בס"ד

מטלה מספר 2

קורס: מבני נתונים

שם: כפיר גולדפרב

ת.ז: 208980359

אימייל: kfir.goldfarb@msmail.ariel.ac.il

פתרון שאלה 1:

$countIslands$ זו הפונקציה העיקרית שמטרת הוא לספור את מספר האיים במטריצה שהפונקציה מקבלת נקרא לה $matrix$, נגדיר מטריצה נוספת באותו הגודל של המטריצה $matrix$ שמכילה ערכים מסוג בוליאני, כל הערכים מאופסים ל-$false$ בתור התחלה, הלולאה בתוך לולאה תעבר על כל איבר במטריצה וכאשר תמצא תא שערכו הוא 1 ושערכו באותו המקום ב-$visited$ הוא $false$ כלומר עוד לא התבצעה בדיקה על התא הנ"ל, היא תשלח לפונקצית העזר $depthSearch$ את המטריצה, את המטריצה $visited$ ואת מקום ה-i, j במקום שנמצא, הפונקציה $depthSearch$ תשתמש בשתי המערכי עזר $helpRow$ ו-$helpCol$ שמטרתם היא לבדוק עבור כל נקודה במטריצה המקורית את הנקודות השכנות אליה האם הנקודות הללו מחוברות לעוד נקודות עם ערך 1 שיוצר המשך לאי או עם ערך 0 שיוצר "ים", נגדיר באותו מקום במערך $visited$ ערך אמת כדי לדעת שהפונקציה בדקה את המיקום הזה, נעבור בלולאה מ-0 עד 8 (כדי לבדוק את כל המיקומים בעזרת המערכי עזר) במעבר נשלח לעוד פונקצית עזר בשם $isSafe$ אשר תקבל את המערך ,את הערך במקום ה-k של המערך עזר $i + helpRow$, את הערך במקום ה-k של המערך עזר $j + helpCol$, וגם את מערך ה-$visited$, מטרת הפונקציה הזאת היא לבצע את הבדיקות הבאות כאשר $i = i + helpRow, j = j + helpCol$:

```
(i >= 0) and (i < matrix.length) and (j >= 0) and (j < matrix[0].length) and (matrix[i][j] =
= 1 && !visited[i][j])
```

מטרת הפונקציה היא לבדוק אם הערך שבתא הוא תקין ואם כן היא תחזיר $true$ והלולאה בפונקצית $depthSearch$ שעוברת על איברי המערכי עזר עם האיבר שאותו היא בודקת, מקיימת בצורה רקורסיבית וקוראת לעצמה עם אותם נתונים – שולחת לעצמה את המערך ,את הערך במקום ה-k של המערך עזר $i + helpRow$, את הערך במקום ה-k של המערך עזר $j + helpCol$, וגם את מערך ה-$visited$, ותתקבצע שוב באופן רקורסיבי וככה עד שהפונקציה $countIslands$ עוברת על כל איברי המערך בפעם אחת, **<u>קוד הפונקציות בעמוד הבא:</u>**

```
public static int countIslands(int matrix[ ][ ]) {
  boolean visited[ ][ ] = new boolean[matrix.length][matrix[0].length];
  int count = 0;
  for(int i = 0; i < matrix.length; i++) {
    for(int j = 0; j < matrix[0].length; j++) {
      if(matrix[i][j] == 1 && !visited[i][j]) {
        depthSearch(matrix, visited, i, j);
        count++;
      }
    }
  }
  return count;
}
public static void depthSearch(int matrix[][], boolean visited[][], int i, int j) {
  int helpRow[] = new int[] { -1, -1, -1, 0, 0, 1, 1, 1 };
  int helpCol[] = new int[] { -1, 0, 1, -1, 1, -1, 0, 1 };
  visited[i][j] = true;
  for (int k = 0; k < 8; k++) {
    if(isSafe(matrix, helpRow[k] + i, helpCol[k] + j, visited)) {
      depthSearch(matrix, visited, helpRow[k] + i, helpCol[k] + j);
    }
  }
}

public static boolean isSafe(int matrix[ ][ ], int i, int j, boolean visited[ ][ ]) {
  return (i >= 0) && (i < matrix.length) && (j >
          = 0) && (j < matrix[0].length) && (matrix[i][j] == 1 && !visited[i][j]);
}
```

פתרון שאלה 2:

הפונקציה $isBST$ מטרת היא לבדוק אם עץ בינארי הוא עץ חיפוש בינארי כלומר האם איבריו מסודרים בו בצורה ממויינת, הרעיון הכללי של הדרך שבה עשיתי אותה הוא ליצור מחסנית שאליה אני יכניס את כל איברי העץ בעזרת הפונקצית עזר $inOrderForIsBST$ שזו פונקציה שמקבלת את העץ בינארי ומחזירה מחסנית שבה נמצאים כל איברים של העץ כאשר הם נכנסו בדרך $inOrder$, כמובן שידוע לנו שכאשר מבצעים הדפסת או סדר מסוג $inOrder$ איברי העץ בעץ חיפוש בינארי יוצאים בצורה ממויינת, ולכן הדבר היחידי שנשאר לעשות הוא לבדוק אם כל האיברים במחסנית הם בצורה ממויינת כלומר כל איבר קטן מקודמו (pop מה-$peek$) ואת הבדיקה הזאת ביצעתי בעזרת לולאת $while$ אם התנאי לא מתקיים הפונקציה תחזיר $false$ ואם כן היא תעבור על כל איברי התור ותראה שכולו ממוין ומכאן תחזיר $true$ כיוון שאיברי העץ ממוינים ומכאן נובע שהעץ הוא עץ חיפוש בינארי, להלן הקוד:

```
public static boolean isBST(BinaryTree bt) {
    // if tree is empty
    if(bt.root == null) throw new NoSuchElementException("tree is empty");
    // if tree is not empty
    else {
      // getting a stack with tree values by inOrder order
      Stack < Integer > insertTreeValues = new Stack <> ();
      Stack < Integer > treeValues = inOrderForIsBST(bt.root, insertTreeValues);
      // checking if the pop value is grater than the peek value
      // checking if the tree is not a binary search tree - O(n)
      while(treeValues.size() > 1) {
        Integer temp = treeValues.pop();
        if(temp <= treeValues.peek()) return false;
      }
    }
    return true; // return true if the tree is a binary search tree
  }
  // auxiliary function that helping me to to insert tree values to stack by inOrder order recurly
  public static Stack < Integer > inOrderForIsBST(Node node, Stack < Integer > insertTreeValues) {
    if(node ! = null) {
      inOrderForIsBST(node.left, insertTreeValues);
      insertTreeValues.push(node.data);
      inOrderForIsBST(node.right, insertTreeValues);
    }
    return insertTreeValues; // return the stack with tree value
  }
```

בס"ד

פתרון שאלה 3:

כדי להוכיח שהעץ הוא עץ מאוזן נרצה להראות כי הגבוה הוא לא יותר מ-$O(\log n)$ (הגדרת עץ מאוזן),

נגדיר את h להיות גובה העץ (המסלול הכי גדול מהשורש לעלים), נגדיר את m_h להיות הקודקוד המינימאלי של העץ עם הגובה h,

על מנת לקבל את m_h הקודקוד המינימאלי של העץ ידוע כי $m_h = m_{h-1} + m_{h-3}$ כאשר מתקיים גם $m_h \geq 2m_{h-1}$,

מכאן נובע כי $m_{h-3} \geq 2 \cdot m_{h-6}$ (לפי האינדקסים), נציב שוב באי-שיוויון שוב ושוב (כמו בחישוב רקורסיבי):

$$m_h \geq 2m_{h-3} \geq 2 \cdot (2m_{h-6}) \geq 2 \cdot 2 \cdot (2m_{h-9}) \geq \cdots \geq 2 \cdot 2 \cdot \ldots \cdot 2 \cdot c$$

נסמן ב-k את מספר הפעמים שנדרש לבצע את הרקורסיה עד לתנאי עצירה כלומר:

$$m_h \geq 2^k \cdot c$$

כאשר c הוא הכמות המינימלית של קודקודים במקרה הבסיס ו-$c > 1$, והתנאי עצירה יתקיים כאשר $2^{\mathrm{k}} = 1$ ולכן נחשב את כמות הפעמים בסיבוכיות של m כלומר:

$$m = 2^k \rightarrow \log_2 m = \log_2 2^k \ \rightarrow k = \log_2 m \ \rightarrow h = O(\log(n))$$

■

פתרון שאלה 4 – סעיף א':

מכיוון שקיימות ארבעה התכונות של עץ אדום-שחור:

1. צומת הוא שחור או אדום (אחד מן השניים).
2. השורש הוא שחור.
3. כל העלים שחורים.
4. שני ילדיו של צומת אדום הם שחורים

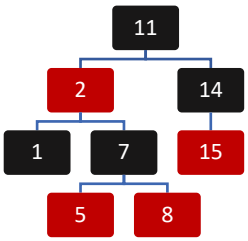
בשילוב של התכונה הראשונה עם כל השאר אפשר להגיד שכל צומת שאינו שחור הוא אדום,

ולכן אם הצומת הוא השורש והוא שחור או שהוא צומת שחור בלי קשר לשורש, אז הבנים שלו חייבים להיות אדומים ולא שחורים מהסיבה שאם יש להם ילדים אז לפי תכונה 4 הוא חייב להיות אדום,

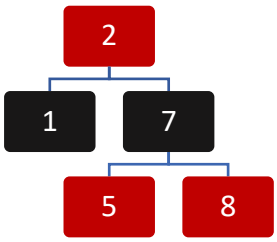
וגם אם אין להם ילדים אז בנוסף לכך ניתן לומר שאין סתירה לכך שהוא יהיה אדום ולכן לפי העקרון אם צומת לא בהכרח שחור אז הוא אדום, ומכיוון שאין מקרה לפי התכונות שבו יש סתירה למצב שבן של צומת שחור חייב להיות שחור אז הוא בהכרח אדום ולא שחור.

פתרון שאלה 4 – סעיף ב':

הטענה אינה נכונה, נראה דוגמה נגדית:



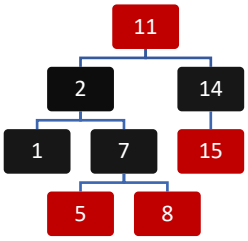
נבחר את 2 להיות השורש של התת עץ אדום שחור כלומר יש לנו את התת עץ הבא:



קיבלנו תת-עץ אדום-שחור של עץ אדום-שחור, אך השורש (2) בצבע אדום ולא שחור – סתירה לתכונה שהשורש חייב להיות שחור ולכן אינו עץ אדום-שחור.

פתרון שאלה 4 – סעיף ג':

כן, ניקח את העץ לדוגמה מסעיף ב', נעבור את השורש לאדום (הנחה) ונשנה את כל הצבעים של הצמתים האחרים לפי שאר התכונות:



קיימת התכונה - שני ילדיו של צומת אדום הם שחורים ולכן את 2 הפכתי לשחור, מעבר להנחה שהנחנו על זה שהשורש האדום, וזה ששינו את הבנים של השורש לשחורים, קיימות עוד שני התכונות - שכל העלים שחורים ושצומת הוא שחור או אדום (אחד מן השניים), וניתן לשים לב בקלות שלא קיבלנו סתירה לתנאים אלה, חוץ מזה שהשורש הוא אדום, עדיין כל העלים שחורים, ולאחר השינוי עדיין שני ילדיו של צומת אדום הם שחורים, ושצומת הוא אדום או שחור.

ובהוכחה על מקרה כללי ניתן לומר אותו דבר, יהי T עץ אדום-שחור, נניח ששורש העץ T הוא אדום, מכאן לפי התכונה שני ילדיו של צומת אדום הם שחורים – אם הבנים של השורש אדומים נשנה אותם לשחורים, הבנים של הבנים של השורש שחורים כיוון שלפני השינוי הבנים של השורש היו אדומים, לא קיבלנו שום סתירה לתכונות עץ אדום-שחור חוץ מההנחה שהשורש אדום - העלים עדיין שחורים, שני ילדיו של צומת אדום הם שחורים ושצומת הוא שחור או אדום (אחד מן השניים), ולכן הטענה מתקיימת.

■

פתרון שאלה 5:

מעל כל פקודה, פונקציה, מחלקה או שורת קוד כתבתי הסבר קצר על מטרת הקוד (כל הקוד העיקרי עם החישובים קורה במעבר אחד על מערך המילים, ההדפסה היא פעולות בסיבוכיות $O(1)$):

```
public static void report(String[ ] sArr) {

    // contains the different words for sum them
    LinkedList < String > differentWords = new LinkedList <> ();

    // contains the words that appear more than once
    LinkedList < String > containsAtLeastWords = new LinkedList <> ();

    // Appears is a class i build down below,
    // that can storage Word objects in an arrays list
    // Word is also a class i build down below,
    // word is an object with two parameters:
    // value = string of the word,
    // sum = the time that the word appears
    Appears appears = new Appears();

    // maxAppearsWord is the word that appears the most
    Word maxAppearsWord = new Word(sArr[0]);

    // max length word
    String max = sArr[0];
    boolean in = false;

    // going throw the array once
    // let n = sArr.length, loop complexity: O(n)
    for(int i = 0; i < sArr.length; i++) {

      // inserting all the different words to differentWords list
      if(!differentWords.contains(sArr[i])) differentWords.add(sArr[i]);

      // inserting all the words that appear more than once to containsAtLeastWords list
      else {
```

```
    // if containsAtLeastWords is already contains the word:
    // make in to true for delete the word for avoid multype words
    // don't delete here for not make truble by inserting the words
    if(containsAtLeastWords.contains(sArr[i])) in = true;
    containsAtLeastWords.add(sArr[i]);

  }

  //deleting word from containsAtLeastWords by in (if already contain word)
  if(in) containsAtLeastWords.removeLast();
  in = false;

  // checking the String sArr[i] is in appears arrays list by string value
  // if it contains the word, using wordSum metod to add 1 to the sum:
  // counting how much tims the word is appears
  // if it not conatins the word, create new word and add it to the appears array list
  if(appears.wordContains(sArr[i])) {
    appears.wordSum(sArr[i]);
  } else {
    Word word = new Word(sArr[i]);
    appears.add(word);
  }

  // by going throw all appears word,
  // cheking who is the word with max of sum = max apearing
  for(int k = 0; k < appears.mostAppears.size(); k++) {
    if(maxappearsWord.sum < appears.mostAppears.get(k).sum) {
        maxappearsWord = appears.mostAppears.get(k);
     }
  }

  // getting the max length word
  if(max.length() < sArr[i].length()) max = sArr[i];
}
```

```
        // output title
        System. out. println("Report:\n");

        // s1
        System. out. println("Number of words: " + sArr. length); // O(1)

        // s2
        System. out. println("Number of different words: " + differentWords. size()); // O(1)

        // s3
        System. out. println(Number of words that appear more than once: + containsAtLeastWords. size());
       // O(1)

        // s4
        System. out. println("Most appears word is: \"" + maxappearsWord. value + "
                        \", time apear: " + maxappearsWord. sum); // O(1)

        // s5
        System. out. println("The longest word is: \"" + max + "\""); // O(1)

    }

    // testing
    public static void main(String[ ] args) {
        String[] sArr = {"to", "and", "to", "to", "good", "data", "data", "or"};
        report(sArr);
    }

// Word is also a class i build, word is an object with two parameters:
// value = string of the word, sum = the time that the word appears
class Word {
    public int sum;
    public String value;
```

```
    public Word(String value) {
        this.value = value;
        this.sum = 1;
    }

    public boolean same(String word) {
        if(this.value == word) return true;
        return false;
    }

    @Override
    public String toString() {
        return "" + "value: \"" + value + "\", sum: " + this.sum;
    }
}

// appears is a class i build, that can storage Word objects in an arrays list
class Appears {
    public ArrayList < Word > mostAppears;

    public Appears() {
        this.mostAppears = new ArrayList <> ();
    }

    public void add(Word word) {
        this.mostAppears.add(word);
    }

    public void wordSum(String word) {
        for(int i = 0; i < this.mostAppears.size(); i++) {
            if(this.mostAppears.get(i).same(word)) mostAppears.get(i).sum++;
        }
    }

    public boolean wordContains(String word) {
```

```
        for(int i = 0; i < this.mostAppears.size(); i++) {
            if(this.mostAppears.get(i).same(word)) return true;
        }
        return false;
    }

    @Override
    public String toString() {
        String ans = "";
        ans += "[";
        for(int i = 0; i < this.mostAppears.size(); i++) {
            ans += this.mostAppears.get(i) + ", ";
        }
        ans += "]";
        return ans;
    }
}
```