DAA ASSIGNMENT

G.BHAVANA

21071A6719

0/1 Knapsack

In the 0–1 Knapsack problem, we are given a set of items, each with a weight and a value, and we need to 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.

Please note that the items are indivisible; we can either take an item or not (0-1 property). For example,

```
Input:

value = [ 20, 5, 10, 40, 15, 25 ]
weight = [ 1, 2, 3, 8, 7, 4 ]
int W = 10

Output: Knapsack value is 60

value = 20 + 40 = 60
weight = 1 + 8 = 9 < W
```

PROGRAM:

```
def knapSack(W, wt, val, n):
    if n == 0 or W == 0 :
        return 0

    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)

    else:
        return max(val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
                   knapSack(W, wt, val, n-1))



val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print(knapSack(W, wt, val, n))
```

output:

```
knapsack value is 60
```

The steps of the algorithm we'll use to solve our knapsack problem are:

Sort items by worth, in descending order.
Start with the highest worth item. Put items into the bag until the next item on the list cannot fit.
Try to fill any remaining capacity with the next item on the list that can fit.

Pseudo-code for solving knapsack problems using the greedy method is;

```
Greedy fractional-knapsack (P[1…n], W[1…n], X[1..n]. M)
/*P[1…n] and W[1…n] contain the profit and weight of the n-objects ordered such that X[1…n] is a solution set and M is the capacity of knapsack*/
{

Knapsack Problem Using Greedy Method
For j ← 1 to n do
X[j]← 0
Profit ← 0 // Total profit of item filled in the knapsack
Weight ← 0 // Total weight of items packed in knapsacks
J ← 1
While (Weight < M) // M is the knapsack capacity

Signup for Free Mock Test
{

If (weight + W[j] =< M)
X[j] = 1
Weight = weight + W[j]
Else{
X[j] = (M – weight)/w[j]
Weight = M

}
Profit = profit + p[j] * X[j]
J++;
} // end of while
} // end of Algorithm
```

As the name suggests, items are indivisible here.
We can not take the fraction of any item.
We have to either take an item completely or leave it completely.
It is solved using dynamic programming approach.