💛

Dynamic Programming

1. Dynamic Programming 개념

✔️ 하나의 큰 문제를 여러개의 작은 문제로 나누어서 그 결과를 저장하여 다시 큰 문제를 해결할 때 사용하는 것

→ 일반적인 재귀 함수를 이용하면 비효율적인 계산이었던 문제들을 효율적으로 계산할 수 있음

✔️ **Memoization**

: 재귀 호출 시, 반복적으로 계산되는 것들의 계산 횟수를 줄이기 위해 이전에 계산했던 값을 저장해두었다가 나중에 재사용하는 방법

1-1. Divide and Conquer과의 차이

- 기준: 작은 문제의 중복이 일어나는지 여부
- Divide and Conquer: 그냥 작게 나누어 푸는 것일 뿐, 똑같은 게 반복해서 답으로 가는 게 아님
- DP: 작은 문제들이 반복되는 것

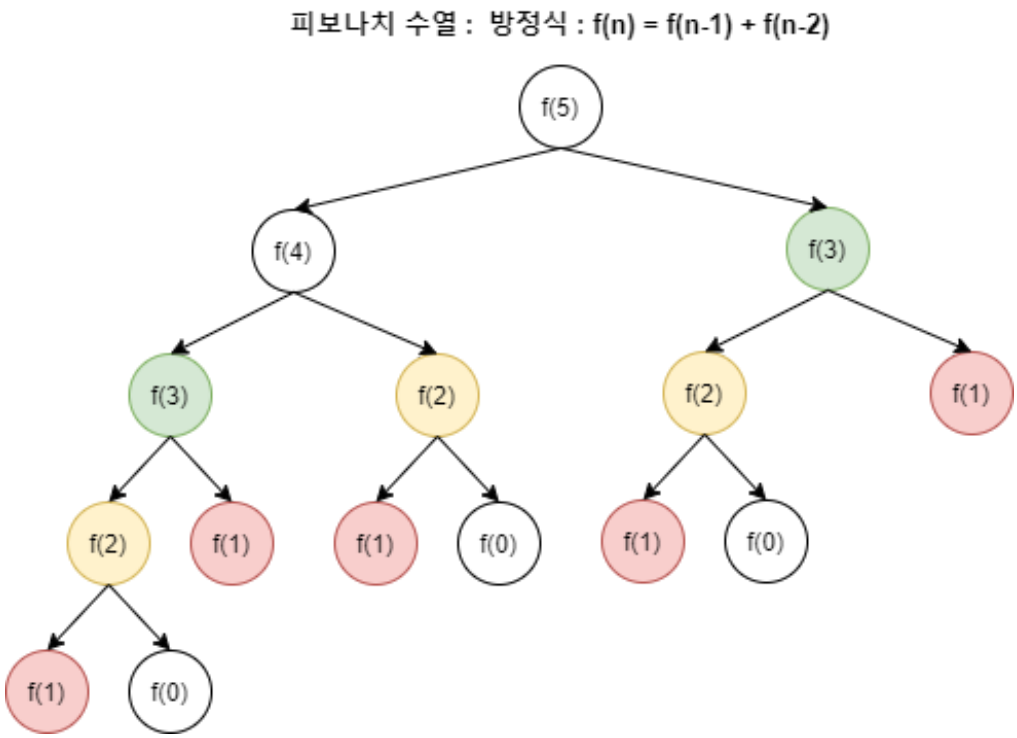
1-2. Dynamic Programming의 조건

1. **Overlapping Subproblems**

: 동일한 같은 문제들이 반복하여 나타나는 경우에 사용

∵ memoization을 사용해야 하는데, 반복적으로 문제가 발생하지 않으면 재사용이 불가능함

ex) Fibonacci

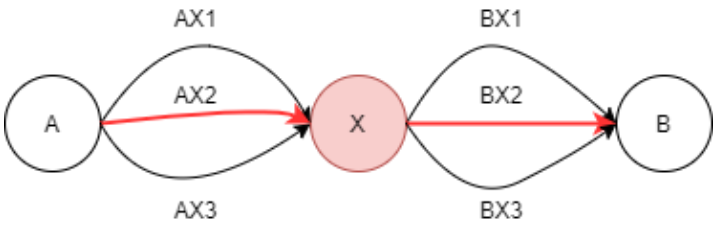


f(0), f(1), f(2) 와 같은 동일한 부분 문제가 중복되어 나타남

```
import sys

def fibo_dp(n):
    memo[0] = 1
    memo[1] = 1

    if n<2:
        return memo[n]

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

    return memo[n]

if __name__ == '__main__':
    n = int(sys.stdin.readline())
    memo = [0 for i in range(n+2)]
    print(fibo_dp(n))
```

2. **Optimal Substructure**

   : 부분 문제의 최적 결과 값을 사용해 전체 문제의 최적 결과를 낼 수 있는 경우



   A에서 X로 가는 최적 경로 = AX2(부분 문제)

   X에서 B로 가는 최적 경로 = BX2(부분 문제)

   AX2와 BX2로 가는 게 A에서 B로 가는 최적의 결과(전체 문제)

2. 구현방법

2-1. 코드 구성

a. **Bottom-up(반복문 사용)**

   작은 문제부터 차근차근 구해나가는 방법

b. **Top-down(재귀함수 사용)**

   대표적인 사례: 재귀함수

   큰 문제를 풀 때 안플리면 작은 문제로 넘어가기

→ 둘 중에 더 나은 것은 없고, 둘 중에 하나로만 풀리는 문제는 있다고 함!

2-2. 코드 짜기 방법론(피보나치 예시)

1. **DP로 풀 수 있는 문제인지 확인**
2. **문제의 변수 파악**
   : n에 따른 결과를 재사용 하는 거니까 n이 변수임
3. **변수 간 관계식 만들기**
   : f(n) = f(n-1) + f(n-2)
4. **Memoization 하기**
   : 변수 값에 따른 결과를 저장할 방식 만들기 like 배열

5. **기저 상태 파악하기**
   : f(0) = 0, f(1) = 1 같이 가장 작은 문제의 상태 파악
6. **구현하기**

3. 사례

LCS(Longest Common Subsequence)문제

- 문제 정의

  Common Subsequence는 순서가 있는 string의 일부분

  ex) X = ABCBDAB, Y=BDCABA → BCA는 CS 중에 하나

  LCS = Common Sequence 중에 제일 긴 것 찾기

- 접근 방식

  1. **Brute Force**

     X, Y의 모든 subsequence를 찾고 그 중 공통된 LCS 찾기

     → string X의 길이가 m이면 X의 subsequence 개수 = 2^m

     → 비효율적!

  2. **DP**

     a. prefix에 번호 붙이기 $X_i = x_1x_2x_3 \cdots x_i$

        ex. X = ABCBDAB, X_4= ABCB

     b. case 나누기

        Z는 LCS를 의미, z_i는 LCS의 prefix

        1. $x_m = y_n$이면, $z_k = x_m = y_n$이고, Z_{k-1}은 X_{m-1}와 Y_{n-1}의 LCS임

           = 각 string의 마지막 문자가 같으면, LCS에 그 문자 붙여주면 되는 것
           = 그러니까 마지막 문자를 제외하고 LCS를 구한다음 마지막 문자를 붙이는 것이나 전체 string의 LCS를 구하는 것이나 같음

           ex. X = ABCBDAB, Y=BDCAB 마지막 글자가 B로 같으니까 이 문자를 제외하고 LCS 구한 다음 B를 붙여주면 됨!

2. $x_m \neq y_n$이면, Z는 X_{m-1}과 Y의 LCS이거나, X와 Y_{n-1}의 LCS임
   = 마지막 문자가 같지 않으면 string 중 하나를 줄이고 다시 비교
   → 둘 중에 큰 subsequence가 LCS가 됨

c. 변수 간 관계식

c[i][j] = X와 Y의 LCS의 길이

$$c[i][j] = \begin{cases} 0 & \text{if } i = 0,\ or\ j = 0, \\ c[i-1][j-1] + 1 & \text{if } i, j > 0,\ and\ x_i = y_i, \\ max(c[i][j-1],\ c[i-1][j]) & \text{if } i, j > 0,\ and\ x_i \neq y_i. \end{cases}$$

d. 적용

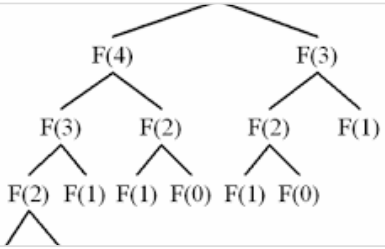
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/ee1fe1fe-6cec-4130-8483-b1ac65c602f6/LCS_problem.mp4

출처

알고리즘 - Dynamic Programming(동적프로그래밍)이란?

Dynamic Programming(동적계획법) 이란 1. Dynamic Programming(동적계획법)이란?큰 문제를 작은문제로 나누어 푸는 문제를 일컫는 말입니다. 동적 계획법이란 말 때문에 어떤 부분에서 동적으로 프로그래밍이 이루

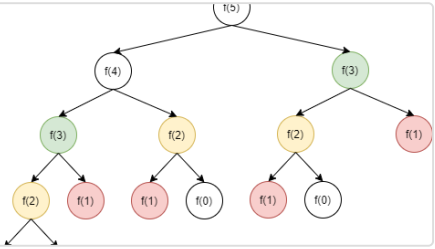
https://galid1.tistory.com/507



알고리즘 - Dynamic Programming(동적 계획법)

1. 개요 DP, 즉 다이나믹 프로그래밍(또는 동적 계획법)은 기본적인 아이디어로 하나의 큰 문제를 여러 개의 작은 문제로 나누어서 그 결과를 저장하여 다시 큰 문제를 해결할 때 사용하는 것으로 특정한 알고리즘이 아닌 하

https://hongjw1938.tistory.com/47



15.Dynamic Programming.pdf https://drive.google.com/file/d/14lGNBocPEJlWBkw1QvJoytjQmvW405FY/view?usp=drivesdk