תוכנה 1 – אביב תשע"ה

תרגיל מספר 8

מבני נתונים מקושרים ו-collection framework

הנחיות כלליות:
קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.

- הגשת התרגיל תיעשה במערכת ה-moodle בלבד (http://moodle.tau.ac.il/).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש aviv יקרא הקובץ aviv_hw8.zip). קובץ ה-zip יכיל:
  א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
  ב. קבצי ה- java של התוכניות אותם התבקשתם לממש.

חלק א' (42 נק')

בחלק זה עליכם לממש שלוש מתודות עבור מבנה מקושר נתון. בכל המקרים, עליכם לבצע מעבר אחד לכל היותר על כל רשימה. כמו כן, אין להעתיק את תוכן הרשימה למבנה נתונים אחר כלשהו (למשל מערך), אלא יש לעבוד עם מבנה הנתונים המקושר. ניתן להניח שהרשימה לא ריקה (מכילה חוליה אחת לפחות). ה-next של החוליה האחרונה ברשימה הוא null.

המחלקה sw1.linkedlist.LinkedListNode מגדירה חוליה בודדת ברשימה מקושרת:

```
public class LinkedListNode {
	private int value;
	private LinkedListNode next;

	public LinkedListNode(int value){
		this.value = value;
	}
	public int getValue() {
		return value;
	}
	public void setValue(int value) {
		this.value = value;
	}
	public LinkedListNode getNext() {
		return next;
	}
	public void setNext(LinkedListNode next) {
		this.next = next;
	}
}
```

ממשו את המתודות הבאות במחלקת העזר sw1.linkedlist.ListUtils. עליכם להגיש רק את המחלקה הזו בתוך תיקיות החבילה.

1. המתודה הבאה מוצאת ומחזירה, עבור רשימה באורך N, את האיבר ה-$\lfloor 2N/3 \rfloor$. למשל, אם הרשימה באורך 1 או 2 תוחזר החוליה הראשונה (1 הוא מקרה קצה משום ש- $\lfloor 2 \cdot 1/3 \rfloor < 1$), אם היא באורך 3 או 4 תוחזר החוליה השניה, אם היא באורך 5 תוחזר החוליה השלישית, וכך הלאה.

```
public static LinkedListNode findTwoThirdsNode(LinkedListNode head)
```

2. ממשו את המתודה הבאה, אשר מקבלת רשימה ומספר שלם אי שלילי n, ומחזירה את ראש הרשימה ב"הזזה מעגלית" של n חוליות לסוף הרשימה.

```
public static LinkedListNode shiftList(LinkedListNode head, int n)
```

~~שימו לב כי ייתכן ש-n גדול מאורך הרשימה, אך יש לדאוג לכך שהתוצאה הרצויה תתקבל לאחר מעבר יחיד על איברי הרשימה (למשל, אין טעם להעביר את אותו איבר לסוף הרשימה יותר מפעם אחת)~~. ניתן להניח כי n קטן או שווה לאורך הרשימה.

לדוגמא, עבור n=5 והרשימה (משמאל לימין):

```
1 2 3 4 5 6 7 8 9
```

לאחר ההזזה המעגלית הרשימה תהיה (משמאל לימין):

```
6 7 8 9 1 2 3 4 5
```

3. ממשו את המתודה הבאה, אשר מקבלת שתי רשימות ומאחדת אותן לרשימה אחת. ראש הרשימה המאוחדת יהיה בעל הערך הגדול מבין ראשי הרשימות המקוריות, ולאחר מכן יופיע לסירוגין איבר מכל אחת מהרשימות. ערך ההחזרה יהיה ראש הרשימה המאוחדת.

```
public static LinkedListNode mergeLists
                    (LinkedListNode head1, LinkedListNode head2)
```

שימו לב כי לא מובטח ששתי הרשימות באורך זהה ובמקרה יש לנהוג כמו בדוגמא למטה. אם ערכי שני ראשי הרשימות שווים ניתן לבחור שרירותית באיזו רשימה להתחיל.

לדוגמא, עבור הרשימות הבאות (משמאל לימין):

```
1 2 4

3 5 6 7 8 9
```

תוחזר הרשימה (משמאל לימין):

```
3 1 5 2 6 4 7 8 9
```

חלק ב' (58 נק')

בחלק זה נתרגל עבודה עם אוספים (Collections) ע"י מימוש מנוע פשוט להשוואה בין קבצי טקסט. בין קבצי העזר של התרגיל תוכלו למצוא את המחלקות `FileUtils` ו- `FileSimilarityTest`, ואת שלד המחלקה `FileIndex` (כולן בחבילה `sw1.filesimilarity`). יש להגיש רק את `FileIndex` ומחלקות עזר נוספות, אם כתבתם כאלה, בתוך תיקיות החבילה.

מנוע ההשוואה שלנו יקבל כקלט תיקיה במערכת הקבצים, יקרא את כל הקבצים בה, וישמור ב"אינדקס" עבור כל אחד מהקבצים את מספר המופעים של כל מילה (token) בו. לאחר מכן, נוכל להשוות בין הקבצים השמורים באינדקס.

4. המתודה `index()` במחלקה `FileIndex` קוראת את הקבצים ומוסיפה אותם לאינדקס, והקוד שלה כבר נתון. השלימו את מתודות העזר בהן היא משתמשת: המתודה `clearPreviousIndex()` מנקה נתונים שנשמרו בקריאה הקודמת ל- `index()`. המתודה `addFileToIndex(File file)` מוסיפה את הנתונים הרלוונטיים לגבי קובץ טקסט יחיד לשדות המחלקה. קריאת המילים מן הקובץ תתבצע בעזרת `readAllTokens(File file)` ממחלקת העזר `FileUtils`, שכבר נתונה לכם.

שימו לב, עליכם לבחור את מבני הנתונים המתאימים לייצוג המידע הדרוש (לשם כך, קראו גם את הסעיפים הבאים), תוך שימוש יעיל באוספים גנריים מתוך `Java collection framework`.

עבור קבצים שאינם מכילים מילים תקינות או שלא ניתן לקרוא מהם, תודפס הודעת שגיאה והם לא יתווספו לאינדקס.

5. המתודה `getCosineSimilarity(File file1, File file2)` מחזירה, עבור שני קבצים השמורים באינדקס, את ציון הדמיון שלהם `cosine similarity`, לפי הנוסחא הבאה:

$$\frac{\sum_{w \in \text{file1} \cap \text{file2}} A_w \cdot B_w}{\sqrt{\sum_{w \in \text{file1}} A_w^2} \cdot \sqrt{\sum_{w \in \text{file2}} B_w^2}}$$

הסבר: אנו מסתכלים על קבצי טקסט כעל וקטורים של מילים, ומחשבים את קוסינוס הזווית ביניהם כמדד לדמיון בין הקבצים. בהינתן מילה w, נסמן ב- A_w את מספר המופעים שלה ב-`file1` (כלומר, כמה פעמים היא חוזרת בתוצאת `readAllTokens`) וב- B_w את מספר המופעים שלה ב-`file2`. במונה אנחנו מחשבים את מכפלת מספרי המופעים עבור כל מילה הנמצאת בשני הקבצים, וסוכמים. לכן, אם יש הרבה מילים משותפות שחוזרות על עצמן, המונה יהיה גדול. במכנה אנחנו מחלקים בנורמות של כל אחד מהקבצים, כדי "לבטל" את השפעת גודל הקובץ ומספר החזרות של מילים בו.

בפרט, תוצאת הנוסחא היא 1 אם משווים קובץ מסוים לעצמו, ו- 0 אם משווים שני קבצים שאין ביניהם מילים משותפות כלל.

השלימו את מימוש המתודה. שימו לב כי לשם כך עליכם גם להשלים את מימוש מתודת העזר `verifyFile(File file)` אשר מקבלת כקלט קובץ ומחזירה `true` אם"ם זהו קובץ ששמור באינדקס. אחרת, עליה להדפיס הודעה למשתמש המתחילה ב- `[ERROR]` (בדומה להודעות השגיאה שמודפסות מהקוד הנתון לכם) ולהחזיר `false`. המתודה `getCosineSimilarity` משתמשת במתודה זו לבדיקה התחלתית של הקובץ.

6. השלימו את מימוש המתודה `getFilesBySimilarity(File file)`. מתודה זו מקבלת כקלט קובץ (כמו בסעיף הקודם, הוא צריך להיות קובץ ששמור באינדקס) ומחזירה את רשימת כל הקבצים באינדקס, ממוינת בסדר יורד לפי ציוני ה-`cosine similarity` שלהם (מהגדול = הדומה ביותר לקובץ הקלט, לקטן). הרשימה לא תכלול את קובץ הקלט.

כדי למיין את רשימת הקבצים השתמשו ב- `Collections.sort(...)` ובמחלקה משלכם שתממש את המנשק `Comparator` ותשווה בין שני קבצים `file1` ו-`file2` לפי הדמיון שלהם לקובץ הקלט `file`. היעזרו במתודה `getCosineSimilarity` כדי לחשב את הדמיון הזה, ובמתודה `Double.compare` כדי להשוות בין שני ערכי `double`.

שדרוג (רשות): כדי להימנע מלחשב את ה-`cosine similarity` של שני קבצים מסוימים פעמים רבות, תוכלו לשמור בשדה של המחלקה תוצאות השוואה שכבר חישבתם ולעשות בהן שימוש חוזר. מנגנון כזה נקרא `cache` או מנגנון `memoization`. ניתן גם לשמור תוצאות חישובי ביניים כמו הנורמה של כל קובץ.

7. השלימו את מימוש המתודה `getNumIndexedFiles()` המחזירה את מס' הקבצים השמורים כרגע באינדקס.

בדקו את עצמכם: המחלקה `FileSimilarityTest` מכילה תכנית בדיקה קצרה עבור המחלקה שכתבתם. כדי להריץ אותה, שימרו את התיקיה `simfiles` תחת תיקיית הפרויקט ב- `Eclipse` או תחת התיקייה ממנה אתם מריצים את קוד הג'אווה. תיקיה זו מכילה 3 קבצים. בהרצה אמור להתקבל הפלט הבא (המסלול המלא לקבצים יכול להשתנות כתלות במקום בו שמרתם את התיקייה):

```
Indexing C:\java\simfiles\file1.txt
Indexing C:\java\simfiles\file2.txt
Indexing C:\java\simfiles\file3.txt
Indexed 3 files.
similarity to C:\java\simfiles\file1.txt
C:\java\simfiles\file3.txt: 0.503
C:\java\simfiles\file2.txt: 0.324
```

התוצאה, אגב, מראה כי לפי המדד שלנו, שיר של רוברט פרוסט (ב-`file1`) דומה יותר לשיר אחר של אותו משורר, מאשר לשיר של שייקספיר, כפי שהיינו מצפים.

מומלץ לשנות ולהוסיף קבצים לצרכי הבדיקה.

בהצלחה!