

תוכנה 1 – אביב 2019/20

תרגיל מספר 5

הנחיות כלליות:

- קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.
- הגשת התרגיל תעשה במערכת ה-moodle בלבד (<http://moodle.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש aviv1 יקרא הקובץ aviv1_hw5.zip). קובץ ה-zip יכול:
 - קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
 - את תיקיית **src** ובתוכה היררכיית התיקיות כפי שקיבלתם בקובץ הזיפ:

```
src\il\ac\taucs\sw\ex5
```

 כאשר בתיקייה ex5 יהיה הקובץ BigramModel.java. ניתן גם להשאיר בתיקיית ex5 את הקובץ BigramModelTester.java, אך לא חובה, כיוון שקובץ זה אינו נבדק (הוא נועד לסייע לכם בבדיקה עצמית). אין לצרף קבצים ותיקיות נוספות. שימו לב, כי **אין** לצרף גם את תיקיית ה-resources שקיבלתם בקובץ הזיפ. היא נועדה גם כן לבדיקה מקומית בלבד.
- נא לא להשתמש בפקודה (`System.exit()`)! היא מחבלת בבדיקות אוטומטיות. אין כל צורך לעשות בה שימוש, כאשר תוכניות יכולות להסתיים ע"י הגעה לסוף מתודת ה-main או בפקודות `.return`.

יצירת פרוייקט והגשה: חזרו על ההוראות ממטלה 3 לגבי יצירת פרוייקט וייבוא הקבצים. התהליך הוא זהה במטלה זו: יש ליצור פרוייקט ג'אווה חדש באקליפס (ולשים לב, למיקום של workspace במחשבכם). כעת יש להיכנס לתיקיית הפרויקט במחשב, ולהעתיק לשם את תוכן קובץ הזיפ המצורף, כך שתתווסף תיקיית resources ותיקיית src תידרס. תיקיית ה-src הזו היא התיקייה שתצרפו לזיפ בתום כתיבת הקוד. חזרו כעת לאקליפס, ובלחיצה ימנית על הפרוייקט בחרו `.refresh`.

בתרגיל זה נבנה מודל בסיסי של שפה (השפה האנגלית) מתוך טקסט נתון, תוך שימוש בסטטיסטיקה פשוטה של הופעת צמדי מילים בטקסט. לדוגמא, אם הטקסט שאנו לומדים עליו הוא " here comes the sun", הרי שבשפה שלנו יש 4 מילים, והצמדים <the,sun>, <comes, the>, <here, comes> מופיעים כל אחד פעם אחת. הצמד <here,sun> לא מופיע בקובץ שלנו, ולכן סה"כ ראינו אותו 0 פעמים. מודל השפה שלנו למעשה יכול את אוצר המילים ואת מספרי המופעים של כל צמד מילים.

שימו לב, בתוכנית זאת עליכם להשתמש **במערכים בלבד**, ולא במבני נתונים גנריים כמו Lists/Maps/Sets וכו'. שימוש במבני נתונים גנריים יכול לגרום להורדת ניקוד משמעותית עד כדי ציון נכשל. בפרט, אין להשתמש ב `Arrays.asList` שכן שירות זה מייצר אוסף גנרי מטיפוס `List`.

- בתרגיל זה אתם יכולים להניח שכל שמות הקבצים הם חוקיים ואין צורך לממש טיפול נפרד למקרה שקריאה/הכתיבה לקובץ נכשלת. כמו כן, הניחו כי כל הקלט לכל מתודה הוא חוקי וכמצופה, אלא אם כן צויין אחרת בגוף הפונקציה ויש התייחסות מפורשת לטיפול בערכים שאינם חוקיים.
- מותר לכתוב מתודות עזר, אך יש לשמור את כולן באותו הקובץ, ולא ליצור עבורן מחלקות חדשות.
- חשוב **לעבוד עם המחלקה BufferedReader במקום Scanner** עבור קריאת הקבצים (הסבר לכך מצורף בסוף קובץ ההנחיות). אופן השימוש במחלקה זו מתואר במדריך קלט/פלט ללימוד עצמי באתר הקורס.
- דרך נוחה לקבל את מערך המילים בשורה, לאחר שקראתם שורה דרך `BufferedReader` היא להשתמש במתודה `split` של `String`.
- אם פעולת הקריאה תיכשל (התוכנית תעוף על שגיאה, או שפשוט לא יקרא כלום), חשוב לוודא קודם כל (לפני שמתייחסים) שהתוכנית מוצאת את הקובץ במיקום הנכון. שימו לב, כי בנתיבים רלטיביים, נקודת היחוס אליה מצמידים את הסיומת הרלטיבית יכולה להיות שונה כאשר תעברו להריץ את הקוד מהמחשב שלכם לשרתי האוניברסיטה (בגלל אופן ההרצה על השרת, וקונקרטיית התיקה שממנה ניתנת פקודת ההרצה בשורת הפקודה בלינוקס לעומת מה שקורה באקליפס), כך שיתכן שבעיה כזו תתעורר רק כאשר תעברו למערכת הפעלה אחרת (אבל בהחלט סביר שתתעורר עוד בהרצה על מחשבכם). דרך פשוטה לדבג (בטסטר, ולא בקוד שתגישו לבדיקה) היא להשתמש בפקודה שמובאת במדריך הקלט/פלט שמדפיסה את "התיקה הנוכחית", שאל הנתיב שלה מוצמד כסיפא הנתיב הרלטיבי שסיפקתם לקובץ. כך אתם יכולים לבדוק אם יוצא הנתיב שאליו התכוונתם, ואם לא, לשנות את הנתיב הרלטיבי שאתם מספקים בהתאם. בשביל כמה שיותר גמישות בהרצה, עדיף לספק נתיבים בתור ארגומנטים לתוכנית, ולא לקודד ישירות את הטסטר, כך אפשר לשנות את הנתיב רק על ידי שינוי פקודת ההרצה, בלי לשנות את הקוד. שימו לב שסינטקס של נתיב יכול לכלול גם "לעלות חזרה למעלה" בהיררכיית התיקיות.
- עוד סיבה נפוצה לכך שלא מצליחים לקרוא כלום מהקובץ היא שבמתודה אחרת (למשל אחת שכותבת לקבצים) דרסתם את הקובץ וכעת הוא ריק. אז תוודאו קודם שבקובץ שאתם קוראים ממנו עדיין יש טקסט, ושהוא לא הפך לריק. עוד סיבה נפוצה לבעיות קריאה היא הפרת ההנחיות ושימוש ב-`Scanner` במקום `BufferedReader`.
- אם התוכנית קורסת על שרתי האוניברסיטה עם שגיאה המדווחת על מחסור בזיכרון (אל חשש- זאת שגיאה שלא אמורה להופיע בכל מקרה בסביבה של הבודקים), פנו לצוות System להנחיות לגבי שינוי הגדרות הזיכרון שניתן להרצת תוכניות הג'אווה שלכם.
אפשר גם לנסות קודם בעצמכם את הפקודה `setenv _JAVA_OPTIONS -Xmx256m`.
- כמובן, שתמיד קיים סיכוי קטן שהבעיה נובעת מכך שמימשתם לוגיקה כלשהי באופן לא יעיל בצורה קיצונית עם סיבוכיות אקספוננציאלית, אז חשוב לקחת בחשבון גם את האפשרות הזאת.
- ניתן להניח לכל אורך התרגיל אלא אם נאמר אחרת שבכל קלט של טקסט, בין כל שתי מילים עוקבות באותה שורה יש בדיוק רווח אחד בודד.

בניית מודל שפה

המחלקה `BigramModel` אשר נמצאת בחבילה `il.ac.tau.cs.sw1.ex5` מייצגת את מודל השפה. מחלקה זו מכילה שני שדות:

שדה	משמעות
<code>String[] mVocabulary</code>	מערך שמכיל את כל המילים באוצר המילים
<code>int[][] mBigramCounts</code>	מערך דו מימדי המכיל את מספר הפעמים שבו נראה כל צמד מילים בקובץ ממנו למדנו את מודל השפה. האיבר במקום ה <code>j</code> , <code>i</code> יהיה מספר הפעמים שראינו את המילה ה <code>j</code> מופיעה אחרי המילה ה <code>i</code>

(1) [15 נקודות] ממשו את המתודה `buildVocabularyIndex` המקבלת נתיב לקובץ (כולל שם הקובץ כמובן), ומחזירה מערך מחרוזות (`vocabulary`) אשר יהווה את אוצר המילים שלנו. המערך `vocabulary`:

- יכיל עד 14,000 מילים חוקיות מתוך הקובץ. (השתמשו בקבוע שהוגדר עבור מספר המילים המקסימלי).
 - מילה חוקית היא מילה העונה לאחת משתי ההגדרות:
 - i. מילה שמכילה לפחות אחת בשפה האנגלית.
 - ii. מספר שלם (למשל, 13 או 1984). מבחינת מקרי קצה, מספר שלם מוגדר בקונטקסט שלנו כרצף ספרות ללא כל תו אחר. מותר שהמספר יתחיל ב-0, אך אסור סימן פלוס או מינוס בהתחלה או נקודה עשרונית. אם יש תווים נוספים מלבד ספרות במילה, אז אם אחד מהם הוא אות אנגלית מדובר במילה מהסוג הראשון. ואם יש תווים נוספים כך שאף אחד מהם אינו אות אנגלית, אז לא מדובר במילה חוקית.
 - כל מילה חוקית שמכילה לפחות אחת באנגלית יש להמיר ל lowercase.
 - כל מספר שלם יומר למחרוזת "some_num" (השתמשו בקבוע שהוגדר בתחילת המחלקה). לפיכך 55 ו-67 הם בעצם אותה מילה: "some_num", וממופים לערך יחיד במילון.
 - יש להתעלם מכל המילים הלא חוקיות.
 - הכוונה בהמרה היא לא שינוי "פיזי" בקובץ הקלט, אלא שהתוכנית שלכם צריכה לרוץ על הקלט באופן זהה לחלוטין לריצתה על גירסה של הקלט בה כל המילים בו הוחלפו ל-lowercase וכל המספרים הוחלפו ב-"some_num". באופן דומה, מילים לא חוקיות לא נמחקות, אלא מתעלמים מהן במהלך יצירת מילון המילים. אך אם בין שתי מילים חוקיות ישנה מילה לא חוקית, אז זה לא נכון להחשיב את שתי המילים האלה כעוקבות, כאילו המילה הלא חוקית כלל לא קיימת.
 - אם נצמד למילה מסוימת (שיש בה אותה אנגלית אחת לפחות) סימן כלשהו, כמו סימן פיסוק, אז לשם הפשטות, אנחנו נחשיב אותה כמילה נפרדת למילה ללא הסימן. כלומר המילה "man" ו-"man's" הינן שתי מילים שונות. אותו דבר לגביי "java" ו-"java".
 - יש להיצמד להוראות ולא להתייחס למשמעויות של מילים. כלומר המילים "gone" ו-"gone.." י- "gone" וגם "gone." נחשבות כשלוש מילים שונות (על אף שבפרשנות שלנו הן לכאורה זהות קונספטואלית). אותו דבר בכל הנוגע לכל סימני הפיסוק. כלומר מספיק שתו אחד שונה בין שתי מילים, ומדובר במילים שונות (בודקים אחרי שעשינו כבר את ההמרות: אותיות גדולות לקטנות, מספרים וכו'). וגם אם שני תווים נראים כמעט זהים כמו למשל גרשיים עם הטייה לכיוונים מעט שונים, אז אלה בכל זאת שני תווים שונים ולכן נמצאים במילים שונות.
 - מותר, אך לא חובה, להיעזר במתודה Character.isDigit. במחרוזות ישנן גם המתודות השימושיות charAt ו-toCharArray, שמומלץ להכיר (והן אכן הוזכרו בתרגולים).
 - ניתן להניח שבבדיקה לא יהיו שום בעיות קידוד.
 - זכיר את ההנחיה הכללית לכל התרגיל: ניתן להניח שבין כל שתי מילים עוקבות באותה שורה יש רווח אחד בודד.
- נכניס את המילים החוקיות לתוך המערך vocabulary (בגירסה המומרת של כל מילה, בהתאם להנחיות הקודמות) על פי סדר הופעתן בקובץ (המילה הראשונה תיכנס לאינדקס 0, וכן הלאה). במידה ומילה מופיעה פעמיים, היא תישמר רק פעם אחת, תחת האינדקס שבו נשמרה בפעם הראשונה שנראתה (למשל, אם המילה היא המילה השלישית בקובץ ונכנסה ל vocabulary תחת האינדקס 2, בפעם הבאה שנראה אותה ה vocabulary לא ישתנה).
- במידה והקובץ מכיל יותר מ-14K מילים חוקיות שונות, נשמור רק 14K מילים ב vocabulary (זה לא אומר שאנחנו מתעלמים במטלה הזאת משארית הקובץ אם הגענו לנקודה בה יש 14K מילים שונות, שכן הוא רלוונטי לשאלות אחרות). אם הקובץ מכיל פחות מ-14K מילים שונות, גודלו של המערך vocabulary יותאם למספר המילים השונות בקובץ.

חתימת המתודה:

```
public String[] buildVocabularyIndex(String fileName) throws
IOException
```

- (2) **[15 נקודות]** ממשו את המתודה `buildCountsArray` המקבלת נתיב לקובץ (`filename`) ומערך מחרוזות (`vocabulary`). המתודה תחזיר מערך דו מימדי של מספרים שלמים (`int`) אשר יכיל את מספר המופעים של צמדי מילים בקובץ `filename` על פי הפירוט הבא:
- אם `vocabulary[x] = word1` ו `vocabulary[y]=word2`, אזי `bigramCounts[x][y]` יכיל את מספר הפעמים שבהן המילה `word2` הופיעה **מיד** אחרי `word1` באותו המשפט.
 - הניחו כי בקבצי הקלט שלכם כל משפט יתחיל בשורה חדשה. כמו כן, כל שורה חדשה שהיא לא ריקה מתחילה משפט חדש. מכך נובע שאם שתי מילים הן בשורות שונות (כלומר יש ביניהן לפחות פעם אחת ירידת שורה) אז הן לא נחשבות כצמד מילים שנאסף לסטטיסטיקה.
 - הסטטיסטיקות תיאספנה רק עבור מילים שנמצאות באוצר המילים. אם הקובץ מכיל יותר מ-14K מילים, צמדים שמכילים מילה שלא נמצאת ב `vocabulary` לא יספרו. שימו לב: זה לא אומר שלא ממשכים לסרוק את הקובץ לאחר שהגענו ל-14 אלף מילים שונות, אלא רק שמילים חדשות מעבר לנקודה הזאת כבר לא נחשבות חוקיות (אך עדיין סופרים זוגות של מילים חוקיות עוקבות עד סוף הקובץ).
 - אם המילה ה `y` מעולם לא ניראתה אחרי המילה ה `x`, אזי `bigramCounts[x][y]` יכיל את הערך 0.
 - שים לב, בשאלה זו ובקודמת אתם מתבקשים רק לבנות מערכים מתאימים ולהחזיר אותם, אך לא לשים אותם בשדות `mVocabulary` ו `mBigramCount`!
 - גם בשאלה זו יש "להמיר" את המילים בקלט כמו בשאלה 1 (להפוך אותיות גדולות לקטנות, להפוך מספרים ל-`some_num`), ולהתייחס רק למילים לאחר ה"המרה". כמוכן שלא צריך לעשות שינויים בקלט עצמו, אלא רק במה שמוכנס בסוף למילון.
 - נזכיר שוב: אם בין שתי מילים חוקיות ישנה מילה לא חוקית, אז זה **לא נכון** להחשיב את שתי המילים האלה כעוקבות, כאילו המילה הלא חוקית כלל לא קיימת.
 - נזכיר את ההנחיה הכללית לכל התרגיל: ניתן להניח שבין כל שתי מילים עוקבות באותה שורה יש רווח אחד בודד.

חתימת השירות:

```
public int[][] buildCountsArray(String fileName, String[] vocabulary)
throws IOException
```

- (3) **[10 נקודות]** החל מסעיף זה, ניתן להניח שלפני כל קריאה לאחד השירותים, הופעל השירות `initModel`. שירות זה, שמימושו כבר נתון לכם, מקבל שם של קובץ (שכרגיל נניח שכולל נתיב לקובץ ולא רק את שמו, אך הפעם בלי הסיומת) ומייצר ממנו מודל שפה ע"י קריאה לשני השירותים מסעיפים 1 + 2. אוצר המילים וטבלת הספירות ישמרו בשדות `mVocabulary` ו `mBigramCounts`, אליהם תוכלו לגשת מכל שירות אשר מובטח בתנאי הקדם שלו שהמודל מאותחל.
- ממשו את המתודה `saveModel` המקבלת מחרוזת `fileName` ושומרת את המודל הנלמד (אוצר המילים וספירות הזוגות) לתוך שני קבצים. אוצר המילים ישמר לקובץ `filename.voc` ואילו הספירות ישמרו בקובץ `filename.counts` (העזרו בקבועים המוגדרים במחלקה בשביל לקבל את שתי הסיומות).
- בקובץ `voc` ישמר אוצר המילים בפורמט הבא:
- השורה הראשונה תכיל את מספר המילים שנמצאות ב `mVocabulary`.
 - החל מהשורה השניה תופיע כל מילה, יחד עם האינדקס שלה, בסדר עולה של האינדקסים.

עבור הקובץ all_you_need.txt אשר מכיל את הטקסט הבא:

Love love love

All you need is love All you need is love love , Love is all you need

נקבל קובץ .voc שנראה כך:

5 words

0,love

1,all

2,you

3,need

4,is

בקובץ counts. הספירות של זוגות המילים ישמרו באופן הבא:

- בכל שורה נכתוב את מספר המופעים של כל זוג מילים, כאשר זוג המילים ייוצג באמצעות האינדקסים שלהם באוצר המילים (vocabulary). את מספר המופעים נכתוב באותה השורה ונפריד בינו לבין שני האינדקסים באמצעות נקודותיים.
- הצמד $\langle i1, i2 \rangle$ יכתב לפני הצמד $\langle i3, i4 \rangle$ אם $i1$ הוא נמוך יותר מ $i3$, או אם $i1$ ו $i3$ זהים, ו- $i2$ נמוך יותר מ $i4$.
- מכיוון שרוב צמדי המילים האפשריים לא ניראו בטקסט (מספר המופעים הוא 0), נשמור רק את הצמדים שכן ניראו (כלומר, ערכים שאינם 0).
- קובץ ה count. עבור all_you_need.txt יראה כך.

0,0:3

0,1:1

0,4:1

1,2:3

2,3:3

3,4:2

4,0:2

4,1:1

(הסבר לפורמט: השורה הראשונה מייצגת את הצמד love, love שנראה 3 פעמים בקובץ)

הטסטר מייצר את שני הקבצים האלה תחת השמות all_you_need.voc ו

all_you_need.counts. העזרו בו להבין כיצד נבנים שמות הקבצים מהפרמטר filename. ניתן

להניח כי שם הקובץ בפרמטר filename אינו מכיל סימנים.

הערה: בכל הנוגע לרישום רווחים אם יש איזושהי התאמה בין הפלט המוצג לטסטר שקיבלתם, או באופן

כללי ספק לגבי רישום רווחים, הכלל הוא שכל מה שעובר את הטסטר כפי שסופק הוא בסדר (אבל גם

לגבי כל מה שנובע מאי התאמה עם קובץ ההנחיות, אם יש כזאת, נקבל גם גירסה לפי קובץ ההנחיות –

לא לדאוג).

חתימת המתודה:

```
public void saveModel(String fileName)
```

(4) **[10 נק']** השלימו את המימוש של המתודה `loadModel` המקבלת נתיב של קובץ (בלי הסיומת) `(fileName)` וטוענת משני הקבצים `fileName.voc` ו `fileName.counts` את מודל השפה לתוך השדות `mVocabulary` ו `mBigramCounts`. במידה ושדות אלה מאותחלים לערכים אחרים, ערכים אלה נדרסים (כלומר, אם המשתנים האלה כבר מצביעים למערכים כלשהם, אז אנחנו מפנים אותם למערכים חדשים – דורסים את המערך עצמו - ולא דורסים תאים של מערך קיים). הניחו כי שני קבצי המודל `fileName.voc` ו `fileName.counts` קיימים. ניתן להניח כי שם הקובץ בפרמטר `filename` אינו מכיל סיומת. כמו כן, ניתן להניח שתוכן הקבצים הוא בפורמט הנכון (שתואם לאופן שבו אנחנו יוצרים את הקבצים האלה), אם כי התוכן עצמו הוא לא בהכרח משהו שיצרנו בעצמינו במתודה אחרת בתוכנית. ניתן להניח שהקלט עומד במגבלות הגודל שנובעות מהשאלות הקודמות.

חתימת המתודה:

```
public void loadModel(String fileName) throws IOException
```

(5) **[5 נק']** ממשו את המתודה `getWordIndex` המקבלת מחרוזת ומחזירה את האינדקס שלה ב `mVocabulary`. במידה ומחרוזת זו לא מופיעה, הפונק' תחזיר -1 (העזרו בקבוע שהוגדר בראש המחלקה). בשאלה זו (ובשאלות הבאות) ניתן להניח שכבר אין צורך לעשות המרות, כלומר מספרים או מילים עם אות באנגלית שאינה ב-`lowercase` כבר אינן חוקיות, ואין צורך לבדוק את ההמרה שלהן.

חתימת המתודה:

```
public int getWordIndex(String word)
```

(6) **[5 נקודות]** ממשו את המתודה `getBigramCount` המקבלת שתי מחרוזות `word1` ו `word2` ומחזירה את מספר הפעמים ש `word2` הופיעה אחרי `word1`. אם אחת מהמילים לא קיימת באוצר המילים (`mVocabulary`) הפונק' תחזיר 0.

חתימת המתודה:

```
public int getBigramCount(String word1, String word2)
```

(7) **[10 נקודות]** ממשו את המתודה `getMostFrequentProceeding` המקבלת מילה `word` ומחזירה את המילה שהופיע אחריה הכי הרבה פעמים. במידה ויש כמה מילים עם אותו מספר מופעים, תוחזר המילה בעלת האינדקס הנמוך יותר ב `vocabulary`. במידה ואחרי המילה `word` לא ניראתה אף מילה אחרת (חשבו באיזה מצב זה יכול לקרות) המתודה תחזיר `null`. שימו לב, כי מניחים תקינות, כלומר שהמילה ניתנת ב-`lowercase` ושהיא קיימת ב-`mVocabulary`, ואין צורך לטפל במקרה לא תקין.

חתימת המתודה:

```
public String getMostFrequentProceeding(String word)
```

(8) **[5 נקודות]** ממשו את המתודה `isLegalSentence` המקבלת משפט `sentence` ובודקת אם מודל השפה שבנינו מאפשר קיומו של משפט זה. מכיוון שמודל השפה מתיימר ללמוד את מבנה השפה, ואת הקשרים בין המילים, המשפט יהיה חוקי אם כל זוג מילים צמודות שמופיעות במשפט הופיע ביחד לפחות פעם אחת בטקסט עליו למדנו את מודל השפה (בענייני המרות, כמובן, מה שקובע זה המודל שבנינו, ולא הגירסה ה"לא מומרת" של מילים בטקסט המקורי – במילים פשוטות, מניחים שהקלט ניתן לנו "מומר", כך שכבר אין אותיות גדולות, מספרים וכו'). בין היתר נובע שבמקרה שבו יש במשפט מילה שלא מופיעה במילון (או באופן כללי כל מילה לא חוקית), יש להחזיר מיד `false`.

דוגמאות:

- המשפט `love all you` הוא משפט "חוקי" במודל שנבנה על סמך הקובץ המתואר בסעיף 3. הצירוף `love all` נראה בקובץ עליו התאמנו, וכן הצירוף `all you`. שימו לב שלא ראינו את כל המשפט בשלמותו בטקסט האימון, אבל זה לא משנה כי אנחנו מסתכלים רק על זוגות.
- המשפט `love is is` אינו חוקי. הצירוף `is is` לא נראה בשום מקום בטקסט האימון.
- המשפט `love the beatles` גם כן אינו חוקי. המילים `the,beatles` כלל לא מופיעות באוצר המילים שלנו, ולכן ברור שלא ראינו את הצירוף `love the` וכן את הצירוף `the beatles` (שימו לב, קלט זה הוא קלט חוקי לפונקציה).
- אם המשפט ריק או מכיל רק מילה אחת שנמצאת במילון יש להחזיר `true`.
- אם המשפט מכיל רק מילה אחת שאינה נמצאת במילון יש להחזיר `false`.

חתימת המתודה:

```
public boolean isLegalSentence(String sentence)
```

(9) [10 נקודות] נרצה לבדוק אם באמצעות מבנה הנתונים שלנו ניתן ללמוד משהו על התנהגות השפה והיחסים בין מילים. לצורך בדיקה זו, נשתמש בקובץ טקסט גדול, ונסה לראות אם יש קשר בין מילים שמופיעות בקובץ הטקסט בהקשר דומה: כלומר, בסמיכות לאותן המילים. לצורך כך, ממשו את המתודה הסטטית `calcCosineSim` המקבלת שני מערכים של מספרים שלמים שאורכם זהה, ומחשבת את דמיון הקוסינוסים ביניהם על פי הנוסחה הבאה:

$$\frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

כאשר A ו B הם וקטורים באורך n הממומשים ע"י מערכים של `int`-ים. A_i הוא האיבר ה i במערך A , ו B_i הוא האיבר ה i במערך B . ככל שהוקטורים הם דומים יותר, הערך שנקבל יהיה גדול יותר. לדוגמא: עבור שני הוקטורים $[1,2,3]$, $[1,0,5]$ נבצע את החישוב הבא:

$$\frac{1 * 1 + 0 * 2 + 5 * 3}{\sqrt{1^2 + 0^2 + 5^2} * \sqrt{1^2 + 2^2 + 3^2}}$$

אם אחד הוקטורים מכיל רק אפסים, הפונקציה תחזיר 0. בגלל שאנחנו מניחים קלט תקין, אין צורך לטפל במקרה שבו אחד המערכים או שניהם ריקים. כמו כן, אפשר להניח שהמספרים מספיק קטנים כך ששום חישוב לא יבצע חריגה בקיבולת של `double`.

חתימת המתודה הסטטית:

```
public static double calcCosineSim(int[] arr1, int[] arr2)
```

(10) [15 נקודות] כעת, נשתמש בפונקציה שמימשנו בסעיף הקודם בשביל למצוא את המילה הכי דומה למילה word כלשהי על פי ההקשר שלהן בטקסט.
נגדיר את המושג "וקטור מייצג עבור מילה word כלשהי".

- עבור מילה word הקיימת באוצר המילים שלנו, הוקטור המייצג הוא וקטור הנגזר מתוך מודל השפה ואורכו הוא כמספר המילים באוצר המילים (mVocabulary).
- האיבר במקום ה x יכיל את מספר הפעמים שבהם הצירוף word, mVocabulary[x] נראה במודל השפה (כלומר, כמה פעמים הצמד word, mVocabulary[x] נספר במודל).
- לדוגמא: עבור מודל השפה שמתואר בסעיף (2), המילה love תיוצג באמצעות הוקטור הבא: [3, 1, 0, 0, 1]:
 - סידור המילים באוצר המילים הוא: love, all, you, need, is, ולכן האיבר הראשון בוקטור ייצג את המופע של love אחרי love, האיבר השני ייצג את המופעים של all אחרי love, וכן הלאה.
 - הצירוף love love מופיע שלוש פעמים על פי מודל השפה.
 - הצירופים love all מופיע פעם אחת.
 - הצירופים love you | love need | love אינם מופיעים כלל.
 - הצירוף love is מופיע פעם אחת.
- את הוקטורים המייצגים עבור כל מילה נייצג ע"י מערכים חד מימדיים של מספרים שלמים שאורכם הוא כגודל אוצר המילים.
- אם אוצר המילים הוא בגודל 1: כאשר אין מה להחזיר (כלומר יש תיקו בין כל המילים שלכולן יש אפס דמיון) מחזירים את המילה עם האינדקס הכי נמוך, כלומר הראשונה. במקרה הזה יש רק מילה אחת, אז נחזיר את המילה עצמה.

ממשו את הפונקציה getClosestWord אשר מקבלת מילה word ומחזירה את המילה הדומה לה ביותר על פי השוואת הוקטורים המייצגים של כל המילים לזה של word.
עליכם לחלץ את הוקטור המייצג את word מתוך השדה bigramCounts, ולאחר מכן, למצוא מילה w כך שה cosineSim בין הוקטור המייצג של w לבין זה של word הוא הגבוה ביותר מבין שאר המילים באוצר המילים. הציפיה שלנו היא שמילים בעלות וקטורים "דומים" יותר יהיו דומות במשמעות או באופן השימוש בהן בטקסט. במידה ויש שתי מילים להן דמיון זהה, יש להחזיר את זו שהאינדקס שלה קטן יותר.
נזכיר שאין צורך להתייחס למקרה שקיבלנו כקלט מילה שאינה באוצר המילים.

הערה: לצורך התרגיל הזה נחשיב דמיון של מילה לעצמה בתור 0 (אחרת, תמיד היינו צריכים להחזיר את אותה מילה שקיבלנו כקלט, היא המילה שהכי דומה לה היא היא עצמה).

חתימת המתודה:

```
public String getClosestWord(String word)
```

הערה לידע כללי:

על מנת לקבל תוצאות משמעותיות בשיטה המתוארת בסעיף 10 עלינו להשתמש בכמות מאוד גדולה של טקסט, עם אוצר מילים רחב ודוגמאות שימוש רבות לכל מילה. כמו כן, עלינו להשתמש במידע נוסף: לא להשוות רק את המילים שמופיעות אחרי המילה, אלא גם את המילים הקודמות לה, ולהשתמש ביותר ממילה עוקבת/קודמת אחת. יחד עם זאת, גם בשיטה המאוד בסיסית שמתוארת בסעיף 10 ניתן להגיע לתוצאות מעניינות עבור הקובץ emma.txt (הספר "אמה" של ג'יין אוסטין, מתוך האתר <https://www.gutenberg.org/>). כך לדוגמא, ניראה ש great ו good הם בעלי וקטורים דומים יחסית, וכן "she", "he" ועוד רבים אחרים. מה שיכול לעזור לנו לשפר את התוצאות הוא עיבוד לשוני על הטקסט: למשל, להסב את כל הטיות הפעלים ושמות העצם לצורת היחיד (goes -> go, books -> book). כאמור, המימוש שלכם הוא בסיסי ביותר, אבל מהווה בסיס לטכניקות מתקדמות יותר בהן משתמשים בשביל ללמוד מידע על שפה. בפרט, בסעיף 8 אנחנו משתמשים במדד מאוד מחמיר: ציפוי מילים שלא ניראו בטקסט האימון גורמים לכך שהמשפט "אינו חוקי". מדד זה נקבע רק לצורך התרגיל, וכמובן שאין לו משמעות בעולם האמיתי של עיבוד שפה טבעית. במציאות, אנחנו נותנים למשפטים ציונים: כמה סבירים הם לאור המודל שנלמד.

שימוש בחוזים:

בשלב התרגיל מופיעים חוזים עבור חלק מהמתודות. השתמשו בחוזים על מנת להבין אילו הנחות אפשר להניח על הפלט, ובאילו מקרי קצה צריך לטפל.

קריאת מקבצים:

אילו היינו עובדים עם Scanner בשביל לקרוא מהקבצים, עבור קבצים גדולים, בחלק ממערכות ההפעלה, ה Scanner יכול להיכשל (זה יכול להראות פשוט כאילו הוא לא קרא כלום). זו בעיה מוכרת וידועה ב Java ולכן ההמלצה היא לעבוד עם **BufferedReader** עבור קבצים גדולים, וזאת גם ההנחיה הרשמית בתרגיל זה.

טסטר

לתרגיל זה מצורפת מחלקת טסטר. על מנת להריץ את הטסטר עליכם לוודא שהנתיבים לקבצים מהם קוראים/כותבים תואמים למיקומם על המחשב שלכם, ולעדכן את כתובות הקבצים במידת הצורך. מומלץ להוסיף בדיקות משלכם תוך שימוש בקבצי טקסט חדשים. ברוב המקרים כדאי להשתמש בקבצי קלט קצרים עליהם קל לחשב את התשובה אותה תרצו לקבל מהפונקציות שלכם.

בהצלחה!