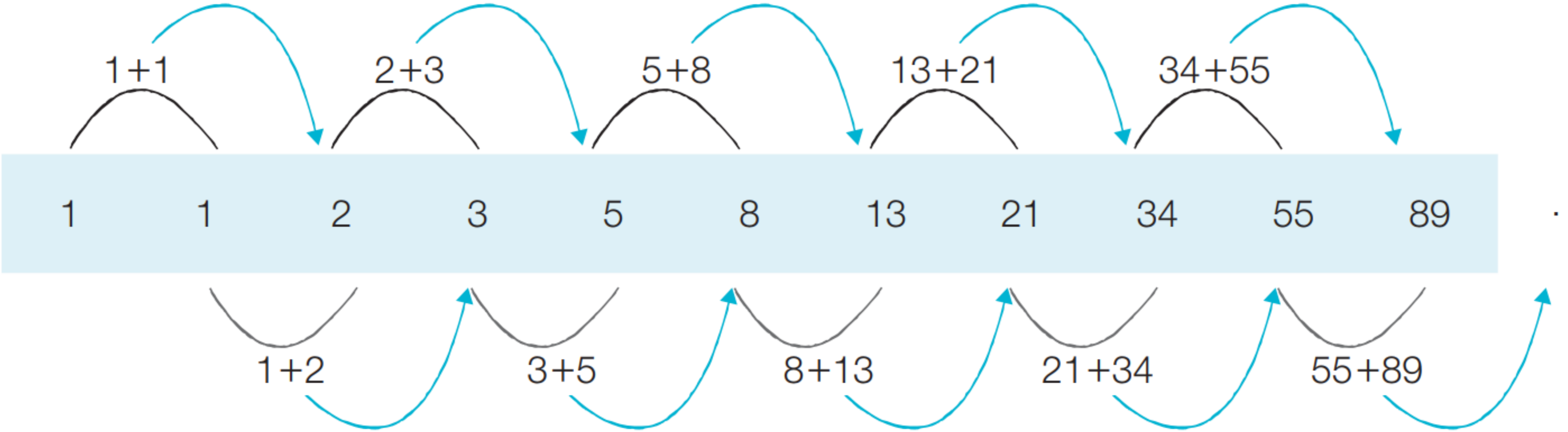
피보나치 수열

- 피보나치 수열 다음과 같은 형태의 수열이며, 다이나믹 프로그래밍으로 효과적으로 계산할 수 있다.

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

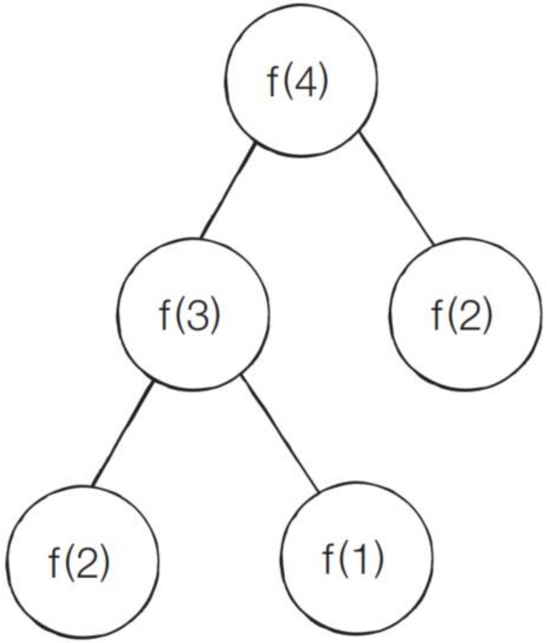
- 점화식이란 인접한 항들 사이의 관계식을 의미한다.
- 피보나치 수열을 점화식으로 표현 $a_n = a_{n-1} + a_{n-2},\ a_1 = 1,\ a_2 = 1$



- 프로그래밍에서는 이러한 수열을 배열이나 리스트를 이용해 표현 가능

피보나치 수열

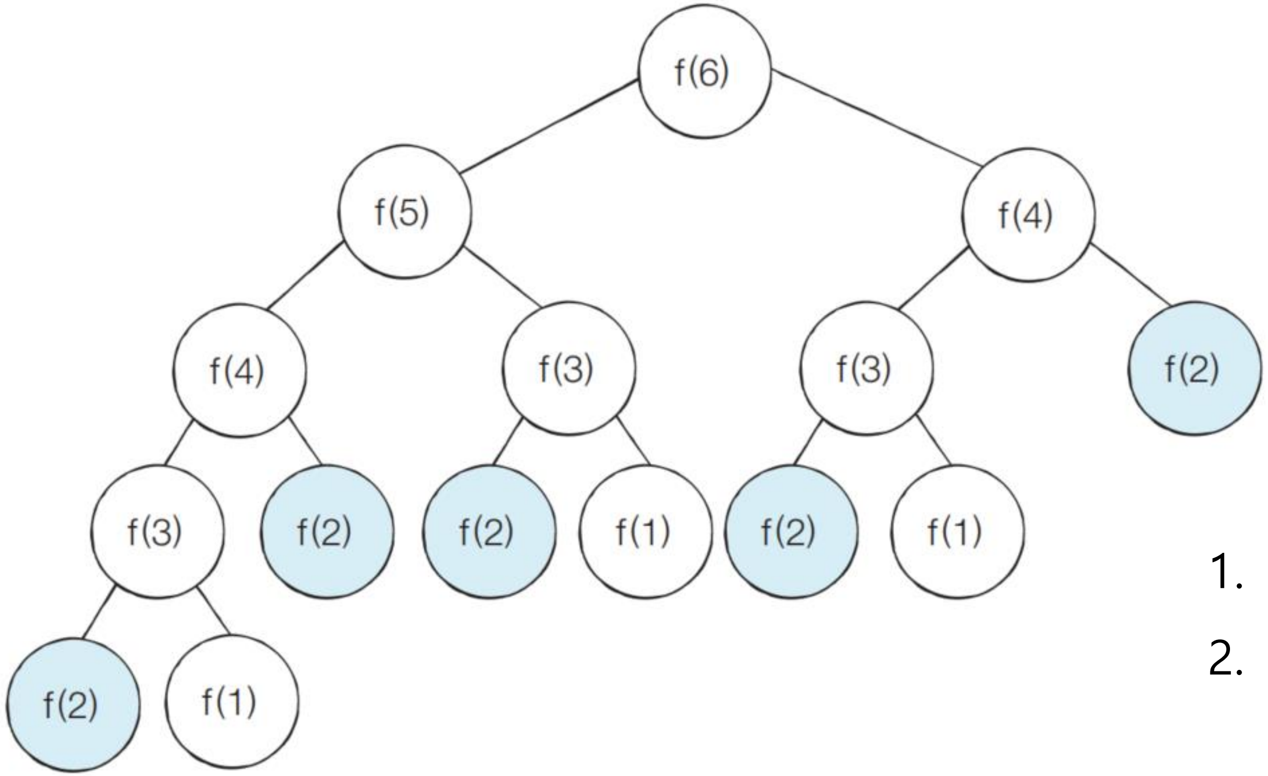
- n번째 피보나치 수를 $f(n)$라고 할 때 4번째 피보나치 수 $f(4)$를 구하는 과정




```
# 피보나치 함수(Fibonacci Function)을 재귀함수로 구현
def fibo(x):
    if x == 1 or x == 2:
        return 1
    return fibo(x - 1) + fibo(x - 2)

print(fibo(4))
```

피보나치 수열의 시간 복잡도 분석

- 단순 재귀 함수로 피보나치 수열을 해결하면 지수 시간 복잡도?
- 다음과 같이 $\boldsymbol{f(2)}$가 <u>여러 번 호출</u>되는 것을 확인 (중복되는 부분 문제)



- 피보나치 수열의 시간 복잡도
  - 세타 표기법: $\theta(1.618\cdots^{N})$
  - 빅오 표기법: $O(2^N)$
- 빅오 표기법을 기준으로 $f(30)$을 계산하기 위해 약 10억가량의 연산을 수행
- 그렇다면 $f(100)$을 계산하기 위해 얼마나 많은 연산을 수행해야 할까?

1. 최적 부분 구조: 큰 문제를 작은 문제로 나눌 수 있다.
2. 중복되는 부분 문제: 동일한 작은 문제를 반복적으로 해결한다.

=>피보나치 수열은 다이나믹 프로그래밍의 사용 조건을 만족

메모이제이션 (Memoization)

- 메모이제이션은 다이나믹 프로그래밍을 구현하는 방법 중 하나
- 한 번 계산한 결과를 메모리 공간에 메모하는 기법
  - 같은 문제를 다시 호출하면 메모했던 결과를 그대로 가져온다.
  - 값을 기록해 놓는다는 점에서 캐싱(Caching)이라고도 한다.

- 탑다운(메모이제이션) 방식은 하향식이라고도 하며 보텀업 방식은 상향식이라고도 한다.
- 다이나믹 프로그래밍의 전형적인 형태는 보텀업 방식
- 엄밀히 말하면 메모이제이션은 이전에 계산된 결과를 일시적으로 기록해 놓는 넓은 개념
  - 따라서 메모이제이션은 다이나믹 프로그래밍에 국한된 개념은 아니다.
  - 한 번 계산된 결과를 담아 놓기만 하고 다이나믹 프로그래밍을 위해 활용하지 않을 수도 있다.

피보나치 수열: 탑다운 다이나믹 프로그래밍 소스코드

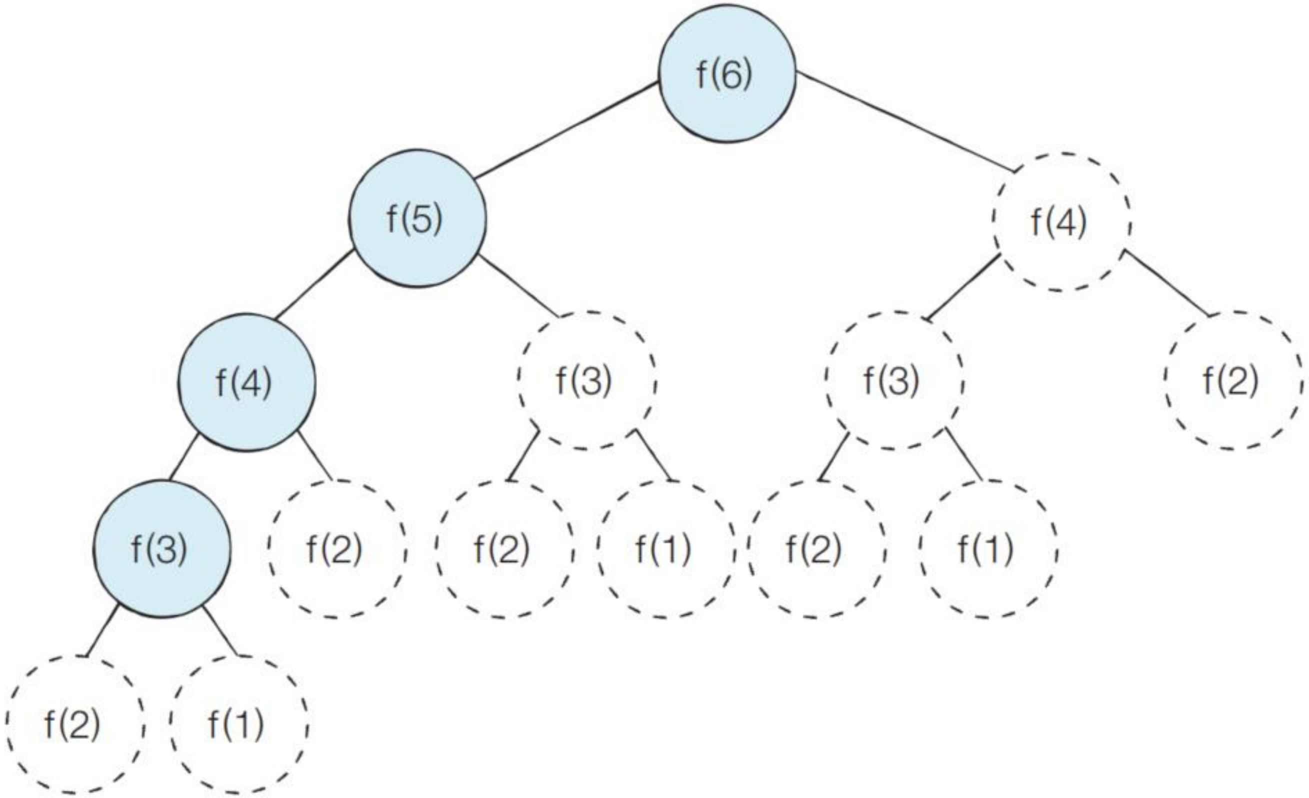
```
# 한 번 계산된 결과를 메모이제이션(Memoization)하기 위한 리스트 초기화
d = [0] * 100

# 피보나치 함수(Fibonacci Function)를 재귀함수로 구현(탑다운 다이나믹 프로그래밍)
def fibo(x):
    # 종료 조건(1 혹은 2일 때 1을 반환)
    if x == 1 or x == 2:
        return 1
    # 이미 계산한 적 있는 문제라면 그대로 반환
    if d[x] != 0:
        return d[x]
    # 아직 계산하지 않은 문제라면 점화식에 따라서 피보나치 결과 반환
    d[x] = fibo(x - 1) + fibo(x - 2)
    return d[x]

print(fibo(99))
```

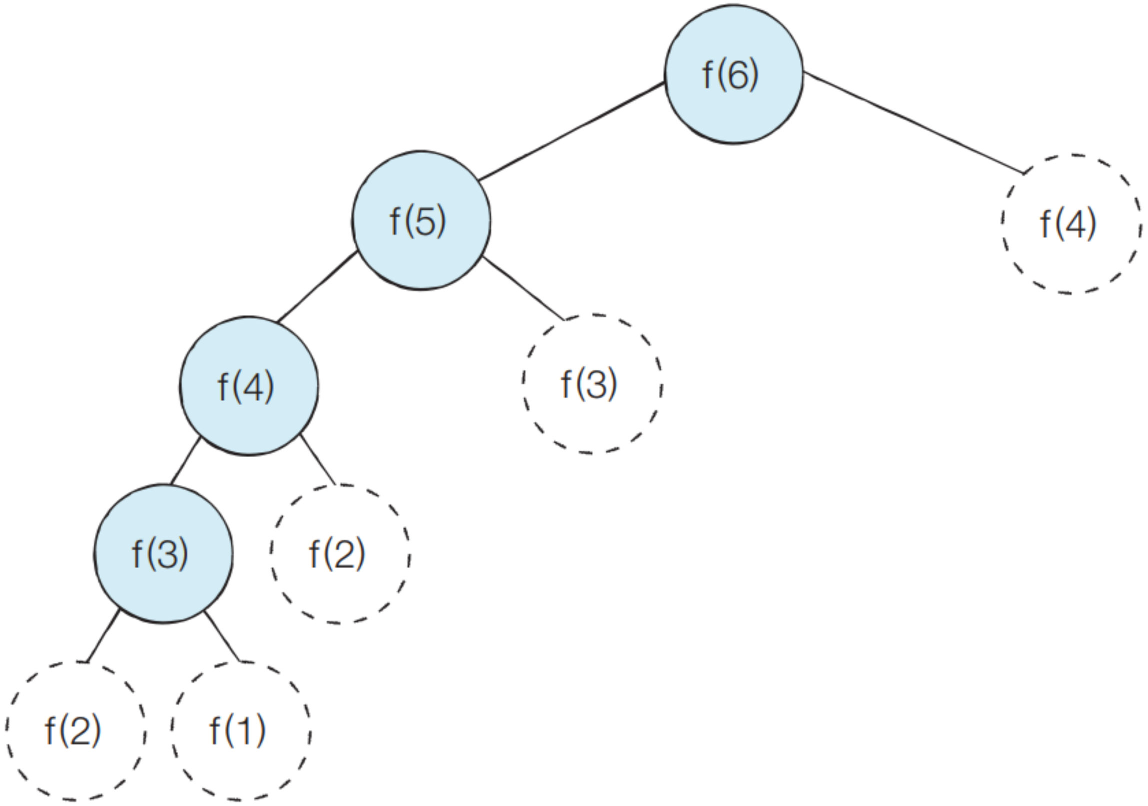
실행 결과

```
218922995834555169026
```

피보나치 수열: 메모이제이션 동작 분석

- 이미 계산된 결과를 메모리에 저장하면 다음과 같이 색칠된 노드만 처리할 것을 기대할 수 있다.





피보나치 수열: 메모이제이션 동작 분석

- 메모이제이션을 이용하는 경우 피보나치 수열 함수의 시간 복잡도는 $O(N)$

```
d = [0] * 100

def fibo(x):
    print('f(' + str(x) + ')', end=' ')
    if x == 1 or x == 2:
        return 1
    if d[x] != 0:
        return d[x]
    d[x] = fibo(x - 1) + fibo(x - 2)
    return d[x]

fibo(6)
```

```
f(6) f(5) f(4) f(3) f(2) f(1) f(2) f(3) f(4)
```