Experiment No.4:

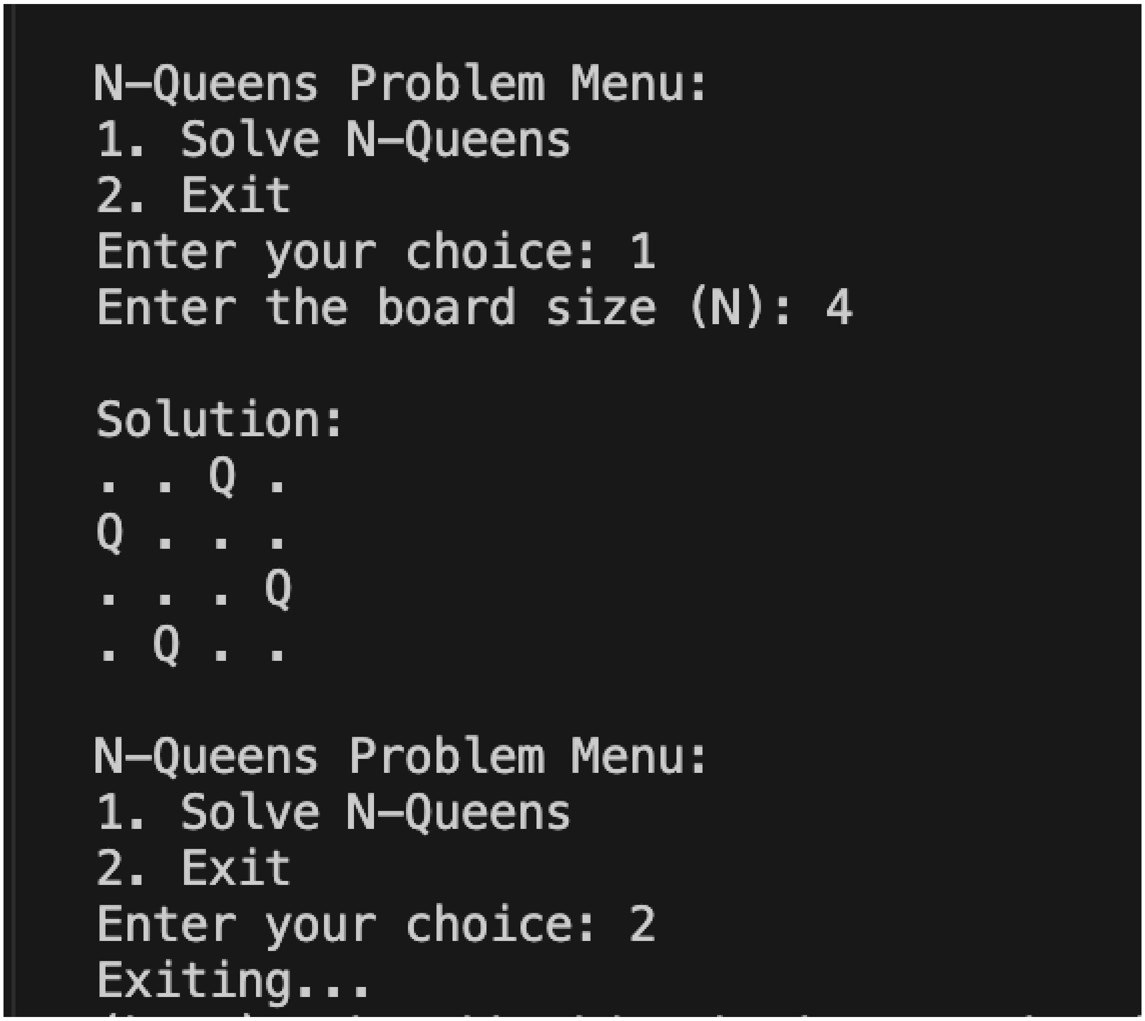
Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem

Code:

□ N-Queens Problem:

```
def is_safe(board, row, col, n):

    for i in range(col):

        if board[row][i] == 1:

            return False


    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

        if board[i][j] == 1:

            return False


    for i, j in zip(range(row, n), range(col, -1, -1)):

        if board[i][j] == 1:

            return False


    return True


def solve_n_queens_util(board, col, n):

    if col >= n:

        return True


    for i in range(n):

        if is_safe(board, i, col, n):
            board[i][col] = 1
            if solve_n_queens_util(board, col + 1, n):
                return True
            board[i][col] = 0

    return False

def solve_n_queens(n):
    board = [[0 for _ in range(n)] for _ in range(n)]

    if not solve_n_queens_util(board, 0, n):
        return None

    solutions = []
    for row in board:
        solutions.append(["Q" if cell == 1 else "." for cell in row])

    return solutions

def print_solution(solution):
    if solution is None:
        print("No solution exists.")
    else:
        for row in solution:
            print(" ".join(row))

def main():
    while True:
        print("\nN-Queens Problem Menu:")
        print("1. Solve N-Queens")
        print("2. Exit")
        choice = input("Enter your choice: ")

        if choice == "1":
            n = int(input("Enter the board size (N): "))
            solution = solve_n_queens(n)
            print("\nSolution:")
            print_solution(solution)
        elif choice == "2":
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please choose a valid option.")

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

Output:

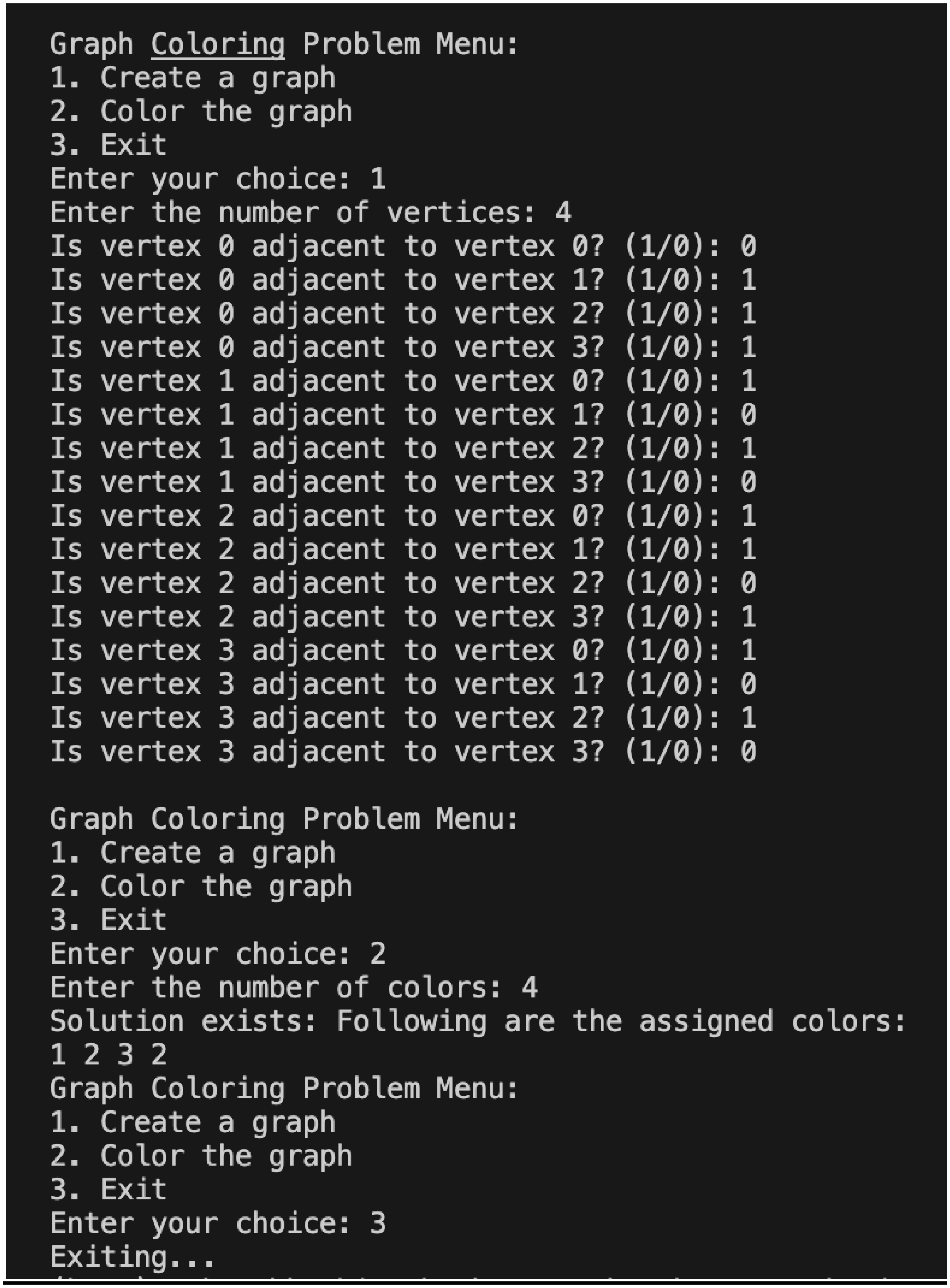


```
N-Queens Problem Menu:
1. Solve N-Queens
2. Exit
Enter your choice: 1
Enter the board size (N): 4

Solution:
. . Q .
Q . . .
. . . Q
. Q . .

N-Queens Problem Menu:
1. Solve N-Queens
2. Exit
Enter your choice: 2
Exiting...
```

Graph Coloring

```
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = [[0 for _ in range(vertices)] for _ in range(vertices)]

    def is_safe(self, v, colour, c):
        for i in range(self.vertices):
            if self.graph[v][i] == 1 and colour[i] == c:
                return False
        return True

    def graph_colouring_util(self, m, colour, v):
        if v == self.vertices:
            return True

        for c in range(1, m + 1):
            if self.is_safe(v, colour, c):
                colour[v] = c
                if self.graph_colouring_util(m, colour, v + 1):
                    return True
                colour[v] = 0

    def graph_colouring(self, m):
        colour = [0] * self.vertices
        if not self.graph_colouring_util(m, colour, 0):
            return False
        print("Solution exists: Following are the assigned colors:")
        for c in colour:
            print(c, end=" ")
        return True



def create_graph():
    vertices = int(input("Enter the number of vertices: "))
    g = Graph(vertices)

    for i in range(vertices):
        for j in range(vertices):
            g.graph[i][j] = int(input(f"Is vertex {i} adjacent to vertex {j}? (1/0): "))

    return g



def main():
    while True:
        print("\nGraph Coloring Problem Menu:")
        print("1. Create a graph")
        print("2. Color the graph")
        print("3. Exit")
        choice = int(input("Enter your choice: "))

        if choice == 1:
            g = create_graph()
        elif choice == 2:
            if 'g' in locals():
                m = int(input("Enter the number of colors: "))
                g.graph_colouring(m)
            else:
                print("Please create a graph first.")
        elif choice == 3:
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please choose a valid option.")


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

Output:



```
Graph Coloring Problem Menu:
1. Create a graph
2. Color the graph
3. Exit
Enter your choice: 1
Enter the number of vertices: 4
Is vertex 0 adjacent to vertex 0? (1/0): 0
Is vertex 0 adjacent to vertex 1? (1/0): 1
Is vertex 0 adjacent to vertex 2? (1/0): 1
Is vertex 0 adjacent to vertex 3? (1/0): 1
Is vertex 1 adjacent to vertex 0? (1/0): 1
Is vertex 1 adjacent to vertex 1? (1/0): 0
Is vertex 1 adjacent to vertex 2? (1/0): 1
Is vertex 1 adjacent to vertex 3? (1/0): 0
Is vertex 2 adjacent to vertex 0? (1/0): 1
Is vertex 2 adjacent to vertex 1? (1/0): 1
Is vertex 2 adjacent to vertex 2? (1/0): 0
Is vertex 2 adjacent to vertex 3? (1/0): 1
Is vertex 3 adjacent to vertex 0? (1/0): 1
Is vertex 3 adjacent to vertex 1? (1/0): 0
Is vertex 3 adjacent to vertex 2? (1/0): 1
Is vertex 3 adjacent to vertex 3? (1/0): 0

Graph Coloring Problem Menu:
1. Create a graph
2. Color the graph
3. Exit
Enter your choice: 2
Enter the number of colors: 4
Solution exists: Following are the assigned colors:
1 2 3 2
Graph Coloring Problem Menu:
1. Create a graph
2. Color the graph
3. Exit
Enter your choice: 3
Exiting...
```