LAB REPORT

Assignment 4

NAME : P.SATHWIKA

REGISTRATION NUMBER : 21BCE8118

SUBJECT : Design and Analysis of Algorithms.

LAB SLOT : L53+L5

Implement fractional knapsack problem using greedy strategy.

Brute Force Approach

The brute force approach to solving the Knapsack Problem involves generating all possible combinations of items and then selecting the combination that maximizes the value while not exceeding the weight capacity of the knapsack.

Pseudocode:

```
Step1: Check if the current weight exceeds the capacity
if curr_weight > capacity:
return
Step 2: Check if all items are considered
if index == n:
Step 3: Update max_value if the current combination has a higher value
max_value = max(max_value, curr_val)
return
Step 4: Include the current item and move to the next item
generate_combinations(curr_val + values[index], curr_weight + weights[index], index + 1)
Step 5: Exclude the current item and move to the next item
generate_combinations(curr_val, curr_weight, index + 1)
Step 6: Call the recursive function with initial values
generate_combinations(0, 0, 0)
```

Algorithm:

Generate all combinations: Enumerate all possible subsets of items. This can be done using recursive backtracking or by using binary representations of numbers (e.g., bit manipulation).

Evaluate each combination: For each subset of items generated in step 1, calculate the total value and total weight of the items in the subset.

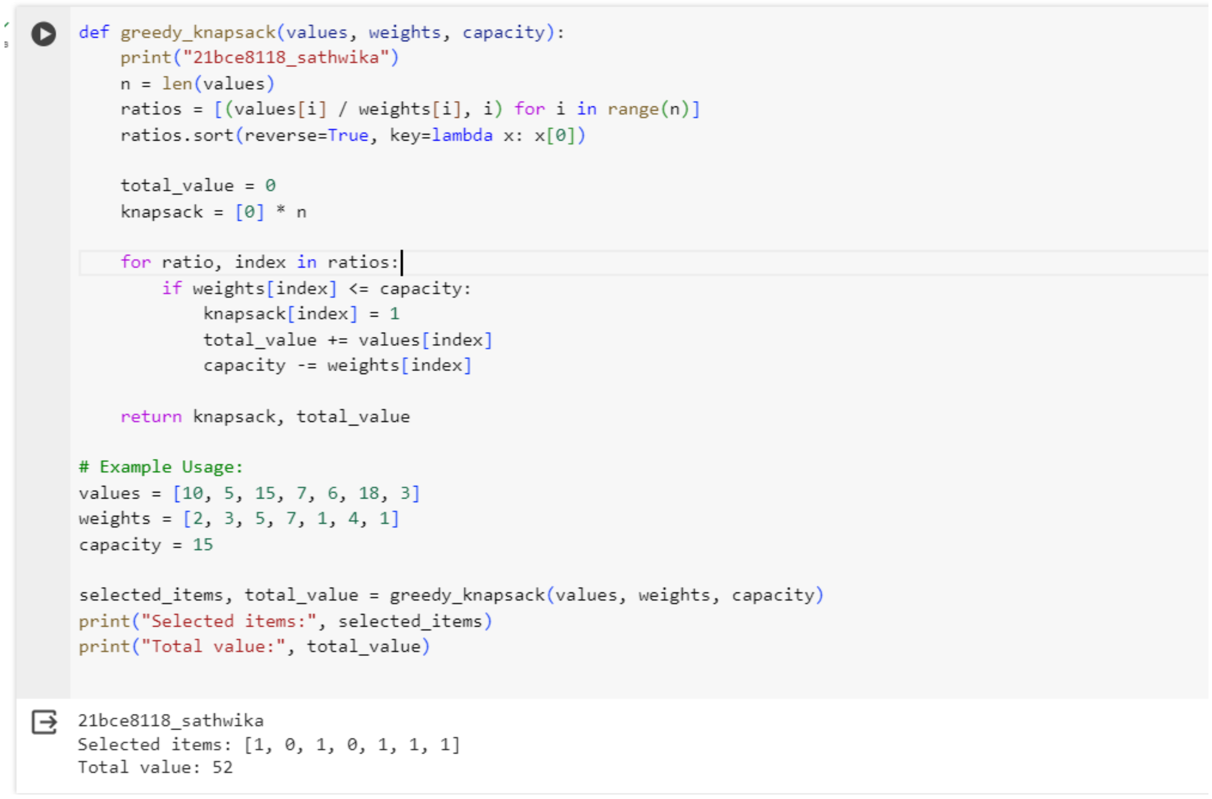
Check feasibility: Ensure that the total weight of each subset does not exceed the capacity of the knapsack. If the weight exceeds the capacity, discard that subset.

Select the best combination: Among the feasible combinations, select the one with the maximum total value.

Return the solution: Once all combinations have been evaluated, return the combination with the maximum total value as the solution to the Knapsack Problem.

Code:

```python
def greedy_knapsack(values, weights, capacity):
    print("21bce8118_sathwika")
    n = len(values)
    ratios = [(values[i] / weights[i], i) for i in range(n)]
    ratios.sort(reverse=True, key=lambda x: x[0])

    total_value = 0
    knapsack = [0] * n

    for ratio, index in ratios:
        if weights[index] <= capacity:
            knapsack[index] = 1
            total_value += values[index]
            capacity -= weights[index]

    return knapsack, total_value

# Example Usage:
values = [10, 5, 15, 7, 6, 18, 3]
weights = [2, 3, 5, 7, 1, 4, 1]
capacity = 15

selected_items, total_value = greedy_knapsack(values, weights,
capacity)
print("Selected items:", selected_items)
print("Total value:", total_value)
```



```python
def greedy_knapsack(values, weights, capacity):
    print("21bce8118_sathwika")
    n = len(values)
    ratios = [(values[i] / weights[i], i) for i in range(n)]
    ratios.sort(reverse=True, key=lambda x: x[0])

    total_value = 0
    knapsack = [0] * n

    for ratio, index in ratios:
        if weights[index] <= capacity:
            knapsack[index] = 1
            total_value += values[index]
            capacity -= weights[index]

    return knapsack, total_value

# Example Usage:
values = [10, 5, 15, 7, 6, 18, 3]
weights = [2, 3, 5, 7, 1, 4, 1]
capacity = 15

selected_items, total_value = greedy_knapsack(values, weights, capacity)
print("Selected items:", selected_items)
print("Total value:", total_value)
```

```
21bce8118_sathwika
Selected items: [1, 0, 1, 0, 1, 1, 1]
Total value: 52
```