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

"

Name: Prakhar

GuptaUID:

202130040

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:

Calculate the ratio(value/weight) for each item.

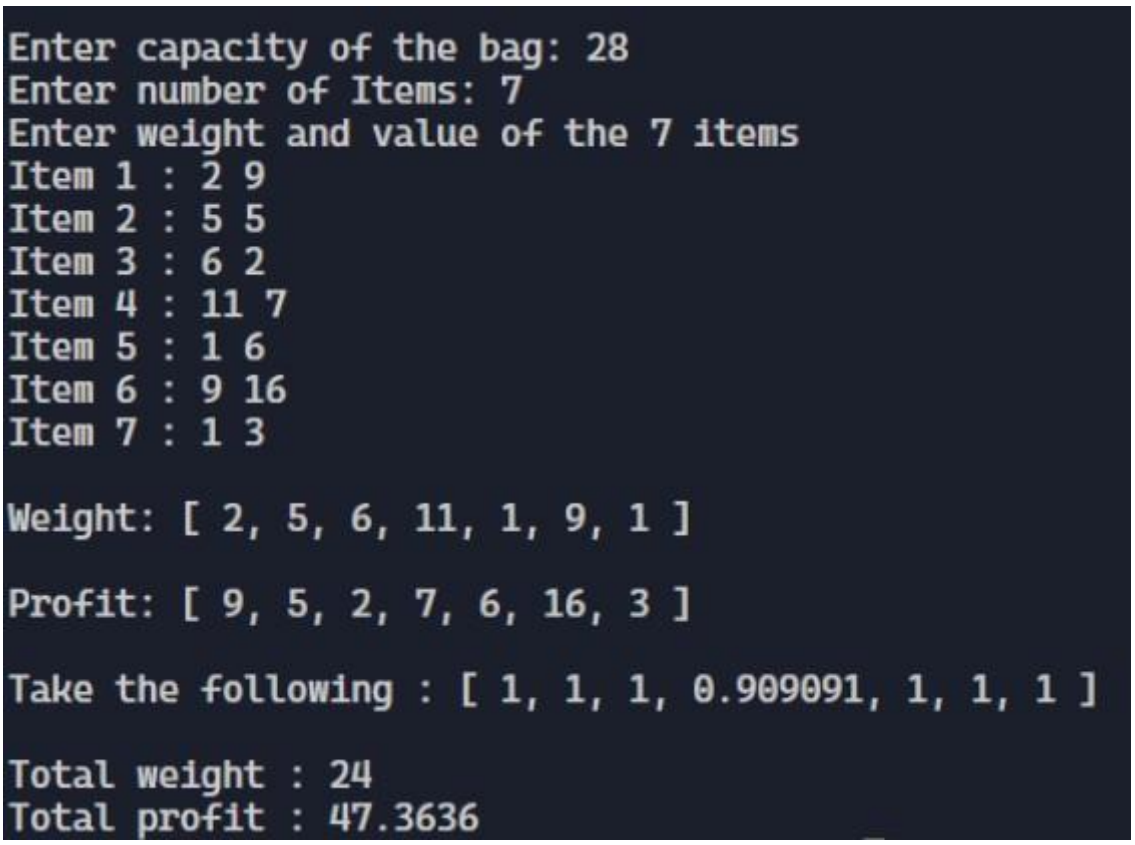
- Sort all the items in decreasing order of the ratio.
- Initialize res =0, curr_cap = given_cap.
- Do the following for every item “i” in the sorted order:
    - If the weight of the current item is less than or equal to the remaining capacity then add the value of that item into the result
    - Else add the current item as much as we can and break out of the loop.
- Return res.

```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;
```

Result:



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.