****

**Green University of Bangladesh**

**Department of Computer Science and Engineering(CSE)**

**Faculty of Sciences and Engineering**

**Semester: (Spring, Year:2024), B.Sc. in CSE (Day)**

**LAB REPORT NO #05**

**Course Title: Artificial Intelligence Lab**

**Course Code: CSE 316 Section: 213D3**

**Lab Experiment Name: Solve N-Queen Problem Using Backtracking**

**Algorithm**

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
| **1.** | Shoaib Ahmed | 212902019 |

**Lab Date : 11-05-24**

**Submission Date : 18-05-24**

**Course Teacher’s Name : Md. Zahidul Hasan**

**[For Teachers use only: Don’t Write Anything inside this box]**

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature:.....................**  **Comments:.............................................. Date:..............................** |

**1. TITLE OF THE LAB EXPERIMENT**

Solve N-Queen Problem Using Backtracking Algorithm

**2. OBJECTIVES/AIM**

* Implement the backtracking algorithm to solve the N-Queens problem.
* Given a chessboard size (number of queens), the code determines the legal

arrangements of queens such that no queens can attack each other (no queens share the same row, column, or diagonal).

* It counts the total number of valid placements (solutions) for the N queens on

the chessboard

**3. PROCEDURE / ANALYSIS / DESIGN**

* The program starts by asking the user for input (num\_of\_queens).
* It initializes a board with zeros.
* It recursively tries to place queens column by column, checking safety with is\_position\_safe.
* Once all queens are placed correctly (col >= len(board)), it counts it as a valid solution.
* After exploring all possibilities, it prints the total number of valid solutions found.

**4. IMPLEMENTATION**

Code:

|  |
| --- |
| def is\_position\_safe(board, row, col):  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, len(board), 1), range(col, -1, -1)):  if board[i][j] == 1:  return False    return True  def solve\_n\_queen(board, col):  if col >= len(board):  return 1    count = 0  for i in range(len(board)):  if is\_position\_safe(board, i, col):  board[i][col] = 1  count += solve\_n\_queen(board, col + 1)  board[i][col] = 0 # Backtrack    return count  def main():  num\_of\_queens = int(input("Enter the number of queens: "))  board = [[0] \* num\_of\_queens for \_ in range(num\_of\_queens)]  print("Number of solutions: ", solve\_n\_queen(board, 0))  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**5. TEST RESULT / OUTPUT**

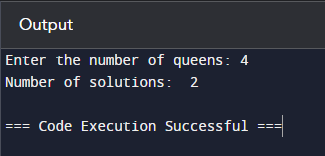


Fig-5.1: First Output

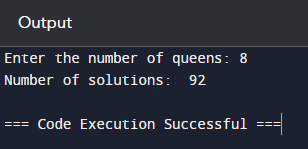


Fig-5.2: Second Output

**6. ANALYSIS AND DISCUSSION**

* **Approach**: Uses recursive backtracking to systematically explore and validate queen placements.
* **Efficiency**: The algorithm efficiently prunes invalid paths and explores only valid configurations.
* **Output**: Provides the total count of valid N-Queens configurations for a given num\_of\_queens.

This method effectively solves the N-Queens problem by leveraging recursive exploration and backtracking, ensuring all constraints are met for placing queens on the board.