

1. Prims Algorithm

```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
import sys

def prims_algorithm(graph):
    num_vertices = len(graph)
    selected_vertices = [False] * num_vertices
    mst_edges = []

    # Start from the first vertex
    selected_vertices[0] = True

    while len(mst_edges) < num_vertices - 1:
        min_edge = (None, None, sys.maxsize)

        for i in range(num_vertices):
            if selected_vertices[i]:
                for j in range(num_vertices):
                    if not selected_vertices[j] and graph[i][j]:
                        if graph[i][j] < min_edge[2]:
                            min_edge = (i, j, graph[i][j])

        mst_edges.append(min_edge)
        selected_vertices[min_edge[1]] = True

    return mst_edges

# Example usage
graph = [
    [0, 2, 0, 6, 0],
    [2, 0, 3, 8, 5],
    [0, 3, 0, 0, 7],
    [6, 8, 0, 0, 9],
    [0, 5, 7, 9, 0]
]

mst = prims_algorithm(graph)
print("Edges in the Minimum Spanning Tree:")
for edge in mst:
    print(f'{edge[0]} - {edge[1]}: {edge[2]}')
```

```
IDLE Shell 3.12.1
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:\Users\sleva\Desktop\Duplicate.py
Edges in the Minimum Spanning Tree:
0 - 1: 2
1 - 2: 3
1 - 4: 5
0 - 3: 6
>>>
```

2. Backtracking

```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 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, -1, -1), range(col, n)):
        if board[i][j] == 1:
            return False
    return True

def solve_n_queens_util(board, row, n):
    if row >= n:
        return True
    # Consider this row and try placing a queen in all columns one by one
    for col in range(n):
        if is_safe(board, row, col, n):
            # Place this queen in board[row][col]
            board[row][col] = 1
            # Recur to place the rest of the queens
            if solve_n_queens_util(board, row + 1, n):
                return True
            board[row][col] = 0
    return False

def solve_n_queens(n):
    board = [[0] * n for _ in range(n)]
    if solve_n_queens_util(board, 0, n):
        return board
    else:
        return None

def print_board(board):
    for row in board:
        print("".join('Q' if col else '.' for col in row))

# Example usage
n = 8
solution = solve_n_queens(n)
if solution:
    print(f"One of the solutions for {n}-Queens problem:")
    print_board(solution)
else:
    print(f"No solution exists for {n}-Queens problem.")
```

```
IDLE Shell 3.12.1
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:\Users\sleva\Desktop\Duplicate.py
One of the solutions for 8-Queens problem:
Q . . . . .
. . . . . Q
. . . . . Q
. Q . . . .
. . . . . Q
. Q . . . .
. Q . . . .
. Q . . . .
>>>
```

3. N Queens Problem

```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
def is_valid(board, row, col):
    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

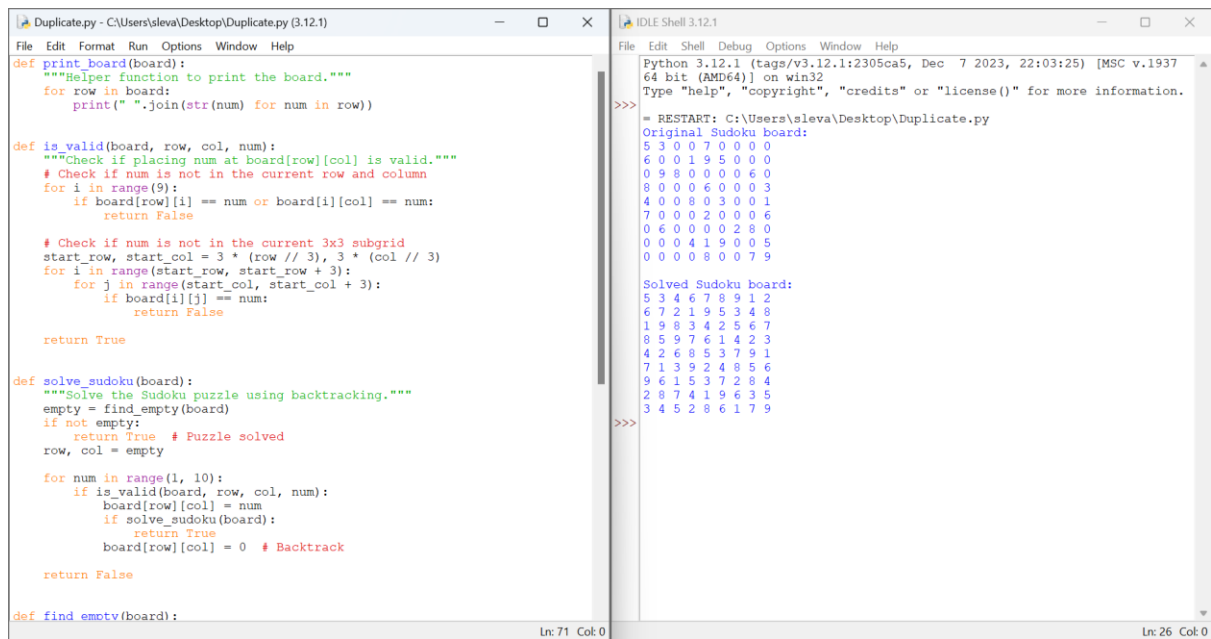
def solve_n_queens(n):
    def backtrack(row, board):
        if row == n:
            result.append(board[:])
            return
        for col in range(n):
            if is_valid(board, row, col):
                board[row] = col
                backtrack(row + 1, board)
                board[row] = -1

    result = []
    board = [-1] * n
    backtrack(0, board)
    return result

# Example usage:
n = 4
solutions = solve_n_queens(n)
for sol in solutions:
    print(''.join('Q' + ' ' * (n - row - 1) for row, col in enumerate(sol)))
    print()
```

```
IDLE Shell 3.12.1
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:\Users\sleva\Desktop\Duplicate.py
. Q . .
. . . Q
Q . . .
. . Q .
>>>
```

4. Sudoku Problem



The image shows a screenshot of an IDE with two windows. The left window, titled 'Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)', contains a Python script for solving a Sudoku puzzle. The script includes functions for printing the board, validating a move, and solving the puzzle using backtracking. The right window, titled 'IDLE Shell 3.12.1', shows the output of the script, which includes the original Sudoku board and the solved board.

```
def print_board(board):
    """Helper function to print the board."""
    for row in board:
        print(" ".join(str(num) for num in row))

def is_valid(board, row, col, num):
    """Check if placing num at board[row][col] is valid."""
    # Check if num is not in the current row and column
    for i in range(9):
        if board[row][i] == num or board[i][col] == num:
            return False

    # Check if num is not in the current 3x3 subgrid
    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(start_row, start_row + 3):
        for j in range(start_col, start_col + 3):
            if board[i][j] == num:
                return False

    return True

def solve_sudoku(board):
    """Solve the Sudoku puzzle using backtracking."""
    empty = find_empty(board)
    if not empty:
        return True # Puzzle solved
    row, col = empty

    for num in range(1, 10):
        if is_valid(board, row, col, num):
            board[row][col] = num
            if solve_sudoku(board):
                return True
            board[row][col] = 0 # Backtrack

    return False

def find_empty(board):
```

```
>>>
= RESTART: C:\Users\sleval\Desktop\Duplicate.py
Original Sudoku board:
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9

Solved Sudoku board:
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9

>>>
```