תוכנה 1 – אביב 2022/23

תרגיל מספר 8

אוספים גנריים ו-collection framework

<u>הנחיות כלליות:</u>

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

את התרגיל הבא צריך להגיש באופן הבא:

- הגשה במערכת ה-Git תתבצע על פי ההנחיות שראיתם בתרגול 1.
  צרו את ה repository שלכם מתוך הקישור הבא:
  https://classroom.github.com/a/FxCBG18J
  יש לוודא שבתיקיית הגיט שלכם נמצאים הקבצים הבאים:
  a. קובץ פרטים אישיים בשם details.txt המכיל את שם המשתמש שלכם ב Moodle ואת מספר תעודת הזהות שלכם.
  b. 2 התקיות src ו-resources. יש להגיש את ה-repository ואת מבנה התיקיות שבתוכו בדיוק באותה היררכיה שקיבלת אותם

- הגשה במערכת ה Moodle (http://moodle.tau.ac.il/): עליכם להגיש את קובץ הטקסט assignment.txt ובו קישור לgit repository האישי שלכם.

<u>הנחיות כלליות נוספות לתרגיל:</u>

1. מומלץ ראשית לקרוא את כלל ההוראות עבור החלק שאתם ניגשים לפתור על מנת לוודא שהמימוש שלכם יהיה מותאם בצורה הטובה ביותר לדרישות התרגיל.
2. בכל אחד מחלקי התרגיל ניתן להוסיף שירותים ומחלקות לפי הצורך, אך אין לשנות חתימות של שירותים קיימים והגדרות של מנשקים (גם אין להוסיף או להוריד throws).
3. אין הגבלות על import-ים או שימוש במבני נתונים מסוימים, אך שימו לב שאתם מוחקים import-ים מיותרים או כאלה שאינם מתקמפלים בנובה.
4. בכל חלק קיים טסטר קצר המבצע בדיקות שפיות (חפשו בגוגל sanity tests). כדאי ומומלץ להוסיף בדיקות משלכם שכן הטסטרים הם בסיסיים ביותר ולא בודקים את כל המקרים.

חלק א' (50 נק')

בתרגיל זה עליכם לממש מבנה נתונים של היסטוגרמה באמצעות אוספים גנריים. נגדיר היסטוגרמה בתור מבנה נתונים אשר סופר מופעים של עצמים מטיפוס T כלשהו (טיפוס גנרי). הקוד ימומש בחבילה `il.ac.tau.cs.sw1.ex8.histogram`.

לדוגמא, עבור אוסף האיברים הבא: 1, 2, 3, 1, 2, ההיסטוגרמה תכיל את האיברים 1, 2, 3 ואת מספר המופעים שלהם.

יחד עם קבצי התרגיל מסופק לכם הממשק IHistogram את השירותים הבאים (כולל החוזה של כל שירות):

```
public void addItem(T item);
public boolean removeItem(T item);
public void addAll(Collection<T> items);
public int getCountForItem(T item);
public void clear();
public Set<T> getItemsSet();
public int getCountsSum();
```

א. השירות `addItem` מוסיף מופע אחד של הפריט item להיסטוגרמה.

ב. השירות removeItem מוריד מופע אחד של הפריט item. השירות מחזיר true אם הפריט הוסר, ו false אם הפריט לא היה קיים, ולכן גם לא הוסר.

ג. השירות `addAll` מוסיף אוסף של פריטים להיסטוגרמה (האוסף יכול להכיל את אותו הפריט יותר מפעם אחת. בנוסף, לא ניתן להניח דבר על הקיום של פריטים אלה בהיסטוגרמה לפני הקריאה ל addAll).

ד. השירות getCountForItem יחזיר את מספר הפעמים שהאיבר item נספר. אם item הוא פריט שלא קיים בהיסטוגרמה, יוחזר הערך 0.

ה. השירות `clear` ירוקן את ההיסטוגרמה מכל האיברים והספירות (כלומר, לאחר clear, השירות `getCountForItem` יחזיר ספירה 0 לכל איבר).

ו. השירות `getItemsSet` יחזיר אוסף מטיפוס Set אשר מכיל את כל האיברים בהיסטוגרמה אשר מספר המופעים שלהם גדול מ-0, ללא הספירות שלהם.

ז. השירות getCountsSum יחזיר את סכום המופעים של איברי ההיסטוגרמה.

בנוסף לשירותים שפורטו כאן, המנשק IHistogram מממש את Iterable, כך שתצטרכו לממש את הפונקציה iterator כפי שיוסבר בהמשך.

סעיף 1 (25 נק'):

ממשו את המחלקה HashMapHistogram אשר מממשת את המנשק IHistogram עבור כל טיפוס T המממש את המנשק Comparable (כלומר, T יכול לקבל ערך של כל מחלקה המממשת את המנשק Comparable. נזכיר כי הטיפוסים המובנים הבסיסיים כמו Integer ו String מממשים מנשק זה). לדרישה הזו יש סיבה אותה ניראה בהמשך.

פרקטית, זה אומר שנגדיר את HashMapHistogram באופן הבא:

```
public class HashMapHistogram<T extends Comparable<T>> implements
IHistogram<T>
```

משמעות הגדרה הזו: המנשק IHistogram מצריך פרמטר גנרי. את הפרמטר הגנרי נגדיר כ T, ונוסיף עליו את האילוץ שהוא צריך לממש את המנשק Comparable<T>. כלומר, T יהיה פרמטר גנרי מתאים אם הוא Comparable עם עצמים אחרים מטיפוס T. את התחביר הזה ניראה בהמשך הקורס.

המימוש יעשה באמצעות הכלה (aggragation) של HashMap, כלומר, כל מופע של HashMapHistogram יכיל שדה מטיפוס HashMap. שדה זה יהיה אחראי על שמירת הספירות עבור כל אובייקט מטיפוס T.

סעיף 2 (25 נק')

המנשק IHistogram יורש מהמנשק Iterable, מה שמחייב את HashMapHistogram לממש את השירות iterator(). הגדרת המנשק היא:

```
public interface IHistogram<T> extends Iterable<Map.Entry<T, Integer>>
```

כלומר, האיטרטור שההיסטוגרמה אמורה להחזיר עובר על זוגות (מטיפוס Map.Entry). הזוגות מייצגים מפתחות וערכים בהיסטוגרמה. תזכורת - השירות entrySet של המחלקה Map מחזיר אוסף של איברים מטיפוס Map.Entry.

האיטרטור יחזיר את הזוגות על פי הסדר הטבעי של המפתחות (האיברים מטיפוס T). לדוגמא – אם ההיסטוגרמה מכילה מחרוזות, הסידור יהיה לפי סדר לקסיקוגרפי עולה. אם ההיסטוגרמה מכילה מספרים שלמים, הסידור יהיה סידור המספרים בסדר עולה.

לצורך כך עליכם לממש:

א. מחלקה חדשה המממשת את המנשק Iterator. שם המחלקה הוא HashMapHistogramIterator והשלד שלה נתון לכם. אין צורך לממש את השירות remove.

ב. כדאי לממש גם Comparator בשביל סידור האיברים שהאיטרטור יחזיר. ניתן לממש מחלקה זו כמחלקה פנימית במחלקת האיטרטור או כמחלקה בקובץ Java נפרד משלה.

שימו לב, ניתן, ואף כדאי, להעביר למחלקות ה-Comparator וה-Iterator את המידע הרלוונטי מתוך המופע של HashMapHistogram וכן להשתמש בהכלה של אוספים לפי הצורך.
להבין איזה מידע כל אובייקט צריך לקבל הוא חלק מהאתגר מהתרגיל, חישבו על כך בעת בניית המחלקות.

שלד כל המחלקות אותן אתם נדרשים לממש נתון לכם בחבילה il.ac.tau.cs.sw1.ex8.histogram המופיעה בקבצי התרגיל.

העזרו ב HashMapHistogramTester בשביל לבדוק את עצמכם, והוסיפו לו בדיקות משלכם.

חלק ב' (50 נק')

בחלק זה נתרגל עבודה עם אוספים (Collections) ע"י מימוש מחלקה אשר מנתחת קבצי טקסטים ומזהה מילים משמעותיות בכל טקסט וממיינת את הטקסטים לפי הדמיון הלקסיקלי (מילולי) שלהם.

הקוד בחלק זה ימומש בחבילה `il.ac.tau.cs.sw1.ex8.tfIdf` אך ישתמש גם בקוד של ההיסטוגרמה אותה מימשתם בחלק א' (כלומר, ישתמש בקוד שמופיע בחבילה אחרת – וודאו ששני החלקים האלה מופיעים אצלם באותו הפרוייקט ב eclipse)

המחלקה שלנו תבצע את הניתוח באופן הבא: עבור תיקיית קבצים כלשהי, היא תקרא את התוכן של כל קובץ ותשמור את הנתונים ההרלוונטיים לכל קובץ (אנחנו קוראים לזה – ביצוע אינדוקס, `indexing`). פעולה זו תקרא פעם אחת בלבד, ולאחר מכן כל הפעולות תתבצענה על תוכן האינדקס שתיצרו.

<u>סעיף 1 (הבדיקה של סעיף מתבצעת דרך הסעיפים האחרים ולכן לא ניתן לו ניקוד)</u>

המתודה`indexDirectory()` במחלקה `FileIndex` קוראת את הקבצים ומוסיפה אותם לאינדקס. המימוש של פונקציה זו נתון לכם חלקית ואתם רשאים לערוך אותו. קריאת המילים מן הקובץ תתבצע בעזרת `readAllTokens(File file)` ממחלקת העזר `FileUtils`, שכבר נתונה לכם. שימו לב שהמחלקה זורקת שגיאה ואין לשנות זאת, ניתן במקום זאת להיעזר ב `try` ו`catch` אם יש צורך בכך.

המטרה של שירות זה היא לקרוא את תוכן כל הקבצים, לנתח אותו ולשמור אותו כך שהמימוש של שאר השירותים במחלקה `FileIndex` יהיה יעיל ומהיר. בתרגיל זה, חלק מהשירותים צריכים להיות מאוד יעילים, כך שבשלב ה `index` נעשה את החישובים הכבדים ונשמור אותם במבני נתונים מתאימים. הגדרה נכונה של מבני הנתונים, והוצאת קוד משותף למתודות פרטיות תהפוך את המימוש של חלק מהשירותים המוגדרים ב `fileIndex` לפעולות שליפה פשוטות ממבני נתונים. אתם לא נמדדים על זמן הריצה של המתודות, אך זוהי שיטת העבודה הרצויה בתרגיל זה.

שימו לב, עליכם לבחור את מבני הנתונים המתאימים לייצוג המידע הדרוש. לשם כך, **עליכם לקרוא ולהבין את כל הסעיפים של חלק ב'**, ורק לאחר מכן לקבל את ההחלטה על מבני הנתונים שישמשו אתכם. עליכם להשתמש ביעילות במבני נתונים גנריים מתוך `Java collection framework`. בפרט, <u>עליכם להשתמש במבנה הנתונים `HashMapHistogram`</u> אשר מומש בחלק א' על מנת לשמור את מספר המופעים של ה token-ים בכל קובץ.

הערות נוספות:

- שם תיקיית הקבצים יהיה שם חוקי של תיקיה המכילה לפחות קובץ אחד.
- השירות `readAllTokens` של `FileUtils` מבטל סימני פיסוק ומחזיר מילים שאינן ריקות, אין לבצע עיבוד או סינון נוסף בגוף המימוש שלכם: כל המילים שחוזרות ע"י `readAllTokens` הן חוקיות מבחינתכם.
- הניחו כי כל קובץ מכיל לפחות מילה חוקית אחת.
- החוזה של כל שירות ב `FileIndex` נתון בשלד הקוד – **קריאת החוזה היא חלק מהתרגיל ומהווה השלמה לפירוט הקצר שמופיע עבור כל שירות.**

<u>סעיף 2 (10 נק׳)</u>

ממשו את שלושת השירותים הבאים:

1. **`getCountInFile`** אשר מקבל מחרוזת `fileName` ומחרוזת `word` ומחזיר את מספר המופעים של המילה `word` בקובץ `fileName`. עבור מילה שאינה מופיעה בקובץ יוחזר הערך 0.

2. **`getNumOfUniqueWordsInFile`** אשר מקבל מחרוזת `filename` ומחזיר את מספר המילים השונות שמופיעות בקובץ `filename`.

3. השירות **`getNumOfFilesInIndex`** אשר מחזיר את מספר הקבצים שמופיעים באינדקס.

חתימת השירותים:

```
public int getCountInFile(String word, String fileName) throws FileIndexException
public int getNumOfUniqueWordsInFile(String fileName) throws FileIndexException
public int getNumOfFilesInIndex()
```

הנחיות כלליות לסעיף זה והסעיפים הבאים:

- בכל שירות המקבל שם של קובץ, המחרוזת `fileName` מכילה שם קובץ בלבד (ללא נתיב), ויש לחפש אותו בתיקיה עליה בוצע שלב ה `index` (ראו דוגמת שימוש במחלקת הטסטר). עליכם לקרוא את התוכן של כל קובץ פעם אחת בלבד, בשלב האינדקס. לאחר מכן, עליכם לעשות שימוש באינדקס בשביל לממש את השירותים של המחלקה.
- בכל שירות המקבל שם של קובץ, במידה ושם הקובץ אינו קיים בתיקיה זו, יש לזרוק חריג מטיפוס `FileIndexException` (מומש עבורכם) עם הודעה אינפורמטיבית לבחירתכם.
- בכל שירות שמקבל מילה `word` יש להמירה ל `lowercase` לצורך ביצוע החיפוש באינדקס.

סעיף 3 (5 נק׳)

בסעיף זה נתחיל לממש את הממד tf-idf שבו נשתמש בהמשך התרגיל.

עבור אוסף מסמכים (כל מסמך מופיע בקובץ נפרד), נגדיר את המדדים הבאים:

המדד tf (term frequency) מחושב עבור מילה (word) ומסמך (doc) באופן הבא:

$$\text{tf}(word, doc) = \frac{\text{number of repetitions of } word \text{ in } doc}{\text{number of words (with repetitions) in } doc}$$

המדד idf (inverse document frequency) מחושב עבור מילה (word) באופן הבא:

$$\text{idf}(word) = \log\left(\frac{number\ of\ documents}{number\ of\ document\ containing\ word}\right)$$

המדד tf-idf הוא מדד אשר נותן ציון רלוונטיות לכל מילה word במסמך doc, והוא מחושב ע״י מכפלת tf ב idf. ככל שמילה מסויימת מופיעה יותר פעמים במסמך מסויים, היא יותר משמעותית במסמך זה - לזה אחראי החלק של ה tf. מצד שני, אם המילה מופיעה בכל המסמכים שיש לנו במאגר, זה אומר שהיא פחות ״מיוחדת״ עבור המסמך עליו אנחנו מסתכלים – ועל החלק הזה אחראי ה idf. אם המילה מופיעה בכל מהמסכים במאגר, ה idf שלה הוא log(1), כלומר 0. לכן, המדד tf-idf יגדל ככל שהמילה מופיעה בפחות מסמכים.

מדד זה שימושי לכל מני משימות בעיבוד טקסטים, כולל באחזור (retrieval) של מסמכים ומציאת דמיון בין מסמכים. הסבר מפורט יותר, כולל הסבר מלא לדוגמא שבה השתמשנו בחלק הראשון של תוכנית הבדיקות FileIndexTester ניתן למצוא כאן:

https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3

ממשו את השירות getTF אשר מקבל מילה word ושם של קובץ fileName ומחזיר את ערך ה tf.

חתימת השירות:

```
public double getTF(String fileName, String word) throws FileIndexException
```

העזרו בשירות calcTF אשר מבצע את חישוב ה TF עבור שני הפרמטרים המצייגים את המונה והמכנה בנוסחא. השירות מקבל שני שלמים ומחזיר double, והוא ניתן לכם על מנת שכל החישובים העשרוניים שלכם שלכם יבוצעו באותה הצורה.

סעיף 4 (5 נק׳)

ממשו את השירות getIDF אשר מקבל מילה word ומחזירה עבורה את ערך ה idf.

חתימת השירות:

```
public double getIDF(String word)
```

כמו בסעיף הקודם, נתונה לכם פונקציית עזר בשם calcIDF אשר מבצעת את החישובים העשרוניים ומקבלת שני שלמים המייצגים את המונה והמכנה בשבר המופיע בנוסחת ה idf.

בשלב הזה אתם יכולים להריץ את החלקה הראשון בתוכנית הבדיקות fileIndexTester. החלק הראשון מופיע בפונקציה testFileIndexFirstPart.

סעיף 5 (10 נק׳).

ממשו את השירות getTopKMostSignificantWords אשר מקבל שם של קובץ filename ומחזיר רשימה של k זוגות של מילה-ציון tf-idf. הזוגות שיחזרו יהיו זוגות של המילים הכי ״משמעותיות״ (בעלות ציון tf-idf הכי גבוה) ב filename, מוסדרים בסדר יורד של ציון tf-idf. אם יש שתי מילים בעלות ציון tf-idf זהה, המילה שתופיע קודם היא המילה שמופיעה קודם בסידור לקסיקוגרפי.

מכיוון שהמימוש של שירות זה מצריך מיון, ומיון הוא פעולה יקרה, נרצה לבצע מיון יחיד עבור כל קובץ. אפשרות אחת היא לבצע את המיון בשלב האינדקס, ואפשרות נוסף היא לבצע את המיון בקריאה הראשונה לשירות זה עבור קובץ מסוים. בכל מקרה, עליכם להימנע מביצוע מיונים במימוש שירות זה.

```
public List<Map.Entry<String, Double>>
      getTopKMostSignificantWords(String fileName, int k) throws FileIndexException
```

הערה: השירות getTFIDF כבר מומש עבורכם (שירות זה עושה שימוש בשירותים שמימשתם בסעיפים 3+4).

סעיף 6 (10 נק׳)

נרצה לבדוק דמיון מילולי בין הקבצים השונים המופיעים באינדקס.

בתרגיל בית 5 ייצגתם מילים באמצעות מערכים של מספרים, וחישבתם cosine similarity בין ייצוגים של מילים שונות בשביל למצוא מילים דומות מבחינת ההקשרים שבהם הן מופיעות (שזהו בעצם דמיון סמנטי – דמיון במשמעות). כעת, נשתמש בשיטה דומה בשביל לחשב דמיון בין מסמכים. על מנת להשתמש ב cosine similarity, עלינו לייצג כל מסמך ע״י וקטור מספרי, ואת הייצוג הזה נבנה ע״י שימוש ב tf-idf.

בייצוג ע״י וקטור אנחנו צריכים להתייחס לאוצר המילים (שקובע את גודל ה-וקטור) ולתוכן של כל תא. לשם פשטות, נניח כי אוצר המילים שלנו כולל את כל המילים שהופיעו בכל הקבצים שמופיעים באינדקס, ולכן אורך הוקטור יהיה כמספר המילים השונות שמופיעות בכל הקבצים, נסמן מספר זה ב N.

כל מסמך ייוצג, אם כך, ע״י וקטור באורך N. עבור מסמך doc כלשהו, התא ה-i בוקטור יכיל את ערך ה tf-idf עבור המילה ה i באוצר המילים.

שימו לב – אין באמת צורך לבנות וקטורים בגודל כל אוצר המילים. הנוסחא של cosine similarity היא:

$$cosSim(A,B) = \frac{\sum_{i=0}^{n} A_i * B_i}{\sqrt[2]{\sum_{i=0}^{n} A_i^2 * \sum_{i=0}^{n} B_i^2}}$$

התאים שמעניינים אותנו בוקטורים שמייצגים את A ו B הם שבהם יש ערך שאינו 0 או ב A או ב B. אחרת, אם בתא ה i יש 0 גם ב A וגם ב B, הם לא משפיעים על תוצאת החישוב. בנוסף, אין חשיבות לסדר האיברים ב-וקטור. מה שצריך הוא לוודא שבחישוב המונה, הערך המתאים למילה w כלשהי במסמך A יוכפל בערך המתאים למילה w כלשהי במסמך B.

מסיבה זו, אתם לא נדרשים לממש שירות אשר מייצג מסמך כווקטור. בחישובים שלכם אתם יכולים לעשות שימוש ישיר בהיסטוגרמות או במבני נתונים אחרים שאתם מחזיקים בשביל לחשב את ה cosine similarity.

ממשו את השירות getCosineSimilarity המקבל שמות של שני קבצים, ומחזיר את תוצאת חישוב ה cosine similarity בין יצוגי ה df-idf שלהם.

```
public double getCosineSimilarity(String fileName1, String fileName2)
            throws FileIndexException
```

סעיף 7 (10 נק׳)

כעת, נרצה לקבל את המסמכים הכי דומים למסמך כלשהו על פי דמיון בוקטורי ה tf-idf. בדומה לסעיף 5, נממש שירות אשר מחזיר את k המסמכים הכי דומים למסמך filename כלשהו, יחד עם הציון של מידת הדמיון (ערך ה cosine similarity).

חתימת השירות:

```
public List<Map.Entry<String, Double>>
      getTopKClosestDocuments(String fileName, int k) throws FileIndexException
```

הנחיה: ניתן, אך לא חובה (בניגוד לסעיף 5) לשמור את תוצאת המיון עבור כל קובץ, כך שיתבצעו פעם אחת בלבד. לחילופין, ניתן לבצע את החישובים בכל פעם מחדש בקריאה לפונקציה (אנחנו מניחים שמספר הקבצים הוא קטן ולכן שמירת תוצאות המיונים מראש לא תשפר משמעותית את זמן הריצה של השירות).

במציאות, את ההחלטה לגבי ביצוע פעולות בזמן ריצה מול שמירת תוצאות החישובים נבצע לפי דרישות המערכת וגודל בסיס הנתונים. יש לנו 2 ״ישויות״ עליהן אנחנו צריכים לבצע חישובים – חישוב ברמת המילה (המילים הכי ״משמעותיות״) וחישוב ברמת הקובץ (הקבצים הכי ״דומים״), ולכן גודל אוצר המילים ומספר הקבצים הם משמעותיים בקבלת ההחלטה. כמו כן, צריך להבין מה הסיכוי לעשות שימוש חוזר בערכים שנשמרים מראש.

טסטרים:

כמו בכל תרגיל אחר, הטסטרים הם טסטרים בסיסיים שאינם בודקים את כל המקרים, וריצה מוצלחת שלהם מהווה תנאי הכרחי אך לא מספיק בשביל לוודא שהתרגיל שלכם עובד כנדרש. הוסיפו בדיקות משלכם!

(אין בעיה לשנות את קובץ הבדיקות הקיים, רצוי להוסיף בדיקות חדשות ולא לשנות את הקיימות)

בהצלחה!