**Experiment No. - 3.3**

Aim of the Experiment:

Write a program to solve the Knapsack Problem Using Greedy Technique.

**Objective of the Experiment:**

To understand Knapsack Problem Using Greedy Technique.

Algorithm:

for i in range(1,n):

calculate p/w

Sort objects in descending order of p/w ratio

if M>0 and wi<=M:

M = M-wi

p = p + pi

else:

p = p + pi(M/wi)

**Complexity Analysis:**

We begin with a for loop starting from the first object to the last object and the complexity of the for loop is nothing but o(n), now the next step is an important step, we have to sort the objects in descending order of power/weight ratio.

We can use several sorting algorithms like the bubble sort, merge sort, quick sort, and more.

Let us say that we used merge sort for sorting as it is one of the fastest sorting algorithms, then the complexity of merge sort is o(nlogn). Lastly, we have used the if statement, and the complexity of that if statement is 0(n).

Now, as the maximum complexity in our analysis is o(nlogn), therefore the complexity of the knapsack algorithm is 0(nlogn).

**Pseudo Code:**

knapsack(W, arr):

sort(arr)

res = 0

for i in arr:

if W >= i[1]:

res += i[0]

W -= i[1]

else:

res += i[0] \* W / i[1]

break

return res

Source Code:

#include <bits/stdc++.h>

using namespace std;

struct Item {

int profit, weight;

Item(int profit, int weight)

{

this->profit = profit;

this->weight = weight;

}

};

// Comparison function to sort Item

// according to profit/weight ratio

static bool cmp(struct Item a, struct Item b)

{

double r1 = (double)a.profit / (double)a.weight;

double r2 = (double)b.profit / (double)b.weight;

return r1 > r2;

}

// Main greedy function to solve problem

double fractionalKnapsack(int W, struct Item arr[], int N)

{

sort(arr, arr + N, cmp);

double finalvalue = 0.0;

for (int i = 0; i < N; i++) {

if (arr[i].weight <= W) {

W -= arr[i].weight;

finalvalue += arr[i].profit;

}

else {

finalvalue

+= arr[i].profit

\* ((double)W / (double)arr[i].weight);

break;

}

}

return finalvalue;

}

int main()

{

int W = 50;

Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };

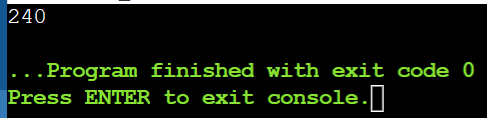
int N = sizeof(arr) / sizeof(arr[0]);

cout << fractionalKnapsack(W, arr, N);

return 0;

}

Output:



**Learning outcomes (What I have learnt):**

* Learnt and implemented Knapsack Problem Using Greedy Technique.
* Learned about the complexity of Knapsack Problem.