Dynamic Programming

문제를 하위 문제들로 나누어 해결하는 알고리즘→반복되는 하위 문제들의 효율적 해결

DP 필수 조건

▼ **Overlapping Subproblems** (계산의 효율성)

- 동일한 하위 문제가 여러 번 반복되는 특성을 활용하여 중복 계산을 줄일 수 있음

▼ **Optimal Substructure (구조적 특성)**

- 문제의 최적 해답이 그 하위 문제들의 최적 해답으로부터 유도 가능

DP 구현 방법

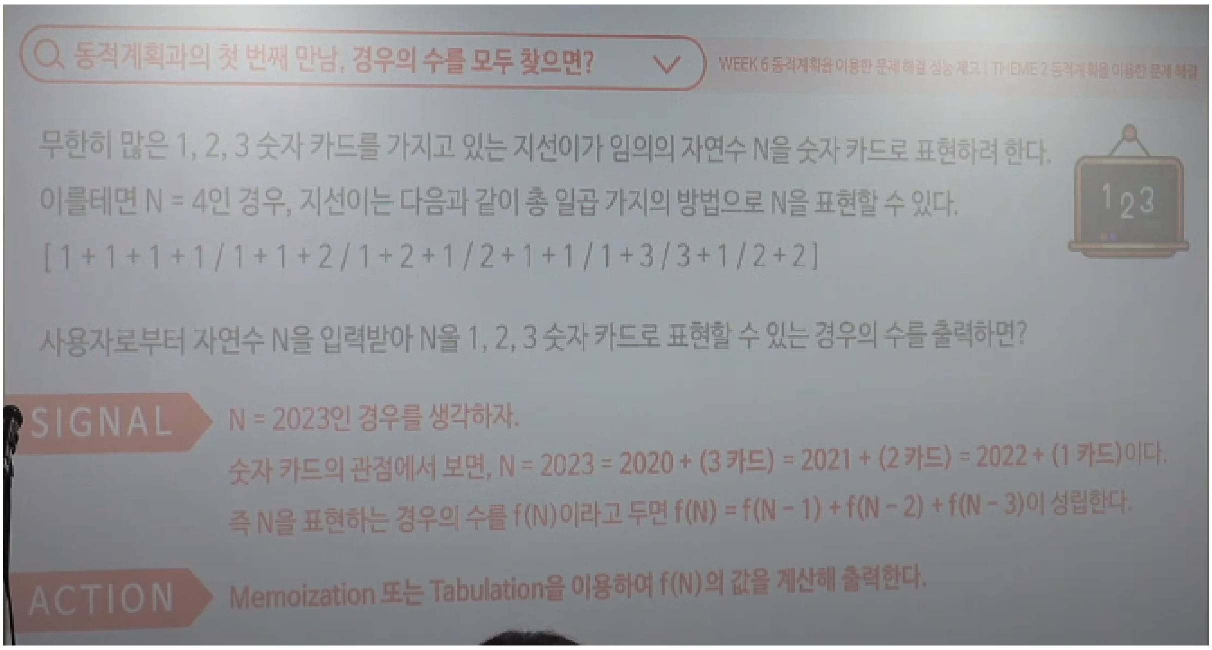
Top-Down (Memoization)

- 각 하위 문제의 해를 메모리에 저장해두고 필요할 때 꺼내 쓰는 방법
- 최상위 문제에서 시작, 중간 과정 값들 기록 하여 효율성 제고

Bottom-Up (Tabulation)

- 최하위 문제에서 시작, 그 해답을 이용해 더 큰 문제를 해결하는 방법

점화식



1. Fibonacci sequence

Overlapping Subproblems : O(2^n)

```python
def fib_recursive(n):
    if n <= 2:
        return 1
    return fib_recursive(n-1) + fib_recursive(n-2)
```

Top-Down (Memoization) : O(n)

```python
def fibonacci_memoization(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 2:
        return 1
    memo[n] = fibonacci_memoization(n-1, memo) + fibonacci_memoization(n-2, memo)
    return memo[n]
```

Bottom-Up (Tabulation)

```python
def fibonacci_tabulation(n):
    if n <= 2:
        return 1
    fib_table = [0] * (n + 1)
    fib_table[1] = fib_table[2] = 1

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

    return fib_table[n]
```

2. Knapsack problem

Optimal Structure : O(2^n)

```python
def knapsack_recursive(W, wt, val, n):
    if n == 0 or W == 0:
        return 0

    if wt[n-1] > W:
        return knapsack_recursive(W, wt, val, n-1)

    else:
        return max(
            val[n-1] + knapsack_recursive(W - wt[n-1], wt, val, n-1),
            knapsack_recursive(W, wt, val, n-1)
        )
```

Tabulation : O(n*w)

```python
def knapsack_dp(W, wt, val, n):
    dp = [[0 for x in range(W + 1)] for x in range(n + 1)]

    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                dp[i][w] = 0
            elif wt[i-1] <= w:
                dp[i][w] = max(val[i-1] + dp[i-1][w-wt[i-1]], dp[i-1][w])
            else:
                dp[i][w] = dp[i-1][w]

    return dp[n][W]
```

3. Edit distance problem

O(3^n)

```python
def edit_distance_recursive(s1, s2, m, n):
    if m == 0:
        return n
    if n == 0:
        return m

    if s1[m-1] == s2[n-1]:
        return edit_distance_recursive(s1, s2, m-1, n-1)

    return 1 + min(
        edit_distance_recursive(s1, s2, m, n-1),
        edit_distance_recursive(s1, s2, m-1, n),
        edit_distance_recursive(s1, s2, m-1, n-1)
    )
```

Tabulation : O(m*n)

```python
def edit_distance_dp(s1, s2):
    m = len(s1)
    n = len(s2)

    dp = [[0 for x in range(n + 1)] for x in range(m + 1)]

    for i in range(m + 1):
        for j in range(n + 1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif s1[i-1] == s2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i][j-1],
                                   dp[i-1][j],
                                   dp[i-1][j-1])

    return dp[m][n]
```