**פתרון מועד Y**

**שאלה 1 (40 נקודות):**

בשאלה זו נשמור מאגר שירים: לכל שיר נשמור את שמו, מי האמן שכתב אותו ומי האמן האחרון שביצע אותו. לכל אמן נשמור שמו, את רשימת השירים שכתב ואת רשימת השירים שביצע.

להלן תיאור של מחלקות הקיימות במערכת ומחלקות **שעליכם להוסיף או לעדכן**, בהמשך יוצג main לדוגמה ופלט תוצאת הרצתו.

1. נתונה בהמשך באופן **חלקי** המחלקה עבור אמן (Artist):
   1. תכונות המחלקה: שם האמן, מערך השירים שכתב ומערך השירים שביצע.
   2. קונסטרקטור המקבל את שם האמן ומאתחל את שני המערכים להיות בגודל 0.
   3. מתודת getName המחזירה את שם האמן.
   4. נתונה מעטפת המתודה hasWroteSong המקבלת שיר ומחזירה true האם האמן ביצע שיר זה, אחרת תחזיר false. ניתן להניח כי המימוש של המתודה תקין ועובד, וניתן להשתמש בו (**לא נדרש לממש מתודה זו!**).
   5. נתונה מעטפת המתודה addWrittenSong המקבלת שיר ומנסה להוסיף הפניה שלו למערך השירים שהאמן כתב, ומחזירה true אם השיר הוסף למערך, אחרת תחזיר false. ניתן להוסיף שיר למערך רק אם לא קיים במערך השירים שנכתבו ע"י האמן שיר אחר עם שם זהה. במידה ומערך השירים מלא, המתודה מגדילה את גודלו פי 2. ניתן להניח כי המימוש של המתודה תקין ועובד, וניתן להשתמש בו (**לא נדרש לממש מתודה זו!**).
   6. **(4 נק') הוסיפו** את המתודה hasPerformedSong המקבלת שיר ומחזירה true אם האמן ביצע שיר עם שם זה, אחרת תחזיר false.
   7. **(6 נק') הוסיפו** את המתודה addPerformedSong המקבלת שיר ומנסה להוסיף הפניה שלו לאוסף השירים שהאמן ביצע. המתודה מחזירה true אם השיר הוסף למערך, אחרת תחזיר false. ניתן להוסיף שיר למערך השירים שבוצעו ע"י האמן רק אם לא קיים במערך השירים שביצע שיר אחר עם שם זהה. במידה ומערך השירים מלא, המתודה מגדילה את גודלו ב- 1. כמו כן, יש לדאוג לעדכן עבור השיר שאמן זה הוא האחרון שביצע שיר זה (ראו התייחסות בהמשך במחלקה Song). לצורך מימוש סעיף זה ניתן להשתמש במתודות מהמחלקה Arrays.
   8. **(5 נק') הוסיפו** את המתודה toString המחזירה מחרוזת עם נתוני האמן כפי שמופיע בפלט בהמשך.
2. נתונה באופן **חלקי** מחלקה עבור שיר (Song):
   1. תכונות המחלקה הן שם השיר, הפניה לאמן שכתב אותו והפניה לאמן האחרון שביצע אותו.
   2. **(7 נק') כתבו** את קונסטרקטור המחלקה המקבל את שם השיר, את האמן שכתב אותו ואת האמן האחרון שביצע אותו (שדה זה יכול להיות null). המתודה מאתחלת את נתוני האובייקט ותדאג גם לדברים הבאים: עבור האמן שכתב אותו, להוסיף את השיר לרשימת השירים שנכתבו על-ידו, ובדומה, עבור האמן שביצע אותו, להוסיף את השיר לרשימת השירים שבוצעו על ידו (במידה ואכן הועבר אומן מבצע).
   3. **(6 נק') כתבו** את המתודה setLastPerformer המקבלת אמן (יכול להיות גם null) ומעדכנת אותו כאמן האחרון שביצע את השיר, וכן מעדכנת עבור האמן את שיר זה ברשימת השירים שהוא ביצע.
   4. נתונה המתודה getName המחזירה את שם השיר.
   5. נתונה המתודה toString המחזירה מחרוזת עם נתוני המשתמש.
3. נתונה באופן **חלקי** מחלקה עבור מאגר השירים (SongsRepository):
   1. תכונות המחלקה: מערך האמנים ומספר האיברים מתוך המערך שבשימוש (גודל לוגי), מערך השירים ומספר האיברים מתוך המערך שבשימוש (גודל לוגי). שני מערכים אלו יגדלו תמיד פי 2 כאשר יגמר בהם המקום.
   2. מוגדר במחלקה טיפוס enum בשם eAddSongStatus אשר מגדיר ערכי הצלחה/כשלון בעת הוספת שיר למאגר.
   3. נתון קונסטרקטור המאתחל את המערכים לגודל התחלתי של 2.
   4. נתונות המתודות getArtistByName ו- getSongByName המקבלות מחרוזת המייצגת שם ומחזירות הפניה לאמן/שיר בהתאמה עם שם זה. במידה ולא קיים אמן/שיר עם שם זה, יוחזר null.
   5. נתונה המתודה addArtist המקבלת שם של אמן, ובמידה והוא עדיין אינו במאגר, מוסיפה אותו ומחזירה true, אחרת מחזירה false. המתודה מוודאת שיש מספיק מקום במערך האמנים, ובמידה שאין מגדילה אותו ב- 1.
   6. **(12 נק') הוסיפו** את המתודה addSong המקבלת שם של שיר, את שם האמן שכתב את השיר ואת שם האמן שביצע אחרון את השיר. במידה ואין אמן שביצע את השיר, תועבר לפרמטר זה מחרוזת ריקה. המתודה תנסה להוסיף את השיר למאגר ותצליח רק אם השיר עדיין אינו נמצא במאגר, האמן שכתב את השיר קיים במאגר והאמן שביצע אחרון את השיר (במידה והועבר) קיים במאגר.

המתודה תוודא שאכן קיים במאגר אמן עם שם האמן שכתב את השיר, תוודא שאכן קיים במאגר אמן עם שם האמן האחרון שביצע את השיר (במידה והועברה מחרוזת שאינה ריקה עבור האמן האחרון שביצע את השיר) ותבדוק שיש מספיק מקום במאגר להוספת השיר החדש (ותגדיל במידת הצורך).

לבסוף כשהכל תקין, תוסיף את השיר למאגר. כמו כן, המתודה תדאג להוסיף את השיר לאמן שכתב את השיר למערך השירים שלו אותם כתב, וכן במידה ויש אמן אחרון שביצע את השיר, להוסיף את השיר למערך השירים אותם ביצע.  
המתודה תחזיר ערך מהטיפוס eAddSongStatus בהתאם לאופי הכישלון או ההצלחה.

* 1. נתונה המתודה toString המחזירה מחרוזת עם נתוני המאגר.

להלן main לדוגמה:

**public** **class** Program {

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

SongsRepository repository = **new** SongsRepository();

repository.addArtist("gogo");

repository.addArtist("momo");

repository.addSong("Corona Hey!", "gogo", "gogo");

repository.addSong("Antigen sucks", "gogo", "momo");

repository.addSong("PCR rocks!", "gogo", "momo");

repository.addSong("Be Healthy Choopie-Doop", "momo", "");

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

System.***out***.println("Repository after change:");

repository.getSongByName(  
 "Corona Hey!").setLastPerformer(repository.getArtistByName("momo"));

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

}

}

להלן פלט עבור main זה:

The artists in the repository:

The artist gogo wrote 3 songs:

Corona Hey!

Antigen sucks

PCR rocks!

and performed these songs:

Corona Hey!

The artist momo wrote 1 songs:

Be Healthy Choopie-Doop

and performed these songs:

Antigen sucks

PCR rocks!

The songs in the repository:

The song 'Corona Hey!' was written by gogo and was last performed by gogo

The song 'Antigen sucks' was written by gogo and was last performed by momo

The song 'PCR rocks!' was written by gogo and was last performed by momo

The song 'Be Healthy Choopie-Doop' was written by momo

Repository after change:

The artists in the repository:

The artist gogo wrote 3 songs:

Corona Hey!

Antigen sucks

PCR rocks!

and performed these songs:

Corona Hey!

The artist momo wrote 1 songs:

Be Healthy Choopie-Doop

and performed these songs:

Antigen sucks

PCR rocks!

Corona Hey!

The songs in the repository:

The song 'Corona Hey!' was written by gogo and was last performed by momo

The song 'Antigen sucks' was written by gogo and was last performed by momo

The song 'PCR rocks!' was written by gogo and was last performed by momo

The song 'Be Healthy Choopie-Doop' was written by momo

**השלימו את המחלקה Artist עפ"י הנדרש בהגדרות למעלה:**

**public** **class** Artist {

**private** String name;

**private** Song[] writtenSongs;

**private** Song[] performedSongs;

**public** Artist(String name) {

**this**.name = name;

writtenSongs = **new** Song[0];

performedSongs = **new** Song[0];

}

**public** String getName() {

**return** name;

}

**public** **boolean** hasWroteSong(Song theSong) {

// this function returns true if the artists already has a song   
// with that name in hist written songs list

}

**public** **boolean** addWrittenSong(Song newSong) {

// this functions a reference adds the song if it doesn’t appear yet  
// in the list, and returns *true* if it succeed, else returns *false*.

// if the array is full, it enlarges it by \*2

...

}

**public** **boolean** addPerformedSong(Song newSong) {

**if** (hasPerformedSong(newSong))

**return** **false**;

performedSongs = Arrays.*copyOf*(performedSongs, performedSongs.length+1);

performedSongs[performedSongs.length-1] = newSong;

newSong.setLastPerformer(**this**);

**return** **true**;

}

**public** **boolean** hasPerformedSong(Song theSong) {

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

**if** (performedSongs[i].getName().equals(theSong.getName()))

**return** **true**;

}

**return** **false**;

}

@Override

**public** String toString() {

StringBuffer sb = **new** StringBuffer("The artist " + name + " wrote "   
 + writtenSongs.length + " songs:\n");

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

sb.append("\t" + writtenSongs[i].getName()+ "\n");

sb.append("and performed these songs: \n");

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

sb.append("\t" + performedSongs[i].getName()+ "\n");

**return** sb.toString();

}

}

**השלימו את המחלקה Song עפ"י הנדרש בהגדרות למעלה:**

**public** **class** Song {

**private** String name;

**private** Artist writer;

**private** Artist lastPerformer;

**public** Song(String name, Artist writer, Artist lastPerformer) {

**this**.name = name;

**this**.writer = writer;

**this**.writer.addWrittenSong(**this**);

**this**.lastPerformer = lastPerformer;

**if** (**this**.lastPerformer != **null**)

**this**.lastPerformer.addPerformedSong(**this**);

}

**public** **void** setLastPerformer(Artist lastPerformer) {

**if** (lastPerformer!= **null** && **this**.lastPerformer != lastPerformer) {

**this**.lastPerformer = lastPerformer;

**this**.lastPerformer.addPerformedSong(**this**);

}

}

**public** String getName() {

**return** name;

}

@Override

**public** String toString() {

StringBuffer sb = **new** StringBuffer("The song '" + name   
 + "' was written by " + writer.getName());

**if** (lastPerformer != **null**)

sb.append(" and was last performed by "   
 + lastPerformer.getName());

**return** sb.toString();

}

}

**השלימו את המחלקה SongsRepositoryעפ"י הנדרש בהגדרות למעלה:**

**public** **class** SongsRepository {

**public** **enum** eAddSongStatus {

***Success***, ***SongAlreadyExist***, ***WriterDoesntExist***, ***PerformerDoesntExist***

};

**private** Artist[] allArtists;

**private** **int** numOfArtists;

**private** Song[] allSongs;

**private** **int** numOfSongs;

**public** SongsRepository() {

allArtists = **new** Artist[2];

numOfArtists = 0;

allSongs = **new** Song[2];

numOfSongs = 0;

}

**public** Artist getArtistByName(String name) {

**for** (**int** i = 0; i < numOfArtists; i++) {

**if** (allArtists[i].getName().equals(name))

**return** allArtists[i];

}

**return** **null**;

}

**public** Song getSongByName(String name) {

**for** (**int** i = 0; i < numOfSongs; i++) {

**if** (allSongs[i].getName().equals(name))

**return** allSongs[i];

}

**return** **null**;

}

**public** **boolean** addArtist(String name) {

Artist temp = getArtistByName(name);

**if** (temp == **null**) {

**if** (numOfArtists == allArtists.length) {

allArtists = Arrays.*copyOf*(allArtists, allArtists.length\*2);

}

allArtists[numOfArtists++] = **new** Artist(name);

**return** **true**;

}

**return** **false**;

}

**public** eAddSongStatus addSong(String songName, String writerName,   
 String performerName) {

Song temp = getSongByName(songName);

**if** (temp != **null**)

**return** eAddSongStatus.***SongAlreadyExist***;

Artist writer = getArtistByName(writerName);

**if** (writer == **null**)

**return** eAddSongStatus.***WriterDoesntExist***;

Artist performer = **null**;

**if** (!performerName.equals("")) {

performer = getArtistByName(performerName);

**if** (performer == **null**)

**return** eAddSongStatus.***PerformerDoesntExist***;

}

**if** (numOfSongs == allSongs.length) {

allSongs = Arrays.*copyOf*(allSongs, allSongs.length \* 2);

}

allSongs[numOfSongs] = **new** Song(songName, writer, performer);

writer.addWrittenSong(allSongs[numOfSongs]);

**if** (performer != **null**)

performer.addPerformedSong(allSongs[numOfSongs]);

numOfSongs++;

**return** eAddSongStatus.***Success***;

}

@Override

**public** String toString() {

StringBuffer sb = **new** StringBuffer("The artists in the repository:\n");

**for** (**int** i = 0; i < numOfArtists; i++)

sb.append(allArtists[i].toString() + "\n");

sb.append("The songs in the repository:\n");

**for** (**int** i = 0; i < numOfSongs; i++)

sb.append(allSongs[i].toString() + "\n");

**return** sb.toString();

}

}

**שאלה 2 (25 נקודות):**

**חלק א' (10 נקודות)**

להלן פונקציה רקורסיבית:

**public** **static** **boolean** foo(**int** num) {

**if** (num < 10)

**return** **true**;

**if** (num < 100)

**return** num % 10 == num / 10;

String strNum = num + "";

**if** (strNum.charAt(0) != strNum.charAt(strNum.length() - 1))

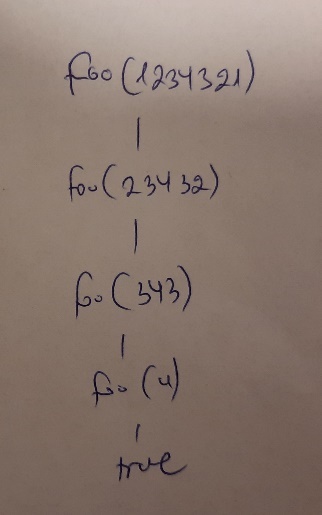
**return** **false**;

**return** *foo*(Integer.*parseInt*(strNum.substring(1, strNum.length() - 1)));

}

1. **(7 נקודות)** ציירו עץ מעקב עבור הפונקציה foo עבור num=1234321 וכתבו מה הפונקציה תחזיר עבור זימון הפונקציה עם ערכים אלו.
2. **(3 נקודות)** כתבו במשפט מה הפונקציה עושה. (שימו לב, אין להסביר מהן הפקודות המבוצעות או לתת דוגמת הרצה - כלומר אין להסביר את ה"איך").

**פתרון:**

****

הפונקציה מקבלת מספר ומחזירה true אם המספר פלינדרום, אחרת הפונקציה תחזיר false.

**חלק ב' (15 נקודות)**

כתבו את הפונקציה הרקורסיבית הבאה:

**public** **static** **boolean** checkIfNumExistsExactly(**int**[] arr, **int** size,

**int** num, **int** count)

הפונקציה מקבלת מערך של מספרים, מספר המעיד על מספר האיברים במערך ושני מספרים נוספים: num ו- count.

הפונקציה תחזיר true אם המספר num מופיע במערך בדיוק count פעמים, אחרת תחזיר false.

דוגמאות:

עבור ה- main הבא עם הזימונים הבאים, יוחזרו התשובות המוצגות בהערה בסיום כל שורה:

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

System.***out***.println(*checkIfNumExistsExactly*(**new** **int**[] {1, 4, 7, 1, 2, 1}, 6, 1, 3 )); // true

System.***out***.println(*checkIfNumExistsExactly*(**new** **int**[] {1, 4, 7, 1, 2, 1}, 6, 1, 2 )); // false

System.***out***.println(*checkIfNumExistsExactly*(**new** **int**[] {1, 4, 7, 1, 2, 1}, 6, 8, 3 )); // false

System.***out***.println(*checkIfNumExistsExactly*(**new** **int**[] {1, 4, 7, 1, 2, 1}, 6, 8, 0 )); // true

}

**פתרון:**

**public** **static** **boolean** checkIfNumExistsExactly(**int**[] arr, **int** size,

**int** num, **int** count) {

**if** (size == 0)

**return** count == 0;

**if** (arr[size - 1] == num)

**return** *checkIfNumExistsExactly*(arr, size - 1, num, count - 1);

**else**

**return** *checkIfNumExistsExactly*(arr, size - 1, num, count);

}

**שאלה 3 (35 נקודות):**

**סעיף א' (15 נק'):**

כתבו את הפונקציה הבאה:

**public** **static** **int** biggestSquareWithSameValueFromPos(**int**[][] matrix, **int** row, **int** col)

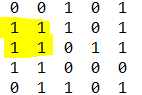
הפונקציה מקבלת מטריצה של מספרים ושני מספרים שלמים row ו- col המייצגים תא במטריצה שמיקומו באינדקסים [row][col].

הפונקציה תבדוק מהו מימד **הריבוע** הגדול ביותר שמתחיל במיקום [row][col] וערכיו זהים.

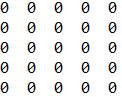
שימו לב: המטריצה יכולה להיות בכל מימד NXM.

דוגמה:

עבור המטריצה הבאה והמיקום [0][1] יוחזר 2 כי החל מאינדקס זה ישנו ריבוע שערכיו זהים בגודל 2X2.



עבור המטריצה הבאה והמיקום [0][0] יוחזר 5 כי החל מאינדקס זה ישנו ריבוע שערכיו זהים בגודל 5X5.



הערה חשובה: שימו לב בפתרון לא לגרום לתעופה עקב חריגה מגבולות המטריצה.

**סעיף ב' (15 נק'):**

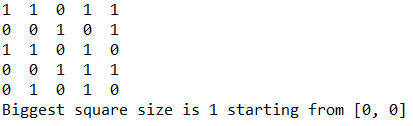
כתבו את הפונקציה הבאה:

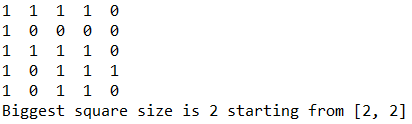
**public** **static** **int** biggestSquareWithSameValue(**int**[][] matrix, **int**[] res)

הפונקציה מקבלת מטריצה של מספרים ומערך פלט של מספרים בגודל 2 ומחזירה מספר שלם.

הפונקציה תבדוק החל מאיזה אינדקס במטריצה מתחיל ריבוע מספרים עם ערכים זהים בעל המימד הגדול ביותר ותעדכן את האינדקסים במערך res (אינדקס השורה יהיה במיקום ה- 0 במערך ואינדקס העמודה יהיה במיקום ה- 1 במערך). הפונקציה תחזיר את מימד הריבוע הגדול ביותר שערכיו זהים.

דוגמאות:





**סעיף ג' (5 נק'):**

נתונות הפונקציות הבאות שאתם מוזמנים להשתמש בהן בפתרון שלכם בהמשך

**private** **static** **void** printMatrix(**int**[][] matrix) {

**...**

}

**private** **static** **void** randomMatrix(**int**[][] matrix) {

**...**

}

כתבו את גוף ה- main כך שיגדיר מטריצה של מספרים בגודל 5X5, יגריל לתוכה ערכים וידפיס אותה.  
ה- main יקרא לפונקציה שהגדרתם בסעיף ב' ולבסוף ידפיס הודעה כמו בפלט שהוצג בסעיף ב'.

**פתרון:**

**public** **static** **int** biggestSquareWithSameValueFromPos(**int**[][] matrix, **int** row, **int** col) {

**int** size = 1;

**while** (row+size <= matrix.length && col+size <= matrix[0].length) {

**for** (**int** i = 0; i < size && row + i < matrix.length; i++) {

**for** (**int** j = 0; j < size && col + j < matrix[0].length; j++) {

**if** (matrix[row + i][col + j] != matrix[row][col]) {

**return** size-1;

}

}

}

size++;

}

**return** size-1;

}

**public** **static** **int** biggestSquareWithSameValue(**int**[][] matrix, **int**[] res) {

**int** size = 0;

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

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

**int** curSize = *biggestSquareWithSameValueFromPos*(matrix, i, j);

**if** (curSize > size) {

size = curSize;

res[0] = i;

res[1] = j;

}

}

}

**return** size;

}

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

**int**[][] matrix = **new** **int**[5][5];

*randomMatrix*(matrix);

*printMatrix*(matrix);

**int**[] res = **new** **int**[2];

**int** size = *biggestSquareWithSameValue*(matrix, res);

System.***out***.println("Biggest square size is " + size + " starting from [" + res[0] + ", " + res[1] + "]");

}