기초빅데이터프로그래밍 과제3: Fibonacci number

전공: 컴퓨터공학 학년: 3학년 학번: 20201635 이름: 전찬

**0. 목차**

1. Fibonacci number

2. Python을 이용한 Fibonacci number 구하기

2-1. recursive function(재귀 함수)을 활용한 구현

2-2. loop을 활용한 구현

2-3. Dynamic programming을 활용한 구현

3. 2의 각 방법에서의 시간 비교

**1. Fibonacci number**

Fibonacci number(피보나치 수)는 다음 식을 만족하는 수들을 의미한다.

(1) F0 = 0

(2) F1 = 1

(3) Fn = Fn-1 + Fn-2

따라서 피보나치 수는 F0부터 각각 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …. 를 만족한다.

**2. Python을 이용한 Fibonacci number 구하기**

위에서 설명한 피보나치 수를 python을 이용해 구하는 방법은 다양하게 존재한다. 각각은 아래와 같다.

(1) recursive function(재귀 함수)을 이용한 구현

(2) loop(반복문)을 이용한 구현

(3) Dynamic programming(동적 계획법)을 이용한 구현

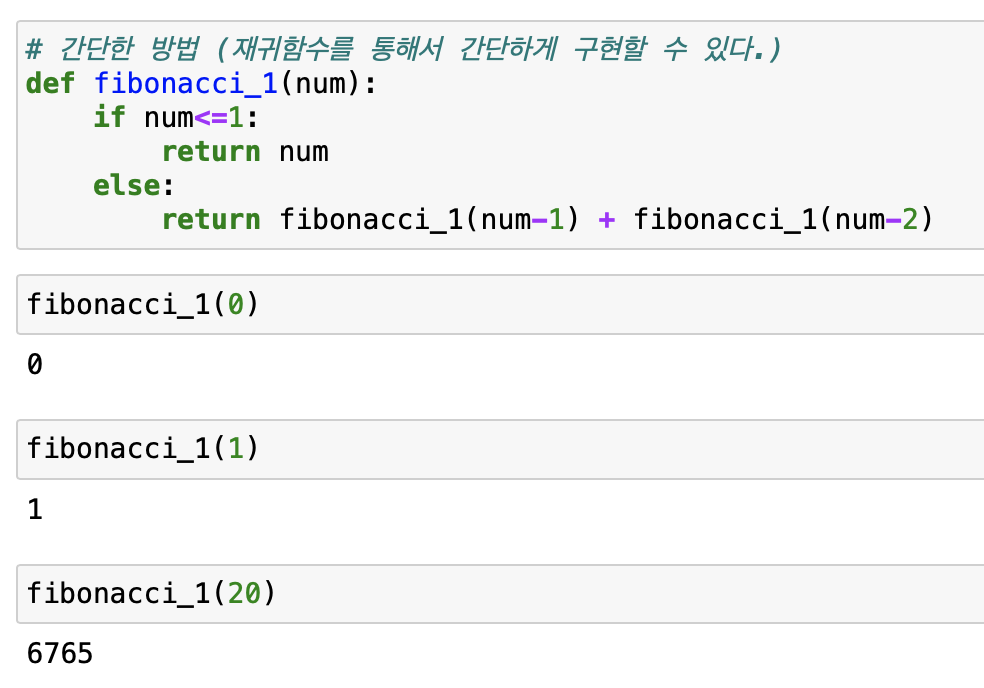
(4) 피보나치 수의 일반항을 이용한 구현

(5) 행렬의 곱을 이용한 구현

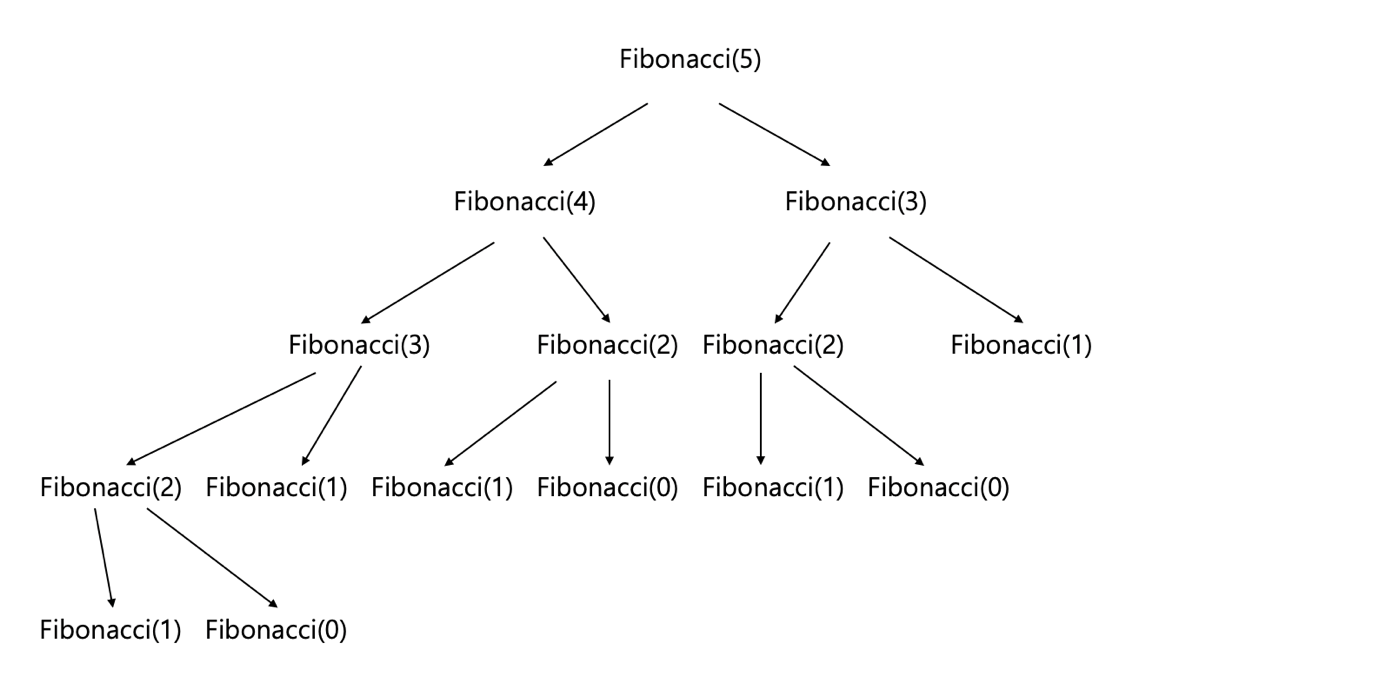
이 중에서 피보나치 수의 일반항과 행렬의 곱을 이용한 구현은 수학적 개념 위주의 구현이므로 생략하고, (1), (2), (3) 을 직접 구현해보자.

**2-1. recursive function(재귀 함수)을 이용한 구현**

recursive function을 이용한 피보나치 수의 구현은 가장 간단하게 생각해낼 수 있는 방법이다. 피보나치 수 자체가 점화식의 형태로 표현되기 때문에, 이를 재귀 함수로 구현하며, 함수의 이름이 Fibonacci라면, return Fibonacci(n-1) + Fibonacci(n-2) 형태로 간단하게 구현해낼 수 있다. 구현은 아래와 같다.



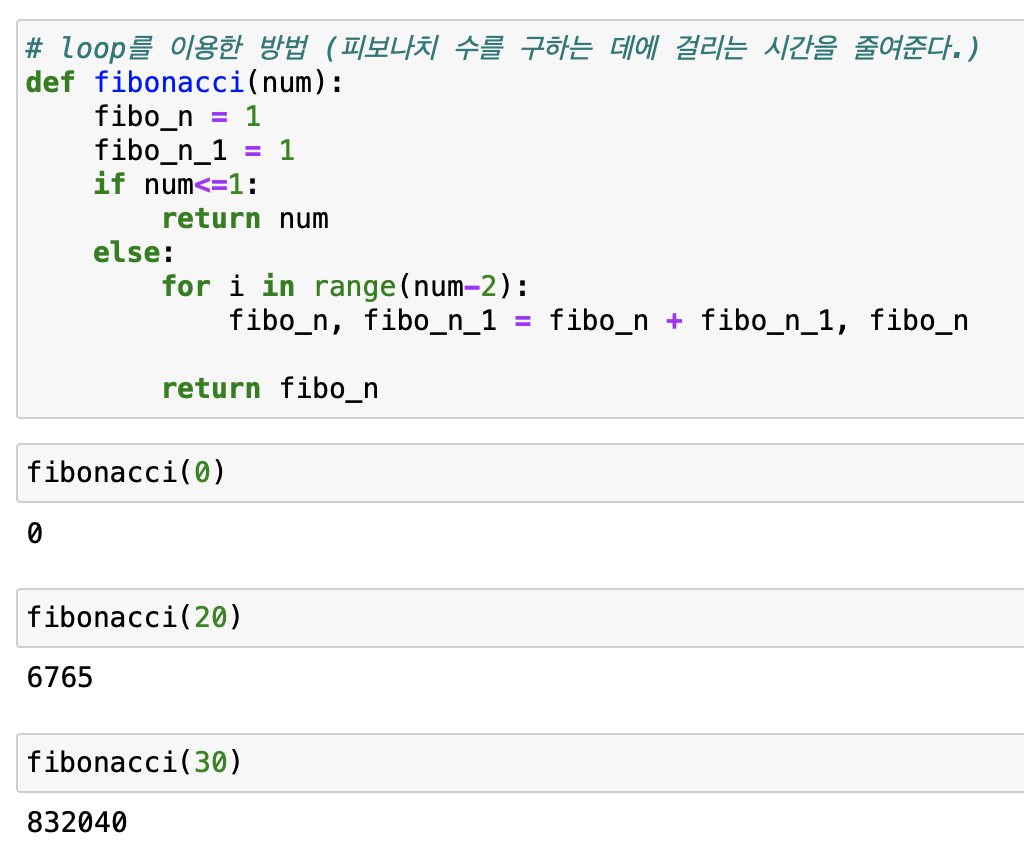
위와 같이 피보나치 수를 구하는 함수를 구현한 경우, 장점으로는 코드 상에서 가장 간단하게 구현할 수 있다는 장점이 있다. 하지만 Fibonacci(5)를 구하는 경우를 생각해 보자. 그 형태는 아래와 같을 것이다.



위 그림에서 볼 수 있듯이, Fibonacci(5)를 구하기 위해서는 13번의 function call이 필요하다. 이는 구하는 피보나치 수가 커질수록, 점점 더 커질 것이며, Fibonacci(n)을 구하기 위해서는 약 2n의 시간 형태로 지수함수 형태로 구하는 시간이 증가한다는 것을 알 수 있다.

**2-2. loop(반복문)을 이용한 구현**

loop을 이용해서도 피보나치 수를 구해낼 수 있다. 이 경우에, 구현은 재귀 함수를 이용해 피보나치 수를 구하는 형태보다는 약간 복잡해지며, 직관적이지 않게 된다. 구현은 아래와 같다.



위와 같이 피보나치 수를 구하는 함수를 구현한 경우, 재귀 함수를 통한 구현과 다르게 피보나치 수를 구하는 데에 걸리는 속도가 크게 감소할 수 있다. 재귀 함수를 통해 피보나치 수를 구하기 위해서는 약 2n의 시간이 필요하다고 위에서 설명했는데, loop를 이용한 방식은, 아래와 같은 형태로 피보나치 수를 구하게 된다.

Fibonacci(2) -> Fibonacci(3) -> Fibonacci(4) -> … -> Fibonacci(n)

따라서 Fibonacci(n)을 구하기 위해서는 약 n번의 반복, 혹은 시간이 필요하며, 이는 2n의 시간보다는 훨씬 작을 것임을 파악할 수 있다. 하지만 loop을 이용해서 구한 경우에, 위에서 설명한 것처럼 구현이 직관적이지 않다는 단점이 존재하기도 한다.

**2-3. Dynamic programming(동적 계획법)을 이용한 구현**

재귀 함수, 반복문을 이용한 방법은 각각 시간이 오래 걸리거나, 직관적으로 이해하기 어렵다는 단점이 존재한다. 이러한 단점이 없는 방법 또한 존재하는데, Dynamic programming을 이용한 방법이다.

기본적으로 Dynamic programming은 프로그래밍을 통한 문제 해결 분야에서 다양한 형태로 활용되는 방법이다. 요즘에 취업, 대학원 시험 등 다양한 분야에서 진행하는 코딩 테스트(코테)에서도 단골로 출제되는 개념이기도 하다. Dynamic programming의 기본적인 형태는 아래와 같다.

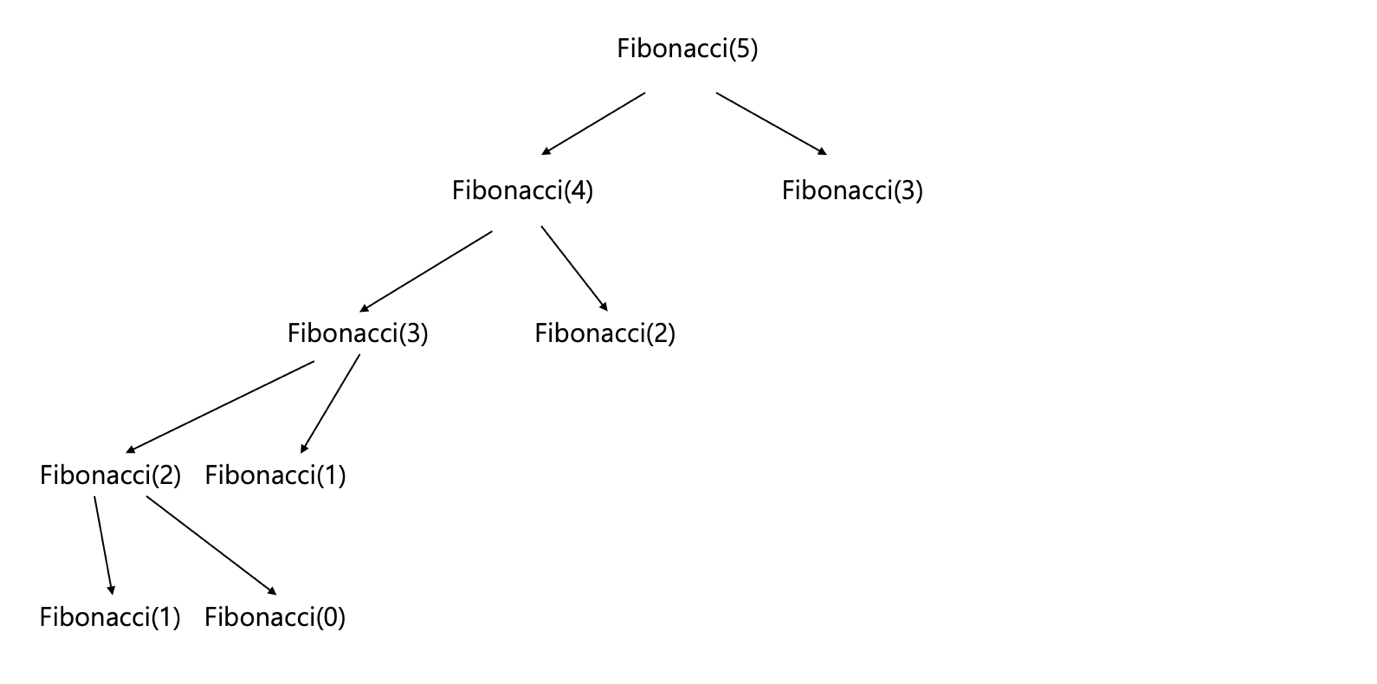
(1) 큰 문제를 작은 문제들로 나눈다.

(2) 작은 문제의 solution(해답)을 저장하며, 한 번 구한 작은 문제는 다시 구하지 않는다.

따라서 Fibonacci(n)을 구할 때, (1)의 방법으로 Fibonacci(n-1) / Fibonacci(n-2) 형태로 분할하며, (2)의 방법으로 Fibonacci(3), Fibonacci(4) 등을 한 번 구한 이후 저장해서 나중에 다시 사용하는 형태로 피보나치 수를 구할 수 있다. python을 통한 구현은 아래와 같다.



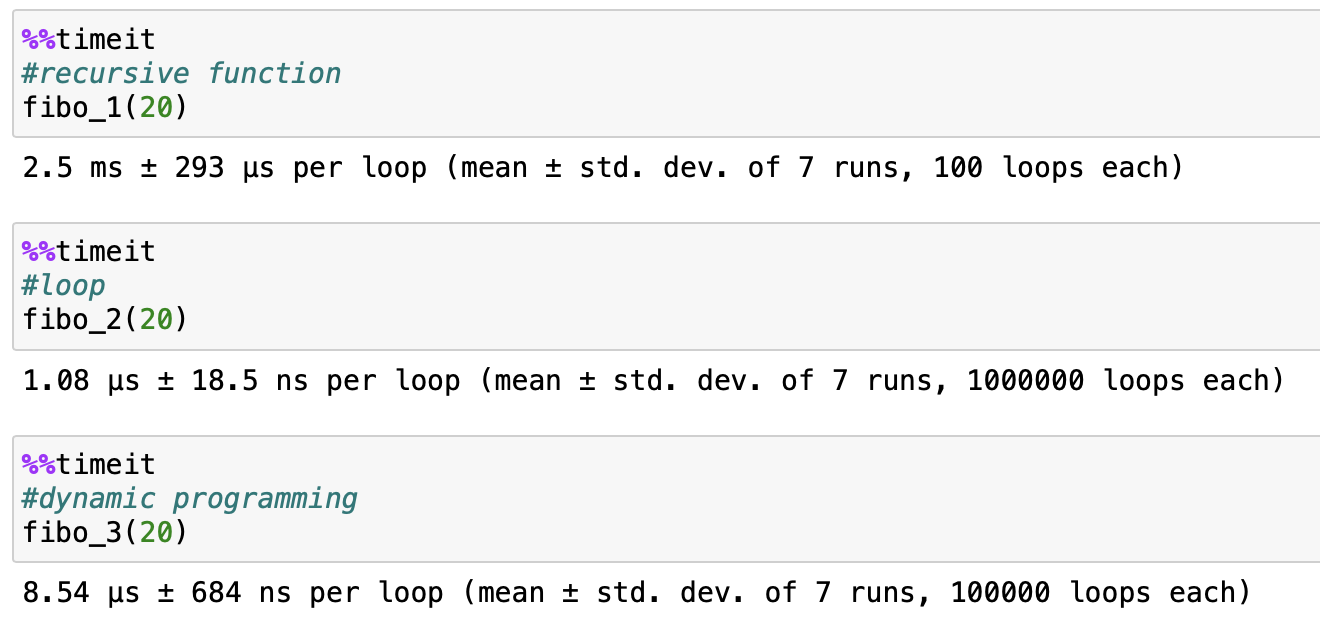
위와 같은 방법으로, 피보나치 수를 구하는 경우, recursive function을 통해 구하는 방법보다 더욱 적은 시간으로 피보나치 수를 구할 수 있다. 위에서 재귀 함수로 Fibonacci(5)를 본 것처럼, 이번에는 Dynamic programming을 이용한 Fibonacci(5)를 구하는 형태를 파악해 보자. 그 형태는 아래와 같다.



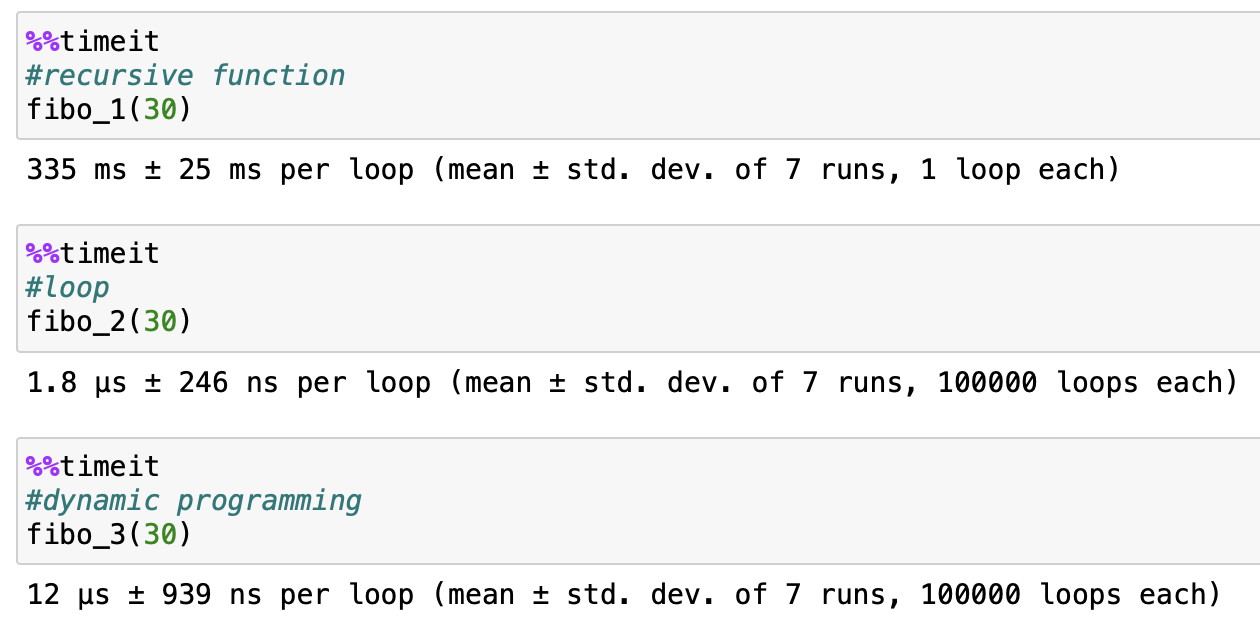
이 경우에, 왼쪽 부분은 재귀 함수 형태와 비슷하지만, 이후에 수행되는 오른쪽 부분이 재귀 함수의 형태보다 function call이 적다는 것을 알 수 있다. 이 이유는 왼쪽에서 Fibonacci(2), Fibonacci(3), Fibonacci(4)을 이미 구하고 저장했기 때문에, 오른쪽에서 Fibonacci(2), Fibonacci(3)을 구할 때 더 이상의 함수의 호출 없이, 바로 저장된 데이터를 사용하면 되기 때문이다. 따라서 Dynamic programming 또한 약 n번의 function call, 혹은 시간이 걸리는 형태이며(저장을 한 번씩만 하며, 이후에는 function call 없이 data를 바로 사용하면 되기 때문에) loop의 형태에 비해서는 이해하기 쉬운 형태이라는 장점 또한 가지고 있다.

**3. 2의 각 방법에서의 시간 비교**

마지막으로, 위에서 구현한 3가지 각각의 방법을 fibo\_1, fibo\_2, fibo\_3으로 설정하고 걸리는 시간을 Jupyter notebook에서 Magic keyword를 통해서 구해보자. 각각 20번째 수를 구하며, 그 결과는 아래와 같다.



이를 통해서 loop가 가장 빠르지만, dynamic programming 또한 그렇게 큰 차이가 존재하지 않으며 빠르다는 것을 파악할 수 있다. 또한 recursive function는 구하는 데에 다른 방법보다 100~1000배까지 속도차이가 난다는 것을 파악할 수 있다. 추가로 30번째 수도 구해보면 아래와 같다.



위 결과를 통해, fibo\_2, fibo\_3은 20번째 수를 구하는 것과 비교해서 그렇게 큰 차이가 나지 않지만, fibo\_1(recursive function)은 20번째 수를 구하는 것에 비해 약 100배의 시간이 더 걸린다는 것을 파악할 수 있다. 이를 통해 fibo\_1은 걸리는 시간이 지수함수 형태로 커진다는 것 또한 유추해낼 수 있다.