Sudoku validator

Given a Sudoku board (complete or incomplete) check whether the current state of the board is valid

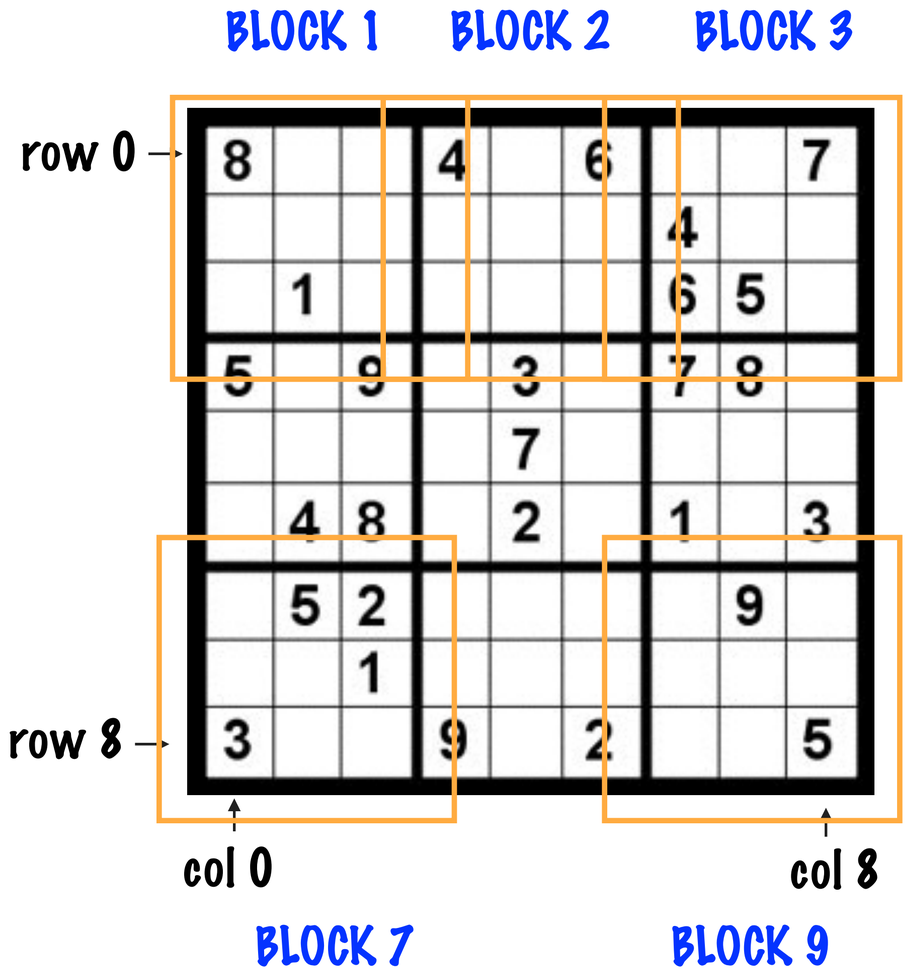
A Sudoko board is a 9x9 board which can hold numbers from 1-9. Any other number on that board is invalid.

For a Sudoku board to be valid
1. no row or column should have numbers 1-9 repeated
2. no designated 3x3 block within the board should have numbers 1-9 repeated

There are many edge cases and checks in this one, it is not an easy solution, make sure you hit all the conditions

HINT: Have a special state to represent the case where no number is entered in a cell yet, say -1

SUDOKU BOARD



NO ROW OR COLUMN SHOULD HAVE ANY OF THE NUMBERS FROM 1-9 REPEATED I.E EACH NUMBER IS PRESENT EXACTLY ONCE

NO 3X3 BLOCK SHOULD HAVE THE NUMBERS 1-9 PRESENT MORE THAN ONCE IN THAT BLOCK, EACH NUMBER CAN BE PRESENT EXACTLY ONCE

SUDOKU BOARD VALIDATOR

VALIDATE THAT THE ROWS AND COLUMNS HAVE 1-9 PRESENT ONLY ONCE, NO REPEATS

```java
/**
 * This checks whether a Sudoku board that is passed in is valid. A board is valid when no row
 * or column contain any of the numbers 1-9 more than once. Also every 3x3 group of 9 cells
 * should not contain any of the numbers from 1-9 more than once.
 * @param sudokuBoard
 * @return
 */
public static boolean isValid(int[][] sudokuBoard) {
    // Check whether all rows and columns have unique numbers 1-9. We will use a set to check
    // whether the number has been added before in that row or column.
    if (!isValidRowsAndColumns(sudokuBoard)) {
        return false;
    }

    // Now check each 3x3 block to see if the numbers between 1-9 are repeated within that block.
    // Assume the 3x3 blocks are numbered as follows
    // 0-1-2
    // 3-4-5
    // 6-7-8
    if (!isValidBlocks(sudokuBoard)) {
        return false;
    }

    // If we fall through here then all our checks indicate that the board is valid.
    return true;
}
```

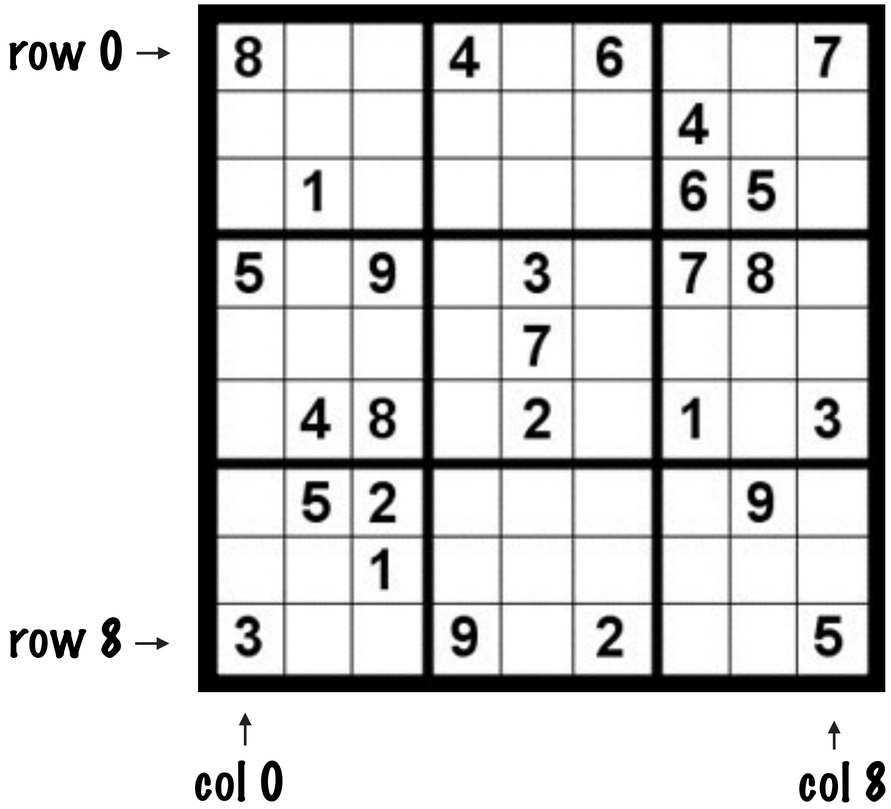
VALIDATE THAT EACH BLOCK HAS 1-9 PRESENT EXACTLY ONCE, NO REPEATS

IF BOTH VALIDATION CHECKS PASS THEN WE RETURN TRUE

ROW AND COLUMN VALIDATION

SET OF DIGITS SEEN IS STORED FOR EACH ROW AND COLUMN

HAVE 9 + 9 = 18 SETS INITIALISED WHICH STORES INFORMATION ABOUT EVERY ROW AND EVERY COLUMN



AS YOU ITERATE THROUGH EVERY CELL ADD THE VALUE TO THE SET ASSOCIATED WITH THAT ROW AND COLUMN

IF THE VALUE WAS ALREADY PRESENT IN THE SET IT MEANS THAT THE SUDOKU BOARD IS INVALID

VALIDATE ROWS AND COLUMNS

```java
private static boolean isValidRowsAndColumns(int[][] sudokuBoard) {
    // Set up a list of sets one for each row and one for each column.
    List<Set<Integer>> rowList = new ArrayList<Set<Integer>>();
    List<Set<Integer>> columnList = new ArrayList<Set<Integer>>();

    // Initialize a set associated with each row and column.
    for (int i = 0; i < 9; i++) {
        rowList.add(new HashSet<Integer>());
        columnList.add(new HashSet<Integer>());
    }

    for (int row = 0; row < 9; row++) {
        for (int col = 0; col < 9; col++) {
            // Get the value in that sudoku cell.
            int cellValue = sudokuBoard[row][col];
            // If no value has been assigned to a cell then continue, don't perform any checks.
            if (cellValue == -1) {
                continue;
            }
            if (cellValue < 1 || cellValue > 9) {
                return false;
            }

            // If the value has been seen in that row or column before return false.
            if (rowList.get(row).contains(cellValue)) {
                return false;
            }
            if (columnList.get(col).contains(cellValue)) {
                return false;
            }

            // Add the current cell value to the row or column set.
            rowList.get(row).add(cellValue);
            columnList.get(col).add(cellValue);
        }
    }
    return true;
}
```

THESE ARE THE SETS ASSOCIATED WITH EVERY ROW AND EVERY COLUMN WHICH TRACKS THAT THE NUMBERS ARE UNIQUE

NO VALUE HAS BEEN ASSIGNED SO CONTINUE TO NEXT CELL

GO THROUGH EVERY CELL IN THE SUDOKU

ANY VALUE OUTSIDE THE 1-9 RANGE MEANS THE BOARD IS INVALID

IF THE CURRENT CELL VALUE HAS BEEN SEEN IN THE CURRENT ROW OR COLUMN THE BOARD IS INVALID

ADD THE CURRENT CELL VALUE TO THE SETS KEEPING TRACK OF THE ROWS AND THE COLUMNS

BLOCK VALIDATION

HAVE ONE SET ASSOCIATED WITH EACH BLOCK WHICH KEEPS TRACK OF THE NUMBERS SEEN IN A BLOCK

BLOCK COL 0

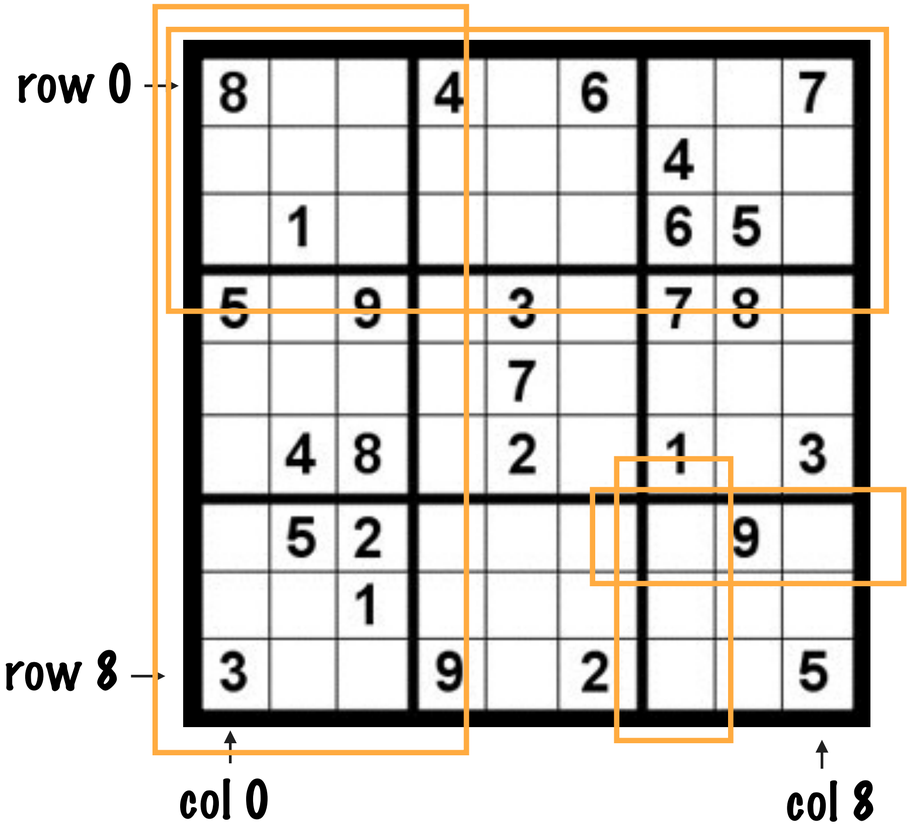
BLOCK COL 1

BLOCK COL 2

BLOCK ROW 0

BLOCK ROW 1

BLOCK ROW 2



BLOCK 9

- SAY BLOCKS ARE IN ROWBLOCKS 0-2 AND COLBLOCKS 0-2

- AND THE CELLS IN A BLOCK ARE NUMBERED AS MINIROW 0-2 AND MINICOL 0-2

THE ACTUAL ROW AND COLUMN FOR EACH CELL CAN BE GOT BY:

ROW = ROWBLOCK * 3 + MINIROW
COL = COLBLOCK * 3 + MINICOL

VALIDATE ROWS AND COLUMNS

```java
private static boolean isValidBlocks(int[][] sudokuBoard) {
    // Have an integer set associated with each to check whether a number in that cell
    // has occurred in the block before.
    List<Set<Integer>> blockList = new ArrayList<Set<Integer>>();
    for (int i = 0; i < 9; i++) {
        blockList.add(new HashSet<Integer>());
    }

    for (int rowBlock = 0; rowBlock < 3; rowBlock++) {
        for (int colBlock = 0; colBlock < 3; colBlock++) {
            // Here we iterate over the cells in each block.
            for (int miniRow = 0; miniRow < 3; miniRow++) {
                for (int miniCol = 0; miniCol < 3; miniCol++) {
                    // This calculation gives us the actual cell in the sodoku board.
                    // Since each block is a 3x3 block and the mini rows and columns are
                    // rows and columns in that block this moves us to the right row and
                    // the right cell within it.
                    int row = rowBlock * 3 + miniRow;
                    int col = colBlock * 3 + miniCol;

                    int cellValue = sudokuBoard[row][col];
                    // If no value has been assigned to a cell then continue, don't perform
                    // any checks.
                    if (cellValue == -1) {
                        continue;
                    }
                    if (cellValue < 1 || cellValue > 9) {
                        return false;
                    }
                    int blockNumber = rowBlock * 3 + colBlock;
                    if (blockList.get(blockNumber).contains(cellValue)) {
                        return false;
                    }
                    blockList.get(blockNumber).add(cellValue);
                }
            }
        }
    }
    return true;
}
```

THESE ARE THE SETS ASSOCIATED WITH EVERY BLOCK

GO THROUGH THE BLOCKROWS AND BLOCKCOLUMNS, A BLOCK ROW IS 3 ROWS PUT TOGETHER AND A BLOCK COLUMN IS 3 COLUMNS PUT TOGETHER

ITERATE THROUGH THE CELLS IN EACH BLOCK, A BLOCK IS A 3X3 MATRIX

NO VALUE HAS BEEN ASSIGNED SO CONTINUE TO NEXT CELL

GET THE ACTUAL ROW AND COLUMN OF THE CELL AS PER THE VALUE DISCUSSED

ANY VALUE OUTSIDE THE 1-9 RANGE MEANS THE BOARD IS INVALID

ADD THE CELL VALUE TO THE SET ASSOCIATED WITH THAT BLOCK

Increment number by 1

Suppose that you invent your own numeral system (which is neither decimal, binary nor any of the common ones). You specify the digits and the order of the digits in that numeral system.

Given the digits and the order of digits used in that system and a number, write a function to increment that number by 1 and return the result

Say that your numeric system comprises of the digits ‘A’, ‘B’, ‘C’ and ‘D’ in that order i.e. D comes after C which comes after B which comes after A. Let’s look at some examples inputs and their corresponding outputs of the function which increments the number by 1.

Input -> Output
ABA -> ABB
BAC -> BAD
CAD -> CBA

Note that incrementing “D” causes that digit to “wrap around” to the first digit which is “A”

HINT: Remember to handle the case where we have a number like DD incremented by 1

INCREMENT A NUMBER

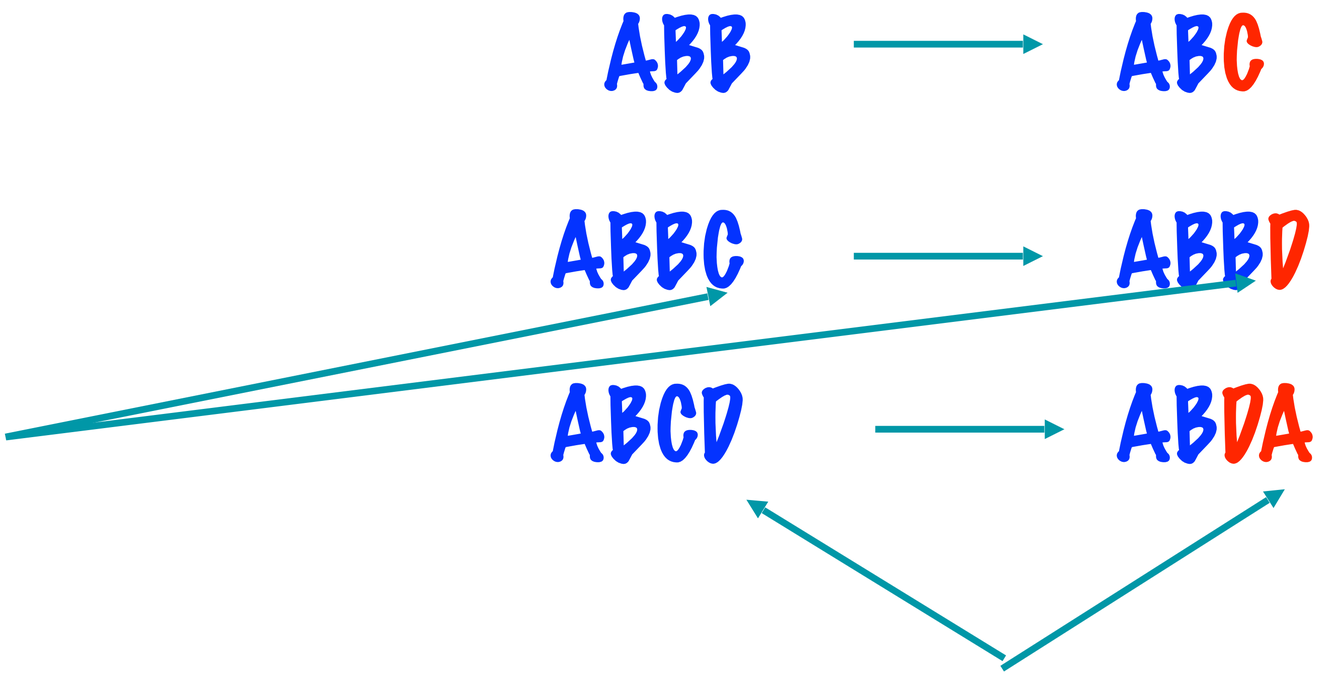
CONSIDER THAT THE NUMERAL SYSTEM THAT YOU SET UP IS:

A, B, C, D

THESE ARE THE ONLY VALID DIGITS IN A NUMBER AND

A < B < C < D

LET'S SEE SOME EXAMPLES OF HOW INCREMENTING NUMBERS IN THIS NUMERIC SYSTEM WORKS



INCREMENTING THE LEAST SIGNIFICANT DIGIT I.E "C" GIVES US THE NEXT DIGIT IN THE SEQUENCE WHICH IS "D"

"D" IS THE LAST VALID DIGIT SO THE NUMBER WRAPS AROUND TO "A" JUST LIKE 9 WRAPS AROUND TO 0 WHEN 1 IS ADDED IN THE DECIMAL SYSTEM

THE NEXT MOST SIGNIFICANT DIGIT NOW GET A "CARRY" AND HAS TO BE INCREMENT AS WELL SO "C" BECOMES "D"

INCREMENT NUMBER

USE CHARACTERS TO REPRESENT THE NUMBER

```
public static List<Character> increment(List<Character> originalNumber) {
    List<Character> incrementedNumber = new ArrayList<Character>();

    boolean incrementComplete = false;
    int currentIndex = originalNumber.size() - 1;
    incrementedNumber.addAll(originalNumber);

    while (!incrementComplete && currentIndex >= 0) {
        char currentDigit = originalNumber.get(currentIndex);
        int indexOfCurrentDigit = digitList.indexOf(currentDigit);

        int indexOfNextDigit = (indexOfCurrentDigit + 1) % digitList.size();

        incrementedNumber.remove(currentIndex);
        incrementedNumber.add(currentIndex, digitList.get(indexOfNextDigit));

        if (indexOfNextDigit != 0) {
            incrementComplete = true;
        }

        if (currentIndex == 0 && indexOfNextDigit == 0) {
            incrementedNumber.add(0, digitList.get(0));
        }

        currentIndex--;
    }


    return incrementedNumber;
}
```

START WITH THE LEAST SIGNIFICANT DIGIT WHICH IS THE LAST CHARACTER IS THE LIST

WE GO THROUGH THE NUMBER DIGIT BY DIGIT TILL WE GET TO THE MOST SIGNIFICANT DIGIT AT INDEX 0 OF THE LIST

INITIALIZE THE VARIABLE WHICH HOLDS THE INCREMENTED NUMBER TO BE THE SAME AS THE ORIGINAL

GET THE POSITION OF THE LEAST SIGNIFICANT DIGIT IN THE DIGIT LIST WHICH HAS THE NUMBERS IN THE ASCENDING ORDER

GET THE NEXT DIGIT ON INCREMENT, THIS WILL WRAP AROUND TO THE FIRST DIGIT WHICH IS WHY WE USE THE MODULO OPERATOR

IF THE NEXT DIGIT DID NOT WRAP AROUND WE'RE DONE! WE CAN EXIT THE LOOP, OTHERWISE CONTINUE INCREMENTING THE NEXT MOST SIGNIFICANT DIGIT

UPDATE THE CURRENT DIGIT TO BE THE INCREMENTED VALUE

IF WE'RE AT THE MOST SIGNIFICANT DIGIT AND THAT WRAPPED AROUND WE ADD A NEW DIGIT TO THE INCREMENTED NUMBER LIKE GOING FROM 9->10