Pimpri Chinchwad Education Trust's
Pimpri Chinchwad College of Engineering (PCCoE)
(An Autonomous Institute)
Affiliated to Savitribai Phule Pune University(SPPU)
ISO 21001:2018 Certified by TUV

Assignment -3

· **Name: Khushalsing Pawar**
· **Branch: Information Technology** **Division: B** **PRN No.: 124B2F011**
· **Course Name : Design and Analysis of Algorithms**

Problem Statement: Emergency Relief Supply Distribution

A devastating flood has hit multiple villages in a remote area, and the government, along with NGOs, is organizing an emergency relief operation. A rescue team has a limited-capacity boat that can carry a maximum weight of W kilograms. The boat must transport critical supplies, including food, medicine, and drinking water, from a relief center to the affected villages.

Each type of relief item has:

- A weight (wi) in kilograms.
- Utility value (vi) indicating its importance (e.g., medicine has higher value than food).
- Some items can be divided into smaller portions (e.g., food and water), while others must be taken as a whole (e.g., medical kits).

Goals:

1. Implement the Fractional Knapsack algorithm to maximize the total utility value of the supplies transported.
2. Prioritize high-value items while considering weight constraints.
3. Allow partial selection of divisible items (e.g., carrying a fraction of food packets).
4. Ensure that the boat carries the most critical supplies given its weight limit W.

Course Objectives:

1. To know the basics of computational complexity of various algorithms.
2. To select appropriate algorithm design strategies to solve real-world problems.

Course Outcomes: After learning the course, students will be able to:

1. Analyze the asymptotic performance of algorithms.
2. Solve computational problems by applying suitable paradigms such as Divide and Conquer or Greedy methodologies.

Theory:

The Fractional Knapsack Problem is a classic optimization problem: given a set of items—each with weight w_i and value v_i—and a maximum capacity W, the objective is to maximize total value by selecting items (or fractions of them) up to capacity.

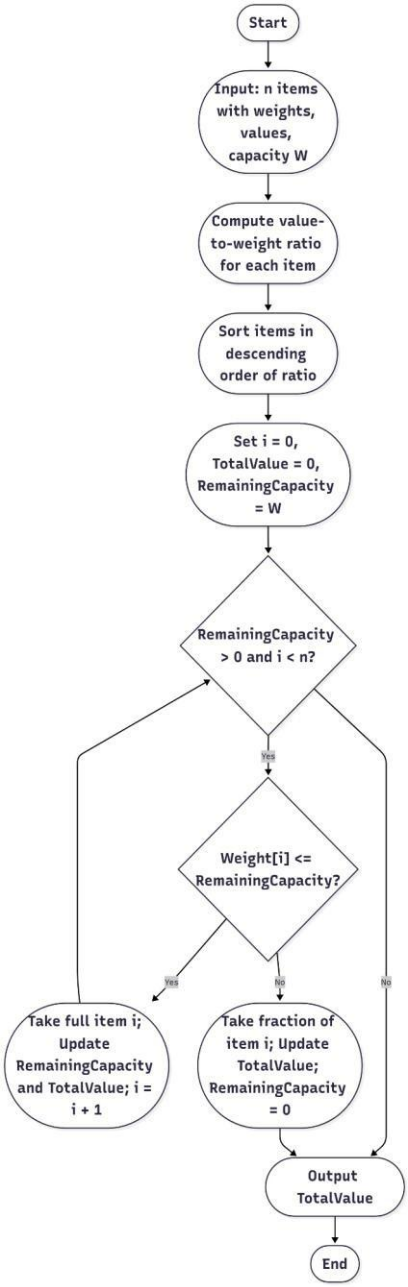
Unlike the 0/1 Knapsack, the fractional version allows partial selection, making it solvable optimally in polynomial time.

Algorithm (Greedy Strategy):

1. Compute the value-to-weight ratio $\mathbf{v_i / w_i}$ for each item.
2. Sort items in **descending order** of this ratio.
3. Fill the knapsack:
    - Take items fully if they fit.
    - If capacity runs out, take the exact fraction needed of the current item (only if divisible).
    - Skip indivisible items that do not fit.

Working of Greedy:

- **Greedy-choice property:** Choosing the highest ratio item at each step is always part of an optimal solution.
- **Optimal substructure:** Once part of the knapsack is filled, the remaining capacity forms a smaller instance of the same problem.



Time Complexity:

- Ratio computation: **O(n)**
- Sorting: **O(n log n)**
- Selection: **O(n)**
- **Total: O(n log n)**

Implementation (Pseudocode)

```
Function MaxUtilFractional(items, W):
    Input:
        items = list of tuples (weight, value, isDivisible)
        W = maximum capacity
    Output:
        maximum total utility value

    // Step 1: Compute value-to-weight ratio for each item
    For each item in items:
        item.ratio ← item.value / item.weight

    // Step 2: Sort items by descending ratio
    Sort items in descending order of item.ratio

    totalValue ← 0
    remainingCapacity ← W

    // Step 3: Pick items greedily
    For each item in sorted items:
        If remainingCapacity = 0:
            Break

        If item.weight ≤ remainingCapacity:
```

totalValue ← totalValue + item.value

remainingCapacity ← remainingCapacity - item.weight

Else if item.isDivisible:

fraction ← remainingCapacity / item.weight

totalValue ← totalValue + fraction * item.value

remainingCapacity ← 0

// else: item is indivisible and too big → skip

Return totalValue

End Function

Code-

```
import java.util.*;
public class knapsack {
    public static void main(String[] args) {
        int val[] = {500, 300, 400, 200, 250};
        int wei[] = {10, 20, 30, 15, 5};

        int w=50;

        double[][] ratio=new double[val.length][2];

        for(int i=0;i<val.length;i++){
            ratio[i][0]=i;
            ratio[i][1]=val[i]/(double)wei[i];
        }

        Arrays.sort(ratio,(a,b)->Double.compare(b[1],a[1]));
        int finalValue=0;
        System.out.println("Selected Supplyes!");
        for(double[] r:ratio){

            int idx=(int)r[0];
            if(w == 0){
                break;
            }

            if(w>wei[idx]){
```

```
            finalValue+=val[idx];
            w-=wei[idx];
            System.out.println("-"+ val[idx] +"(full)");
        }else{
            double fraction=(double)w/wei[idx];
            finalValue += val[idx] * fraction;
            System.out.println("- " + val[idx] + " (" + (fraction*100) + "%)");
            w = 0;
        }
    }
    System.out.println("Maximum utility value: " + finalValue);

  }
}
```

Output:

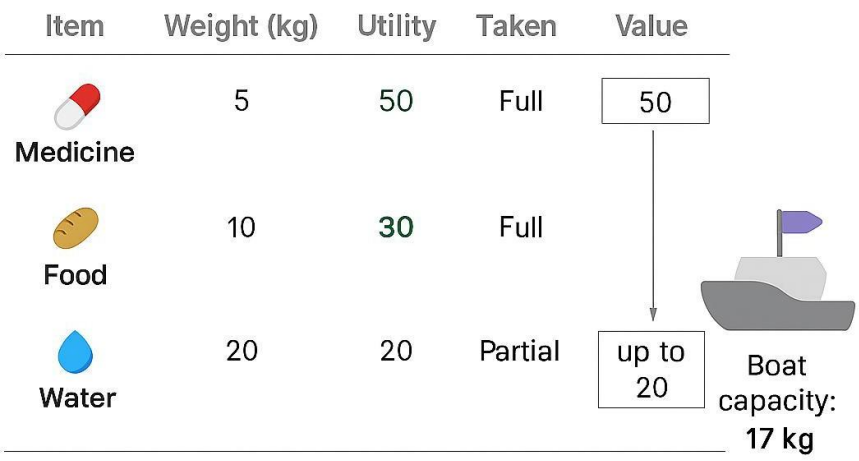
```
PS C:\Users\khush>  & 'C:\Program Files\Java\jdk-23\bin\java.exe'
p\vscodesws_b666b\jdt_ws\jdt.ls-java-project\bin' 'knapsack'
Selected Supplyes!
-500(full)
-250(full)
-300(full)
- 400 (50.0%)
Maximum utility value: 1250
PS C:\Users\khush>
```

Sample Example:

Item	Weight (kg)	Utility	Divisible	Taken	Value Obtained
Medicine	5	50	No	Full	50
Food	10	30	Yes	Full	30
Water	20	20	Yes	Partial (if needed)	up to 20

Maximum Utility = 100 (if all fit in given W).

Boat capacity: 17 kg



Step 1: Calculate utility per kg

- Medicine: 50 / 5 = 10
- Food: 30 / 10 = 3
- Water: 20 / 20 = 1

Preference order: Medicine → Food → Water

Step 2: Fill the boat

1. **Take Medicine fully**:
    - Weight used = 5 kg
    - Utility = 50
    - Remaining capacity = 17 - 5 = 12 kg
2. **Take Food fully** (weight 10 kg, divisible):
    - Fits completely
    - Utility = 30
    - Remaining capacity = 12 - 10 = 2 kg
3. **Take Water partially** (weight 20 kg, divisible):
    - Only 2 kg fits
    - Utility = 20 × (2/20) = 2

Step 3: Total Utility

- Medicine: 50
- Food: 30
- Water (partial): 2

Total Utility = 82

Conclusion:

The **Fractional Knapsack Algorithm** maximizes utility by prioritizing items with the **highest value-to-weight ratio**. In emergency relief logistics, this ensures that life-saving items like **medicine kits** are transported first, followed by food and water as space allows.

The greedy algorithm is both **efficient (O(n log n))** and **optimal** for fractional cases, making it highly effective in disaster management where quick, resource-optimized decisions are critical.