|  |  |
| --- | --- |
| Name : | Ms.Neerja Doshi |
| UID | 2021300029 |
| Div | SE CE DIV A – BATCH B |
| Experiment No. 7: Backtracking strategy – N-Queen problem | |

|  |  |
| --- | --- |
| AIM : | Backtracking strategy – N-Queen problem |
| THEORY : | 1. The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other 2. **ALGORITHM :**    1. Initialize an empty chessboard of size NxN.    2. Start with the leftmost column and place a queen in the first row of that column.    3. Move to the next column and place a queen in the first row of that column.    4. Repeat step 3 until either all N queens have been placed or it is impossible to place a queen in the current column without violating the rules of the problem.    5. If all N queens have been placed, print the solution.    6. If it is not possible to place a queen in the current column without violating the rules of the problem, backtrack to the previous column.    7. Remove the queen from the previous column and move it down one row.    8. Repeat steps 4-7 until all possible configurations have been tried.  |  | | --- | | 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) |   **3. Time Complexity :**   * 1. Time Complexity: O(N!) where N is number of queens , and also the the number of rows and columns in given board . |
| CODE: | #include <stdio.h>  #include <math.h>  #include <stdlib.h>  //board index --> row no , borad[index] --> col no at which queen is placed  *int* board[20], count;  *int* main()  {  *int* n, i, j;  *void* queen(*int* *row*, *int* *n*);      printf(" - N Queens Problem Using Backtracking -");      printf("\n\nEnter number of Queens:");      scanf("%d", &n);      queen(1, n);      return 0;  }  // function for printing the solution  *void* print(*int* *n*)  {  *int* i, j;      printf("\n\nSolution %d:\n\n", ++count);      for (i = 1; i <= *n*; ++i){      //column indexes          printf("\t%d", i);      }      for (i = 1; i <= *n*; ++i)      {          //row indexes          printf("\n%d\t", i);          for (j = 1; j <= *n*; ++j) // for nxn board          {              if (board[i] == j)              // queen at i,j position                  printf("Q\t");              else                  printf(".\t"); // empty slot          }      }      printf("\n\n");  }  /\*funtion to check conflicts  If no conflict for desired postion returns 1 otherwise returns 0\*/  *int* place(*int* *row*, *int* *column*)  {  *int* i;      for (i = 1; i <= *row* - 1; ++i)      {          // checking column cond.          // if board[current\_row] has value == current col .. not allowed          if (board[i] == *column*)              return 0;          // check diagonal          else if (abs(board[i] - *column*) == abs(i - *row*))              return 0;      }      return 1; // all cond met  }  // function --> if postion locked .. place queen .. move to next  *void* queen(*int* *row*, *int* *n*)  {  *int* column;      for (column = 1; column <= *n*; ++column)      {          // printf("(%d-%d)\n" , row , column);          if (place(*row*, column))          {              // printf("yes\n");              // if all condition met .. place queen              board[*row*] = column;              // all rows handled ... print final board config              if (*row* == *n*)                  print(*n*);              else                  // one row done .. move to next                  queen(*row* + 1, *n*);          }      }  } |
| OUTPUT : |  |
| CONCLUSION : | By performing the above experiment , i have successfully understood to perform Backtracking by solving N – Queen Problem by Taking N = 8 .  A total of 92 solution were seen out of which , 6 are shown above. |