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

Experiment	2
Aim	To understand and implement greedy Approach
Objective	1) Learn GREEDY Approach 2) Implement Greedy approach problem 3) Solve Greedy approach problem
Name	Durgesh Dilip Mandge
UCID	2023510032
Class	FYMCA
Batch	B
Date of Submission	12-2-2024

Algorithm and Explanation of the technique used	Theory will consist of Pseudocode/Algorithm of the approach along with time complexity (Solve the given problem statement in book and pen, the answer which you have got through coding should match with the answer in the notebook).
	
	Fractional Knapsack (Array W, Array V, int M)
	for i <- 1 to size (V)
	    calculate cost[i] <- V[i] / W[i]
	Sort-Descending (cost)
	i ← 1
	while (i <= size(V))
	    if W[i] <= M
	        M ← M – W[i]
	        total ← total + V[i];
	    if W[i] > M
	        i ← i+1

Roll no 35 Values : [35, 71, 107, 143, 179, 215, 251,

Weights = [6, 7, 8, 9, 10, 5, 6,

capacity : 25

	V	W	V/W	In desc. V	W
x_1	35	6	5.83	(x_6) 215	1
x_2	71	7	10.14	(x_7) 251	2
x_3	107	8	13.3	(x_8) 287	3
x_4	143	9	15.8	(x_9) 323	4
x_5	179	10	17.9	(x_{10}) 359	5
x_6	215	1	215	(x_5) 179	10
x_7	251	2	125.5	(x_4) 143	9
x_8	287	3	95.6	(x_3) 107	8
x_9	323	4	80.75	(x_2) 71	7
x_{10}	359	5	71.8	(x_1) 35	6

capacity = 25

25

10	25
5	15
4	10
3	6
2	3
1	1

$= 0x_1 + 0x_2 + 0x_3 + 0x_4 + 1x_6 + 1x_7 + 1x_8$

$= 215 + 251 + 287 + 323 + 35$

$= 1614$

Max profit : 1614

Program(Code)

```
import java.util.*;

public class FractionalKnapsack {


  static class Item {
    int weight, value;
```

```java
    double valuePerUnitWeight;

    Item(int weight, int value) {
      this.weight = weight;
      this.value = value;

      valuePerUnitWeight = (double) (value) / (weight);
    }
  }

  static double getMax(int weight[], int value[], int capacity) {
    int n = weight.length;

    List<Item> list = new ArrayList<>();

    for (int i = 0; i < n; i++) {
      list.add(new Item(weight[i], value[i]));
    }

    Collections.sort(
      list,
      new Comparator<Item>() {

        public int compare(Item i1, Item i2) {
          if (i1.valuePerUnitWeight > i2.valuePerUnitWeight) return -1;
          return 1;
        }
      }
    );

    double ans = 0;

    for (int i = 0; i < n; i++) {
      int wt = list.get(i).weight;
      int val = list.get(i).value;
      double valuePerUnitWeight = list.get(i).valuePerUnitWeight;
      // If we can take the whole item.
      if (capacity >= wt) {
        // Adding value of current item.
        ans += val;

        // Reducing capacity by wt.
        capacity -= wt;
      }
      // Otherwise we can take a fraction.
      else {
        // Adding the value of the part that we can take.
        ans += valuePerUnitWeight * capacity;
```

<table>
<tr><td></td><td>

```java
        // Now we are left with no space so
        // we will terminate the loop.
        capacity = 0;
        break;
      }
    }
    // Returning the answer
    return ans;
  }

  public static void main(String args[]) {
    // Defining the weights and values
    // of the item.
    int weight[] = { 10, 30, 20, 50 };
    int value[] = { 40, 30, 80, 70 };
    int capacity = 60;

    System.out.println(
      "Maximum value that" +
      " can be obtained is " +
      getMax(weight, value, capacity)
    );
  }
}
```

</td></tr>
<tr><td>Output</td><td>

```
PS C:\Users\smart\Documents\SPIT-lab\Sem 2\DAA\Lab2>  c:; cd 'c:\Users\smart\Documents\SPIT-lab\Sem 2\DAA\Lab2'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\smart\AppData\Roaming\Code\User\workspaceStorage\0918afcfa027d4e04d2daa85268c23e8\redhat.java\jdt_ws\Lab2_c32016bd\bin' 'FractionalKnapsack'
Maximum value that can be obtained is 1805.0
PS C:\Users\smart\Documents\SPIT-lab\Sem 2\DAA\Lab2>
```

</td></tr>
<tr><td>Justification of the complexity calculated</td><td>The dominant operation in the algorithm is the sorting step, which typically takes O(nlogn) time using efficient sorting algorithms like Merge Sort or Quick Sort.
Sorting n items takes O(nlogn) time using algorithms like Merge Sort or Quick Sort.
After sorting, the greedy selection process takes O(n) time because it iterates through the sorted list once.
The fractional selection process involves constant-time operations for each item.</td></tr>
<tr><td>Conclusion</td><td>The Fractional Knapsack algorithm provides an efficient solution for optimizing resource allocation in various scenarios. It's key advantage lies in its ability to handle fractional item selections, allowing for more flexible utilization of available resources. This approach finds applications across diverse fields such as logistics, finance, and resource management, where maximizing value while respecting capacity constraints is crucial. Additionally, its time complexity of O(nlogn) ensures scalability and suitability for large-scale optimization problems. Overall, the Fractional Knapsack algorithm stands as a fundamental tool for optimizing resource utilization and decision-making in real-world scenarios.</td></tr>
</table>