

2021 여름학기 동국대학교 SW역량강화캠프

16일차. 다이나믹 프로그래밍 1

● 다이나믹 프로그래밍(DP)

- ▶ 값의 재사용 (중복되는 계산은 한 번만)
- ▶ 큰 문제를 작은 문제들로 나누어 해결하는 방법
- ▶ 동일한 형태의 문제들끼리의 관계식을 찾는 것이 핵심

● 코드의 실행시간을 최적화하는 3가지 방법

- ▶ 1. 더 빠르게 처리할 수 있는 과정이 있는가?
- ▶ 2. 불필요한 정보를 구하기 위해 시간을 낭비하고 있는가?
- ▶ 3. 이미 알고 있는 정보를 다시 구하기 위해 시간을 낭비하고 있는가?

● $2 \times n$ 타일링 (4067)

문제

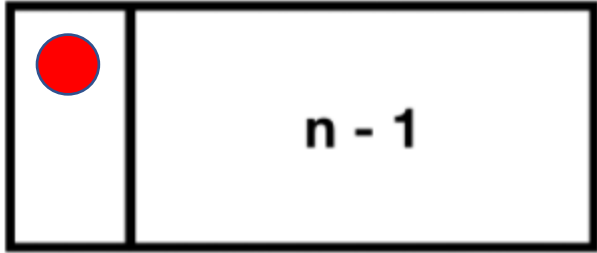
$2 \times n$ 크기의 직사각형을 1×2 , 2×1 타일로 채우는 방법의 수를 구해보자.

입력

첫째 줄에 n 이 주어진다. ($1 \leq n \leq 1,000$)

출력

첫째 줄에 문제의 답을 10,007로 나눈 나머지를 출력한다.



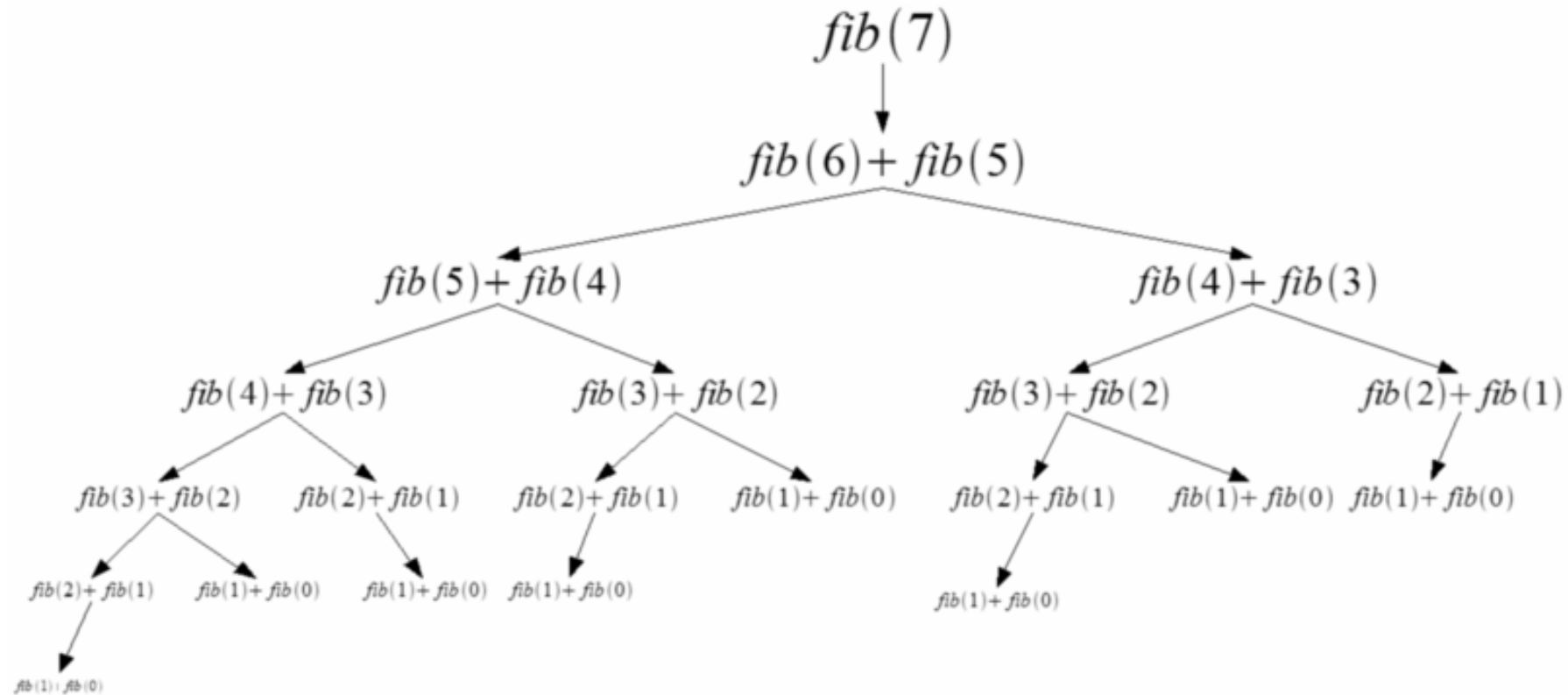
- ▶ 맨 왼쪽 위 칸을 채운 타일에 따라 경우의 수를 나누자
- ▶ 세로로 채울 경우 나머지 칸은 $2 \times (n-1)$ 의 직사각형
- ▶ 가로로 채울 경우 나머지 칸은 $2 \times (n-2)$ 의 직사각형

$dp(n) = 2 \times n$ 의 직사각형을 채우는 경우의 수

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

$$dp(1) = 1, dp(2) = 2$$

```
public static int dp(int x) {  
    if (x == 1)  
        return 1;  
    if (x == 2)  
        return 2;  
    return (dp(x - 1) + dp(x - 2)) % 10007;  
}
```



```
public static int dp(int x) {  
    if (dt[x] != 0)  
        return dt[x];  
    if (x == 1)  
        return 1;  
    if (x == 2)  
        return 2;  
    return dt[x] = (dp(x - 1) + dp(x - 2)) % 10007;  
}
```

- ▶ $dp(x)$ 는 값이 x 에 따라서만 변하는 함수
- ▶ $dp(x)$ 를 배열에 저장해두면 값을 재활용할 수 있다.
- ▶ 메모이제이션 기법 (하향식 DP)

- ▶ 다른 해결 방법
- ▶ $dp(x) = dp(x-1) + dp(x-2)$
- ▶ $x=1$ 부터 $x=N$ 까지 dp 값을 구하는 반복문을 작성

```
dp[1] = 1;  
dp[2] = 2;  
for(int i=3; i<=N; i++) {  
    dp[i] = (dp[i-1] + dp[i-2])%10007;  
}  
System.out.println(dp[N]);
```

- ▶ 1부터 N까지 반복문을 돌리면 $dp[i]$ 를 구할 때, $dp[i-1]$ 와 $dp[i-2]$ 가 구해져 있다.
- ▶ 상향식 DP

● 3이하로 분할하기 (4069)

| 문제

정수 n 이 주어졌을 때, n 을 3이하의 수(1, 2, 3)들의 합으로 표현할 수 있는 가지수를 구해보자.

예를 들어, 3은 아래와 같이 4가지 방법으로 표현할 수 있다.

$$\begin{aligned} 3 &= 1 + 1 + 1 \\ &= 1 + 2 \\ &= 2 + 1 \\ &= 3 \end{aligned}$$

| 입력

첫째 줄에 테스트케이스의 개수 T 가 주어진다.

둘째 줄부터 $T + 1$ 번째 줄까지 1, 2, 3의 합으로 표현할 정수 n 이 주어진다. ($1 \leq n \leq 11$)

| 출력

첫째 줄부터 T 번째 줄까지 각각의 테스트케이스에 대한 답을 출력한다.

- ▶ DP를 이용한 풀이
- ▶ N을 3 이하의 수로 분할하는 방법의 수 = $dp(N)$
- ▶ $dp(x) = dp(x-1) + dp(x-2) + dp(x-3)$
- ▶ dp 값은 테스트케이스에 상관없다. → 입력 받기 전 미리 전처리

```
dp[1] = 1;
dp[2] = 2;
dp[3] = 4;
for(int i=4; i<=11; i++) {
    dp[i] = (dp[i-1] + dp[i-2] + dp[i-3]);
}

while(T-- > 0) {
    int N = Integer.parseInt(br.readLine());
    sb.append(dp[N]+"\\n");
}
System.out.println(sb.toString());
```



● 수의 분할 (4070)

문제

어떤 수 n 을 주어진 수들의 합으로 표현할 수 있는 방법의 수를 세어보자.

예를 들면 3을 1과 2로 분할하고자 한다면 아래와 같이 3가지 방법이 있다.

$$3 = 1 + 2 = 2 + 1 = 1 + 1 + 1$$

입력

첫째 줄에 주어질 수의 개수 x 와 표현할 수 n 이 주어진다. ($1 \leq x \leq 100$), ($1 \leq n \leq 10^6$)

둘째 줄에 x 개의 수들(x_i)이 주어진다. ($1 \leq x_i \leq 10^6$)

출력

표현할 수 n 을 x 개의 주어진 수들의 합으로 표현할 수 있는 방법의 수를 $10^9 + 7$ 로 나눈 나머지를 출력한다.



- ▶ DP를 이용한 풀이
- ▶ N을 주어진 수들의 합으로 표현하는 경우의 수 = $dp(N)$
- ▶ $dp(x) = dp(x - arr[0]) + dp(x - arr[1]) + \dots$
- ▶ 10을 2,3,5로 나타내는 방법의 수 = $dp(10-2) + dp(10-3) + dp(10-5)$

```
dp[0] = 1;
for (int i = 1; i <= N; i++) {
    dp[i] = 0;
    for (int j = 0; j < X; j++) {
        if (i - arr[j] >= 0) {
            dp[i] += dp[i - arr[j]];
            dp[i] %= mod;
        }
    }
}
System.out.println(dp[N]);
```

● 길 찾기 (4075)

| 문제

북극곰 율리는 $N \times N$ 격자미로에 갇혀 있다. 미로의 시작점은 가장 왼쪽 위 점이고, 미로의 도착점은 가장 오른쪽 아래 점이다. 미로는 길과 벽으로 이루어져 있다. 길은 .으로 표시되고, 벽은 *로 표시된다. 길은 북극곰 율리가 지나갈 수 있지만 벽은 지나갈 수 없고, 북극곰 율리는 오른쪽이나 아래쪽으로만 이동할 수 있다. 북극곰 율리가 미로의 시작점에서 출발해 도착점까지 갈 때 가능한 경로의 수를 구해보자.

| 입력

첫째 줄에 격자의 가로, 세로의 길이 N 이 주어진다. ($1 \leq N \leq 1000$)

| 출력

첫째 줄에 북극곰 율리가 찾을 수 있는 경로의 수를 $10^9 + 7$ 로 나눈 나머지를 출력한다.

- ▶ DP를 이용한 풀이
- ▶ (1,1)에서 (x,y)까지 길을 따라 이동하는 방법의 수 = $dp(x,y)$
- ▶ 오른쪽 또는 아래쪽으로만 이동할 수 있다.
- ▶ $dp(x,y) = dp(x-1, y) + dp(x, y-1) \leftarrow (x,y)$ 가 빈 칸일 경우

```
if (arr[1][1] == 1)
    dp[1][1] = 1;

for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        if (i == 1 && j == 1) continue;
        if (arr[i][j] == 0) continue;
        dp[i][j] = (dp[i - 1][j] + dp[i][j - 1]) % mod;
    }
}

System.out.println(dp[N][N]);
```

● 용돈 받는 율리 (4094)

| 문제

용돈을 다 쓴 율리는 돈을 벌기 위해 N 일동안 부모님의 가게를 도우려고 한다.

율리가 일하는 가게에서는 설거지하기, 요리하기, 마당쓸기 세 가지의 일이 있다.

율리는 지루한 일을 굉장히 싫어하기 때문에, 이틀연속 같은 일을 하는 것은 불가능하다.

i 일에 설거지하기, 요리하기, 마당쓸기에 대한 용돈이 주어질 때, 벌 수 있는 돈의 최대를 구하자.

| 입력

첫째 줄에 율리가 일하는 날 N 이 주어진다. ($1 \leq N \leq 10^5$)

둘째 줄부터 N 개의 줄에 설거지, 요리, 마당쓸기를 할때의 용돈 x_i, y_i, z_i 들이 주어진다. ($1 \leq x_i, y_i, z_i \leq 10^4$)

- ▶ 출력: N일 동안 일해서 받을 수 있는 용돈의 최대값
- ▶ $dp(x)$ = 첫 x일 동안 일해서 받을 수 있는 용돈의 최대값
- ▶ $dp(x)$ 와 $dp(x-1)$ 의 관계식은?
- ▶ x일자에 어느 것을 선택했는가에 따라 $x-1$ 일에 선택할 수 있는 값이 달라짐

- ▶ $dp(x, i)$ = 첫 x 일 동안 일해서 받을 수 있는 용돈의 최대값, x 일자에는 작업 i 를 진행
- ▶ $dp(x, i) = \max(dp(x-1, j), dp(x-1, k)) + (x\text{일에 } i\text{를 진행해서 받는 돈})$
- ▶ $dp(n, *)$ 의 최대값 출력

10	40	70
20	50	80
30	60	90

10	40	70
$\max(70, 40) + 20$ = 90	$\max(10, 70) + 50$ = 120	$\max(10, 40) + 80$ = 120
$\max(120, 120) + 30$ = 150	$\max(90, 120) + 60$ = 180	$\max(90, 120) + 90$ = 210

```
for(int i=1; i<=N; i++) {  
    st = new StringTokenizer(br.readLine());  
    int x = Integer.parseInt(st.nextToken());  
    int y = Integer.parseInt(st.nextToken());  
    int z = Integer.parseInt(st.nextToken());  
    dp[i][0] = Math.max(dp[i-1][1], dp[i-1][2]) + x;  
    dp[i][1] = Math.max(dp[i-1][0], dp[i-1][2]) + y;  
    dp[i][2] = Math.max(dp[i-1][0], dp[i-1][1]) + z;  
}  
System.out.println(Math.max(dp[N][0], Math.max(dp[N][1], dp[N][2])));
```