Knapsack Problem

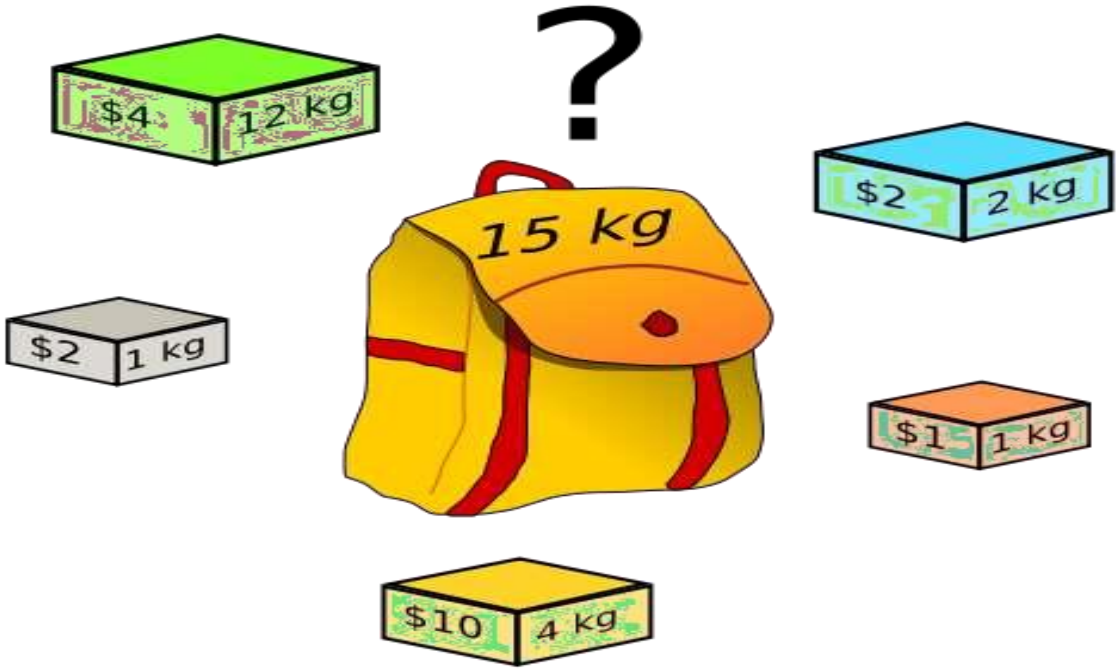
Mashiwat Tabassum Waishy

Lecturer

Department of CSE

STAMFORD UNIVERSITY BANGLADESH

Definition



Knapsack problem states that, Given a set of items, each with a weight and a profit. Determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total profit is as large as possible.

Version of Knapsack Problem

There are two versions of the problem:

1. "0-1 knapsack problem"
   - Items are indivisible; you either take an item or not.
   - Some special instances can be solved with *dynamic programming*

2. "Fractional knapsack problem"
   - Items are divisible: you can take any fraction of an item .
   - Solved using greedy method.

Explanation

- We are given n objects and a knapsack or a bag.
- Object i has a weigh w_i and the knapsack has a capacity m.
- If a fraction x_i, $0 \leq x_i \leq 1$, of object i is placed into the knapsack, then a profit of $p_i x_i$ is earned.
- The objective is to obtain a filling of the knapsack that maximizes the total profit earned.
- Since the total capacity of the knapsack is m, we require the total weight of all chosen objects to be at most m.
- Formally the problem can be stated as,

$$maximize \sum_{1 \leq i \leq n} p_i x_i$$

$$subject\ to \sum_{1 \leq i \leq n} w_i x_i \leq m$$

$$\text{and} \quad 0 \leq x_i \leq 1, 1 \leq i \leq n$$

Explanation

Camera
Weight: 1 kg
Value: 1000$

Laptop
Weight: 3 kg
Value: 2000$

Necklace
Weight: 4 kg
Value: 4000$

Vase
Weight: 5 kg
Value: 4500$

Knapsack
Capacity: 7 kg
Max value: ???

Greedy Approach

- Either we can take the most valuable items until maximum weight is reached.
- Or can take less weight items to fill the maximum weight with most items so that we can get maximum profit.
- Best approach is to fill the knapsack with the highest unit price (profit/ weight) items until maximum weight is reached.

Greedy Algorithm

- Take as much of the item with the highest value per unit (p_i / w_i) as you can. If you run out of that item, take from the next highest (p_i / w_i) item. Continue until knapsack is full.

Example of Fractional Knapsack

- Number of items n=3, Capacity m=20, & the weight and profits are given below:

Weight(w_i)	**18**	**15**	**10**
Profit (p_i)	**25**	**24**	**15**

Now find out the fraction of chosen items with maximum p.

- ***<u>Greedy approach to solve Fractional knapsack problem:</u>***
  - ➢Find the unit u, using the formula $u_i = {}^{p_i}/_{w_i}$
  - ➢Find the fraction of the items x_i that will be taken in order to get maximum profit.

Solution

Step1: Find the unit u_i.

Step 2: Sort the item in descending order of u_i.

Step 3: Find the maximum profit & fraction x_i of the items.

<u>Step1:</u>

$\boldsymbol{w_i}$	18	15	10
$\boldsymbol{p_i}$	25	24	15
$\boldsymbol{u_i = {}^{p_i}/_{w_i}}$	1.4	1.6	1.5

<u>Step 2:</u>

$\boldsymbol{w_i}$	15	10	18
$\boldsymbol{p_i}$	24	15	25
$\boldsymbol{u_i = {}^{p_i}/_{w_i}}$	1.6	1.5	1.4

Solution

Step 3:

i.

w_i	15	10	18
p_i	24	15	25
$u_i = {p_i}/{w_i}$	1.6	1.5	1.4
x_i	0	0	0

Total Capacity, m=20
Rest Capacity, U=20-0=20

ii.

w_i	15	10	18
p_i	24	15	25
$u_i = {p_i}/{w_i}$	1.6	1.5	1.4
x_i	1	0	0

Rest Capacity, U=20-15=5

Solution

Step 3:

iii.

$\boldsymbol{w_i}$	15	10	18
$\boldsymbol{p_i}$	24	15	25
$\boldsymbol{u_i = {p_i}/{w_i}}$	1.6	1.5	1.4
$\boldsymbol{x_i}$	1	5/10 = 1/2	0

Rest Capacity, U=5-5=0

$$\boldsymbol{Maximum\ profit} = \sum_{1 \le i \le n} p_i x_i = (24 * 1) + \left(15 * \frac{1}{2}\right) + (25 * 0)$$

= 31.5

Total weight $= \sum_{1 \le i \le n} w_i x_i = (15 * 1) + \left(10 * \frac{1}{2}\right) + (18 * 0)$ = 20

Fraction taken of the items: $(x_1, x_2, x_3) =$ (0, 1, 0.5)

and $0 \le x_i \le 1, 1 \le i \le n$

Example

- Number of items n=7, Capacity m=15, & the weight and profits are given below:

Objects (O_i)	1	2	3	4	5	6	7
Profit (p_i)	10	5	15	7	6	18	3
Weight (w_i)	2	3	5	7	1	4	1

Solution

Step1: Find the unit u_i.

Step 2: Sort the item in descending order of u_i.

Step 3: Find the maximum profit & fraction x_i of the items.

Step1:

O_i	1	2	3	4	5	6	7
p_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
$u_i = {p_i}/{w_i}$	5	1.6	3	1	6	4.5	3

Step 2:

O_i	5	1	6	3	7	2	4
p_i	6	10	18	15	3	5	7
w_i	1	2	4	5	1	3	7
$u_i = {p_i}/{w_i}$	6	5	4.5	3	3	1.6	1

Solution

Step 3:

O_i	5	1	6	3	7	2	4
p_i	6	10	18	15	3	5	7
w_i	1	2	4	5	1	3	7
$u_i = {p_i}/{w_i}$	6	5	4.5	3	3	1.6	1
x_i	0	0	0	0	0	0	0

Total Capacity, m=15
Rest Capacity, U=15-0=15

O_i	5	1	6	3	7	2	4
p_i	6	10	18	15	3	5	7
w_i	1	2	4	5	1	3	7
$u_i = {p_i}/{w_i}$	6	5	4.5	3	3	1.6	1
x_i	1	0	0	0	0	0	0

Rest Capacity, U=15-1=14

Solution

O_i	5	1	6	3	7	2	4
p_i	6	10	18	15	3	5	7
w_i	1	2	4	5	1	3	7
$u_i = {p_i}/{w_i}$	6	5	4.5	3	3	1.6	1
x_i	1	1	0	0	0	0	0

Rest Capacity, U=14-2=12

O_i	5	1	6	3	7	2	4
p_i	6	10	18	15	3	5	7
w_i	1	2	4	5	1	3	7
$u_i = {p_i}/{w_i}$	6	5	4.5	3	3	1.6	1
x_i	1	1	1	0	0	0	0

Rest Capacity, U=12- 4=8

Solution

O_i	5	1	6	3	7	2	4
p_i	6	10	18	15	3	5	7
w_i	1	2	4	5	1	3	7
$u_i = {p_i}/{w_i}$	6	5	4.5	3	3	1.6	1
x_i	1	1	1	1	0	0	0

Rest Capacity, U=8 - 5=3

O_i	5	1	6	3	7	2	4
p_i	6	10	18	15	3	5	7
w_i	1	2	4	5	1	3	7
$u_i = {p_i}/{w_i}$	6	5	4.5	3	3	1.6	1
x_i	1	1	1	1	1	0	0

Rest Capacity, U=3 - 1=2

Solution

$\boldsymbol{O_i}$	5	1	6	3	7	2	4
$\boldsymbol{p_i}$	6	10	18	15	3	5	7
$\boldsymbol{w_i}$	1	2	4	5	1	3	7
$\boldsymbol{u_i = {}^{p_i}/_{w_i}}$	6	5	4.5	3	3	1.6	1
$\boldsymbol{x_i}$	1	1	1	1	1	2/3	0

Rest Capacity, U= 2 – (3*2/3)
= 0

$$\boldsymbol{Maximum\ profit} = \sum p_i x_i = (6 * 1) + (10 * 1) + (18 * 1) + (15 * 1) + (3 * 1) + (5 * 1.6) + (7 * 0)$$

= 60

Total weight =$\sum w_i x_i = (1 * 1) + (2 * 1) + (4 + 1) + (5 * 1) + (1 * 1) + \left(3 * \frac{2}{3}\right) + (1 * 0)$ = 15

Fraction taken of the items: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ = (1, 2/3, 1, 0, 1, 1, 1)

and $0 \le x_i \le 1, 1 \le i \le n$

Algorithm

Greedy Knapsack(m,n)

```
//p[1:n] and w[1:n] contain the profits and weights respectively of n objects ordered such that
//p[i]/w[i] ≥ p[i+1]/w[i+1].
//m is the knapsack size and x[1:n] is the solution vector
{
        for i :=1 to n do
                x[i]=0.0; //Initialize x
        U := m;
        for i :=1 to n do
        {
            if (w[i] > U) then break;
            x[i] := 1.0; U := U – w[i];
        }
        if (i ≤ n) then x[i] := U/w[i];
}
```

Analysis

- The main **time** taking step is the sorting of all items in decreasing order of their value / weight ratio. If the items are already arranged in the required order, then loop takes O(n) **time**.
- If the sorting is done using Quick sort technique,The average **time complexity** of Quick Sort is O(nlogn).
- Therefore, total **time** taken including the sort is O(nlogn).

Thank you