5) Given N items with their corresponding weights and values, and a package of capacity C, choose either the entire item or fractional part of the item among these N unique items to fill the package such that the package has maximum value.

Aim: To fill a package of capacity C with whole or fraction of items among N items such that the package has maximum value.

Description: The algorithm takes in N items along with their corresponding weights and values, as well as a package with capacity C. The goal is to fill the package with either entire items or fractional parts of items, in such a way that the total value of the items in the package is maximized. It sorts the items by value-to-weight ratio in descending order, and selects each item, adding it to the package either entirely or partially, until the package is full. Finally, the algorithm returns the selected items and their total value.

Design Technique: This algorithm uses a greedy approach to maximize the value of the items in the package by selecting items with the highest value-to-weight ratio first and filling the package with as much of each item as possible. A greedy algorithm is a type of algorithmic approach which makes locally optimal choices with the hope of finding a global optimal solution. Greedy algorithms are often used for optimization problems.

Algorithm:

Input: Capacity of package, number of items with weights and profits of each item

Output: Solution vector(quantities) and total value

1. Algorithm GreedyKnapSack(C,N) :

2. //v[1:N] and w[1:N] contain values and weights respectively of N objects ordered

3. // v[i]/w[i]>v[i+1]/w[i+1]

4. //C is the capacity of the package

5. //x[1:N] is the solution vector

6. {

7. for i :=1 to N do x[i]:=0.0

8. u:=c;

9. for i:=1 to N do

10. {

11. if(w[i]>u) then break;

12. x[i]:=1.0;

13. u=u-w[i];

14. }

15. if (i<=N) then x[i]:=u/w[i]

16. }

Program:

p=[]

w=[]

x=[]

total=0

index=[]

N=int(input("Enter the number of items:"))

C=float(input("Enter the capacity of the bag:"))

for i in range(N):

print("Item",i+1)

p.append(float(input("Enter profit:")))

w.append(float(input("Enter weight:")))

index.append(i+1)

profit=p.copy()

weights=w.copy()

for i in range(N):

max=i

for j in range(i+1,N):

if p[j]/w[j]>p[max]/w[max]:

max=j

p[max],p[i]=p[i],p[max]

w[max],w[i]=w[i],w[max]

index[max],index[i]=index[i],index[max]

for i in range(N):

x.append(0.0)

U=C

for i in range(N):

if w[i]>U:

break

x[i]=1.0

U=U-w[i]

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

x[i]=round(x[i],2)

if i<=N:

x[i]=U/w[i]

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

x[i]=round(x[i],2)

for i in range(N):

j=index.index(i+1)

x.append(x[j])

x=x[N:]

print("\nProfits\tWeights\tQuantities")

for i in range(N):

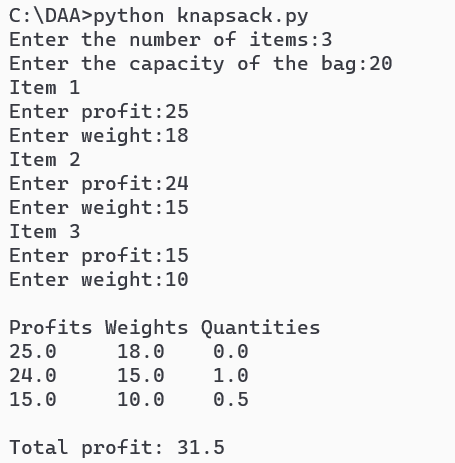
print(profit[i],"\t",weights[i],"\t",x[i],)

print("\nTotal profit:",end=" ")

print(round(total,2))

Output:

*Test Case 1:*



*Test Case 2:*

*Table

Description automatically generated*

*Test Case 3:*

Table

Description automatically generated

Time & Space Complexity:

Time Complexity: O(n2)

Space Complexity: O(n)

Conclusion: In conclusion, the given algorithm is a greedy algorithm that solves the knapsack problem by selecting items with the highest value-to-weight ratio first and filling the package with as much of each item as possible. While greedy algorithms are efficient and commonly used for optimization problems, they may not always produce the optimal solution.

References

https://www.geeksforgeeks.org/greedy-algorithms/