동적 프로그래밍

알고리즘 스터디#9

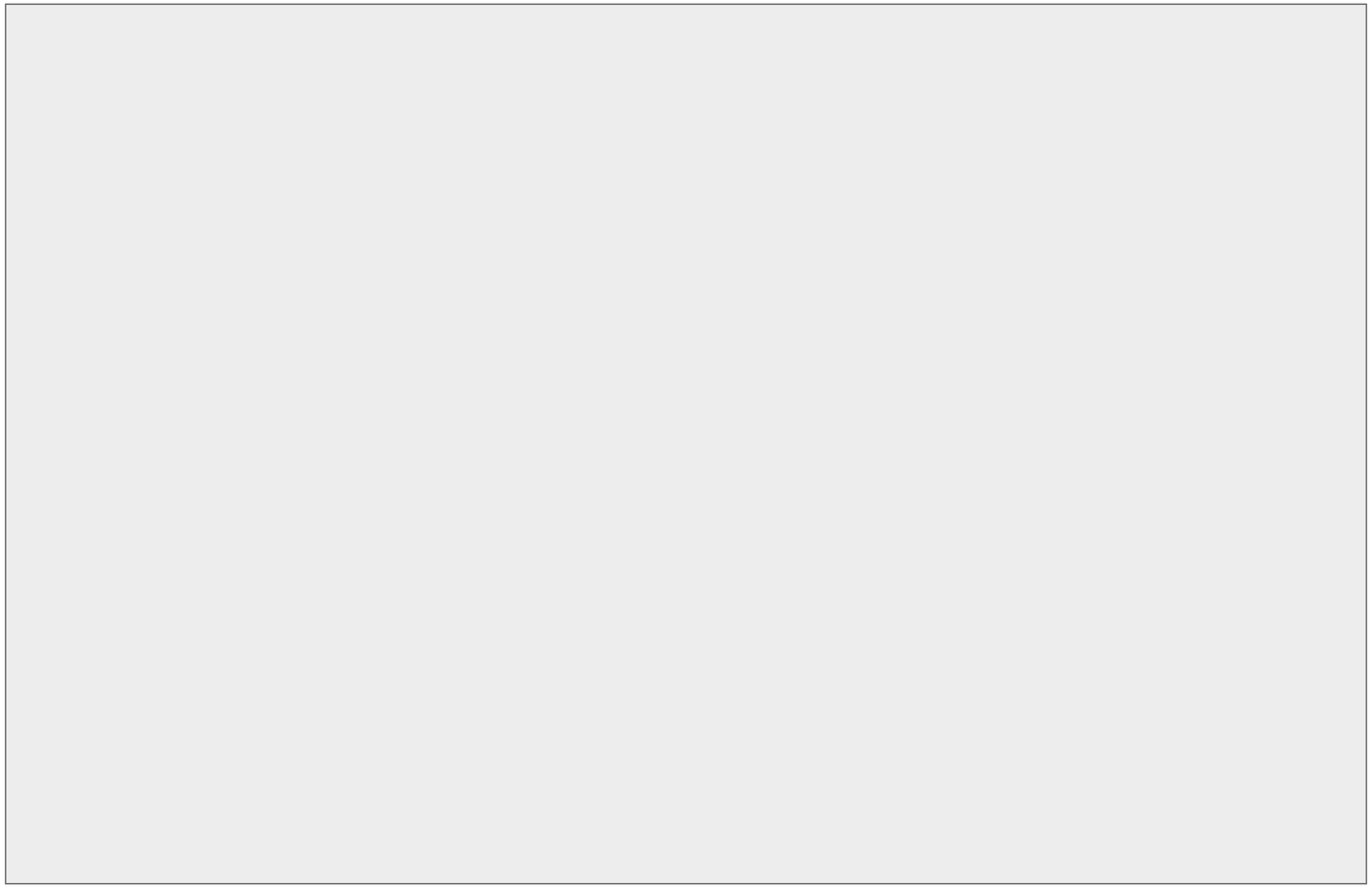
2021.10.12

동적 프로그래밍이란?

- 동적 계획법
- Dynamic Programming (DP)
- 복잡한 문제를 간단한 여러 개의 문제로 나누어 푸는 방법
- 큰 문제를 작은 문제로 나누어 푸는 문제
- 부분 문제 반복, 최적 부분 구조를 가지고 있는 알고리즘을 더 적은 시간 내에 풀 때 사용

피보나치 수열

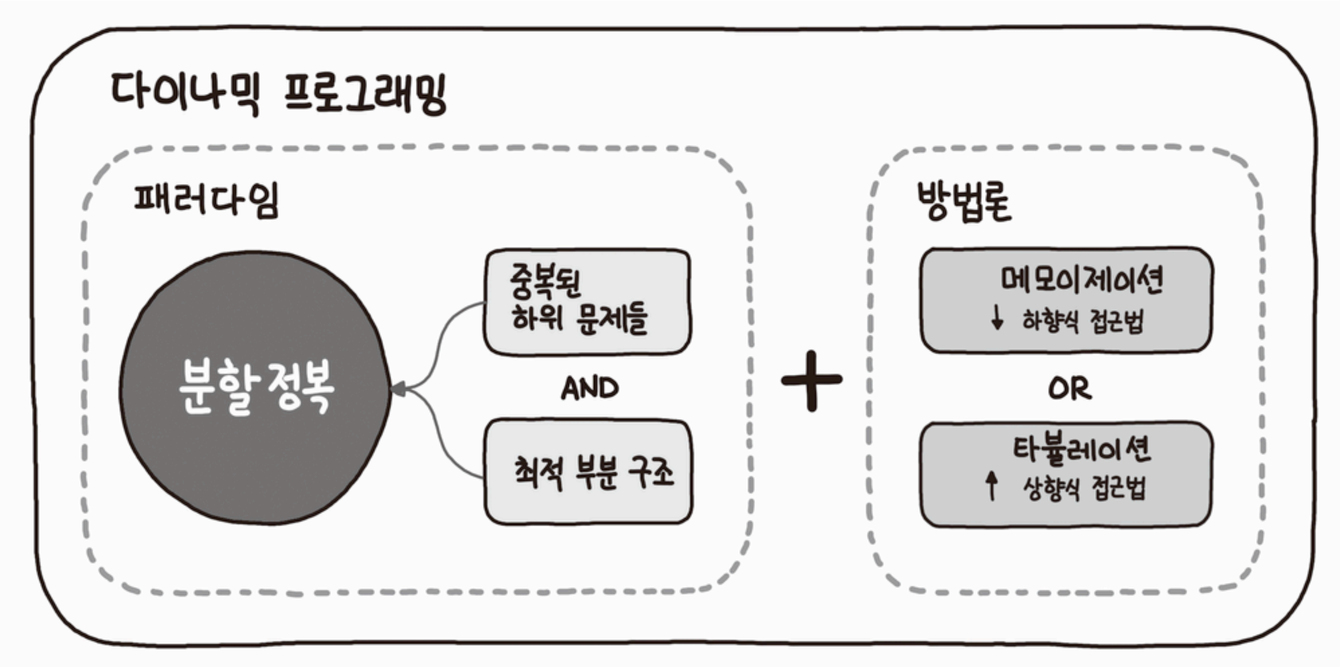
- (0), 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89…



fibo(6)의 실행 과정

분할 정복과의 차이

- 분할 정복: 단지 큰 문제를 해결하기 어려워 작은 문제로 나누어 푸는 방식
  - 재귀적 성격을 띄며 동일한 문제를 추후 다시 푼다는 단점이 있음
- 동적 계획법: 작은 문제들이 반복되기 때문에 나누어 푸는 방식



동적 프로그래밍의 조건 및 방법

[조건]

1. 작은 문제가 반복되어 발생
2. 같은 문제는 구할 때마다 정답이 같음

→ 점화식을 세울 수 있어야 함

[방법]

1. 모든 작은 문제들은 한번만 풀어야 함
2. 정답을 구한 문제는 저장해둠
3. 그보다 큰 문제를 풀어나갈 때 똑같은 작은 문제가 발생하면 이전의 결과값을 호출하여 이용

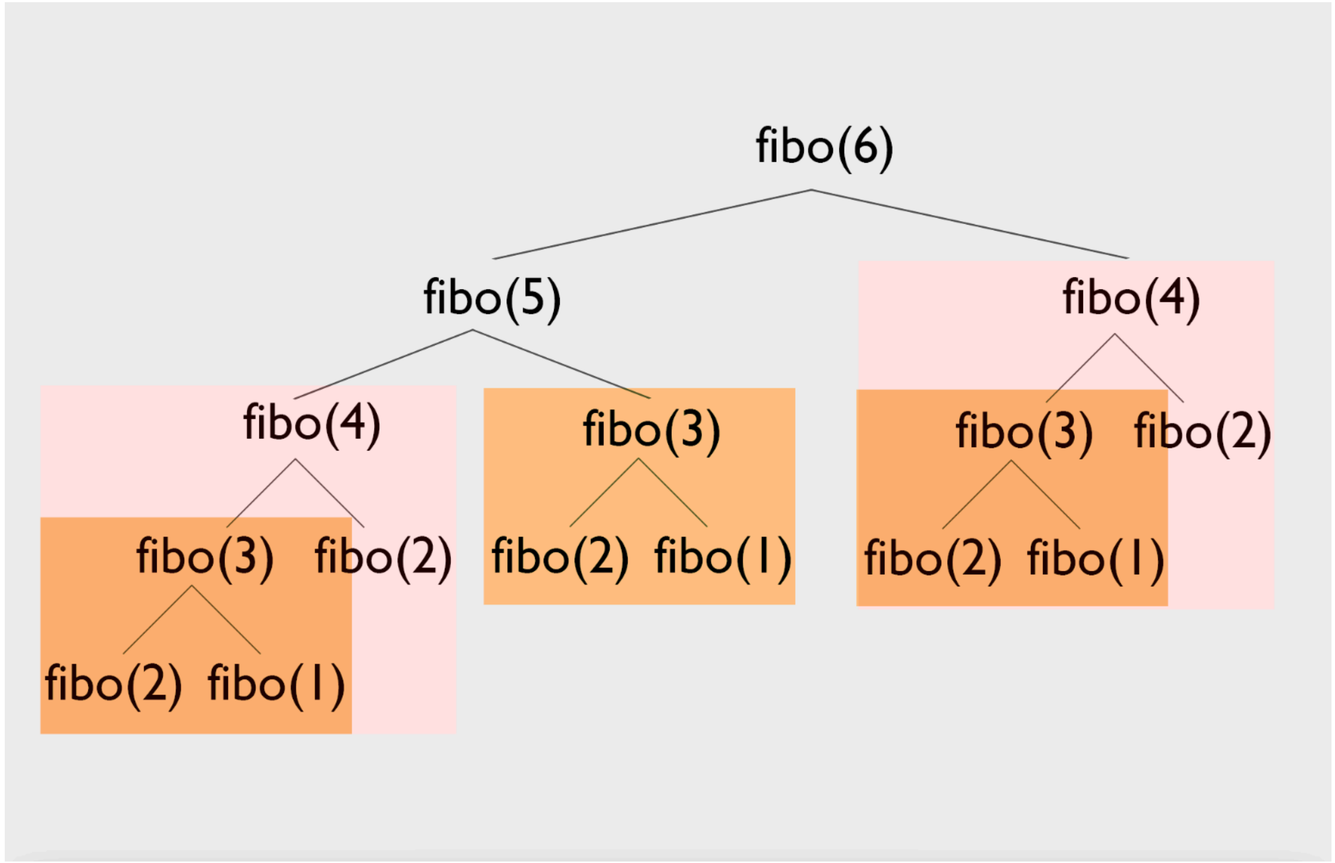
Memoization

- 동일한 문제를 반복해야 할 경우, 한 번 계산된 결과를 저장해 두었다가 활용하는 방식으로 중복 계산을 줄이는 방법을 **메모이제이션(Memoization)**이라 함

피보나치 수열 - 1, 1, 2, 3, 5, 8 …

[조건]

1. 작은 문제가 반복되어 발생
2. 같은 문제는 구할 때마다 정답이 같음



피보나치 수열

- 피보나치 수열의 n번째 수를 구하는 함수

```
int fibo(int n)
  {
    if (n<=2)
      return 1;
    else
      return fibo(n-1) + fibo(n-2);
   }
```

재귀적으로 구현

```
int fiboData[100] = {0,};

int fibo(int n)
{
  if (n<=2)
    return 1;
  if (fiboData[n]==0)
    fiboData[n] = fibo(n-1) + fibo(n-2);
  return fiboData[n];
}
```

동적 계획법으로 구현

구현 방법1

Bottom-up

- 작은 문제부터 구해서 올라가는 방법

```
int fibo(int n)
{
  fibodata[0] = 0;
  fiboData[1] = 1;
  for (int i=2; i<=n; i++)
    fiboData[i] = fiboData[i - 1] + fiboData[i - 2];
  return fiboData[n];
}
```

구현 방법2

Top-down

- 큰 문제를 풀 때 작은 문제가 풀리지 않았다면 그제서야 작은 문제를 해결하는 방법
- 재귀 함수로 구현하는 경우가 대부분 이 경우

```
int fiboData[100] = {0,};

int fibo(int n)
{
  if (n<=2)
    return 1;
  if (fiboData[n]==0)
    fiboData[n] = fibo(n-1) + fibo(n-2);
  return fiboData[n];
}
```

뭐가 더 좋을까?

- Bottom-up : 작성이 쉬우나 소스의 가독성이 저하될 수 있음
- Top-down : 소스의 가독성이 좋으나 작성하기 어려움

- 둘 중 하나만 써야되는 경우도 있으나 거의 없음

→ 편한 걸로 골라 쓰자.

그리디 알고리즘과의 비교

- 그리디 알고리즘: 모든 해를 구하지 않고 순간마다 그 순간에서의 최적의 해를 찾는 방식
  - 닥치는 순간만을 고려해서 해를 구하기 때문에 도출된 값이 항상 최적의 해라고 할 수는 없다.
- 동적 계획법: 모든 방법을 일일이 검토하여 최적의 해를 찾아내는 방식의 알고리즘
  - 동적 계획법은 모든 방법을 검토해 보고 결과적으로 효율적인 값을 택함.
  - 그런 면에서 동적 계획법은 그리디 알고리즘에 비해 시간이 오래 걸리지만, 결과적으로는 항상 최적의 해를 구할 수 있다는 이점을 가지고 있다.

풀어올 문제

- 프로그래머스 - 등굣길
- https://programmers.co.kr/learn/courses/30/lessons/42898

출처

- https://galid1.tistory.com/507#recentEntries
- https://velog.io/@chelsea/1-%EB%8F%99%EC%A0%81-%EA%B3%84%ED%9A%8D%EB%B2%95Dynamic-Programming-DP