|  |  |
| --- | --- |
| **NAME:** | Hardik Garg |
| **UID:** | 2021300036 |
| **SUBJECT** | Design and Analysis of Algorithms |
| **EXPERIMENT NO:** | 07 |
| **AIM:** | To implement Backtracking (N Queen Problem) |
| **Algorithm:** | **Backtracking Algorithm**  function solveNQueens(board, col, n):  if col >= n:  print board  return true  for row from 0 to n-1:  if isSafe(board, row, col, n):  board[row][col] = 1  if solveNQueens(board, col+1, n):  return true  board[row][col] = 0  return false  function isSafe(board, row, col, n):  for i from 0 to col-1:  if board[row][i] == 1:  return false  for i,j from row-1, col-1 to 0, 0 by -1:  if board[i][j] == 1:  return false  for i,j from row+1, col-1 to n-1, 0 by 1, -1:  if board[i][j] == 1:  return false  return true  board = empty NxN chessboard  solveNQueens(board, 0, N) |
| **CODE:** | #include <stdio.h>  #include <stdbool.h>  void printSolution(int n, int board[n][n]) {  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  printf("%c ", board[i][j] ? 'Q' : '.');  }  printf("\n");  }  printf("\n");  }  bool isSafe(int n, int board[n][n], int row, int col) {  int i, j;  // Check the left side of the row  for (i = 0; i < col; i++) {  if (board[row][i]) {  return false;  }  }  // Check upper diagonal on left side  for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {  if (board[i][j]) {  return false;  }  }  // Check lower diagonal on left side  for (i = row, j = col; j >= 0 && i < n; i++, j--) {  if (board[i][j]) {  return false;  }  }  return true;  }  void solveNQueensUtil(int n, int board[n][n], int col) {  if (col == n) {  printSolution(n, board);  return;  }  for (int i = 0; i < n; i++) {  if (isSafe(n, board, i, col)) {  board[i][col] = 1;  solveNQueensUtil(n, board, col+1);  board[i][col] = 0;  }  }  }  void solveNQueens(int n) {  int board[n][n];  // Initialize the board to all 0s  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  board[i][j] = 0;  }  }  solveNQueensUtil(n, board, 0);  }  int main() {  int n = 0;  printf("\nEnter the dimension of the chessboard : ");  scanf("%d",&n);  solveNQueens(n);  return 0;  } |
| **Output:** | **6**    **4** |
| **Conclusion:** | In conclusion, the N-Queen problem is a challenging puzzle that requires finding a solution to place N queens on an N x N chessboard such that no two queens threaten each other. In this experiment, we have used backtracking to solve the N-Queen problem efficiently. We first implemented a recursive backtracking algorithm that generated all possible configurations of the board and checked whether they were valid solutions. Then, we optimized the algorithm by using pruning techniques, which significantly reduced the search space and improved the runtime. |