Chinmay N. Dandekar
SE - A - 9

Experiment No. 4

Aim: Write a Program to implement Fractional Knapsack Problem

Theory:

Fractional Knapsack Problem:

Given a set of items, each with a weight and a value, determine a subset of items 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.
The knapsack problem is in combinatorial optimization problem. It appears as a subproblem in many, more complex mathematical models of real-world problems. One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, and somehow adjust the solution to satisfy the ignored constraints.

Applications:

In many cases of resource allocation along with some constraint, the problem can be derived in a similar way of Knapsack problem. Following is a set of examples.

- Finding the least wasteful way to cut raw materials
- Portfolio optimization
- Cutting stock problems

Chinmay N. Dandekar
SE - A - 9

Algorithm:

Fractional Knapsack Algorithm:

```
Algorithm knapsack Greedy (w, n)
{
    // p[i] contains the profit of i items such
that 1 ≤ i ≤ n
    // w[i] contains weights of items
    // x[i] is the solution vector.
    // W is the total size of knapsack.

    for i = 1 to n do
    {
        if (w[i] < W) then   // capacity of knapsack
                                is a constraint
        {
            x[i] = 1.0;
            W = W - w[i];
        }
    }
    if (i <= n) then
        x[i] = W/w[i];
}
```

Chinmay N. Dandekar
SE - A - 9

Time Complexities:

Fractional Knapsack Problem:

1. Time Complexities

Time complexity of the sorting + Time complexity of the loop to maximize profit = O(N log N) + O(N) = O(N log N)

Space Complexity: O(1).

2. Space Complexity

The space complexity for fractional knapsack is O (1).

Code:

Fractional Knapsack:

```
using namespace std;

#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
int main()
{
    int n, W;
    cout << "Enter number of items: ";
    cin >> n;
    cout << "Enter max capacity: ";
    cin >> W;
    float max_profit;
    int w, p;
    float r;
    vector<pair<float, pair<int, int>>> a;
    vector<float> b;
    float te;

    for (int i = 0; i < n; i++)
    {
        cout << "Weight: ";
        cin >>w;
```

```
        cout << "Profit: ";
        cin >> p;
        r = (float)p / w;

        a.push_back(make_pair(r, make_pair(w, p)));
    }

    sort(a.rbegin(), a.rend());

    for (int i = 0; i < n; i++)
    {
        if (W > 0)
        {
            if (a[i].second.first < W)
            {
                b.push_back(a[i].second.second);
                W = W - a[i].second.first;
            }
            else
            {

                te = (float)a[i].second.second * (float)W /
a[i].second.first;

                b.push_back(te);
                W = W - (a[i].second.first);
            }
        }
        else
            break;
    }

    max_profit = accumulate(b.begin(), b.end(), 0.0f);

    cout << "The Maximum Profit is: "<<max_profit << endl;



    return 0;
}
```

Output:

```
Enter number of items: 4
Enter max capacity: 70
Weight: 20
Profit: 30
Weight: 40
Profit: 20
Weight: 10
Profit: 15
Weight: 30
Profit: 50
The Maximum Profit is: 100
```

Conclusion:

Hence, we studied the implementation of Fractional Knapsack Problem.