Sudoku Programming

姓名：姜雨童
学号：3220103450

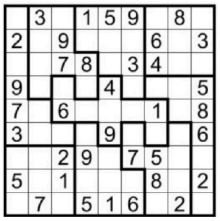
Chapter 1：任务说明

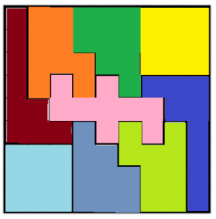
本次作业分为两个小任务：

1　数独生成器

根据用户在终端输入的提示数（1~81）以及MASK矩阵（作为分区依据），自动生成数独题目，要求每行、每列、每格中空格能尽可能均匀分布（示例见下图）。

- **根据用户的提示数，生成Sudoku题目**
- 1. 用户从命令行输入提示数（1~81）[在制作谜题时，提示数在22以下就非常困难，所以常见的数独题其提示数在23~30之间]。
- 2. 根据用户输入数字，以及MASK矩阵，自动生成数独题目，要求每行、每列、每格中空格能尽可能均匀分布





2　数独求解器

根据用户在终端输入的数独题目和分区MASK矩阵，系统自动给出解答，并从终端输出。

Chapter 2：算法及实现

尽管两个子任务在实现上有共通之处，为了测试和使用方便，我将两个任务分别写在`SudoGen.java`和`SudoSolver.java`文件中（其中部分函数的实现是一致的），下面我将分别对其进行说明：

1　SudoGen

在该文件下定义了`SudoGen`类，其中包含数独的尺寸大小，包含解答的`board`矩阵和只有提示不含解答的`unsolved`矩阵，存储每行/每列/每个分区现有数字信息、用来检查填入数字是否符合条件的`row`，`col`，`box`矩阵，以及包含分区信息的`mask`矩阵和提示数`hint`。

```java
public class SudoGen {
    private static final int SIZE = 9;
    private int[][] board;
    private int[][] unsolved;
    private boolean[][] row, col, box; // check if num is used in row, col, box
    private int[][] mask;
    private int hint;
    ...
}
```

主函数如下，主要包含三个过程：生成数独`sudo.generate()`，生成未解答数独`sudo.generateUnsolved()`，将两个矩阵输出到终端`sudo.printBoard(sudo.xxx)`。

```java
    public static void main(String[] args) {
        int[][] mask = {...};
        int hint = 23; // default value
```

```
4
5           Scanner scanner = new Scanner(System.in);
6           // hint and input info (omitted)
7
8           SudoGen sudo = new SudoGen(mask, hint);
9           sudo.generate();
10          sudo.generateUnsolved();
11          sudo.printBoard(sudo.unsolved);
12          System.out.println("\33[33m=====Solved board=====\33[0m");
13          sudo.printBoard(sudo.board);
14      }
```

实现的重点在于生成数独部分：

从左上角开始，通过递归回溯的方法逐行逐列填充数独中的每个小格。第12行开始为主要的实现部分。先查找三个矩阵确定要填充的数字不与已存在的数字存在冲突，将其填入数独中相应的小格，并修改三个矩阵中的标记，最后递归调用函数自身实现对下一个小格的填充。若是填充失败，则进行回溯（清空本次在小格中填的内容和三个矩阵中的标记）。

（该算法的一个缺点是分区确定时，生成的数独完全一致，不能做到随机生成。）

```
1       public void generate() {
2           fillGrid(0, 0);
3       }
4
5       private boolean fillGrid(int r, int c) {
6           if (r == SIZE) return true;
7           if (c == SIZE) return fillGrid(r + 1, 0); // next row
8           if (board[r][c] != 0) return fillGrid(r, c + 1); // skip used cell
9
10          int boxIndex = mask[r][c] - 1;
11
12          for (int num = 1; num <= 9; num++) {
13              if (!row[r][num] && !col[c][num] && !box[boxIndex][num]) {
14                  // fill cell
15                  board[r][c] = num; // [TODO] fill at random
16                  row[r][num] = col[c][num] = box[boxIndex][num] = true;
17                  // iterate next cell
18                  if (fillGrid(r, c + 1)) return true;
19                  // backtrack
20                  board[r][c] = 0;
21                  row[r][num] = col[c][num] = box[boxIndex][num] = false;
22              }
23          }
24          return false;
25      }
```

2 SudoSolver

数独求解器的基本结构和代码都与数独生成器一致，这里仅做简单分析。

首先是类`SudoSolver`的定义，比起数独生成器来说减少了几个不需要的变量，其他变量作用不变。

```java
public class SudoSolver {
    private static final int SIZE = 9;
    private int[][] board;
    private boolean[][] row, col, box; // check if num is used in row, col, box
    private int[][] mask;
}
```

主函数如下，利用函数`solver.solve()`的返回值来判断是否生成了数独的解，并做对应输出。而求解/输出过程与数独生成器大致一样，这里不做赘述。

```java
    public static void main(String[] args) {
        int[][] board = new int[SIZE][SIZE];
        int[][] mask = new int[SIZE][SIZE];

        Scanner scanner = new Scanner(System.in);
        // hint and input info (omitted)

        SudoSolver solver = new SudoSolver(board, mask);
        if (solver.solve()) {
            System.out.println("\33[33mSudoku puzzle solved:\33[0m");
            solver.printBoard();
        } else
            System.out.println("\33[31mSudoku puzzle cannot be solved.\33[0m");
    }
```

Chapter 3: 结果测试

1 SudoGen

数独生成器中以黄色字体表示终端输出信息，用不同背景色来表示数独的不同分区；在未解决的数独问题中，用下划线作为待填入数字的占位符。

图一是一个合法提示数（34）和程序默认分区的输出结果；图二则是非法提示数（程序采用默认值23）和输入分区mask矩阵的生成结果。二者均符合预期。

```
D:\Java>
D:\Java> d: && cd d:\Java && cmd /C ""C:\Program Files\Java\jdk-1.8\bin\java.exe" -agentlib:jdwp=tra
spend=y,address=localhost:54641 -cp C:\Users\Lenovo\AppData\Roaming\Code\User\workspaceStorage\2d664
redhat.java\jdt_ws\Java_72da403b\bin SudoGen "
Please enter the number of hints (1-81, default 23): 34
Please enter the mask (9x9, enter 0 to use default mask):
0
Generating board...
_ 2 3 4 5 6 7 _ 9
2 _ _ 7 _ _ 1 _ 6
6 7 _ 8 _ _ _ 4 5
4 _ _ 1 _ 2 _ _ 7
_ 5 1 2 _ _ _ 6 _
8 9 7 _ _ _ 5 1 2
_ _ 2 _ 4 _ 8 9 3
5 3 _ _ 2 _ 6 _ 1
_ 8 _ _ _ 1 _ _ 4
=====Solved board=====
1 2 3 4 5 6 7 8 9
2 4 5 7 8 9 1 3 6
6 7 9 8 1 3 2 4 5
4 6 8 1 3 2 9 5 7
3 5 1 2 9 7 4 6 8
8 9 7 3 6 4 5 1 2
7 1 2 6 4 5 8 9 3
5 3 4 9 2 8 6 7 1
9 8 6 5 7 1 3 2 4
```

```
Please enter the number of hints (1-81, default 23): 92
[Error]the number of hints should be between 1 and 81.
Using default value 23.
Please enter the mask (9x9, enter 0 to use default mask):
1 1 1 2 2 2 3 3 3
1 1 1 2 2 2 3 3 3
1 1 1 2 2 2 3 3 3
4 4 4 5 5 5 6 6 6
4 4 4 5 5 5 6 6 6
4 4 4 5 5 5 6 6 6
7 7 7 8 8 8 9 9 9
7 7 7 8 8 8 9 9 9
7 7 7 8 8 8 9 9 9
Generating board...
1 2 3 _ _ 6 _ 8 9
4 _ 6 7 _ _ 1 _ _
7 _ 9 1 2 _ _ 5 6
2 _ 4 _ 6 _ 8 _ 7
_ _ _ 8 _ _ _ 1 4
8 _ _ 2 1 4 _ 6 _
5 3 _ _ 4 _ _ _ 8
6 _ 2 9 7 8 _ _ 1
_ _ 8 5 3 _ _ _ 2
=====Solved board=====
1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 1 4 3 6 5 8 9 7
3 6 5 8 9 7 2 1 4
8 9 7 2 1 4 3 6 5
5 3 1 6 4 2 9 7 8
6 4 2 9 7 8 5 3 1
9 7 8 5 3 1 6 4 2
```

2 SudoSolver

数独求解器中同样使用黄色字体作为系统输出，用彩色背景表示数独的分区，用数字0作为待输入数字的占位符。

图一是一个无法求解的案例，图二则是一个输入数独和mask矩阵后的求解结果（作业要求中的示例）。

二者均符合预期。

```
redhat.java\jdt_ws\Java_72da403b\bin SudoSolver "
Please input the sudoku puzzle (9x9, with 0 for unknown numbers):
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
Please input the sudoku mask (9x9, with 1-9 for each number's position in the box):
1 2 2 3 3 3 4 4 4
1 2 2 3 3 3 4 4 4
1 2 2 2 3 3 4 4 4
1 2 5 2 5 3 6 6 6
1 1 5 5 5 5 5 6 6
1 1 1 8 5 9 5 9 6
7 7 7 8 8 9 9 9 6
7 7 7 8 8 8 9 9 6
7 7 7 8 8 8 9 9 6
Sudoku puzzle cannot be solved.
```

```
redhat.java\jdt_ws\Java_72da403b\bin SudoSolver "
Please input the sudoku puzzle (9x9, with 0 for unknown numbers):
0 3 0 1 5 9 0 8 0
2 0 9 0 0 0 6 0 3
0 0 7 8 0 3 4 0 0
9 0 0 0 4 0 0 0 5
7 0 6 0 0 0 1 0 8
3 0 0 0 9 0 0 0 6
0 0 2 9 0 7 5 0 0
5 0 1 0 0 0 8 0 2
0 7 0 5 1 6 0 2 0
Please input the sudoku mask (9x9, with 1-9 for each number's position in the box):
1 2 2 3 3 3 4 4 4
1 2 2 3 3 3 4 4 4
1 2 2 2 3 3 4 4 4
1 2 5 2 5 3 6 6 6
1 1 5 5 5 5 5 6 6
1 1 1 8 5 9 5 9 6
7 7 7 8 8 9 9 9 6
7 7 7 8 8 8 9 9 6
7 7 7 8 8 8 9 9 6
Sudoku puzzle solved:
6 3 4 1 5 9 2 8 7
2 5 9 4 7 8 6 1 3
1 2 7 8 6 3 4 5 9
9 1 8 6 4 2 3 7 5
7 4 6 3 2 5 1 9 8
3 8 5 2 9 1 7 4 6
4 6 2 9 8 7 5 3 1
5 9 1 7 3 4 8 6 2
8 7 3 5 1 6 9 2 4
```

Chapter 4: 源代码

1 SudoGen.java

```java
import java.util.Random;
import java.util.Scanner;

public class SudoGen {
    private static final int SIZE = 9;
    private int[][] board;
    private int[][] unsolved;
    private boolean[][] row, col, box; // check if num is used in row, col, box
    private int[][] mask;
    private int hint;

    // constructor
    public SudoGen(int[][] mask, int hint) {
        this.mask = mask;
        this.hint = hint;
        board = new int[SIZE][SIZE];
        unsolved = new int[SIZE][SIZE];
        row = new boolean[SIZE][SIZE + 1];
        col = new boolean[SIZE][SIZE + 1];
        box = new boolean[SIZE][SIZE + 1];
    }

    public void generate() {
        fillGrid(0, 0);
    }

    private boolean fillGrid(int r, int c) {
        if (r == SIZE) return true;
        if (c == SIZE) return fillGrid(r + 1, 0); // next row
        if (board[r][c] != 0) return fillGrid(r, c + 1); // skip used cell

        int boxIndex = mask[r][c] - 1;

        for (int num = 1; num <= 9; num++) {
            if (!row[r][num] && !col[c][num] && !box[boxIndex][num]) {
                // fill cell
                board[r][c] = num; // [TODO] fill at random
                row[r][num] = col[c][num] = box[boxIndex][num] = true;
                // iterate next cell
                if (fillGrid(r, c + 1)) return true;
                // backtrack
                board[r][c] = 0;
                row[r][num] = col[c][num] = box[boxIndex][num] = false;
            }
        }
        return false;
    }

    public void generateUnsolved() {
        for (int r = 0; r < SIZE; r++) // copy board to unsolved
            for (int c = 0; c < SIZE; c++)
```

```
52                  unsolved[r][c] = board[r][c];
53
54          int count = SIZE * SIZE - hint;
55          while (count != 0) { // clear the grid in unsolved board with random index
56              int i = new Random().nextInt(SIZE);
57              int j = new Random().nextInt(SIZE);
58              if (board[i][j] != 0) {
59                  unsolved[i][j] = 0;
60                  count--;
61              }
62          }
63      }
64
65      public void printBoard(int[][] map) {
66          for (int r = 0; r < SIZE; r++) {
67              for (int c = 0; c < SIZE; c++) {
68                  // System.out.print(board[r][c] + " ");
69                  int boxIndex = mask[r][c] - 1;
70                  switch(boxIndex) {
71                      case 0: System.out.print("\33[41m"); break;
72                      case 1: System.out.print("\33[42m"); break;
73                      case 2: System.out.print("\33[43m"); break;
74                      case 3: System.out.print("\33[44m"); break;
75                      case 4: System.out.print("\33[45m"); break;
76                      case 5: System.out.print("\33[46m"); break;
77                      case 6: System.out.print("\33[47m"); break;
78                      case 7: System.out.print("\33[48m"); break;
79                      case 8: System.out.print("\33[42m"); break;
80                  }
81                  printGrid(r, c, map);
82                  System.out.print("\33[0m");
83              }
84              System.out.println();
85          }
86      }
87
88      public void printGrid(int r, int c, int[][] map) {
89          if (map[r][c] == 0) System.out.print("_ ");
90          else System.out.print(map[r][c] + " ");
91      }
92
93      // main method
94      public static void main(String[] args) {
95          int[][] mask = {
96              {1, 2, 2, 3, 3, 3, 4, 4, 4},
97              {1, 2, 2, 3, 3, 3, 4, 4, 4},
98              {1, 2, 2, 2, 3, 3, 4, 4, 4},
99              {1, 2, 5, 2, 5, 3, 6, 6, 6},
100             {1, 1, 5, 5, 5, 5, 5, 6, 6},
101             {1, 1, 1, 8, 5, 9, 5, 9, 6},
102             {7, 7, 7, 8, 8, 9, 9, 9, 6},
103             {7, 7, 7, 8, 8, 8, 9, 9, 6},
104             {7, 7, 7, 8, 8, 8, 9, 9, 6}
105         };
106         int hint = 23; // default value
107
```

```
        Scanner scanner = new Scanner(System.in);
        System.out.print("\33[33mPlease enter the number of hints (1-81, default 23): \33[0m");
        hint = scanner.nextInt();
        if (hint < 1 || hint > 81) {
            System.out.println("\33[31m[Error]the number of hints should be between 1 and 81.\33[0m");
            System.out.println("\33[33mUsing default value 23.\33[0m");
            hint = 23;
        }
        System.out.println("\33[33mPlease enter the mask (9x9, enter 0 to use default mask):\33[0m");
        mask:
        for (int r = 0; r < SIZE; r++) {
            for (int c = 0; c < SIZE; c++) {
                int num = scanner.nextInt();
                if (r == 0 && c == 0 && num == 0)
                    break mask;
                mask[r][c] = num;
            }
        }
        scanner.close();

        System.out.println("\33[33mGenerating board...\33[0m");
        SudoGen sudo = new SudoGen(mask, hint);
        sudo.generate();
        sudo.generateUnsolved();
        sudo.printBoard(sudo.unsolved);
        System.out.println("\33[33m=====Solved board=====\33[0m");
        sudo.printBoard(sudo.board);
    }
}
```

2 SudoSolver.java

```
import java.util.Scanner;

public class SudoSolver {
    private static final int SIZE = 9;
    private int[][] board;
    private boolean[][] row, col, box; // check if num is used in row, col, box
    private int[][] mask;

    // Constructor
    public SudoSolver(int[][] board, int[][] mask) {
        this.board = board;
        this.mask = mask;
        row = new boolean[SIZE][SIZE + 1];
        col = new boolean[SIZE][SIZE + 1];
        box = new boolean[SIZE][SIZE + 1];
        initialize();
    }

    private void initialize() {
        for (int r = 0; r < SIZE; r++) {
            for (int c = 0; c < SIZE; c++) {
```

```
                int num = board[r][c];
                if (num != 0) {
                    int boxIndex = mask[r][c] - 1;
                    row[r][num] = col[c][num] = box[boxIndex][num] = true;
                }
            }
        }
    }

    public boolean solve() {
        return fillGrid(0, 0);
    }

    // the same as in SudoGen.java
    private boolean fillGrid(int r, int c) {
        if (r == SIZE) return true;
        if (c == SIZE) return fillGrid(r + 1, 0); // next row
        if (board[r][c] != 0) return fillGrid(r, c + 1); // skip used cell

        int boxIndex = mask[r][c] - 1;

        for (int num = 1; num <= 9; num++) {
            if (!row[r][num] && !col[c][num] && !box[boxIndex][num]) {
                // fill cell
                board[r][c] = num;
                row[r][num] = col[c][num] = box[boxIndex][num] = true;
                // iterate next cell
                if (fillGrid(r, c + 1)) return true;
                // backtrack
                board[r][c] = 0;
                row[r][num] = col[c][num] = box[boxIndex][num] = false;
            }
        }
        return false;
    }
    public void printBoard() {
        for (int r = 0; r < SIZE; r++) {
            for (int c = 0; c < SIZE; c++) {
                // System.out.print(board[r][c] + " ");
                int boxIndex = mask[r][c] - 1;
                switch(boxIndex) {
                    case 0: System.out.print("\33[41m"); break;
                    case 1: System.out.print("\33[42m"); break;
                    case 2: System.out.print("\33[43m"); break;
                    case 3: System.out.print("\33[44m"); break;
                    case 4: System.out.print("\33[45m"); break;
                    case 5: System.out.print("\33[46m"); break;
                    case 6: System.out.print("\33[47m"); break;
                    case 7: System.out.print("\33[48m"); break;
                    case 8: System.out.print("\33[42m"); break;
                }
                System.out.print(board[r][c] + " ");
                System.out.print("\33[0m");
            }
            System.out.println();
        }
```

```
78      }
79
80      // main method
81      public static void main(String[] args) {
82          int[][] board = new int[SIZE][SIZE];
83          int[][] mask = new int[SIZE][SIZE];
84
85          Scanner scanner = new Scanner(System.in);
86          System.out.println("\33[33mPlease input the sudoku puzzle (9x9, with 0 for
    unknown numbers):\33[0m");
87          for (int i = 0; i < SIZE; i++) {
88              for (int j = 0; j < SIZE; j++)
89                  board[i][j] = scanner.nextInt();
90          }
91
92          System.out.println("\33[33mPlease input the sudoku mask (9x9, with 1-9 for
    each number's position in the box):\33[0m");
93          for (int i = 0; i < SIZE; i++) {
94              for (int j = 0; j < SIZE; j++)
95                  mask[i][j] = scanner.nextInt();
96          }
97          scanner.close();
98
99          SudoSolver solver = new SudoSolver(board, mask);
100         if (solver.solve()) {
101             System.out.println("\33[33mSudoku puzzle solved:\33[0m");
102             solver.printBoard();
103         } else
104             System.out.println("\33[31mSudoku puzzle cannot be solved.\33[0m");
105     }
106 }
```

Declaration

I hereby declare that all the work done in this project titled "Sudoku Programming" is of my independent effort.