**פתרון מועד Y**

**שאלה 1 (20 נקודות):**

נתונה חתימת הפונקציה הרקורסיבית הבאה:

**public** **static** **void** reduceFraction(**int**[] fraction)

הפונקציה מקבלת מערך מספרים שלמים בגודל 2 המייצג שבר עשרוני, כך שהערך במקום ה- 0 הוא המונה והערך במקום ה- 1 הוא המכנה.

דוגמה: השבר ¾ ייוצג במערך כך שבמקום ה- 0 יהיה הערך 3 ובמקום ה- 1 יהיה הערך 4.

השלימו את גוף הפונקציה באופן **רקורסיבי** כך שתעדכן את מערך הפרמטר שיהיה השבר המצומצם ביותר.

דוגמאות:

Before: 18/30, after: 3/5

Before: 10/60, after: 1/6

Before: 24/96, after: 1/4

Before: 24/100, after: 6/25

**פתרון:**

**public** **static** **void** reduceFractionRec(**int**[] fraction) {

**boolean** canReduceMore = **false**;

**for** (**int** i = 2; i <= fraction[0] && !canReduceMore; i++) {

**if** (fraction[0] % i == 0 && fraction[1] % i == 0)

canReduceMore = **true**;

}

**if** (!canReduceMore)

**return**;

**for** (**int** i = 2; i <= fraction[0] ; i++) {

**if** (fraction[0] % i == 0 && fraction[1] % i == 0) {

fraction[0] /= i;

fraction[1] /= i;

*reduceFractionRec*(fraction);

}

}

}

**שאלה 2 (40 נקודות):**

**סעיף א' (10 נק'):**

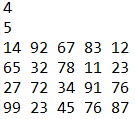
ממשו את הפונקציה הבאה:

**public** **static** **int**[][] readMatrixFromFile(String fileName)

הפונקציה מקבלת שם של קובץ טקסט ובו נתוני מטריצה באופן הבא: מספר המייצג את מספר שורות מטריצה, מספר המייצג את מספר עמודות המטריצה ואז כל ערכי המטריצה.

דוגמה:

הקובץ



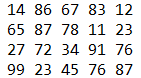
מחזיק נתוני מטריצה עם 4 שורות שכל שורה 5 עמודות ולאחריו כל איברי המטריצה.

**סעיף ב' (15 נק'):**

נגדיר "שכן דומיננטי" ערך במטריצה שגדול מכל שמונת שכניו. שימו לב שערך בדפנות המטריצה אינו יכול להיות שכן חזק.

דוגמה:

עבור המטריצה הבאה:



87 ו- 91 הם "שכנים דומיננטיים".

כתבו את הפונקציה הבאה:

**public** **static** **boolean** isDominantNeighbour(**int**[][] mat, **int** row, **int** cols)

הפונקציה המקבלת מטריצה ומיקום במטריצה ומחזירה true אם הערך במיקום זה הוא "שכן דומיננטי", אחרת הפונקציה תחזיר false.

הגבלה: ניקוד מלא יתקבל רק קוד שיהיה בו משפט if יחיד.

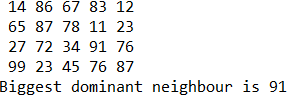
ניתן להניח ש- row ו- col הינם מיקום תקין במטריצה לצורך בדיקת "שכן דומיננטי", ואין צורך לבדוק בעיה של חריגה מגבולות המערך

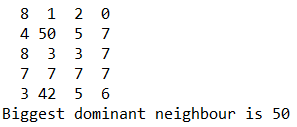
**סעיף ג' (15 נק'):**

**public** **static** **int** getBiggestDominantNeighbour(**int**[][] mat)

הפונקציה מקבלת מטריצה של מספרים שלמים חיוביים ומחזירה את השכן הדומיננטי הגדול ביותר. במידה ואף איבר אינו שכן דומיננטי הפונקציה תחזיר 0.

דוגמאות:



’

הנחיות כלליות:

* בפתרון סעיף זה יש להשתמש בפתרון סעיף ב', גם אם לא פתרתם אותו במלואו
* באחריות הפונקציה לוודא שלא תהיה חריגה מגבולות המערך

**פתרון:**

**public** **static** **int** getBiggestDominantNeighbour(**int**[][] mat) {

**int** maxDominantNeighbour = 0;

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

**for** (**int** j = 1; j < mat[0].length - 1; j++) {

**boolean** isDominant = *isDominantNeighbour*(mat, i, j);

**if** (isDominant) {

**if** (mat[i][j] > maxDominantNeighbour)

maxDominantNeighbour = mat[i][j];

}

}

}

**return** maxDominantNeighbour;

}

**public** **static** **boolean** isDominantNeighbour(**int**[][] mat, **int** row, **int** cols) {

**for** (**int** i = -1; i <= 1; i++) {

**for** (**int** j = -1; j <= 1; j++) {

**if** (mat[row][cols] < mat[row + i][cols + j]) {

**return** **false**;

}

}

}

**return** **true**;

}

**public** **static** **int**[][] readMatrixFromFile(String fileName) **throws** FileNotFoundException {

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

Scanner scan = **new** Scanner(file);

**int**[][] mat = **new** **int**[scan.nextInt()][scan.nextInt()];

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

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

mat[i][j] = scan.nextInt();

}

}

scan.close();

**return** mat;

}

**שאלה 3 (40 נקודות):**

איבר במערך דו-מימדי מלבני (מספר עמודות זהה בכל השורות) של מספרים שלמים נקרא "**מלך האלכסונים"** אם מתקיימים 2 התנאים הבאים (**בו זמנית**):

* ערכו הינו הערך הגדול ביותר באלכסון היוצא ממנו – "שמאלה-למעלה" ו- "ימינה-למטה"
* ערכו הינו הערך הגדול ביותר באלכסון היוצא ממנו – "ימינה-למעלה" ו- "שמאלה-למטה"

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 63 | **30** | 1 | 79 | 50 | 61 | 59 |
| 1 | 16 | 18 | **28** | 92 | 84 | 77 | **29** |
| 2 | 95 | 22 | 37 | **32** | 41 | **44** | 100 |
| 3 | 36 | 19 | 67 | 34 | **87** | 49 | 14 |
| 4 | 54 | 2 | 15 | **8** | 35 | **65** | 31 |

**דוגמה**:

המערך משמאל הינו מערך בגודל של 5x7.

הערך 87, במקום [3][4] הוא "מלך האלכסונים" כי הינו הערך הגדול ביותר באלכסון היוצא ממנו "שמאלה-למעלה" ו- "ימינה-למטה", כלומר, הערכים במקומות: [0][1], [1][2], [2][3], [3][4], [4][5].

כמו כן, הינו גם הערך הגדול ביותר באלכסון היוצא ממנו "ימינה-למעלה" ו- "שמאלה-למטה", כלומר, הערכים במקומות: [1][6], [2][5], [3][4], [4][3].

הערך 100 במקום [2][6] הינו "מלך האלכסונים" כי הוא הערך הגדול ביותר מהאלכסונים היוצאים ממנו:

* אלכסון ראשון: [0][4], [1][5], [2][6] (במקרה זה אין "ימינה-למטה")
* אלכסון שני: [2][6], [3][5],[4][4] (במקרה זה אין "ימינה-למעלה")

**סעיף א' (20 נק'):**

כתבו את הפונקציה הבאה:

**public** **static** **boolean** isKing(**int**[][] arr, **int** row, **int** col)

הפונקציה מקבלת מערך דו-ממדי מלבני ושני ערכים שורה/עמודה – המייצגים מיקום במערך.

הפונקציה תבדוק האם הערך במיקום הנתון – הינו "מלך האלכסונים".

הפונקציה תחזיר true במידה וכן, אחרת יוחזר הערך false.

**יש לממש את הפתרון כך שסיבוכיות זמן הריצה של הפונקציה תהיה O(n).** פתרון אשר סיבוכיות זמן הריצה שלו יהיה איטי יותר מ O(n) – **יורדו מהפתרון 10 נקודות**.

**סעיף ב' (20 נק'):**

כתבו את הפונקציה הבאה:

**public** **static** **void** printAllKings(**int**[][] arr)

הפונקציה מקבלת מערך דו-ממדי מלבני.

הפונקציה תסרוק את המערך ותאתר את כל הערכים/מיקומים בהם יש "מלך האלכסונים". הפונקציה תדפיס עבור כל "מלך אלכסונים" את ערכו, ואת מיקומו (שורה/עמודה) (ראו בהמשך כיצד הפלט צריך להיראות).

בנוסף, הפונקציה תדפיס הודעה המציינת את מספר התאים בהם יש "מלך אלכסונים".

לבסוף הפונקציה תדפיס את הערך של "מלך מלכי האלכסונים" – הערך הגבוה ביותר מבין ערכי "מלך האלכסונים" שנמצאו.

על פונקציה זו לבצע את משימתה בעזרת הפונקציה שנכתבה בסעיף א'.

להלן פלט לדוגמה של התוכנית, עבור המערך הדו-ממדי הבא:

Enter array size, rows and cols: 5 7

96 81 56 82 61 13 24

40 6 54 85 90 41 18

52 29 17 77 67 69 91

10 7 1 39 95 79 30

49 28 12 43 86 35 36

The found "diagonal king" and their indexes:

96 --> [0][0]

85 --> [1][3]

90 --> [1][4]

91 --> [2][6]

95 --> [3][4]

There are 5 kings.

The king of "diagonals' king" is: 96

**פתרון**

**public** **static** **boolean** isKing(**int**[][] arr, **int** row, **int** col) {

**int** cRow, cCol;

**boolean** aKing = **true**;

// check main diagonal

cRow = Math.*max*(row - col, 0);

cCol = Math.*max*(col - row, 0);

**while** (cRow < arr.length && cCol < arr[cRow].length && aKing) {

**if** (arr[row][col] < arr[cRow][cCol])

aKing = **false**;

cRow++;

cCol++;

}

// check secondary diagonal

cRow = Math.*max*(row - (arr[row].length - 1 - col), 0);

cCol = Math.*min*(col + row, arr[row].length - 1);

**while** (cRow < arr.length && cCol >= 0 && aKing) {

**if** (arr[row][col] < arr[cRow][cCol])

aKing = **false**;

cRow++;

cCol--;

}

**return** aKing;

}

**public** **static** **void** printAllKings(**int**[][] arr) {

**int** count = 0, kingOfKings = -1;

System.***out***.println("The found \"diagonal king\" and their indexes:");

**for** (**int** row = 0; row < arr.length; row++) {

**for** (**int** col = 0; col < arr[row].length; col++) {

**if** (*isKing*(arr, row, col)) {

System.***out***.printf("%d --> [%d][%d]\n", arr[row][col], row, col);

kingOfKings = Math.*max*(kingOfKings, arr[row][col]);

count++;

}

}

}

System.***out***.println("There are " + count + " kings.");

System.***out***.println("The king of \"diagonals' king\" is: " + kingOfKings);

}