🔷

DP 예제

다이나믹 프로그래밍

fibonacci DP 구현 방법

우리가 알고 있는 피보나치

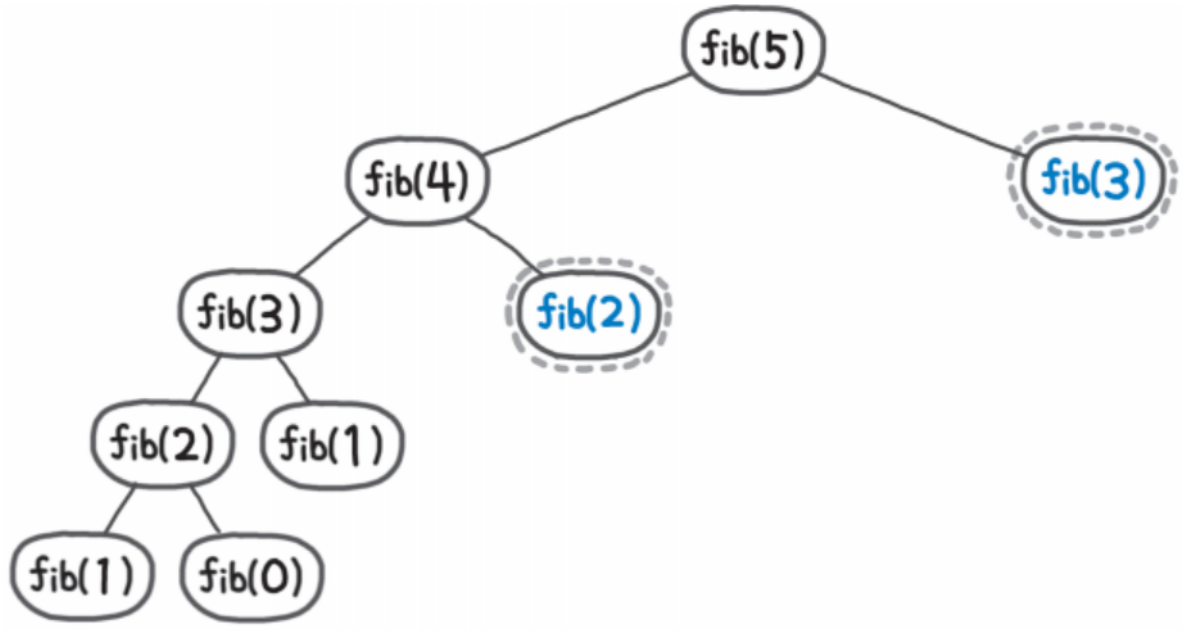
```
Fib(n):
    if n <= 1:
        return n
    else:
        return Fib(n-1) + Fib(n-2)

# n = 5 일때, 15번 연산
```

메모이제이션 (Top-down)

- 재귀를 사용한다는 점은 같지만, 이미 계산한 값은 저장해두고 필요할때 리턴해서 쓰는 방식.

```
class Memoization:
    # 계산 한 값을 저장하는 딕셔너리
    dp = collections.defaultdict(int)
    # dp = {0:0, 1:1, 2:1, 3:2, 4:3, 5:5, 6:8, 7:13, 8:21, 9:34, 10:55}
    def fib(self, N: int) ->:
      if N <= 1:
          return
      #이미 한번 계산한 적있을때 dp에서 불러오기
      if self.dp[N]:
          return self.dp[N]
      # 처음 계산 한 값은 dp에 저장
      self.dp[N] = self.fib(N - 1) + self.fib(N - 2)
      return self.dp[N]
# n = 5 일때, 9번 연산
```



타뷸레이션 (Bottom-up방식)

- 재귀를 사용하지 않고 반복문을 통해 구현.
- 작은 값부터 직접 계산하면서 타뷸레이션

```
class Tabulation:
    #계산한 값을 저장하는 딕셔너리
    dp = collecitons.defaultdict(int)
    #초기 설정
    def fib(self, N: int):
        self.dp[0] = 0
        self.dp[1] = 1
    #미리 계산한 값을 활용
    for i in range(2,N+1):
        self.dp[i] = self.dp[i-1] + self.dp[i-2]
    return self.dp[N]
```

계산 속도 비교

재귀 브루트 포스 : 888밀리초

메모이제이션 : 28밀리초

타뷸레이션 : 24밀리초

비슷한 유형

-백준_계단 오르기_2579

계단을 한번에 한칸 또는 두칸만 오를 수 있다고 할때, 정상까지 올라가는 최적의 방법을 찾는 문제

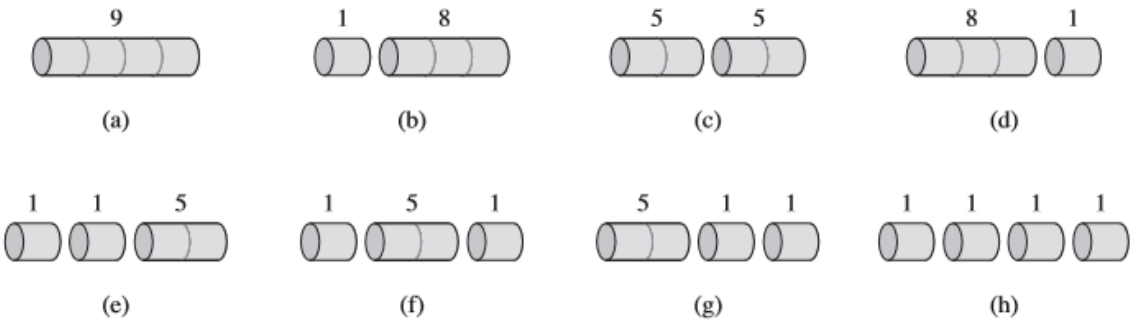
https://www.acmicpc.net/problem/2579

Rod cutting 막대 자르기

- 11052 카드 구매하기

길이가 n인 막대와 길이별 가격 i가 주어졌을때, 길이가 n인 막대를 잘라서 얻을 수 있는 최대 수익을 구하는 문제.

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30



길이가 4인 막대가 주어졌을때, 얻을 수 있는 최대 수익은?

C. 길이가 2인 막대를 2개 만드는 것이 가장 비쌈

풀이 과정:

- 각 길이에 대한 최적해 (최대 수익 길이)를 구한다.
- 임시 리스트에 최적해를 저장한다.
- 계산 과정에서 임시 리스트의 값을 반환한다.

메모이제이션 (Top-down)

수도코드

```
#메인 함수
Memoized-Cut-Rod (p,n)
    r[0,...n]  선언
    for i = 0 to n:
        r[i] = -float('inf') # 매우 작은 값으로 초기화
    return Memoized-Cut-Rod-Aux(p,n,r)

#memoization 계산 함수
Memoized-Cut-Rod-Aux(p,n,r):
    # 최적 해가 이미 r에 존재한다면 반환 중복 계산 방지
    # r : 부분에 대한 최적해를 가지는 리스트
    # p : 길이에 대한 가격을 가지는 리스트
    if r[n] >= 0 :
        return r[n]
    if n == 0
        q = 0
    # 최적해 구하는 과정
    else q = -99999 # 매우 작은 값
        #막대 길이 n에 대하여 가능한 모든 부분 해를 구하고 그중에서 최대 값을 q에 리턴
        for i = 1 to n
            q = max(q,p[i] + Memoized-Cut-Rod-Aux(p,n-i,r))
    r[n] = q
    return q
```

테블레이션(Bottom-up)

```
#Pseudo code
Bottom-Up-Cut-Rod(p,n):
    #최적해 저장할 리스트 선언
    r[0,..n]
    r[0] = 0 #인덱스와 길이를 맞춰주기 위해
    for j in 1 to n:
        q = -9999
        for i = 1 to j:
            q= max(q,p[i] + r[j-i])
        r[j] = q
    return r[n]
#  n = 3 일때
#  j    i      q                                                  return
```

```
#  1    1     max(-9999, p[1] + r[1-1]) = p1,  p1 = 1,             r[1] = 1
#  2    1     max(-9999, p[1] + r[2-1]) = p1 + 1, q = 2
#  2    2     max(2, p[2] + r[2-2]) = p2 + r[0] = 5+0 = q=5        r[2] =5
#  3    1     max(-9999, p[1] + r[3-1]) = p1 + r[2] = 1 + 5 =6
#  3    2     max(6, p[2] + r[3-2]) = p2 + r[1] = 5 + 1 =6
#  3    3     max(6, p[3] + r[3-3]) = p3 + r[0] = 8 + 0 = 8        r[3] =8
```

LCS 최장 공통 부분수열 (Longest Common Subsequence)

주어진 두 시퀀스에 동일한 순서로 존재하는 가장 긴 부분 시퀀스의 길이를 찾는 문제

X: ABCBDAB

Y: BDCABA

length of LCS: 4

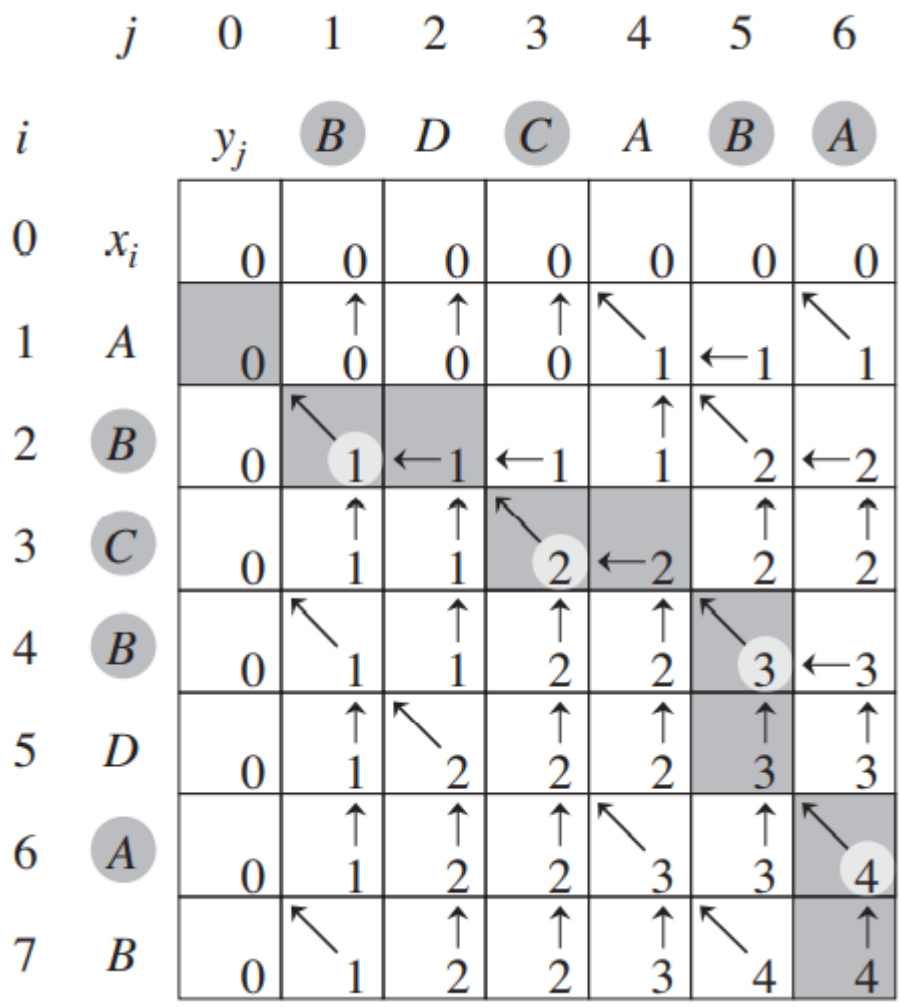
LCS: BDAB, BCAB, BCBA

브루트 포스 방식 접근

X의 모든 부분 집합 과 Y의 모든 부분 집합을 비교해서 LCS를 찾아내야함

시간 복잡도: O(2^n)

다이나믹 프로그래밍



i		j	0	1	2	3	4	5	6
				y_j	B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

테이블 채우는 방법

- 첫 행 과 열은 0으로 채운다.
- 행과 열이 같은 문자를 나타내면, 왼쪽위의 값을 가지고온 뒤 +1 해준다.
- 행과 열이 다른 문자를 나타내면, 위의 값과 왼쪽 값을 비교하여 큰 값을 가지고온다.

▼ Bottom up 방식 코드

```
def LCSLength(X, Y):

    m = len(X)
    n = len(Y)

    # 조회 테이블은 이미 계산된 하위 문제에 대한 값을 저장
    # 즉, `T[i][j]`는 하위 문자열의 LCS 길이를 저장
    # `X[0…i-1]` 및 `Y[0…j-1]`
    T = [[0 for x in range(n + 1)] for y in range(m + 1)]

    # Bottom up 방식으로 조회 테이블 채우기
    for i in range(1, m + 1):
```

```
        for j in range(1, n + 1):
            # `X`와 `Y`의 현재 문자가 일치하는 경우
            #print(f'X[{i-1}]: {X[i-1]} Y[{j-1}]: {Y[j-1]}')
            if X[i - 1] == Y[j - 1]:
                T[i][j] = T[i - 1][j - 1] + 1
            # 그렇지 않으면 `X`와 `Y`의 현재 문자가 일치하지 않는 경우
            else:
                T[i][j] = max(T[i - 1][j], T[i][j - 1])

    # LCS는 조회 테이블의 마지막 항목이 된다.
    for t in range(m+1):
        print(T[t])
    return T[m][n]


if __name__ == '__main__':

    X = 'BDCABA'
    Y = 'ABCBDAB'

    print('The length of the LCS is', LCSLength(X, Y))
```

비슷한 유형

-

0-1 knapsack problem

문제 설명

가방의 용량이 주어졌을때, 가방에 담을 수 있는 물건의 가치를 최대로 하는 방법을 찾는 문제.

단, 물건은 쪼갤 수 없음

예제)

```
weight = [12, 1, 4, 1, 2] #짐의 용량
value = [4, 2, 10, 1, 2] #짐의 가치
W =15  # 가방 용량
```

테뷸레이션

배낭 용량 / 짐 개수	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6
3	0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12
4	0	2	3	3	10	12	13	13	13	13	13	13	13	13	13	13
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

세로축 : 짐의 수

가로축 : 배낭 용량

각각의 셀: 짐의 개수와 배낭의 용량에 따른 최댓값

ex) 짐의 개수가 4개고, 배낭의 용량이 4일때, 가치가 10인 짐 하나만 담는게 가장 이득.

짐의 수가 5개고, 배냥의 용량이 10일때, 최대 가치는 15

테뷸레이션 코드

```
def zero_one_knapsack(cargo):
    capacity =15
    pack = []

    for i in range(len(cargo) + 1):
        pack.append([])
        for c in range(capacity +1):
            if i == 0 or c == 0:
                pack[i].append(0)
            elif cargo[i-1][1] <= c:
                pack[i].append(
                    max(
                        cargo[i-1][0] + pack[i-1][c - cargo[i-1][1]],
                        pack[i-1][c]
                    )
                )
            else:
                pack[i].append(pack[i-1][c])
    return pack[-1][-1]
```

- 백준 12865

https://www.acmicpc.net/problem/12865

공통문제

- 백준 2294 동전2

https://www.acmicpc.net/problem/2294

- 백준 11052 카드구매하기

https://www.acmicpc.net/problem/11052

개인문제

- 백준 12865 배낭 문제

https://www.acmicpc.net/problem/12865

- 백준 15483 최소 편집 문제 (Levenshtein distance)

https://www.acmicpc.net/problem/15483

- 백준_계단 오르기_2579

https://www.acmicpc.net/problem/2579

- 알고리즘 수업 - 피보나치 수1

https://www.acmicpc.net/problem/24416

그외 문제 추천

https://won-percent.tistory.com/3