# 다이내믹 프로그래밍

이준규

# 목차

# 다이내믹 프로그래밍이란?

# 다이내믹 프로그래밍이란?

## 다이내믹 프로그래밍(Dynamic Programming)

하나의 문제를 단 한 번만 풀도록(Memoization)하여, 메모리를 적절히 사용해 수행 시간 효율성을 비약적으로 향상시키는 알고리즘이다.

일반적으로 상당수 분할 정보 기법은 동일한 문제를 다시 푼다는 단점이 있다.
이로 인해 심각한 비효율성을 낳는 대표적인 예시가 피보나치 수열이 있다.

ex ) 피보나치 수열의 점화식 : d[i] = d[i-1] + d[i-2]
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

# 다이내믹 프로그래밍 조건

## 부분 반복 문제(Overlapping Subproblem)

어떤 문제가 여러 개의 부분 문제로 쪼개질 수 있을 때 적용 가능하다.
모든 문제를 부분으로 쪼개 재귀 함수를 통해 해결가능하다.

cf ) 부분 문제 : 계속해서 여러 번 재사용 되거나 재귀 알고리즘으로 통해 해결되는 문제

## 최적 부분 구조(Optimal Substructure)

작은 부분 문제에서 구한 최적의 답을 통해 큰 문제의 최적의 답을 구할 수 있어야 한다.

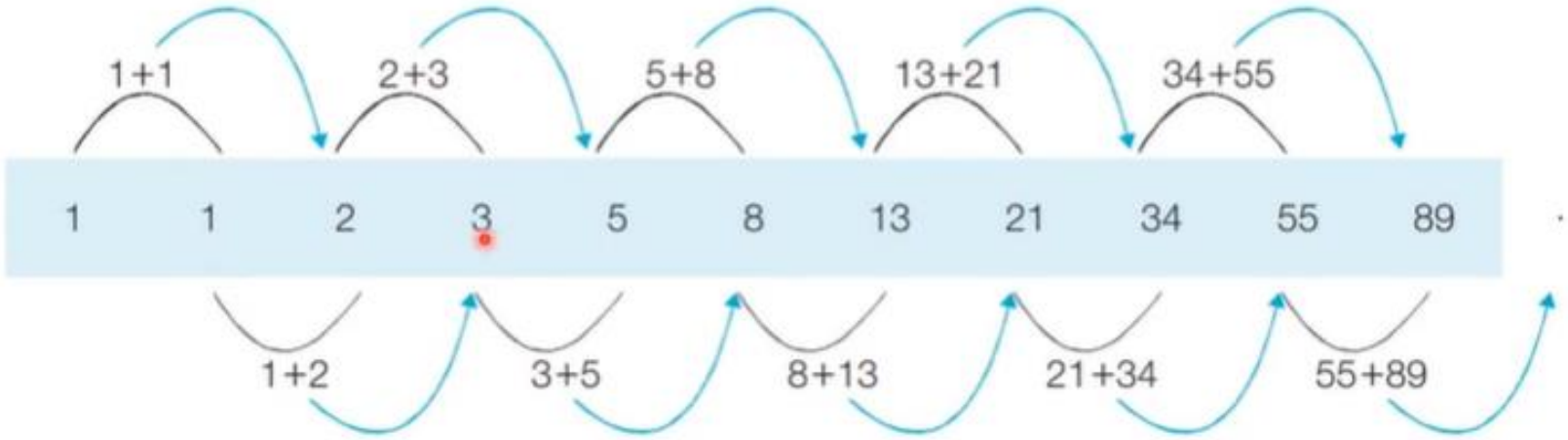
이 경우 문제의 크기에 상관 없이 어떤 작은 부분 문제는 항상 반복된 연산이 된다.

# 메모이제이션

# 메모이제이션

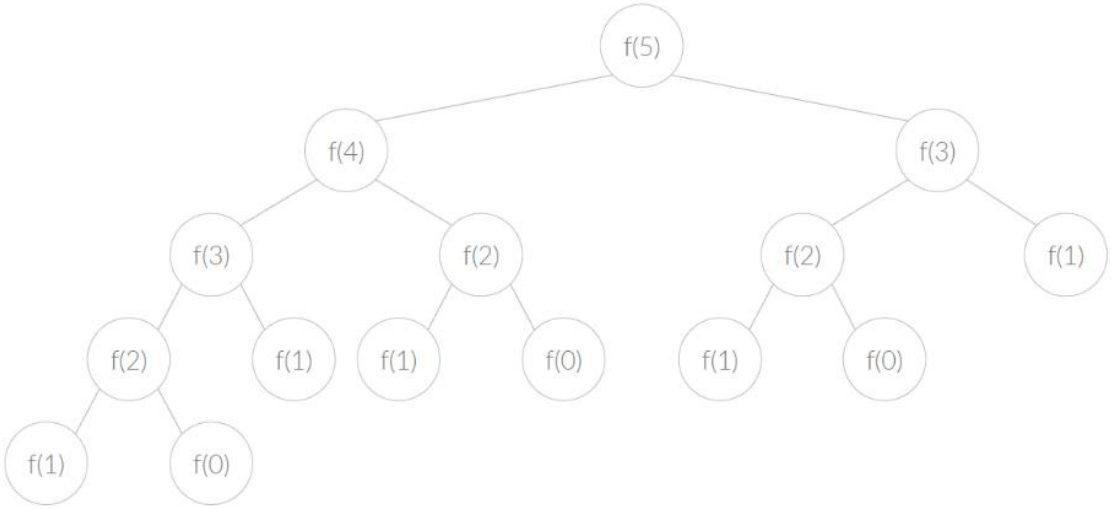
## 메모이제이션(Memoization)

다이나믹 프로그래밍 구현 방법 중 하나로 중복 계산을 방지하기 위해 한 번 계산한 결과를 배열이라는 메모리 공간에 저장(메모)해놓는 방법. 캐싱(Caching)이라고도 함. ( 파이썬에서는 리스트 사용)



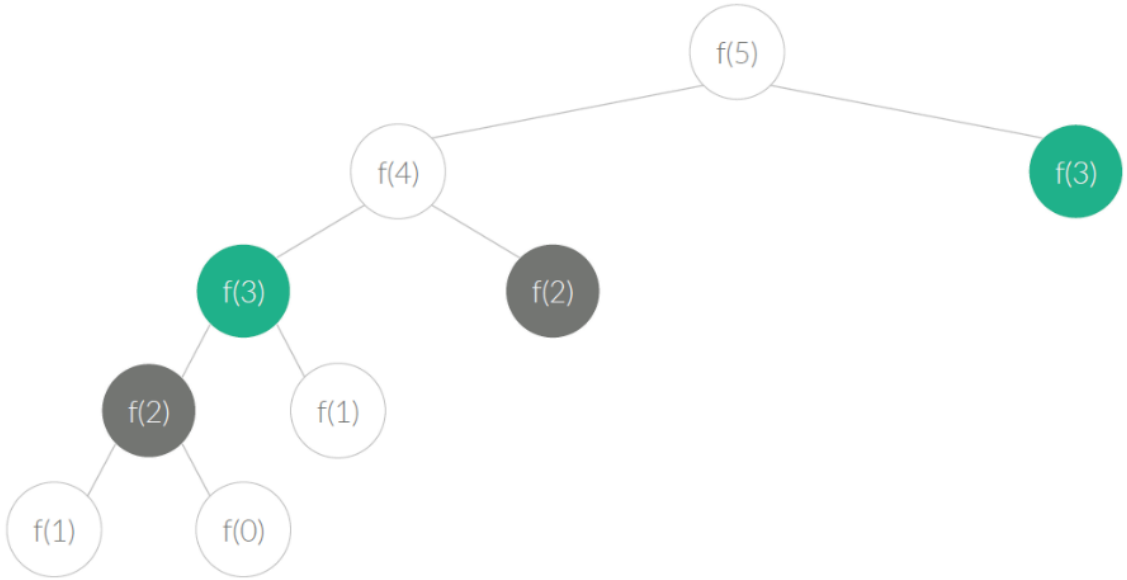
# 메모이제이션 사용 유무에 따른 시간 복잡도 분석

구하고자 하는 n 값이 커질수록 계산속도차는 더욱 커진다.




메모이제이션 미사용시

시간복잡도 : $O(2^n)$




메모이제이션 사용시

시간복잡도 : $O(n)$

# Top-down vs Bottom-up

# Top-down vs Bottom-up

**두 방법 모두 큰 문제를 여러 개의 부분 문제들로 나누어 푸는건 동일함.**

### Top-down 방식

가장 큰 문제를 방문 후 작은 문제를 호출하여 답을 찾는 방식, 재귀로 구현한다.

큰 문제를 작은 문제로 나눈다 -> 작은 문제를 푼다 -> 작은 문제를 풀었으니, 이제 큰 문제를 푼다.

### Bottom-up 방식

가장 작은 문제들부터 답을 구해가며 전체의 문제의 답을 찾는 방식, 반복문으로 구현한다.

크기가 작은 문제부터 차례대로 푼다 -> 문제의 크기를 늘려가며 푼다 -> 작은 문제를 풀었듯이 큰 문제는 항상 풀 수 있다

-> 반복하면 결국에는 가장 큰 문제를 풀 수 있다.

# Top-down vs Bottom-up

**Top-down 방식**

가장 큰 문제를 방문 후 작은 문제를 호출하여 답을 찾는 방식, 재귀로 구현한다.

큰 문제를 작은 문제로 나눈다 -> 작은 문제를 푼다 -> 작은 문제를 풀었으니, 이제 큰 문제를 푼다.

```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 vs Bottom-up

**Bottom-up 방식**

가장 작은 문제들부터 답을 구해가며 전체의 문제의 답을 찾는 방식, 반복문으로 구현한다.

크기가 작은 문제부터 차례대로 푼다 -> 문제의 크기를 늘려가며 푼다 -> 작은 문제를 풀었듯이 큰 문제는 항상 풀 수 있다

-> 반복하면 결국에는 가장 큰 문제를 풀 수 있다.

```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])
```

실행 결과

```
218922995834555169026
```

# 다이나믹 프로그래밍 문제에 접근하는 방법

- 주어진 문제가 **다이나믹 프로그래밍 유형임을 파악**하는 것이 중요합니다.
- 가장 먼저 그리디, 구현, 완전 탐색 등의 아이디어로 문제를 해결할 수 있는지 검토할 수 있습니다.
  - 다른 알고리즘으로 풀이 방법이 떠오르지 않으면 다이나믹 프로그래밍을 고려해 봅시다.
- 일단 재귀 함수로 비효율적인 완전 탐색 프로그램을 작성한 뒤에 (탑다운) 작은 문제에서 구한 답이 큰 문제에서 그대로 사용될 수 있으면, 코드를 개선하는 방법을 사용할 수 있습니다.
- 일반적인 코딩 테스트 수준에서는 기본 유형의 다이나믹 프로그래밍 문제가 출제되는 경우가 많습니다.

다이내믹 프로그래밍 문제풀이 과정

1. 문제에서 요구하는 답을 문장으로 표현
2. 문장에 나와있는 변수 개수만큼 메모를 위한 캐시배열 생성
3. 문제를 부분문제로 나누고, 점화식을 구하여 문제를 함수로 표현
4. Topdown은 재귀함수, Bottomup은 for문을 활용해 답을 도출

# Q&A

# 감사합니다