Vashisth Zatakia
60003200163 B3 IT

Aim: Implementation of Fractional Knapsack Greedy Method

Theory:

In Fractional Knapsack, we can break items for maximizing the total profit of knapsack. The basic idea of the greedy approach is to calculate the ratio profit/weight for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can.

Problem Scenario

A thief is robbing a store and can carry a maximal weight of ***W*** into his knapsack. There are n items available in the store and weight of ***i^{th}*** item is ***w_i*** and its profit is ***p_i***. What items should the thief take?

In this context, the items should be selected in such a way that the thief will carry those items for which he will gain maximum profit. Hence, the objective of the thief is to maximize the profit.

In Fractional Knapsack, items can be broken into smaller pieces, hence the thief can select fractions of items.

According to the problem statement,

- There are **n** items in the store
- Weight of **i^{th}** item wi>0wi>0
- Profit for **i^{th}** item pi>0pi>0 and
- Capacity of the Knapsack is **W**

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction ***x_i*** of **i^{th}** item.

The **i^{th}** item contributes the weight to the total weight in the knapsack and profit $x_i . p_i$ to the total profit.

Hence, the objective of this algorithm is to

Maximize profit subject to constraint,

$$\sum_{n=1}^{n} (x_i . wi) \leqslant W$$

Complexity Analysis : The main time taking step is the sorting of all items in decreasing order of their profit/weight ratio. If the items are already arranged in the required order, then while loop takes O(n) time. The time complexity of sorting is O(nlogn). Therefore, total time taken including the sort is O(nlogn).

Vashisth Zatakia
60003200163 B3 IT

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Sack
{
    float value, weight;
    int id;
};
int comparator(const void *a, const void *b)
{
    struct Sack *p = (struct Sack *)a;
    struct Sack *p1 = (struct Sack *)b;
    return (p1->value / p1->weight - p->value / p->weight);
}
int main()
{
    int items, i;
    float total_profit = 0, fract, capacity, wht = 0;
    printf("Enter number of items:");
    scanf("%d", &items);
    struct Sack *s = (struct Sack *)malloc(items * sizeof(struct Sack));
    printf("Enter capacity of Sack:");
    scanf("%f", &capacity);
    float flag[items];
    for (i = 0; i < items; i++)
    {
        printf("Enter profit and weight of item %d:", i + 1);
        scanf("%f %f", &s[i].value, &s[i].weight);
        s[i].id = i;
        flag[i] = 0;
    }
```

```
    qsort(s, items, sizeof(struct Sack), comparator);
    for (i = 0; i < items; ++i)
    {
        if (wht + s[i].weight <= capacity)
        {
            wht += s[i].weight;
            total_profit += s[i].value;
            flag[s[i].id] = 1;
            printf("Item %d added in the Sack and capacity= %f\n", s[i].id + 1, capacity - wht);
        }
        else
        {
            fract = ((capacity - wht) / s[i].weight);
            flag[s[i].id] = fract;
            total_profit += fract * s[i].value;
            wht += s[i].weight * fract;
            printf("%f %% of Item %d  is added in the Sack and capacity= %f\n", fract * 100, s[i].id + 1,
capacity - wht);
            break;
        }
    }
    printf("Total profit=%f\n", total_profit);
    printf("Flag array:");
    for (i = 0; i < items; i++)
    {
        printf("%f ", flag[i]);
    }
    return 0;
}
```

Vashisth Zatakia
60003200163 B3 IT

Example

Let us consider that the capacity of the knapsack ***W* = 60** and the list of provided items are shown in the following table −

Item	A	B	C	D
Profit	280	100	120	120
Weight	40	10	20	24
Ratio $(\frac{p_i}{w_i})$	7	10	6	5

As the provided items are not sorted based on $\frac{p_i}{w_i}$. After sorting, the items are as shown in the following table.

Item	B	A	C	D
Profit	100	280	120	120
Weight	10	40	20	24
Ratio $(\frac{p_i}{w_i})$	10	7	6	5

Solution

After sorting all the items according to ratio. First all of ***B*** is chosen as weight of ***B*** is less than the capacity of the knapsack. Next, item ***A*** is chosen, as the available capacity of the knapsack is greater than the weight of ***A***. Now, ***C*** is chosen as the next item. However, the whole item cannot be chosen as the remaining capacity of the knapsack is less than the weight of ***C***.

Hence, fraction of ***C*** (i.e. (60 − 50)/20) is chosen.

Now, the capacity of the Knapsack is equal to the selected items. Hence, no more item can be selected.

The total weight of the selected items is **10 + 40 + 20 * (10/20) = 60**

And the total profit is **100 + 280 + 120 * (10/20) = 380 + 60 = 440**

This is the optimal solution. We cannot gain more profit selecting any different combination of items

Output:

```
Enter number of items:4
Enter capacity of Sack:60
Enter profit and weight of item 1:100 10
Enter profit and weight of item 2:120 24
Enter profit and weight of item 3:120 20
Enter profit and weight of item 4:280 40
Item 1 added in the Sack and capacity= 50.000000
Item 4 added in the Sack and capacity= 10.000000
50.000000 % of Item 3  is added in the Sack and capacity= 0.000000
Total profit=440.000000
Flag array:1.000000 0.000000 0.500000 1.000000
```