Solving the 5-Queens Problem Using Depth First Search (DFS)

- Problem Description

The objective of the problem is to place N queens on an N×N chessboard such that no two queens threaten each other.

In chess, a queen can attack another queen if they are placed on the same row, the same column, or the same diagonal.
In this project, the problem is solved for N = 5 (5-Queens problem), where five queens must be placed on a 5×5 chessboard while satisfying all the constraints.

- Algorithm Description – DFS

Depth First Search (DFS) is a search algorithm that explores a search tree by going as deep as possible along one branch before backtracking. In the 5-Queens problem, DFS is combined with backtracking to systematically try all possible positions for placing queens' row by row. At each row, the algorithm attempts to place a queen in every column that does not violate the constraints.
If no valid position is found for a row, the algorithm backtracks to the previous row and tries a different position.
This process continues until a valid solution is found or all possibilities are exhausted.

- State Representation

The chessboard is represented using a one-dimensional array.
The index of the array represents the row number, and the value stored at each index represents the column position of the queen in that row.
For example, if board [2] = 3, this means that a queen is placed at row 2 and column 3

- Constraint Checking

Before placing a queen in a specific position, the algorithm checks whether the position is safe. A position is considered unsafe if another queen exists in the same column or on the same diagonal. The diagonal conflict is checked by comparing the absolute difference between rows and columns.

- C++ code (5-Queens using DFS)

```
/* Hello , My Names : Karim Taha , Ahmed Walid
This Code about Solving the 5-queen problem using the DFS algorithm */
#include <iostream>
#include <vector>
#include <cmath>
```

```cpp
using namespace std;

int N = 5;
vector<int> board(5, -1);

// Make sure the place is safe by bool isSafe Function
bool isSafe(int row, int col) {
    for (int i = 0; i < row; i++) {
        // same Column or same diameter
        if (board[i] == col ||
            abs(board[i] - col) == abs(i - row)) {
            return false;
        }
    }
    return true;
}

// DFS + Backtracking
bool dfs(int row) {
    if (row == N) {
        return true;
    }

    for (int col = 0; col < N; col++) {
        if (isSafe(row, col)) {
            board[row] = col;      // نحط الملكة

            if (dfs(row + 1)) {    // نكمل
                return true;
            }

            board[row] = -1;       // Backtrack
        }
    }
    return false;
}

int main() {
    cout << "Solving 5-Queens using DFS : \n\n";

    if (dfs(0)) {
        cout << "Solution Found:\n\n";
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (board[i] == j)
                    cout << "Q ";
                else
                    cout << ". ";
            }
            cout << endl;
        }
    } else {
        cout << "No solution found \n";
    }

    cin.get();
    return 0;
}
```

- Finally

The DFS with backtracking approach provides an efficient and systematic solution to the 5-Queens problem. It reduces the search space by pruning invalid states early and guarantees finding a valid solution if one exists.

==

----Two people worked on this algorithm (Ahmed Walid Al-Shami and Karim Taha Awad).