DAA_File

https://github.com/dhananjay-deshmukh/SPPU_DAA_Pracs

1.	Write a program to calculate Fibonacci numbers and find its step count.

```
# Recursive Fibonacci with Step Count
def fibonacci_recursive(n, steps):
    steps[0] += 1  # Counting each function call as a step
    if n <= 1:
        return n
    return fibonacci_recursive(n - 1, steps) + fibonacci_recursive(n - 2,
steps)


# Iterative Fibonacci with Step Count
def fibonacci_iterative(n):
    steps = 0
    if n <= 1:
        return n, steps + 1  # Counting initial check as a step

    a, b = 0, 1
    for _ in range(2, n + 1):
        steps += 1  # Each iteration is a step
        a, b = b, a + b
    return b, steps


# Main program
def main():
    n = int(input("Enter the value of n to find the nth Fibonacci number:
"))

    # Recursive method
    steps_recursive = [0]  # Using list for pass-by-reference
    fib_recursive = fibonacci_recursive(n, steps_recursive)
    print(f"Recursive: Fibonacci({n}) = {fib_recursive}, Steps =
{steps_recursive[0]}")

    # Iterative method
```

```
    fib_iterative, steps_iterative = fibonacci_iterative(n)
    print(f"Iterative: Fibonacci({n}) = {fib_iterative}, Steps =
{steps_iterative}")


if __name__ == "__main__":
    main()
```

```
Enter the value of n to find the nth Fibonacci number: 8
Recursive: Fibonacci(8) = 21, Steps = 67
Iterative: Fibonacci(8) = 21, Steps = 7

=== Code Execution Successful ===
```

For Series:

```
def fib_non_recursive(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series

def fib_recursive(n):
    if n <= 1:
        return n
    return fib_recursive(n - 1) + fib_recursive(n - 2)

def fib_series_recursive(n):
    return [fib_recursive(i) for i in range(n)]

n = int(input("Enter total numbers to print in Fibonacci series:\t"))

print("Fibonacci Series (non-recursive):", fib_non_recursive(n))
print("Fibonacci Series (recursive):    ", fib_series_recursive(n))
```

```
Enter total numbers to print in Fibonacci series:   8
Fibonacci Series (non-recursive): [0, 1, 1, 2, 3, 5, 8, 13]
Fibonacci Series (recursive):     [0, 1, 1, 2, 3, 5, 8, 13]

=== Code Execution Successful ===
```

3.	Write a program to solve a fractional Knapsack problem using a greedy method.

```python
class Item:
    def __init__(self, weight, value):
        self.weight = weight
        self.value = value

def fractional_knapsack(items, capacity):
    # Calculate the value-to-weight ratio for each item
    value_weight_ratio = [(item.value / item.weight, item) for item in
items]

    # Sort items in decreasing order of value-to-weight ratio
    value_weight_ratio.sort(reverse=True)

    total_value = 0.0
    knapsack = []

    for ratio, item in value_weight_ratio:
        if capacity == 0:
            break

        weight = min(item.weight, capacity)
        total_value += weight * ratio
        capacity -= weight

        knapsack.append((item, weight))

    return total_value, knapsack
```

```python
# Example usage
if __name__ == "__main__":
    items = [Item(10, 60), Item(20, 100), Item(30, 120)]
    max_capacity = 50

    max_value, selected_items = fractional_knapsack(items, max_capacity)

    print("Selected Items:")
    for item, weight in selected_items:
        print(f"Item with weight {item.weight} and value {item.value}
(fraction taken: {weight / item.weight})")

    print(f"Maximum value achievable: {max_value}")
```

4.	Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

```python
def knapsack_01(values, weights, capacity):
    n = len(values)

    # Create a table to store the maximum values for different subproblems
    dp = [[0 for _ in range(capacity + 1)] for _ in range(n + 1)]

    for i in range(n + 1):
        for w in range(capacity + 1):
            if i == 0 or w == 0:
                dp[i][w] = 0
            elif weights[i - 1] <= w:
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i -
1]], dp[i - 1][w])
            else:
                dp[i][w] = dp[i - 1][w]

    # Backtrack to find the items selected
    selected_items = []
    i, w = n, capacity
    while i > 0 and w > 0:
        if dp[i][w] != dp[i - 1][w]:
```

```
            selected_items.append(i - 1)
            w -= weights[i - 1]
        i -= 1

    selected_items.reverse()

    for row in dp:
        print(row)

    return dp[n][capacity], selected_items

# Example usage
if __name__ == "__main__":
    values = [60, 100, 120]
    weights = [10, 20, 30]
    max_capacity = 50

    max_value, selected_items = knapsack_01(values, weights, max_capacity)

    print("Selected Items:")
    for item in selected_items:
        print(f"Item with weight {weights[item]} and value
{values[item]}")

    print(f"Maximum value achievable: {max_value}")
```

6.	Design 8-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final 8-queen's matrix.

```
def is_safe(board, row, col, n):
    # Check if there's a Queen in the same column
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check upper left diagonal
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
```

```
            return False

    # Check upper right diagonal
    for i, j in zip(range(row, -1, -1), range(col, n)):
        if board[i][j] == 1:
            return False

    return True

def solve_n_queens_with_known_first_queen(n, first_queen_row,
first_queen_col):
    board = [[0 for _ in range(n)] for _ in range(n)]

    # Place the first Queen at the specified row and column
    board[first_queen_row][first_queen_col] = 1

    def solve_n_queens_util(row):
        if row == n:
            return True

        for col in range(n):
            if is_safe(board, row, col, n):
                board[row][col] = 1
                if solve_n_queens_util(row + 1):
                    return True
                board[row][col] = 0

        return False

    if not solve_n_queens_util(first_queen_row + 1):
        print("No solution exists.")
    else:
        for row in board:
            print(row)

n = 4
first_queen_row = 0
first_queen_col = 1
solve_n_queens_with_known_first_queen(n, first_queen_row, first_queen_col)
```

Huffman Encoding

```
import heapq
from collections import defaultdict, Counter

class HuffmanNode:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

def build_huffman_tree(freq_dict):
    priority_queue = [HuffmanNode(char, freq) for char, freq in
freq_dict.items()]
    heapq.heapify(priority_queue)

    while len(priority_queue) > 1:
        left = heapq.heappop(priority_queue)
        right = heapq.heappop(priority_queue)
        merged_node = HuffmanNode(None, left.freq + right.freq)
        merged_node.left = left
        merged_node.right = right
        heapq.heappush(priority_queue, merged_node)

    return priority_queue[0]

def build_huffman_codes(root, current_code, huffman_codes):
    if root is not None:
        if root.char is not None:
            huffman_codes[root.char] = current_code
        build_huffman_codes(root.left, current_code + '0', huffman_codes)
        build_huffman_codes(root.right, current_code + '1', huffman_codes)

def huffman_encoding(data):
```

```
    if not data:
        return None, None

    freq_dict = dict(Counter(data))
    root = build_huffman_tree(freq_dict)
    huffman_codes = {}
    build_huffman_codes(root, '', huffman_codes)

    encoded_data = ''.join(huffman_codes[char] for char in data)
    return encoded_data, root

def huffman_decoding(encoded_data, root):
    if not encoded_data:
        return None

    decoded_data = ""
    current_node = root
    for bit in encoded_data:
        if bit == '0':
            current_node = current_node.left
        else:
            current_node = current_node.right

        if current_node.char is not None:
            decoded_data += current_node.char
            current_node = root

    return decoded_data

# Example usage
if __name__ == "__main__":
    data = "this is an example for huffman encoding"

    encoded_data, tree = huffman_encoding(data)
    print(f"Encoded data: {encoded_data}")

    decoded_data = huffman_decoding(encoded_data, tree)
    print(f"Decoded data: {decoded_data}")
```

2.	Implement job sequencing with deadlines using a greedy method.

5.	Write a program to generate binomial coefficients using dynamic programming.