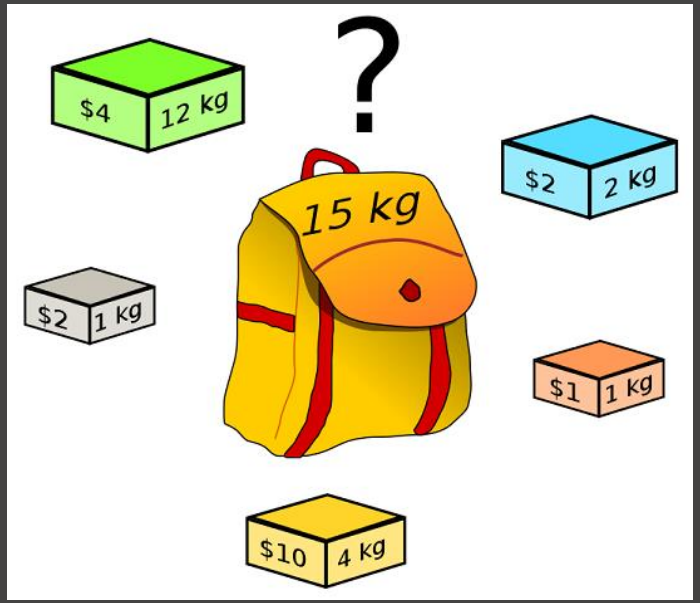
Knapsack Problem

What is Knapsack Problem?

Given a Knapsack of a maximum capacity of W and N items each with its own value and weight, we have to throw in items inside the Knapsack such that the final contents have the maximum value.



Here, the general way the problem is explained – Consider a thief gets into a home to rob and he carries a knapsack. There are fixed number of items in the home. Each with its own weight and value – Jewelry with less weight and highest value vs with less value but a lot heavy. To add fuel to the fire, the thief has an old knapsack which has limited capacity. So, he has to chose a way which will maximize his profit.

There are two types of knapsack problem:

01

0/1 knapsack Problem

Here, items are indivisible. We either take them or not. And it is solved using Dynamic Programming.

02

Fractional Knapsack Problem

Items are divisible. We can take any fraction of an item. And it is solved using Greedy algorithm.

Fractional Knapsack Problem

Example of Fractional Knapsack

Here, the items are divisible into any fraction as per necessity. And this is solved using Greedy Algorithm.

Example: Let, the number of items n=4, capacity of the Knapsack m=60. The individual weight and profit/value of the items are given.

Item	A	B	C	D
Profit (P)	280	100	120	120
Weight (W)	40	10	20	24

Solution:

1) A first we find the ratio of profit by weight i.e. P/W.

Item	A	B	C	D
Profit(P)	280	100	120	120
Weight (W)	40	20	20	24
Ratio (P/W)	7	10	6	5

2) We sort the items according to the descending order of the ratio (P/W).

Item	B	A	C	D
Profit(P)	100	280	120	120
Weight (W)	10	40	20	24
Ratio (P/W)	10	7	6	5

3) We find the maximum profit and fraction of the items that can be taken in the knapsack.

Item	B	A	C	D
Profit(P)	100	280	120	120
Weight (W)	10	40	20	24
Ratio(P/W)	10	7	6	5
Fraction	1	1	10/20=1/2	0

First of all, ***B*** is chosen as weight of ***B*** is less than the capacity of the knapsack. Next, item ***A*** is chosen, as the available capacity of the knapsack is greater than the weight of ***A***. Now, ***C*** is chosen as the next item. However, the whole item cannot be chosen as the remaining capacity of the knapsack is less than the weight of ***C***.
Hence, fraction of ***C*** (i.e. (60 − 50)/20) is chosen.
Now, the capacity of the Knapsack is equal to the selected items. Hence, no more item can be selected.
The total weight of the selected items is **10 + 40 + 20 * (10/20) = 60**
And the total profit is **100 + 280 + 120 * (10/20) = 380 + 60 = 440**

Pseudocode

```
for i = 1 to n
 do x[i] = 0
weight = 0
for I = 1 to n
   if weight + w[i] <= W then
      x[i] = 1
      weight = weight + w[i]
else
   x[i] = (W-weight) / w[i]
   weight = W
   break
return x
```

Working code:

```
# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity) {
   float x[20], tp = 0;
   int i, j, u;
   u = capacity;

   for (i = 0; i < n; i++)
      x[i] = 0.0;

   for (i = 0; i < n; i++) {
      if (weight[i] > u)
         break;
      else {
         x[i] = 1.0;
         tp = tp + profit[i];
         u = u - weight[i];
      }
   }
```

```
 if (i < n)
      x[i] = u / weight[i];

   tp = tp + (x[i] * profit[i]);

   printf("\nThe result vector is:- ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);

   printf("\nMaximum profit is:- %f", tp);

}

int main() {
   float weight[20], profit[20], capacity;
   int num, i, j;
   float ratio[20], temp;

   printf("\nEnter the no. of items:- ");
   scanf("%d", &num);

   printf("\nEnter the weights and profits of each item:- ");
   for (i = 0; i < num; i++) {
      scanf("%f %f", &weight[i], &profit[i]);
   }
```

```
printf("\nEnter the capacity of knapsack:- ");
   scanf("%f", &capacity);

   for (i = 0; i < num; i++) {
      ratio[i] = profit[i] / weight[i];
   }

   for (i = 0; i < num; i++) {
      for (j = i + 1; j < num; j++) {
         if (ratio[i] < ratio[j]) {
            temp = ratio[j];
            ratio[j] = ratio[i];
            ratio[i] = temp;

            temp = weight[j];
            weight[j] = weight[i];
            weight[i] = temp;

            temp = profit[j];
            profit[j] = profit[i];
            profit[i] = temp;
         }
      }
   }

   knapsack(num, weight, profit, capacity);
   return(0);
}
```

Sample output:

```
Enter the no. of items:- 4

Enter the weights and profits of each item:- 40 280
10 100
20 120
24 120

Enter the capacity of knapsack:- 60

The result vector is:- 1.000000 1.000000        0.500000        0.000000
Maximum profit is:- 440.000000
Press any key to continue . . .
```

Complexity:

First sort according to profit to weight ratio time required: nlogn
Then we need one loop to find out maximum profit and for that tim
e needed is: n.

Overall time complexity:

nlogn + n = O(nlogn)

Application of fractional knapsack problem:

One application of this algorithm is int download managers (eg: Internet Download Manager) .
The data is broken into chunks. As per the maximum size of data that can be retrieved in one go , the server uses this algorithm and packs the chunks so as to utilize the full size limit.
This algorithm is one of the many algorithms that download managers use apart from compressing, encrypting etc etc.