基础解法

Java代码

```
class Solution {
    public List<List<String>> solveNQueens(int n) {
        char[][] chess = new char[n][n];//初始化数组
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                chess[i][j] = '.';
        List<List<String>> res = new ArrayList<>();
        solve(res, chess, 0);
        return res;
    }
    private void solve(List<List<String>> res, char[][] chess, int row) {
        // 递归出口，最后一行都走完了，说明找到了一组，把它加入到集合res中
        if (row == chess.length) { res.add(construct(chess)); return; }
        for (int col = 0; col < chess.length; col++) { //遍历每一列
            if (valid(chess, row, col)) { // 判断这个位置是否可以放皇后
                char[][] temp = copy(chess); //数组复制一份
                temp[row][col] = 'Q';//在当前位置放个皇后
                solve(res, temp, row + 1); //递归到下一行继续
            }
        }
    }
  //把二维数组chess中的数据测下copy一份
    private char[][] copy(char[][] chess) {
        char[][] temp = new char[chess.length][chess[0].length];
        for (int i = 0; i < chess.length; i++) {
            for (int j = 0; j < chess[0].length; j++) {
                temp[i][j] = chess[i][j];
            }
        }
        return temp;
    }
    //把数组转为list
    private List<String> construct(char[][] chess) {
        List<String> path = new ArrayList<>();
        for (int i = 0; i < chess.length; i++) {
            path.add(new String(chess[i]));
        }
        return path;

    }
```

```
}
```

解法二

Java代码

```
class Solution {
    public List<Integer> nums;
    public int[] sums;// 用来存储正方形四条边可能的边长
    public int possibleSquareSide;
    public Solution() { this.sums = new int[4];}
    public boolean makesquare(int[] nums) {
        if (nums == null || nums.length < 4) {
            return false;// 如果火柴棍数目小于4 则必不能组成正方形
        }
        int L = nums.length;// 计算数组总和
        int sum = 0;
        for(int i = 0; i < L; i++) { sum += nums[i]; }
        if(sum%4 != 0){// 优化一
            return false;
        }
        this.possibleSquareSide =  sum / 4;// 计算正方形的边长
        this.nums = Arrays.stream(nums).boxed().collect(Collectors.toList());
        Collections.sort(this.nums, Collections.reverseOrder());// 优化二
        return this.dfs(0);// 进行深度优先搜索，遍历所有可能的组合，是否存在某一种组合四条边长相等
    }
  public boolean dfs(int index) {
        // 递归出口，如果所有的火柴棍都已经分配到四个同桶里了，检查此时四个桶里的总长度是否一致
        if (index == this.nums.size()) {
            return sums[0] == sums[1] && sums[1] == sums[2] && sums[2] == sums[3];
        }
        int element = this.nums.get(index);// 获得当前要进行分配的火柴棍
        // 将这个火柴棍分别分配到四个桶里
        for(int i = 0; i < 4; i++) {
                if (this.sums[i] + element <= this.possibleSquareSide) {// 优化一
                this.sums[i] += element;// 每个桶只记录已经分配到的火柴棍的总长度即可
                if (this.dfs(index + 1)) {// 继续深搜分配剩余的火柴棍
                    return true;
                }
                this.sums[i] -= element;//注意复原修改过的状态
```

```
            }
        }
        return false;
    }
}
```

最优解法

Java代码

```
class Solution {
    public List<List<String>> solveNQueens(int n) {
        List<List<String>> solutions = new ArrayList<List<String>>();// 用来记录结果的数组
        int[] queens = new int[n];// (i,queens[i])放置皇后
        Arrays.fill(queens, -1);// 初始化为-1
        Set<Integer> columns = new HashSet<Integer>();// columns[i]表示第i列上以及放置了皇后
        Set<Integer> diagonals1 = new HashSet<Integer>();// 表示左上到右下的斜线，同一斜线上行列坐标之差相等
        Set<Integer> diagonals2 = new HashSet<Integer>();// 表示右上到左下的斜线，同一斜线上行列坐标之和相等
        backtrack(solutions, queens, n, 0, columns, diagonals1, diagonals2);// 开始递归
        return solutions;
    }
    //生成最终的结果
    public List<String> generateBoard(int[] queens, int n) {
        List<String> board = new ArrayList<String>();
        for (int i = 0; i < n; i++) {
            char[] row = new char[n];
            Arrays.fill(row, '.');
            row[queens[i]] = 'Q';
            board.add(new String(row));
        }
        return board;
    }
  // n表示是n皇后问题；row 表示当前将要填充的行号
    public void backtrack(List<List<String>> solutions, int[] queens, int n, int row, Set<Integer> columns, Set<Integer> diagonals1, Set<Integer> diagonals2) {
        if (row == n) {// 递归出口
            List<String> board = generateBoard(queens, n);
            solutions.add(board);
        } else {
```

```
            for (int i = 0; i < n; i++) {// 对于当前的第row行，试图去填充每一列
                if (columns.contains(i)) {continue; } // 如果当前列上已经有皇后了，
退出此次循环，填充下一列
                int diagonal1 = row - i;// 使用行列坐标只差来表示方向一(左上到右下)的
斜线
                if (diagonals1.contains(diagonal1)) { continue; } // 如果方向一已
经有皇后了，退出此次循环，填充下一列
                int diagonal2 = row + i;// 使用行列坐标只和来表示方向二(右上到左下)的
斜线
                if (diagonals2.contains(diagonal2)) { continue; } // 如果方向二已
经有皇后了，退出此次循环，填充下一列
                // 如果当前位置(row,i)可以放置皇后，更新状态
                queens[row] = i; columns.add(i); diagonals1.add(diagonal1);
diagonals2.add(diagonal2);
                //继续遍历下一行
                backtrack(solutions, queens, n, row + 1, columns, diagonals1, d
iagonals2);
                // 上面的递归完成后，在(row,i)放置皇后对的前提下所有可能已经记录在
solutions里面了，复原状态
                queens[row] = -1; columns.remove(i);
diagonals1.remove(diagonal1); diagonals2.remove(diagonal2);
            }
        }
    }
}
```

C++代码

```
class Solution {
public:
    vector<vector<string>> solveNQueens(int n) {
        auto solutions = vector<vector<string>>();
        auto queens = vector<int>(n, -1);
        auto columns = unordered_set<int>();
        auto diagonals1 = unordered_set<int>();
        auto diagonals2 = unordered_set<int>();
        backtrack(solutions, queens, n, 0, columns, diagonals1, diagonals2);
        return solutions;
    }

    void backtrack(vector<vector<string>> &solutions, vector<int> &queens, int
n, int row, unordered_set<int> &columns, unordered_set<int> &diagonals1,
unordered_set<int> &diagonals2) {
        if (row == n) {
            vector<string> board = generateBoard(queens, n);
            solutions.push_back(board);
```

```
        } else {
            for (int i = 0; i < n; i++) {
                if (columns.find(i) != columns.end()) {
                    continue;
                }
                int diagonal1 = row - i;
                if (diagonals1.find(diagonal1) != diagonals1.end()) {
                    continue;
                }
                int diagonal2 = row + i;
                if (diagonals2.find(diagonal2) != diagonals2.end()) {
                    continue;
                }
                queens[row] = i;
                columns.insert(i);
                diagonals1.insert(diagonal1);
                diagonals2.insert(diagonal2);
                backtrack(solutions, queens, n, row + 1, columns, diagonals1,
diagonals2);
                queens[row] = -1;
                columns.erase(i);
                diagonals1.erase(diagonal1);
                diagonals2.erase(diagonal2);
            }
        }
    }

    vector<string> generateBoard(vector<int> &queens, int n) {
        auto board = vector<string>();
        for (int i = 0; i < n; i++) {
            string row = string(n, '.');
            row[queens[i]] = 'Q';
            board.push_back(row);
        }
        return board;
    }
};
```

Python代码

```
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        def generateBoard():
            board = list()
            for i in range(n):
                row[queens[i]] = "Q"
                board.append("".join(row))
                row[queens[i]] = "."
```

```
            return board

        def backtrack(row: int):
            if row == n:
                board = generateBoard()
                solutions.append(board)
            else:
                for i in range(n):
                    if i in columns or row - i in diagonal1 or row + i in
diagonal2:
                        continue
                    queens[row] = i
                    columns.add(i)
                    diagonal1.add(row - i)
                    diagonal2.add(row + i)
                    backtrack(row + 1)
                    columns.remove(i)
                    diagonal1.remove(row - i)
                    diagonal2.remove(row + i)

        solutions = list()
        queens = [-1] * n
        columns = set()
        diagonal1 = set()
        diagonal2 = set()
        row = ["."] * n
        backtrack(0)
        return solutions
```