Exp-2.14

Title:

0-1 Knapsack Problem Using Exhaustive Search

Aim:

To find the subset of items with maximum total value without exceeding the knapsack capacity using exhaustive search.

Procedure:

1. Define total_value(items, values) to calculate the total value of selected items.
2. Define is_feasible(items, weights, capacity) to check if the sum of selected items' weights is within capacity.
3. Use exhaustive search by generating all subsets of items (using bit masking or combinations).
4. For each subset, check feasibility and calculate total value.
5. Track the feasible subset with maximum total value.
6. Return and print the optimal selection and total value.

Algorithm:

1. For all subsets of items:
    - Check if subset is feasible (weight $\leq$ capacity).
    - Calculate total value if feasible.
    - Update optimal subset if value is better.
2. Return optimal subset and its value.

Input:

weights1 = [2, 3, 1]

values1 = [4, 5, 3]

capacity1 = 4

weights2 = [1, 2, 3, 4]

values2 = [2, 4, 6, 3]

capacity2 = 6

Output:

Test Case 1:

Optimal Selection: [0, 2]

Total Value: 7

Test Case 2:

Optimal Selection: [0, 1, 2]

Total Value: 10

Program:

```
from itertools import combinations
def total_value(items, values):
    return sum(values[i] for i in items)
def is_feasible(items, weights, capacity):
    return sum(weights[i] for i in items) <= capacity
def knapsack_exhaustive(weights, values, capacity):
    n = len(weights)
    max_value = 0
    best_selection = []
```

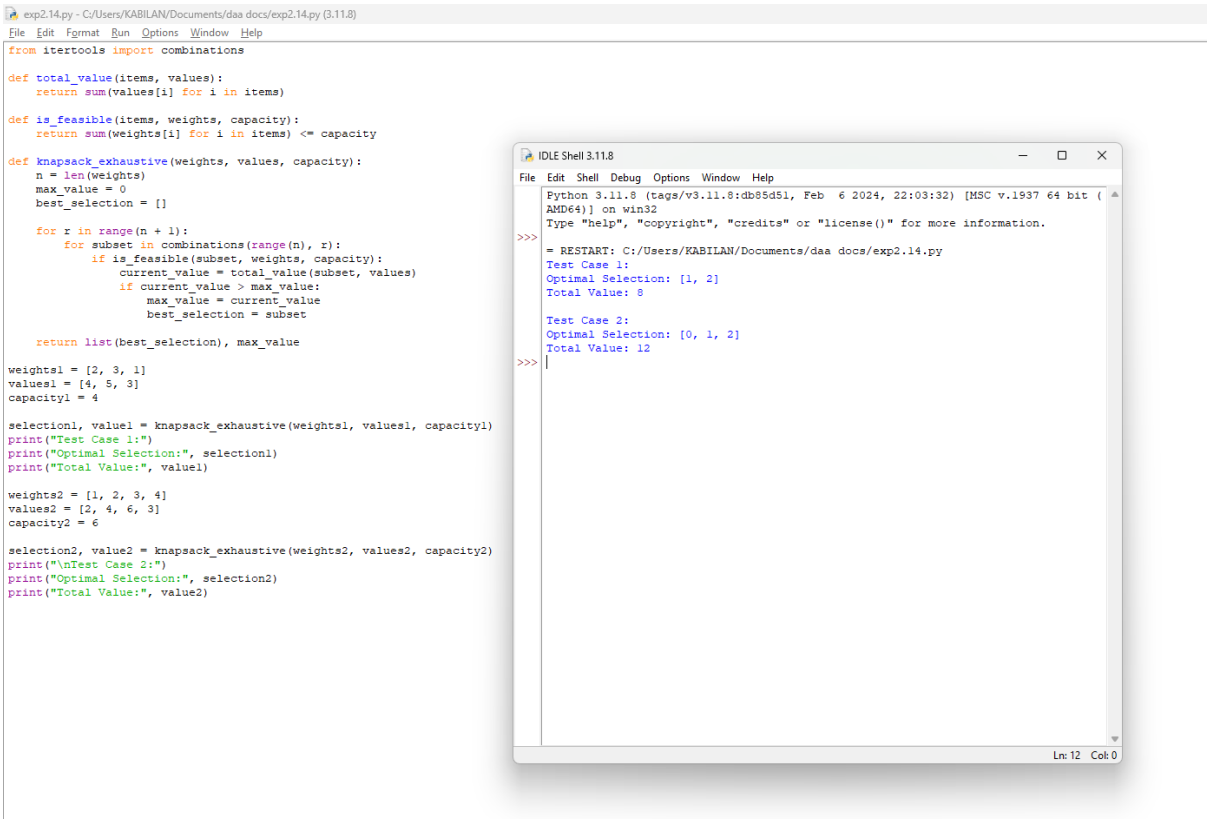
```
    for r in range(n + 1):
        for subset in combinations(range(n), r):
            if is_feasible(subset, weights, capacity):
                current_value = total_value(subset, values)
                if current_value > max_value:
                    max_value = current_value
                    best_selection = subset
    return list(best_selection), max_value
# Test Case 1
weights1 = [2, 3, 1]
values1 = [4, 5, 3]
capacity1 = 4
selection1, value1 = knapsack_exhaustive(weights1, values1, capacity1)
print("Test Case 1:")
print("Optimal Selection:", selection1)
print("Total Value:", value1)

# Test Case 2
weights2 = [1, 2, 3, 4]
values2 = [2, 4, 6, 3]
capacity2 = 6

selection2, value2 = knapsack_exhaustive(weights2, values2, capacity2)
print("\nTest Case 2:")
print("Optimal Selection:", selection2)
print("Total Value:", value2)
```

Performance Analysis:

Time Complexity: O(2^n)

Space Complexity: O(n)

Program Output:



```
exp2.14.py - C:/Users/KABILAN/Documents/daa docs/exp2.14.py (3.11.8)
File  Edit  Format  Run  Options  Window  Help
from itertools import combinations

def total_value(items, values):
    return sum(values[i] for i in items)

def is_feasible(items, weights, capacity):
    return sum(weights[i] for i in items) <= capacity

def knapsack_exhaustive(weights, values, capacity):
    n = len(weights)
    max_value = 0
    best_selection = []

    for r in range(n + 1):
        for subset in combinations(range(n), r):
            if is_feasible(subset, weights, capacity):
                current_value = total_value(subset, values)
                if current_value > max_value:
                    max_value = current_value
                    best_selection = subset

    return list(best_selection), max_value

weights1 = [2, 3, 1]
values1 = [4, 5, 3]
capacity1 = 4

selection1, value1 = knapsack_exhaustive(weights1, values1, capacity1)
print("Test Case 1:")
print("Optimal Selection:", selection1)
print("Total Value:", value1)

weights2 = [1, 2, 3, 4]
values2 = [2, 4, 6, 3]
capacity2 = 6

selection2, value2 = knapsack_exhaustive(weights2, values2, capacity2)
print("\nTest Case 2:")
print("Optimal Selection:", selection2)
print("Total Value:", value2)
```

```
IDLE Shell 3.11.8
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.8 (tags/v3.11.8:db85d51, Feb  6 2024, 22:03:32) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/KABILAN/Documents/daa docs/exp2.14.py
Test Case 1:
Optimal Selection: [1, 2]
Total Value: 8

Test Case 2:
Optimal Selection: [0, 1, 2]
Total Value: 12
>>>
Ln: 12  Col: 0
```

Result:

The exhaustive search program for 0-1 Knapsack correctly produces optimal selections for the given test cases.