

תוכנה 1

תרגיל מספר 3

הנחיות כלליות:

- קראו בעיון את קובץ נוהלי הגשת התרגילים אשר נמצא באתר הקורס.
- הגשת התרגיל תעשה במערכת ה VirtualTAU בלבד (<http://virtual2002.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש (לדוגמא, עבור המשתמש zvainer יקרא הקובץ zvainer.zip) קובץ ה zip יכיל:
 - א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז. הזהות שלכם.
 - ב. קבצי ה java של התוכניות אותם התבקשתם לממש.
 - ג. קובץ טקסט עם העתק של כל קבצי ה java
 - ד. קובץ טקסט בשם answers עם התשובות לשאלות

חשוב: בתרגיל זה יש לממש את כל המתודות הנדרשות במחלקה אחת אשר תקרא Assignment03.

חלק א':

כתבו את המתודה binaryAdder, המקבלת שתי מחרוזות (טיפוסים מסוג String) המייצגות מספרים בייצוג בינארי, ומחזירה מחרוזת של תוצאת החיבור בין מספרים אלה. ניתן להניח כי הקלט תקין (כלומר מחרוזות הקלט מכילות רק את התווים '0' ו-'1'). לא ניתן להניח חסם על אורך מחרוזות הקלט. להלן דוגמאות:

```
binaryAdder("0","0") -> "0"  
binaryAdder("0","1") -> "1"  
binaryAdder("1","0") -> "1"  
binaryAdder("1","1") -> "10"  
binaryAdder("10100101","111") -> "10101100"  
binaryAdder("1010010100101","1110111") -> "1010100011100"
```

ניתן להגדיר מבני עזר או שרותים חדשים לצורך המימוש.

```
public static String binaryAdder(String a, String b) {  
}
```

חלק ב' :

כתבו את המתודה `areAnagrams`, המקבלת שתי מחרוזות (טיפוסים מסוג `String`). ובודקת האם האחת מתקבלת מסיכול אותיות השנייה. לצורך שאלה זו, רווחים אינם נחשבים כאות. בנוסף, המתודה צריכה להיות `case insensitive`, כלומר, אין הבחנה בין אותיות גדולות (`uppercase`) כאשר משווים בין המחרוזות. ניתן להניח כי הקלט תקין, כלומר שהמחרוזות אינן `null`.

להלן מספר דוגמאות:

```
areAnagrams("Debit Card","Bad Credit") -> true
areAnagrams("The eyes","They see") -> true
areAnagrams("Conversation","Voices rant on") -> true
areAnagrams("Radar","Tartar") -> false
```

ניתן להגדיר מבני עזר או שרותים חדשים לצורך המימוש.

```
public static boolean areAnagrams(String a, String b) {
}
```

חלק ג':

כתבו את המתודה `findEquation`, המקבלת מערך של מספרים ומחזירה `true` אם ניתן לחלק את המערך לשני תתי מערכים רציפים כך שערכו של ביטוי חשבוני הנוצר מהפעולות האריתמטיות חיבור, חיסור וכפל בין אברי תת-המערך הראשון, יהיה שווה לערכו של ביטוי אריתמטי דומה הנוצר מאיברי תת-המערך השני.

לא ניתן לשנות את סדר האיברים במערך, אך ניתן להשתמש בסוגריים. כלומר, בדוגמאות למטה, אם מסירים את סימני הפעולות והסוגריים מ"הוכחה", מתקבלים המספרים לפי סדרם במערך המקורי.

דוגמאות:

```
findEquation({1,2,3,6}) -> true
```

```
// proof: 1+2+3=6
```

```
findEquation({1,2,3,9}) -> true
```

```
// proof: (1+2)*3=9
```

```
findEquation({1,2,20}) -> false
```

```
findEquation({1,2,-1,4,1}) -> true
```

```
// proof: 1 = (2*-1)+4-1
```

```
findEquation({4,6,1,2,5,2,2,6}) -> true
```

```
// proof: 4 = (6+1+2+5+2)-(2*6)
```

```
findEquation({1,5,8,3}) -> true
```

```
// proof (1*5) = (8-3)
```

```
public static boolean findEquation(int[] numbers) {  
}
```

דרך אפשרית לפתרון:

1. ראשית, יש לקבוע את מיקומו של סימן השוויון
2. לאחר מכן יש לנסות ולבנות את כל הביטויים האפשריים על כל תת מערך (בצורה רקורסיבית).
3. אם מתקבלים ביטויים שערכם שווה, ניתן לעצור.
4. אחרת יש לקבוע מיקום שונה לסימן השוויון ולחזור על סעיפים 2-3.
5. אם בכל זאת לא הצלחנו, נחזיר false.

שימו לב: אין צורך לספק את הביטויים המספקים את השוויון (כלומר את ה"הוכחה") אלא רק אם ניתן או לא ניתן להגיע לפתרון.

חלק ד':

א. ממשו את השרות `minRun` אשר בהינתן מחרוזת (`string`) מחזיר את הריצה (`run`) הקצרה ביותר במחרוזת. **ריצה** מוגדרת בתור מספר הפעמים שבו מופיע אותו התו ברצף (השרות מחזיר את אורך הרצף הקצר ביותר במחרוזת).
להלן כמה דוגמאות:

```
minRun("hoopla") -> 1
minRun("aaaabbbbCCCCC") -> 3
minRun("bbbbbbbbbaa") -> 2
minRun("") -> 0
```

ניתן להגדיר מבני עזר או שרותים חדשים לצורך המימוש.

```
public static int minRun(String str) {
}
```

ב. ממשו את השרות `maxRun` אשר בהינתן מחרוזת מחזיר את הריצה הארוכה ביותר במחרוזת.
להלן מספר דוגמאות:

```
maxRun("hoopla") -> 2
maxRun("aaaabbbbCCCCC") -> 6
maxRun("bbbbbbbbbaa") -> 8
maxRun("") -> 0
```

ניתן להגדיר מבני עזר או שרותים חדשים לצורך המימוש.

```
public static int maxRun(String str) {
}
```