|  |  |
| --- | --- |
| **Ex. No:1** | **Greedy solution for Fractional Knapsack Problem** |
| **Date:8/12/22** |  |

# Objective:

To find the solution for Fractional Knapsack problem using greedy technique.

# Description:

Greedy Algorithm works in phases. At each phase:

* Takes the best solution right now, without regard of our future consequences
* Choose a local optimum at each step, and end up at a global optimal solution

The **knapsack problem** is a **problem** in combinatorial optimization: Given a set of items, each with a weight and a value, 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 value is as large as possible. In the **fractional knapsack problem**, items are divisible and this can be solved in greedy method.

Input: An integer n, positive values wi and vi such that 1≤ i ≤ n and a positive value W

Output: n values of xi such that 0 ≤ xi ≤ 1

Example:

Consider that the capacity of the knapsack *W* = 60 and the list of provided items are shown in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **A** | **B** | **C** | **D** |
| Profit (v[ ]) | 280 | 100 | 120 | 120 |
| Weight (w[ ]) | 40 | 10 | 20 | 24 |
| Ratio w[i] / v[i] | 7 | 10 | 6 | 5 |

The provided items are not sorted based on v[i] / w[i]. After sorting, the items are as shown in the following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **B** | **A** | **C** | **D** |
| Profit (v[ ]) | 100 | 280 | 120 | 120 |
| Weight (w[ ]) | 10 | 40 | 20 | 24 |
| Ratio w[i] / v[i] | 10 | 7 | 6 | 5 |

# Algorithm:

//Input: Items are in the descending order of their value/weight ration Greedy-Fractional-Knapsack (w[1..n], v[1..n], W) {

for (i = 1 to n )

do x[i] = 0; weight = 0; profit = 0.0; for (i = 1 to n) {

if weight + w[i] ≤ W then {

x[i] = 1;

weight = weight + w[i] ; profit = profit +

v[i};

}

else {

x[i] = (W - weight) / w[i];

weight = W; profit = profit + v[i]\*x[i]; break;

}

}

return x

}

# Analysis:

If the provided items are already sorted into a decreasing order of the ratio w[i] / v[i], then the while loop takes a time in ***O(n)***; Therefore, the total time including the sort is in ***O(n logn)***.

# Program (Screen Shot):

def greedy(W,V,w,n):

r=[]

x=[]

p=0.0

for i in range(n):

x.append(0)

r.append(V[i]/W[i])

for i in range(n):

for j in range(n-i-1):

if r[j]<r[j+1]:

r[j],r[j+1]=r[j+1],r[j]

W[j],W[j+1]=W[j+1],W[j]

V[j],V[j+1]=V[j+1],V[j]

elif r[j] == r[j + 1]:

if V[j] < V[j + 1]:

r[j], r[j + 1] = r[j + 1], r[j]

W[j], W[j + 1] = W[j + 1], W[j]

V[j], V[j + 1] = V[j + 1], V[j]

for i in range(n):

if W[i] <= w:

x[i] = 1

w -= W[i]

p += V[i]

else:

x[i] = w / W[i]

p += V[i] \* x[i]

break

return x, p, W, V, r

a = int(input("Enter no of items:"))

Value = [0] \* a

Weight = [0] \* a

for i in range(a):

Value[i] = int(input("Enter value of item:"))

Weight[i] = int(input("Enter weight of item:"))

w\_sack = int(input("Enter weight of sack:"))

sack, profit, we, va, re = greedy(Weight, Value, w\_sack, a)

print("Value -", va)

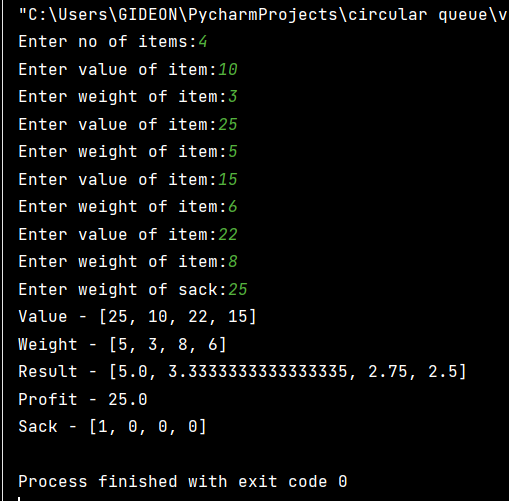
print("Weight -", we)

print("Result -", re)

print("Profit -", profit)

print("Sack -", sack)

**Output (Screen Shot):**

****

# Result: The program is successfully executed.