈 나머지 = $(A-B)\%X$ = $\big((A\%X)-(B\%X)\big)\%X$
- A*B 를 X로 나눈 나머지 = $(A*B)\%X$ = $\big((A\%X)*(B\%X)\big)\%X$

- ☆A/B를 X로 나눈 나머지
- 나눗셈의 MOD 연산은 곱셈 역원(Multiplicative inverse)을 사용한다.
- 곱셈 역원 = B^{MOD}
- (A/B)%X = $((A\%X)*(B^{MOD}\quad\%MOD))\%MOD$ MOD가 큰 경우가 대부분이기 때문에 따로 최적화 필수

팁
나누는 수 X는
#define MOD X
로 따로 상수로 빼 두면 편하다.

참고 문제

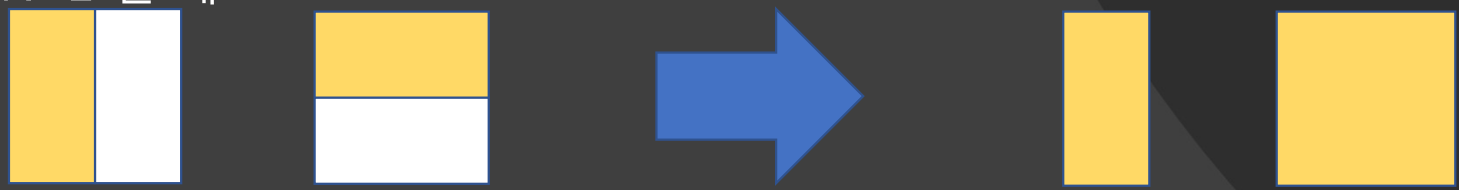
15988번 1,2,3 더하기 3

11726번 2xn 타일링

2xn 타일링

- N= 1 일 때

- N=2 일 때



2xn 타일링

길이가 1인 선분과 2인 선분을 가지고 N인 선분을 채워라

1과 2만으로 N을 만드는 방법의 수

2xn 타일링

- DP 테이블 정의
- D[n] = n을 나타낼 수 있는 방법의 수

- 점화식
- $D[n] = D[n-1] + D[n-2]$

10844번 쉬운 계단 수

쉬운 계단 수

- Dp 테이블 정의
- D[i][j] = 길이가 i인 j로 끝나는 수

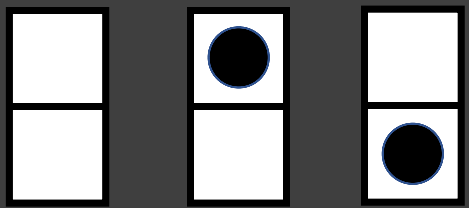
- 점화식
- $D[i][j] = D[i-1][j+1] + D[i-1][j-1]$

- 예외 처리 필요

1309번 동물원

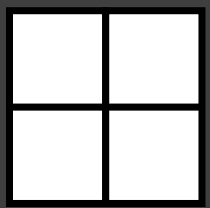
동물원

- N=1



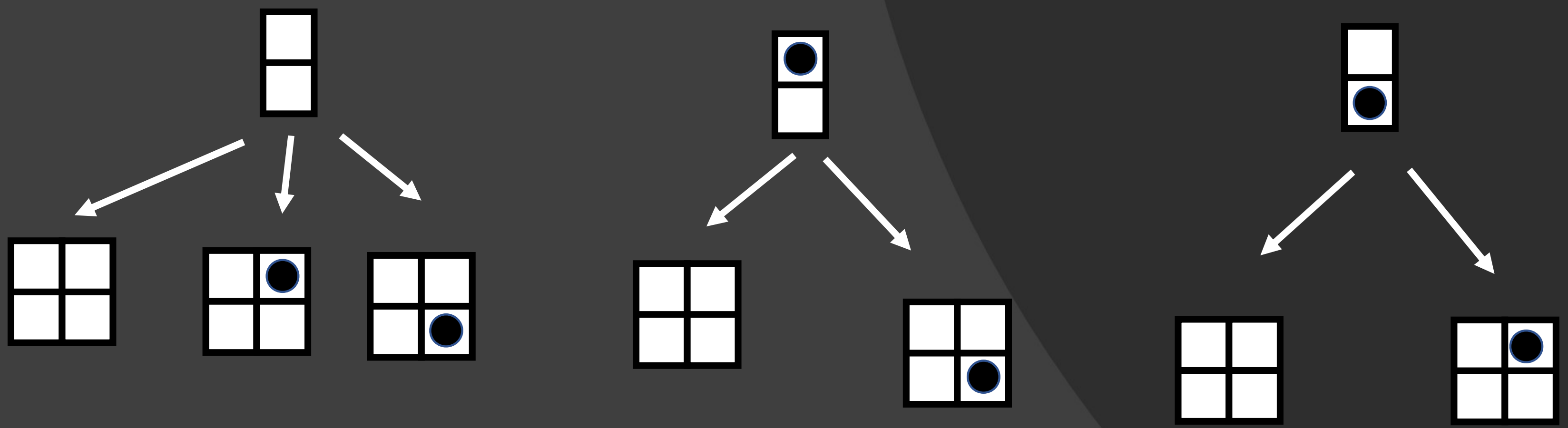
세가지 상태가 존재!

- N=2

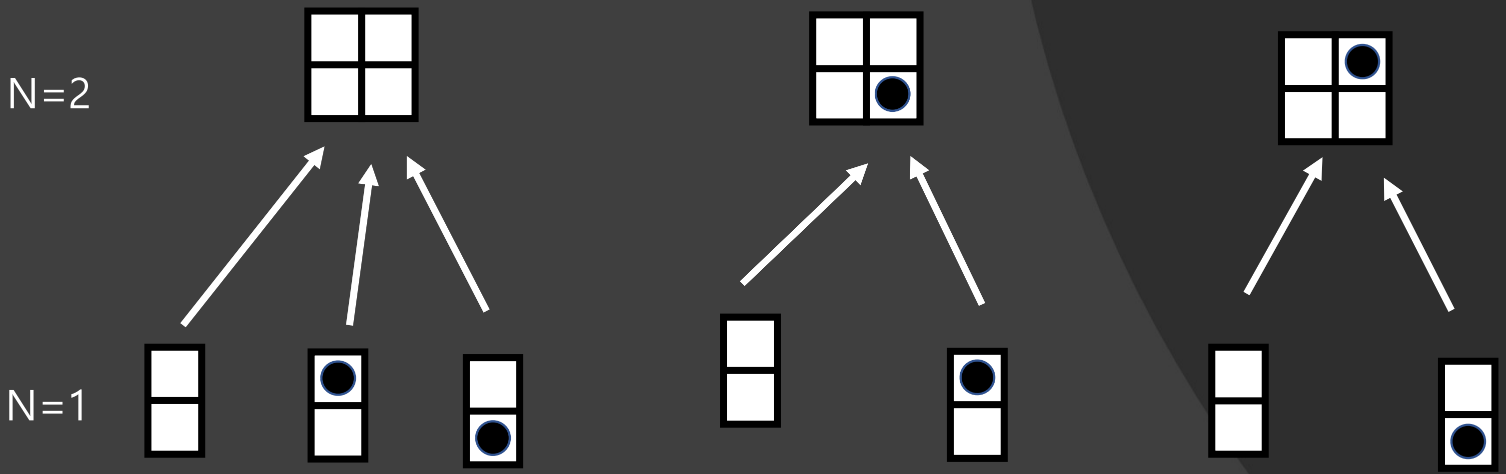


앞줄의 상태에 따라서 놓을 수 있는 위치가 달라짐

동물원



동물원



동물원

- Dp 테이블 정의
- D[i][j] = i번째 줄에서 j 상태일때의 가지 수

- 점화식
- $D[i][0] = D[i-1][0] + D[i-1][1] + D[i-1][2]$
- $D[i][1] = D[i-1][0] + D[i-1][2]$
- $D[i][2] = D[i-1][0] + D[i-1][1]$

문제에서 경로를 요구하는 경우

- 직전 값을 저장하는 배열을 따로 만들어 준 후,
- DP값이 갱신될 때 직전 값을 같이 갱신 한다.
- 이 값들을 가지고 경로를 재구축

관련 문제

2618번 경찰차

연습 문제들을
풀어봅시다.