# 3기 / 2주차_다이나믹 프로그래밍(DP)

## 🌟 DP

여러 개의 하위 문제를 먼저 푼 후에 그 결과를 쌓아올려서 주어진 문제를 해결하는 알고리즘이다.

DP를 사용하는 이유는 피보나치 수열을 생각하면 쉽다.
재귀로 n번째 수를 구한다고 생각했을 때 호출되는 함수의 횟수는 엄청나게 많을 것이다.
하지만 앞의 값들을 저장해두고 사용하면 이미 계산한 값을 반복할 필요가 없어진다.

DP를 사용하기 위해서는 2가지 조건을 만족해야 한다.

1. Overlapping Subproblems(겹치는 부분 문제)
2. Optimal Substructure(최적 부분 구조)

쉽게 말하자면 동일한 작은 문제들이 반복해서 나타나야 하고, 작은 문제의 최적 값을 사용해 전체 문제의 최적 값을 구할 수 있어야 한다는 것이다.

내가 생각했을 때 DP 문제의 핵심은 이 문제를 DP로 풀 수 있는지 알아내는 것이다. 이게 생각보다 매우 어렵다.

## 1. 백준 1463_1로 만들기

### 🔑 풀이

dp 배열에 각 값을 만들 수 있는 최소 횟수를 채웠다.

dp[7]로 생각을 해 본다면

1. dp[6] + 1 = (dp[2] + 1) + 1 혹은 (dp[3] + 1) + 1 …
2. dp[5] + 2

이렇게 여러 가지 경우의 수가 생기는데 이 중 가장 작은 값으로 넣어주는 것이다.

### 💻 코드

152 ms, 18212 KB

```
package week2;

import java.io.*;

public class Baekjoon_1463 {
    private static int number;
    private static int[] dp;

    public static void main(String[] args) throws IOException
        input();
        solve();
        output();
    }

    private static void output() throws IOException {
        BufferedWriter bw = new BufferedWriter(new OutputStre
        bw.write(dp[number]);
    }

    private static void solve() {
        dp = new int[number + 1];
        dp[0] = 0;
        dp[1] = 0;

        for (int i = 2; i < dp.length; i++) {
            dp[i] = dp[i-1] + 1;
            if (i % 2 == 0) dp[i] = Math.min(dp[i / 2] + 1, d
            if (i % 3 == 0) dp[i] = Math.min(dp[i / 3] + 1, d
        }
    }

    private static void input() throws IOException {
```

```
        BufferedReader br = new BufferedReader(new InputStrea
        number = Integer.parseInt(br.readLine());
    }
}
```

## 2. 백준 2156_포도주 시식

### 🔑 풀이

3가지의 경우의 수가 있다고 생각했다.

1. n-1 포도주까지 마시는 경우
2. n-1 포도주를 마시지 않고 n 포도주를 마시는 경우
3. n-3 포도주까지 마시고 n-1, n 포도주를 마시는 경우

초항값을 먼저 넣어주고 3가지 중 최댓값으로 dp 배열을 채워주었다.

### 💻 코드

152 ms, 14996 KB

```
package week2;

import java.io.*;

public class Baekjoon_2156 {
    private static int n;
    private static int[] quantity;
    private static int[] dp;

    public static void main(String[] args) throws IOException
        input();
        solve();
        output();
    }

    private static void input() throws IOException {
```

```
        BufferedReader br = new BufferedReader(new InputStrea
        n = Integer.parseInt(br.readLine());

        quantity = new int[n];
        dp = new int[n];

        for (int i = 0; i < n; i++) {
            quantity[i] = Integer.parseInt(br.readLine());
        }
    }

    private static void solve() {
        dp[0] = quantity[0];

        if (n > 1) dp[1] = dp[0] + quantity[1];
        if (n > 2) dp[2] = Math.max(Math.max(dp[1], dp[0] + q

        for (int i = 3; i < n; i++) {
            dp[i] = Math.max(Math.max(dp[i - 1], dp[i - 2] +
        }
    }

    private static void output() throws IOException {
        BufferedWriter bw = new BufferedWriter(new OutputStre
        bw.write(String.valueOf(dp[n - 1]));
        bw.flush();
    }
}
```

## 3. 백준 1520_내리막 길

### 🔑 풀이

dfs와 메모이제이션을 사용했다.

방문한 적이 있는 칸은 탐색을 하지 않는 방법이다.

### 💻 코드

428 ms, 36496 KB

```
package week2;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class Baekjoon_1520 {
    private static int row, column;
    private static int[][] map;
    private static int[][] dp;
    private final static int[] dx = {0, 1, 0, -1};
    private final static int[] dy = {1, 0, -1, 0};

    public static void main(String[] args) throws IOException
        input();
        System.out.println(dfs(0, 0));
    }

    private static void input() throws IOException {
        BufferedReader br = new BufferedReader(new InputStrea
        StringTokenizer st = new StringTokenizer(br.readLine(
        row = Integer.parseInt(st.nextToken());
        column = Integer.parseInt(st.nextToken());
        map = new int[row][column];
        dp = new int[row][column];

        for (int i = 0; i < row; i++) {
            st = new StringTokenizer(br.readLine());
            for (int k = 0; k < column; k++) {
                map[i][k] = Integer.parseInt(st.nextToken());
                dp[i][k] = -1;
            }
        }
    }
```

```
    private static int dfs(int x, int y) {
        if (x == row - 1 && y == column - 1) {
            return 1;
        }

        if (dp[x][y] == -1) {
            dp[x][y] = 0;
            for (int i = 0; i < 4; i++) {
                int nextX = x + dx[i];
                int nextY = y + dy[i];

                if (nextX < 0 || nextY < 0 || nextX >= row ||

                if (map[x][y] > map[nextX][nextY]) {
                    dp[x][y] += dfs(nextX, nextY);
                }
            }
        }

        return dp[x][y];
    }
}
```