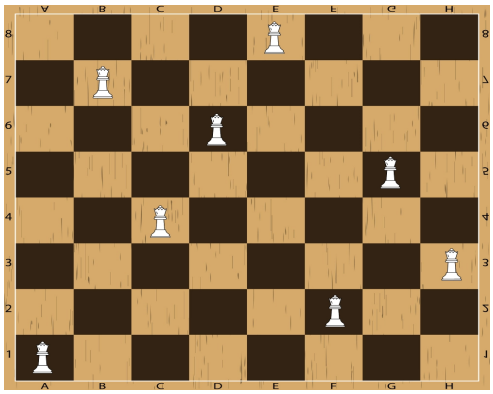
Assignment No.3

Aim: Implement 8 queens problem using Backtracking.

Theory:

Backtracking is a recursive approach for solving any problem where we must search among all the possible solutions following some constraints. More precisely, we can say that it is an improvement over the brute force technique.

Example: One possible solution to the 8 queens problem using backtracking is shown below. In the first row, the queen is at E8 square, so we have to make sure no queen is in column E and row 8 and also along its diagonals. Similarly, for the second row, the queen is on the B7 square, thus, we have to secure its horizontal, vertical, and diagonal squares. The same pattern is followed for the rest of the queens.



Output:

0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0

Bruteforce Approach

One brute-force approach to solving this problem is as follows:

- Generate all possible permutations of the numbers 1 to 8, representing the columns on the chessboard.
- For each permutation, check if it represents a valid solution by checking that no two queens are in the same row or diagonal.
- If a valid solution is found, print the board layout.

While this approach works for small numbers, it quickly becomes inefficient for larger sizes as the number of permutations to check grows exponentially. More efficient algorithms, such as backtracking or genetic algorithms, can be used to solve the problem in a more optimized way.

Backtracking Approach

This approach rejects all further moves if the solution is declined at any step, goes back to the previous step and explores other options.

Algorithm

Let's go through the steps below to understand how this algorithm of solving the 8 queens problem using backtracking works:

- **Step 1:** Traverse all the rows in one column at a time and try to place the queen in that position.
- **Step 2:** After coming to a new square in the left column, traverse to its left horizontal direction to see if any queen is already placed in that row or not. If a queen is found, then move to other rows to search for a possible position for the queen.
- **Step 3:** Like step 2, check the upper and lower left diagonals. We do not check the right side because it's impossible to find a queen on that side of the board yet.
- **Step 4:** If the process succeeds, i.e. a queen is not found, mark the position as '1' and move ahead.
- **Step 5:** Recursively use the above-listed steps to reach the last column. Print the solution matrix if a queen is successfully placed in the last column.
- **Step 6:** Backtrack to find other solutions after printing one possible solution.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int countt=0;
// A function to print a solution
void print(int board[][8]){
    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            cout<<board[i][j]<<" ";
        }
        cout<<endl;
    }
    cout<<"-----------------\n";
}
//Function to check whether a position is valid or not
bool isValid(int board[][8],int row,int col){
    //loop to check horizontal positions
    for(int i=col;i>=0;i--){
        if(board[row][i])
        return false;
    }
    int i=row,j=col;
    //loop to check the upper left diagonal
    while(i>=0&&j>=0){
        if(board[i][j])
        return false;
        i--;
        j--;
    }
    i=row;
    j=col;
    //loop to check the lower left diagonal
    while(i<8&&j>=0){
        if(board[i][j])
        return false;
        i++;
        j--;
    }
    return true;
```

```
}
//function to check all the possible solutions
void ninjaQueens(int board[][8],int currentColumn){
   if(currentColumn>=8)
   return;
   //loop to cover all the columns
   for(int i=0;i<8;i++){
      if(isValid(board,i,currentColumn)){
         board[i][currentColumn]=1;
         if(currentColumn==7){
            print(board);
            countt++;
         }
         //recursively calling the function
         ninjaQueens(board,currentColumn+1);
         //backtracking
         board[i][currentColumn]=0;
      }

   }
}
int main() {
   //initial board situation
   int board[8][8]={{0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0}};
   ninjaQueens(board,0);
   /* In total, 92 solutions exist for 8x8 board. This statement will verify our code*/
   cout<<countt<<endl;
   return 0;
}
```

Output:

A total of 92 solutions will exist, we are showing the last two along with the count.

```
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
----------------
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
----------------
92
```