מבוא לחישוב 2-7015710 סמסטר ב' – מועד א' 16/7/2023 + הצעת פתרון

- מרצה: בעז בן משה
- משך המבחן: שעתיים וחצי (2.5 שעות).
- מחברת שורות. אין שימוש בחומר עזר.
- יש להחזיר את דף המבחן בסוף המבחן.
- במבחן ארבע שאלות, כולן חובה. לבחינה זו מצורפים 5 דפי דוגמאות קוד.
- בכל שאלה ניתן לכתוב פונקציות ואו מחלקות עזר כרצונכם.
- כל עוד לא נאמר אחרת ניתן בהחלט לפתור סעיף אחד בעזרת סעיף אח.

שאלה 1:
מספר ראשוני הוא מספר טבעי (2 ומעלה) שמתחלק בעצמו וב 1 בלבד. נגדיר שני מספרים ראשוניים (p1,p2) להיות "סמוכים" אם ורק אם כל המספרים בין p1 ל p2 הם פריקים (לא ראשוניים).

1.1 (18 נקודות) כתבו פונקציה שמקבלת שני מספרים טבעיים n1<n2 ומחזירה את **המרחק המקסימאלי** מעל כלל זוגות הראשוניים **הסמוכים** בתחום [n1,n2], ניתן להניח שקיים לפחות זוג אחד של ראשוניים סמוכים בתחום, **וש 2<=n1<n2**
לדוגמא עבור [10,19] המרחק **המקסימאלי** בין הראשוניים הסמוכים הוא 4 (המרחק בין 13 ל 17)
{10,**11**,12,**13**,14,15,16,**17**,18,**19**}

```
public static int maxDistPrimes(int n1, int n2) {
```

1.2 (7 נקודות) כתבו פונקציית בדיקה (Junit) לפונקציה שמימשתם ב 1.1.

```
@Test
public void maxDistPrimesTest() {…}
```

תשובה:
1.1

```
public static int maxDistPrimes(int n1, int n2) {
	int max = -1;
	int p1 = n1;
	while(p1<=n2 && p1!=-1) {
		if(isPrime(p1)) {
			int p2 = nextPrime(p1+1,n2);
			if(p2!=-1 && p2-p1>max) {
				max = p2-p1;
			}
			p1 = p2;
		}
	return max;
}
public static int nextPrime(int start, int end) {
	int ans = -1;
	int i=start;
	while(ans==-1 && i<=end) {
		if(isPrime(i)) {ans = i;}
```

```
                i++;
        }
}

public static boolean isPrime(int n) {
        boolean ans = true;
        int i=2;
        while(ans && i*i<=n) {
                if(n%i==0) {ans = false;}
                i++;
        }
}
```

1.2

```
@Test
public void maxDistPrimesTest() {
        int a0 = maxDistPrimes(10,10);
        assertEquals(a0,-1);
        a0 = maxDistPrimes(10,12);
        assertEquals(a0,-1);
        int a1 = maxDistPrimes(10,20);
        assertEquals(a1,4);
}
```

שאלה 2:

בשאלה זו נתייחס לממשק של צורות (GeoShape),
כזכור, הפעלת השיטה getClass().getSimpleName() מחזירה מחרוזת עם שם המחלקה ממנה נוצר האובייקט.
בשאלה זו נרצה לתכנן פונקציה שמקבלת מערך של צורות ומפצלת את כלל הצורות במערך לתוך אוסף של מערכים של צורות כאשר כל מערך באוסף מכיל צורות מאותה מחלקה.
לדוגמא נניח שבמערך s יש שני מלבנים, שלושה משולשים ועיגול אחד, הפונקציה תחזיר ArrayList שמכיל 3 מערכים (של צורות) אחד שמכיל את שני המלבנים, שני שמכיל את שלושת המשולשים, ושלישי שמכיל את העיגול. ניתן להניח שמערך הקלט אינו ריק וכלל הצורות בו שונות מ null.
2.1 (8 נקודות) הסבירו (ללא קוד, אפשר בעברית או בפסאדו-קוד) כיצד תממשו את הפונקציה.

תשובה:

1. נגדיר משתנה עזר ריק (נניח בשם shapes) מסוג ArrayList<ArrayList<GeoShape>>
2. נעבור על כל מערך הצורות (s)
    a. עבור כל צורה sh (ב s) נקבל את המחרוזת של שם המחלקה (name).
    b. נעבור כל אחד מהאוספים (tmp) ב shapes:
        i. ונבדוק האם הצורה הראשונה ב tmp היא מאותה מחלקה כמו name:
            1. אם כן נוסיף את sh לtmp
            2. אם לא קיים נייצר אוסף חדש (col) מסוג ArrayList<GeoShape>, נכניס את sh לתוכו, ואת col נכניס ל shapes
3. נגדיר משתנה תשובה (ריק) מסוג **ArrayList<GeoShape[]>**
4. ונמלא אותו ע"י מעבר על כל אוסף ב shapes (אפשר לעשות זאת בשורה אחת ע"י קריאה לשיטה toArray()) או פשוט לממש זאת בעצמינו – כפי שמוצג בתשובה מטה.

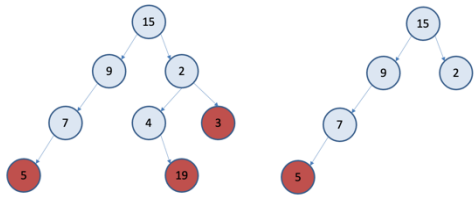
2.2 (17 נקודות) כתבו פונקציה שמקבלת מערך של צורות, ומפצלת אותו לאוסף של מערכים לפי טיפוסי הצורות.
תשובה:

```
public static ArrayList<GeoShape[]> splitToClasses(GeoShape[] s) {
	ArrayList<ArrayList<GeoShape> shapes = new ArrayList<ArrayList<GeoShape>();
	for(int i=0;i<s.length;i++) {
		GeoShape sh = s[i];
		ArrayList<GeoShape> f = find(sh.getClass().getSimpleName(), shapes);
		if(f==null) {f = new ArrayList<GeoShape>(); shapes.add(f);}
		f.add(sh);
	}
	ArrayList<GeoShape[]> ans = new ArrayList<GeoShape[]>();
	for(int i=0;i<shapes.size();i++) {  // ans.add(shapes.get(i).toArray();}
		ArrayList<GeoShape> sh = shapes.get(i);
		GeoShape[] arr = new GeoShape[sh.size()];
		for(int j=0;j<sh.size();j++) {arr[j]=sh.get[j];}
		ans.add(arr);
	}
	return ans;
}

// assumes each ArrayList<GeoShape> in shapes is none empty!
private static ArrayList<GeoShape> find(String cls, ArrayList<ArrayList<GeoShape> shapes) {
	ArrayList<GeoShape> ans = null;
	for(int i=0;i<shapes.size() && ans==null; i++) {
		GeoShape sh = shapes.get(i).get(0);
		If(cls.equals(sh. getClass().getSimpleName()) {ans = shapes.get(i);}
	}
	return ans;
}
```

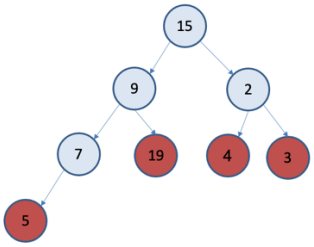
שאלה 3:
בשאלה זו נתייחס לעצים בינאריים כפי שמופיעים בחומר העזר המצורף.

3.1 (5 נקודות) שרטטו דוגמא לעץ בינארי **בעל 8 קודקודים**, **מתוכם ארבעה בדיוק הם עלים**, ורק אחד מהעלים הוא **ברמה הכי רחוקה מהשורש**.



3.2 (20 נקודות) כתבו פונקציה סטטית שמקבלת עץ בינארי ומחזירה **את הרמה הכי רחוקה ביותר מהשורש שאין בה אף עלה**. משמע, בעץ ריק או בעל שורש יחיד היא תחזיר 1-, בעץ בעל שני קודקודים הפונקציה תחזיר 0, בעץ בתמונה משמאל הפונקציה תחזיר 1, ובימני תחזיר 2.

public static int maxLevelWithNoLeafs(BinaryTree<T> bt) {…}



3.1 עץ בעל 8 קודקודים, 4 עלים שרק אחד מהעלים (מספר 5) הוא רחוק ביותר מהשורש.

3.2

```
public static int maxLevelWithNoLeafs(BinaryTree<T> bt) {
	int ans = -1;
	if(bt!=null && bt.size()>1) {
		int[] leafs = new int[bt.height()];
		countLeafs(bt, leafs, 0);
		for(int i=0;i<leafs.length;i++) {
			if(leafs[i]==0) {ans=I;}
		}
	}
	return ans;
}
public static void countLeafs(BinaryTree<T> bt, leafs, int level) {
	if(bt!=null) {
		countLeafs(bt.getLeft, leafs, level+1);
		if(isLeaf(bt)) {leafs[level]+=1;}
		countLeafs(bt.getRight, leafs, level+1);
	}
}
public static boolean isLeaf(BinaryTree<T> bt) {
	return bt!=null && bt.size()==1;
}
```

שאלה 4:

בשאלה זו נניח שקיימת לכם המחלקה Q שמייצגת קוביית משחק **הוגנת** (בכל "זריקה" הקובייה יכולה לקבל ערך 1,2,3,4,5,6 בלבד, בהסתברות **אחידה**). למחלקה יש בנאי ריק והיא מממשת את הממשק Game בעל השיטות הבאות:

```
public interface Game() {
	void roll();   // roll the dice
	int getVal(); // return the value of the last roll (the sum of all the dices – in the last roll).
}
```

1. השיטה roll() לא מקבלת פרמטרים, ולא מחזירה ערכים, למעשה "זורקת" את הקובייה (או הקוביות).
2. השיטה getVal() מחזירה את ערך של הקובייה (או הקוביות) מהזריקה האחרונה (אם היא לא נזרקה עדיין מחזירה 0). שימו לב שכל עוד לא בוצע זריקה נוספת, הערך המוחזר לא ישתנה.

4.1 (8 נקודות) השלימו את המחלקה Q2 שמייצגת זוג קוביות משחק (הוגנות) בעלת בנאי ריק ושמממשת את הממשק Game, וכן כוללת שיטה שמחזירה אמת אם ורק אם שתי הקוביות היו בעלות אותו ערך ב"זריקה" האחרונה ("דאבל – בלשון שש-בש").

4.2 (10 נקודות) השלימו את המחלקה Q2_Err **שיורשת** מהמחלקה Q2, ומייצגת זוג קוביות "מזויפות", זוג הקוביות יכול להגיע לכל זוג ערכים (מעל [1,6]) אך לזוג הקוביות יש לו "עדיפות" סטטיסטית עבור מספרים זהים (עבור "דאבלים").

```
public class Q2_Err extends Q2 {//...

}
```

4.3 (7 נקודות) הסבירו (במילים, אין צורך לכתוב קוד) כיצד ניתן לבדוק שהמחלקה Q2 מייצגת זוג קוביות **הוגנות**, והמחלקה Q2_Err אינה מייצגת זוג קוביות **הוגנות**.

תשובה:

4.1

```
public class Q2 implements Game {  //...
        private Q _q1, _q2;
        public Q2() {_q1=new Q(); _q2=new Q();}
        @Override
        public void roll() {_q1.roll; _q2.roll;}
        @Override
        public int getVal(){return _q1.getVal() + _q2.getVal();}
        public boolean isDouble() { return _q1.getVal() == _q2.getVal();}
}
```

4.2

```
public class Q2_Err extends Q2 {//...
        public Q2() {super();}
        @Override
        public void roll() {
                super.roll();
                if(!isDouble()) { super.roll();}
        }
}
```

4.3

נכתוב פונקציית מספר פונקציות בדיקה

1. פונקציה שזורקת זוג קוביות בסוג Q2 נניח 600 פעמים, אנו מצפים שבכשישית מהמקרים (כ 100 פעמים) השיטה isDouble תחזיר אמת. נבדוק שאכן כמות הפעמים שיצא double היא נניח בין 80-120 פעמים.
2. את אותה בדיקה נעשה על זוג קוביות מסוג Q2_Err ואז אנו מצפים לקבל שבשליש מהמקרים (ראו מימוש מעלה) יצא double.
3. מבחינת תקינות כללית נוכל לבדוק שממוצע סכום הקוביות הוא סביב 7 (כך יהיה בשני המקרים של Q2, Q2_Err).

הערה: באופן דומה ניתן לבדוק זאת גם ע"י כך שרק ב1/36 מהמקרים בערך יצא לנו סכום 2, (או סכום 4,6,8,10,12) וכן נוכל לבדוק את "כל הסכומים" משמע לבדוק שסכום 3 (ו 11) יוצא 2/36 מהמקרים, וסכום 4 (ו 10) יוצא 3/36 מהמקרים ,סכום 5 (ו 9) יוצא 4/36 מהמקרים, סכום 6 (ו 8) יוצא 5/36 מהמקרים, וסכום 7 יוצא כ 6/36 מהמקרים – **יודגש**, בדיקה (טרחנית) שכזו אינה נדרשת לצורך קבלה של ניקוד מלא בשאלה זו.

בהצלחה!!!

חומר עזר שמצורף לבחינה

- אם לא נאמר אחרת, ניתן לעשות שימוש בחומר העזר בפתרון השאלות.
- דוגמאות הקוד: String, Sort, Junit, Point2D, GeoShape, MyListInterface, BinaryTree

```
// Math.random(); // returns a random double in [0,1)

public class StringFunctions {  // This is a very simple “main” that uses String and ArrayList
	public static void main(String[] a) {
		ArrayList<String> arr = new ArrayList<String>();
		String s = "12345", s2 = “12321”;
		arr.add(s);
		arr.add(s2);
		arr.add(s2.subString(1,4)); // “232”
		if(!arr.contains("232")) {arr.add(s);}
		for(int i=0;i<arr.size();i++) {
			boolean isSim = isSimetric(arr.get(i));
			System.out.println("arr[" +i+ "] "+arr.get(i)+" isSimetric: " +isSim);
		}
		while(!arr.isEmpty()) {
			s = arr.remove(0);
			System.out.println("rev("+s+")=" +reverse(s));
		}
		String words = "these are few words ...";
		String[] ww = words.split(" ");
		for(String w:ww) {System.out.println(w);}
	}

	public static boolean isSimetric(String s) {
		boolean ans = false;
		String t = reverse(s);
		ans = t.equals(s);
		return ans;
	}

	public static String reverse(String s) {
		String ans = "";
		for(int i=s.length()-1; i>=0;i=i-1) {
			ans=ans+s.charAt(i);
		}
		return ans;
	}
}

/** This is a simple interface for Binary Trees as published in: */
public interface BinaryTree<T> {
	/** @return the left (sub) tree - might be null. */
	public BinaryTree<T> getLeft();
	/** @return the right (sub) tree - might be null. */
	public BinaryTree<T> getRight();
```

```
        public T getRoot(); // The root data (type T).
        public boolean isEmpty();
        /** @return the number of nodes in this tree. */
        public int size();
        /** @return the maximum distance between the root and the farthest leaf from it*/
        public int height();
        /** Adds the data "a" to this tree, in a regular BT can be implemented using a random */
        public void add(T a);
        /** @return the i'th node using inorder “indexind”*/
        public T get(int i);
        /** search the binary tree for the first node that equals to t. If none returns null */
        public T find(T t);
        /** returns an in_order iterator */
        public Iterator<T> iterator();
        /** removes the first node that equals to t. If exists - returns it, else returns null */
        public T remove(T element);
}

/** Basic String Comparator – as defined in java.util*/
class StringComparator implements Comparator<String> {
        public StringComparator(){;}
        public int compare(String obj1, String obj2) {
                if (obj1 == obj2) {return 0;}
                if (obj1 == null) {return -1;}
                if (obj2 == null) {return 1;}
                return obj1.compareTo(obj2);
        }
}

/** This interface represents a set of operations on list of T’s. */
public interface MyListInterface<T> {
        /** Adds a String to the i"th link of the List. */
        public void addAt(T a, int i);
        /** Removes the i"th element (link) of this List. */
        public void removeElementAt(int i);
        /**Tests if ‘data’ is a member of this List. */
        public boolean contains(T data);
        /** Returns the i"th element in this List. */
        public T get(int i);
        /** Returns the number of Links in this List. */
        public int size();
}

/** This class represents a simple 2D Point in the plane */
public class Point2D {
   public static final double EPS = 0.001;
   public static final Point2D ORIGIN = new Point2D(0,0);
   private double _x, _y;

  public Point2D(double a,double b) {_x=a; _y=b;  }  // Standard Constructor.
   public Point2D(Point2D p) { this(p.x(), p.y()); }.       // Copy Constructor
```

```
    /** String Constructor: following this String structure:  "-1.2,5.3" --> (-1.2,5.3) ; */
    public Point2D(String s) {
        String[] a = s.split(",");
        _x = Double.parseDouble(a[0]);
        _y = Double.parseDouble(a[1]);
    }
    public double x() {return _x;}
    public double y() {return _y;}
    public Point2D add(Point2D p) {
        return new Point2D(p.x()+x(),p.y()+this.y());
    }
    /** Translates this point by a vector like representation of p. */
    public void move(Point2D p) {_x += p.x(); _y += p.y();}
    public String toString() {return _x+","+_y; }
    public double distance() {return this.distance(ORIGIN);
    }
    /** distance(this,p2) = Math.sqrt(dx^2 + dy^2) */
    public double distance(Point2D p2) {
        double dx = this.x() - p2.x(), dy = this.y() - p2.y();
        return Math.sqrt(dx*dx+dy*dy);
    }
    /**return true iff: this point equals to p.  */
    public boolean equals(Object p)  {
      if(p==null || !(p instanceof Point2D)) {return false;}
      Point2D p2 = (Point2D)p;
      return (_x==p2._x) && (_y==p2.y());
    }
    public boolean equals(Point2D p) {
      if(p==null) {return false;}
      return ((_x==p._x) && (_y==p._y));
    }
    public boolean close2equals(Point2D p2, double eps) {
      return ( this.distance(p2) < eps );
    }
}

public interface GeoShape {
        /** Computes if the point (ot) falls inside this (closed) shape. */
        public boolean contains(Point2D ot);
        /** Computes the area of this shape */
        public double area();
        /** Computes the perimeter of this shape. */
        public double perimeter();
        /** Move this shape by the vector 0,0-->vec
         * Note: this method changes the inner state of the object. */
        public void move(Point2D vec);
        public GeoShape copy(); /** computes a new (deep) copy of this GeoShape. */
        public String toString();   /** This method returns an String representing this shape. */
        /** This method returns an inner point – within this GeoShape. */
        public Point2D innerPoint();
}
```

```
//////////// MERGE SORT ////////////////
public static void mergeSort(int[] a) {
	int len = a.length;
	double[] tmp = new double[len];
	for(int i=0;i<len;i=i+1) {tmp[i]=a[i];}
	mergeSort(tmp);
	for(int i=0;i<len;i=i+1) {a[i] = (int)tmp[i];}
}

public static void mergeSort(double[] a) {
	int size = a.length;
	if(size>=2) {
		int mid = size/2;
		double[] left = getSubArray(a,0,mid);
		double[] right = getSubArray(a,mid,size);
		mergeSort(left); // recursive call
		mergeSort(right); // recursive call
		double[] merge = mergeArrays(left,right);
		for(int i=0;i<merge.length;i=i+1) {a[i] = merge[i];}
	}
}

public static double[] getSubArray(double[] a, int min, int max) {
	double[] ans = new double[max-min];
	or(int i=min;i<max;i=i+1) {ans[i-min] = a[i];}
	return ans;
}

/** This function merges two sorted arrays into a single sorted array. */
public  static  double[]  mergeArrays(double arr1[],  double arr2[]) {
	double[]  res = new double[arr1.length + arr2.length];
	int i=0, j=0;
	while ( i < arr1.length && j < arr2.length )	{
		if (arr1[i] <= arr2[j]) { res[i+j] = arr1[i]; i=i+1;}
		else {res[i+j] = arr2[j]; j=j+1;}
	}
	while ( i < arr1.length) {res[i+j] = arr1[i++];}
	while ( j < arr2.length) {res[i+j] = arr2[j++];}
	return res;
}

public static int[] randomIntArray(int size, int range){
	int[]arr = new int[size];
	++range;
	for(int i=0; i<size; i=i+1) {arr[i] = (int)(Math.random()*range);}
	return arr;
}

public static boolean isSortedAscending(int[] arr){
	for (int i = 1; i < arr.length; i++) {
		if (arr[i-1] > arr[i]) {return false; }
```

```
        }
        return true;
}

class SortTest {
        public static final int K = 1000, M = K*K;
        public static int[] arrK = null, arrM = null;
        @BeforeEach
        void setUp() {
                arrK = randomIntArray(K, K);
                arrM = randomIntArray(M, M);
        }

        @Test
        void testMergeSort() {
                int[] a1 = {3,1,2,1,42};
                mergeSort(a1);
                boolean isSorted =isSortedAscending(a1);
                assertTrue(isSorted);
        }

        @Test
        void testInsertionSort() {
                int[] arr = {5,1,2,0,9};
                insertionSort(arr);
                if(MyArrayLibrary.isSortedAscending(arr)!=true) {
                        fail("arr should be sorted");
                }
        }

/////////////// Performance Testing /////////////
        @Test
        void testMergeSort1() {
                long start = System.currentTimeMillis();
                mergeSort(arrM);
                long end = System.currentTimeMillis();
                double dt_sec = (end-start)/1000.0;
                boolean isSorted = isSortedAscending(arrM);
                System.out.println("Merge sort dt: "+dt_sec+" secs,  is sorted? "+ isSorted);
                assertTrue(isSorted);
                assertTrue(dt_sec<1.0);
        }
        @Test
        @Timeout(value = 1000, unit = TimeUnit.MILLISECONDS)
        void testMergeSort2() {
                mergeSort(arrDoubleM);
                boolean isSorted = isSortedAscending(arrDoubleM);
                assertTrue(isSorted);
        }
```