**פתרון מועד X**

**שאלה 1 (20 נקודות):**

נתונה חתימת הפונקציה הרקורסיבית הבאה:

**public** **static** **void** mergeArrays(**int**[] a, **int** len1,   
 **int**[] b, **int** len2,   
 **int**[] c)

הפונקציה מקבלת מערך **ממוין בסדר עולה** a ואורכו len1, מערך **ממוין בסדר עולה** b ואורכו len2 וכן מערך c שאורכו כסכום שני האורכים של a ו- b.

השלימו את גוף הפונקציה באופן **רקורסיבי** כך שתשים את ערכי מערך a וערכי מערך b בתוך מערך c באופן ממוין בסדר עולה.

הנחיה: אין להשתמש בפתרון בלולאות

**פתרון:**

**public** **static** **void** mergeArrays(**int**[] a, **int** len1,

**int**[] b, **int** len2,

**int**[] c) {

**if** (len1 == 0 && len2 == 0)

**return**;

**if** (len1 == 0)

{

c[len2-1] = b[len2-1];

*mergeArrays*(a, len1, b, len2-1, c);

}

**else** **if** (len2 == 0)

{

c[len1-1] = a[len1-1];

*mergeArrays*(a, len1-1, b, len2, c);

}

**else** {

**if** (a[len1-1] > b[len2-1]) {

c[len1+len2-1] = a[len1-1];

*mergeArrays*(a, len1-1, b, len2, c);

}

**else** {

c[len1+len2-1] = b[len2-1];

*mergeArrays*(a, len1, b, len2-1, c);

}

}

}

**שאלה 2 (40 נקודות):**

נתונה מטריצה M של מספרים: מערך דו-ממדי מלבני (אורכי כל השורות זהים).

נגדיר "**שלשה**" כרצף של 3 ערכים צמודים זה לזה באותה שורה

* כלומר, עבור שורה row, ועמודה col, שלשה של ערכים הינה ערכי התאים:  
   [row][col] , [row][col+1] , [row][col+2]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 8 | 1 | 2 | 0 |
| 1 | 4 | 5 | **0** | 7 |
| 2 | 8 | 3 | 3 | 7 |
| 3 | 0 | 1 | 7 | **7** |

* דוגמאות עבור המטריצה הבאה:
* רצף הערכים 1,2,0 הינו שלשה של ערכים צמודים זה לזה (מקומות: [0][**1**] , [0][**2**] , [0][**3**] )
* רצף הערכים 8,3,3 הינו שלשה של ערכים צמודים זה לזה (מקומות: [2][**0**] , [2][**1**] , [2][**2**] )

נגדיר "**שלשה טובה**" כשלשה המקיימת את התנאים הבאים:

* ערך ראשון בשלשה – משמש כאינדקס לשורה במטריצה
* ערך שני בשלשה – משמש כאינדקס לעמודה במטריצה
* ערך שלישי בשלשה – שווה לערך במקום ששני הערכים הראשונים מתייחסים עליו
  + כלומר מתקיים:   
      <ערך שלישי במערך> = [ <ערך שני במערך> ][ <ערך ראשון במערך> ]M
* במטריצה הנ"ל, לדוגמה, יש 2 **"שלשות טובות"**:
  + השלשה 1,2,0 בשורה 0. כי הערך במקום M[1][2] הוא 0 – והוא שווה לערך השלישי שבמקום M[0][3]
  + השלשה 3,3,7 בשורה 2. כי הערך במקום M[3][3] הוא 7 – והוא שווה לערך השלישי שבמקום M[2][3]
* דוגמאות לשלשות **לא** **טובות**:
* השלשה 8,3,3 בשורה 2. שימו לב, הערך 8 אמור לשמש כאינדקס שורה, ואין שורה כזו במערך
* השלשה 0,1,7 בשורה 3. הערך השלישי בשלשה הוא 7, וערך זה לא שווה לערך במקום M[0][1], שהוא 1.

**סעיף א' (15 נק'):**

ממשו את הפונקציה הבאה:

**public** **static** **boolean** isValidTriple(**int**[][] mat, **int** startRow, **int** startCol)

הפונקציה מקבלת מערך דו-מימדי של מספרים אינדקס שורה ואינדקס עמודה. הפונקציה תחזיר true אם החל ממיקום זה במטריצה זוהי שלשה טובה, אחרת הפונקציה תחזיר false.

הפונקציה תחזיר false במקרה בו מהאינדקסים שהתקבלו לא ניתן לבדוק שלושה ערכים צמודים.

**סעיף ב' (15 נק'):**

ממשו את הפונקציה הבאה:

**public** **static** **boolean[][]** findAllGoodTriples(**int**[][] mat)

הפונקציה מקבלת מערך דו-מימדי של מספרים אשר אורכי שורותיו שווים.  
הפונקציה תחזיר מערך דו-מימדי בוליאני שגודלו כגודל המטריצה שהתקבלה, ועבור כל מיקום במטריצה שהתקבלה שהוא תחילת שלשה טובה, יושם הערך true באיבר המתאים במטריצה המוחזרת.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | false | **true** | false | false |
| 1 | false | false | false | false |
| 2 | false | **true** | false | false |
| 3 | false | false | false | false |

עבור הדוגמה הנ"ל תחזור המטריצה הבוליאנית הבאה:

שכן במיקומים [0][1] ו- [2][1] במטריצה המקורית מתחילה שלשה טובה.

הנחיה: יש להשתמש בפונקציה בסעיף א'. במידה ולא מימשתם אותה במלואה/באופן נכון, עדיין ניתן להשתמש בה בסעיף זה כבקופסה שחורה.

**סעיף ג' (10 נק'):**

ממשו את הפונקציה הבאה:

**public** **static** **int** writeTriplesToFile(**int**[][] mat, String name) **throws** IOException

הפונקציה מקבלת מטריצה, ומחרוזת המכילה של קובץ. הפונקציה תכתוב לקובץ את כל השלשות הטובות וכן תחזיר את מספר השלשות הכתובות בקובץ.

ניתן להניח ששם הקובץ המתקבל נמצא במיקום אשר קיים במחשב.

דוגמה:

עבור המטריצה:



הקובץ שיווצר הוא:



והפונקציה תחזיר 5 כי יש 5 שלשות טובות.

**פתרון:**

**public** **static** **boolean** isValidTriple(**int**[][] mat, **int** startRow, **int** startCol) {

**int** firstTripleValue = mat[startRow][startCol];

**int** secondTripleValue = mat[startRow][startCol+1];

**int** thirdTripleValue = mat[startRow][startCol+2];

// check if row is valid

**if** (firstTripleValue < 0 || firstTripleValue >= mat.length)

**return** **false**;

// check if col is valid

**if** (secondTripleValue < 0 || secondTripleValue >= mat[0].length)

**return** **false**;

**return** mat[firstTripleValue][secondTripleValue] == thirdTripleValue;

}

**public** **static** **boolean**[][] findAllGoodTriples(**int**[][] mat) {

**boolean** res[][] = **new** **boolean**[mat.length][mat[0].length];

**for** (**int** i = 0; i < mat.length; i++) {

**for** (**int** j = 0; j < mat[0].length - 2 ; j++) {

**if** (*isValidTriple*(mat, i, j))

res[i][j] = **true**;

}

}

**return** res;

}

**public** **static** **int** writeTriplesToFile(**int**[][] mat, String name) **throws** IOException {

**int** total = 0;

File file = **new** File(name);

PrintWriter writer = **new** PrintWriter(file);

**boolean**[][] goodTriplesLocation = *findAllGoodTriples*(mat);

**for** (**int** i = 0; i < mat.length; i++) {

**for** (**int** j = 0; j < mat[i].length ; j++) {

**if** (goodTriplesLocation[i][j]) {

writer.println(mat[i][j] + " " + mat[i][j + 1] + " " + mat[i][j + 2]);

total++;

}

}

}

writer.close();

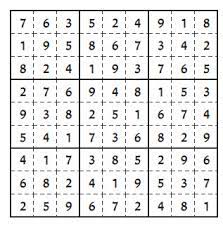
**return** total;

}

**שאלה 3 (40 נקודות):**

**"לוח סודוקו תקין"** הוא לוח בגודל 9X9 שבכל שורה ובכל עמודה ישנם את כל הערכים 1 עד 9.  
בנוסף, כאשר מחלקים את הלוח ל- 9 חלקים שכל אחד בגודל 3X3 כפי שמוצג בתמונה, גם בכל חלק כזה יש את כל הערכים 1 עד 9.

להלן דוגמה ללוח סודוקו תקין:



ממשו את הפונקציה הבאה:

**public** **static** **boolean** isValidSudoku(**int**[][] board)

הפונקציה המקבלת מערך דו-מימדי של מספרים שלמים (ניתן להניח שהוא בגודל 9X9)

הפונקציה תחזיר true אם הערכים בלוח מהווים לוח סודוקו תקין, אחרת הפונקציה תחזיר false.

הנחיה: הקפידו על מודולריות!

לנוחיותכם נתונה פונקציה המדפיסה את הלוח בצורת לוח סודוקו.

**פתרון:**

**public** **class** Sudoku {

**public** **static** **boolean** isValidSudoku(**int**[][] board) {

**boolean** isCorrectBoard = **true**;

// check 3X3 squares

**for** (**int** row = 0; row < 9 && isCorrectBoard; row += 3)

**for** (**int** col = 0; col < 9 && isCorrectBoard ; col += 3)

isCorrectBoard = *sudokuOneSquareCheck*(board, row, col);

// check the rows

**for** (**int** row = 0; row < 9 && isCorrectBoard; row++)

isCorrectBoard = *sudokuRowCheck*(board[row]);

// check the columns

**for** (**int** col = 0; col < 9 && isCorrectBoard; col++)

isCorrectBoard = *sudokuColumnCheck*(board, col);

**return** isCorrectBoard;

}

**public** **static** **boolean** arrayAllTrue(**boolean**[] arr) {

**for** (**int** i = 1; i < arr.length; i++) {

**if** (arr[i] == **false**)

**return** **false**;

}

**return** **true**;

}

**public** **static** **boolean** sudokuSquareCheck(**int**[][] board, **int** row, **int** col) {

**boolean**[] allNumbers = **new** **boolean**[10];

**for** (**int** cRow = row; cRow < row + 3; cRow++)

**for** (**int** cCol = col; cCol < col + 3; cCol++)

allNumbers[board[cRow][cCol]] = **true**;

**return** *arrayAllTrue*(allNumbers);

}

**public** **static** **boolean** sudokuRowCheck(**int**[] boardRow) {

**boolean**[] allNumbers = **new** **boolean**[10];

**for** (**int** cCol = 0; cCol < 9; cCol++)

allNumbers[boardRow[cCol]] = **true**;

**return** *arrayAllTrue*(allNumbers);

}

**public** **static** **boolean** sudokuColumnCheck(**int**[][] board, **int** col) {

**boolean**[] allNumbers = **new** **boolean**[10];

**for** (**int** cRow = 0; cRow < 9; cRow++)

allNumbers[board[cRow][col]] = **true**;

**return** *arrayAllTrue*(allNumbers);

}

**public** **static** **boolean** sudokuOneSquareCheck(**int**[][] board, **int** row, **int** col) {

**boolean**[] allNumbers = **new** **boolean**[10];

**for** (**int** i = row ; i < row+3 ; i++) {

**for** (**int** j = col; j < col+3 ; j++)

allNumbers[board[i][j]] = **true**;

}

**return** *arrayAllTrue*(allNumbers);

}

**public** **static** **void** printSudokuBoard(**int**[][] board) {

**for** (**int** i=0 ; i < board.length ; i++) {

**if** (i > 0 && i%3 == 0) {

**for** (**int** j=0 ; j < 7\*3 ; j++)

System.***out***.print("-");

System.***out***.println();

}

**for** (**int** j=0 ; j < board[i].length ; j++) {

**if** (j > 0 && j%3 == 0)

System.***out***.print(" |");

System.***out***.print(" " + board[i][j]);

}

System.***out***.println();

}

}

**public** **static** **void** main(String[] args) {

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

{ 8, 7, 6, 2, 1, 9, 3, 4, 5 }, { 4, 3, 2, 7, 6, 5, 8, 9, 1 }, { 7, 6, 5, 1, 9, 8, 2, 3, 4 },

{ 1, 9, 8, 4, 3, 2, 5, 6, 7 }, { 3, 2, 1, 6, 5, 4, 7, 8, 9 }, { 6, 5, 4, 9, 8, 7, 1, 2, 3 },

{ 9, 8, 7, 3, 2, 1, 4, 5, 6 } };

*printSudokuBoard*(sudokuBoard);

System.***out***.println("\nIs the board valid? " + *isValidSudoku*(sudokuBoard));

}

}