Sudoku Solver Documentation (CS 4A - Fall 2024)

Programmed and Documented by Kaveh Zare

Date: 10/26/2024

High-Level Description:

- For the Sudoku problem solver, recursive backtracking will allow us to place all necessary numbers (1-9) in all empty slots while adhering to the rules of the challenge. In essence, the solver cannot place any number more than once in a given row, column, or 3x3 sub-grid.
- The backtracking will exhaust all available numbers in a given column and once it's not able to place any number in a given column while not having reached the final row, the algorithm will recursively backtrack.
    - This means that the prior choice is remade with the next valid number if possible and the recursion carries on as such.
- Please Note: Empty slots are marked with zeros.

Model:

- There are two shared states passed between recursive calls: `board` and `grids`
    - `board` is a *9x9* grid and is used to place numbers and check values in each row and column.
    - `grids` is a representation of the *3x3* sub-grid system which consists of a hash map with the grid number (ranging between 1 and 9) serving as keys and values are Hash Sets of unique numbers in each of the 9 sub-grids.

Testing Program:

```
int[][] board1 = {
        {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}
};
```

```
System.out.println("Test Case 1:");

printBoard(board1);

if (solveSudoku(board1)) {

        System.out.println("Solved:");

        printBoard(board1);

}
else {

        System.out.println("No Solution");

}

System.out.println("\n-------------------\n");

// Test Case 2: Already solved Sudoku board

int[][] board2 = {
        {5, 3, 4, 6, 7, 8, 9, 1, 2},
        {6, 7, 2, 1, 9, 5, 3, 4, 8},
        {1, 9, 8, 3, 4, 2, 5, 6, 7},
        {8, 5, 9, 7, 6, 1, 4, 2, 3},
        {4, 2, 6, 8, 5, 3, 7, 9, 1},
        {7, 1, 3, 9, 2, 4, 8, 5, 6},
        {9, 6, 1, 5, 3, 7, 2, 8, 4},
        {2, 8, 7, 4, 1, 9, 6, 3, 5},
        {3, 4, 5, 2, 8, 6, 1, 7, 9}
};

System.out.println("Test Case 2:");

printBoard(board2);

if (solveSudoku(board2)) {
        System.out.println("Solved (must be unchanged):");
        printBoard(board2);
}
else {
System.out.println("No Solution");
}
```

```
System.out.println("\n-------------------\n");

// Test Case 3: Unsolvable Sudoku board

int[][] board3 = {
        {5, 3, 4, 6, 7, 8, 9, 1, 2},
        {6, 7, 2, 1, 9, 5, 3, 4, 8},
        {1, 9, 8, 3, 4, 2, 5, 6, 7},
        {8, 5, 9, 7, 6, 1, 4, 2, 3},
        {4, 2, 6, 8, 5, 3, 7, 9, 1},
        {7, 1, 3, 9, 2, 4, 8, 5, 6},
        {9, 6, 1, 5, 3, 7, 2, 8, 4},
        {2, 8, 7, 4, 1, 9, 6, 3, 5},
        {3, 4, 5, 2, 8, 6, 1, 7, 0}
};

System.out.println("Test Case 3 (unsolvable): ");

printBoard(board3);

if (solveSudoku(board3)) {
        System.out.println("Solved:");
        printBoard(board3);
}
else {
        System.out.println("No Solution");
}// end of testing program
```

Sample Input and Output

Test Case 1:

```
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
```

Solved:

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

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

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