|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Analysis of Algorithms** | **Semester:** | **IV** |
| **Date of Performance:** | **03 / 04 / 2024** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Dr. Aarti Phadke** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 9**

**Title: 8 - Queens Problem**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Implementation of N-Queen Problem using Backtracking Algorithm.  **Objective:** To learn the Backtracking strategy of problem solving for N-Queens problem. |

|  |
| --- |
| **COs to be achieved:** |
| **CO2:** Describe various algorithm design strategies to solve different problems. |

|  |
| --- |
| **Apparatus / Software Tools Used:** |
| 1. VS Code 2. Microsoft Excel |

|  |
| --- |
| **Theory:** |
| **Historical Profile:**  The N-Queens puzzle is the problem of placing N queens on an N×N chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.  **New Concepts to be learned:**  Application of algorithmic design strategy to any problem, Backtracking method of problem-solving Vs other methods of problem solving, 8- Queens problem and its applications.  **Example 8-Queens Problem:**  The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other i.e. no two queens share the same row, column, or diagonal.  **Solution Using Backtracking Approach:**  The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false. |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| **Algorithm Insertion Sort**  void NQueens(int k, int n)  // Using backtracking, this procedure prints all possible placements of n queens on an n X n chessboard so that they are nonattacking.  {  for (int i = 1; i <= n; i++)  {  if (Place(k, i))  {  x[k] = i;  if (k == n)  for (int j = 1; j <= n; j++)  Print x[j];  else  NQueens(k + 1, n);  }  }  }  **Boolean Place(int k, int i)**  // Returns true if a queen can be placed in kth row and ith column. Otherwise, it returns false.  // x[] is a global array whose first (k-1) values have been set. abs(r) returns the absolute value of r.  {  for (int j = 1; j < k; j++)  {  if ((x[j] == i) // Two in the same column  || (abs(x[j] - i) == abs(j - k))) // or in the same diagonal  return false;  }  return true;  } |

|  |
| --- |
| **Upload the code / Output:** |
| Code:  def place(k, i):      for j in range(1, k):          if x[j] == i or abs(x[j] - i) == abs(j - k):              return False      return True  def print\_solution(n):      for i in range(1, n+1):          for j in range(1, n+1):              if x[i] == j:                  print("Q", end=" ")              else:                  print("x", end=" ")          print()  def n\_queens(k, n):      global count        for i in range(1, n+1):          if place(k, i):              x[k] = i              if k == n:                  count += 1                  print("\nsolution", count, ": ")                  print\_solution(n)                  print()              else:                  n\_queens(k+1, n)  print("\nN-Queens Problem")  n = int(input("\nenter number of queens: "))  x = [0] \* (n+1)  count = 0  n\_queens(1, n)  print("total solutions:" , count)  Code Output:  n = 6    n = 4    Handwritten Solution (4 x 4): |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **Draw the State Space tree for 4-Queens problem (Solution).** |

|  |
| --- |
| **Conclusion:** |
| We successfully implemented and learnt N-Queen Problem using Backtracking Algorithm. |

**Signature of faculty in-charge with Date:**