6. 조합론

▼ 목차

1. knapstack

배낭에 담을 수 있는 무게의 최댓값이 정해져 있고, 일정한 가치의 무게가 정해진 짐들을 배낭에 담을 때, 가치의 합이 최대가 되는 조합을 찾는 알고리즘

예시)

	A 물건	B 물건	C 물건	D 물건	E 물건
무게	6	4	2	3	5
가치	13	9	5	3	12
무게당 가치 (Fractional Knapsack)	2.16	2.25	2.5	1	2.4

⇒ 가방에 넣은 수 있는 총 무게가 7kg이라고 할 때, 최대 가치를 가지는 물건의 조합을 구하라

(0). 두 가지 문제 유형

(1). **0-1 Knapsack Problem**
물건을 쪼갤 수 없는 배낭 문제로, **동적계획법**(Dynamic Programming)을 활용해 해결 가능

(2). **Fractional Knapsack Problem**
물건을 쪼갤 수 있는 배낭 문제로, 가치가 높은 순으로 정렬한 뒤 배낭에 담고, 텍스트남은 부분은 물건을 쪼개어 넣어 조합을 찾을 수 있음. **그리디 알고리즘**으로 해결 가능

(1). 0-1 Knapsack Problem (해결 방법 _ DP 동적계획법)

물건을 쪼갤 수 없는 경우

- **모든 경우의 수 넣어보기 (Brute-Force)**
n개의 물건이 있다고 치면, n개 물건으로 만들 수 있는 가능한 부분집합 (power set) 의 수는 2^n개이다. 최악의 경우에 2^n번까지 봐야 한다. **완전 탐색으로 풀리긴 하지만, 시간 복잡도가 높다.**

- **가치가 높은 물건부터 먼저 골라서 넣기 (Greedy)**
가치가 제일 높은 물건을 먼저 넣는 것은 최적의 해를 보장해주지 못한다. 가끔 가치가 낮은 물건을 여러개 고르는 것이 더 큰 가치를 가지기도 하기 때문이다. **그리디로는 풀 수 없는 문제이다.**

- **동적계획법 (DP)**

동적 계획법은 일반적으로 문제를 풀기 위해, 문제를 여러 개의 작은 문제로 쪼개서, 그 값들을 결합하여 최종적인 결과를 얻는 것

---

ex) 백준 _ 12865

문제

이 문제는 아주 평범한 배낭에 관한 문제이다.

한 달 후면 국가의 부름을 받게 되는 준서는 여행을 가려고 한다. 세상과의 단절을 슬퍼하며 최대한 즐기기 위한 여행이기 때문에, 가지고 다닐 배낭 또한 최대한 가치 있게 싸려고 한다.

준서가 여행에 필요하다고 생각하는 N개의 물건이 있다. 각 물건은 무게 W와 가치 V를 가지는데, 해당 물건을 배낭에 넣어서 가면 준서가 V만큼 즐길 수 있다. 아직 행군을 해본 적이 없는 준서는 최대 K만큼의 무게만을 넣을 수 있는 배낭만 들고 다닐 수 있다. 준서가 최대한 즐거운 여행을 하기 위해 배낭에 넣을 수 있는 물건들의 가치의 최댓값을 알려주자.

<풀이>

(1) 2차원 배열 만들기

무게 / 가치	0	1	2	3	4	5	6	7
0 / 0	0	0	0	0	0	0	0	0
6 / 13	0	0	0	0	0	0	0	0
4 / 8	0	0	0	0	0	0	0	0
3 / 6	0	0	0	0	0	0	0	0
5 / 12	0	0	0	0	0	0	0	0

(2)

첫 번째 물품 (6 / 13)

무게 / 가치	0	1	2	3	4	5	6	7
0 / 0	0	0	0	0	0	0	0	0
6 / 13	0	0	0	0	0	0	**13**	**13**
4 / 8	0	0	0	0	0	0	0	0
3 / 6	0	0	0	0	0	0	0	0
5 / 12	0	0	0	0	0	0	0	0

(3)

두 번째 물품 (4 / 8)

무게 / 가치	0	1	2	3	4	5	6	7
0 / 0	0	0	0	0	0	0	0	0
6 / 13	0	0	0	0	0	0	13	13
4 / 8	0	0	0	0	**8**	**8**	**13**	**13**
3 / 6	0	0	0	0	0	0	0	0
5 / 12	0	0	0	0	0	0	0	0

(4)

세 번째 물품(3 / 6)

무게 / 가치	0	1	2	3	4	5	6	7
0 / 0	0	0	0	0	0	0	0	0
6 / 13	0	0	0	0	0	0	13	13
4 / 8	0	0	0	0	8	8	13	13
3 / 6	0	0	0	**6**	**8**	**8**	**13**	**14**
5 / 12	0	0	0	0	0	0	0	0

(5)

네 번째 물품(5 / 12)

무게 / 가치	0	1	2	3	4	5	6	7
0 / 0	0	0	0	0	0	0	0	0
6 / 13	0	0	0	0	0	0	13	13
4 / 8	0	0	0	0	8	8	13	13
3 / 6	0	0	0	6	8	8	13	14
5 / 12	0	0	0	**6**	**8**	**12**	**13**	**14**

⇒ 결과적으로 표의 마지막에 담긴 **14가 물건들의 가치합의 최대값**이 된다.

<구현 코드>

```
n, k = map(int, input().split())                # n 물품위 수, k 배낭에 넣은 수 있는 무게
lst=[[0, 0]]
for _ in range(n):
    lst.append(list(map(int, input().split()))) # 물건들의 무게와 가치를 묶어서 리스트에 추가
dp = [[0]*(k+1) for _ in range(n+1)]            # dp을 사용하기 위한 2차원 배열 생성

for i in range(1, n+1):                         # n 물품의 수
    for j in range(1, k+1):                     # k 배낭에 넣은 수 있는 무게
        weight = lst[i][0]                      # 리스트에 저장한 무게와 가치 각각 가져오기
        value = lst[i][1]
        if j < weight:                          # 가방에 넣을 수 없으면
            dp[i][j] = dp[i - 1][j]             # 배열에서 위에 값 그대로 가져오기
        else:                                   # 가방에 넣을 수 있으면
            dp[i][j] = max(dp[i - 1][j], dp[i][j - weight] + value)
                                                # 배열의 위의 값과 현재 비교중인 값에 무게를 더한 값중 큰 것을 선택
print(dp[n][k])                                 # 끝까지 돌았을 때, 마지막 값이 최적의 답이 된다
```

(2). Fractional Knapsack Problem (해결 방법 _ Greedy)

물건을 쪼갤 수 있는 경우
무게당 이익이 높은 물건을 먼저 넣으면 되는 문제

(1) 무게 대비 가치가 가장 높은 순으로 물건들을 정렬
(2) 정렬된 물건들을 순서대로 배낭에 넣기
(3) 전부 넣을 수 없는 경우에는 남은 비율 만큼 value 값을 추가

물건(i)	물건1	물건2	물건3	물건4	물건5
무게(w)	10	15	20	25	30
가치(v)	10	12	10	8	5

```python
knapsack = [(10,10), (15,12), (20,10), (25,8), (30,5)]  # (무게, 가치) 리스트
k = 30   # 무게 제한

def get_max_value(k, knapsack):
    # (가치/무게)를 기준으로 오름차순 정렬
    knapsack = sorted(knapsack, key=lambda x: x[1]/x[0], reverse=True)
    total_value = 0
    details = list()

    for element in knapsack:
        if k - element[0] >= 0:
            k -= element[0]
            total_value += element[1]
            details.append([element, 1])
        else:
            fraction = k/element[0]
            total_value += element[1] * fraction
            details.append([element, fraction])
            break

    return total_value, details
```

2. coin change

- DP문제

ex) 백준_2294

n가지 종류의 동전이 있다. 이 동전들을 적당히 사용해서, 그 가치의 합이 k원이 되도록 하고 싶다. 그러면서 동전의 개수가 최소가 되도록 하려고 한다. 각각의 동전은 몇 개라도 사용할 수 있다.

사용한 동전의 구성이 같은데, 순서만 다른 것은 같은 경우이다.

	1원	2원	3원	4원	5원	6원	7원	8원
동전 1	1개	2개	3개	4개	5개	6개	7개	8개
동전 4				1개	2개	3개	4개	2개
동전 6						1개	2개	2개

코드

```
n, k = map(int, input().split())    # n 동전의 개수, k 원하는 동전들의 가치 합
coin = []
for i in range(n):
```

```
    coin.append(int(input()))       # 주어진 동전들을 리스트에 담기

dp = [10001] * (k+1)                # 동전의 가치는 10,000보다 작거나 같으므로 배열을 만들어주기
dp[0] = 0                           # 배열의 0번째 값은 0


for c in coin:
    for i in range(c,k+1):
        if dp[i]>0:
            dp[i] = min(dp[i], dp[i-c]+1)

if dp[k]==10001:        # 동전을 교환할 수 없는 경우
    print(-1)
else:
    print(dp[k])
```