**קראו בעיון את ההנחיות שלהלן:**

* במבחן יש חמש שאלות. עליכם לענות על **כולן**.
* כל התכניות צריכות להיות מתועדות היטב. יש לכתוב תחילה **בקצרה** את האלגוריתם וכל הסבר נוסף הדרוש להבנת התכנית. יש לבחור בשמות משמעותיים למשתנים, לשיטות ולקבועים שבתכנית. תכנית שלא תתועד כנדרש לעיל תקבל לכל היותר 85% מהניקוד. **אפשר לתעד בעברית.** אין צורך בתיעוד API.
* יש להקפיד לכתוב את התכניות בצורה מבנית ויעילה. **תכנית לא יעילה לא תקבל את מלוא הנקודות.**
* **אם ברצונכם להשתמש בתשובתכם בשיטה או במחלקה הכתובה בחוברת השקפים, אין צורך שתעתיקו את השיטה או את המחלקה למחברת הבחינה. מספיק להפנות למקום הנכון, ובלבד שההפניה תהיה מדויקת (פרמטרים, מיקום וכו').**
* **אין להשתמש במחלקות קיימות ב- Java, חוץ מאלו המפורטות בשאלות הבחינה.**
* **יש לשמור על סדר; תכנית הכתובה בצורה בלתי מסודרת עלולה לגרוע מהציון.**
* בכתיבת התכניות יש להשתמש **אך ורק** במרכיבי השפה שנלמדו בקורס זה. **אין להשתמש במשתנים גלובליים!**

**כל התשובות צריכות להיכתב בתוך קובץ המבחן במקומות המתאימים בלבד. תשובה שתיכתב שלא במקומה לא תיבדק.**

שאלה 1 (25 נקודות)

השאלה הבאה מדמה משחק "תפזורת".

נתון מערך דו-ממדי ריבועי המלא בתווים (char). נגדיר שמחרוזת תווים קיימת במערך אם כל תווי המחרוזת נמצאים כסדרם במערך, ואפשר להגיע מתו אחד לתו שאחריו בסדרה על-ידי מעבר מתא לאחד מארבעת שכניו (ימין, שמאל, למעלה, למטה). לא באלכסון.

כך לדוגמא, אם המערך הוא זה:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T | Z | x | c | D |
| **S** | **H** | **a** | z | X |
| H | W | **l** | **o** | **M** |
| O | R | n | t | N |
| A | B | r | i | N |

המחרוזת "shalom" קיימת ומסומנת במערך. מתחילה בתא (1,0) ומסתיימת בתא (2,4).

כתבו שיטה **סטטית רקורסיבית** המקבלת מערך שכזה וכן מחרוזת המייצגת מילה. השיטה בודקת אם המילה קיימת במערך או לא. **אם המילה קיימת במערך, עליכם להדפיס את "הנתיב" שלה**.

ההדפסה תיעשה בעזרת מטריצת מעקב שתגדירו, שהיא בגודלה של המטריצה המקורית, ובתאים שלה יהיו מספרים שלמים. (בהמשך השאלה יש הסבר מדויק לגבי ההדפסה.) בסוף השיטה, במטריצת המעקב צריכים להיות בתאים של המחרוזת, מספרי התווים המרכיבים את המחרוזת, ובתאים שאין בהם תוים של המחרוזת יהיו אפסים.

לדוגמא, עבור המערך שלעיל והמחרוזת "shalom" מטריצת המעקב תהיה:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 0 | 0 |
| 0 | 0 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

אם המילה אינה קיימת יש להדפיס למסך "No Such Word". אם המילה מופיעה יותר מפעם אחת, עליכם להדפיס את אחד ה"נתיבים" בלבד. לא משנה איזה. לא ניתן להשתמש באותו תו יותר מפעם אחת ב"נתיב", כלומר לא ניתן לחזור לתא יותר מפעם אחת.

חתימת השיטה היא:

**public static void findWord(char [][] arr, String word)**

**שימו לב:** ניתן להניח שהמערך מכיל אותיות קטנות בלבד וגם במחרוזת יש אותיות קטנות בלבד. אין צורך לבדוק זאת.

**לגבי הדפסת הנתיב** – עליכם ליצור את מטריצת המעקב, ואז לקרוא לשיטה בשם printArr המקבלת כפרמטר מערך דו-מימדי ומדפיסה אותו. **אינכם צריכים לממש את השיטה printArr בעצמכם.**

השיטה צריכה להיות רקורסיבית ללא שימוש בלולאות כלל. כך גם כל שיטות העזר שתכתבו (אם תכתבו) לא יכולות להכיל לולאות.

אפשר להשתמש בהעמסת-יתר (overloading).

**מותר לשנות את המערך במהלך השיטה, אבל בסופה הוא צריך לחזור למצבו המקורי.**

**אין צורך לדאוג ליעילות השיטה, אבל כמובן שצריך לשים לב לא לעשות קריאות רקורסיביות מיותרות!**

**אל תשכחו לתעד את מה שכתבתם!**

**את התשובה עליכם לכתוב בעמודים הבאים.**

**התשובה היא:**

אלגוריתם:

אני עובר ברקורסיה על גבי המערך של התווים מ0,0 עד לתא האחרון I,j כאשר אני בודק האם התא הנוכחי הוא האות הראשונה של המילה שהתקבלה(word), כאשר אני מוצא אותה אני קורא לפונקציה נוספת אשר מתחילה לבדוק מהמיקום של האות הראשונה שמצאתי האם המחרוזת קיימת על פי החוקים של השאלה, כלומר האם בתזוזה אפשרית(ימינה,למטה,שמאלה,למעלה) האות הבאה במחרוזת נמצאת ועבור כל תנועה אני קורא רקורסיבית לאותה הפונקציה רק עם תו הבא במחרוזת. כאשר התרחשו כל הקריאות הרקורסיביות והגעתי לאות האחרונה אני מחזיר TRUE וזה מתגלגל אחורה עד לקריאה של הפונקציה שמצאה את האות הראשונה והיא מחזירה TRUE אם מצאתי את כל המילה על פי החוקיות של השאלה.

public static void findWord(char[][] arr, String word)

{

//אם המערך ריק או המילה ריקה

if(arr.length == 0 || arr[0].length == 0 || word.length() == 0)

{

System.out.println("No Such Word");

}

int[][] track = new int[arr.length][arr[0].length];

if (findStart(arr, word, 0, 0, track))

{

printArr(track);

}

else

{

System.out.println("No Such Word");

}

}

public static boolean findStart(char[][] arr, String word, int i, int j, int[][] track)

{

if(j == arr[0].length)

{

return findStart(arr, word, i+1, 0, track);

}

if(word.charAt(0) == (arr[i][j]))

{

//אם מצאתי את האות הראשונה של המילה במערך אני בודק האם היא מצליחה להשלים את כל המילה

if(findWord(arr, word, i, j, track, 0))

{

return true;

}

}

//אם לא מצאתי את המילה והגעתי למיקום האחרון במערך

if(i==arr.length - 1 && j == arr[0].length - 1)

{

return false;

}

return findStart(arr, word, i, j+1, track);

}

public static boolean findWord(char[][] arr, String word, int i, int j, int[][] track, int indexStr)

{

boolean right = false;

boolean down = false;

boolean left = false;

boolean up = false;

//אם זאת לא האות שאני מחפש

if(arr[i][j] != word.charAt(indexStr))

{

return false;

}

//אם הגעתי לאות האחרונה

if(indexStr == word.length() - 1)

{

track[i][j] = indexStr + 1;

return true;

}

//right

if(canMove(arr,i,j+1))

{

char temp = arr[i][j];

arr[i][j] = 'A';

right = findWord(arr,word, i, j+1,track, indexStr + 1);

arr[i][j] = temp;

if(right)

{

track[i][j] = indexStr + 1;

}

}

//down

if(canMove(arr,i+1,j))

{

char temp = arr[i][j];

arr[i][j] = 'A';

down = findWord(arr,word, i+1, j,track, indexStr + 1);

arr[i][j] = temp;

if(down)

{

track[i][j] = indexStr + 1;

}

}

//left

if(canMove(arr,i,j-1))

{

char temp = arr[i][j];

arr[i][j] = 'A';

left = findWord(arr,word, i, j-1,track, indexStr + 1);

arr[i][j] = temp;

if(left)

{

track[i][j] = indexStr + 1;

}

}

//up

if(canMove(arr,i-1,j))

{

char temp = arr[i][j];

arr[i][j] = 'A';

up = findWord(arr,word, i, j+1,track, indexStr + 1);

arr[i][j] = temp;

if(up)

{

track[i][j] = indexStr + 1;

}

}

return right || down || up || left;

}

public static boolean canMove(char[][] arr, int i, int j)

{

if( (i == arr.length || j == arr[0].length || i == -1 || j == -1))

{

return false;

}

return true;

}

שאלה 2 (25 נקודות)

המשפט היסודי של האריתמטיקה אומר כי **אפשר לפרק כל מספר טבעי (כלומר שלם חיובי) לגורמים ראשוניים, והפירוק הזה הוא יחיד.** (סדר הגורמים אינו משנה)

למשל, את המספר 40 ניתן לפרק לגורמים ראשוניים הבאים:

2 \* 2 \* 2 \* 5 = 23 \* 51 = 40.

המספרים 2 ו-5 הם מספרים ראשוניים. (כזכור, מספר ראשוני הוא מספר שמתחלק רק ב-1 ובעצמו ללא שארית).

שימו לב שאין שום דרך אחרת לכתוב את המספר הזה בתור מכפלת ראשוניים. לכן הפירוק הזה הוא יחיד.

**חלק מהמספרים** הם מכפלה של **שני מספרים ראשוניים בלבד**. לא כמו המספר 40 לעיל, שהגורם הראשוני שלו 2, מופיע 3 פעמים במכפלה.

**לדוגמא**,

* המספר 35 הוא מכפלה של 5 ו- 7.
* המספר 493 הוא מכפלה של 17 ו- 29.
* המספר 8509 הוא מכפלה של 67 ו- 127.

הניחו כי **נתונה לכם** השיטה הבוליאנית הבאה: **אין צורך לממש את השיטה הזו.**

public boolean isHighFactorInRange(int n, int low, int high)

שיטה זאת מקבלת מספר טבעי n ו-2 מספרים low ו- high ,אשר מייצגים שני קצוות של טווח חיפוש. **אנו מניחים, כי n הוא מכפלה של בדיוק שני מספרים ראשוניים p ו-q.** השיטה בודקת האם הגורם הראשוני הגדול מבין q ,p נמצא בטווח החיפוש. אם כן היא מחזירה true , אחרת היא מחזירה false (טווח החיפוש **כולל את הקצוות**(.

**לדוגמא**,

אנחנו יודעים כי המספר 8509 הוא מכפלה של 67 ו- 127. לכן, אם נקרא לשיטה isHighFactorInRange עם המספר n = 8509, וטווח החיפוש 1 – 100 (low =1, high = 100), השיטה תחזיר false. כיון שהגורם הראשוני הגדול יותר מבין 67 ו- 127 הוא 127, והוא לא נמצא בטווח בין 1 ל- 100. אם היינו נותנים טווח low = 1, high = 200, השיטה היתה מחזירה true, כי 127 הוא בין 1 ל-200.

שימו לב שהפרמטרים לשיטה לא כוללים את הגורמים p ו- q עצמם. רק את המספר n ואת טווח החיפוש. זכרו ש-n הוא מאותם מספרים שהם **מכפלה של** **בדיוק שני מספרים ראשוניים** p ו- q.

**עליכם לכתוב שיטה בשם findFactors שתקבל מספר n שהוא שלם חיובי והוא מכפלה של 2 מספרים ראשוניים, p ו- q, (q \* p = n). השיטה תמצא ותדפיס את p ו- q.**

חתימת השיטה:

public void findFactors (int n)

**לדוגמה**, עבור n=8509 השיטה תדפיס 67 ו-127 מאחר ו- 67\*127=8509.

כמובן שכדאי לכם מאד להשתמש בשיטה isHighFactorInRange הנתונה לכם.

**להזכירכם, אינכם צריכים לממש את השיטה isHighFactorInRange, אלא רק להשתמש בה.**

אין צורך להתייחס לסיבוכיות של השיטה isHighFactorInRange, אתם יכולים להניח שהשיטה יעילה. זה לא משנה מהי הסיבוכיות שלה. אתם צריכים להתייחס רק לסיבוכיות של השיטה findFactors שאתם כותבים.

**שימו לב:**

**השיטה findFactors שתכתבו צריכה להיות יעילה ככל הניתן, גם מבחינת סיבוכיות הזמן וגם מבחינת סיבוכיות המקום. תשובה שאינה יעילה מספיק כלומר, שתהיה בסיבוכיות גדולה יותר מזו הנדרשת לפתרון הבעיה תקבל מעט נקודות בלבד.**

**מה סיבוכיות זמן הריצה וסיבוכיות המקום של השיטה שכתבתם? הסבירו תשובתכם.**

**אל תשכחו לתעד את מה שכתבתם!**

**את התשובה עליכם לכתוב בעמודים הבאים:**

**התשובה היא:**

סיבוכיות הזמן שלי הוא o(log(n)) כי אני מבצע פה חיפוש בינארי שידוע שסיבוכיות שלו היא o(log(n)) עד שאני מוצא את אחד הגורמים, וכך אני יכול למצוא את הגורם השני בכך שאני מחלק את N בגורם הראשון שנמצא

סיבוכיות המקום שלי היא o(1) כי אני לא מקצה פה זיכרון על פי N מסוים אלא רק שלושה משתנים שכל אחד מהם הוא o(1)

האלגוריתם שלי הוא לבצע חיפוש בינארי כאשר החיפוש מחפש את אחד הגורמים ואת השני אפשר למצוא על ידי חלוקה של N בגורם הראשון.

public static void findFactors(int n)

{

int low = 0;

int high = n;

int mid;

while(low<high)

{

mid = (low + high)/2;

if(isHighFactorInRange(n, low, mid))

{

high = mid;

}

else if(isHighFactorInRange(n, mid + 1, high))

{

low = mid + 1;

}

}

System.out.println(n / low + ", " + low);

}

שאלה 3 (17 נקודות)

נניח שהמחלקה Node שלהלן מממשת צומת עץ בינרי שערכיו הם מספרים שלמים (int).

**שימו לב שבנוסף לשני המצביעים לבן השמאלי ולבן הימני, יש לכל אובייקט במחלקה Node גם שדה \_father שמצביע על האב של הצומת.**

**בשורש העץ, המצביע לאב הוא null.**

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Node**](file:///H:\Java\%D7%A7%D7%95%D7%A8%D7%A1%20%D7%9E%D7%91%D7%95%D7%90%20%D7%97%D7%93%D7%A9\2020%D7%90\%D7%91%D7%97%D7%99%D7%A0%D7%95%D7%AA\Node.html#Node(int))(int num)            Constructs a Node object. |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| [Node](file:///H:\Java\%D7%A7%D7%95%D7%A8%D7%A1%20%D7%9E%D7%91%D7%95%D7%90%20%D7%97%D7%93%D7%A9\2020%D7%90\%D7%91%D7%97%D7%99%D7%A0%D7%95%D7%AA\Node.html) | get[**Father**](file:///H:\Java\%D7%A7%D7%95%D7%A8%D7%A1%20%D7%9E%D7%91%D7%95%D7%90%20%D7%97%D7%93%D7%A9\2020%D7%90\%D7%91%D7%97%D7%99%D7%A0%D7%95%D7%AA\Node.html#getRightSon())()           Returns the father of the node. |
| [Node](file:///H:\Java\%D7%A7%D7%95%D7%A8%D7%A1%20%D7%9E%D7%91%D7%95%D7%90%20%D7%97%D7%93%D7%A9\2020%D7%90\%D7%91%D7%97%D7%99%D7%A0%D7%95%D7%AA\Node.html) | [**getLeftSon**](file:///H:\Java\%D7%A7%D7%95%D7%A8%D7%A1%20%D7%9E%D7%91%D7%95%D7%90%20%D7%97%D7%93%D7%A9\2020%D7%90\%D7%91%D7%97%D7%99%D7%A0%D7%95%D7%AA\Node.html#getRightSon())()           Returns the left son of the node. |
| int | [**getData**](file:///H:\Java\%D7%A7%D7%95%D7%A8%D7%A1%20%D7%9E%D7%91%D7%95%D7%90%20%D7%97%D7%93%D7%A9\2020%D7%90\%D7%91%D7%97%D7%99%D7%A0%D7%95%D7%AA\Node.html#getNumber())()           Returns the value of the node. |
| [Node](file:///H:\Java\%D7%A7%D7%95%D7%A8%D7%A1%20%D7%9E%D7%91%D7%95%D7%90%20%D7%97%D7%93%D7%A9\2020%D7%90\%D7%91%D7%97%D7%99%D7%A0%D7%95%D7%AA\Node.html) | [**getRightSon**](file:///H:\Java\%D7%A7%D7%95%D7%A8%D7%A1%20%D7%9E%D7%91%D7%95%D7%90%20%D7%97%D7%93%D7%A9\2020%D7%90\%D7%91%D7%97%D7%99%D7%A0%D7%95%D7%AA\Node.html#getRightSon())()           Returns the right son of the node. |

המחלקה BinaryTree מאגדת בתוכה שיטות סטטיות לטיפול **בעץ חיפוש בינרי**.

נתונות **חמש** השיטות הבאות: f , g , what, secret ו- something המקבלות כפרמטר שורש של **עץ חיפוש בינרי** (מטיפוס Node ):

**public static Node f(Node root)**

{

if (root.getRightSon() == null)

return root;

return f (root.getRightSon());

}

**private static Node g (Node node)**

{

if (node.getFather() == null)

return null;

if (node.getFather().getRightSon() == node)

return node.getFather();

return g(node.getFather());

}

**public static Node what (Node root)**

{

if (root == null)

return null;

if (root.getLeftSon() != null)

return f (root.getLeftSon());

return g (root);

}

**public static boolean secret(Node root)**

{

if (root == null)

return true;

Node temp1 = what(root);

if (temp1 == null)

return true;

Node temp2 = what(temp1);

if (temp2 == null)

return true;

if (root.getData() != temp1.getData() + temp2.getData())

{

System.out.println(root.getData() + " != " +

temp1.getData() + " + " + temp2.getData());

return false;

}

return secret(temp1);

}

**public static boolean something(Node root)**

{

return secret (f(root));

}

**סעיף א (2 נקודות):**

מה מבצעת השיטה **f** כאשר היא מקבלת שורש של **עץ חיפוש בינרי** כלשהו? הסבירו בקצרה **מה** מבצעת השיטה ולא **כיצד** היא מבצעת זאת. מה משמעותו של הערך שהוחזר? **התייחסו למקרי קצה**

**התשובה היא:**

כאשר הפונקציה מקבלת את שורש עץ חיפוש בינארי, הפונקציה מחזירה את הצומת הימני ביותר, וכאשר מדובר בעץ חיפוש בינארי הצומת הכי ימנית היא גם הצומת בעלת הערך המקסימלי בעץ. כאשר לעץ אין צד ימני, יוחזר השורש עצמו.

**סעיף ב (3 נקודות):**

מה מבצעת השיטה **what** כאשר היא מקבלת **צומת כלשהו** **בעץ חיפוש בינרי** כלשהו? הסבירו בקצרה **מה** מבצעת השיטה ולא **כיצד** היא מבצעת זאת. מה משמעותו של הערך שהוחזר? שימו לב שהפרמטר המועבר אינו בהכרח שורש העץ. **התייחסו למקרי קצה. אפשר להניח שהצומת נמצא בעץ.**

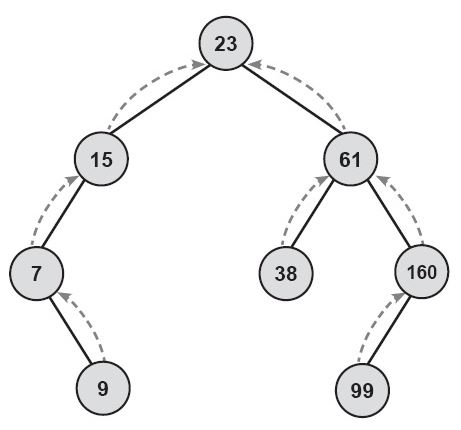
**התשובה היא:**

הפונקציה what מבצעת כאשר היא מקבלת צומת כלשהו בעץ חיפוש בינארי זה להחזיר NULL במקרה שקיבלה NULL כצומת, כאשר לצומת הנוכחית יש בן שמאלי אז מחזירה את הצומת הכי ימנית של הבן השמאלי שלו, ובמידה ולא קיים בן שמאלי אז מחזירה את האבא הראשון בהירכיה כלפי מעלה בעץ, של הצומת שהתקבלה כפרמטר, שהבן הימני שלה הוא או הצומת שהתקבלה או אחד מהאבות שלה או NULL אם לא נמצא אחד כזה עד שורש העץ. המשמעות של הערך שהוחזר היא הצומת בעלת הערך הגדול ביותר שקטן מהערך של הצומת שהתקבלה או NULL במידה ולא קיימת אחת כזאת

**סעיף ג (3 נקודות):**

בהינתן עץ חיפוש בינרי ששורשו root המצויר להלן:

root



איזה ערך תחזיר השיטה **something** ומה היא תדפיס על הפלט, אם נקרא לה עם הפרמטר root שהוא שורשו של העץ לעיל?

**התשובה היא:**

השיטה something תחזיר FALSE ותדפיס:

23 != 15 + 9

**סעיף ד (3 נקודות):**

איזה שינוי **מינימלי** אפשר לעשות **על העץ** לעיל, כדי שהקריאה לשיטה something (root) תחזיר ערך אחר מזה שהוחזר בסעיף ג? **שימו לב, השינוי צריך להיות בעץ ולא בשיטה (כגון, הוספת צומת, הורדת צומת, שינוי ערך של צומת וכד'). מינימלי במובן של מינימום פעולות על העץ.** כך, אם מורידים צומת, ויש לצומת הזה בנים, במספר הפעולות נספרים גם הבנים של הצומת שהורדו.

**התשובה היא:**

השינוי המינמאלי שנצטרך לעשות הוא לשנות את הצומת שערכה הוא 9 לערך 8 וכך יוחזר לנו ערך חדש מSECRET שהוא TRUE

**סעיף ה (6 נקודות):**

מה מבצעת השיטה **something** באופן כללי כשהיא מקבלת כפרמטר שורש של **עץ חיפוש בינרי** root? שימו לב, עליכם לתת תיאור ממצה של **מה עושה השיטה באופן כללי**, ולא תיאור של מה עושה כל שורה בשיטה, או איך היא מבצעת זאת. **כלומר, מה המשמעות של הערך שהשיטה מחזירה? התייחסו למקרי קצה.**

**התשובה היא:**

הפונקציה something כשהיא מקבלת כפרמטר שורש של עץ חיפוש בינארי מחזירה במידה וקיבלה שורש שהוא NULL או עץ בעל פחות משלושה צמתים TRUE, כאשר בעץ שקיבלה יש יותר משני צמתים (כלומר שלושה ומעלה צמתים כולל השורש) הפונקציה מחזירה האם הצומת בעלת הערך הגדול ביותר בעץ שווה לסכום שני המספרים הקטנים הראשונים ממנו(הכי קטנים שקרובים לערך שלו) בעץ וכך הלאה למספר הראשון שקטן מן המספר הגדול ביותר, כלומר אם המספר הגדול ביותר בעץ היה 160 ושני המספרים הקטנים הראשונים ממנו הם 99 ו61 אז הפונקציה ממשיכה לבדוק האם מתקיים הדבר גם עבור 99 (שהוא המספר הקטן מן 160 אך הגדול מבין 61 ו 99), כלומר האם 99 שווה לסכום שני הערכים של הצמתים הראשונים הקטנים ממנו.

שאלה 4 (15 נקודות)

**נתונות המחלקות One, Two, Three, Four, Five הבאות, כל אחת בקובץ נפרד, כמובן, וכולן באותן חבילה:**

public class One

{

protected int \_a;

public One() {

\_a=1;

}

public One(int a) {

\_a=a;

}

public int getA(){

return \_a;

}

public int f(){

return \_a;

}

}

//------------------------------------------------------------

public class Two extends One

{

public Two(){

super();

}

public Two(int a){

super(a);

}

public int f(){

return \_a+1;

}

}

//------------------------------------------------------------

public class Three extends Two

{

public Three(){

super();

}

public Three(int a){

super(a);

}

public int f(){

return \_a+2;

}

public int g(){

return \_a;

}

}

public class Four extends Three

{

public Four(){

super();

\_a++;

}

public Four(int a){

super(a);

}

public Four(int a,int b){

super();

\_a=\_a+a+b;

}

public int f(){

return \_a-1;

}

}

//------------------------------------------------------------

public class Five extends Three

{

public Five(){

super();

}

public int g()

{

return \_a+1;

}

}

**כתבנו שיטה המקבלת אובייקט של אחת המחלקות One, Two, Three, Four, Five לעיל, ומדפיסה מהו סוג האובייקט. כלומר, מאיזו מחלקה הוא נוצר. עליכם להשלים את החסר בשיטה.**

**לדוגמא,**

אם היינו כותבים ב- main את הפקודה:

Three t = new Three(5);

ואחר כך היינו קוראים לשיטה whichClass(t)

אז היה מודפס על הפלט:

Three

**חתימת השיטה היא:**

**public static void whichClass(Object obj)**

**שימו לב, עליכם להשלים את השיטה שאנחנו כתבנו. אם לא תעשו כך, ותכתבו שיטה משלכם, עליכם לעמוד בהגבלות הכתובות להלן, ובכל מקרה, התשובה שלכם תקבל לכל היותר 10 נקודות, גם אם היא תהיה נכונה.**

כדי לבדוק מהו סוג האובייקט, יש להיעזר בשיטות getA, f, g שהוגדרו במחלקות לעיל.

**אסור להשתמש בפעולה instanceOf או בפעולות אחרות של המחלקה Object.**

**אסור לשנות את המחלקות One, Two, Three, Four, Five.**

אפשר להניח שהאובייקט obj שייך לאחת המחלקות הנ"ל, והוא אינו null.

**התשובה היא:**

public static void whichClass(Object obj)

{

int a =((One) obj).getA();

int f = ((One) obj).f();

int g;

if(a==f)

System.out.println("One");

If(a==f-1)

System.out.println("Two");

if(a==f+1)

System.out.println("Four");

if(a==f-2)

{

g= ((Three) obj).g();

if(a==g)

System.out.println("Three");

else

System.out.println("Five");

}

}

שאלה 5 (18 נקודות)

נגדיר: "תור ישראלי" (Israeli Queue) הוא מעין תור רגיל (Queue) שהכניסה אליו נעשית מסוף התור (זנבו), והיציאה מהתור היא מראשו, אלא שבתור ישראלי ניתן גם להידחף ישירות לראש התור. היציאה מהתור הישראלי היא תמיד מראשו.

המחלקה IsraeliQueue מממשת תור ישראלי שהאיברים שבו הם מספרים שלמים.

להלן נתונות השיטות במחלקה IsraeliQueue הפועלות על תור ישראלי:

|  |  |
| --- | --- |
| בנאי הבונה תור ישראלי ריק | IsraeliQueue( ) |
| השיטה מכניסה את t לסוף התור הישראלי | **void** insertAtEnd ( int t) |
| השיטה מכניסה את t לראש התור הישראלי | **void** insertAtHead (int t) |
| השיטה מוציאה את האיבר מראש התור הישראלי ומחזירה אותו | int removeFromHead( ) |
| השיטה מחזירה "אמת" אם התור הישראלי ריק ו-"שקר" אחרת | **boolean** isEmpty( ) |

לפניכם שתי שיטות המוגדרות בתוך המחלקה IsraeliQueue :

public static int what(int [] a, int x, int y)

{

int t = a[x];

IsraeliQueue q = new IsraeliQueue();

q.insertAtEnd(t);

int p=x;

for (int i = x+1; i <= y; i++) {

if (a[i]<t) {

q.insertAtHead(t);

p++;

}

else

q.insertAtEnd(t);

}

for (int i = x; i <= y; i++)

a[i] = q.removeFromHead ();

return p;

}

public static void something(int [] a)

{

int p = a.length;

while (p!=0)

p = what(a, 0, p-1);

}

נתון המערך **a** הבא:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | ד0 |
| 3 | 9 | 6 | 2 | 1 | 12 | 2 | 8 | 5 | 7 |

**סעיף א (2 נקודות)**

איזה ערך תחזיר השיטה what לאחר הקריאה what(a, 0, 9) ? כיצד יראה המערך לאחר הקריאה הזו?

**התשובה היא:**

השיטה WHAT תחזיר את הערך 6.

המערך יראה לאחר הקריאה לפונקציה זו כך:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 9 | 12 | 8 | 7 | 5 | 2 | 1 | 2 | 6 | 3 |

**סעיף ב (5 נקודות)**

מה מבצעת השיטה **what** כאשר היא מקבלת **מערך חד-ממדי** כלשהו מלא במספרים שלמים, ושני פרמטרים שלמים x ו- y שמייצגים את האינדקס של התא הראשון והאינדקס של התא האחרון במערך (בהתאמה)? הסבירו בקצרה **מה** מבצעת השיטה ולא **כיצד** היא מבצעת זאת. מה משמעותו של הערך שהוחזר? **התייחסו למקרי קצה.**

**התשובה היא:**

השיטה עוברת על המערך וסופרת כמה ערכים במערך קטנים מהערך במקום a[x] כלומר בערך במקום a[0] ומחזירה אותו. בנוסף היא שמה את הערכים שקטנים מa[x] כלומר מa[0] בתחילת התור הישראלי ומי שגדול מהערך הנ"ל ידחף לסוף התור ואז המערך ישתנה להיות כל הערכים שקטנים מהערך שהיה בa[x] ואחריהם כל הערכים שגודלים מהערך שהיה בa[x]

**סעיף ג (3 נקודות)**

כיצד יראה המערך a לאחר הקריאה something(a)?

**התשובה היא:**

המערך a יראה לאחר קריאה לsomething(a) כך:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 9 | 12 | 8 | 7 | 5 | 6 | 3 | 2 | 2 | 1 |

**סעיף ד (8 נקודות)**

מה מבצעת השיטה **something** כאשר היא מקבלת **מערך חד-ממדי** כלשהו מלא במספרים שלמים? הסבירו בקצרה **מה** מבצעת השיטה ולא **כיצד** היא מבצעת זאת. **התייחסו למקרי קצה.**

**התשובה היא:**

הפונקציה something מבצעת כל עוד המערך לא ריק, סידור מחדש של המערך עד כאשר האיבר קטן ביותר ברשימה נמצא באיבר a[0].

**ב ה צ ל ח ה!**