Dynamic Programming

동적 계획법

출처- 이것이 취업을 위한 코딩 테스트다 with 파이썬 (나동빈 저)

- 메모리를 적절히 사용하여 수행 시간 효율성을 비약적으로 향상시키는 방법

- 이미 계산된 결과(작은문제)는 별도의 영역에 저장하여 다시 계산하지 않는다.

  “한 번 계산해서 해결한 문제는 다시 해결하지 않는다!”

- 일반적으로 두 가지 방식(Top Down, Bottom Up)으로 구성된다.

조건

1. **최적 부분 구조 (Optimal Substructure)**

 큰 문제를 작은 문제로 나눌 수 있으며 작은 문제의 답을 모아서 큰 문제 해결 가능

2. **중복되는 부분 문제 (Overlapping Subproblem)**

 동일한 작은 문제를 반복적으로 해결해야 한다.

피보나치 수열

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 …

- 피보나치 수열을 점화식으로 표현

$$a_n = a_{n-1} + a_{n-2},\ a_1 = 1,\ a_2 = 1$$

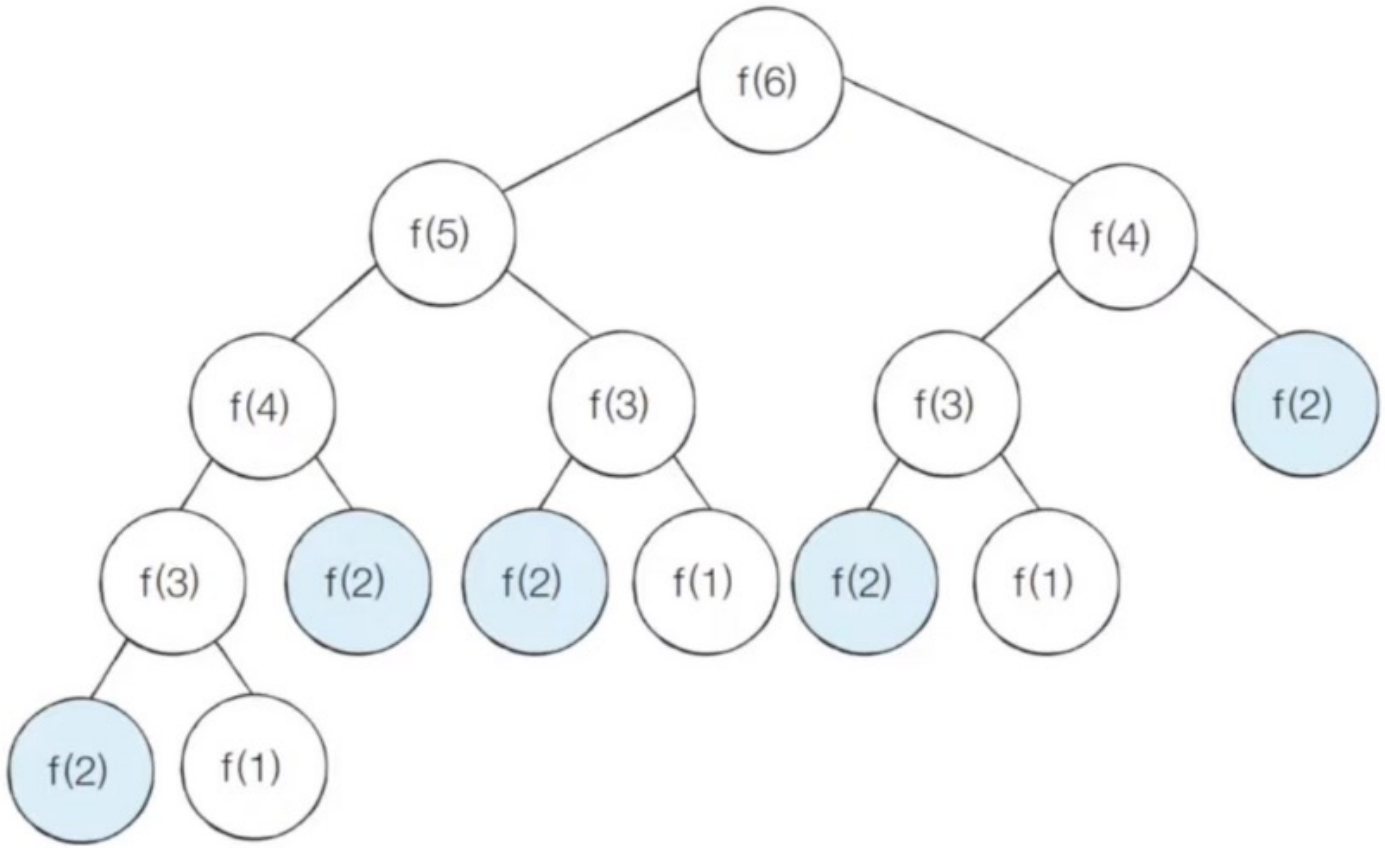
* 점화식이란 인접한 항들 사이의 관계식을 의미

* 프로그래밍에서 수열은 배열이나 리스트를 이용해 표현

단순 재귀 코드

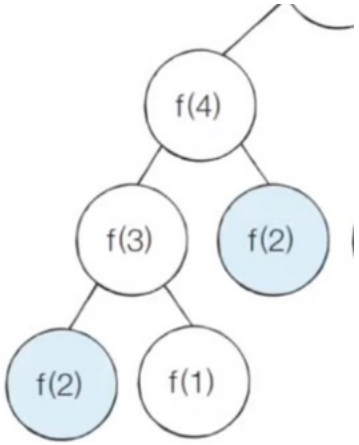
```
def fibo(x):
        if x == 1 or x == 2:
                return 1
        return fibo(x – 1) + fibo(x – 2)
```

점화식은 재귀함수로 표현이 가능함
단순 재귀 함수로 피보나치 수열을 해결하면 O(N^2)의 시간 복잡도



f(2)가 여러 번 호출되는 것 확인 가능함
동일한 작은 문제들이 여러 번 반복 -> 비효율적인 계산

DP의 사용 조건을 만족?



최적 부분 구조 : 큰 문제를 작은 문제로 나눌 수 있는 지

중복되는 부분 문제: 동일한 작은 문제를 반복적으로 해결하는 지

Memoization

- DP를 구현하는 방법 중 하나

- 한 번 계산한 결과를 메모리 공간에 메모하는 기법
  - 같은 문제를 다시 호출하면 메모했던 결과를 그대로 가져온다.
  - 값을 기록한다는 점에서 캐싱(Caching)이라고도 한다.

  “이전에 계산된 결과를 일시적으로 기록해 놓는 넓은 개념”

  * 값을 기록하는 배열의 변수명: 보통 cash, table, dp, d.. 를 사용

Top-Down(하향식) vs Bottom-Up(상향식)

- Top Down : Memoization – 재귀 함수 사용
- Bottom Up: 반복문 사용

- DP의 전형적인 형태는 BottomUp 방식
  - 결과 저장용 리스트는 DP 테이블 이라 부름

소스코드 비교

```python
# 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
```

Top Down

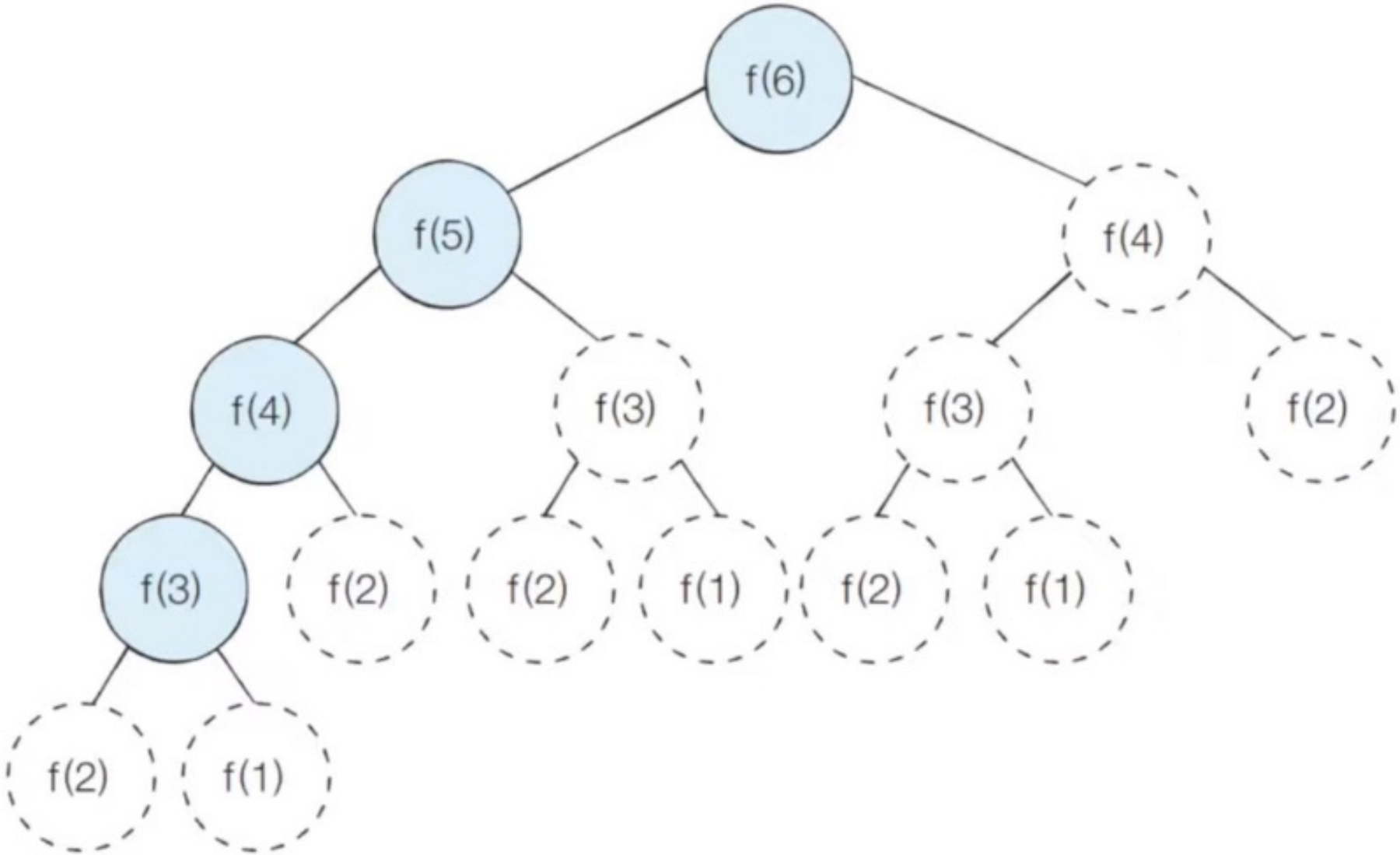
```python
# 앞서 계산된 결과를 저장하기 위한 DP 테이블 초기화
d = [0] * 100

# 첫 번째 피보나치 수와 두 번째 피보나치 수는 1
d[1] = 1
d[2] = 1
n = 99

# 피보나치 함수(Fibonacci Function) 반복문으로 구현(보텀업 다이나믹 프로그래밍)
for i in range(3, n + 1):
    d[i] = d[i - 1] + d[i - 2]

print(d[n])
```

Bottom Up


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

DP vs 분할 정복

- 공통점: 최적 부분 구조를 가질 때 사용할 수 있다.

- 차이점: 부분 문제의 중복
  - DP에서는 각 부분 문제들이 서로 영향을 미치며 부분 문제가 중복됨
  - 분할 정복 문제에서는 동일한 부분 문제가 반복적으로 계산되지 않는다.

- **분할 정복**의 대표적인 예시인 퀵 정렬을 살펴봅시다.
  - 한 번 기준 원소(Pivot)가 자리를 변경해서 자리를 잡으면 그 기준 원소의 위치는 바뀌지 않습니다.
  - 분할 이후에 해당 피벗을 다시 처리하는 부분 문제는 호출하지 않습니다.

분할이 이루어진 뒤에 피벗인 5의 위치는 더 이상 바뀌지 않는다. => 부분 문제가 중복되지 않는다.

문제 접근 방법

- 주어진 문제가 DP 유형임을 파악하자!
- 그리디, 구현, 완전 탐색 등의 아이디어로 문제를 해결할 수 있는지 검토 -> 떠오르지 않는다면 DP를 고려해본다.

- 일단 재귀 함수로 비효율적인 완전 탐색 프로그램을 작성한 뒤에(TopDown) 작은 문제에서 구한 답이 큰 문제에서 그대로 사용될 수 있으면 코드를 개선한다.