CSCI: 3901 Assignment 4
Winter 2020

External Documentation:

Overview:

This program Mathdoku accepts a puzzle passed by the user and checks for its solvability, based on it the puzzle is solved. It has been written in Java language without using the collections framework.
A summary of the Mathdoku requirements is in the CSCI 3901 course assignment #4 problem 4 information in the course's brightspace space.

The solution consists of class "Mathdoku" and it includes the following methods as mentioned in brightsapce:

- boolean loadPuzzle(BufferedReader stream):- The method reads files from user calls and loads the puzzle in separate hashMaps i.e. hashKeys for storing the group character and ArrayList of Indexes, mathdokuOperand stores the value and operator as key and group character as values. It checks for validation of the sequence of stream inputs and based on input validations the method returns Boolean value.

- boolean readyToSolve():- This method is used to check if the puzzle is solvable by checking for the locations size obtained from the values of each group in hashKeys HashMap and compare it with the gridLengthMax* gridLengthMax. If the comparision is false, then the method returns boolean false and resets the globally declared hashMaps hashKeys, mathdokuOperand and mathdokuPuzzle.

- boolean solve():- The solve method recursively calls solveRecursively for each and every grid and computes the puzzle. If there is any issue while solving the puzzle the method returns null value.

- String print():- This method prints the puzzle in string format. str attribute is used to append the values obtained by passing the ArrayList<Integer> position as key to mathdokuPuzzle and is dispayed as output. In case of null values in the position we return the group character in the output format.

- int choices( ):- This method returns the number of wrong choices made using count_Wrong_Choices variable.

Predefined classes used as part of the program:

- No predefined classes were used as part of the program.

Predefined Interfaces used as part of the program:

- No predefined classes were used as part of the program.

References:

- Referred //https://stackoverflow.com/questions/17192796/generate-all-combinations-from-multiple-lists for generating combinations

Files and external data:

The program consists of two file:

- Main.java - main for the program that calls the Mathdoku.java file.
- Mathdoku.java - class that makes the calls and runs various Mathdoku methods and solves the puzzle in case of valid scenarios.

Data structures and their relations to each other:

- Used HashMap<Character, ArrayList<ArrayList<Integer>>> hashKeys for storing the group as key and their indexes as values .
- Used HashMap<String, String> mathdokuOperand to store the operation and computation number as key and the group name as value .
- Used HashMap<ArrayList<Integer>, Integer> mathdokuPuzzle to store the puzzle where the position is the key and the computed value is the value.
- Used BufferedReader for reading the input from the main method.
- Used IOException for checking if the file is empty.

Assumptions:

- We have assumed that there will not be any reverse order of loadPuzzle sequence between puzzle group sequence and constraints for each cell.
- We are removing the whitespaces from the stream inputs passed during loadPuzzle method.
- During every loadPuzzle method call, reset the globally declared hashMaps hashKeys, mathdokuOperand and mathdokuPuzzle for handilng the Data flow.
- If the readyToSolve method is returning Boolean value as false then, reset the globally declared hashMaps hashKeys, mathdokuOperand and mathdokuPuzzle for handilng the Data flow.
- Have assumed the puzzle length as the first input sequence from stream.
- If a constraint is missing the apart from those group indexes fill the rest of the puzzle with values and the missing indexes will save the group name.

Choices:

- Could have returned false for empty or null stream inputs during loadPuzzle but instead have skipped the input line.

Fundamental algorithms and design elements:

For boolean loadPuzzle(BufferedReader stream) method:

- The method checks the input validations of the String passed and returns false in case of invalid scenarios.
- Initially the puzzle strings are fetched in hashKeys Hashmap with group character as key and their corresponding indexes as ArrayList<ArrayList<Integer>>. gridLength is used as a counter to add the values in hashKeys by comaring each string length with gridLengthMax.
- After the list1 of input now the cell constraints are added.
- In case of invalid scenarios, the hashKeys is cleared and Boolean false is returned to the calling method.
- The list 2 inputs are added in mathdokuOperand HashMap.
- This method returns Boolean value in case of the mathdokuOperand size is not empty.

For boolean readyToSolve() method:

- This method returns false if the gridLengthMax of the puzzle is 0.
- It computes whether the puzzle is solvable by checking for the locations size obtained from the values of each group in hashKeys HashMap and assigning the sizes to count variable.
- count is then compared with the gridLengthMax* gridLengthMax of the puzzle and if it matches then the method returns true else the method returns false and resets the globally declared hashMaps hashKeys, mathdokuOperand and mathdokuPuzzle for handilng the Data flow.

For boolean solve() method:

- If the readyToSolve method returns false then the puzzle is not solvabe and retuns false to the calling method and resets the globally declared hashMaps hashKeys, mathdokuOperand and mathdokuPuzzle for handilng the Data flow.
- This method solves the puzzle by recursively calling the solveRecursively method by passing the constraints keySets from mathdokuOperand Hashmap and idx as index.
- The idx is incremented for every iteration and the next value is recursively called.
- The ArrayList<String> strs size is compared with idx and if true the method returns true.
- Based on the strs we obtain the the strKeys which consists the constraints and if the constraints contain = operator then we replace it with + to reduce computations.
- groupingElement is obtained by mathdokuOperand.get(strKeys) and based on that we pass this as a key to hashKeys Map and obtain the indexes of the constraint groupings.
- generateCombination method is called to generate all possible combination based on the size of the index.

- This combination is further used to obtain efficientCombination by passing combinations(generateCombinations(indexes), strKeys).
- The grenerateCombination method calls the generatePermutations method which recursively calls generatePermutations method for generating combinations based on various depth levels.
- In each generatePermutation recursive calls we pass the operations i.e "+-/*" and the expected value to the corresponding switch cases. Based on the switch cases we call sMultiplication, sAddition, sSubtraction, sDivision method and the combinations are added only if the expected value is obtained by the above passed operation and list of current combination values.
- These sMultiplication, sAddition, sSubtraction and sDivision methods returns boolean based on the expected output.
- After obtaining the efficientCombination, we iterate over each combinations and put the values to mathdokuPuzzle based on row- column wise ambiguity, group wise ambiguity and other group unsolvability.
- In case of bad choice, the program backtracks and removes the ecombos and checks for the next combinations.
- We remove the bad combinations from the mathdokuPuzzle HashMap and increment the count_Wrong_Choices.
- We follow the backtracking for every possible combinations of the group and if still we are unable to solve the puzzle then we check for the previous group combination by backtracking and deleting the recent choices.
- After all permutations and combinations, still if the puzzle remains unsolved then Boolean false is returned to the calling method.

For String print() method:

- The print method prints the puzzle in string format.
- It checks for gridLengthMax and readToSolve invalid cases.
- Assigning a 2d character array puzzleGroupNames for elements of the group.
- Iterate each position of the mathdokuPuzzle HashMap and append the integer values to the str variable.
- In case the position of the mathdokuPuzzle is empty then we add the group character in str using the puzzleGroupNames array.

For choices() method:

- This method return the globally declared count_Wrong_Choices variable to the calling method.

Limitations:

- The Main class is user-friendly, but in this case, have hardcoded the parameters passed as per the question requirement.
- This program has a limitation for solving puzzles till 8*8 grid with lower level of difficulty.

Steps taken to perform degree of efficiency:

- Each group assigned has indexes and for each index, I have optimised the combinations and stored in efficientCombination variable. The combination method calls the generatePermutations method which filters the all possible combinations passed to the combinations method and performs the mathematical computation(sAddition, sSubtraction, sMultiplication, sDivision) based on operation and expected values.

- After obtaining the list of efficient combinations, selected each combinations and check for row and column wise ambiguity for each group. In case of ambiguity, we are checking next pair of combinations.

- Eliminated the possibility of ambiguity with different groups before assigning the values to the mathdokuPuzzle.

- Eliminated the possibility of row-column ambiguity within the same group before assigning the values to the mathdokuPuzzle.

- In case of any failures, backtracking is performed and the elements pushed in mathdokuPuzzle HashMap is removed and the program backtracks to the previous group efficient combinations.

- By this, efficiency is achieved by reducing eliminating the no. of combinations.

Test cases

Input Validation:

- For boolean loadPuzzle(BufferedReader stream) method:
  - If stream is empty, then return false.
  - If stream is null, then return false.
  - If stream contains constraints for part2 in different sequence, then return false.
  - If stream contains puzzle strings for part1 of different length, then return false.

Boundary Cases:

- For boolean loadPuzzle(BufferedReader stream) method:
  - Loading the puzzle with one cell.
  - Loading the puzzle with many cells.
  - Loading the puzzle with no constraints.
  - Loading the puzzle with more constraints.
  - Loading the puzzle with less constraints.
  - Loading the puzzle with n*n gridLengthMax
  - Loading the puzzle with less number of cells.

Control Flow:

- For boolean loadPuzzle(BufferedReader stream) method:
  - Loading the puzzle with normal cell size.
  - Loading the puzzle with same number of groups and constraints.
  - Loading the puzzle with different number of groups and constraints.
  - Loading the puzzle with multiple spaces in constraints.
- For boolean readyToSolve()method:
  - Checking the method when the puzzle has no solution.
  - Checking the method when puzzle has a solution.
  - Checking the method when puzzle has a smaller number of positions covered in the puzzle grid.
  - Number if cells in division and subtraction.
  - Check for a constraint missing for a group

- For boolean solve()method:
  - Solving the puzzle with easy computation.
  - Solving the puzzle with zero solutions.
  - Solving the puzzle with difficult computations.
  - Solving the puzzle having multiple solutions.
  - Solve the puzzle with missing operators or computing values.

- For String print()method:
  - Print the puzzle having zero solution.
  - Print the puzzle with valid solution.
  - Print the puzzle with invalid solution.

- For int choices()method:
  - If tolerance passed is less than or equals to 0, return null.

Data Flow:

  - Call loadPuzzle method multiple times with same File.
  - Call loadPuzzle method multiple times with different File.
  - Call readyToSolve method before calling the loadPuzzle method.
  - Call print method before calling the loadPuzzle method.
  - Call print method before calling the solve method.
  - Call print method after calling the loadPuzzle and solve method.
  - Call solve method after calling the loadPuzzle method.
  - Call choices method before calling the loadPuzzle method.
  - Call choices method before calling the loadPuzzle and after solve method.
  - Call solve method before calling the loadPuzzle method
  - Call choices method before calling the loadPuzzle,solve and readyToSolve method.