BOJ

#12865. 평범한 배낭

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

24.11.25

한 가게에 도둑이 들었다.
도둑이 훔치고 싶은 물건들은 다 각각의 값어치(가치)와 무게가 있다.
무게 때문에 도둑은 고를 수 있는 물건이 한정되어 있다.
그러므로 가방에 담을 수 있는 내에서 가장 비싼 (가치가 높은) 물건을 훔치고자한다.

3가지 방법이 있다.

1. **모든 경우의 수**
    - 물건 개수 n개라면, 넣었는가, 넣지 않았는가 2가지의 경우가 있다.
    - 2^N가지의 경우의 수를 구한다.
2. **넣을 수 있는 가장 비싼 물건부터 넣기**
    - Greedy Algorithm
    - 최적의 해는 나오지 않는다. (뒤에서 예로 설명하겠다.)
3. **또 다른 방법?**

Problem

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

시간: 1시간 15분

12865번 | 제출 | 맞힌 사람 | 숏코딩 | 재채점 결과 | 채점 현황 | 내 제출 | 난이도 기여 | 강의 | 질문 게시판

평범한 배낭 성공

☆

골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	512 MB	151595	57894	36408	36.272%

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.

입력

첫 줄에 물품의 수 N($1 \le N \le 100$)과 준서가 버틸 수 있는 무게 K($1 \le K \le 100,000$)가 주어진다. 두 번째 줄부터 N개의 줄에 거쳐 각 물건의 무게 W($1 \le W \le 100,000$)와 해당 물건의 가치 V($0 \le V \le 1,000$)가 주어진다.

입력으로 주어지는 모든 수는 정수이다.

출력

한 줄에 배낭에 넣을 수 있는 물건들의 가치합의 최댓값을 출력한다.

예제 입력 1 복사

```
4 7
6 13
4 8
3 6
5 12
```

예제 출력 1 복사

```
14
```

Problem

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

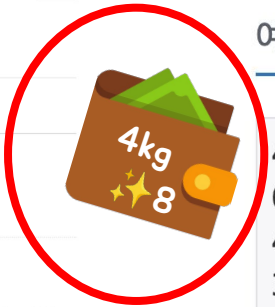
시간: 1시간 15분

12865번 | 제출 | 맞힌 사람 | 숏코딩 | 재채점 결과 | 채점 현황 | 내 제출 | 난이도 기여 | 강의 | 질문 게시판

평범한 배낭 성공

골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	512 MB	151595	5789		36.272%



문제

이 문제는 아주 평범한 배낭에 관한 문제이다.

한 달 후면 국가의 부름을 받게 되는 준서는 여행을 가려고 한다. 세상과의 단절을 슬퍼하며 [illegible] 가치 있게 싸려고 한다.

준서가 여행에 필요하다고 생[illegible]다. 각 물건은 무게 W와 가치 V를 가지[illegible] 아직 행군을 해본 적이 없는 준서는 최대 K만큼의 무[illegible]만 들고 다닐 수 있다. 준서가 최대한 즐[illegible]을 알려주자.

입력

첫 줄에 물품의 수 N(1 ≤ N ≤ [illegible])[illegible] 수 있는 무게 K(1 ≤ K ≤ 100,000)가 주어진다. 두 번[illegible]게 W(1 ≤ W ≤ 100,000)와 하[illegible]의 가치 V(0 ≤ V ≤ 1,000)가 주어진다.

입력으로 주어지는 모든 수는 정수이다.

출력

한 줄에 배낭에 넣을 수 있는 물건들의 가치합의 최댓값을 출력한다.

예제 입력 1 복사

```
4 7
6 13
4 8
3 6
5 12
```

예제 출력 1 복사

```
14
```

Hint

Step 1.

> 주어진 값을 표로 정리해보자.

- **한 물건에 대해 W(무게)와 V(가치) 정보가 있다. 이를 저장할 리스트가 필요하다**
- 가방에 최대 가치를 저장할 수 있도록 할 수 있는 변수 또는 리스트가 필요하다.

Hint

Step 1.

> 주어진 값을 표로 정리해보자.

- 한 물건에 대해 W(무게)와 V(가치) 정보가 있다. 이를 저장할 리스트가 필요하다
- 가방에 최대 가치를 저장할 수 있도록 할 수 있는 변수 또는 리스트가 필요하다.

thing	0	1
i = 1	6	13
i = 2	4	8
i = 3	3	6
i = 4	5	12

Knapsack	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								

K열

N행

DP가 N+1행, K+1열인 이유?
N=1인 경우, DP[i-1][j]값이 존재해야 한다.
즉, 아무것도 담지 않은 **빈 배낭**의 상태를 초기화 해놓기 위해 0행, 0열이 필요하다.

Hint

Step 2.

> 물건을 가방에 담을 수 있는 조건을 생각해보자

- **K의 크기보다 작거나 같아야 가방에 담을 수 있다.**
  - 크기가 크다면 물건을 넣지 못하고 크기가 같거나 작다면 물건을 넣을 수 있다.
- 이를 `i = 1 ~ i = K`인 경우를 생각해보자. **가방에 담을 수 있는 무게가 `i`**가 되는 것이다.
  - 이때 **`i`가 현재 물건의 무게보다 작을 때와 클 때 물건을 담을 수 있는지 없는지** 생각해보자.

Hint

Step 2.

> 물건을 가방에 담을 수 있는 조건을 생각해보자

- **K의 크기보다 작거나 같아야 가방에 담을 수 있다.**
  - 크기가 크다면 물건을 넣지 못하고 크기가 같거나 작다면 물건을 넣을 수 있다.
- 이를 `i = 1 ~ i = K`인 경우를 생각해보자. **가방에 담을 수 있는 무게가 `i`**가 되는 것이다.
  - 이때 **`i`가 현재 물건의 무게보다 작을 때와 클 때 물건을 담을 수 있는지 없는지** 생각해보자.
    i. **i가 현재 무게보다 작을 때: 물건을 담을 수 없다.**
    ii. **i가 현재 무게보다 클 때: 물건을 담을 수 있다.**

💡 Hint

Step 3. 문제 유형이 무엇일까?

> ***Dynamic Programming***

- **하나의 큰 문제를 여러 개의 작은 문제로 나누어서 그 결과를 저장하여 다시 큰 문제를 해결할 때 사용**
- 즉, 큰 문제를 작은 문제로 쪼개서 그 답을 저장해두고 재활용
- '기억하며 풀기'라고 부르는 사람도 있음
- 분할정복과 차이점 : 중복이 일어나느냐 아니냐
- 사용법
    - dp로 풀 수 있는지 확인 -> 변수 파악 -> 점화식

      -> memoization -> 기저 상태 파악 -> 구현

Hint

Step 4. DP 유형이다!

> 어떤 점화식이 들어가야 할까? 무슨 값을 저장해야 할까?

- **저장해야 하는 값: 최대 가치**
- **점화식**
  - **i가 현재 물건의 무게보다 작을 경우:** i번째 물건을 담을 수 없음! 이전 물건 가치 복사
  - **i가 현재 물건의 무게보다 클 경우:**
    i. i번째 물건을 담을 수 있다.
    ii. 물건을 담았을 때와 담지 않을 때(= 이전 값 그대로 둘 때)의 가치를 비교해 더 가치가 큰 값을 할당한다.

Hint

thing	0	1
i = 1	6	13
i = 2	4	8
i = 3	3	6
i = 4	5	12

Knapsack	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	13	13
2	0	0	0	0	8	8	13	13
3	0	0	0	6	8	8	13	14
4	0	0	6	8	8	12	13	14

Step 4. DP 유형이다!

> 어떤 점화식이 들어가야 할까? 무슨 값을 저장해야 할까?

- **저장해야 하는 값: 최대 가치**
- **점화식**
  - **i가 현재 물건의 무게보다 작을 경우:** i번째 물건을 담을 수 없음! 이전 물건 가치 복사

    ```
    dp[i][j] = dp[i][j-1]
    ```

  - **i가 현재 물건의 무게보다 클 경우:**
    i. i번째 물건을 담을 수 있다.
    ii. 물건을 담았을 때와 담지 않을 때(= 이전 값 그대로 둘 때)의 가치를 비교해 더 가치가 큰 값을 할당한다.

    ```
    dp[i][j] = max(dp[i][j-1], dp[i-w][j-1] + v)
    ```

Hint

Step 5. ?표 부분을 구현해보자.

```
n, k = map(int, input().split()) # 물품의 수 n, 준서가 버틸 수 있는 무게 k

thing = [[0,0]]
knapsack = [[0]*(k+1) for _ in range(n+1)]

for i in range(n):
    thing.append(list(map(int, input().split())))
```

?

thing	0	1
i = 1	6	13
i = 2	4	8
i = 3	3	6
i = 4	5	12

Knapsack	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	13	13
2	0	0	0	0	8	8	13	13
3	0	0	0	6	8	8	13	14
4	0	0	6	8	8	12	13	14

```
print(knapsack[n][k])
```

✨Solution

```python
import sys
input = sys.stdin.readline

# 입력
N, K = map(int, input().split()) # N: 물품의 수, K: 버틸 수 있는 무게
items = [[0, 0]]
for _ in range(N):
    items.append(list(map(int, input().split())))

# DP
knapsack = [[0] * (K+1) for _ in range(N+1)]

for i in range(1, N+1):
    for j in range(1, K+1):
        weight = items[i][0]
        value = items[i][1]
        if j >= weight:

            knapsack[i][j] = max(knapsack[i-1][j], knapsack[i-1][j-weight] + value)

        else:
            knapsack[i][j] = knapsack[i-1][j]


print(knapsack[N][K])
```

대체로 코드가 비슷하다.

1. 코드 1
2. 코드 2

✨Solution

Knapsack 알고리즘

Fraction Knapsack(분할가능 배낭문제) : 담을 수 있는 물건이 나누어질 때
(ex. 설탕)
-> greedy로 해결 가능

0-1 Knapsack(0-1 배낭문제) : 담을 수 있는 물건이 나누어질 수 없을 때
(담는다 or 안 담는다)
-> dp로 해결가능
참고로, 백트래킹을 이용하여서도 구현 가능하다

✨Solution

Knapsack 알고리즘

0-1 Knapsack

본 문제, #12865번은 물건을 자를 수 없으므로 0-1 Knapsacp에 해당

문제를 다시 리뷰해보면

- 담을 수 있는 무게는 K
- 물건의 무게W와 가치 V
- 가방에 최대로 담았을 때, 최대의 가치값을 구함

How use?

냅색알고리즘은 어떤 방법을 이용할까?

⇒ 동적프로그래밍 (dp)

📌 가방에 최대 `K` kg까지 물건을 담을 수 있고,
`dp[i][j]` = 가방에 담은 물건의 무게 합이 `j` 일때, 처음 `i`개의 물건 중 담을 수 있는 최대 가치 (1 < j ≤ K)

1. j가 현재 물건 무게 W보다 작을 때
   - 현재 물건을 담을 수 없음 → 이전의 값 복사

```
dp[i][j] = dp[i-1][j]
```

2. j가 현재 물건의 무게 W와 같거나 클 때
   - 현재 물건 담을 수 있다.
   - 물건을 담았을 때와 담지 않았을 때의 가치를 비교해준 뒤 더 큰 값을 할당한다.
   - 현재 물건의 가치는 V이다.

```
dp[i][j] = max( dp[i-1][j] , dp[i-1][j-w] + v)
```

3. 따라서 물건의 **최대 가치**는 `dp[가방크기][물건개수]`로 구할 수 있다.

💻 Assignment

백준 #17845. 수강 과목 (골드5)

> *DP 문제이면서 , Knapsack 문제.*

오늘 문제 풀이와 관련하여 참고하면 좋을 자료 모음 (문제에 대한 정답X, 공부하기 좋은 자료)

- 배낭 문제(KnapSack Problem) 그림으로 쉽게 이해하기
- [알고리즘 트레이닝] 5장 - 동적계획법과 냅색(Knapsack) (백준 12865번 평범한 배낭 문제로 살펴보기)
- 나무위키 - 배낭 문제