Algorithmics	Student information	Date	Number of session
	UO: 295180	17/04/24	6
	Surname: Orviz Viesca		
	Name: Mario		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Backtracking algorithms

I didn't manage to finish the problem before the deadline, although I have some code done. In order to solve this problem using backtracking, I tried to divide the problems into smaller ones.

My class NumericSquareOne.java provides some methods to convert the String matrix that the parser generates into a more suitable data structure for the program. The implementation of a new class called Number was done, so my algorithm works over a matrix of Numbers.

The class Number contains two attributes:

- boolean inmutable -> Determines if this number can be changed or if it's fixed by the problem.
- int value -> Contains the numeric value of the Number.

Therefore, my algorithm does not work with number but using methods of the Number instance corresponding to each square. This class has the getters and setters for those two attributes and the most important method for the problem that is increase(). It increments the value of the number in each call and, if the value is 9, it jumps directly to zero.

The method in NumericSquareOne called fillUpNumberMatrix() generates the matrix of the size of the problem filled up by numbers containing 0 and inmmutable==false if a "?" is found or the number that is found and inmutable==true otherwise.

Then, the method fillUpRowOperators() generates a matrix of size x size-1 with all the operators corresponding to the rows and, the fillUpColOperators() generates other matrix of size-1 x size with the ones corresponding to the columns.

Algorithmics	Student information	Date	Number of session
	UO: 295180	17/04/24	6
	Surname: Orviz Viesca		
	Name: Mario		

The method solveRows() is used initially, and gives the first solution to each row of the matrix using the recursive method solveRows as follows:

```
private boolean solveRow(int row, int index) {
        Number currentNumber = variableData[row][index];
        if(index<variableData.length-1) {
            if(solveRow(row, index +1)) return true;
            for(int i = 0; i < 9; i++) {
                if(!currentNumber.isInmutable()) {
                    currentNumber.increase();
                    if(checkRow(row)) return true;
                    if(solveRow(row, index+1)) return true;
                }
            }
            if(index==0) currentNumber.increase();
            return false;
        }

        for(int i = 0; i < 9; i++) {
            if(currentNumber.isInmutable()) break;
            currentNumber.increase();
            if(checkRow(row)) return true;
        }
        return false;
    }
```

This method starts on the Number instance at the row index that is provided (the row it will solve) and at the index 0 in the first call. It goes recursively until the last index of the row, and start increasing the number nine times. Each time it increases it, it checks if the row is solved. If it is, it returns true through all the recursion trace. If the nine iterations pass, it will go back to the previous element and then increase it once, calling again the solve row for the next index.

This goes on until the row is solved.

Once all the rows are solved, the solve method checks if this is the correct solution. If it's not, it uses another recursive method, solveAll() that will finally solve the problem (or theoretically because I didn't manage to make it work, but I think that I have some idea about it doesn't working).

```
public String[][] solve() {
        solveRows();
        if(!checkIsSolved()) {
            solveAgain = true;
            if(!solveAll(0)) {
                System.out.println("Not working");
            }
        }
        return solutionFormat();
    }
```

```
private boolean solveAll(int rowIndex) {
        int[] initialRow = initialValues(variableData[rowIndex]);
        if(rowIndex == 0)
        if(rowIndex<size-1) {
            if(solveAll(rowIndex + 1)) return true;
        }
        solveRow(rowIndex,0);
        if(rowIndex<size-1) {
            if(solveAll(rowIndex+1)) return true;
        }
        while(!equalRows(initialRow, variableData[rowIndex])) {
            if(checkIsSolved()) return true;
            solveRow(rowIndex,0);
            if(rowIndex<size-1) {
                if(solveAll(rowIndex+1)) return true;
            }
        }
        return false;
    }
```

Like the solveRow() method, this solve all goes down the matrix until the last row (as in the solverRow() with the indexes), and when it arrives to the last one, it executes again the solve row method to find another row solution. Then all the checking occurs and, until it does not arrive to the initial solution of the row, it doesn't go back to the previous one. The previous one is solved once and then all the solutions of the last one are checked again to see if they solve the problem. This continues until the first row, and the solution should appear (It only does appear for the test1 method, test00.txt, for the others the method will return false and give valid solutions for the rows but not for the whole problem)

The methods that checks if the rows and columns are correct are checkRow() and checkCol(). Each of them computes a single row or column. checkRow() is only used in solve rows, but checkCol() is used in the checkIsSolved(), because it's supposed that the rows will always be correct.

Algorithmics	Student information	Date	Number of session
	UO: 295180	17/04/24	6
	Surname: Orviz Viesca		
	Name: Mario		

The operations are performed by the method performOperation() that takes a value (this value is really an accumulator from the previous operation) and perform the corresponding operation with the next value in the row or column (the three values are passed as arguments).

The division is managed checking if value%nextValue==0. If it's not, we know for sure that the operation is not valid and we can ensure that the row or column is wrong.

The method initialValues converts an array of Numbers into an array of int and the method equalRows() compares an array of ints with an array of Numbers.