BA53 Parth Khajgiwale

fibonacci.py

```
class Fibonacci:
    def __init__(self):
        self.series = {}

    def iterative(self, n):
        if n < 0:
            return
        self.series[0] = 0
        if n > 0:
            self.series[1] = 1
        for i in range(2, n + 1):
            self.series[i] = self.series[i - 1] + self.series[i - 2]

    def recursive(self, n):
        if n < 0:
            return
        if n == 0:
            self.series[0] = 0
            return 0
        if n == 1:
            self.series[1] = 1
            return 1
        if n not in self.series:
            self.series[n] = self.recursive(n - 1) + self.recursive(n - 2)
            return self.series[n]

    def print_series(self, n):
        for i in range(n + 1):
            print(self.series[i], end=" ")
        print()

    def main(self):
        while True:
        n = int(input("Enter number of terms in Fibonacci sequence: "))
        choice = int(input("1. Recursive, 2. Iterative, 3. Exit: "))

        if choice == 1:
            self.series.clear()
            self.recursive(n)
            self.print_series(n)
        elif choice == 2:
            self.series.clear()
            self.iterative(n)
            self.print_series(n)
        elif choice == 3:
            print("Exiting the program.")
```

```
                break
        else:
                print("Invalid Input")


fibonacci = Fibonacci()
fibonacci.main()
```

Output:

```
Enter number of terms in Fibonacci sequence: 10
1. Recursive, 2. Iterative, 3. Exit: 1
0 1 1 2 3 5 8 13 21 34 55
Enter number of terms in Fibonacci sequence: 9
1. Recursive, 2. Iterative, 3. Exit: 2
0 1 1 2 3 5 8 13 21 34
Enter number of terms in Fibonacci sequence: 7
1. Recursive, 2. Iterative, 3. Exit: 3
Exiting the program.
```

BA53 Parth Khajgiwale

huffman.py

```
import heapq
from collections import defaultdict

class HuffmanCoding:
    def __init__(self):
    self.codes = {}
    self.reverse_codes = {}

    def calculate_frequencies(self, text):
        frequencies = defaultdict(int)
        for char in text:
            frequencies[char] += 1
        return frequencies

    def build_huffman_tree(self, frequencies):
        heap = [[freq, [char, ""]] for char, freq in frequencies.items()]
        heapq.heapify(heap)

        while len(heap) > 1:
            low = heapq.heappop(heap)
            high = heapq.heappop(heap)
            for pair in low[1:]:
                pair[1] = '0' + pair[1]
            for pair in high[1:]:
                pair[1] = '1' + pair[1]
            heapq.heappush(heap, [low[0] + high[0]] + low[1:] + high[1:])

        return sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))

    def huffman_encoding(self, text):
        if not text:
            return "", {}

        frequencies = self.calculate_frequencies(text)
        huffman_codes = self.build_huffman_tree(frequencies)
        self.codes = {char: code for char, code in huffman_codes}

        encoded_text = ''.join(self.codes[char] for char in text)
        return encoded_text

    def huffman_decoding(self, encoded_text):
        reverse_codes = {v: k for k, v in self.codes.items()}
        current_code = ""
        decoded_text = ""

        for bit in encoded_text:
```

```
                current_code += bit
                if current_code in reverse_codes:
                    decoded_text += reverse_codes[current_code]
                    current_code = ""

        return decoded_text

def main():
    text = input("Enter the text to encode: ")
    huffman = HuffmanCoding()

    encoded_text = huffman.huffman_encoding(text)
    print("Encoded text:", encoded_text)
    print("Huffman Codes:", huffman.codes)

    decoded_text = huffman.huffman_decoding(encoded_text)
    print("Decoded text:", decoded_text)

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

Output:

```
Enter the text to encode: huffman encoding
Encoded text: 1010000100100110000111110010011011101001101010110111110111
Huffman Codes: {'f': '100', 'n': '111', 'u': '000', ' ': '0010', 'a': '0011', 'c': '0100', 'd': '0101', 'e':
'0110', 'g': '0111', 'h': '1010', 'i': '1011', 'm': '1100', 'o': '1101'}
Decoded text: huffman encoding
```

BA53 Parth Khajgiwale

Fractional_knapsack.py

```
def fractional_knapsack(value, weight, capacity):
    # Calculate the value-to-weight ratio
    ratio = [v / w for v, w in zip(value, weight)]

        # Create a list of indices sorted by value-to-weight ratio in decreasing order
    index = list(range(len(value)))

    index.sort(key=lambda i: ratio[i], reverse=True)

    max_value = 0

    fractions = [0] * len(value)

    for i in index:

        if weight[i] <= capacity:
            # Take the whole item
            fractions[i] = 1
            max_value += value[i]
            capacity -= weight[i]

        else:
            # Take the fractional part of the item
            fractions[i] = capacity / weight[i]
            max_value += value[i] * fractions[i]
            break

    return max_value, fractions

# Input from the user
n = int(input('Enter number of items: '))

value = list(map(int, input('Enter the values of the {} item(s) in order: '.format(n)).split()))

weight = list(map(int, input('Enter the positive weights of the {} item(s) in order:
'.format(n)).split()))

capacity = int(input('Enter maximum weight: '))
max_value, fractions = fractional_knapsack(value, weight, capacity)

print('The maximum value of items that can be carried:', max_value)
print('The fractions in which the items should be taken:', fractions)
```

BA53 Parth Khajgiwale

Output:

Enter number of items: 3
Enter the values of the 3 item(s) in order: 60 100 120
Enter the positive weights of the 3 item(s) in order: 10 20 30
Enter maximum weight: 50
The maximum value of items that can be carried: 240.0
The fractions in which the items should be taken: [1, 1, 0.6666666666666666]

BA53 Parth Khajgiwale

0_1_dp.py

```
def knapsack_dp(weights, values, W, n):
    dp = [[0 for _ in range(W+1)] for _ in range(n+1)]
    track = [[0 for _ in range(W+1)] for _ in range(n+1)]

    for i in range(1, n+1):
    for w in range(1, W+1):
    if weights[i-1] <= w:
        if dp[i-1][w] < dp[i-1][w - weights[i-1]] + values[i-1]:
        dp[i][w] = dp[i-1][w - weights[i-1]] + values[i-1]
        track[i][w] = 1
        else:
        dp[i][w] = dp[i-1][w]
    else:
        dp[i][w] = dp[i-1][w]

    w = W
    items_included = []
    for i in range(n, 0, -1):
    if track[i][w] == 1:
    items_included.append(i-1)
    w -= weights[i-1]

    return dp[n][W], items_included

def get_input():
    n = int(input("Enter the number of items: "))
    weights = []
    values = []

    for i in range(n):
    weight = int(input(f"Enter the weight of item {i+1}: "))
    value = int(input(f"Enter the value of item {i+1}: "))
    weights.append(weight)
    values.append(value)

    W = int(input("Enter the maximum weight capacity of the knapsack: "))

    return weights, values, W, n

if __name__ == "__main__":
    weights, values, W, n = get_input()

    max_value, items_included = knapsack_dp(weights, values, W, n)

    print(f"\nMaximum value in Knapsack = {max_value}")
    print("Items included (indices):", items_included)
```

```
    print("Items included (weights and values):")
    for i in items_included:
    print(f"Item {i+1}: Weight = {weights[i]}, Value = {values[i]}")
```

Output:

```
Enter the number of items: 5
Enter the weight of item 1: 10
Enter the value of item 1: 60
Enter the weight of item 2: 20
Enter the value of item 2: 100
Enter the weight of item 3: 30
Enter the value of item 3: 120
Enter the weight of item 4: 5
Enter the value of item 4: 50
Enter the weight of item 5: 15
Enter the value of item 5: 90
Enter the maximum weight capacity of the knapsack: 50

Maximum value in Knapsack = 300
Items included (indices): [4, 3, 1, 0]
Items included (weights and values):
Item 5: Weight = 15, Value = 90
Item 4: Weight = 5, Value = 50
Item 2: Weight = 20, Value = 100
Item 1: Weight = 10, Value = 60
```

BA53 Parth Khajgiwale

0_1_branch_bound.py

```
class Item:
    def __init__(self, value, weight):
    self.value = value
    self.weight = weight
    self.ratio = value / weight if weight != 0 else 0  # Avoid division by zero

def bound(i, weight, value, items, W, n):
    if weight >= W:
    return 0
    result = value
    total_weight = weight
    while i < n and total_weight + items[i].weight <= W:
    total_weight += items[i].weight
    result += items[i].value
    i += 1
    if i < n:
    result += (W - total_weight) * items[i].ratio
    return result

def branch_bound(i, weight, value, max_value, current_items, items, W, n, best_items):
    if i >= n:
    if value > max_value:
    max_value = value
    best_items[:] = current_items[:]
    return max_value

    # Option 1: Include the current item if it fits in the knapsack
    if weight + items[i].weight <= W:
    current_items.append(i)
    max_value = branch_bound(i + 1, weight + items[i].weight, value + items[i].value,
max_value, current_items, items, W, n, best_items)
    current_items.pop()

    # Option 2: Exclude the current item if the bound allows it
    if bound(i + 1, weight, value, items, W, n) > max_value:
    max_value = branch_bound(i + 1, weight, value, max_value, current_items, items, W,
n, best_items)

    return max_value

def get_input():
    n = int(input("Enter the number of items: "))
    items = []

    for i in range(n):
    value = int(input(f"Enter the value of item {i+1}: "))
```

```
        weight = int(input(f"Enter the weight of item {i+1}: "))
        items.append(Item(value, weight))

    W = int(input("Enter the maximum weight capacity of the knapsack: "))

    return items, W, n

if __name__ == "__main__":
    items, W, n = get_input()

    max_value = 0
    best_items = []

    # Start Branch and Bound
    max_value = branch_bound(0, 0, 0, max_value, [], items, W, n, best_items)

    print(f"\nMaximum value in Knapsack using Branch and Bound = {max_value}")
    print("Items included (indices):", best_items)
    print("Items included (weights and values):")
    for i in best_items:
    print(f"Item {i+1}: Weight = {items[i].weight}, Value = {items[i].value}")
```

Output:

```
Enter the number of items: 4
Enter the value of item 1: 10
Enter the weight of item 1: 3
Enter the value of item 2: 5
Enter the weight of item 2: 2
Enter the value of item 3: 15
Enter the weight of item 3: 5
Enter the value of item 4: 7
Enter the weight of item 4: 1
Enter the maximum weight capacity of the knapsack: 7

Maximum value in Knapsack using Branch and Bound = 22
Items included (indices): [0, 1, 3]
Items included (weights and values):
Item 1: Weight = 3, Value = 10
Item 2: Weight = 2, Value = 5
Item 4: Weight = 1, Value = 7
```

BA53 Parth Khajgiwale

n_queens.py

```
def is_safe(board, row, col, n):
    for i in range(row):
    if board[i] == col or board[i] - i == col - row or board[i] + i == col + row:
    return False
    return True

def solve_nqueens(board, row, n, solutions):
    if row == n:
    solutions.append([' '.join(['Q' if col == board[i] else '.' for col in range(n)]) for i in
range(n)])
    return True

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

def nqueens(n):
    board = [-1] * n
    solutions = []
    solve_nqueens(board, 0, n, solutions)
    return solutions

def get_input():
    while True:
    try:
    n = int(input("Enter the number of queens (n): "))
    if n > 0:
        return n
    else:
        print("Please enter a positive integer greater than 0.")
    except ValueError:
    print("Invalid input. Please enter a positive integer.")

if __name__ == "__main__":
    n = get_input()
    solutions = nqueens(n)

    if solutions:
    print(f"\nSolution for {n}-Queens:")
    for row in solutions[0]:
    print(row)
    else:
    print(f"No solution found for {n}-Queens.")
```

BA53 Parth Khajgiwale

Output:

```
Enter the number of queens (n): 4

Solution for 4-Queens:
. Q . .
. . . Q
Q . . .
. . Q .


Enter the number of queens (n): 8

Solution for 8-Queens:
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
```

BA53 Parth Khajgiwale

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @title ContractName
 * @dev ContractDescription
 * @custom:dev-run-script scripts/deploy_with_ethers.ts
 */

contract Student_management {
    struct Student {
        int256 stud_id;
        string name;
        string department;
    }
    Student[] public Students;

    function addStudent(
        int256 stud_id,
        string memory name,
        string memory department
    ) public {
        Student memory stud = Student(stud_id, name, department);
        Students.push(stud);
    }

    function getStudent(int256 stud_id)
        public
        view
        returns (string memory, string memory)
    {
        for (uint256 i = 0; i < Students.length; i++) {
            Student memory stud = Students[i];
            if (stud.stud_id == stud_id) {
                return (stud.name, stud.department);
            }
        }
        return ("Not Found", "Not Found");
    }
}
```

BA53 Parth Khajgiwale