DFS 经典题精讲

主讲人 令狐冲

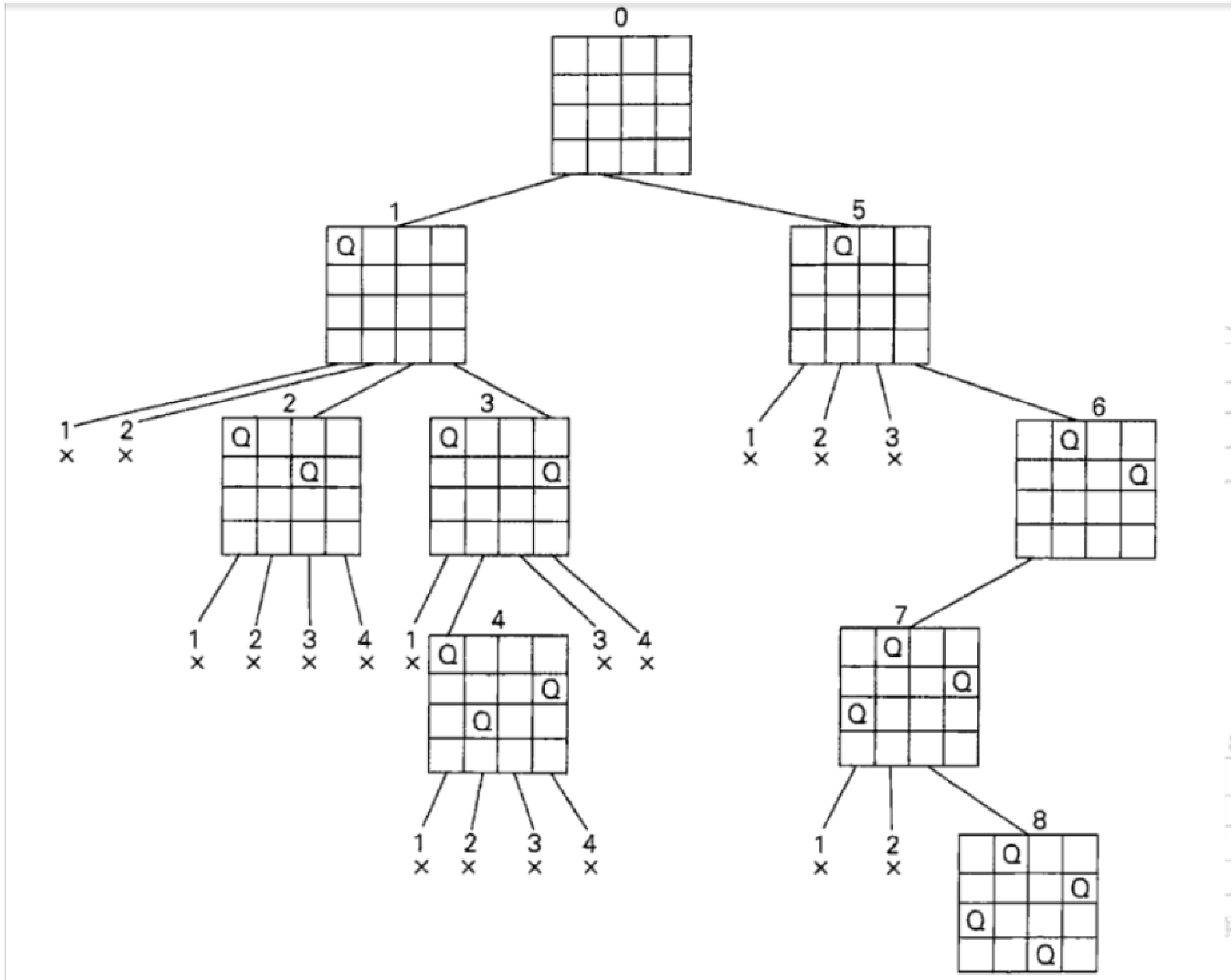
N Queens

http://www.lintcode.com/problem/n-queens/

http://www.jiuzhang.com/solutions/n-queens/

另一种问法：问方案总数（N Queens II）

搜索树



程序结构的艺术

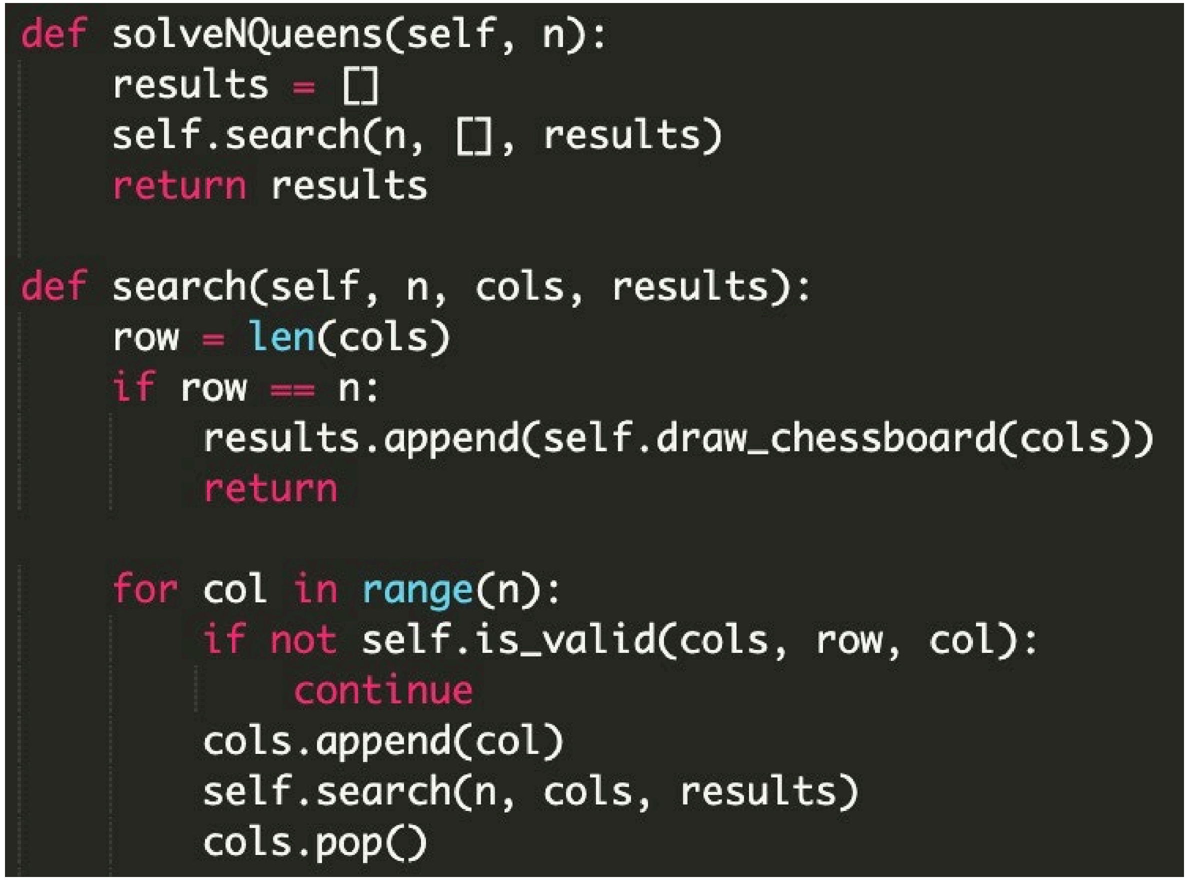
入口函数

搜索函数

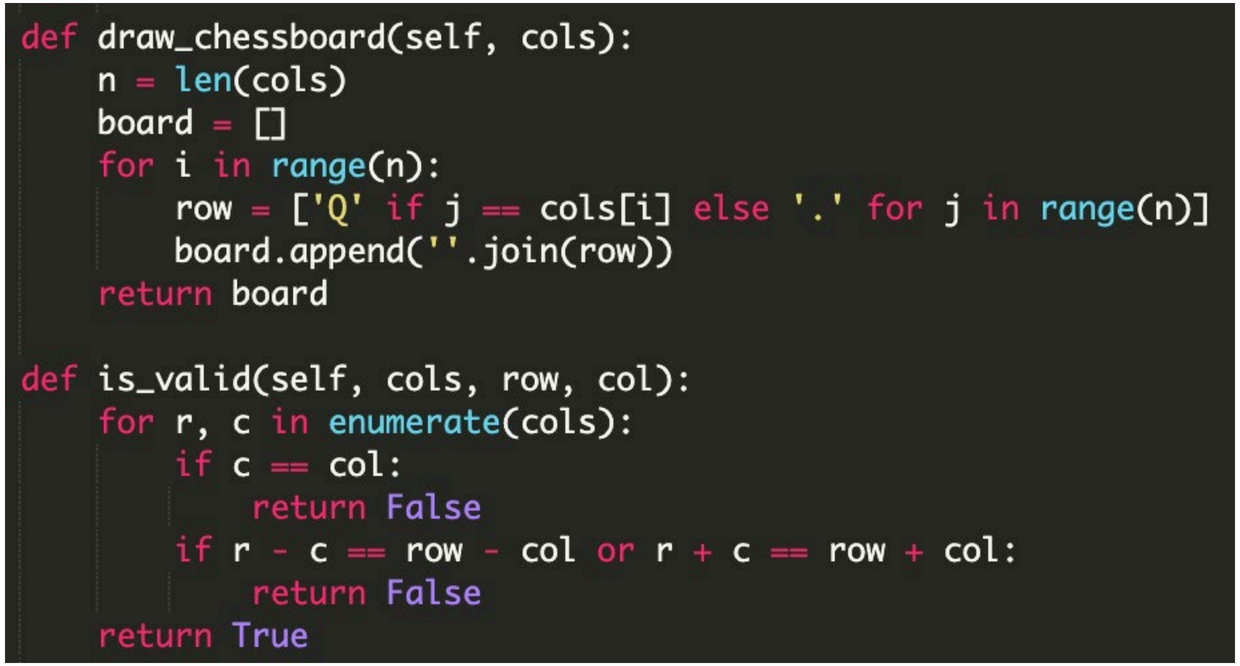
判断函数

打印函数

Python 代码实现

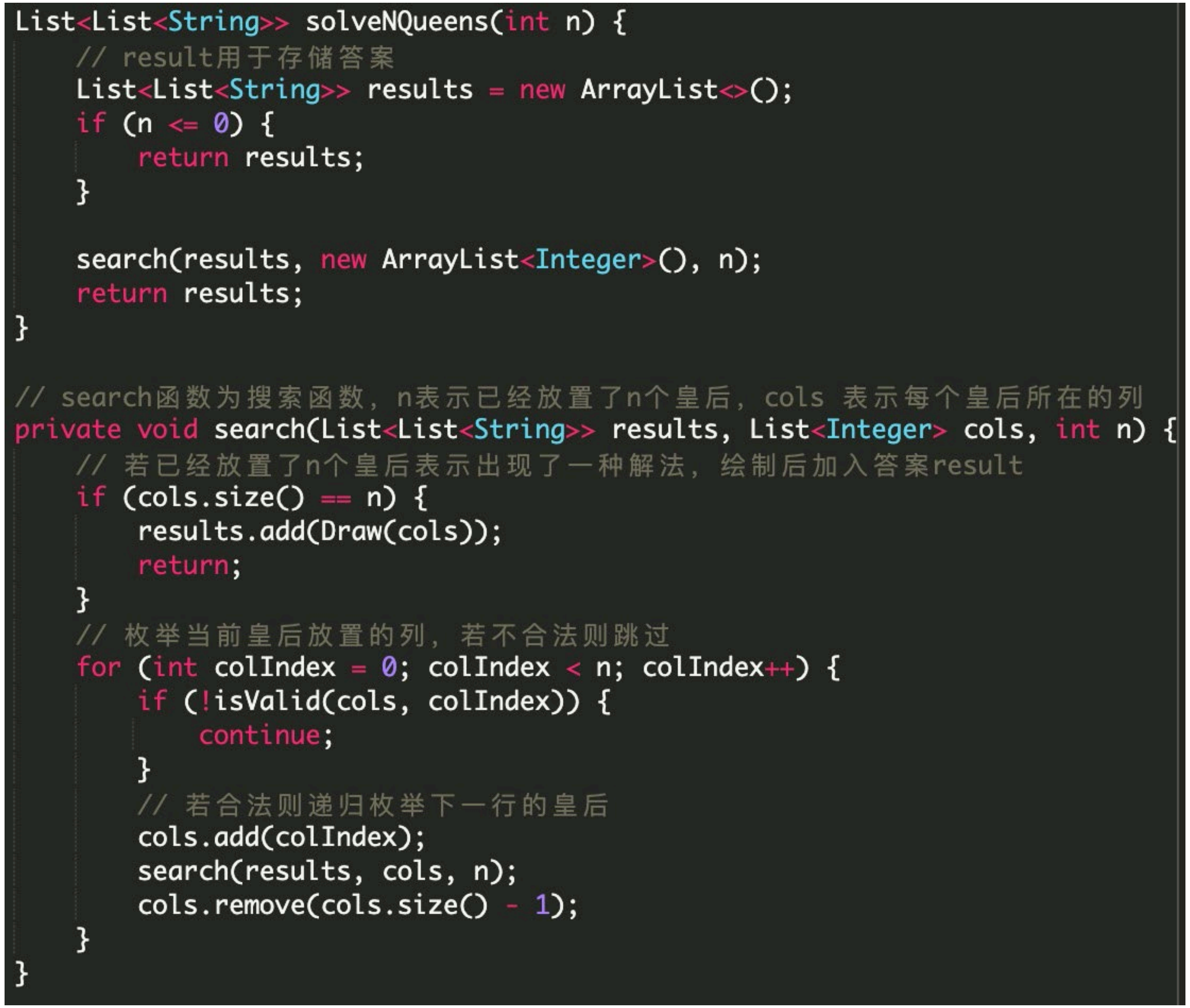


```python
def solveNQueens(self, n):
    results = []
    self.search(n, [], results)
    return results

def search(self, n, cols, results):
    row = len(cols)
    if row == n:
        results.append(self.draw_chessboard(cols))
        return

    for col in range(n):
        if not self.is_valid(cols, row, col):
            continue
        cols.append(col)
        self.search(n, cols, results)
        cols.pop()
```

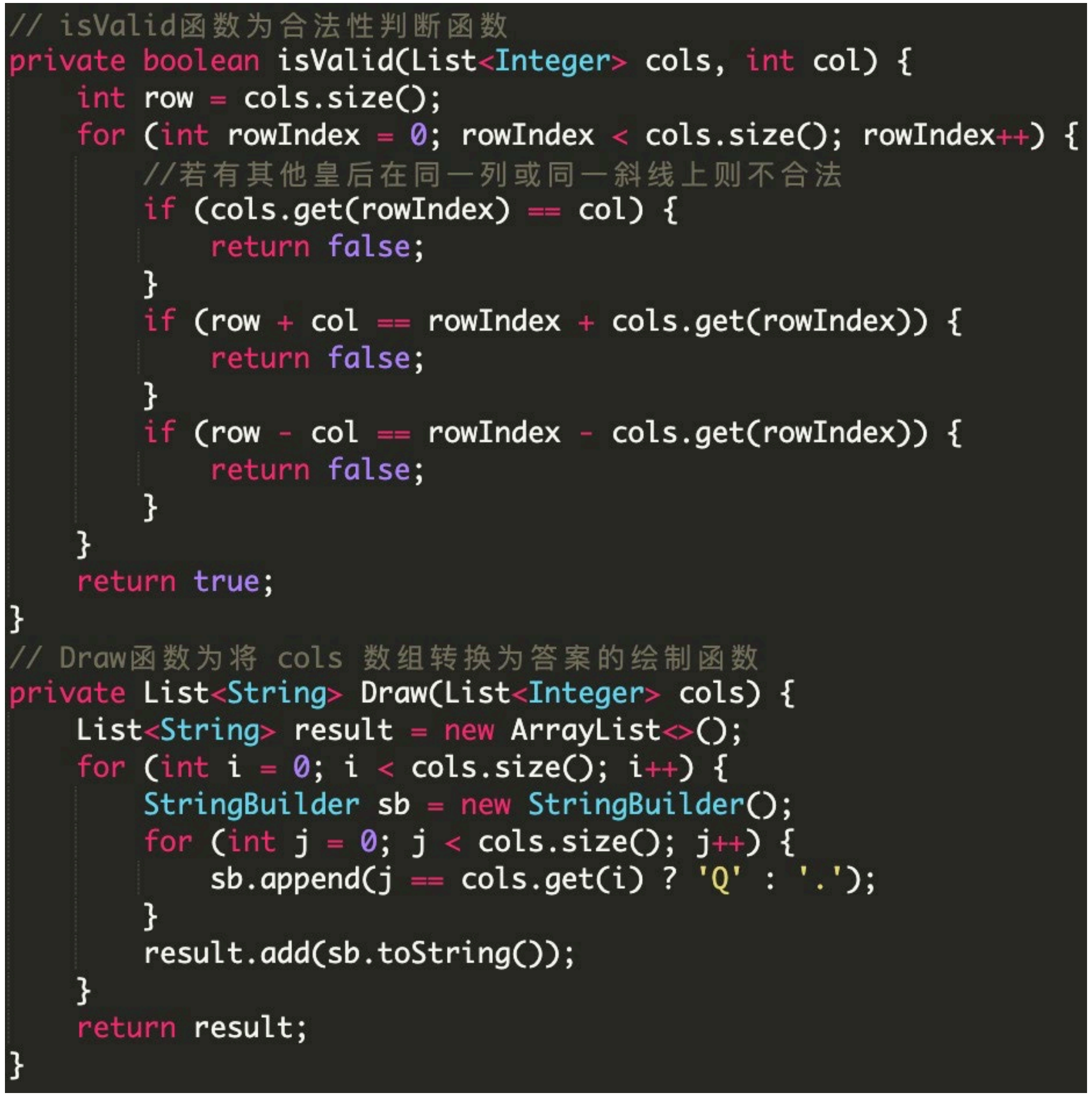


```python
def draw_chessboard(self, cols):
    n = len(cols)
    board = []
    for i in range(n):
        row = ['Q' if j == cols[i] else '.' for j in range(n)]
        board.append(''.join(row))
    return board

def is_valid(self, cols, row, col):
    for r, c in enumerate(cols):
        if c == col:
            return False
        if r - c == row - col or r + c == row + col:
            return False
    return True
```

Java 代码实现



```java
List<List<String>> solveNQueens(int n) {
    // result用于存储答案
    List<List<String>> results = new ArrayList<>();
    if (n <= 0) {
        return results;
    }

    search(results, new ArrayList<Integer>(), n);
    return results;
}

// search函数为搜索函数，n表示已经放置了n个皇后，cols 表示每个皇后所在的列
private void search(List<List<String>> results, List<Integer> cols, int n) {
    // 若已经放置了n个皇后表示出现了一种解法，绘制后加入答案result
    if (cols.size() == n) {
        results.add(Draw(cols));
        return;
    }
    // 枚举当前皇后放置的列，若不合法则跳过
    for (int colIndex = 0; colIndex < n; colIndex++) {
        if (!isValid(cols, colIndex)) {
            continue;
        }
        // 若合法则递归枚举下一行的皇后
        cols.add(colIndex);
        search(results, cols, n);
        cols.remove(cols.size() - 1);
    }
}
```



```java
// isValid函数为合法性判断函数
private boolean isValid(List<Integer> cols, int col) {
    int row = cols.size();
    for (int rowIndex = 0; rowIndex < cols.size(); rowIndex++) {
        //若有其他皇后在同一列或同一斜线上则不合法
        if (cols.get(rowIndex) == col) {
            return false;
        }
        if (row + col == rowIndex + cols.get(rowIndex)) {
            return false;
        }
        if (row - col == rowIndex - cols.get(rowIndex)) {
            return false;
        }
    }
    return true;
}
// Draw函数为将 cols 数组转换为答案的绘制函数
private List<String> Draw(List<Integer> cols) {
    List<String> result = new ArrayList<>();
    for (int i = 0; i < cols.size(); i++) {
        StringBuilder sb = new StringBuilder();
        for (int j = 0; j < cols.size(); j++) {
            sb.append(j == cols.get(i) ? 'Q' : '.');
        }
        result.add(sb.toString());
    }
    return result;
}
```

时间复杂度

O(方案总数 * 构造每个方案的时间) = O(S * N^2)

S 为 N 皇后的方案数

N^2 是画棋盘的时间

如何优化?

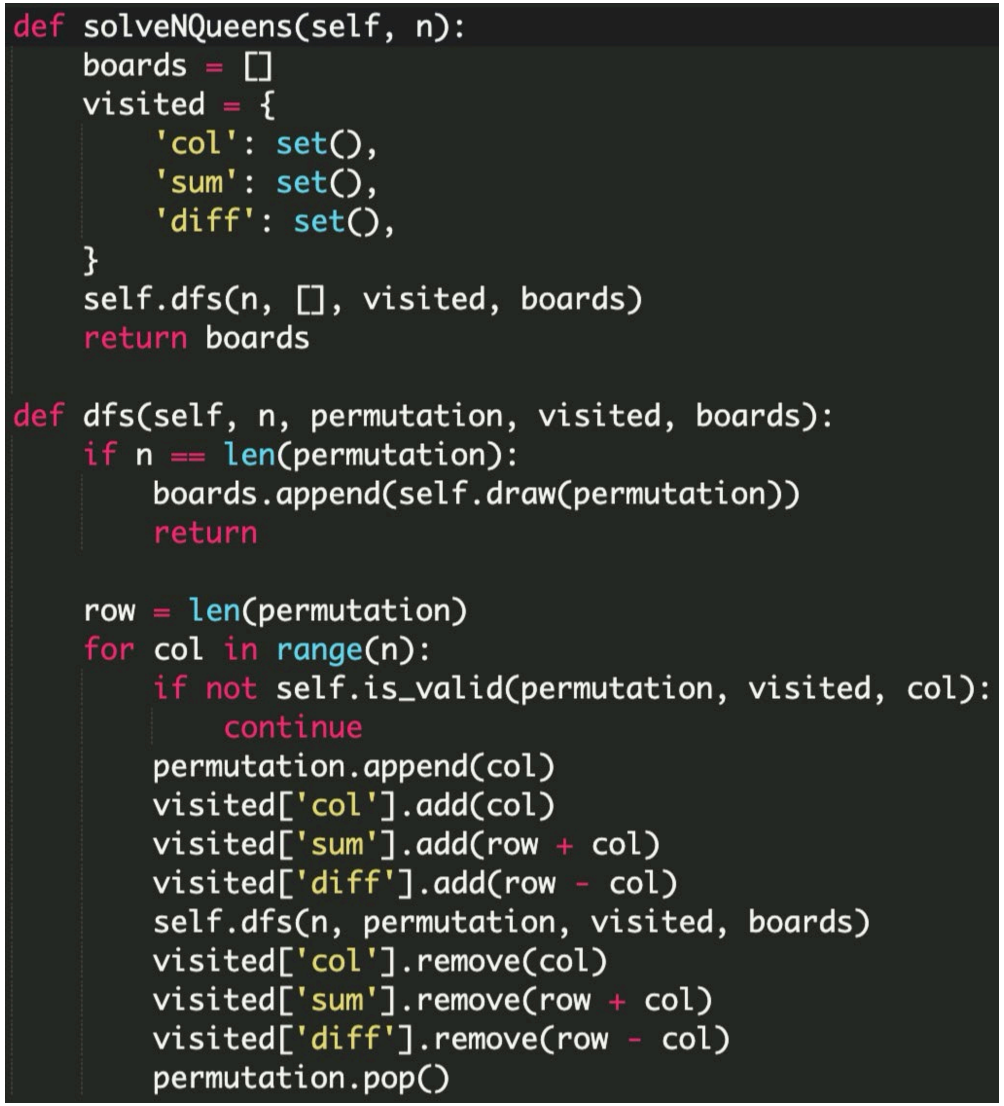
整个程序有没有哪里比较慢可以优化的?

isValid 函数可以优化

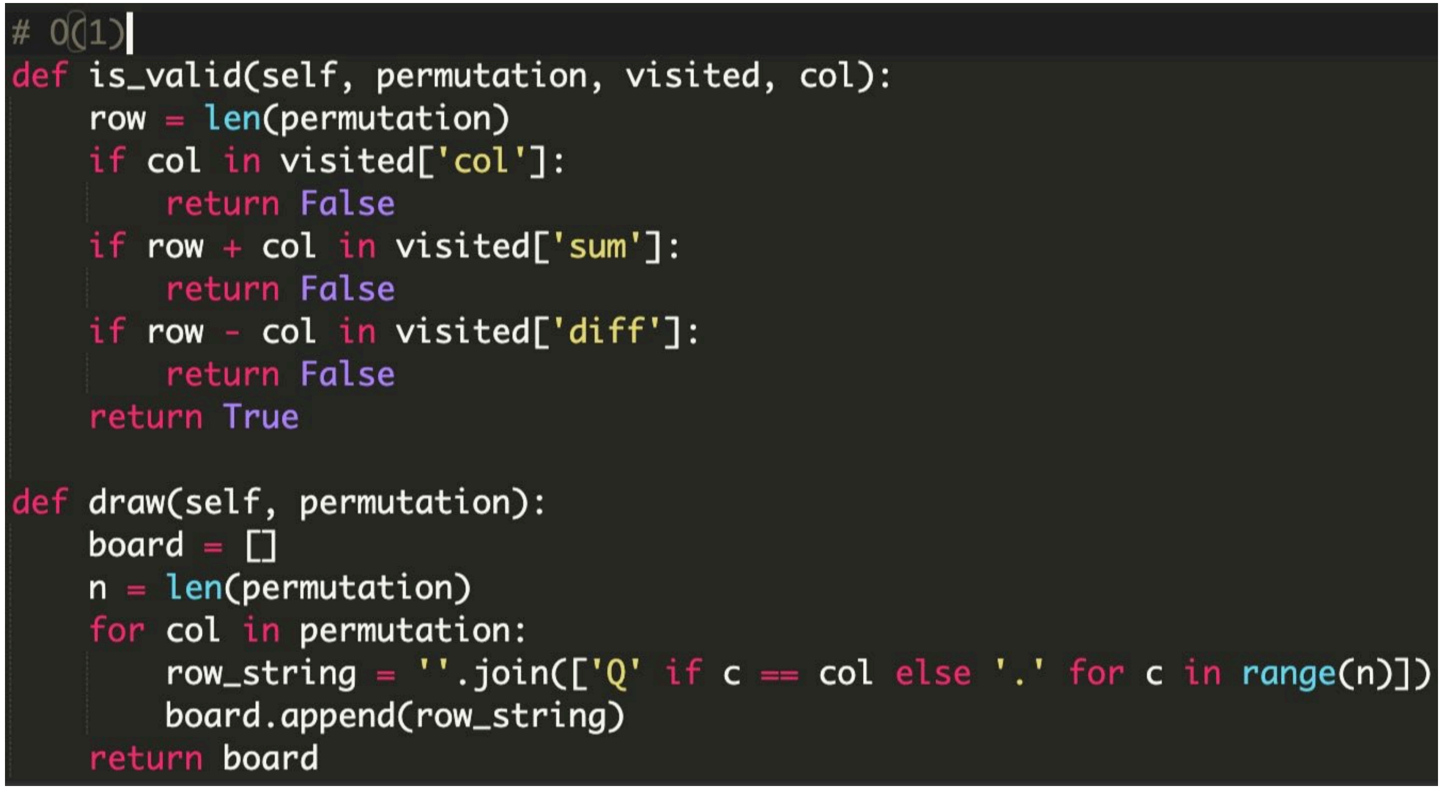
O(N) -> O(1)

通过哈希表记录哪些列、斜对角线已经被占

Python 优化之后的版本

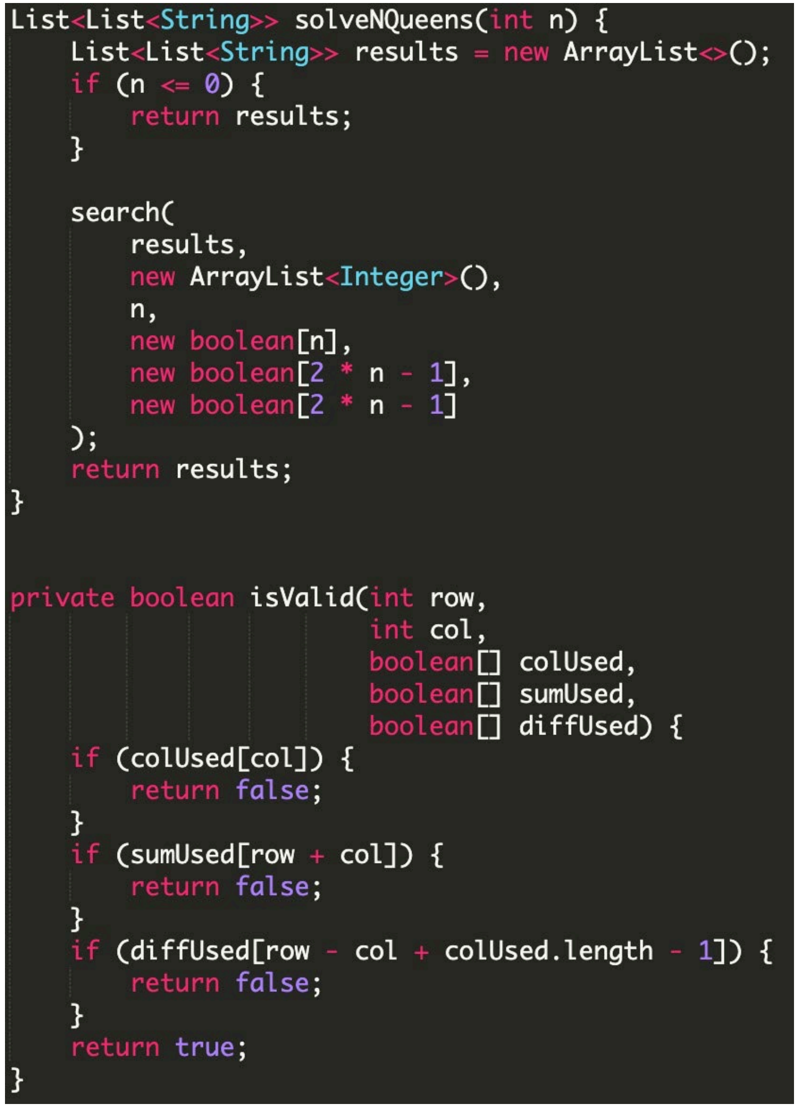


```python
def solveNQueens(self, n):
    boards = []
    visited = {
        'col': set(),
        'sum': set(),
        'diff': set(),
    }
    self.dfs(n, [], visited, boards)
    return boards

def dfs(self, n, permutation, visited, boards):
    if n == len(permutation):
        boards.append(self.draw(permutation))
        return

    row = len(permutation)
    for col in range(n):
        if not self.is_valid(permutation, visited, col):
            continue
        permutation.append(col)
        visited['col'].add(col)
        visited['sum'].add(row + col)
        visited['diff'].add(row - col)
        self.dfs(n, permutation, visited, boards)
        visited['col'].remove(col)
        visited['sum'].remove(row + col)
        visited['diff'].remove(row - col)
        permutation.pop()
```

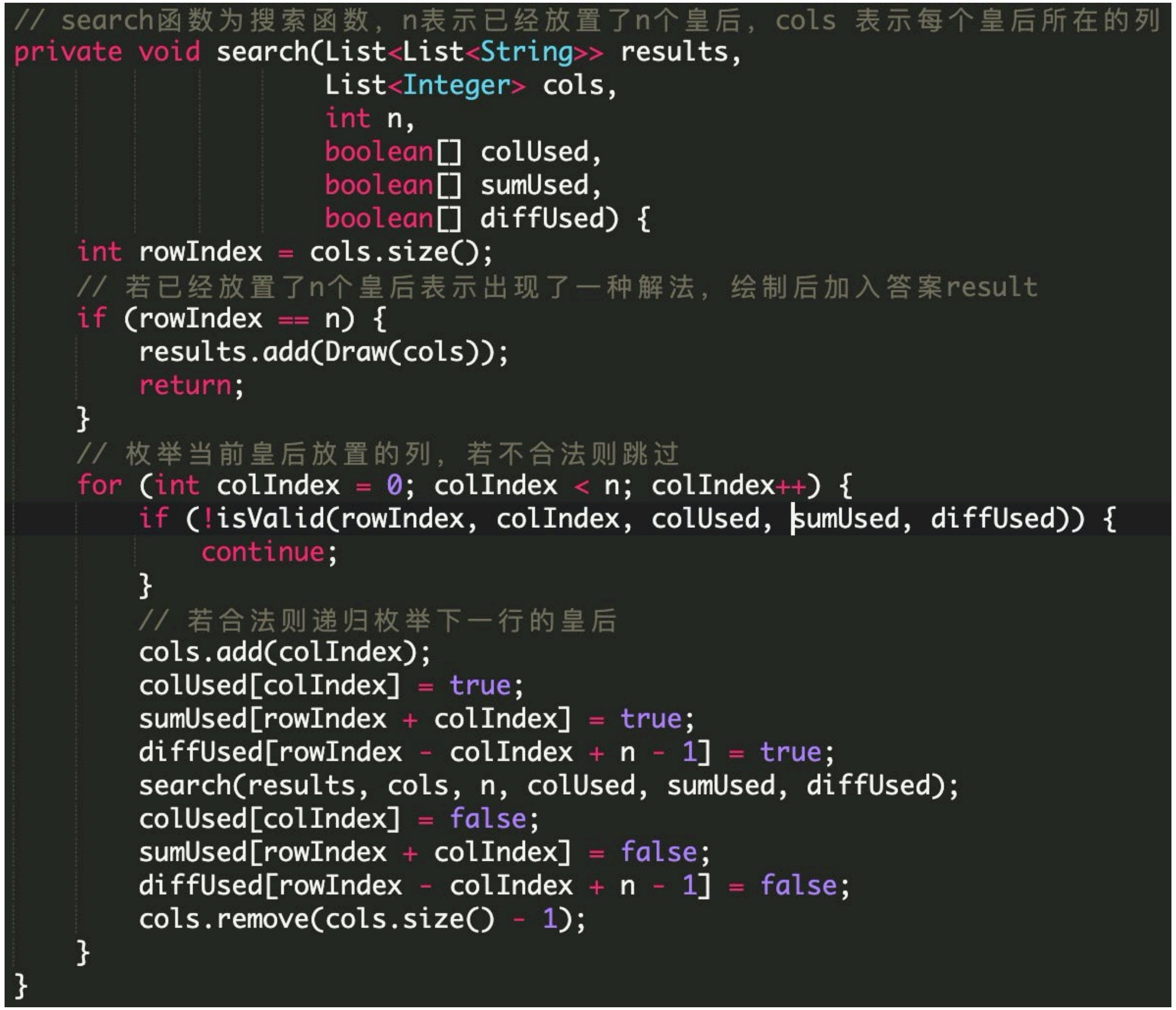


```python
# O(1)
def is_valid(self, permutation, visited, col):
    row = len(permutation)
    if col in visited['col']:
        return False
    if row + col in visited['sum']:
        return False
    if row - col in visited['diff']:
        return False
    return True

def draw(self, permutation):
    board = []
    n = len(permutation)
    for col in permutation:
        row_string = ''.join(['Q' if c == col else '.' for c in range(n)])
        board.append(row_string)
    return board
```

Java 优化之后的版本



```
List<List<String>> solveNQueens(int n) {
    List<List<String>> results = new ArrayList<>();
    if (n <= 0) {
        return results;
    }

    search(
        results,
        new ArrayList<Integer>(),
        n,
        new boolean[n],
        new boolean[2 * n - 1],
        new boolean[2 * n - 1]
    );
    return results;
}


private boolean isValid(int row,
                        int col,
                        boolean[] colUsed,
                        boolean[] sumUsed,
                        boolean[] diffUsed) {
    if (colUsed[col]) {
        return false;
    }
    if (sumUsed[row + col]) {
        return false;
    }
    if (diffUsed[row - col + colUsed.length - 1]) {
        return false;
    }
    return true;
}
```



```
// search函数为搜索函数，n表示已经放置了n个皇后，cols 表示每个皇后所在的列
private void search(List<List<String>> results,
                    List<Integer> cols,
                    int n,
                    boolean[] colUsed,
                    boolean[] sumUsed,
                    boolean[] diffUsed) {
    int rowIndex = cols.size();
    // 若已经放置了n个皇后表示出现了一种解法，绘制后加入答案result
    if (rowIndex == n) {
        results.add(Draw(cols));
        return;
    }
    // 枚举当前皇后放置的列，若不合法则跳过
    for (int colIndex = 0; colIndex < n; colIndex++) {
        if (!isValid(rowIndex, colIndex, colUsed, sumUsed, diffUsed)) {
            continue;
        }
        // 若合法则递归枚举下一行的皇后
        cols.add(colIndex);
        colUsed[colIndex] = true;
        sumUsed[rowIndex + colIndex] = true;
        diffUsed[rowIndex - colIndex + n - 1] = true;
        search(results, cols, n, colUsed, sumUsed, diffUsed);
        colUsed[colIndex] = false;
        sumUsed[rowIndex + colIndex] = false;
        diffUsed[rowIndex - colIndex + n - 1] = false;
        cols.remove(cols.size() - 1);
    }
}
```

优化效果

这样的优化有效果么?
如果有，时间复杂度会变成多少?

没有效果

DFS 的递归实现相当于实现了一个 N 重循环

N 重循环的时间复杂度取决于最内存层循环体的执行次数

这个优化并不会减少最内层主体的循环次数

瓶颈依然是递归出口位置 O(N^2) 的打印函数

数独

https://www.lintcode.com/problem/sudoku-solver/

https://www.jiuzhang.com/problem/sudoku-solver/

数独是典型的 DFS 题

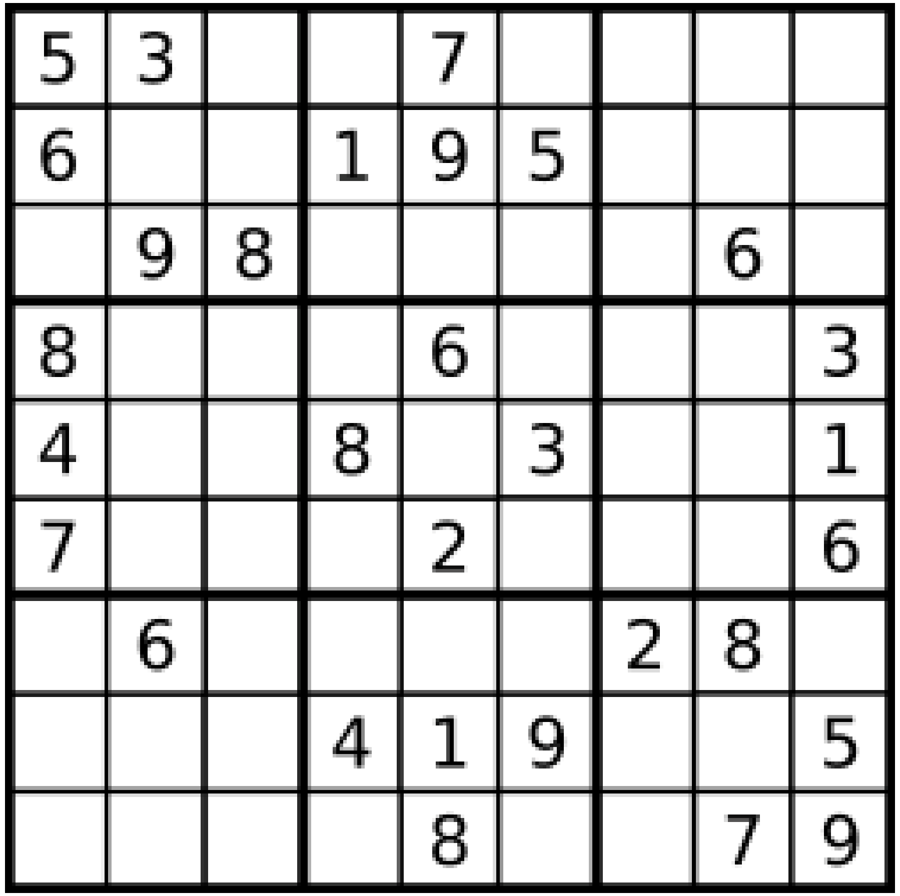
Naive DFS 的大致思路

从上到下从左到右找到每个空格

尝试把 1-9 的数字放进去，判断是否合法

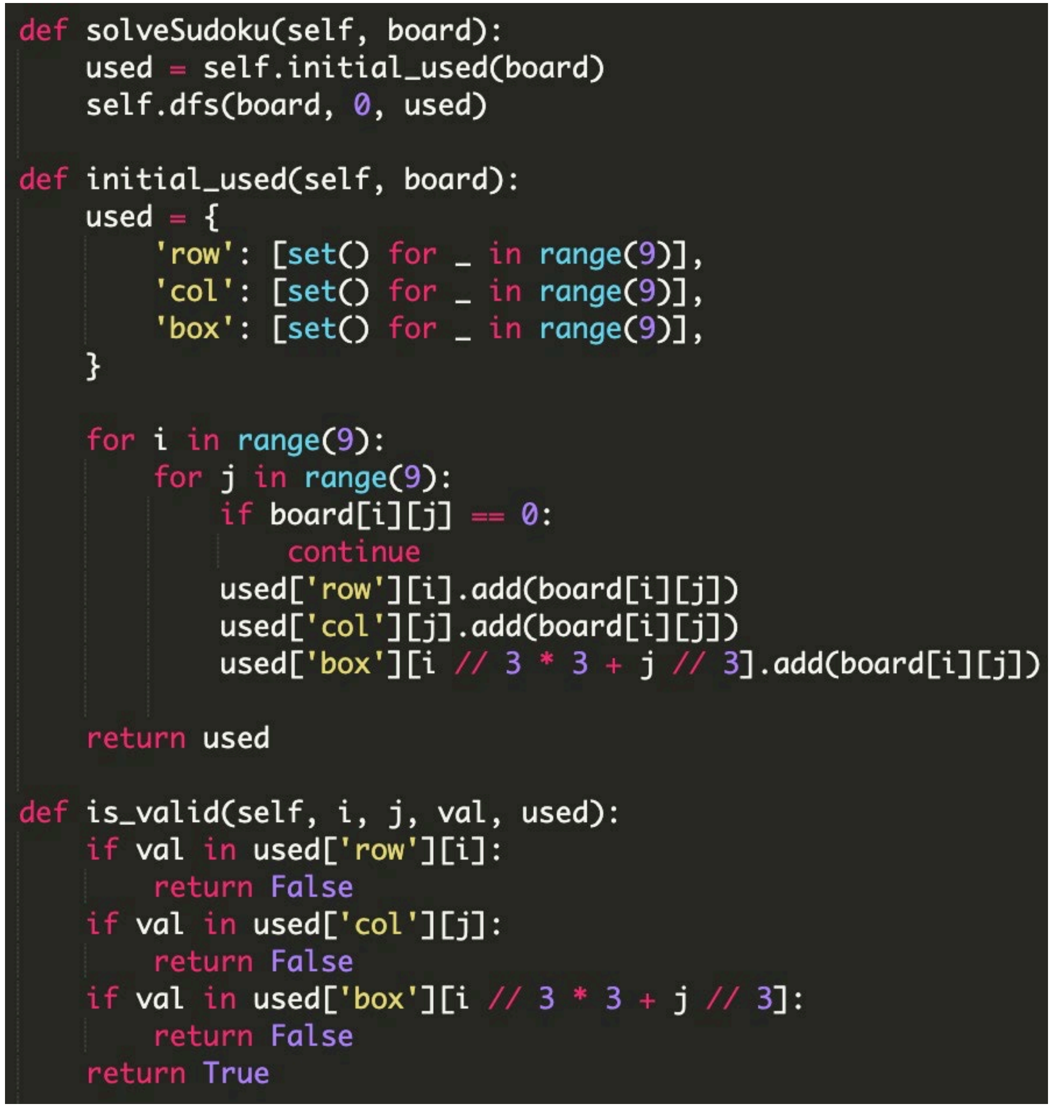
如果合法，重复上述步骤继续寻找下一个空格

直到把所有位置都填满

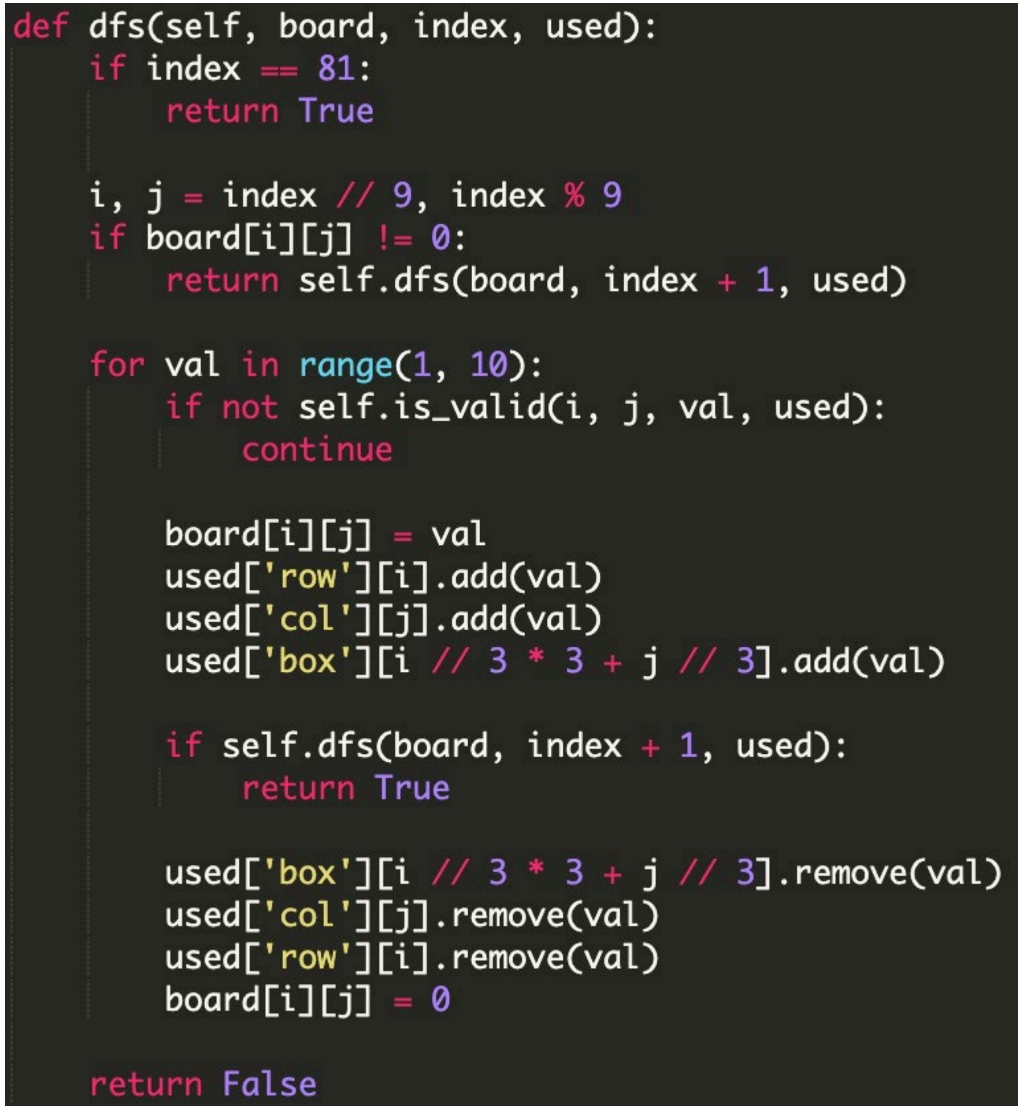


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Python 代码

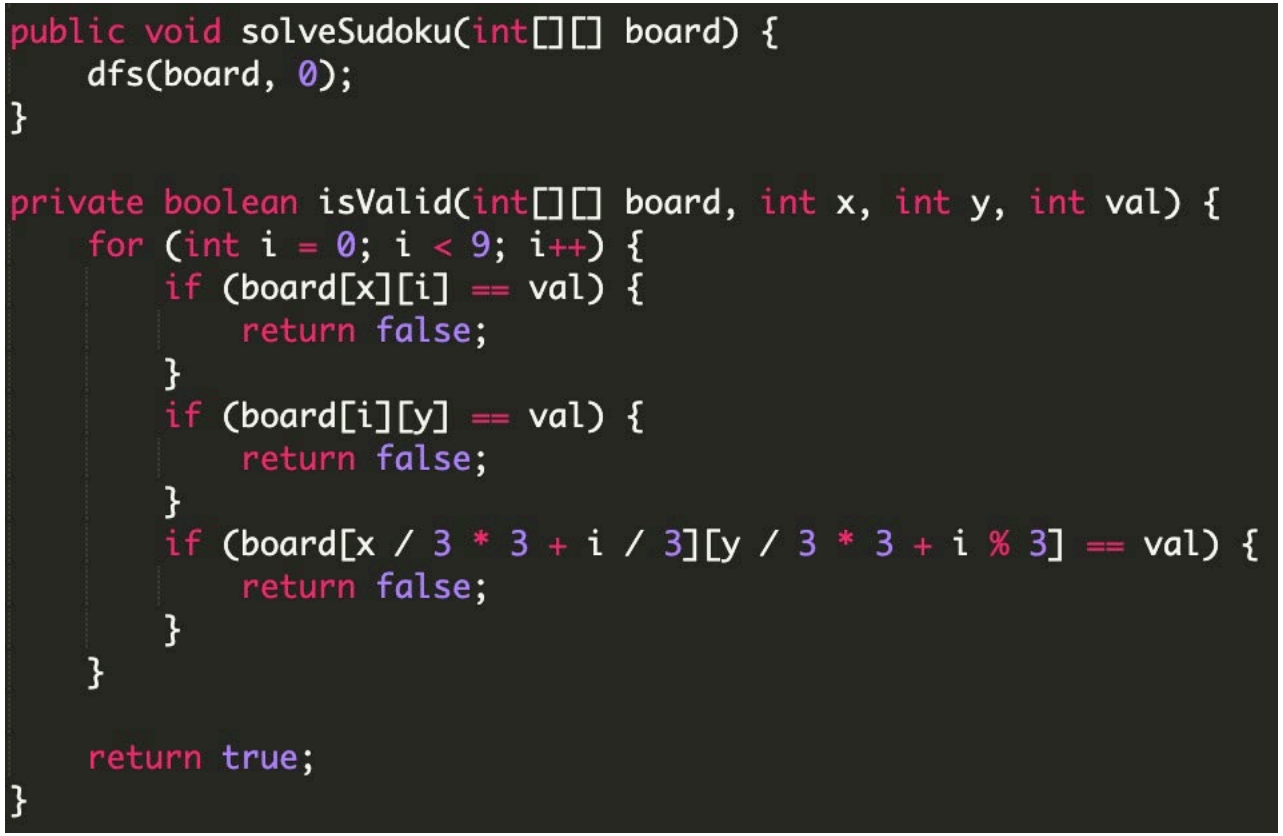


```python
def solveSudoku(self, board):
    used = self.initial_used(board)
    self.dfs(board, 0, used)

def initial_used(self, board):
    used = {
        'row': [set() for _ in range(9)],
        'col': [set() for _ in range(9)],
        'box': [set() for _ in range(9)],
    }

    for i in range(9):
        for j in range(9):
            if board[i][j] == 0:
                continue
            used['row'][i].add(board[i][j])
            used['col'][j].add(board[i][j])
            used['box'][i // 3 * 3 + j // 3].add(board[i][j])

    return used

def is_valid(self, i, j, val, used):
    if val in used['row'][i]:
        return False
    if val in used['col'][j]:
        return False
    if val in used['box'][i // 3 * 3 + j // 3]:
        return False
    return True
```

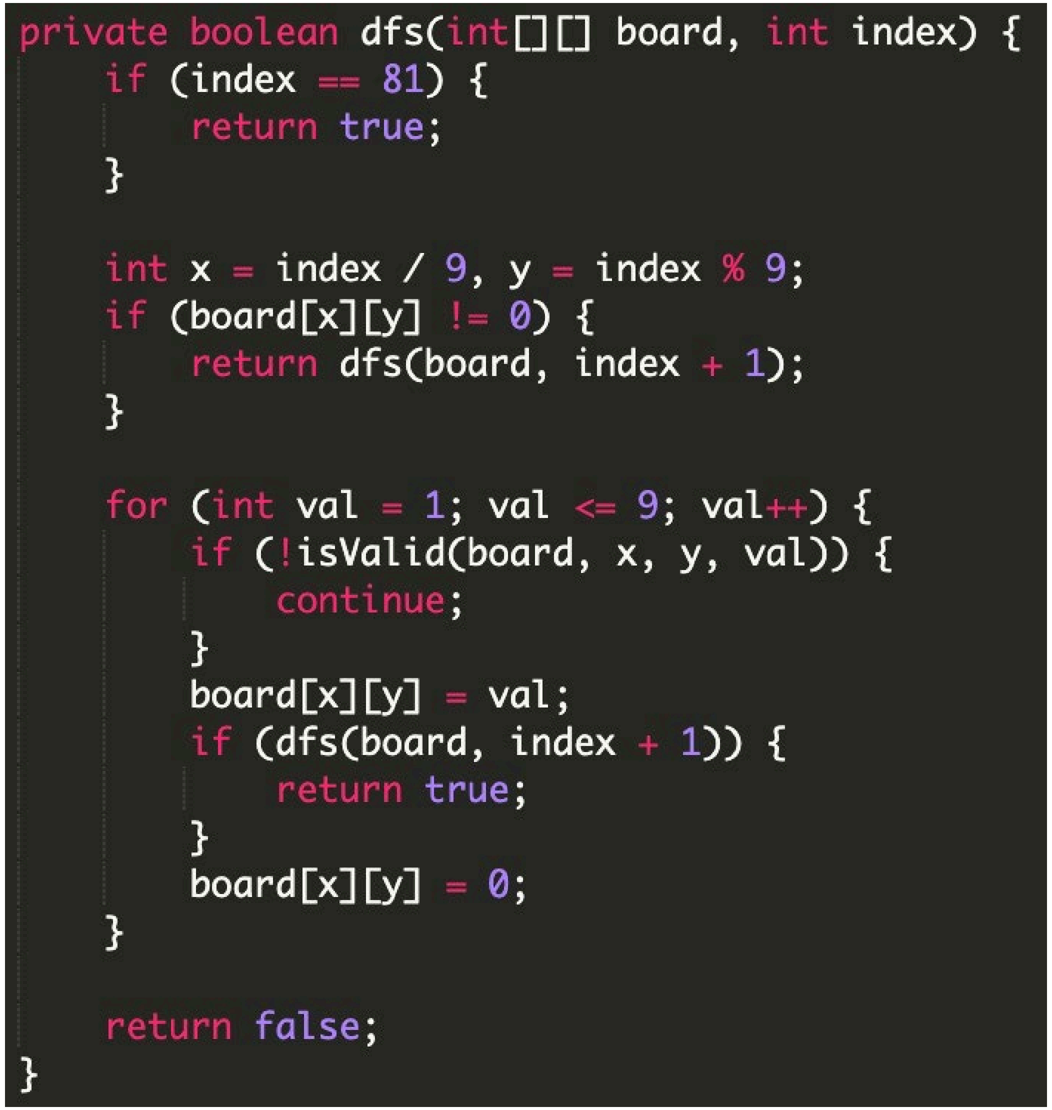


```python
def dfs(self, board, index, used):
    if index == 81:
        return True

    i, j = index // 9, index % 9
    if board[i][j] != 0:
        return self.dfs(board, index + 1, used)

    for val in range(1, 10):
        if not self.is_valid(i, j, val, used):
            continue

        board[i][j] = val
        used['row'][i].add(val)
        used['col'][j].add(val)
        used['box'][i // 3 * 3 + j // 3].add(val)

        if self.dfs(board, index + 1, used):
            return True

        used['box'][i // 3 * 3 + j // 3].remove(val)
        used['col'][j].remove(val)
        used['row'][i].remove(val)
        board[i][j] = 0

    return False
```

Java 代码 - 更简单的一种写法



```java
public void solveSudoku(int[][] board) {
    dfs(board, 0);
}

private boolean isValid(int[][] board, int x, int y, int val) {
    for (int i = 0; i < 9; i++) {
        if (board[x][i] == val) {
            return false;
        }
        if (board[i][y] == val) {
            return false;
        }
        if (board[x / 3 * 3 + i / 3][y / 3 * 3 + i % 3] == val) {
            return false;
        }
    }

    return true;
}
```



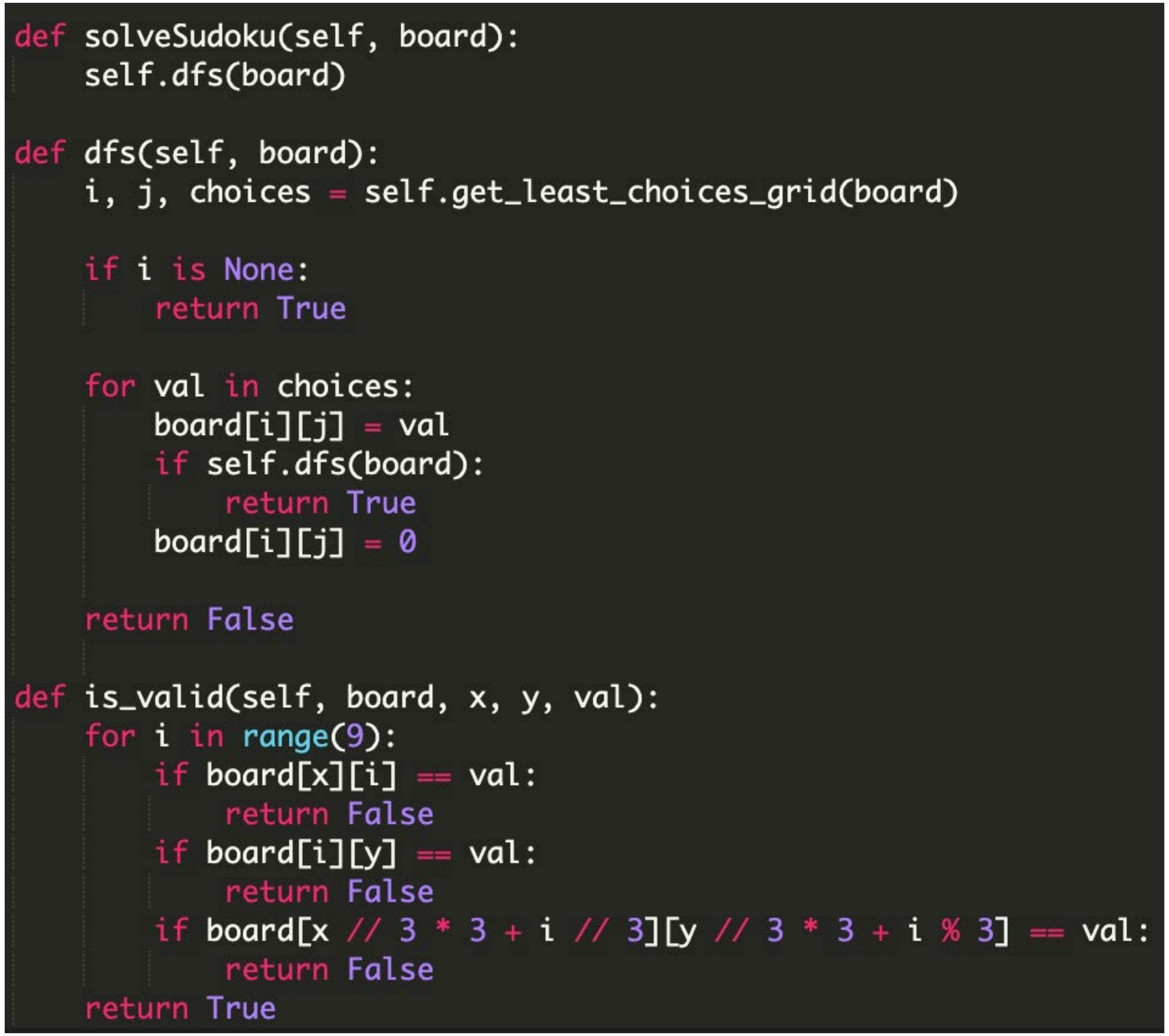
```java
private boolean dfs(int[][] board, int index) {
    if (index == 81) {
        return true;
    }

    int x = index / 9, y = index % 9;
    if (board[x][y] != 0) {
        return dfs(board, index + 1);
    }

    for (int val = 1; val <= 9; val++) {
        if (!isValid(board, x, y, val)) {
            continue;
        }
        board[x][y] = val;
        if (dfs(board, index + 1)) {
            return true;
        }
        board[x][y] = 0;
    }

    return false;
}
```

搜索顺序优化

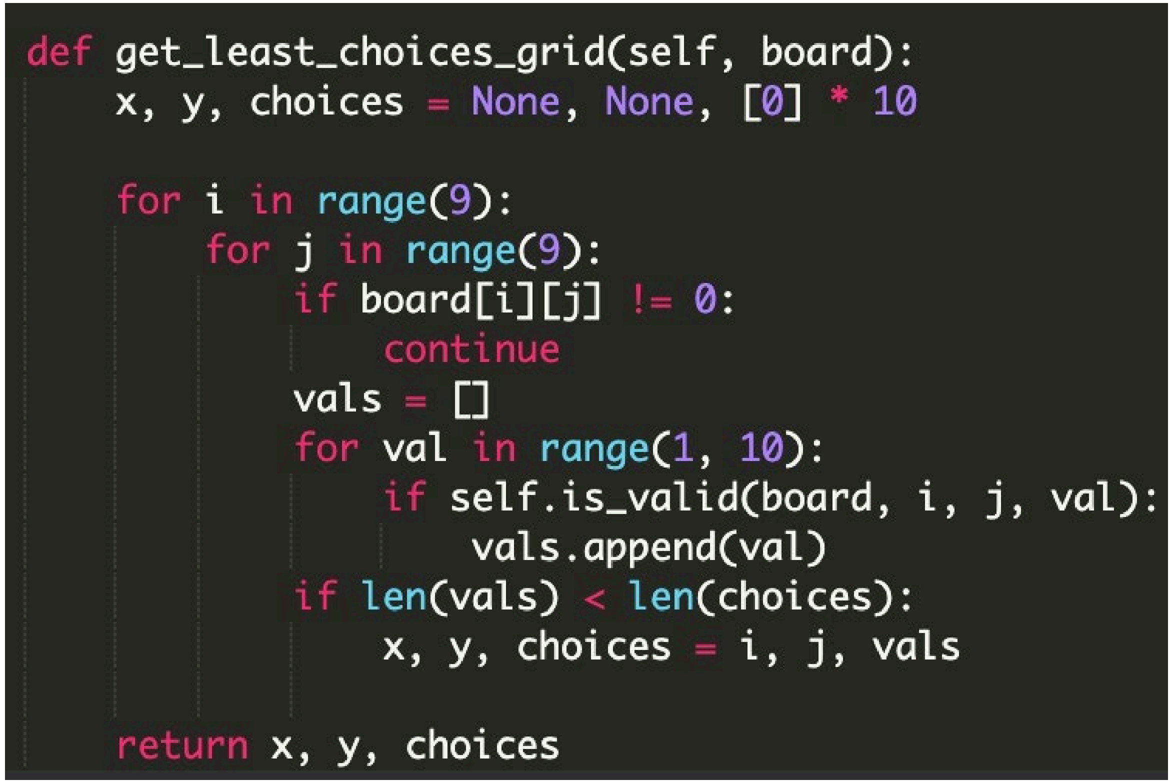
DFS 的常用优化策略之一

优先搜索那些可能方案少的位置

Python 代码

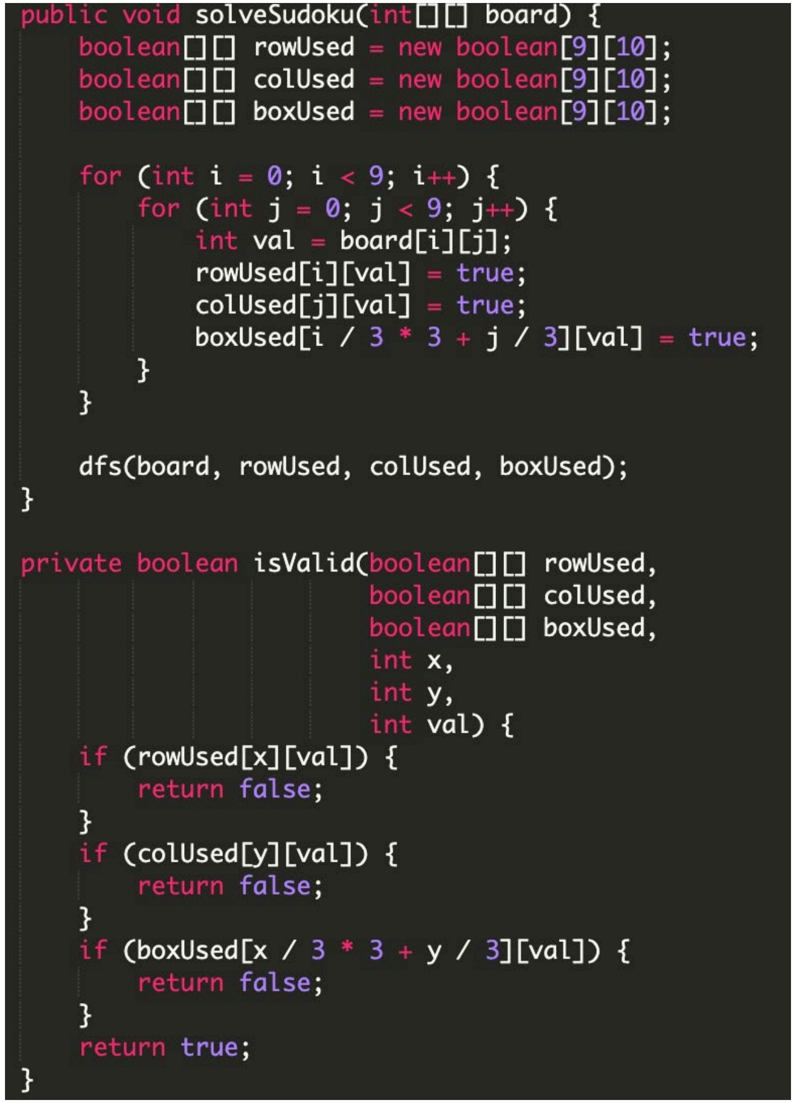


```python
def solveSudoku(self, board):
    self.dfs(board)

def dfs(self, board):
    i, j, choices = self.get_least_choices_grid(board)

    if i is None:
        return True

    for val in choices:
        board[i][j] = val
        if self.dfs(board):
            return True
        board[i][j] = 0

    return False

def is_valid(self, board, x, y, val):
    for i in range(9):
        if board[x][i] == val:
            return False
        if board[i][y] == val:
            return False
        if board[x // 3 * 3 + i // 3][y // 3 * 3 + i % 3] == val:
            return False
    return True
```

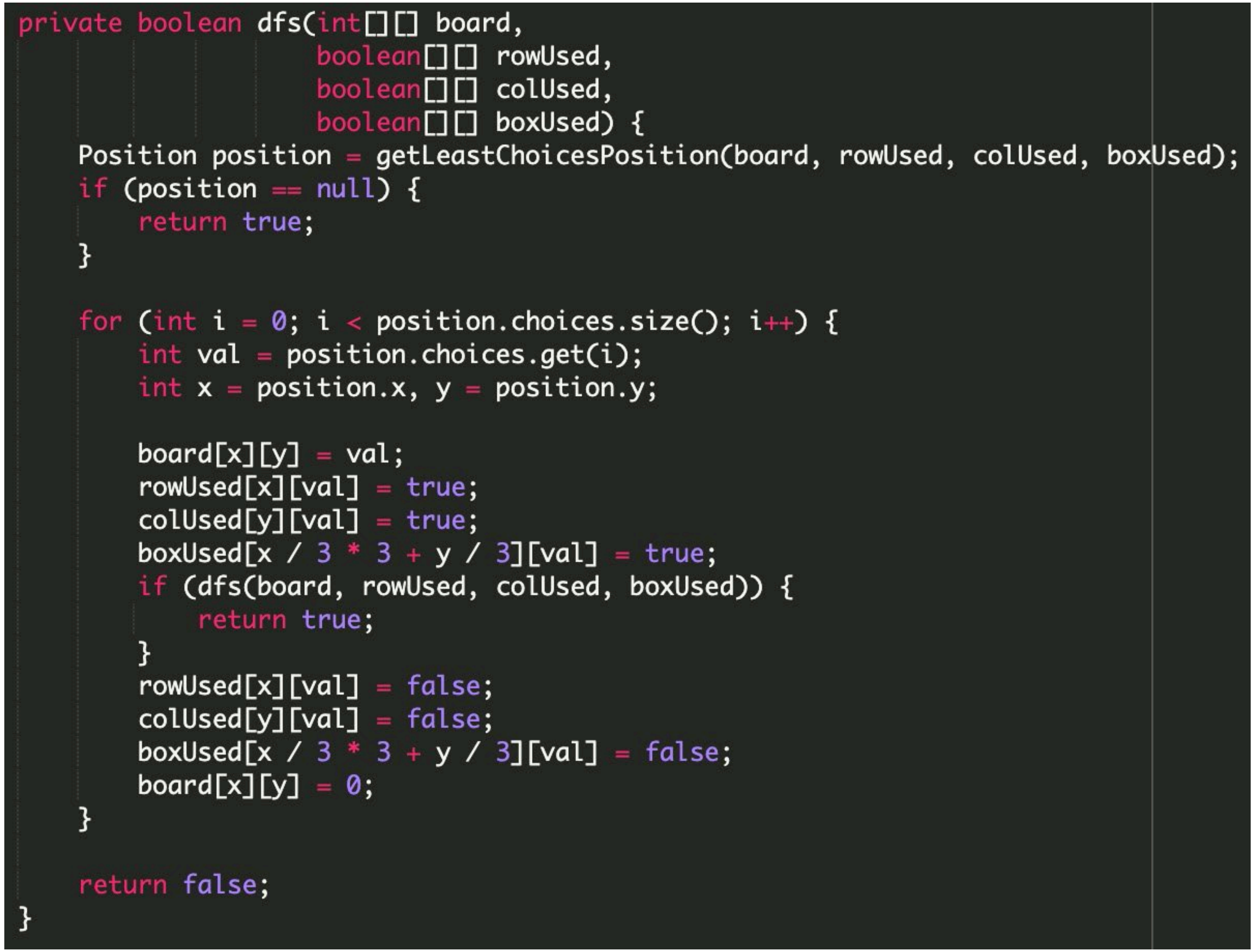


```python
def get_least_choices_grid(self, board):
    x, y, choices = None, None, [0] * 10

    for i in range(9):
        for j in range(9):
            if board[i][j] != 0:
                continue
            vals = []
            for val in range(1, 10):
                if self.is_valid(board, i, j, val):
                    vals.append(val)
            if len(vals) < len(choices):
                x, y, choices = i, j, vals

    return x, y, choices
```

Java 代码

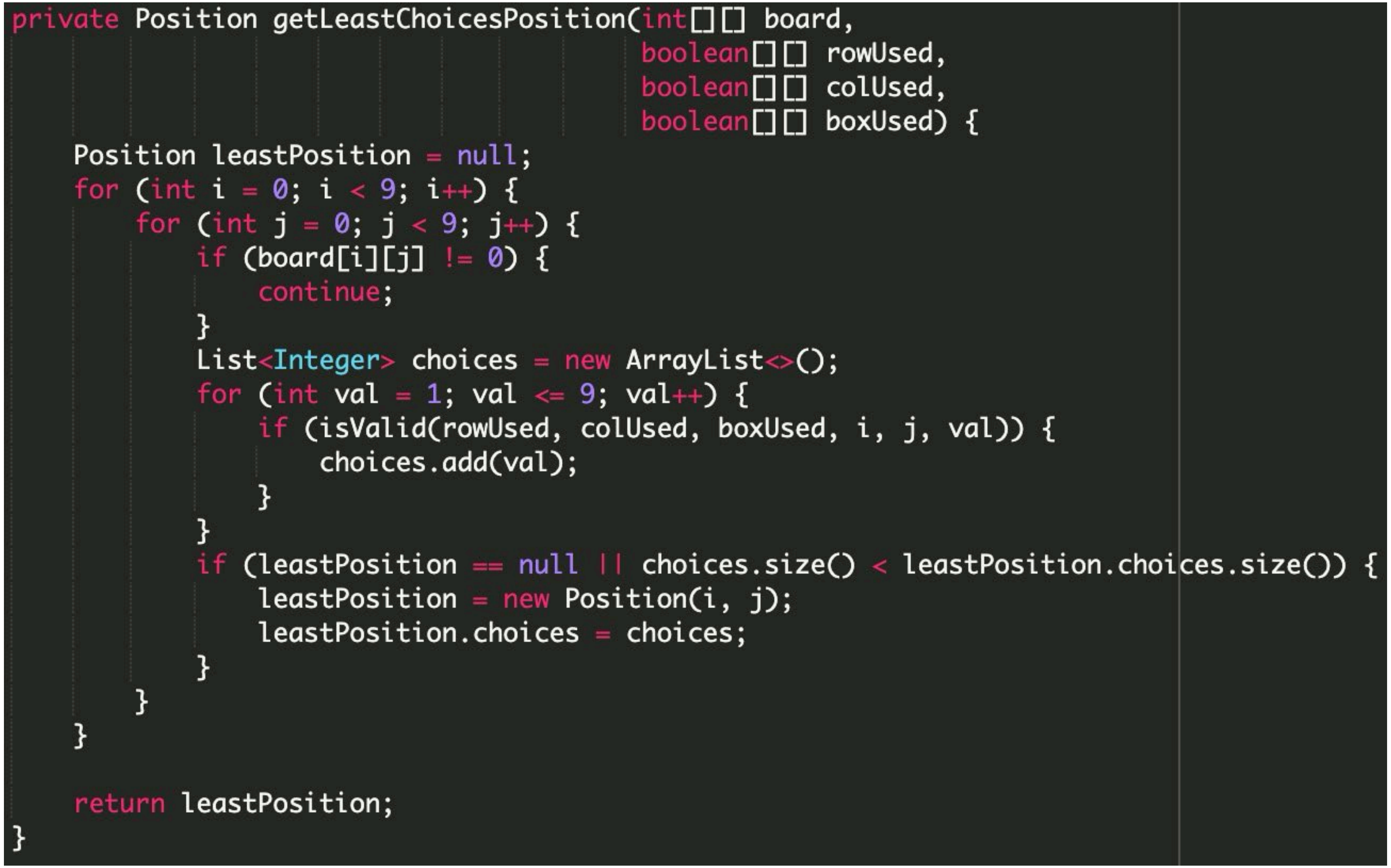


```
public void solveSudoku(int[][] board) {
    boolean[][] rowUsed = new boolean[9][10];
    boolean[][] colUsed = new boolean[9][10];
    boolean[][] boxUsed = new boolean[9][10];

    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            int val = board[i][j];
            rowUsed[i][val] = true;
            colUsed[j][val] = true;
            boxUsed[i / 3 * 3 + j / 3][val] = true;
        }
    }

    dfs(board, rowUsed, colUsed, boxUsed);
}

private boolean isValid(boolean[][] rowUsed,
                        boolean[][] colUsed,
                        boolean[][] boxUsed,
                        int x,
                        int y,
                        int val) {
    if (rowUsed[x][val]) {
        return false;
    }
    if (colUsed[y][val]) {
        return false;
    }
    if (boxUsed[x / 3 * 3 + y / 3][val]) {
        return false;
    }
    return true;
}
```



```
private boolean dfs(int[][] board,
                    boolean[][] rowUsed,
                    boolean[][] colUsed,
                    boolean[][] boxUsed) {
    Position position = getLeastChoicesPosition(board, rowUsed, colUsed, boxUsed);
    if (position == null) {
        return true;
    }

    for (int i = 0; i < position.choices.size(); i++) {
        int val = position.choices.get(i);
        int x = position.x, y = position.y;

        board[x][y] = val;
        rowUsed[x][val] = true;
        colUsed[y][val] = true;
        boxUsed[x / 3 * 3 + y / 3][val] = true;
        if (dfs(board, rowUsed, colUsed, boxUsed)) {
            return true;
        }
        rowUsed[x][val] = false;
        colUsed[y][val] = false;
        boxUsed[x / 3 * 3 + y / 3][val] = false;
        board[x][y] = 0;
    }

    return false;
}
```

Java 代码 - 找最少选择的格子



```java
private Position getLeastChoicesPosition(int[][] board,
                                         boolean[][] rowUsed,
                                         boolean[][] colUsed,
                                         boolean[][] boxUsed) {
    Position leastPosition = null;
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (board[i][j] != 0) {
                continue;
            }
            List<Integer> choices = new ArrayList<>();
            for (int val = 1; val <= 9; val++) {
                if (isValid(rowUsed, colUsed, boxUsed, i, j, val)) {
                    choices.add(val);
                }
            }
            if (leastPosition == null || choices.size() < leastPosition.choices.size()) {
                leastPosition = new Position(i, j);
                leastPosition.choices = choices;
            }
        }
    }

    return leastPosition;
}
```