**Experiment No. – 3.3**

**Aim**:

Write a program to solve the Knapsack Problem Using Greedy Technique.

1. **Problem Description:**

Solve the Knapsack Problem Using Greedy Technique and understand its time complexity.

1. **Algorithm:**

for i in range(1,n):

calculate p/w

Sort objects in descending order of p/w ratio

if M>0 and wi<=M:

M = M-wi

p = p + pi

else:

p = p + pi(M/wi)

1. **Complexity Analysis**

The time complexity for the Knapsack Problem Using Greedy Technique is O(NlogN), where N is the number of items in the knapsack. This is because the algorithm first sorts the items in descending order of their value-to-weight ratio, which takes O(NlogN) time. Then, the algorithm iterates through the items in sorted order, adding each item to the knapsack if there is enough space. This takes O(N) time.

Therefore, the overall time complexity is O(NlogN) + O(N) = O(NlogN).

1. **Pseudo Code**

function knapsack(items, capacity):

sort items by value-to-weight ratio in descending order

current\_weight = 0

total\_value = 0

for item in items:

if current\_weight + item.weight <= capacity:

current\_weight += item.weight

total\_value += item.value

else:

fraction = (capacity - current\_weight) / item.weight

total\_value += fraction \* item.value

current\_weight += fraction \* item.weight

break

return total\_value

1. **Source Code (C/C++):**

#include <bits/stdc++.h>

using namespace std;

// Structure to represent an item

struct Item

{

    int value, weight;

};

// Function to compare two items

bool compare(Item a, Item b)

{

    double r1 = (double)a.value / (double)a.weight;

    double r2 = (double)b.value / (double)b.weight;

    return r1 > r2;

}

// Function to solve the knapsack problem

double fractionalKnapsack(Item items[], int n, int W)

{

    // Sort the items by value/weight ratio

    sort(items, items + n, compare);

    // Initialize the total value

    double totalValue = 0.0;

    // Iterate over the items

    for (int i = 0; i < n; i++)

    {

        // If the weight of the current item is less than the knapsack capacity

        if (items[i].weight <= W)

        {

            // Add the whole item to the knapsack

            totalValue += items[i].value;

            W -= items[i].weight;

        }

        else

        {

            // Add a fraction of the item to the knapsack

            double fraction = (double)W / (double)items[i].weight;

            totalValue += fraction \* items[i].value;

            W = 0;

            break;

        }

    }

    // Return the total value

    return totalValue;

}

// Main function

int main()

{

    // Get the number of items and the knapsack capacity

    int n, W;

    cin >> n >> W;

    // Create an array of items

    Item items[n];

    for (int i = 0; i < n; i++)

    {

        cin >> items[i].value >> items[i].weight;

    }

    // Solve the knapsack problem

    double totalValue = fractionalKnapsack(items, n, W);

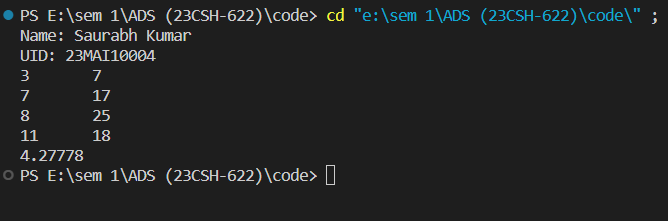
    // Print the total value

    cout << totalValue << endl;

    return 0;

}

1. **Screenshot of Outputs:**

****

1. **Learning Outcomes**
2. Learnt the implemented of Knapsack Problem Using Greedy Technique.
3. Learned about the complexity of Knapsack Problem.