

Dynamic Programming

최혁태



나동빈

이것이 취업을 위한 코딩 테스트다 (2020)

다이나믹 프로그래밍

- 메모리 공간을 사용해서 수행 시간 효율성을 비약적으로 향상시키는 방법
- 이미 계산된 결과는 별도의 메모리 공간에 저장하여 다시 계산하지 않는다
- 다이나믹 프로그래밍은 두가지 방식(top-down , bottom-up)으로 구성된다

다이나믹 프로그래밍의 조건

다이나믹 프로그래밍은 문제가 다음의 조건을 만족할 때 사용할 수 있다.

1. 최적 부분 구조

큰 문제를 작은 문제로 나눌 수 있으며, 작은 문제의 답을 모아서 큰 문제를 해결할 수 있을 때

2. 중복되는 부분 문제

동일한 작은 문제를 반복적으로 해결해야할 때

피보나치 수열

피보나치 수열은 다음과 같은 형태의 수열이다.

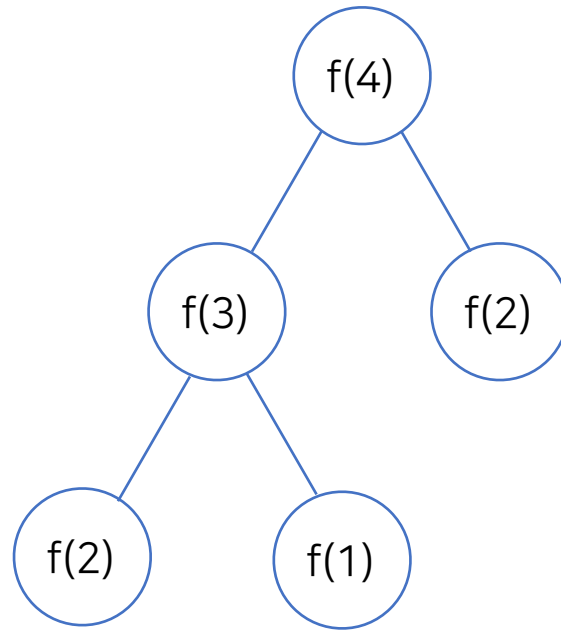
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

피보나치 수열을 점화식으로 표현하면 다음과 같다.

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

피보나치 수열

피보나치 수열이 계산되는 과정은 다음과 같이 표현할 수 있다.



단순 재귀코드

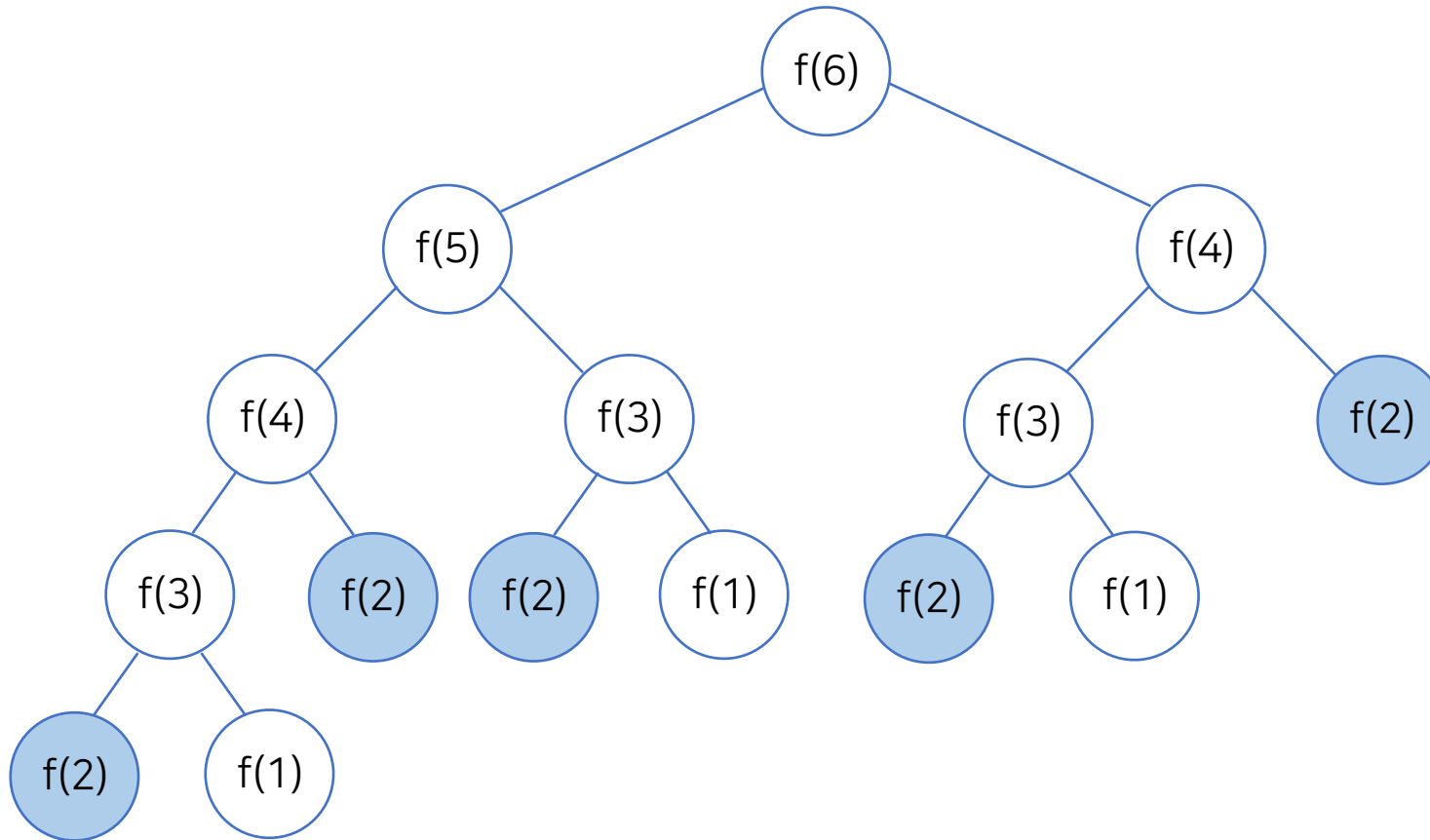
```
def fibo(x):  
    if (x == 1 or x == 2):  
        return 1  
    return fibo(x-1) + fibo(x-2)  
  
print(fibo(4))
```

실행 결과 : 3

피보나치 수열의 시간 복잡도 분석 (재귀함수 사용시)

비효율적인 시간복잡도 $O(2^n)$

f(2) 중복



다이나믹 프로그래밍을 이용한 피보나치 수열 해법

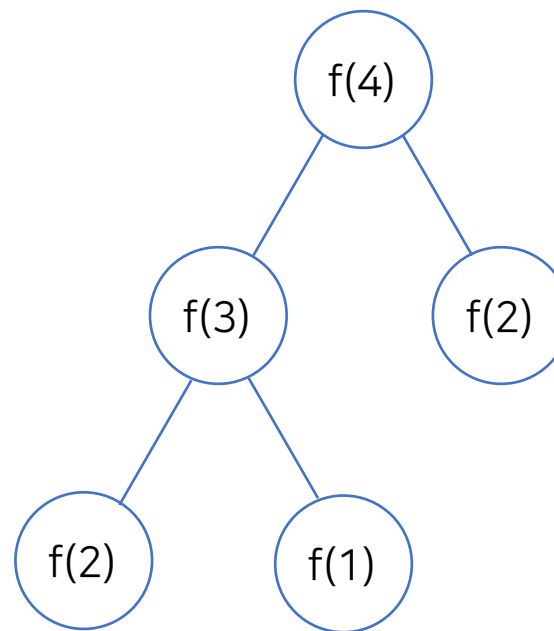
다이나믹 프로그래밍의 사용 조건을 만족하는지 확인한다.

1. 최적 부분 구조

큰 문제를 작은 문제로 나눌 수 있다.

2. 중복되는 부분 문제

동일한 작은 문제를 반복적으로 해결한다.



다이나믹 프로그래밍을 이용한 피보나치 수열 해법

다이나믹 프로그래밍의 사용 조건을 만족하는지 확인한다.

1. 최적 부분 구조

큰 문제를 작은 문제로 나눌 수 있다.



2. 중복되는 부분 문제

동일한 작은 문제를 반복적으로 해결한다.



메모이제이션 (Memoization)

- 메모이제이션은 다이나믹 프로그래밍을 구현하는 방법 중 하나로, 한 번 계산한 결과를 메모리 공간에 메모하는 기법이다.
- 같은 문제를 다시 호출하면 메모했던 결과를 그대로 가져온다.

Top-down 다이나믹 프로그래밍

```
d = [0] * 100
```

한 번 계산된 결과를 메모하기 위한 리스트

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

```
    if (d[x] != 0):  
        return d[x]
```


이미 계산한 적 있는 문제면 그대로 반환

```
    d[x] = fibo(x-1) + fibo(x-2)
```

```
    return d[x]
```

```
print(fibo(99))
```

Bottom-up 다이나믹 프로그래밍

```
  
d = [0] * 100  
  
d[1] = 1  
d[2] = 1  
n = 99  
  
for i in range(3, n+1):  
    d[i] = d[i-1] + d[i-2]  
  
print(d[n])
```

→ 앞서 계산된 결과를 저장할 dp 테이블 초기화

→ 반복문으로 구현

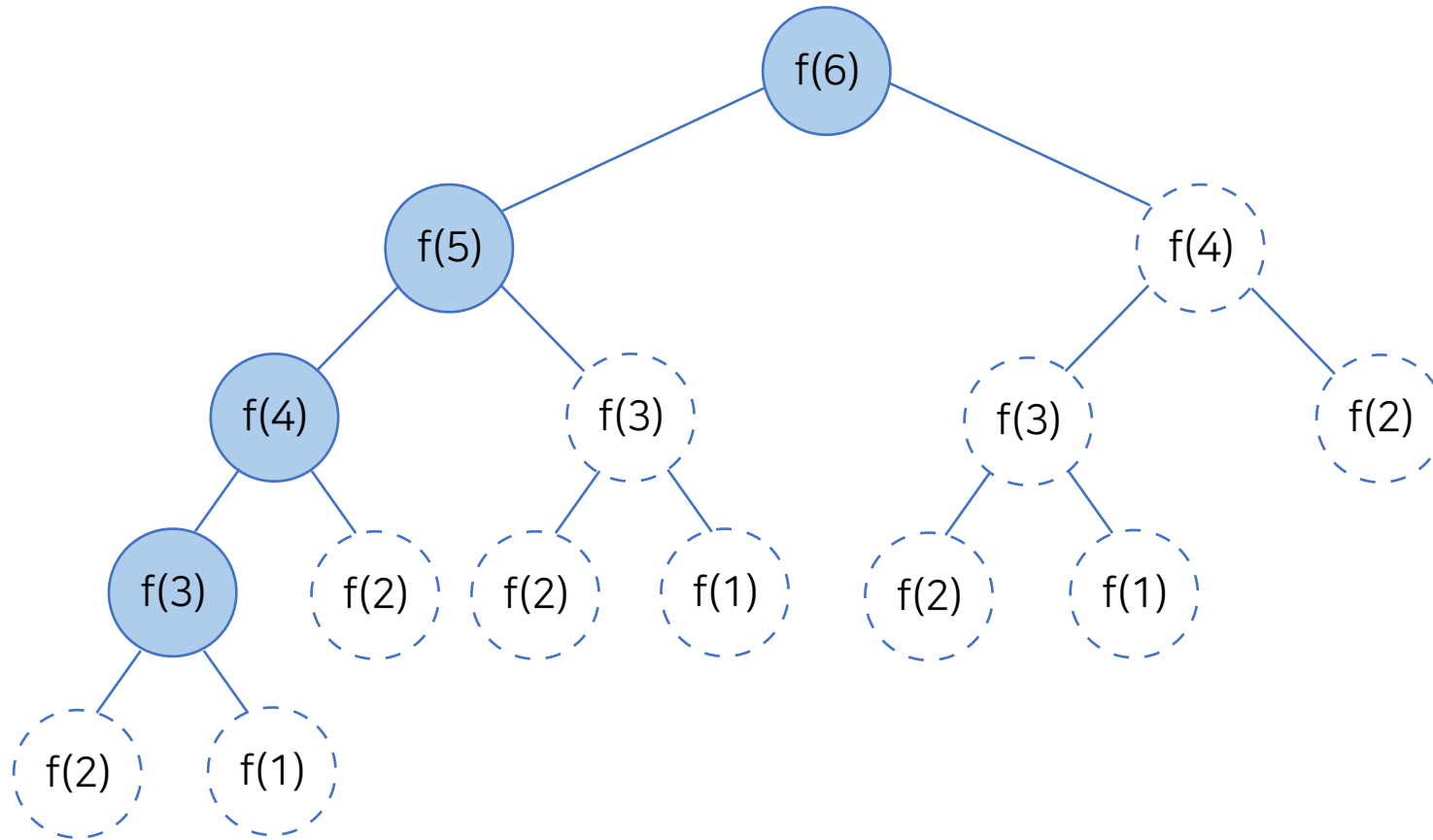
Bottom-up 다이나믹 프로그래밍

```
for i in range(3, n+1):  
    d[i] = d[i-1] + d[i-2]
```

d	1	1	2	3	5	8	13	21
---	---	---	---	---	---	---	----	----	---	---	---	---

메모이제이션 동작 분석

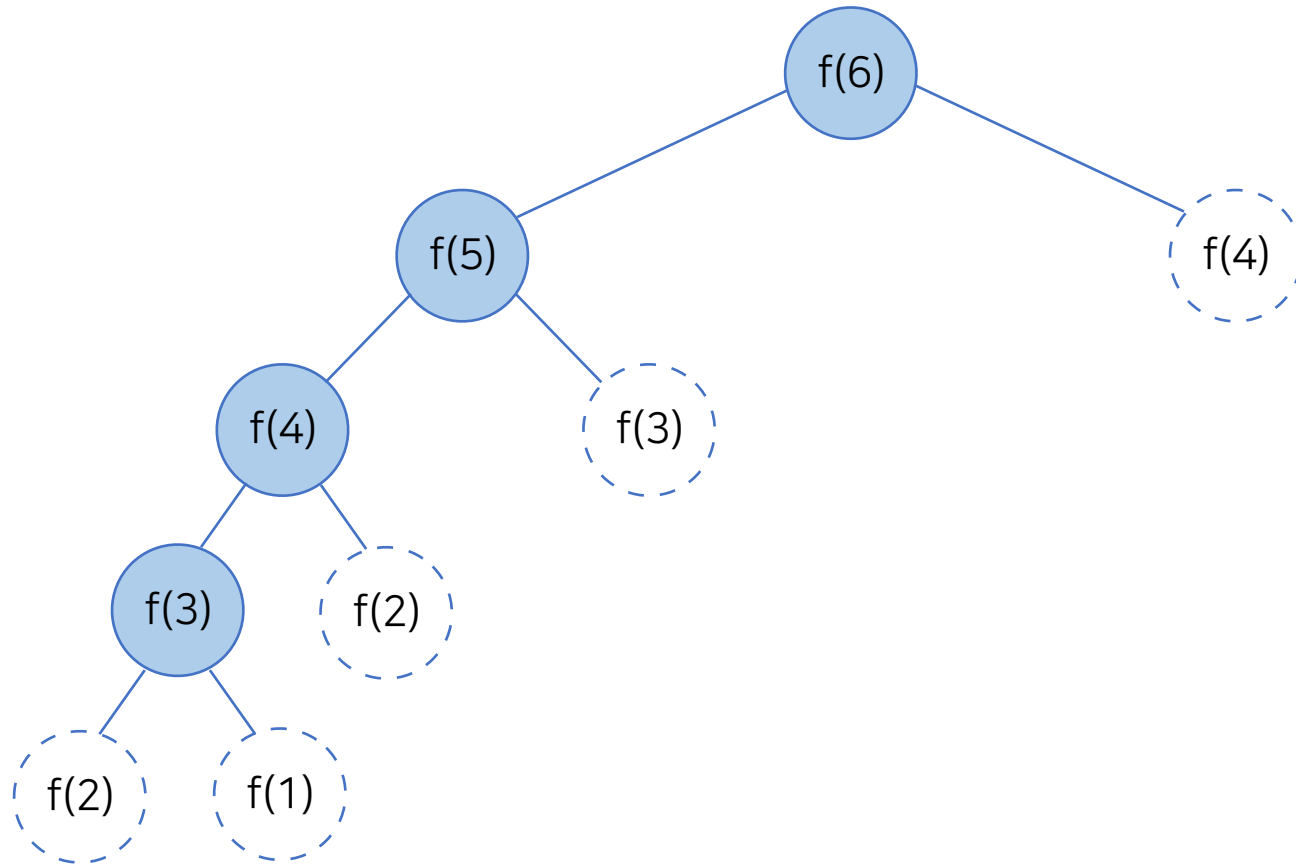
이미 계산된 결과를 리스트에 저장해놓는 경우 (메모이제이션 사용시)



메모이제이션 동작 분석

실제로 호출되는 함수에 대해서만 확인해보면 다음과 같다.

시간복잡도 : $O(N)$



Top-down vs Bottom-up

Top-down

```
● ● ●  
  
d = [0] * 100  
  
def fibo(x):  
    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]  
  
print(fibo(99))
```

Bottom-up

```
● ● ●  
  
d = [0] * 100  
  
d[1] = 1  
d[2] = 1  
n = 99  
  
for i in range(3, n+1):  
    d[i] = d[i-1] + d[i-2]  
  
print(d[n])
```

Top-down vs Bottom-up

Top-down

```
d = [0] * 100

def fibo(x):
    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]

print(fibo(99))
```

Bottom-up

```
d = [0] * 100

d[1] = 1
d[2] = 1

for i in range(3, n+1):
    d[i] = d[i-1] + d[i-2]

print(d[n])
```

시간/메모리 사용량

Top-down vs Bottom-up

Top-down

```
d = [0] * 100

def fibo(x):
    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]

print(fibo(99))
```

Bottom-up

```
d = [0] * 100

d[1] = 1
d[2] = 1
n = 99

for i in range(3, n+1):
    d[i] = d[i-1] + d[i-2]

print(d[n])
```

Top-down vs Bottom-up

Top-down

```
d = [0] * 100

def fibo(x):
    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]

print(fibo(99))
```

설계

Bottom-up

```
d = [0] * 100

d[1] = 1
d[2] = 1
n = 99

for i in range(3, n+1):
    d[i] = d[i-1] + d[i-2]

print(d[n])
```

Top-down vs Bottom-up

Top-down

```
● ● ●  
  
d = [0] * 100  
  
def fibo(x):  
    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]  
  
print(fibo(99))
```

Bottom-up

```
● ● ●  
  
d = [0] * 100  
  
d[1] = 1  
d[2] = 1  
n = 99  
  
for i in range(3, n+1):  
    d[i] = d[i-1] + d[i-2]  
  
print(d[n])
```

시간 비교

피보나치 수열의 999번째 항을 구하는데 걸리는 시간 (소수점 11자리 반올림)

Top-down

time : 0.0020308495

Bottom-up

time : 0.0009317398

시간 비교

피보나치 수열의 999번째 항을 구하는데 걸리는 시간 (소수점 11자리 반올림)

Top-down	time : 0.0020308495
----------	---------------------

약 2.18배

Bottom-up	time : 0.0009317398
-----------	---------------------

다이나믹 프로그래밍에 접근하는 방법

가장 먼저 그리디, 구현, 완전 탐색 등의 아이디어로 문제를 해결할 수 있는지 검토한다.
다른 알고리즘으로 풀이 방법이 떠오르지 않으면 다이나믹 프로그래밍을 고려한다.

재귀함수를 사용하여 비효율적인 완전 탐색 프로그램을 작성한 뒤에(탑다운),
작은 문제에서 구한 답이 큰 문제에서 그대로 사용될 수 있으면, 그 때 코드를 개선한다.