KATHMANDU UNIVERSITY

Dhulikhel, Kavre

COMP 314
LAB REPORT - 3
Knapsack Problem

Submitted By:

Anupama Neupane
CS III/II
Roll No. 36

Submitted to:

Dr. Rajani Chulyadyo
Department of Computer
Science and Engineering

Solve the Knapsack problem using the following strategies:

1. Brute-force method - Fractional Method

   Pseudocode

```
Function KnapsackBruteforce(weights, values, capacity):
    n ← Length of weights (number of items)
    max_value ← 0
    best_combination ← [0] * n
    For each combination in all binary combinations of length n:
        total_weight ← 0
        total_value ← 0
        For i from 0 to n - 1:
            total_weight ← total_weight + (weights[i] * combination[i])
            total_value ← total_value + (values[i] * combination[i])
        If total_weight ≤ capacity AND total_value > max_value:
            max_value ← total_value
            best_combination ← combination

    Return max_value, best_combination
```

2. Brute-force method - Fractional Method

Pseudocode

```
FUNCTION FractionalKnapsackBruteForce(weights, values, capacity):
    n ← LENGTH(weights)  // Number of items
    max_value ← 0        // Maximum value found so far
    best_fractions ← [0] * n  // Initialize fractions of items to take

    FUNCTION CalculateTotal(fractions):
        total_weight ← 0
        total_value ← 0
        FOR i FROM 0 TO n - 1:
            total_weight ← total_weight + (weights[i] * fractions[i])
            total_value ← total_value + (values[i] * fractions[i])
        return (total_weight, total_value)

    FUNCTION GenerateCombinations(current_fractions, index):
        IF index = n:  // Base case: all items processed
            (total_weight, total_value) ← CalculateTotal(current_fractions)
            IF total_weight ≤ capacity AND total_value > max_value:
                max_value ← total_value
                best_fractions ← COPY(current_fractions)
            return

        FOR frac FROM 0 TO 10:
            current_fractions[index] ← frac / 10
            GenerateCombinations(current_fractions, index + 1)
    GenerateCombinations([0] * n, 0)

   return (max_value, best_fractions)
```

3. Greedy method (Fractional Knapsack)

Pseudocode

```
FUNCTION FractionalKnapsack(capacity, items):
    SORT items BY ratio IN descending order
    total_value ← 0.0
    total_weight ← 0
    FOR each item IN items:
        IF total_weight + item.weight ≤ capacity:
            total_value ← total_value + item.value
            total_weight ← total_weight + item.weight
        ELSE:
            remaining_capacity ← capacity - total_weight
            total_value ← total_value + item.value * (remaining_capacity / item.weight)
            BREAK

    return total_value
```

4. Dynamic programming (0/1 Knapsack)

Pseudocode

```
FUNCTION Knapsack01(weights, values, capacity):
  n ← Length of weights (number of items)

  Initialize a 2D DP table with (n + 1) rows and (capacity + 1) columns
  dp ← 2D array of size (n + 1) x (capacity + 1) initialized with 0

  // Fill the DP table
  FOR i FROM 1 TO n:
    FOR w FROM 1 TO capacity:
      IF weights[i - 1] ≤ w:
        // Include the current item or exclude it, take the maximum
        dp[i][w] ← MAX(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])
      ELSE:
        // Exclude the current item
        dp[i][w] ← dp[i - 1][w]

  //Return the maximum value that can be achieved
  return dp[n][capacity]
```