**1. 배낭 문제 (Knapsack Problem)**

**문제 :** 무게제한(W)이 있는 배낭에 담을 수 있는 물건이 n개가 주어졌을 때, 각 물건의 무게(w\_i)

와 가치(v\_i)가 있다. 이때, 배낭에 담을 수 있는 물건의 조합 중 최대 가치를 찾아라

**Brute Force**

모든 가능한 조합을 확인하여 최적의 해를 찾는다.

가능한 총 조합 수는 **2n**으로, 시간 복잡도는 **exponential**이다.

**Dynamic Programming**

DP 배열 dp[i][w]을 사용하여, 물건(i)까지 고려했을 때 배낭무게(w)에서의 최대 가치를 저장한다.

**점화식 :** dp[i][w] = max(dp[i-1][w], dp[i-1][w-{w\_i}] + {v\_i})

**시간 복잡도 : O(nW)**, W는 배낭의 최대 무게이다.

**효율성 비교**

Brute Force는 **O(2n)**, DP는 **O(nW)**으로, 특히 n과 W가 클수록 DP가 훨씬 빠르다.

예를 들어, n = 20, W = 50일 때, Brute Force는 220개의 조합을 확인하지만

DP는 1000개의 경우만 계산하면 된다.

**2. 최장 공통 부분 수열 (Longest Common Subsequence, LCS)**

**문제 :** 두 문자열 A와 B가 주어졌을 때, 공통으로 등장하는 가장 긴 부분 수열의 길이를 찾아라

**Brute Force**

모든 부분 수열을 생성하여 비교한다. 문자열의 길이가 각각 m , n일 때,

시간 복잡도는 **O(2m+n)**이다.

**Dynamic Programming**

DP 배열 dp[i][j]를 사용하여, A의 처음 글자(i)와 B의 처음 글자(j)에서의 LCS 길이를 저장한다.

**점화식:** ( if A[i] = B[j] ) dp[i][j] = dp[i-1][j-1] + 1 &

( if A[i] ≠ B[j] ) dp[i][j] = max(dp[i-1][j], dp[i][j-1]) &

( if i=0 or j=0 ) dp[i][j] = 0

**시간 복잡도: O(mn)**이다.

**효율성 비교**

Brute Force는 지수 시간 복잡도 **O(2n+m)**이며, DP는 **O(mn)**으로 효율적이다.

예를 들어 m = 100, n = 100일 때, DP는 10,000번의 연산으로 해결할 수 있지만

Brute Force는 2200번의 연산을 해야 해서 사실상 불가능하다**.**