Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Name	Tejas Jadhav
UID No.	2022301006
Class	COMPS A (B batch)
Experiment No.	5

Aim: Fractional Knapsack problem

Theory:

- **Greedy Alogrithms**

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.

The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.

Algorithm:

```
knap_sack_greedy(W, n, weight[], profit[]):
   io[] = new float[n]   // array to store the fraction of items taken
   ratio[] = new RatioIndex[n]   // array of RatioIndex pointers

   // calculate ratio of profit to weight for each item
   for i = 0 to n-1:
      ratio[i] = new RatioIndex()
      ratio[i]->r = profit[i] / weight[i]
      ratio[i]->index = i

   // sort items by ratio in descending order
   sort_descending(ratio, n)

   try_w = 0   // current weight of items included in the knapsack
   total_profit = 0   // total profit obtained from included items
```

```
// iterate over the sorted items
for i = 0 to n-1:
    index = ratio[i]->index
    try_w = try_w + weight[index]
    io[index] = 1   // take the entire item

    // if the knapsack is now overfilled, take a fraction of the item instead
    if try_w > W:
        try_w = try_w - weight[index]
        space_needed = W - try_w
        portion_taken = space_needed / weight[index]
        io[index] = portion_taken
        total_profit = total_profit + (portion_taken * profit[index])
    else:
        total_profit = total_profit + profit[index]

// create a new Result object to store the output
res = new Result
res->io_array = io
res->total_profit = total_profit
res->total_weight = try_w

return res
```

Code:

```cpp
#include <iostream>
using namespace std;

// * fractional knapsack using greedy method

struct Result {
    float total_profit;
    float total_weight;
```

```
    float *io_array;

    Result() {
        total_profit = 0;
        total_weight = 0;
        io_array = nullptr;
    }
};

struct RatioIndex {
    int index;
    float r;

    RatioIndex() {
        index = -1;
        r = 0;
    }
};

void sort_descending(RatioIndex **a, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (a[i]->r < a[j]->r) {
                RatioIndex *temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
}

void print_array(float *a, int n) {
    cout << "[ ";
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            cout << a[i] << " ]";
        } else {
            cout << a[i] << ", ";
        }
    }
    cout << endl;
```

```
}

void print_array(int *a, int n) {
    cout << "[ ";
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            cout << a[i] << " ]";
        } else {
            cout << a[i] << ", ";
        }
    }
    cout << endl;
}
void print_array(RatioIndex **a, int n) {
    cout << "[ ";
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            cout << "(" << a[i]->r << ", " << a[i]->index << ")"
                 << " ]";
        } else {
            cout << "(" << a[i]->r << ", " << a[i]->index << ")"
                 << ", ";
        }
    }
    cout << endl;
}

Result *knap_sack_greedy(int W, int n, int weight[], int profit[]) {
    float *io = new float[n];
    RatioIndex **ratio = new RatioIndex *[n];

    // get ratio of profit to weight
    for (int i = 0; i < n; i++) {
        ratio[i] = new RatioIndex();
        ratio[i]->r = (float)profit[i] / weight[i];
        ratio[i]->index = i;
    }

    // sort ratio in descending order
    sort_descending(ratio, n);
```

```
    int try_w = 0;
    float total_profit = 0;

    for (int i = 0; i < n; i++) {
        int index = ratio[i]->index;
        try_w += weight[index];
        io[index] = 1;

        if (try_w > W) {
            try_w -= weight[index];
            int space_needed = W - try_w;
            float portion_taken = (float)space_needed / weight[index];
            io[index] = portion_taken;
            total_profit += portion_taken * profit[index];
        } else {
            total_profit += profit[index];
        }
    }

    Result *res = new Result;
    res->io_array = io;
    res->total_profit = total_profit;
    res->total_weight = try_w;

    return res;
}

int main() {
    int capacity;

    cout << "\nEnter capacity of the bag: ";
    cin >> capacity;

    int items;
    cout << "Enter number of Items: ";
    cin >> items;

    int *weight = new int[items];
    int *profit = new int[items];

    cout << "Enter weight and value of the " << items << " items" << endl;
```

```
    for (int i = 0; i < items; i++) {
        cout << "Item " << i + 1 << " : ";
        cin >> weight[i];
        cin >> profit[i];
    }

    // test input
    // int items = 3;
    // int weight[] = {20, 30, 40};
    // int profit[] = {40, 20, 30};

    cout << "\nWeight: ";

    print_array(weight, items);

    cout << "\nProfit: ";
    print_array(profit, items);

    Result *r = knap_sack_greedy(capacity, items, weight, profit);

    cout << "\nTake the following : ";
    print_array(r->io_array, items);

    cout << "\nTotal weight : " << r->total_weight;
    cout << "\nTotal profit : " << r->total_profit;

    return 0;
}
```

Output:

```
Enter capacity of the bag: 28
Enter number of Items: 7
Enter weight and value of the 7 items
Item 1 : 2 9
Item 2 : 5 5
Item 3 : 6 2
Item 4 : 11 7
Item 5 : 1 6
Item 6 : 9 16
Item 7 : 1 3

Weight: [ 2, 5, 6, 11, 1, 9, 1 ]

Profit: [ 9, 5, 2, 7, 6, 16, 3 ]

Take the following : [ 1, 1, 1, 0.909091, 1, 1, 1 ]

Total weight : 24
Total profit : 47.3636
```

Observation:

The greedy algorithm can easily solve fractional knapsack problem in O(n log n) time

CONCLUSION:

After conducting this experiment, I have learnt how to use greedy approach to solve the fractional knapsack problem.