N - Queen using Backtracking

```cpp
#include <bits/stdc++.h>

using namespace std;

class NQueens
{
private:
    int n;
    vector<vector<string>> ans;
    vector<vector<bool>> board;

public:
    NQueens(int n)
    {
        this->n = n;
    }

    bool isValid(int i, int j)
    {
        for (int row = 0; row < i; row++)
        {
            if (board[row][j])
                return false;
        }

        int x = i, y = j;

        while (x >= 0 && y >= 0)
        {
            if (board[x][y])
                return false;
            x--, y--;
        }

        x = i, y = j;

        while (x >= 0 && y < n)
        {
            if (board[x][y])
                return false;
            x--, y++;
        }

        return true;
    }

    bool util(int i)
    {
        if (i == n)
        {
            vector<string> temp;
            string s;

            for (int p = 0; p < n; p++)
            {
                s = "";
                for (int q = 0; q < n; q++)
                {
                    if (board[p][q])
                        s += 'Q';
                    else
                        s += '.';
                }
                temp.push_back(s);
            }
            ans.push_back(temp);
            return false;
        }

        for (int j = 0; j < n; j++)
        {
            if (isValid(i, j))
            {
                board[i][j] = true;
                bool legal = util(i + 1);
                if (legal)
                    return true;
                board[i][j] = false;
            }
        }

        return false;
    }

    vector<vector<string>> solveNQueens()
    {
        ans.clear();
        board.clear();
        board.resize(n, vector<bool>(n, false));
        util(0);
        return ans;
    }
};

int main()
{
    int n;
    cin >> n;
    auto obj = NQueens(n);
    vector<vector<string>> ans = obj.solveNQueens();

    for (auto board : ans)
    {
        for (auto row : board)
        {
            cout << row << endl;
        }
        cout << "--------------------------" << endl;
    }
}
```

```
}
```

```
# N - Queen using Branch And Bound

#include <bits/stdc++.h>

using namespace std;

class NQueens
{
private:
    int n;
    vector<vector<string>> ans;
    vector<vector<bool>> board;
    vector<bool> col, diag1, diag2;

public:
    NQueens(int n)
    {
        this->n = n;
    }

    bool isValid(int i, int j)
    {
        return !col[j] && !diag1[i - j + n - 1] && !diag2[i + j];
    }

    bool util(int i)
    {
        if (i == n)
        {
            vector<string> temp;
            string s;

            for (int p = 0; p < n; p++)
            {
                s = "";
                for (int q = 0; q < n; q++)
                {
                    if (board[p][q])
                        s += 'Q';
                    else
                        s += '.';
                }
                temp.push_back(s);
            }
            ans.push_back(temp);
            return false;
        }

        for (int j = 0; j < n; j++)
        {
            if (isValid(i, j))
            {
                board[i][j] = true;
                col[j] = true;
                diag1[i - j + n - 1] = true;
                diag2[i + j] = true;
                bool legal = util(i + 1);
                if (legal)
                    return true;
                board[i][j] = false;
                col[j] = false;
                diag1[i - j + n - 1] = false;
                diag2[i + j] = false;
            }
        }

        return false;
    }

    vector<vector<string>> solveNQueens()
    {
        ans.clear();
        board.clear();
        col.clear();
        diag1.clear();
        diag2.clear();
        board.resize(n, vector<bool>(n, false));
        col.resize(n, false);
        diag1.resize(2 * n - 1, false);
        diag2.resize(2 * n - 1, false);
        util(0);
        return ans;
    }
};

int main()
{
    int n;
    cin >> n;
    auto obj = NQueens(n);
    vector<vector<string>> ans = obj.solveNQueens();

    for (auto board : ans)
    {
        for (auto row : board)
        {
            cout << row << endl;
        }
        cout << "--------------------------" << endl;
    }
}
```