동적 계획법(dynamic programming)

'동적 계획법(Dynamic Programming)'은 쉽게 말하자면 **어렵거나 큰 문제를 간단하고 작은 여러 개의 문제로 나누어서 풀고 작은 문제의 답들을 이용하여 원래 문제의 답을 구하는 방식**이다. DP 로 축약하여 부르기도 하는 동적 계획법 문제는 대부분 다음과 같은 특징을 가지고 있다.

1. **최적 부분 구조**
   : **문제의 정답이 작은 문제에 대해서도 정답**이어야 한다. 즉 전체 문제의 정답은 작은 문제들의 정답을 포함하며 작은 문제들을 통해 큰 문제를 풀 수 있어야 한다.
2. **부분 문제 반복**
   : **문제를 여러 개의 작은 문제로** 나눌 수 있으며, 나눈 작은 문제들을 전체 문제를 푸는 방법과 같은 방법으로 풀 수 있어야 한다.

이때 똑같은 문제를 여러 번 푸는 것을 막기 위해 **메모이제이션**이라는 기법을 사용하는데, 이는 **미리 구해둔 정답을 메모해놓고 만약 다음번에 다시 해당 문제를 풀고자 한다면 미리 메모해둔 정답을 가져와서 쓰는 기법**이다. 이러한 조건을 가장 잘 설명해주는 예제는 점화식이다. **점화식은 인접한 항들 사이의 관계식**을 말한다. 예를 들어 등차수열은 $a_n = a_{n-1} + C$, 등비수열은 $a_n = r \bullet a_{n-1}$ 과 같은 점화식을 갖습니다. 이 중 등차수열을 N 번째 항까지 계산한 경우, 이 계산 안에는 첫 번째 항부터 N-1 번째 항까지를 포함하고 있으며 작은 문제인 N-1 번째 항을 통해 N 번째 항을 구할 수 있다. 또한, N 번째 항을 구하는 것은 N-1 번째 항에 공차(C)를 더하여 구할 수 있음으로 크기 N 의 문제를 크기 N-1 의 문제로 나눌 수 있다. 그뿐만 아니라 크기가 N 이건, N-1 이건 같은 방법, 즉 같은 점화식을 이용해 문제를 해결할 수 있다.

DP 문제는 하향식, 상향식 두 가지 방법으로 접근할 수 있다. 먼저 **하향식 방법**의 경우 **큰 문제를 풀 수 있는 작은 문제가 될 때까지 나눈 후, 작은 문제들을 풀어 얻은 정답들을 합쳐가며 큰 문제의 답을 구하는 방식**으로, 주로 **재귀 함수를 이용하여 구현**한다. 다음 코드는 **하향식 방법을 이용한 피보나치 수 계산**이다.

```swift
import UIKit

// top_down
func fiboRecursive(_ n: Int) -> Int {
    guard n > 1 else { return n }
    return fiboRecursive(n-1) + fiboRecursive(n-2)
}
```

상향식 방법은 가장 작은 문제부터 시작하여 큰 문제를 풀 수 있을 때까지 차례대로 문제들을 풀어나가는 방식으로 주로 반복문을 이용해 구현한다. 이번 코드는 **상향식 방법을 이용한 피보나치 수 계산**이다.

```swift
// bottom-up
func fibIterative(_ n: Int) -> Int {
    guard n > 1 else { return n }

    var fibArray = [0, 1]

    // 2부터 시작하는데, 그 이유는 배열에 이미 0, 1을 포함하고 있기 때문이다.
    for i in 2...n {
        fibArray.append(fibArray[i - 1] + fibArray[i - 2])
    }

    return fibArray[n]
}
```

사실 대부분의 문제는 두 방법 모두를 이용하여 해결할 수 있지만, 고난도의 특정 문제는 정해진 방법만을 이용해야 풀이할 수 있다. 사실 한 가지 방법만으로 풀 수 있는 문제는 많지 않아 한 가지 방식으로도 굉장히 쉽게 구현할 수 있지만, 다른 한 가지 방법으로는 쉽게 구현하지 못했던 문제도 있으니 두 가지 방법 모두를 연습해보자.