Fractional Knapsack Problem using Greedy algorithm

The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on the basis of this ratio.

Then take the item with the highest ratio and add them until we can’t add the next item as a whole and at the end add the next item as much as we can.

Which will always be the optimal solution to this problem.

The knapsack problem is in combinatorial optimization problem.

It appears as a subproblem in many, more complex mathematical models of real-world problems.

One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, and somehow adjust the solution to satisfy the ignored constraints.

Problem Statement

A thief is robbing a store and can carry a maximal weight of W into his knapsack. There are n items available in the store and weight of i^{th} item is w_i and its profit is p_i. What items should the thief take?

In this context, the items should be selected in such a way that the thief will carry those items for which he will gain maximum profit. Hence, the objective of the thief is to maximize the profit.

Based on the nature of the items, Knapsack problems are categorized as

- Fractional Knapsack
- Knapsack

According to the problem statement,

- There are n items in the store
- Weight of ith item wi>0wi>0
- Profit for ith item pi>0pi>0 and
- Capacity of the Knapsack is W

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction xi of ith item.

$0⩽xi⩽1$

first, we need to sort those items according to the value of piwi, so that (pi+1)/(wi+1) ≤ pi/wi. Here, x is an array to store the fraction of items.

```
Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)
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
```

Example:

Let us consider that the capacity of the knapsack W = 60 and the list of provided items are shown in the following table −

Item	B	A	C	D
Profit	100	280	120	120
Weight	10	40	20	24
Ratio $\left(\frac{p_i}{w_i}\right)$	10	7	6	5

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

Time Complexity

If the provided items are already sorted into a decreasing order of piwi, then the whileloop takes a time in O(n)

Therefore, the total time including the sort is in O(n logn).