알고리즘 - 다이나믹 프로그래밍

DP

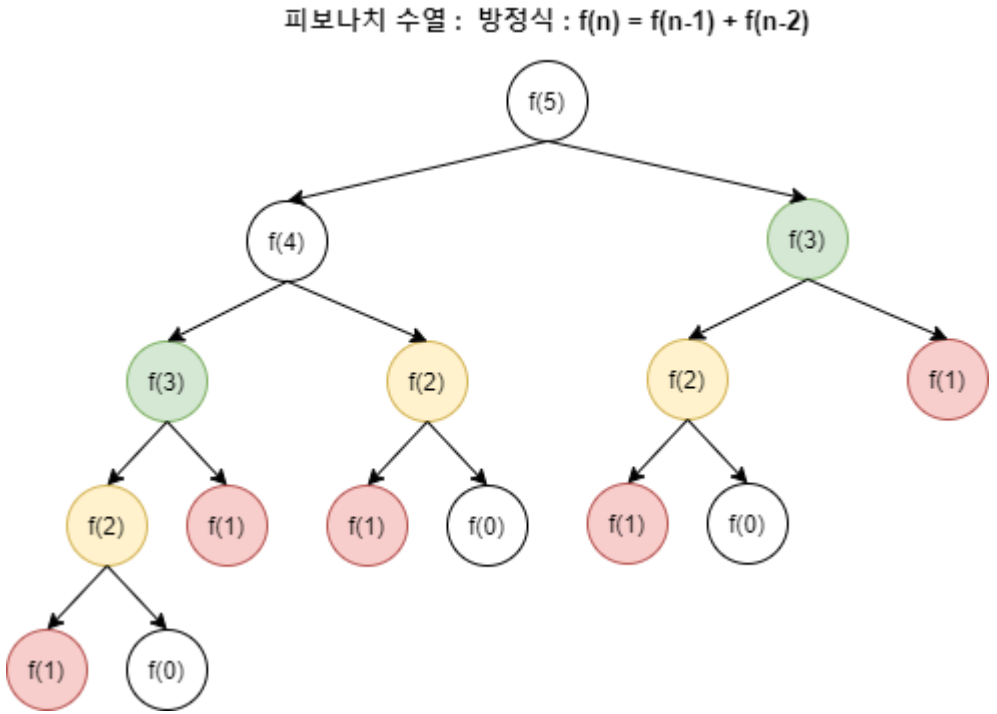
동적 계획법

동적 계획법 (Dynamic Programming)

- 추가적으로 발생하는 불필요한 계산을 줄이면서, 최적해를 찾는 알고리즘

- 일반적으로 재귀함수를 호출하여 풀 수 있지만, 재귀함수 호출 시 엄청나게 많은 연산횟수가 발생함

- 따라서 계산된 값을 저장해두고, 여기서 꺼내서 사용하면 불필요한 연산을 줄일 수 있다.



예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

```
# 재귀함수로 구현
def fibo(n):
    if n == 1:
        return 1
    if n == 0:
        return 1

    return fibo(n-1)+fibo(n-2)
```

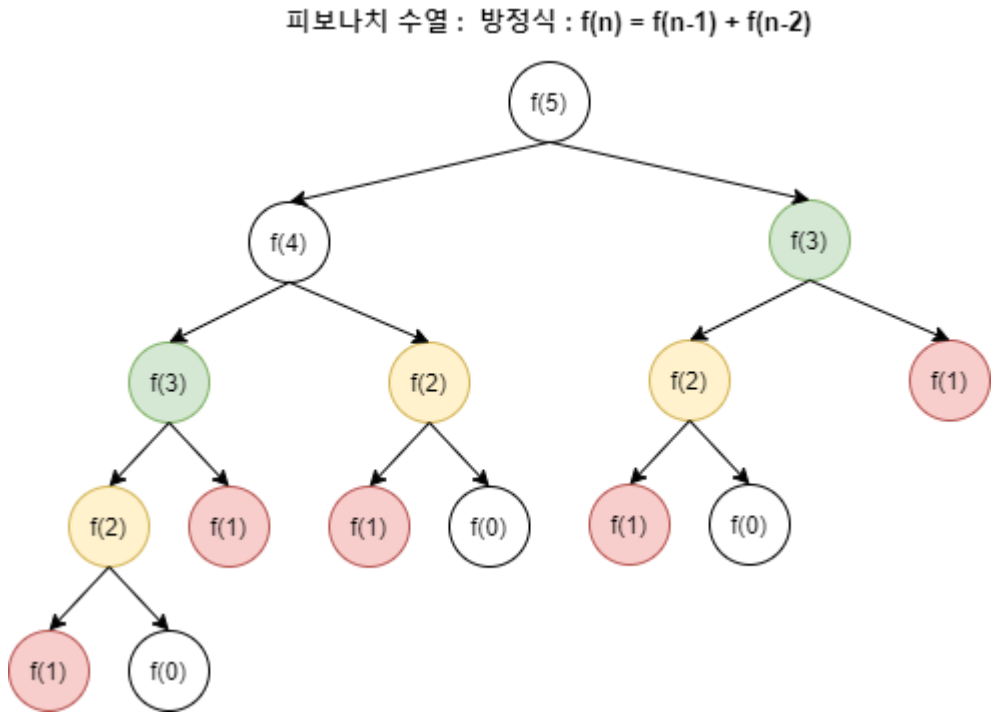
피보나치 수열을 재귀함수로 구현하면 위와 같다.

예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

```
# 재귀함수로 구현
def fibo(n):
    if n == 1:
        return 1
    if n == 0:
        return 1

    return fibo(n-1)+fibo(n-2)
```



피보나치 수열을 재귀함수로 구현하면 위와 같다.

오른쪽과 같이, 피보나치 수열의 5번째 값을 가져오기 위해선
총 14번의 연산이 필요하다

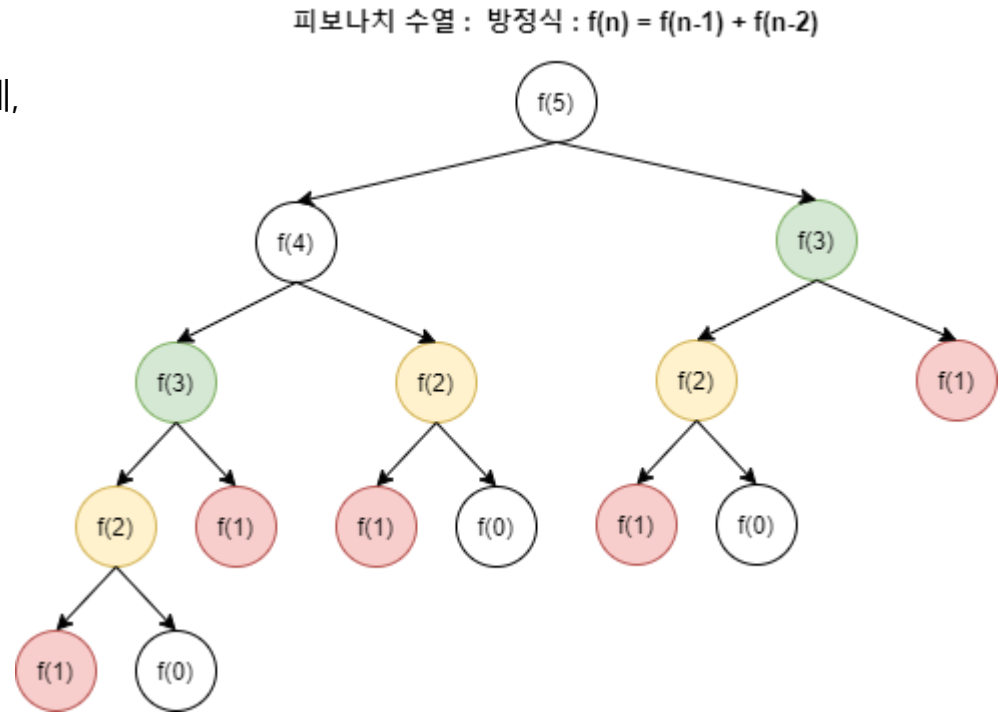
예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

또한, 이것은 파이썬이 내부적으로 재귀함수를 처리하는 방법에 대한 내용인데, 파이썬은 함수를 stack으로 쌓아놓고 하나씩 꺼내서 연산을 처리한다.

stack은 깊이가 깊어질수록, 연산시간이 기하급수적으로 늘어난다는 단점이 있다.

함수에 N을 1000으로 두고 시간이 얼마나 걸리는지 확인해보자.

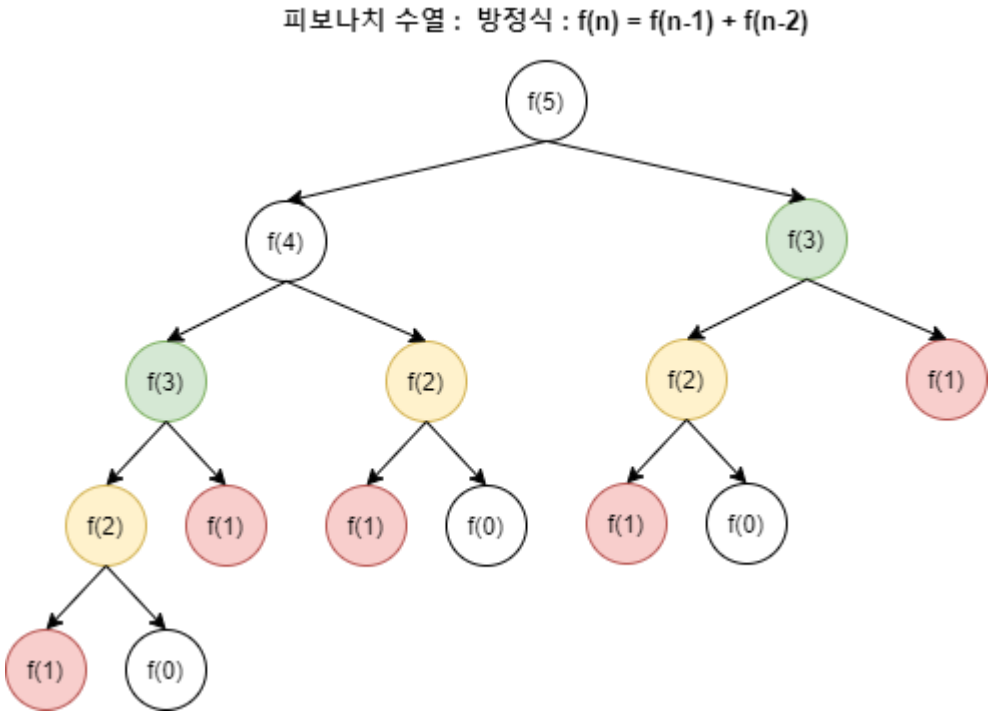


예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

다이나믹 프로그래밍의 기본적인 개념은 **메모이제이션**이다.

메모이제이션?

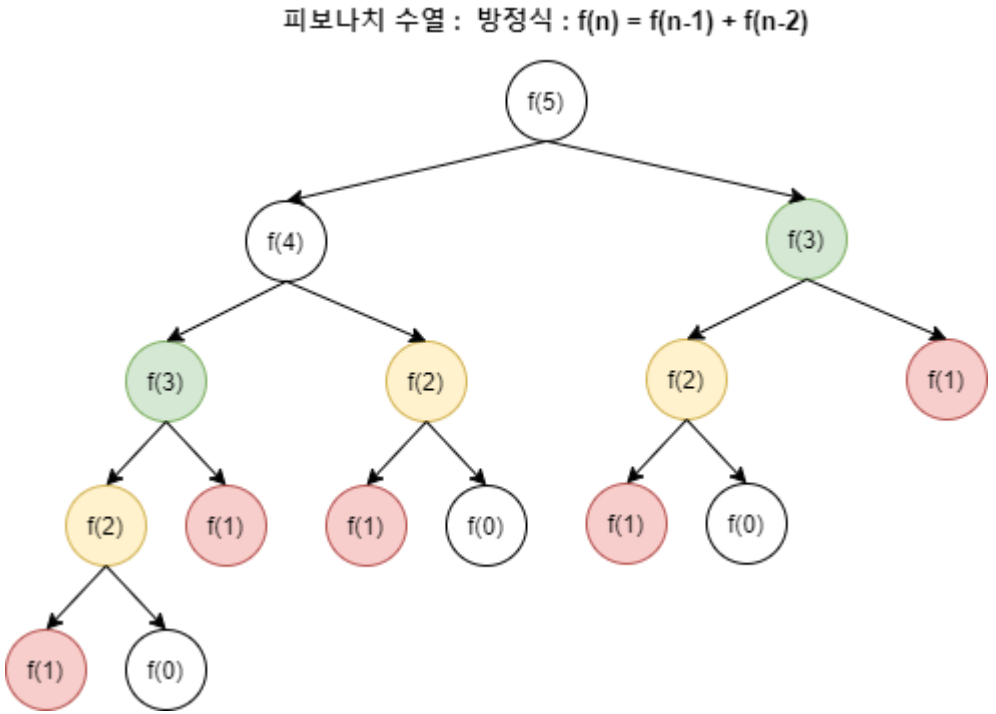


예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

다이나믹 프로그래밍의 기본적인 개념은 **메모이제이션**이다.

메모이제이션?
컴퓨터 프로그램이 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 메모리에 저장함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술

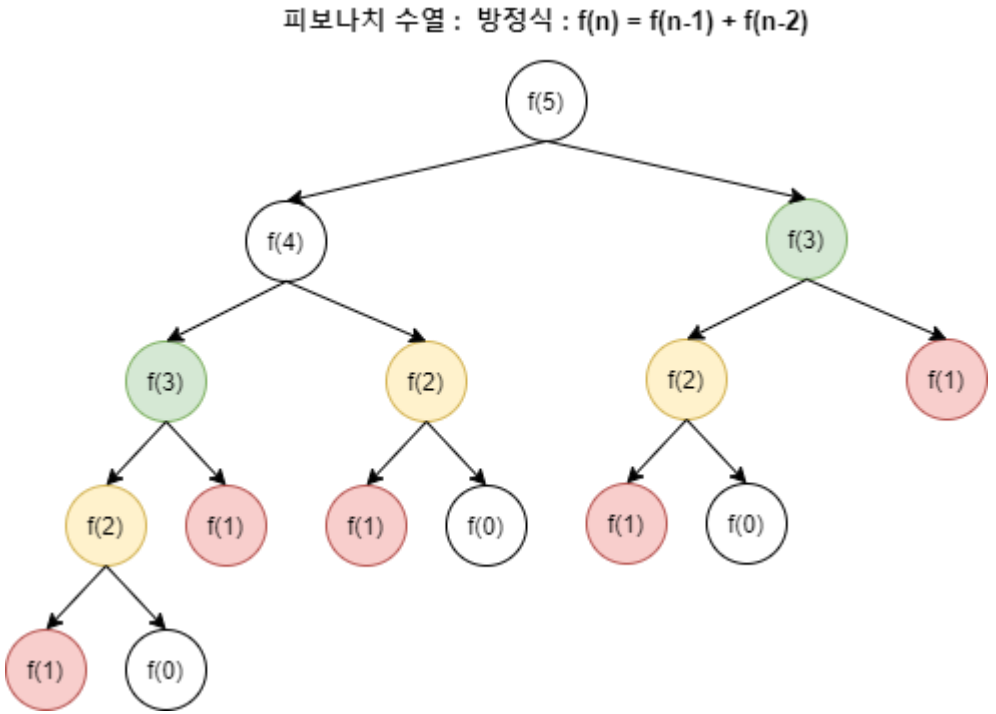


예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

다이나믹 프로그래밍의 기본적인 개념은 **메모이제이션**이다.

메모이제이션?
컴퓨터 프로그램이 **동일한 계산**을 반복해야 할 때, 이전에 계산한 값을 **메모리에 저장**함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술



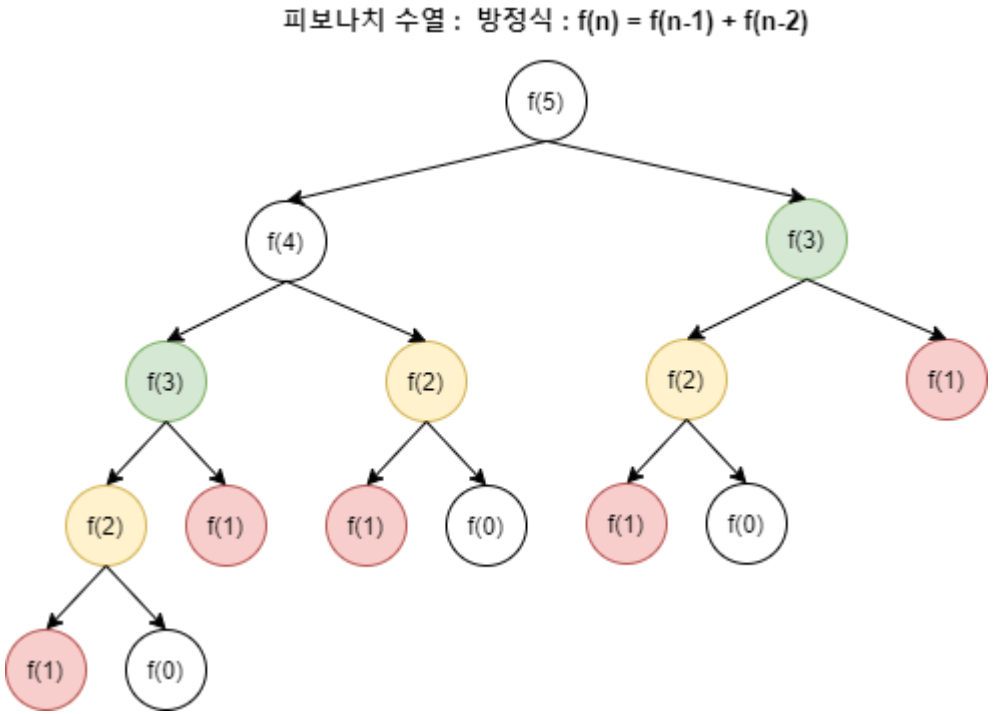
예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

다이나믹 프로그래밍의 기본적인 개념은 **메모이제이션**이다.

메모이제이션?
컴퓨터 프로그램이 **동일한 계산**을 반복해야 할 때, 이전에 계산한 값을 **메모리에 저장**함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술

그렇다. 핵심은 이것이다. 동일한 계산은 저장해두고 필요할 때 꺼내쓰는 것 !

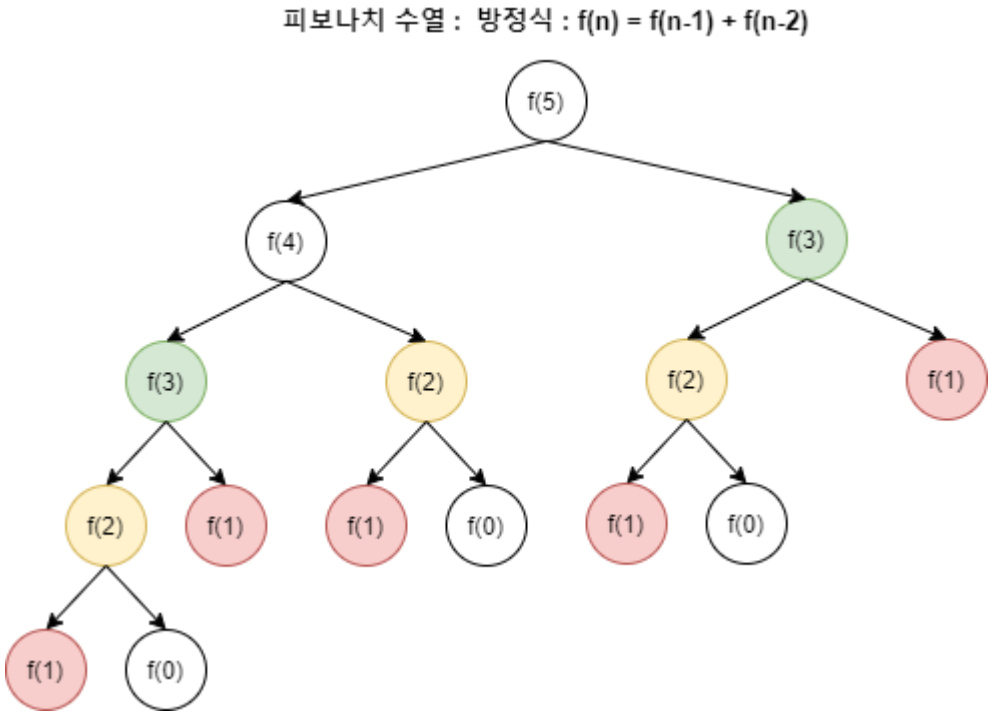


예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

피보나치 수열에서 N번째 항의 값은 앞의 두 항의 합과 같다.



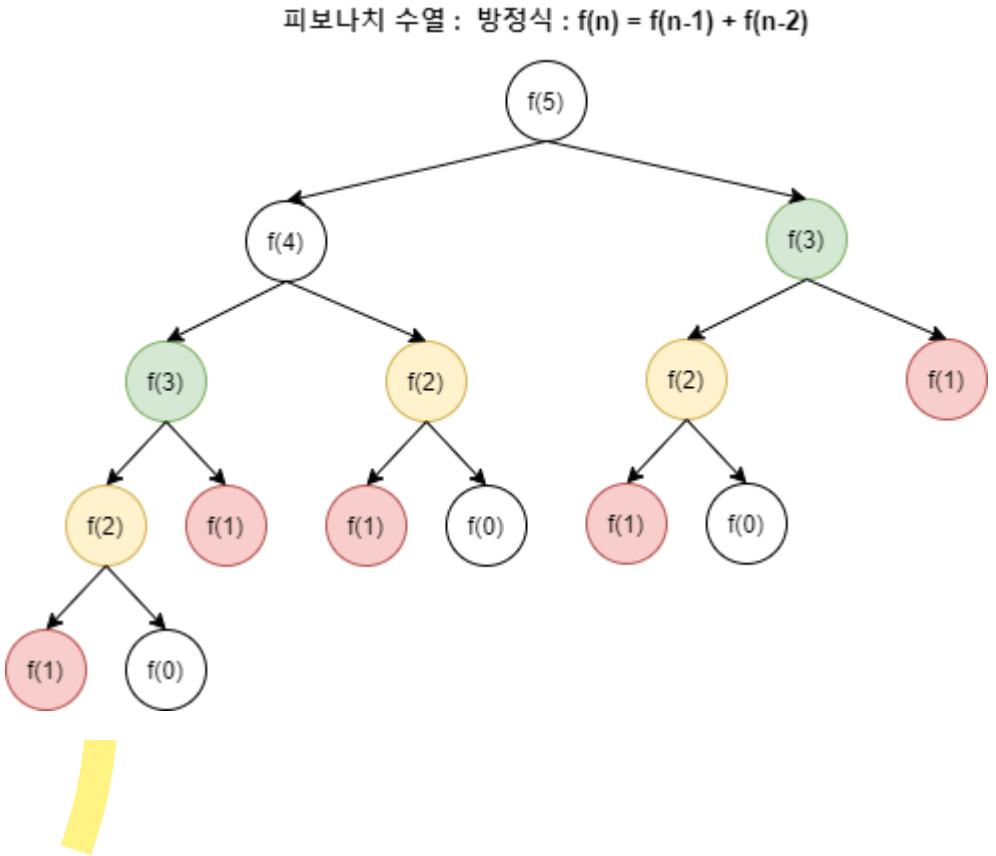
예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

피보나치 수열에서 N번째 항의 값은 앞의 두 항의 합과 같다.

그리고, 0번 항과 1번 항은 값이 각각 1이다.



예제 : 피보나치 수열

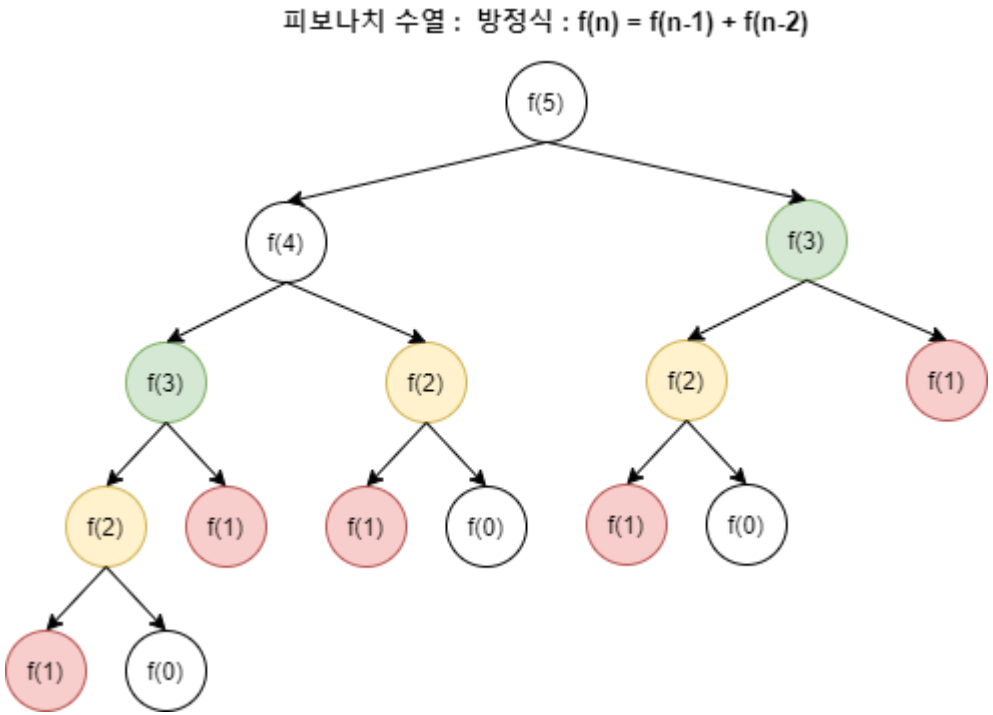
- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

피보나치 수열에서 N번째 항의 값은 앞의 두 항의 합과 같다.

그리고, 0번 항과 1번 항은 값이 각각 1이다.

이것은, 다음과 같은 배열로 표현할 수 있다. -> fibo_list = [1, 1]



예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

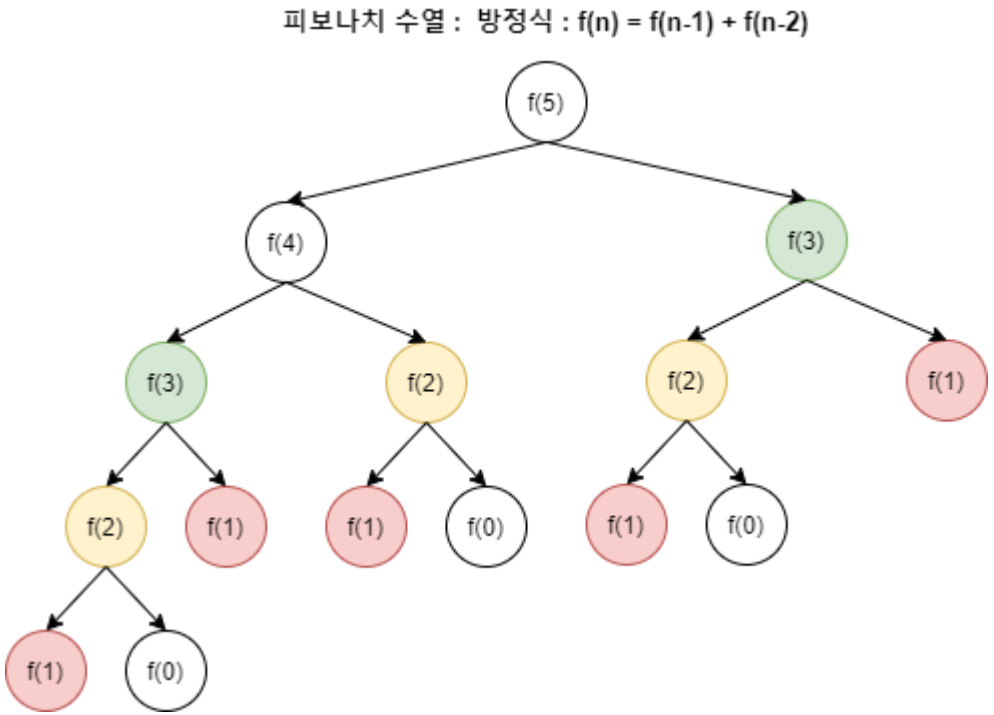
이를 피보나치 수열에 적용해보자.

피보나치 수열에서 N번째 항의 값은 앞의 두 항의 합과 같다.

그리고, 0번 항과 1번 항은 값이 각각 1이다.

이것은, 다음과 같은 배열로 표현할 수 있다. -> fibo_list = [1, 1]

피보나치 수열의 0번 째 항은?
fibo_list[0] = 1



예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

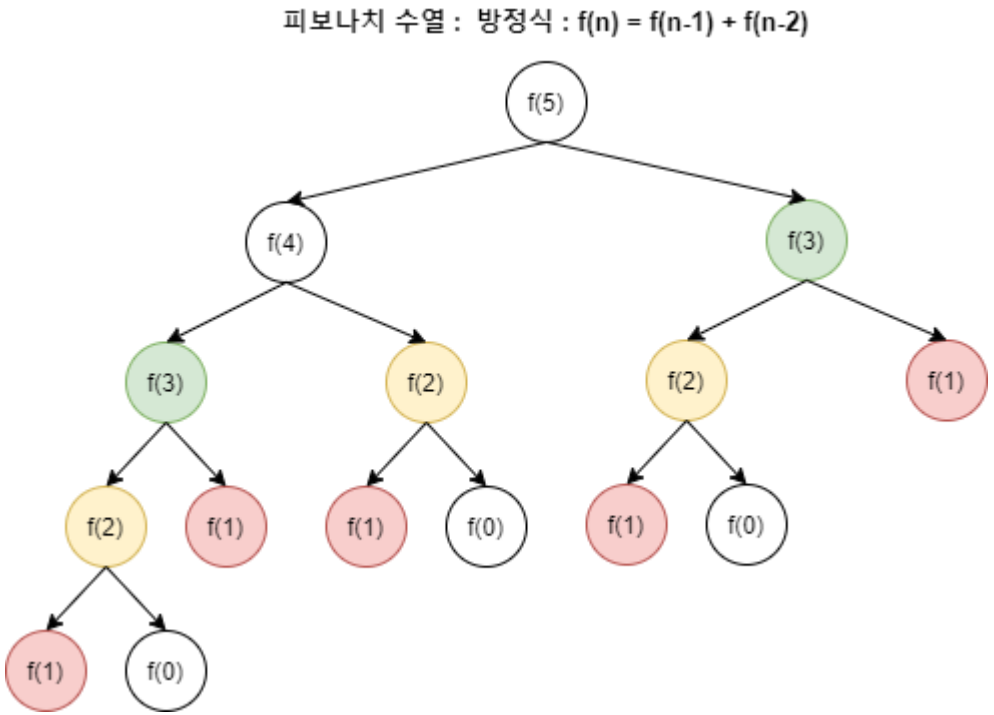
피보나치 수열에서 N번째 항의 값은 앞의 두 항의 합과 같다.

그리고, 0번 항과 1번 항은 값이 각각 1이다.

이것은, 다음과 같은 배열로 표현할 수 있다. -> fibo_list = [1, 1]

피보나치 수열의 0번 째 항은?
fibo_list[0] = 1

그럼, 1번째 항은?
fibo_list[1] = 1



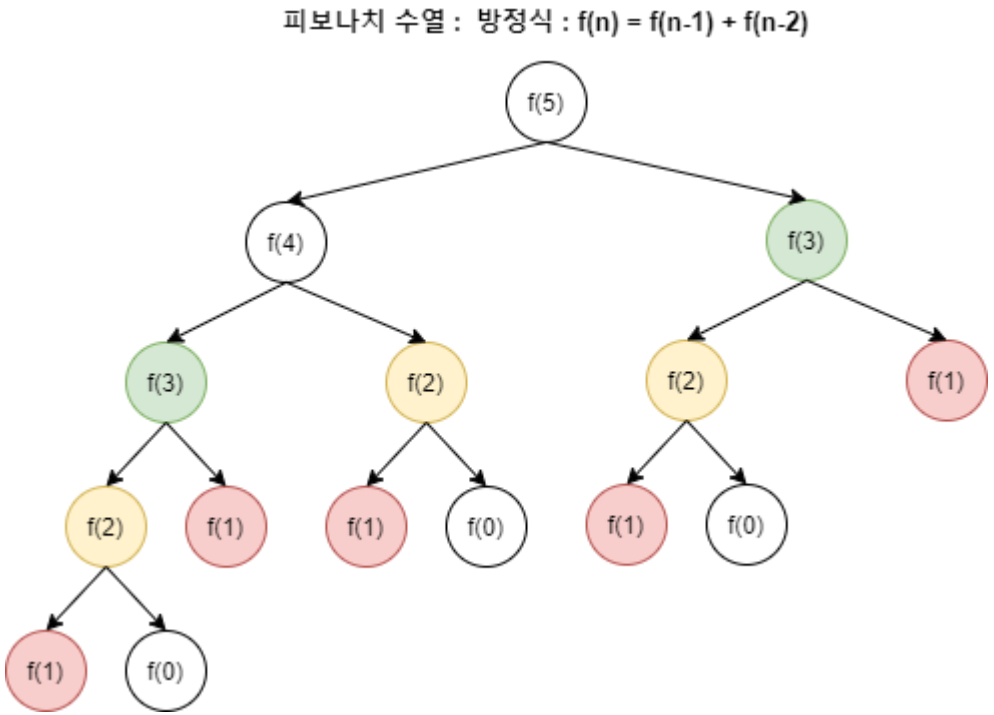
예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1]
이 배열을 가지고, 피보나치 수열을 계산해보자.

n은 2부터 시작해서, 5번째 피보나치 수열의 값을 계산한다고 해보자.



예제 : 피보나치 수열

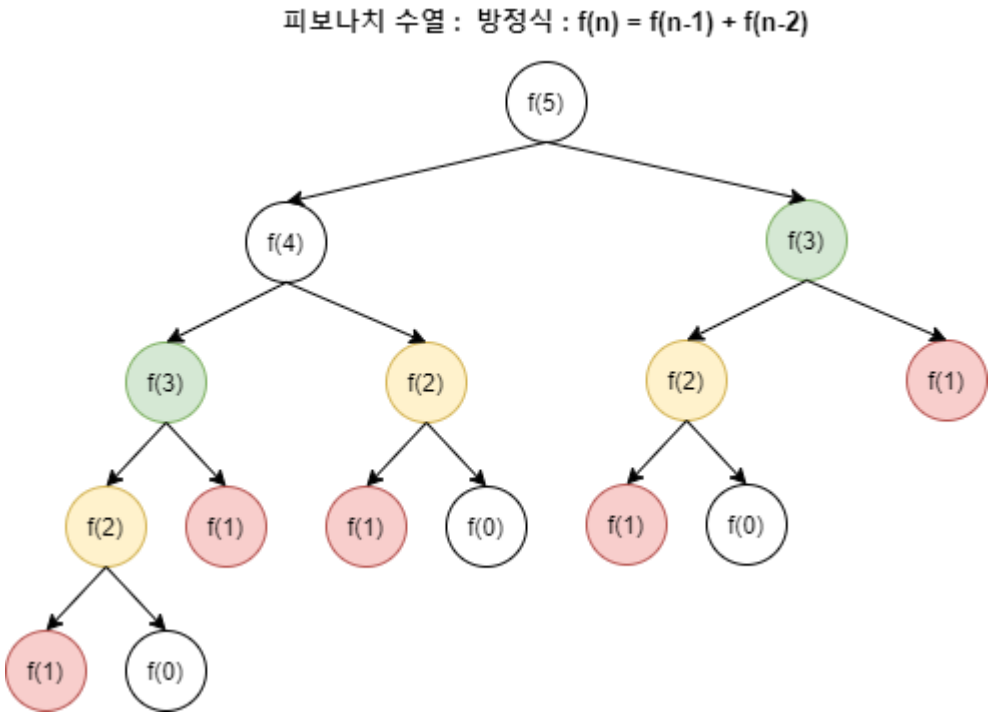
- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1]
이 배열을 가지고, 피보나치 수열을 계산해보자.

n은 2부터 시작해서, 5번째 피보나치 수열의 값을 계산한다고 해보자.

피보나치 수열의 2번째 항은, 1번째 항과 0번째 항의 합과 같다.



예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

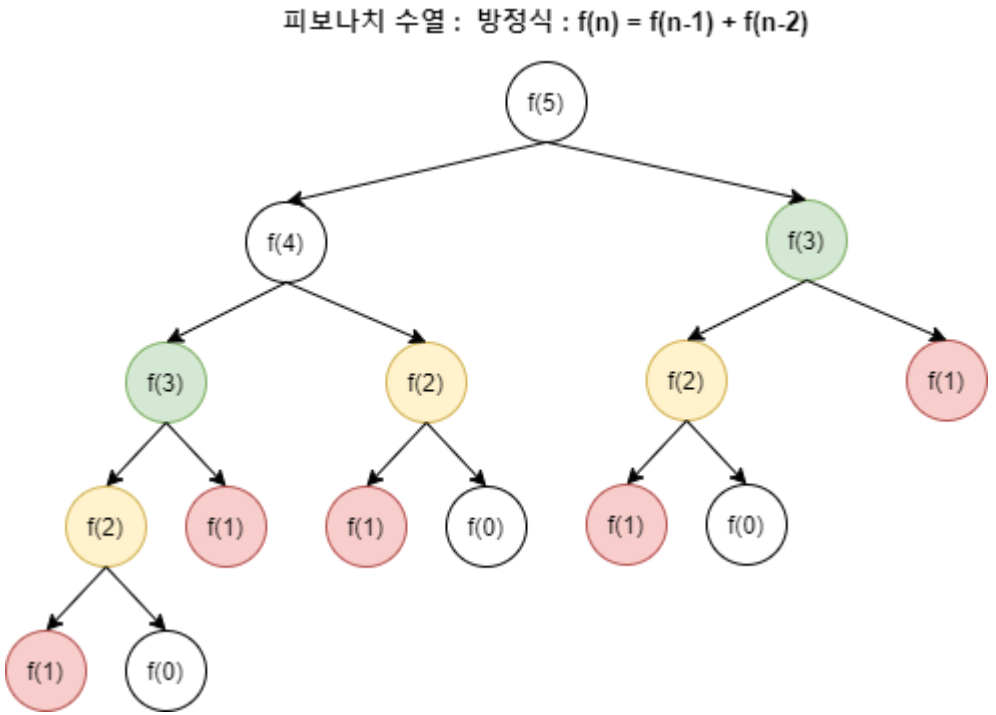
이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1]
이 배열을 가지고, 피보나치 수열을 계산해보자.

n은 2부터 시작해서, 5번째 피보나치 수열의 값을 계산한다고 해보자.

피보나치 수열의 2번째 항은, 1번째 항과 0번째 항의 합과 같다.

이를 수식으로 표현하면,
fibo_list[2] = fibo_list[1] + fibo_list[0]이 된다.



예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

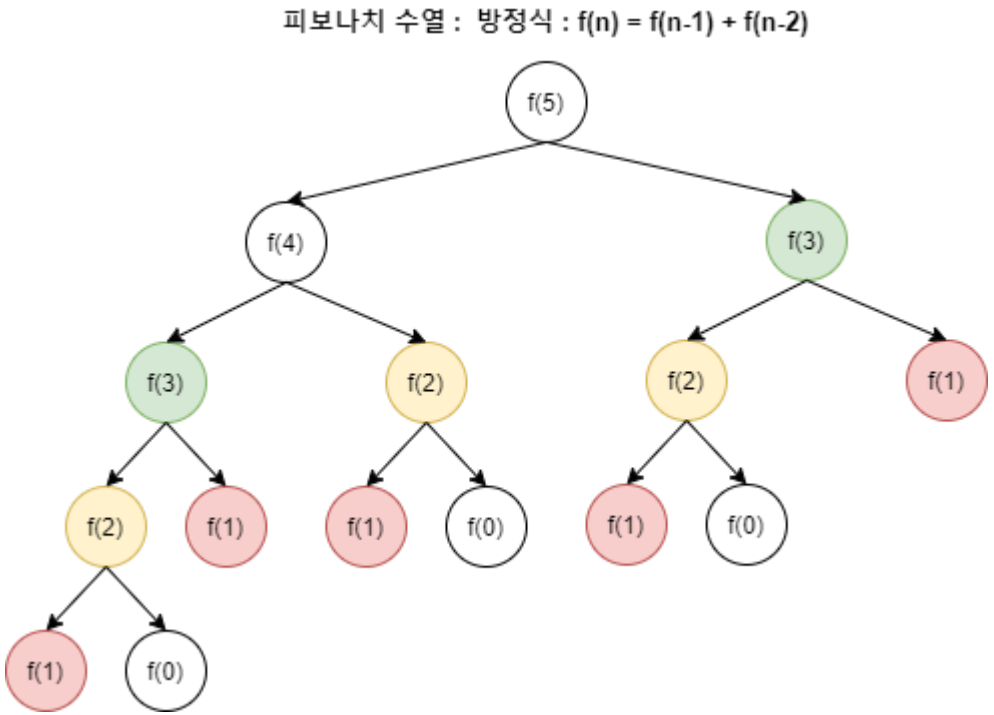
fibo_list = [1, 1, 2]
이 배열을 가지고, 피보나치 수열을 계산해보자.

n은 2부터 시작해서, 5번째 피보나치 수열의 값을 계산한다고 해보자.

피보나치 수열의 2번째 항은, 1번째 항과 0번째 항의 합과 같다.

이를 수식으로 표현하면,
fibo_list[2] = fibo_list[1] + fibo_list[0]이 된다.

따라서, fibo_list[2] = 2가 되고, 이를 fibo_list에 저장한다.



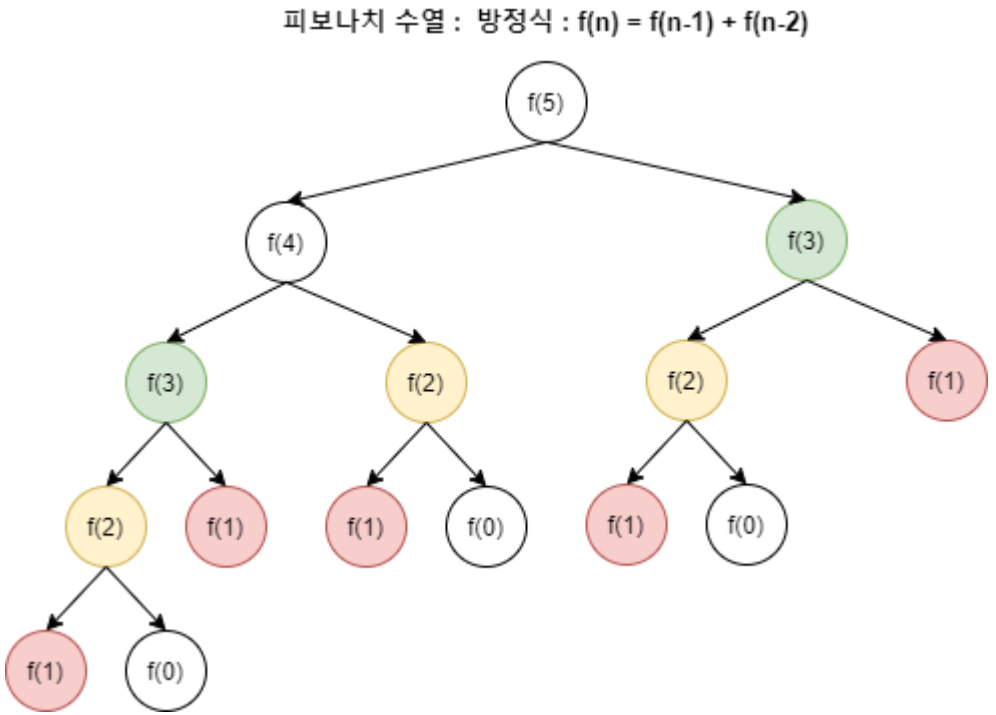
예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1, 2]

앞의 과정처럼, 3번, 4번, 5번 항도 계산해볼 수 있다.



예제 : 피보나치 수열

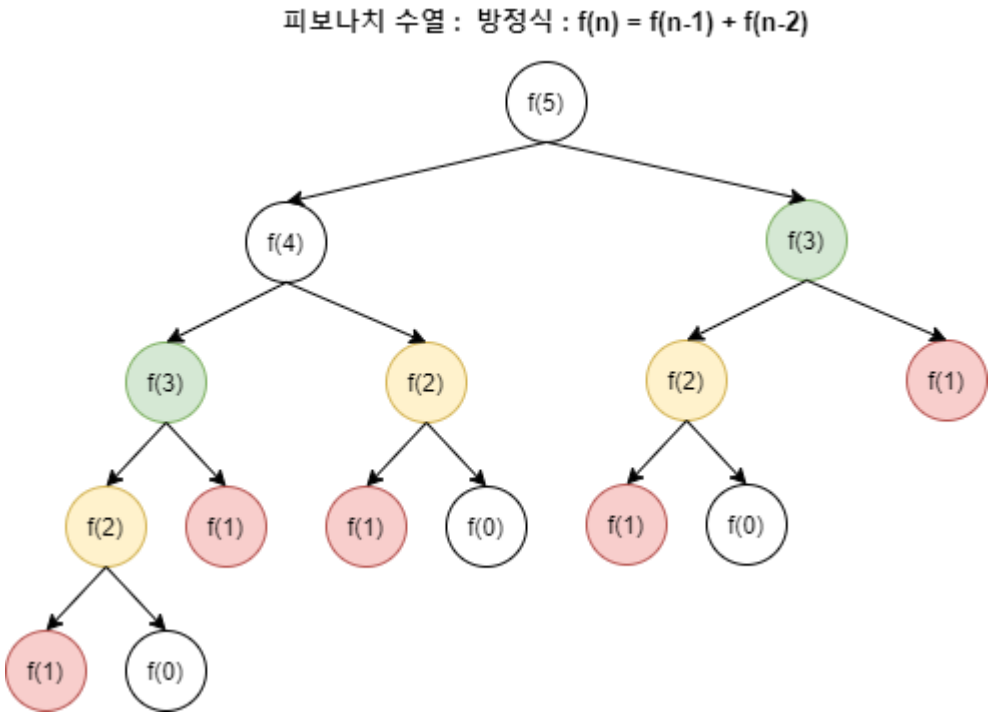
- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1, 2, 3]

앞의 과정처럼, 3번, 4번, 5번 항도 계산해볼 수 있다.

fibo_list[3] = fibo_list[2] + fibo_list[1] = 3



예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

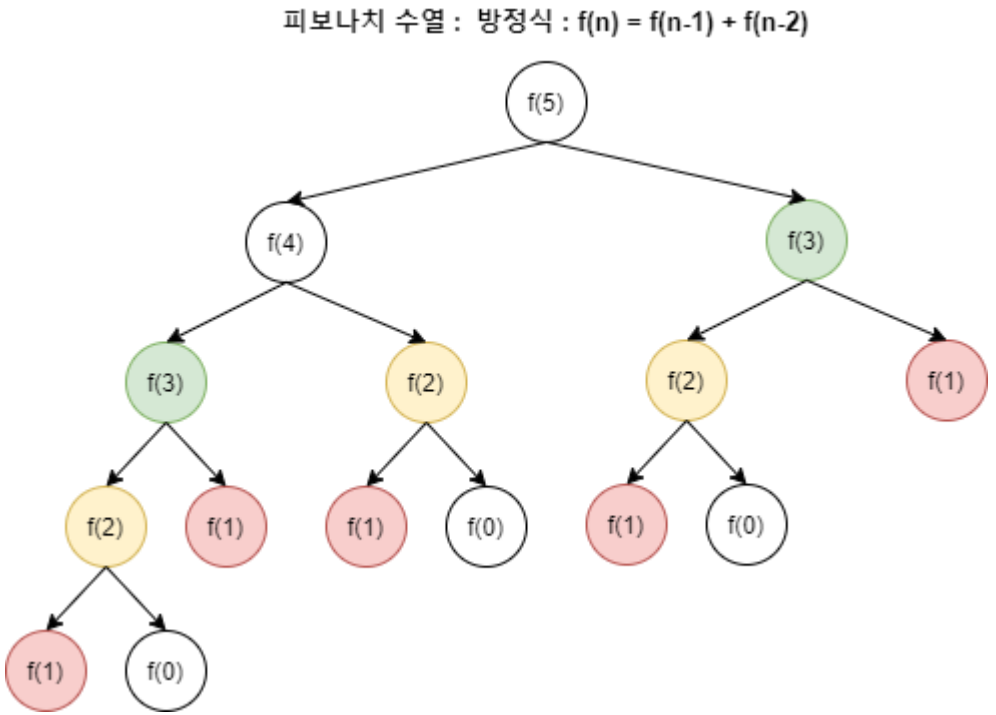
이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1, 2, 3, 5]

앞의 과정처럼, 3번, 4번, 5번 항도 계산해볼 수 있다.

fibo_list[3] = fibo_list[2] + fibo_list[1] = 3

fibo_list[4] = fibo_list[3] + fibo_list[2] = 5



예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

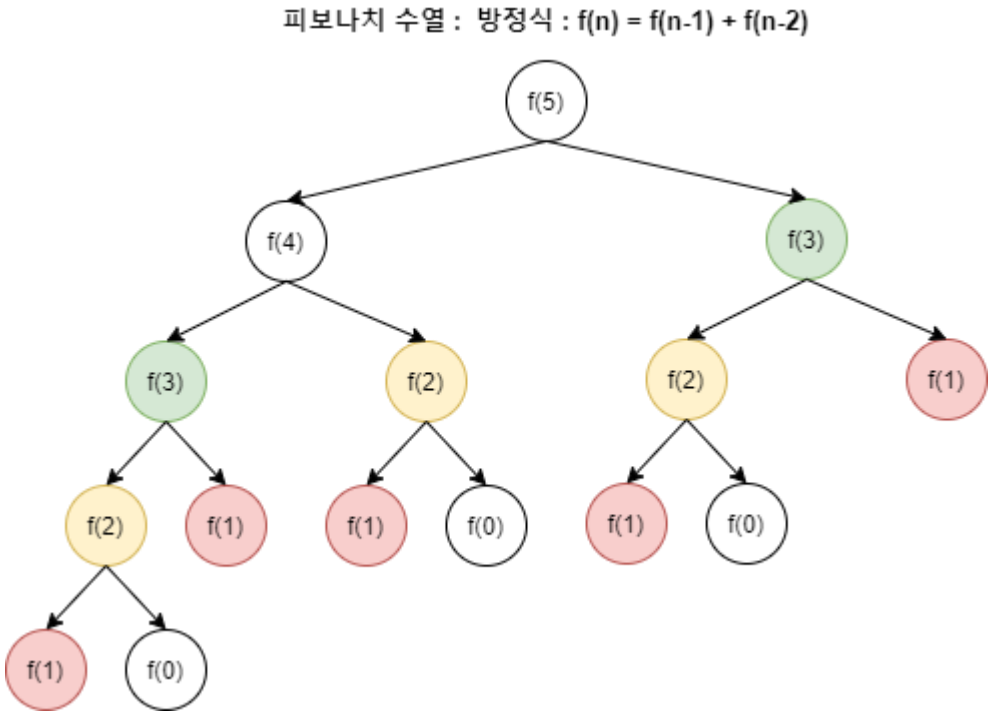
fibo_list = [1, 1, 2, 3, 5]

앞의 과정처럼, 3번, 4번, 5번 항도 계산해볼 수 있다.

fibo_list[3] = fibo_list[2] + fibo_list[1] = 3

fibo_list[4] = fibo_list[3] + fibo_list[2] = 5

fibo_list[5] = fibo_list[4] + fibo_list[3] = 8



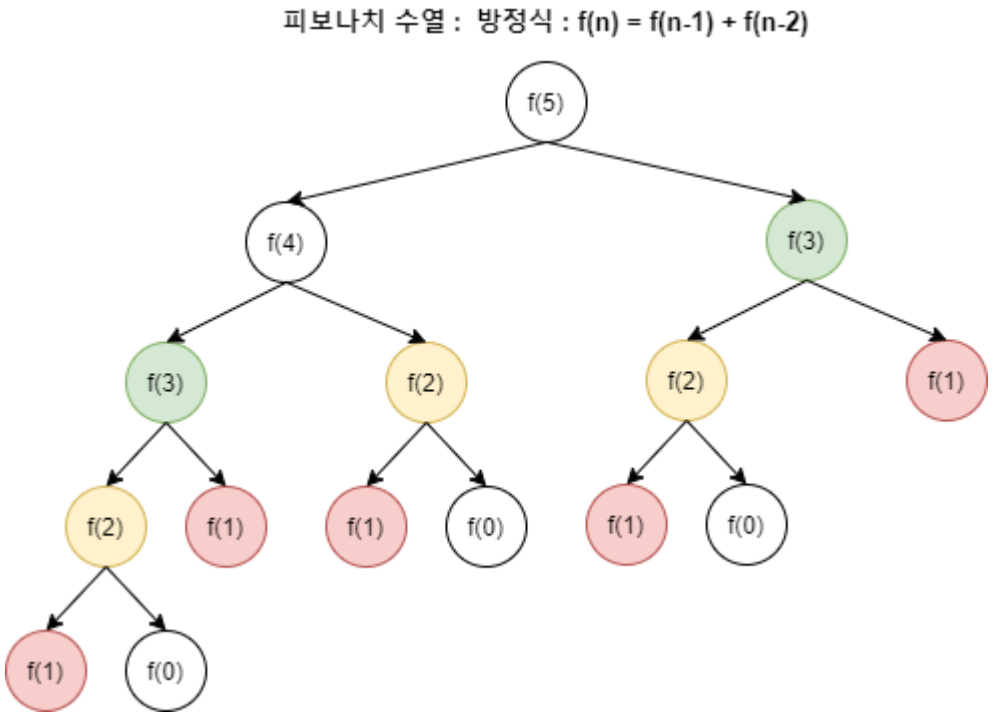
예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1, 2, 3, 5, 8]

이 과정에서 직관적으로 알 수 있듯이, DP는



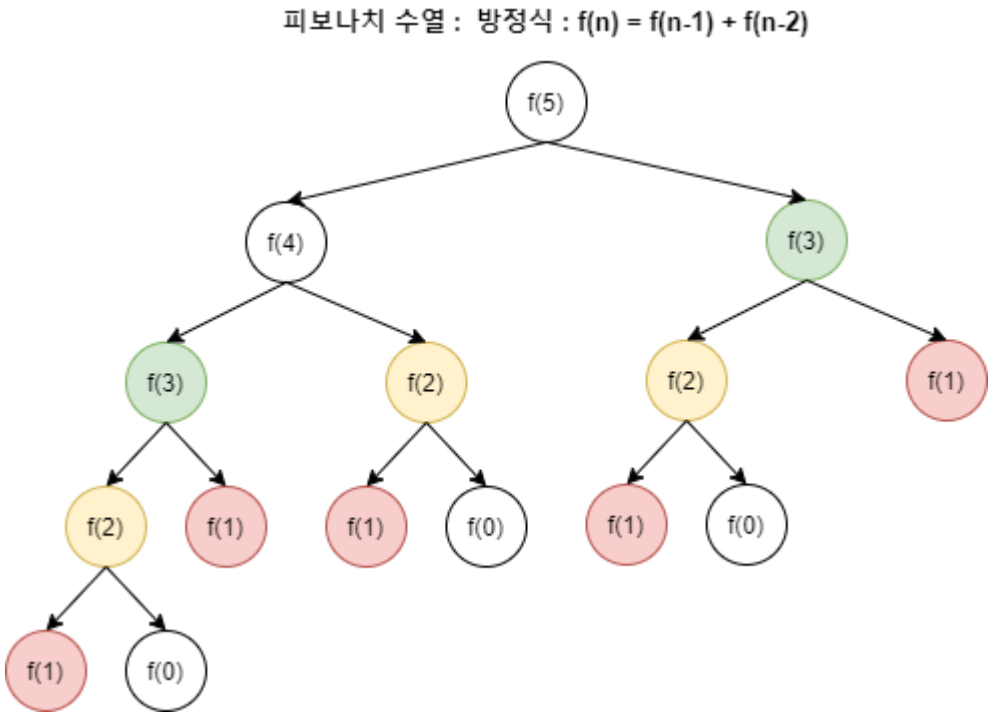
예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1, 2, 3, 5, 8]

이 과정에서 직관적으로 알 수 있듯이, DP는
다음 항을 계산하기 위해 이미 계산된 것들을 **꺼내와서**,



예제 : 피보나치 수열

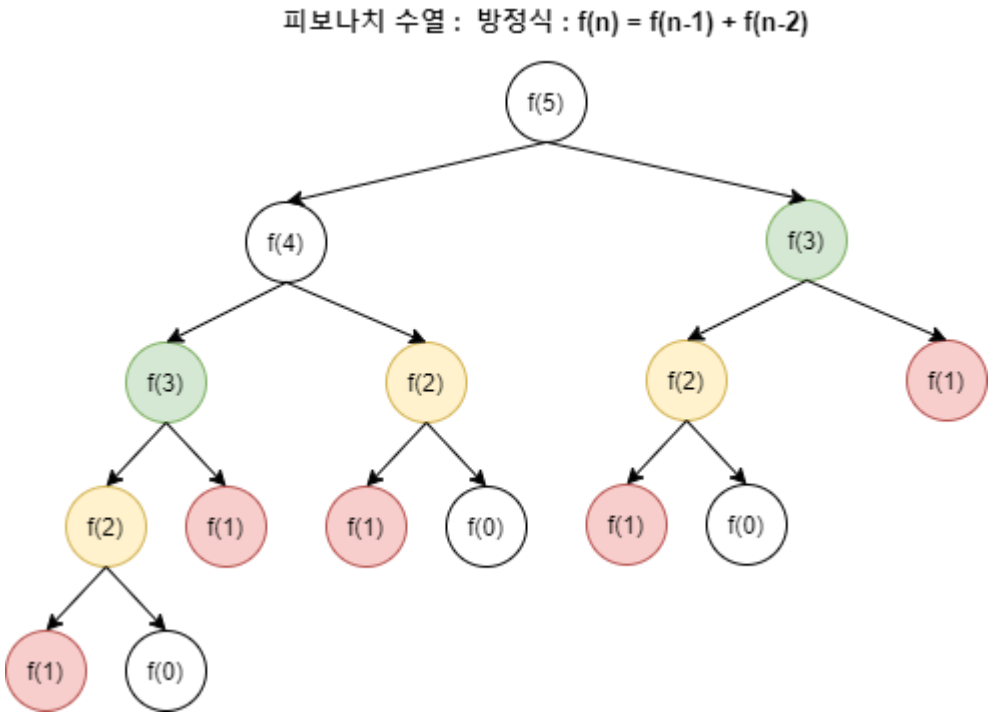
- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1, 2, 3, 5, 8]

이 과정에서 직관적으로 알 수 있듯이, DP는
다음 항을 계산하기 위해 이미 계산된 것들을 **꺼내와서**,

계산을 수행하고



예제 : 피보나치 수열

- 피보나치 수열을 재귀함수로 구현
- 피보나치 수열을 동적계획법으로 구현 (계산된 결과를 저장해놓고, 꺼내서 사용)
- 시간비교

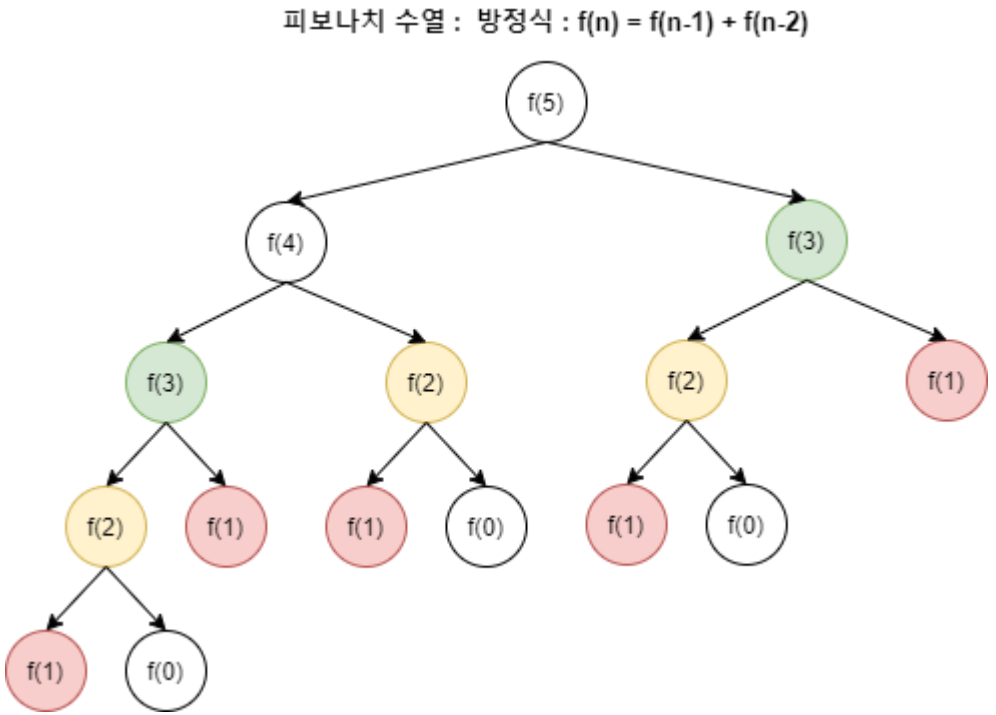
이를 피보나치 수열에 적용해보자.

fibo_list = [1, 1, 2, 3, 5, 8]

이 과정에서 직관적으로 알 수 있듯이, DP는
다음 항을 계산하기 위해 이미 계산된 것들을 **꺼내와서**,

계산을 수행하고

계산된 결과를 **저장한다는 것**이 핵심이다.



실전 : 정수 삼각형 (Lv.3)

https://school.programmers.co.kr/learn/courses/30/lessons/43105?language=python3

실전 : N으로 표현 (Lv.3)

https://school.programmers.co.kr/learn/courses/30/lessons/42895