1주차 : Knapsack

냅색 알고리즘

> Kanpsack : 배낭
> 배낭에 최대치로 물건을 담았을 때, 최대의 가치 값을 구하는 문제.

- **Fractional Knapsack**
  물건을 자를 수 있는 경우, Greedy
- **0-1 Knapsack**
  물건을 자를 수 없는 경우, DP

알고리즘을 알지 못했을 때, 접근하기 쉬운 두가지 방법.

1. 모든 경우의 수 (Brute-Force)

   n개의 물건이 있다고 치면, n개 물건으로 만들 수 있는 가능한 부분집합 (power set) 의 수는 2^n개이다. O(2^n)

2. 가격이 높은 물건, 혹은 (가격/무게) 의 값이 제일 높은 물건부터 먼저 골라서 넣는다 (Greedy)

```
배낭 최대 용량 = 30kg
보석1: 5kg /$5  -- kg당 $1
보석2: 10kg/$6  -- kg당 $0.6
보석3: 20kg/$14 -- kg당 $0.7

최적o : 보석1, 보석3 = 25kg / $19
최적x : 보석2, 보석3 = 30kg / $20

※ Fractional Knapsack 문제는 Greedy로 풀 수 있다. 위 예시에서 보석2를 반으로 잘라서 남은 5kg짜리 공간에 넣으면 그게 최적의 답이 된다
```

DP로 접근하기

- 물건을 넣는다 / 물건을 넣지 않는다.

만약 현재 내가 넣으려고 하는 물건의 무게가 전체 제한무게를 초과한다면, 선택지는 물건을 넣지 않는 것 밖에 없다.

```
dp[i][W] = dp[i-1][W], if wi > W
# i: 현재 넣은 물건의 번호, W: 최대 무게, wi: i번째 물건의 무게
```

만약 물건을 넣은 수 있는 무게가 남아있다면,

- 새로운 물건을 넣지 않는다.
- 새로운 물건을 넣는다.

```
dp[i][W] = max(dp[i-1][W], dp[i][W-wi] + val(wi))
```

점화식

```
# i: 현재 넣은 물건의 번호, W: 최대 무게, wi: i번째 물건의 무게
dp[i][W] = dp[i-1][W], if wi > W
dp[i][W] = max(dp[i-1][W], dp[i][W-wi] + val(wi))
```

$$P[i,w] = \begin{cases} P[i-1,w] & \text{if } w_i > w \\ \max\{v_i + P[i-1, w-w_k], P[i-1,w]\} & \text{else} \end{cases}$$

문제 풀어보기

평범한 배낭 (Gold5) - https://www.acmicpc.net/problem/12865

```
n, k = map(int, input().split())
stuff = [tuple(map(int, input().split())) for _ in range(n)]

# dp : n x k 이차원 배열
```

```
dp = [[0] * (k+1) for _ in range(n)]

for i in range(n):
    wi, vi = stuff[i]
    for W in range(1, k+1):
        if wi > W:
            dp[i][W] = dp[i-1][W]
        else:
            dp[i][W] = max(dp[i-1][W], dp[i-1][W-wi] + vi)

print(dp[n-1][k])


# dp : k 1차원 배열
dp = [0] * (k+1)

for i in range(n):
    wi, vi = stuff[i]
    for W in range(k, -1, -1):
        if wi > W:
            continue
        else:
            dp[W] = max(dp[W], dp[W - wi] + vi)
print(dp[k])
```

시간복잡도

`O(NK) n: 물건의 개수, k: 최대 무게`

추가 문제

양팔저울 (Gold 3) - https://www.acmicpc.net/problem/2629

▼ 코드

```
n = int(input())
weights = list(map(int, input().split())) + [0]
m = int(input())
targets = list(map(int, input().split()))

# dp[i][w] : i번 까지의 추를 사용했을 때 w 무게를 만들 수 있는지에 대한 여부
MAX = sum(weights) + max(targets)
dp = [[False]*(MAX+1) for _ in range(n+1)]


def solve(i, w):
    if i > n or dp[i][w]:
        return
    dp[i][w] = True
    solve(i + 1, w + weights[i])  # 저울 오른쪽
    solve(i + 1, abs(w - weights[i]))  # 저울 왼쪽
```

```
    solve(i + 1, w)  # 저울에 안 놓음


solve(0, 0)

for target in targets:
    if target > (MAX+1):
        print('N', end=' ')
    elif dp[-1][target]:
        print('Y', end=' ')
    else:
        print('N', end=' ')
```

참고

- 배낭문제 카테고리 : https://www.acmicpc.net/problemset?sort=ac_desc&algo=148
- https://velog.io/@uoayop/냅색-알고리즘
- https://gsmesie692.tistory.com/113
- https://jeonyeohun.tistory.com/86
- https://chanhuiseok.github.io/posts/improve-6/