0408 Python **피보나치 수열**

$$a_1 = 1, a_2 = 1$$
$$a_{n+2} = a_n + a_{n+1}\ (\mathrm{n} = 1, 2, \dots)$$

피보나치 수열은 우리가 수학을 배우며 흔하게 접할 수 있으며 동시에 여러 연구거리를 지닌 수열 중 하나다. 여기서 우리는 Python 을 이용해 그 값을 구해볼 것이다.

가장 간단한 접근방법은 a_1 부터 a_n 까지 모두 계산해보는 것이다. 하지만 이를 Python 으로 구현하려면 어떻게 해야 할까? 만약 n 이 정해져 있다면 개수만큼의 변수를 정의하여 그냥 하나하나 계산해보는 방법이 가능하다. 하지만 우리가 해결할 문제는 n 이 정해져 있지 않을뿐더러 이런 방법을 쓴다면 Python 보단 계산기를 사용하는 것이 효율적일 것이다. 따라서 우리는 보다 효율적인 방법에 대해 탐구해보자.

1. **Recursion** 재귀

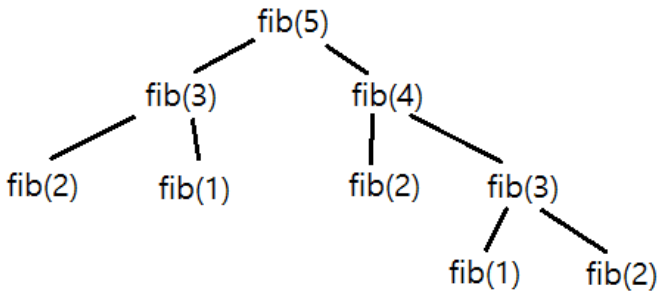
첫 번째로 다뤄볼 방법은 재귀를 통한 접근이다. 사실 이보다 쉬운 방법이 있지만 우리는 아직 여러 개의 변수를 한꺼번에 정의하는 방법을 배우지 않았기 때문에 아직은 다룰 수 없다.

Recursion 즉 **재귀**란 자기 자신을 호출하는 함수를 정의하여 문제를 해결하는 전략이다. 여기서 재귀의 핵심은 함수가 자기 자신을 호출할 수 있다는 것이다. 이는 Python 뿐만 아니라 대부분의 언어에서 가능하다. 하지만 우리는 Python 을 학습할 것이므로 Python 에만 집중할 것이다. 먼저 아래 예제 코드를 보자.

```
1   def fib(n) :
2       if n == 1 or n == 2 :
3           return 1
4       return fib(n - 1) + fib(n - 2)
```

위 코드에는 재귀의 전형적인 구성요소를 모두 나타내고 있다. 재귀함수는 일반적으로 두 부분으로 나뉜다. 첫 번째가 함수의 **중단점을 지정해주는 부분**으로 2, 3 번째 줄이 여기에 해당한다. 이는 수열의 초기조건과도 같은 것으로, 함수가 반복적으로 호출되는 도중의 중단점을 지정해준다. 여기서는 `if n == 1 or n == 2 :` 로 중단 조건을 정하였다. 이는 위의 피보나치 수열의 정의에서 $a_1 = 1, a_2 = 1$ 와 동일하며 `return 1` 을 통해 그 값 또한 1 로 하였다. 두 번째는 자기 자신을 재귀적으로 호출하고 그 값을 이용해 자신의 **반환 값을 지정하는 부분**이다. 이는 `fib(n)` 함수의 정의를 다시 한 번 살피면 쉽게 이해할 수 있다. `fib(n)` 은 n번째 피보나치 값을 반환하는 함수이므로 `fib(n - 1) + fib(n - 2)` 을 반환하는 것은 $\mathrm{n}-1$번째, $\mathrm{n}-2$번째 피보나치 값의 합을 반환하는 것이다. 이는 곧 $a_{n+2} = a_n + a_{n+1}\ (\mathrm{n} = 1, 2, \dots)$ 와 동일함을 알 수 있다.

이제 위의 함수를 직접 구동해보자. `print fib(10)` 을 함수 아래 추가해보면 10 번째 피보나치 수인 55 가 출력된다. 그렇다면 20, 30 번째 피보나치 수 또한 구해보자. 이를 직접 해본다면 20 은 조금 시간이 걸릴 것이고 30 은 너무 긴 시간이 걸려 아마 repl.it 에서 Python 으로는 값을 구하기 힘들 것이다.



이렇게 속도가 느린 이유는 뭘까? 이는 위 그림을 통해 알 수 있다. `fib(5)` 를 호출하면 이는 곧 `fib(3)` 와 `fib(4)` 를 호출하는데 다시 각각에서 2 번의 호출이 이뤄져 `fib(1)` 이나 `fib(2)` 를 호출할 때까지 호출이 반복된다. 하지만 위 그림을 보면 `fib(3)` 은 총 두 번, `fib(2)` 는 3 번, `fib(1)` 은 두 번씩 반복하여 호출된다는 사실을 알 수 있다. 이는 물론 `fib(5)` 를 구하는 과정이기 때문에 무시할 수 있는 횟수지만 `fib(30)` 을 계산하기 위해 `fib(3)` 과 같은 함수가 몇 번씩이나 호출될지 생각한다면 이 방법이 효율적이지 못함을 알 수 있다. 실제로 `fib(n)` 을 구하기 위해서는 총 $2^{n-2}+1$번의 호출이 이뤄진다.

그렇다면 재귀는 비효율적인 전략일까? 사실 재귀는 일반적으로 **divide and conquer** - 분할 정복을 활용하기 위해 쓰이는 전략이다. 여기서 분할 정복이란 문제를 작은 소문제로 분할하여 각각의 문제를 해결하고 이를 다시 결합하는 것을 반복하여 더 큰 문제를 해결하는 것을 의미한다. 이는 곧 작은 문제만 해결하여 큰 문제를 해결할 수 있음을 의미한다. 하지만 이것이 전략의 효율성을 보장하는 것은 아니다. 당장 위의 피보나치의 경우처럼 오히려 비효율을 초래하는 경우도 상당히 많다.

2. **Non-recursive Function** 비재귀 함수

재귀 함수의 비효율성을 해결하는 방법은 여러 가지가 있다. 우리는 그 중에서도 대표적인 비재귀 함수를 활용한 방법을 알아볼 것이다.

비재귀 함수란 문자 그대로 재귀 함수가 아닌 함수를 의미한다. 보통은 재귀 함수로 짜인 함수를 재귀를 사용하지 않고 구현한 것을 의미한다. 비재귀 함수를 만드는 방법 또한 여러 가지가 있다. 우리는 그 중에서 새로운 알고리즘을 사용한 구현에 대해서 알아볼 것이다.

```
1    def fib2(n) :
2        b = True
3        f1 = 1
4        f2 = 1
5
6        while n > 2 :
7            if b :
8                f1 = f1 + f2
9            else :
10               f2 = f1 + f2
11
12           b = not b
13           n -= 1
14
15       if b :
16           return f2
17       else :
18           return f1
```

위 함수는 피보나치 수열을 계산하는 비재귀 함수의 예시다. 위 함수의 각 줄을 분석해보면 다음과 같다.

2 번	위 방법의 핵심이라 할 수 있는 Boolean 타입의 변수다. 이를 활용하여 2 개의 변수만으로 임의의 n번째 피보나치 수를 계산할 것이다
3, 4 번	각 단계의 피보나치 수를 저장할 두 개의 변수다.
6 번	n번째 피보나치 수를 구할 때까지 `while` 문을 반복할 것이다. 여기서 정지 조건이 2 가 되는 것은 이미 `f1` `f2` 에 1, 2 번 피보나치 수를 가지고서 시작하기 때문이다.
7 ~ 10 번	`b` 를 통해 현재 상태를 파악하고 피보나치 수를 구해나가는 부분이다. 자세한 설명은 아래에 추가하였다.
12, 13 번	`b` 를 `True` `False` 상태로 매번 전환하고 `n` 을 1 씩 줄여 현재 상태를 카운팅한다.
15 ~ 18 번	`b` 를 통해 현재 상태를 파악하고 반환 값을 정한다

여기서 사용된 전략은 다음과 같다. 먼저 $n+2$번째 피보나치 수를 구할 때는 n, $n+1$번째의 두 개의 피보나치 수만 필요하다는 것에 주목하였다. 이는 곧 그 전의 수들은 필요가 없음을 의미하고 동시에 2 개의 변수만이 필요하다는 것을 의미한다.

그 이후의 단계는 비교적 간단하다. 변수 `b` 를 정하여 현재 `f1` 과 `f2` 중 어느 것을 버릴 차례인지 확인하고 각각의 경우에 다음 피보나치 수를 계산해주는 것이다.

재귀적 방법과 비교할 때, 이 방법은 어느 정도의 성능 향상을 가져올 수 있을까? 우선 `fib2(30)`을 호출한다면 바로 832040 을 출력하는 것을 확인할 수 있다. 심지어 `fib2(10000)`을 호출하더라도 비교적 빠르게 결과를 계산해내며 `fib2(100000)`조차 조금 기다린다면 그 결과값을 얻을 수 있다.

이러한 성능 차이는 어디서 계산 횟수를 비교한다면 원인을 쉽게 찾을 수 있다. 앞에서 `fib` 가 함수 호출 횟수를 세었듯이 이번에는 `f1` 과 `f2` 의 값을 계산하는 수를 세어본다면 $\mathrm{n}-2$회라는 결론을 얻을 수 있다. 이는 앞의 $2^{\mathrm{n}-2}+1$와 비교할 때 얼마나 적은 횟수인지 직감적으로 알 수 있을 것이다.

우리는 피보나치 수열을 Python 으로 계산하는 과정을 통해 재귀 함수에 대해 배워보았고 여기서 발전하여 비재귀적 접근 또한 배워보았다. 우리가 이 과정을 통해 알 수 있는 것은 단순한 재귀 함수에 대한 이해뿐만 아니라, 같은 문제라도 알고리즘에 따라 여러 풀이가 가능하며 그 풀이의 효율성 또한 크게 차이 남을 확인할 수 있었다. 따라서 이후에 어떠한 문제를 해결하는 과정에서는 단순히 결과값을 찾는 것뿐만 아니라 그 효율성까지 따져봐야 할 것이다. 그리고 앞에서 다룬 방법 외에도 n 번째 피보나치 수열의 값을 약 $\log_2 n$ 번 만에 계산할 수 있는 방법도 있으니 추후에 탐구해볼 수 있겠다.