Experiment No. 5

Aim

To implement the **8 Queen's Problem** using the **Depth-First Search (DFS) approach** in C++.

Theory

The **8 Queen's Problem** is a classic problem in the field of Artificial Intelligence and combinatorial optimization. The problem statement is:

> Place 8 queens on a standard 8×8 chessboard such that no two queens attack each other.

A queen in chess can move:

- Vertically
- Horizontally
- Diagonally

Thus, the constraint is that **no two queens should share the same row, column, or diagonal**.

Key Concepts:

- **Depth-First Search (DFS):**
  DFS is a graph/tree traversal method where the search goes deep along one branch before backtracking. For the N-Queens problem, DFS explores possible queen placements column by column, row by row, backtracking when conflicts arise.
- **Backtracking in DFS:**
  If placing a queen in a particular row/column leads to an invalid configuration (conflict), DFS backtracks to the previous step and tries the next possible option.
- **State Space Representation:**
  - Each level (depth) of DFS corresponds to placing a queen in a column.
  - Each node represents a partial solution.
  - A goal state is reached when 8 queens are placed without conflict.

Complexity:

- **Time Complexity:** O(N!) in the worst case, since we may need to explore all row placements for N queens.
- **Space Complexity:** O(N) for recursion stack and placement storage.

Algorithm

1. Start with an empty chessboard of size 8×8.
2. Begin with the first column (col = 0).
3. For the current column, try placing a queen in each row one by one:
   a. Check if placing the queen is safe (no other queen in the same row, column, or diagonals).
   b. If safe, place the queen and recursively attempt to place the next queen in the next column.
   c. If not safe, move to the next row in the same column.
4. If a placement leads to a solution where all 8 queens are placed:
   - Print/display the configuration as a solution.
5. If no placement works in the current column, backtrack to the previous column and try the next possible row.
6. Repeat until all possible solutions are found.

Code (C++ Implementation)

```cpp
#include <iostream>
using namespace std;

#define N 8

int board[N][N];

bool safe(int r,int c){
    for(int i=0;i<c;i++) if(board[r][i]) return false;
    for(int i=r,j=c;i>=0&&j>=0;i--,j--) if(board[i][j]) return false;
    for(int i=r,j=c;i<N&&j>=0;i++,j--) if(board[i][j]) return false;
    return true;
}

bool solve(int c){
    if(c==N){
        for(int i=0;i<N;i++){
            for(int j=0;j<N;j++) cout<<(board[i][j]?"Q ":". ");
            cout<<"\n";
        }
        cout<<"\n";
        return true;
    }
    bool res=false;
    for(int i=0;i<N;i++){
        if(safe(i,c)){ board[i][c]=1; res=solve(c+1)||res; board[i][c]=0; }
    }
    return res;
}

int main(){
    if(!solve(0)) cout<<"No solution\n";
}
```

Input/Output Examples

Sample Output (One of the Possible Solutions)

```
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . . . . . . Q
Q . . . . . . .
. . Q . . . . .
. . . . Q . . .
. . . . . . Q .
```

- Multiple valid outputs exist, as there are **92 distinct solutions** to the 8 Queen's Problem.

Analysis of Code and Algorithm

- **Algorithm Used:** Depth-First Search with Backtracking.
- **Logic Flow:**
  - Recursively try each row for every column.
  - Place a queen if safe, else backtrack.
  - Continue until all queens are placed or no solution exists.
- **Time Complexity:** O(N!) in the worst case.
- **Space Complexity:** O(N) for recursion stack + O(N^2) for board storage.

- **Efficiency:** Although exponential, backtracking prunes many invalid states, making it practical for N=8.

Real-Life Applications

1. **Constraint Satisfaction Problems (CSPs):**
   Used in scheduling, time-tabling, and resource allocation problems.
2. **Artificial Intelligence Search Problems:**
   Demonstrates DFS with backtracking, which is fundamental in pathfinding, puzzle-solving, and game playing.
3. **Circuit Design and Parallel Processing:**
   Avoiding conflicts (like queen conflicts) is analogous to avoiding overlaps in circuits and processor tasks.

Conclusion

In this experiment, the **8 Queen's Problem** was successfully implemented using the **Depth-First Search (DFS) with backtracking** approach in C++. The program systematically explored the solution space, backtracked when conflicts occurred, and generated valid solutions.
This demonstrates the effectiveness of **DFS and backtracking** in solving combinatorial problems and provides insight into constraint satisfaction techniques in Artificial Intelligence.
