STATEMENT:

To implement 16-puzzle program using Backtracking

EXPLANATION:

Backtracking is an algorithm technique for solving problem recursively by trying to build a solution incrementally,one piece at a time removing those solutions that fail to satisfy the constraints of the problem at any point of time(by time, here, is referred to the time elapsed till reaching any search tree).

ALGORITHM:

step 1:a)Start in the left most column

b)If all the queens are placed return tree

c)Try all rows in the current column Do following for every tired row.

step 2:a)If the queen can be placed safely in this row then mark .This as part of the solution and recursively check if placing queen here leads to a solution.

b)If placing the queen in leads to a solution then return true.

c)If placing queen doesn't lead to a solution then unmark this and go to step(a) to try other rows.

step 3:If all rows have been tried and nothing worked,return falls to trigger backtracking.

PROGRAM IN C:

```
#define N 4
#include <stdbool.h>
#include <stdio.h>
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
         for (int j = 0; j < N; j++)
        printf(" %d ", board[i][j]);
           printf("\n");
     }
}
bool isSafe(int board[N][N], int row, int col)
{
      int i, j;
     for (i = 0; i < col; i++)
          if (board[row][i])
        return false;
      for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
           if (board[i][j])
        return false;
```

```
        for (i = row, j = col; j >= 0 && i < N; i++, j--)
            if (board[i][j])
          return false;
          return true;
}
bool solveNQUtil(int board[N][N], int col)
{
         if (col >= N)
             return true;
       for (int i = 0; i < N; i++) {
           if (is Safe(board, i, col)) {
               board[i][col] = 1;
           if (solveNQUtil(board, col + 1))
          return true;
              board[i][col] = 0;
        }
}
    return false;
}
bool solveNQ()
{
```

```
        int board[N][N] = { { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 } };

        if (solveNQUtil(board, 0) == false) {
            printf("Solution does not exist");
            return false;
        }
        printSolution(board);
        return true;
}
int main()
{
     solveNQ();
      return 0;
}
```

OUTPUT:

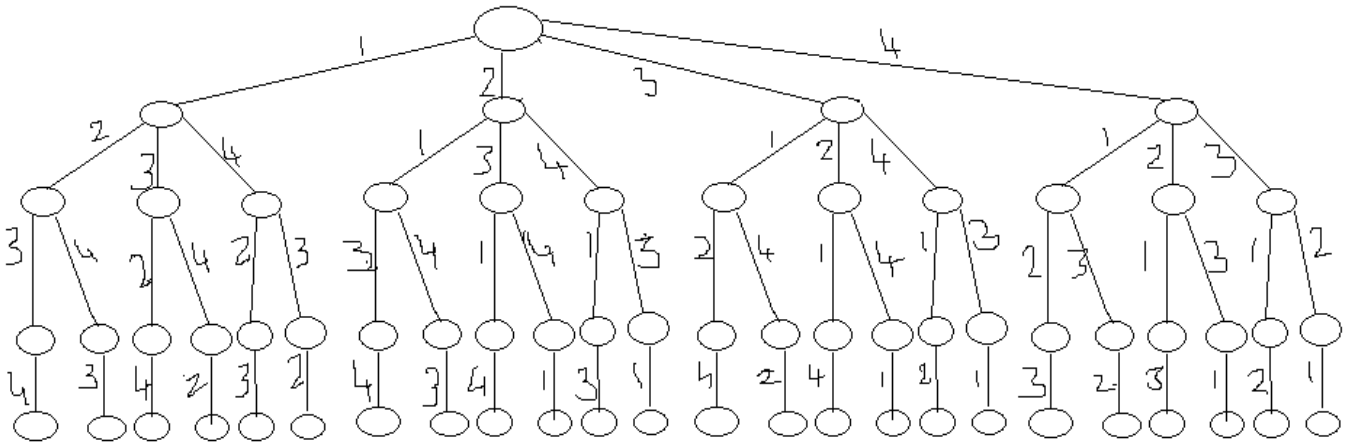
0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

EXPLANATION:



Q1			
	Q2		
		Q3	
			Q4

Q1			
	Q2		
			Q3
		Q4	

Q1			
		Q2	
	Q3		
			Q4

Q1			
		Q2	
			Q3
	Q4		

Q1			
			Q2
	Q3		
		Q4	

Q1			
			Q2
		Q3	
	Q4		

	Q1		
Q2			
		Q3	
			Q4

	Q1		
Q2			
			Q3
		Q4	

	Q1		
		Q2	
Q3			
			Q4

	Q1		
			Q2
		Q3	
Q4			

	Q1		
			Q2
Q3			
		Q4	

	Q1		
		Q2	
			Q3
Q4			

		Q1	
Q2			
	Q3		
			Q4

		Q1	
			Q2
Q3			
	Q4		

		Q1	
	Q2		
Q3			
			Q4

		Q1	
			Q2
	Q3		
Q4			

		Q1	
			Q2
Q3			
	Q4		

		Q1	
	Q2		
			Q3
Q4			

			Q1
Q2			
	Q3		
		Q4	

			Q1
Q2			
	Q3		
	Q4		

			Q1
	Q2		
Q3			
		Q4	

			Q1
		Q2	
	Q3		
Q4			

			Q1
		Q2	
Q3			
	Q4		

			Q1
	Q2		
		Q3	
Q4			

$$FORMULA: 1 + \sum_{i=0}^{3} [\prod_{j=0}^{i} (N - J)]$$

TIME COMPLEXITY:

O(n!)