Report on Fractional Knapsack Problem Visualization

Course : Algorithm Design & Analysis

Couse code: CSE-232
Date: 13-04-25

submitted by

Name:Jannatul Fardous, ID: 232-115-040
Name: Rita Begum, ID: 232-115-010
Batch:59th, Section:A

submitted to:

Nasif Istiak Remon
Senior Lecturer
Metropolitan University , Sylhet

1. Introduction

This report describes an interactive visualizer for the **Fractional Knapsack Problem**, implemented in C++. The program demonstrates the **Greedy**

Algorithm approach by selecting items based on their value-to-weight ratio and visualizes the process step by step in the terminal. This tool is designed to help learners understand how the algorithm prioritizes items and makes decisions during the knapsack filling process.

2. Problem Description

The **Fractional Knapsack Problem** is a classical optimization problem in which the goal is to maximize the total value of items placed in a knapsack without exceeding its weight capacity. Unlike the 0/1 Knapsack problem, where items must be fully included or excluded, the fractional version allows breaking items into smaller parts. This makes it solvable using a greedy strategy.

For example, given a set of items with individual weights and values, the algorithm selects items (fully or fractionally) to achieve the highest possible value while staying within the knapsack's weight limit.

3. Solution Approach

The program uses a **Greedy Algorithm** based on the following approach:

- **Item Sorting:**
  Items are sorted in descending order of their `value-to-weight ratio` to prioritize the most valuable items per unit weight.

- **Selection Process:**
  Iterating through the sorted items:

    - If the item fits entirely, it is added to the knapsack.

    - If the item doesn't fit, the program takes the largest possible fraction that fits in the remaining space.

- **Visualization:**
  After each item (or fraction) is selected, the program:
    - Clears the screen.
    - Displays the current total weight and value.
    - Prints a status message explaining the decision.
    - Pauses briefly using `std::this_thread::sleep_for` to allow real-time observation of the algorithm's progress.

4. Code Structure and Design

4.1 Input Handling

- The user is prompted to enter:
    - The number of items.
    - The maximum weight capacity of the knapsack.
    - Each item’s `value` and `weight`.

4.2 Item Representation and Sorting

- Items are stored in a `struct` containing:

    - `value`, `weight`, `index`, and `ratio` (value-to-weight).

- The `ratio` is computed for each item.

- Items are sorted in descending order based on their ratio using `std::sort()` and a custom comparator.

4.3 Display and Visualization

- **Clearing the Screen:**
  The program clears the terminal using `system("cls")` (Windows) or `system("clear")` (Unix-like systems).

- **Displaying Items:**
  After each selection, the program prints:

    - Total weight used.

    - Total value gained.

    - The list of items with their value, weight, and ratio.

- **Decision Messages:**

    - When a full item is selected, the program prints a message indicating full inclusion.

    - When a fractional part is selected, it displays the percentage of the item taken.

4.4 Greedy Selection Loop

- Iterates through the sorted list of items.
- Decides for each item whether to:
    - Take the full item (if the remaining capacity allows).
    - Take a fraction (if space is limited).
- Updates the total weight and total value accordingly.

5. Time Complexity

- **Sorting:** `O(n log n)`, where `n` is the number of items.
- **Selection Loop:** `O(n)`, for iterating through the items.
- **Overall Time Complexity:** `O(n log n)` (dominated by sorting).
- **Space Complexity:** `O(n)`, for storing item information.

6. Additional Considerations

- **Platform Compatibility:**
  The program uses preprocessor directives to ensure the screen-clearing feature is compatible across both Windows and Unix-like systems.
- **Educational Value:**
  The step-by-step visualization and real-time decision messages help users understand the logic and efficiency of the Greedy approach in solving the Fractional Knapsack problem.

- **Potential Enhancements:**
    - Allowing the user to adjust the delay duration.
    - Adding visual indicators (such as color coding) for full vs. fractional item inclusion.
    - Extending the visualizer to other greedy-based problems like Job Sequencing or Huffman Encoding.

7. Conclusion

This interactive C++ program offers both a functional and educational demonstration of the **Fractional Knapsack Problem**. Through clear visualization and real-time feedback, the program effectively explains the logic behind greedy selection and item prioritization. It is a useful tool for students and beginners aiming to deepen their understanding of greedy algorithms and their practical applications.