Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.

Experiment No. 5

Name: Om Bhutki

UID: 2022301003

Subject: DAA

Experiment No: 5

Aim – Experiment based on greedy approach (fractional knapsack problem).

Objective: Apply the concept of dynamic programming and greedy approach to solve problems.

Theory:

The knapsack problem is the following problem in combinatorial optimization:

Given a set of items, each with a weight and a value, determine which items to include in the collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where the decision-makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively.

The knapsack problem has been studied for more than a century, with early works dating as far back as 1897.[1] The name "knapsack problem" dates back to the early works of the mathematician Tobias Dantzig (1884–1956),[2] and refers to the commonplace problem of packing the most valuable or useful items without overloading the luggage.

Fractional Knapsack Problem

Given the weights and values of N items, in the form of {value, weight} put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In Fractional Knapsack, we can break items for maximizing the total value of the knapsack

This problem can be solved with the help of using two techniques:

- Brute-force approach: The brute-force approach tries all the possible solutions with all the different fractions but it is a time-consuming approach.
- Greedy approach: In Greedy approach, we calculate the ratio of profit/weight, and accordingly, we will select the item. The item with the highest ratio would be selected first.

There are basically three approaches to solve the problem:

- The first approach is to select the item based on the maximum profit.
- The second approach is to select the item based on the minimum weight.
- The third approach is to calculate the ratio of profit/weight.

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
```

Program:

```
#include <bits/stdc++.h>
#include <iostream>
#include <algorithm>
#include <string>
#include <cmath>
using namespace std;

void printarr(double **arr,int n) {
   cout << "item \t\t\t weight \t\t\t Value \t\t\t Value/weight\n";
   for (int i = 0; i < n; i++) {

      for (int j = 0; j < 4; j++) {

         cout << arr[i][j] << "\t\t\t";
      }
      cout << "\n";

   }
}


int main()
{
   int c, n;
   double profit = 0.0,weight = 0.0;

   cout << "Enter the weight of the sack: ";
   cin >> c;
   cout << "Enter the no of items: ";
   cin >> n;
   cout << "Enter weight and value of each item: \n";

   vector<string> s (n);

   double **arr = new double*[n];
   for (int i = 0; i < n; i++) {
      arr[i] = new double[4];
      arr[i][0]=i+1;
      for (int j = 1; j < 4; j++) {

         if (j == 3)
         {
            arr[i][j] = arr[i][2] / arr[i][1];
         }

         else {
         cout << "Enter weight and value for [" << i << "][" << j << "]: ";
         cin >> arr[i][j];
         }
      }
   }
```

```
    printarr(arr,n);
    cout << "Sorted based on ratio: " << endl;

    sort(arr, arr + n, [](const double* a, const double* b) {
        return a[3] > b[3];
    });

    printarr(arr,n);

    int remain = 0;
    double remain_pro = 0.0;
    string coco = "";
    ostringstream ss;

    for (int i = 0; i < n; i++) {
        if (c >= weight + arr[i][1]){
            weight += arr[i][1];
            s[i] = to_string(lround(arr[i][0]));
            profit += arr[i][2];
        }
        else
        {

            remain = c - weight;
            weight += remain;
            remain_pro = (remain * arr[i][2]) / arr[i][1];
            profit += remain_pro;
            ss << remain << "/" << arr[i][1];
            coco = ss.str();
            s[i] = to_string(lround(arr[i][0])) + " (" + coco + ")";
            break;
        }

    }

    cout << "Total weight: " << weight << endl;
    cout << "Total profit: " << profit << endl;
    cout << "All items in the bag: {";
    for (int i = 0; i < s.size(); i++)
    {

        cout << s[i] << " ";
    }
    cout << "}";


    return 0;
}
```

Result:

```
Enter the weight of the sack: 1000
Enter the no of items: 7
Enter weight and value of each item:
Enter weight and value for [0][1]: 20
Enter weight and value for [0][2]: 43
Enter weight and value for [1][1]: 346
Enter weight and value for [1][2]: 246
Enter weight and value for [2][1]: 48
Enter weight and value for [2][2]: 243
Enter weight and value for [3][1]: 578
Enter weight and value for [3][2]: 234
Enter weight and value for [4][1]: 86
Enter weight and value for [4][2]: 389
Enter weight and value for [5][1]: 643
Enter weight and value for [5][2]: 765
Enter weight and value for [6][1]: 253
Enter weight and value for [6][2]: 678
item                    weight                          Value                   Value/weight
1                       20                      43                      2.15
2                       346                     246                     0.710983
3                       48                      243                     5.0625
4                       578                     234                     0.404844
5                       86                      389                     4.52326
6                       643                     765                     1.18974
7                       253                     678                     2.67984
Sorted based on ratio:
item                    weight                          Value                   Value/weight
3                       48                      243                     5.0625
5                       86                      389                     4.52326
7                       253                     678                     2.67984
1                       20                      43                      2.15
6                       643                     765                     1.18974
2                       346                     246                     0.710983
4                       578                     234                     0.404844
Total weight: 1000
Total profit: 2058.51
All items in the bag: {3 5 7 1 6 (593/643)    }
```

Inference:

Thus, we observe that by using the greedy approach which allows us to get the best option possible without worrying about optimization, we can get the maximum profit possible for the total weight by calculating the value and weight ratio and sorting the table accordingly and also get the maximum number of items in the bag.

Conclusion:

Thus, after performing this experiment I understood and implemented the fractional knapsack problem in which you have to put items in a knapsack of capacity W in order to get the maximum total value in the knapsack and we can also break the items in order to maximize the profit.