Assignment: 4
Name: Malav Jani
Banner ID: B00851408

Overview

This given problem is about making a variant of Sudoku solver. As every Sudoku grid contains a unique number in a column and rows, this problem will serve the same idea. The problem is called "Mathdoku". In this n*n matrix will be formed, and a small group will be given with some constraints. Here some operators with the expected result will be provided for each group. There will be one main class "Mathdoku" having multiple methods to solve the puzzle. Some important methods used will be **loadPuzzle, readyToSolve and solve.**

Files and External Data:

- Mathdoku.java – This java file will have the five given methods and five supporting methods which will return the result to the main method as per methods call from main respectively.

- Grid.java – This java file will be supporting class to the Mathdoku method. This class will have 12 different methods for solving the puzzle.

Inputs

- There will be given multiple options to the user for choosing the operations.
- This input will be then passed to the respective case from which particular method will be called.
- There are multiple choices for the user to input and call the method or exit.
    - loadPuzzle (This will take the argument as a BufferReader where the stream will contain the content of the file in which the puzzle detail will be provided).
    - readyToSolve(This will return true if the puzzle can be solved and false if not)
    - solve(This will return true after the puzzle is successfully solved)
    - print(print will return the string having solved grid)
    - choice(this will return the count of backtracking done to obtain final solution)
    - quit (Used to exit the program)

Outputs

- As can be seen, there are multiple options for inputting the output, which will vary as per the input.
- All the output can be expected to work normally if the puzzle is loaded first.
- There are five chances of output if the following input commands are executed:
    a. loadPuzzle - This will return true if the puzzle is loaded successfully and if the given stream is valid. Else it will return false on an unsuccessful attempt of loading the puzzle.

b. readyToSolve – This method will return true if the puzzle has enough data to give solve. If the operator list will be right, and the grid data will be right, then the puzzle will be ready to solve. It will return false if the data given will be wrong.
c. solve –This method will return true if the puzzle data is given right, and the puzzle is solved with all the given constraints. If there will be any problem in solving the puzzle, then it will return false.
d. print – This method will return a string which will be containing the solved puzzle, and if the puzzle is not solved, it will give the cell group provided in the problem.
e. choices – In this method counter of the guesses that the program has to make to reach the final answer will be returned.
f. quit- Output will be quit, and it will exit from execution.

Key Algorithm and Design Elements

- In this program, multiple data structures are used like HashMaps, List, ArrayList, and 2D Arrays to store the data and also to process it.
    - List<String> - In this list, all the lines from the stream will be read and stored for further operations.
    - Hashmap<String, ArrayList<int[]>>- HashMap will have a string as a key and integer as a value. Where integer array will contain cells of the matrix.
    - HashMap<String, String[]> -This will be used for storing operator group. All the group id will be stored as the key, and the expected result and the operator will be stored as the values which will be used in the evaluation of the puzzle later.
    - Int [][] - This will be used to store the given matrix given in the input. The two-dimension array will be used to store the solved group or the given cell group of the puzzle.
    - ArrayList<Integer>- There is some array list used in the program which will store all the details of remaining and present number in the grid. Other than this array list will store the number or the value which will be used by operators to solve the group.
    - ArrayList<int[]>- This array list will store the data of array in which data of the position in the group and the puzzle will be stored.

- **boolean loadPuzzle(BufferReader stream)**
    - This method will be used to load the puzzle from the given BufferReader.
    - From the main method, BufferReader will be passed as an argument to load the puzzle.
    - The stream will be read and converted into the list, and all the lines will be stored in it one by one.
    - After the final list is generated, it will be used to create two hashmaps using two functions **makecell** and **makeOperatorGroup.**

        - **boolean makeCell(int dim)**
            - This method will be entering all the values of the index of each group into the hash map.
            - This method will take dimension as an argument for storing the group until the matrix ends.
            - If the dimension does not match to the given data, it will return false.
            - And on successful creation of the map, it will return true.
            - This method is also responsible for storing the grid data in the static variable of the grid.
        - **boolean makeOperatorGroup(int dim)**
            - This method will assign all the group members with its expected result and the operator to be used.
            - The group id will be given as key to the hashmap, and all other data will be stored in the string array.
            - This will return true if all the operators and expected result are stored successfully.
    - After calling both the methods if both methods return true, the flag of load puzzle will be changed to true, and the method will return true back to the main method.
    - All the flags used are used in further operations.
- **boolean readyToSolve()**
    - The given method will check if there is any chance where a flaw can be caught of not solving the puzzle and if there is an error on the given data.
    - This will return true if all the given data will be right and the puzzle can be solved using that data.
    - Firstly, it will check whether the given data is having the decided operators or not.
    - And the given operators are “+”, “-”, “/”, “*”, “=”.
    - Other than these operators are found the method will return false.
    - Initially this will also check the data, that is the expected result is in integer form or not.
    - Other than this many other supporting methods are used to check the puzzle.
    - Some of the methods are checkSymbols, checkMath and isValidGrid.
        - **checkSymbols()**
            - This method will be used for checking all the group is assigned with the valid operator.
            - If the group is missing with the operator it will return false.
        - **checkMath()**
            - checkMath is used specially for checking two operators and expected result.
            - If the subtraction or division is given greater than the dimension of puzzle it will return false.
        - **isValidGrid()**
            - This method is used for checking that all the given rows in the matrix is even and having same dimension.

        - The dimension will be considered from the first row and will check the whole matrix.
        - If the given matrix will not have same dimension matrix it will return false.
    - All the methods will return Boolean value and accordingly the method will return the value.
    - Also before returning to main method the flag will get updated and before the operations in method starts it will check the puzzle is loaded or not.
- **boolean solve()**
    - Solve method is important method to solve the puzzle.
    - This will use other class which is **Grid** and pass all the data to that class where all the solution will be carried out.
    - Before solving puzzle, it will check the puzzle is loaded and the puzzle is ready for solving.
    - It will then pass all the data to the other class.
    - And after the solution is done successfully it will return true else false.
- **String print()**
    - This method will return the solved puzzle or give the cell group with provided in the problem.
    - This will return the output in string form and give the data of grid.
- **int choices()**
    - In this method the counter will be stored where it will keep count on how many times the code has backtracked a specific solution to get the final answer.
    - And it will also return counter if the solution is not obtained.
- Other than the given class this program contains one additional java class which will be used to solve the puzzle and functionalities of some major methods is given in following points.
- **boolean init()**
    - This method will be called by the solve method.
    - In this the grid will be initialized with zero.
    - Also the group with "=" will be assigned with the given value.
    - This will make the options less for making the decision of the puzzle.
    - After the grid operation is done it will call the solveMathDoku method for the solution.
- **Boolean solveMathDoku(int[][] matrix)**
    - This method will receive the initial matrix with zero value and also having some value of "=" in the grid.
    - This method will be called recursively and it will be used to solve the puzzle.
    - Every time this method will call solveGroup method and it will return the cells of minimum group having all the values filled.
    - After the list is received from the method it will check the grid row and column not having zero value and move further by calling canEnter method to use the numbers that are not present in particular row and column.
    - After entering the values in row and column of the group it will send the grid to checkCurrentMatrix.

- This method will see the particular group is filled or not and if it is filled it will see the constrain given to the group.
- After that it will call the respective method to solve the group.
- If the given values of group servers the constrain it will return true to solveMathDoku method and store the data to the group.
- This operations will be done several times until the final grid is obtained or the given data fails to solve the puzzle.

- **Arraylist<Integer> canEnter(int row, int col)**
  - This method will use other two methods to get the data of row and column.
  - **isInRow and isInCol** will return the numbers that are not present in the row and column.
  - This values will be sorted out and a final list of numbers will be obtained to enter in the grid.
  - The method will help to choose a unique number for the grid.
- **Arraylist<int[]> solveGroup(int[][] matrix)**
  - The major function of this method will be returning the minimum group present in the grid.
  - All the group that contains empty cell will be given to the solveMathDoku method.
- **Boolean checkCurrentMatrix(int[][] matrix)**
  - This method will be used to solve the particular group and check whether the given data and the operator server the constrain or not.
  - This will get the max size amongst all the group and that will be used to check the groups.
  - The max group size will then be used to check all the group and see if any group is filled with data and check the operations.
  - Then the operators will be applied and the result will be checked with the expected result.
  - The supporting methods are **add, sub, mul and div.**

Strategy and Algorithm

- **Solving the puzzle**
  a. Initially the grid will be initialized using zero.
  b. All the cell having equal value will be assigned to the grid. By doing so the number of possibilities will decrease in order to solve the puzzle.
  c. After the grid is initializer I have choose to go group by group in solving the puzzle.
  d. Firstly, I will take one group where the group is having small size. Which means generally I will start searching for group of size two.
  e. After the group is selected I will check if the group is having cells which are unassigned with the value or not.

f. If the group will have unassigned values it will then check which numbers can be inserted in the particular cell such that it is not present in it corresponding row or column.
g. Once we have the number we will pass those number to group cell and then will check the assigned numbers servers the constrain or not.
h. If the constrains are satisfied then other group will be explored.
i. As the method of solving is recursive, if the numbers are invalid it will be backtracked and another numbers will be tried.
j. By going through this approach one major advantage will be at the end of the puzzle biggest group will have low choices of available numbers. As a result bigger group will get filled quickly.

Input Validation

- **boolean loadPuzzle(BufferReader stream)**
  - "stream" is null.
  - "steam" is empty
  - "stream" having all the whitespace character inside it.
  - "stream" having any empty line.
  - "stream" having dimension of grid which is uneven.
- **boolean readyToSolve( )**
  - No input validation required.
- **boolean solve()**
  - No input validation required.
- **String print( )**
  - No input validation required.
- **int choices( )**
  - No input validation required.

Boundary Cases

- **boolean loadPuzzle(BufferReader stream)**
  - Each cell is a unique group.
  - Whole puzzle is given as a group.
  - Grid provided has minimum size of 2*2.
  - Grid provided has minimum size of 9*9.
- **boolean readyToSolve( )**
  - No boundary case is required.
- **boolean solve()**
  - Puzzle cannot be solved.
  - Puzzle is already solved.
- **String print( )**
  - Print the puzzle is the puzzle fails to give solution.
  - Print the puzzle after the puzzle is successfully solved.

- **int choices( )**
    - No boundary cases required.

Control Flow

- **boolean loadPuzzle(BufferReader stream)**
    - Given grid is not in proper dimension of square.
    - Given grid is unevenly given.
- **boolean readyToSolve( )**
    - No constrain is provided for the group.
    - Constrain is provided but the group is not provided.
    - Multiple constrains are provided for one group.
    - Invalid operators are given other than 5 operators.
- **boolean solve()**
    - No control flow required.
- **String print( )**
    - No control flow required.
- **int choices( )**
    - No control flow required.

Data Flow:

- Calling readyToSolve before loadPuzzle.
- Calling solve method before readyToSolve and loadPuzzle.
- Calling print before loadPuzzle.
- Calling choices before solve method.
- **Normal Flow**
    - loadPuzzle
    - readyToSolve
    - solve
    - print/choices in any order

Assumption

- Multiple white spaces will be allowed in input stream.
- For **multiplication and division** no more than 2 cells will be allowed.
- The constrain will have only 5 operators which are, +, -, /, * and =.
- Expected result in operator group cannot exceed the matrix size. That is if the matrix is 4*4 constrain cannot be given as 5.
- No negative number will be given in expected result of group.
- In particular grid only one set of group having id will be entered.

Limitations

- If the grid will be given with dimension more than 9*9 this code will take some time to give the solution.
- Other than mentioned 5 operators no other operations can be carried out.

References

- https://www.geeksforgeeks.org/sudoku-backtracking-7/ referred this link for getting more idea about backtracking.
- https://docplayer.net/45439751-Kenken-puzzle-solver-using-backtracking-algorithm.html referred this paper to know more about how kenken works.