**<8장>-다이나믹 프로그래밍**

-핵심은 **반복된 연산을 피하기 위해 사용**하는 알고리즘! 이다.

-대표적인 알고리즘으로 피보나치 수열이 있다. 🡺이전 두항의 합을 현재 항으로 설정하는 수열

<간단한 피보나치 수열 코딩?>

텍스트이(가) 표시된 사진

자동 생성된 설명

이렇게 재귀 함수의 형태로 피보나치 수열을 만들어 보자. 이럴 경우 x가 1,2 일경우를 제외 하고는 이전 두 항을 재귀적을 호출을 한다.

하지만 이렇게 코드를 작성할 경우 x의 값이 커질 경우 연산의 횟수가 기하급수적으로 늘어난다.

**Why?** 🡺 (x-1)과 (x-2)로 갈 때 중복된 값을 호출하기 때문이다. 즉 fibo1(4)가 fibo1(10)을 계산할때도 fibo1(10)을 계산 할때도 계속 계산되기 때문이다.

-이를 해결하기 위해 메모제이션 방식을 사용하자(**Top-Down** 방식 이라고 함.)

텍스트이(가) 표시된 사진

자동 생성된 설명

해당 방식은 d라는 리스트를 만들어 두고 0으로 초기화 시켜 놓은 다음에 fibo2가 실행 될 때 x 값에 해당되는 인덱스에 값을 저장하는 방식이다. 이때 **d[x]가 0이 아니면 이미 연산 처리가 된 것**이므로 d[x]!=0인 경우에는 연산을 하지 않고 **바로 return**을 시켜준다.

-세번째 방식은 **Bottom-Up**이라는 방식으로 다이나믹 프로그래밍일 때 주로 사용하는 기법이다. (어지간해서는 바텀-업 방식을 사용해라)

**바텀 업의 코드 작성 방식이 중요함**!! 보통 나는 큰 수를 기준으로 값을 줄여가면서 하는 방식을 사용했었다. 하지만 바텀 업 방식은 제일 작은 값부터 주어진 값까지 for문을 사용해 올려서 **작은 값을 먼저 처리하고 상위 값을 처리하는 방식**을 사용한다.

텍스트이(가) 표시된 사진

자동 생성된 설명<예시 코드>

이렇게 하위 인덱스부터 상위 인덱스까지 채워간다. 이 경우 하위 인덱스의 값은 이미 처리되어 있으므로 따로 연산을 할 필요없이 순차적으로 처리가 잘 된다.

어지간 하면 다이나믹 프로그래밍 문제를 풀 때는 Bottom-Up 방식으로 접근하는게 좋다. (재귀 함수의 크기 한계성이라는 문제가 있다)

<실전 문제 예시>