**Experiment-No.6**

**Objective:**Write a program to implement 8 Queen Problems.

| **Scheduled Date:** | **Compiled Date:** | **Submitted Date:** |
| --- | --- | --- |
| 24 Sep 2023 | 28 Sep 2023 | 29- Sep 2023 |

**Description of 8 Queen Problem:**

The 8 Queen Problem is a classic problem in computer science and mathematics that involves placing 8 queens on a standard 8x8 chessboard in such a way that no two queens threaten each other. In chess, a queen can move any number of squares vertically, horizontally, or diagonally, which means that two queens cannot be placed on the same row, column, or diagonal.

**The challenge is to position 8 queens on the chessboard such that:**

* No two queens share the same row.
* No two queens share the same column.
* No two queens share the same diagonal.

**Algorithm for 8 Queen Problem:**

### Steps:

1. **Initialize the Data Structures**:
   * Add the startNode to the visited list.
   * Push the startNode onto the stack.
2. **Iterate While Stack is Not Empty**:
   * Pop the last element (node) from the stack and assign it to m.
   * Check if m is one of the target nodes:

If yes, print that the target has been reached and exit the loop.

* + Otherwise, print the current node (m).

1. **Explore Neighbors**:
   * For each neighbor n of node m (i.e., nodes connected to m in the graph):

If the neighbor n has not been visited:

Mark n has visited by adding it to the visited list.

Push n onto the stack.

1. **Continue** the process until:
   * A target node is found, or
   * The stack becomes empty (indicating all reachable nodes have been visited).
2. **Termination**:
   * If the stack is exhausted without finding any target, the search concludes with no target found.

**Python Code for 8 Queen Problem:**

**# declaring states**

**states = {**

**'A': ['B1', 'C1', 'D1', 'E1', 'F1', 'G1', 'H1', 'I1'],**

**'B1': ['C2', 'D2', 'E2', 'F2', 'G2', 'H2'],**

**'C1': ['A2', 'D2', 'E2', 'F2', 'G2', 'H2', 'I2'],**

**'D1': ['A2', 'B2', 'E2', 'F2', 'G2', 'H2', 'I2'],**

**'E1': ['A2', 'B2', 'C2', 'F2', 'G2', 'H2', 'I2'],**

**'F1': ['A2', 'B2', 'C2', 'D2', 'G2', 'H2', 'I2'],**

**'G1': ['A2', 'B2', 'C2', 'D2', 'E2', 'H2', 'I2'],**

**'H1': ['A2', 'B2', 'C2', 'D2', 'E2', 'F2', 'I2'],**

**'I1': ['A2', 'B2', 'C2', 'D2', 'E2', 'F2', 'G2'],**

**'C2': ['A3', 'D3', 'E3', 'F3', 'G3', 'H3'],**

**'D2': ['A3', 'B3', 'E3', 'F3', 'G3', 'H3', 'I3'],**

**'E2': ['A3', 'B3', 'C3', 'F3', 'G3', 'H3', 'I3'],**

**'F2': ['A3', 'B3', 'C3', 'D3', 'G3', 'H3', 'I3'],**

**'G2': ['A3', 'B3', 'C3', 'D3', 'E3', 'H3', 'I3'],**

**'H2': ['A3', 'B3', 'C3', 'D3', 'E3', 'F3', 'I3'],**

**'I2': ['A3', 'B3', 'C3', 'D3', 'E3', 'F3', 'G3'],**

**'A3': ['D4', 'E4', 'F4', 'G4', 'H4'],**

**'B3': ['A4', 'D4', 'E4', 'F4', 'G4', 'H4', 'I4'],**

**'C3': ['A4', 'B4', 'F4', 'G4', 'H4', 'I4'],**

**'D3': ['A4', 'B4', 'C4', 'E4', 'G4', 'H4', 'I4'],**

**'E3': ['A4', 'B4', 'C4', 'D4', 'F4', 'H4', 'I4'],**

**'F3': ['A4', 'B4', 'C4', 'D4', 'E4', 'G4', 'I4'],**

**'G3': ['A4', 'B4', 'C4', 'D4', 'E4', 'F4', 'H4'],**

**'D4': ['A5', 'B5', 'E5', 'F5', 'G5', 'H5', 'I5'],**

**'E4': ['A5', 'B5', 'C5', 'F5', 'G5', 'H5', 'I5'],**

**'F4': ['A5', 'B5', 'C5', 'D5', 'G5', 'H5', 'I5'],**

**'G4': ['A5', 'B5', 'C5', 'D5', 'E5', 'H5', 'I5'],**

**'H4': ['A5', 'B5', 'C5', 'D5', 'E5', 'F5', 'I5'],**

**'I4': ['A5', 'B5', 'C5', 'D5', 'E5', 'F5', 'G5'],**

**'A5': ['D6', 'E6', 'F6', 'G6', 'H6'],**

**'B5': ['A6', 'D6', 'E6', 'F6', 'G6', 'H6', 'I6'],**

**'C5': ['A6', 'B6', 'F6', 'G6', 'H6', 'I6'],**

**'D5': ['A6', 'B6', 'C6', 'E6', 'G6', 'H6', 'I6'],**

**'E5': ['A6', 'B6', 'C6', 'D6', 'F6', 'H6', 'I6'],**

**'F5': ['A6', 'B6', 'C6', 'D6', 'E6', 'G6', 'I6'],**

**'G5': ['A6', 'B6', 'C6', 'D6', 'E6', 'F6', 'H6'],**

**'D6': ['A7', 'B7', 'E7', 'F7', 'G7', 'H7', 'I7'],**

**'E6': ['A7', 'B7', 'C7', 'F7', 'G7', 'H7', 'I7'],**

**'F6': ['A7', 'B7', 'C7', 'D7', 'G7', 'H7', 'I7'],**

**'G6': ['A7', 'B7', 'C7', 'D7', 'E7', 'H7', 'I7'],**

**'H6': ['A7', 'B7', 'C7', 'D7', 'E7', 'F7', 'I7'],**

**'I6': ['A7', 'B7', 'C7', 'D7', 'E7', 'F7', 'G7'],**

**'A7': ['D8', 'E8', 'F8', 'G8', 'H8'],**

**'B7': ['A8', 'D8', 'E8', 'F8', 'G8', 'H8', 'I8'],**

**'C7': ['A8', 'B8', 'F8', 'G8', 'H8', 'I8'],**

**'D7': ['A8', 'B8', 'C8', 'E8', 'G8', 'H8', 'I8'],**

**'E7': ['A8', 'B8', 'C8', 'D8', 'F8', 'H8', 'I8'],**

**'F7': ['A8', 'B8', 'C8', 'D8', 'E8', 'G8', 'I8'],**

**'G7': ['A8', 'B8', 'C8', 'D8', 'E8', 'F8', 'H8'],**

**'D8': [],**

**'E8': [],**

**'F8': [],**

**'G8': [],**

**'H8': [],**

**'I8': [],**

**'A2': [],**

**'A3': [],**

**'A4': [],**

**'A6': [],**

**'A8': [],**

**'B3': [],**

**'B4': [],**

**'B5': [],**

**'B2': [],**

**'H3': [],**

**'I3': [],**

**'C4': [],**

**'H5': [],**

**'B6': [],**

**'C6': [],**

**'H7': [],**

**'I7': [],**

**'I5': []**

**}**

**target = ['D8', 'E8', 'F8', 'G8', 'H8', 'I8']**

**visited = []**

**stack = []**

**# finding path using dfs algo**

**def dfs(visited, states, node):**

**visited.append(node)**

**stack.append(node)**

**while stack:**

**m = stack.pop()**

**if m in target:**

**print("(Target Reached)")**

**break**

**print(m, end=" ")**

**for n in states[m]:**

**if n not in visited:**

**visited.append(n)**

**stack.append(n)**

**dfs(visited, states, 'A')**

**Output :- A I1 G2 I3 H3 E3 I4 G5 H6 I7 F7 (Target Reached)**