SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

APS 응용

SAMSUNG SW ACADEMY FOR YOUTH

목차

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

배낭 채우기

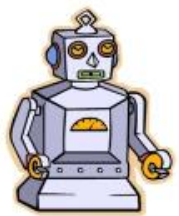
문제 제시 : 생일 선물

- 10kg 용량의 배낭에 4가지 선물 중 선택해서 넣을 수 있다.
  최대 가치가 되도록 선택하려면?

5kg/10만원

4kg/40만원

6kg/30만원



3kg/50만원

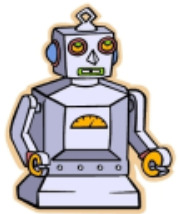
문제 제시 : 생일 선물

- 배낭 (Knapsack) 문제는 n개의 물건과 각 물건 i의 무게 w_i 와 가치 v_i가 주어지고, 배낭의 용량은 W일 때, 배낭에 담을 수 있는 물건의 최대 가치를 찾는 문제이다. 단, 배낭에 담은 물건의 무게의 합이 W를 초과하지 말아야 하고, 각 물건은 1개씩만 있다.

5kg/10만원

4kg/40만원

6kg/30만원



3kg/50만원

배낭 채우기(Knapsack)

Knapsack 문제의 정형적 정의

- S = { $item_1$, $item_2$, . . ., $item_n$ }, 물건들의 집합
- w_i : $item_i$ 의 무게, P_i = $item_i$ 의 값
- W : 배낭이 수용 가능한 총 무게

- 문제 정의

$\sum_{item_i \in A} w_i \leq W$를 만족하면서 $\sum_{item_i \in A} P_i$가 최대가 되도록

$A \subseteq S$가 되는 A 를 결정하는 문제

배낭 채우기(Knapsack)

Knapsack 문제 유형

- 0-1 Knapsack
  - 배낭에 물건을 통째로 담아야 하는 문제
  - 물건을 쪼갤 수 없는 경우

- Fractional Knapsack
  - 물건을 부분적으로 담는 것이 허용되는 문제
  - 물건을 쪼갤 수 있는 경우

배낭 채우기(Knapsack)

- **Knapsack에 대한 탐욕적 방법1**
  - 값이 비싼 물건부터 채운다.

- **Knapsack에 대한 탐욕적 방법2**
  - 무게가 가벼운 물건부터 채운다.

- **Knapsack에 대한 탐욕적 방법3**
  - 무게 당 (예> kg당) 값이 높은 순서로 물건을 채운다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

동적 계획법 활용 - 0/1 Knapsack

0-1 Knapsack

- 배낭 문제를 DP로 접근해 보자.
- 먼저 배낭 문제의 부분 문제를 찾아내기 위해 문제의 주어진 조건을 살펴보면
  - 물건, 물건의 무게, 물건의 가치, 배낭의 용량, 모두 4가지의 요소가 있다.
- 이 중에서 물건과 물건의 무게는 부분 문제를 정의하는데 반드시 필요하다.
- 왜냐하면 배낭이 비어 있는 상태에서 시작하여 물건을 하나씩 배낭에 담는 것과 안 담는 것을 현재 배낭에 들어 있는 물건의 가치의 합에 근거하여 결정해야 하기 때문이다.
- 또한 물건을 배낭에 담으려고 할 경우에 배낭 용량의 초과 여부를 검사해야 한다.

0-1 Knapsack

- **따라서 배낭 문제의 부분문제를 아래와 같이 정의할 수 있다.**
  - W = 배낭의 용량(kg)
  - (v_i, w_i) = 가치(만원), 무게(kg) 물건
  - K[i, w] = 물건 1 ~ i까지만 고려하고, (임시) 배낭의 용량이 w일 때의 최대 가치
    단, i = 1, 2, ···, n이고, w = 1, 2, 3, ···, W이다.

- **K[i, w]를 재귀적으로 정리하면**

$$K[i, w] = \begin{cases} 0 & \text{if } i = 0 \text{ or } w = 0 \\ K[i-1, w] & \text{if } w_i > w \\ \max(v_i + K[i-1, w-w_i], K[i-1, w]) & \text{if } i > 0 \text{ and } w_i \le w \end{cases}$$

0-1 Knapsack

i번째 물건을 고려 할 때

$$K[i, w] = \begin{cases} 0 & \text{if } i = 0 \text{ or } w = 0 \\ K[i-1, w] & \text{if } w_i > w \\ \max(v_i + K[i-1, w-w_i], K[i-1, w]) & \text{if } i > 0 \text{ and } w_i \le w \end{cases}$$

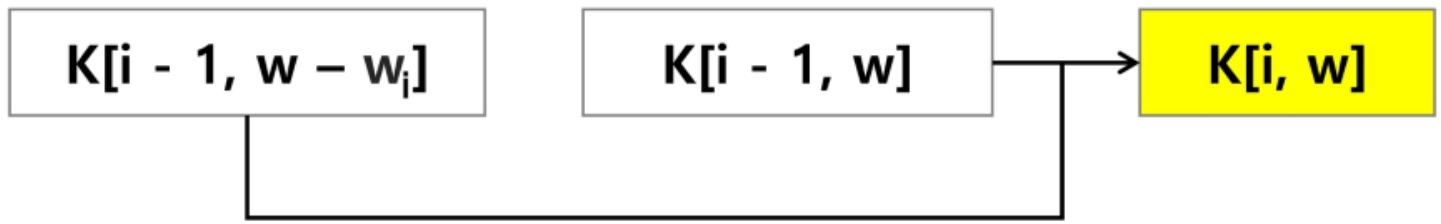
- Case1: 최적해는 물건i를 포함하지 않는다.
  - 전체 가치는 그 전(물건1 ~ (i - 1)까지 고려한 상항)과 동일하다.

- Case2 : 최적해는 물건i를 포함한다.
  - 전체 가치는 물건i의 가치 + 물건1 ~ (i - 1)까지 고려하여 배낭의 용량이 $(w - w_i)$인 경우의 최대 가치

w_i: 물건 i를 담을 공간을 의미

0-1 Knapsack

- 배낭 문제의 부분 문제간의 **함축적 순서**는 다음과 같다.
- 즉, 2개의 부분 문제 $K[i-1,w-w_i]$과 $K[i-1,w]$가 미리 계산되어 있어야만 $K[i,w]$를 계산할 수 있다.



0-1 Knapsack 알고리즘

배낭의 용량 W
n개의 물건과 각 물건 i의 무게 w_i와 가치 v_i, 단, i = 1, 2, …, n
K[n, W]

```
FOR i in 0 → n : K[i, 0] ← 0
FOR w in 0 → W : K[0, w] ← 0

FOR i in 1 → n
        FOR w in 1 → W
                IF w_i > w
                        K[i, w] ← K[i - 1, w]
                ELSE
                        K[i, w] ← max(v_i + K[i-1, w - w_i], K[i-1, w])
RETURN K[n, W]
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

동적 계획법 활용 - 0/1 Knapsack 수행 과정

0-1 Knapsack 알고리즘 수행 과정

- 배낭의 용량 W = 10kg이고, 각 물건의 무게와 가치는 다음과 같다.

i	v	w
1	10	5
2	40	4
3	30	6
4	50	3

0-1 Knapsack 알고리즘 수행 과정

- Line 1~2에서는 아래와 같이 배열의 0번 행과 0번 열의 각 원소를 0으로 초기화한다.

```
FOR i in 0 → n : K[i, 0] ← 0
FOR w in 0 → W : K[0, w] ← 0
FOR i in 1 → n
   FOR w in 1 → W
     IF w_i > w
        K[i, w] ← K[i - 1, w]
     ELSE
        K[i, w]
          ← max(v_i + K[i-1, w - w_i], K[i-1, w])
RETURN K[n, W]
```

i	v	w
1	10	5
2	40	4
3	30	6
4	50	3

W=10

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0										
2	{1, 2}	0										
3	{1, 2, 3}	0										
4	{1, 2, 3, 4}	0										

- Line 3에서는 물건을 하나씩 고려하기 위해서, 물건 번호 i가 1~4까지 변하며, line 4에서는 배낭의 (임시) 용량 w가 1kg씩 증가되어 마지막엔 배낭의 용량인 10kg이 된다.

0-1 Knapsack 알고리즘 수행 과정

- i=1일 때 (즉, 물건 1만을 고려한다.)

- w=1 (배낭의 용량이 1kg)일 때, 물건 1을 배낭에 담아보려고 한다. 그러나 w_1>w 이므로, (즉, 물건 1의 무게가 5kg이므로, 배낭에 담을 수 없기 때문에) K[1,1] = K[i-1,w] = K[1-1,1] = K[0,1] = 0이다.

  K[1,1]=0

5kg/10만원

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0									
	...	.										

0-1 Knapsack 알고리즘 수행 과정

- w=2, 3, 4일 때, 각각 w_1>w 이므로, 물건 1을 담을 수 없다.

  즉, 배낭의 용량을 4kg까지 늘려 봐도 5kg의 물건 1을 배낭에 담을 수 없다.

  K[i-1,w]

  K[1,2]=0, K[1,3]=0, K[1,4]=0

5kg/10만원

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0						
	...	.										

0-1 Knapsack 알고리즘 수행 과정

- w=5 (배낭의 용량이 5kg)일 때, 물건 1을 배낭에 담을 수 있다.
  왜냐하면 w_1=w이므로, 즉, 물건 1의 무게가 5kg이기 때문이다.
  따라서

$K[1, 5] = \max(K[i-1, w], K[i-1, w-w_i] + v_i)$

$= \max(K[1-1, 5], K[1-1, 5-5] + 10)$

$= \max(K[0, 5], K[0, 0] + 10)$

$= \max(0, 0 + 10)$

$= \max(0, 10) = 10$

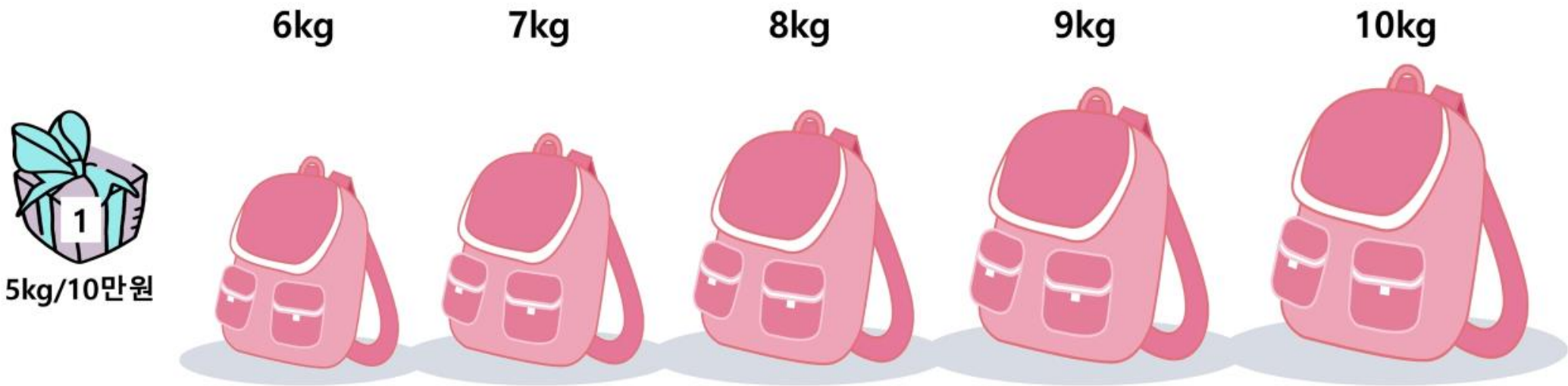
5kg/10만원

5kg

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0	10					
	…	.										

0-1 Knapsack 알고리즘 수행 과정

- w=6, 7, 8, 9, 10일 때, 각각의 경우가 w=5일 때와 마찬가지로 물건 1을 담을 수 있다. 따라서 각각 K[1,6] = K[1,7] = K[1,8] = K[1,9] = K[1,10] = 10이다.



K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0	10	10	10	10	10	10
	...	.										

0-1 Knapsack 알고리즘 수행 과정

- i=2일 때 (즉, 물건 1에 대한 부분 문제들의 해는 i=1일 때 위에서 이미 구하였고, 이를 이용하여 물건 2를 고려한다.)

- w=1, 2, 3 (배낭의 용량이 각각 1, 2, 3kg)일 때, 물건 2를 배낭에 담아보려고 한다. 그러나 w_2>w이므로, 즉, 물건 2의 무게가 4kg이므로, 배낭에 담을 수 없다.
- K[2,1]=0, K[2,2]=0, K[2,3]=0

4kg/40만원

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0	10	10	10	10	10	10
2	{1,2}	0	0	0	0							
	...	.										

0-1 Knapsack 알고리즘 수행 과정

- **w=4 (배낭의 용량이 4kg)일 때, 물건 2를 배낭에 담을 수 있다.**

$K[2, 4] = \max(K[i-1, w], K[i-1, w-w_i] + v_i)$

$= \max(K[2-1, 4], K[2-1, 4-4] + 40)$

$= \max(K[1, 4], K[1, 0] + 40)$

$= \max(0, 0 + 40)$

$= \max(0, 40) = 40$

4kg/40만원

4kg

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0	10	10	10	10	10	10
2	{1,2}	0	0	0	0	40						
	...	.										

0-1 Knapsack 알고리즘 수행 과정

- w=6, 7, 8일 때, 각각의 경우도 **물건 1을 빼내고 물건 2를 배낭에 담는 것**이 더 큰 가치를 얻는다. 따라서 각각 K[2,6] = K[2,7] = K[2,8] = 40이 된다.

4kg/40만원

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0	10	10	10	10	10	10
2	{1,2}	0	0	0	0	40	40	40	40	40		
	…	.										

0-1 Knapsack 알고리즘 수행 과정

- w=9 (배낭의 용량이 9kg)일 때, 물건 2를 배낭에 담아보려고 한다.
  역시, 물건 2를 배낭에 담을 수 있다.

$$K[2, 9] = \max(K[i-1, w], K[i-1, w-w_i] + v_i)$$
$$= \max(K[2-1, 9], K[2-1, 9-4] + 40)$$
$$= \max(K[1, 9], K[1, 5] + 40)$$
$$= \max(10, 10 + 40)$$
$$= \max(10, 50) = 50$$

4kg/40만원

5kg/10만원

9kg

- 즉, 이때에는 배낭에 물건 1, 2 둘 다를 담을 수 있는 것이고, 그때의 가치가 50이 된다는 의미이다.

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0	10	10	10	10	10	10
2	{1,2}	0	0	0	0	40	40	40	40	40	50	
	…	.										

0-1 Knapsack 알고리즘 수행 과정

- **w=10 (배낭의 용량이 10kg)일 때, 물건 2를 배낭에 담을 수 있다, w=9일 때와 마찬가지로 K[2,10]=50이고, 물건 1, 2를 배낭에 둘 다 담을 때의 가치인 50을 얻는다는 의미이다.**

$K[2, 10] = \max(K[i-1, w], K[i-1, w-w_i] + v_i)$

$= \max(K[2-1, 10], K[2-1, 10-4] + 40)$

$= \max(K[1, 10], K[1, 6] + 40)$

$= \max(10, 10 + 40)$

$= \max(10, 50) = 50$

4kg/40만원

5kg/10만원

10kg

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0	10	10	10	10	10	10
2	{1,2}	0	0	0	0	40	40	40	40	40	50	50
	…	.										

0-1 Knapsack 알고리즘 수행 과정

- i=3,4일 때에 대해서도 처리한다.
- 다음은 물건 3과 4에 대해서 배낭의 용량을 1부터 W(10)까지 늘려가며 알고리즘을 수행한 결과이다.

i	v	w
1	10	5
2	40	4
3	30	6
4	50	3

K[i, w]		w										
i	고려 물건들	0	1	2	3	4	5	6	7	8	9	10
0	{ }	0	0	0	0	0	0	0	0	0	0	0
1	{1}	0	0	0	0	0	10	10	10	10	10	10
2	{1,2}	0	0	0	0	40	40	40	40	40	50	50
3	{1,2,3}	0	0	0	0	40	40	40	40	40	50	70
4	{1,2,3,4}	0	0	0	50	50	50	50	90	90	90	**90**

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미