פרויקט בניית מנוע לאחזור מסמכים חלק ב'

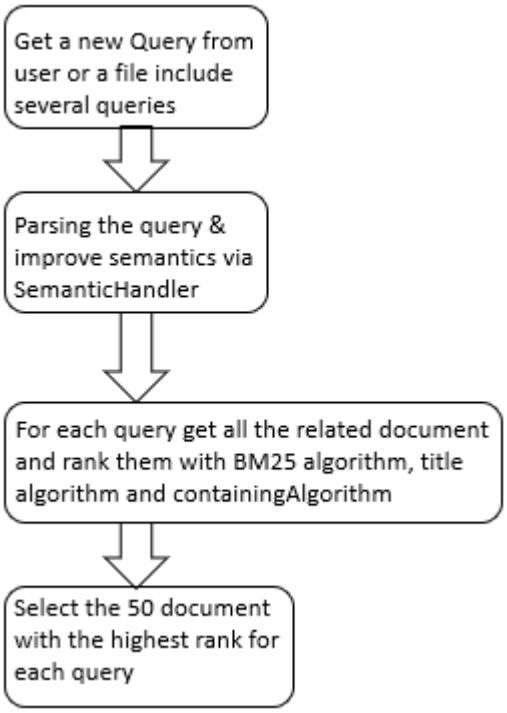
אביחי צרפתי 204520803

ירדן לוי 204341580

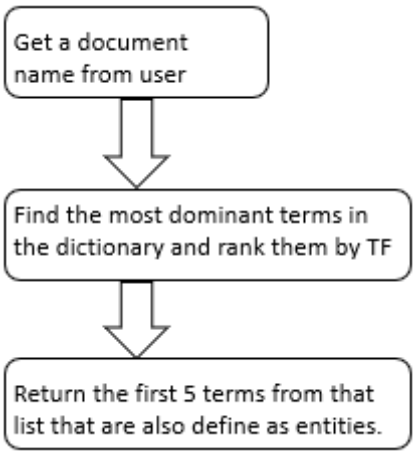
הסבר כללי:

בחלק ב' של פרויקט המנוע אנו מתמקדים באחזור המסמכים, לאחר שבחלק א' התמקדנו באינדוקס המסמכים. שני התהליכים העיקריים שאותם פיתחנו בחלק זה הם:

אחזור מסמכים ודירוגם בהתאם לשאילתה ספציפית



זיהוי 5 ישויות דומיננטיות בכל אחד מהמסמכים שאוחזרו והצגתם למשתמש



חלק 1- הסבר מפורט על מחלקות הפרויקט:

חלק 1.1- הסבר על שינויים במחלקות מחלק א' בפרויקט:

מחלקת InvertedIndex:

```
public String getPostingLink(String word)
```

השיטה תחזיר עבור word את מספר השורה המתאים בקובץ ה-Posting, אם קיים.

מחלקת DocumentIndex:

נתון ייחודי לחלק ב' שנדרש מאיתנו הוא שמירת 5 היישויות הדומיננטיות ביותר בכל מסמך, לצורך הצגת למשתמש, אם ירצה, לאחר אחזור המסמכים.

לכן הוספנו למחלקה זו את השדה mainEntities שהינו מסוג Pair אשר תפקידו להחזיק ישויות אלו.

בנוסף הוספנו את השיטה getFiveEntities שהיא getter אשר מחזיר את מבנה הנתונים הזה. כמו כן הוספנו את השיטה get5words שמחזירה את היישויות כמחרוזת אחת.

מחלקת Model:

```
public void startBoogleSearch(String postingPath, String queries, String
outLocation, boolean stem, boolean semantic, boolean offline)
```

זוהי שיטת מעטפת. הפרמטרים שהשיטה מקבלת:

1. נתיב לקובץ Posting.
2. נתיב לקובץ שאילתות או, במידה והמשתמש הכניס שאילתה חדשה אז את המחרוזת של השאילתה עצמה.
3. נתיב לקובץ אליו ייכתבו התוצאות.
4. משתנים בוליאניים אשר מגדירים האם יש להשתמש בסטמינג, במודל הסמנטי ו/או האם נדרש לעשות זאת באמצעות חיבור לאינטרנט או ללא.

ראשית ייבדק אם הנתיבים שניתנו לקבצי ה-Posting ולתיקייה אליה ייכתבו התוצאות הם תקינים. במידה וכן, תופעל השיטה boogleMainLogic המחזירה את המסמכים הרלוונטיים ביותר עבור כל שאילתה כפי שיפורט בהמשך. במידה ואחזור המסמכים הרלוונטיים לא יצליח- תוצג הודעה מתאימה.

```
synchronized HashMap<String, LinkedList<String>> boogleMainLogic(String
postingPath, String queryField, boolean stem, boolean semantic, boolean
offline)
```

הפרמטרים שהשיטה מקבלת (משיטת boogleMainLogic):

1. נתיב לקובץ Posting.
2. נתיב לקובץ שאילתות או, במידה והמשתמש הכניס שאילתה חדשה אז את המחרוזת של השאילתה עצמה.
3. משתנים בוליאניים אשר מגדירים האם יש להשתמש בסטמינג, במודל הסמנטי ו/או האם נדרש לעשות זאת באמצעות חיבור לאינטרנט או ללא.

השיטה תיצור רשימה מקושרת אשר תחזיק את כל האובייקטים מסוג Query- כלומר, במידה והמשתמש בחר לייבא את קובץ השאילתות אזי כל השאילתות יוכנסו לרשימה ובמידה והמשתמש בחר להכניס שאילתה מקורית אזי הרשימה תכיל רק איבר אחד.

בסוף השיטה יוחזר מבנה נתונים מסוג HashMap אשר כל כניסה בו תכיל מס' שאילתה מסוימת והערך עבור כל כניסה תהיה רשימת מספרי המסמכים הרלוונטיים ביותר עבור אותה שאילתה כאשר הם מדורגים לפי מידת הרלוונטיות שלהם.

```
private LinkedList<String> getLimited(LinkedList<String> queryResults)
```

שיטה זו תקבל רשימה המורכבת מכל המסמכים הרלוונטיים עבור שאילתה מסוימת ותחזיר את 50 המסמכים הרלוונטיים ביותר, או פחות.

```
private ObservableList<ResultDisplay>
resultsToObservableList(HashMap<String, LinkedList<String>> results)
```

השיטה תקבל את התוצאות שחזרו מ-boogleMainLogic ותהפוך אותן להיות observable- ותחזיר את הרשימה החדשה.

```
public String showFiveEntities(String docName)
```

השיטה תקבל שם של מסמך ותחזיר את מחרוזת היישויות הדומיננטיות ביותר במסמך.

```
public boolean writeResultsToDisk(String dest)
```

שיטה זו אחראית לכתוב את התוצאות עבור השאילתה בצורה ממויינת. השיטה תקבל נתיב ותכתוב אליו את התוצאות כקובץ טקסט results.txt.

מחלקת MiniDictionary:

```
public HashMap<String, Integer> countAppearances()
```

השיטה תחזיר מבנה נתונים מסוג HashMap הסופר את כמות ההופעות של terms במסמך.

```
public Pair<String, Integer>[] getPrimaryWords()
```

השיטה תחזיר את מבנה הנתונים המחזיק את 5 ה-terms הדומיננטיים והמידע עליהם.

```
public void setPrimaryWords()
```

השיטה אחראית לאגד את היישויות הדומיננטיות כסט ממויין.

בשיטה זו נבנה Map אשר אליו מוכנסים כל ה-terms מתוך ה-MiniDictionary בצורתם הממויינת ע"י השיחה sorted שתפורט בהמשך. בשלה הבא, נבדוק מיהן 5 היישויות הדומיננטיות ביותר, כלומר שהינן מוגדרות כישות וגם בעלות שכיחות גבוהה, ונכניס אותן לשדה places אשר נועד לשמור אותן.

```
private Map<String, LinkedList<Integer>> sorted(Map<String,
LinkedList<Integer>> toSort)
```

שיטה זו מקבלת מבנה נתונים מסוג Map, ממיינת אותו ומחזירה את אותו מבנה נתונים בצורתו הממויינת.

מחלקת Parse:

במחלקה זו לא נוספו שיטות נוספות. נעשה שינוי מזערי בשיטה המרכזית של המחלקה Parse (boolean isQuery) כך שנדע להתייחס לטקסט שבתהליך ה-Parsing בהתאם אם הוא חלק משאילתה או חלק מטקסט גיל של מסמך. בנוסף, המשתנה הבוליאני הזה יועבר גם לשיטה int lastCheckAndAdd כך שה-term הנוכחי יעבור stemming ויתווסף ל-MiniDictionary של השאילתה הנוכחית.

מחלקת ReadFile:

```
public static LinkedList<Query> readQueries(File queries)
```

השיטה אשר אחראית לבצע את קריאת השאילתות. תקבל כפרמטר את הקובץ בו נמצאות כל השאילתות. הקריאה תתבצע ע"י Jsoup בו גם השתמשנו בחלק א' בפרויקט ותפריד את הנתונים על כל שאילתה לפי תגיות.

ניצור עבור כל שאילתה אובייקט מסוג Query אשר יכיל את המספר המזהה של השאילתה, הכותרת שלה והתיאור שלה בהתאם למידע אשר נחלץ מקריאת הקובץ.

לבסוף תוחזר רשימה של Query המורכבת מכל השאילתות שנקראו מהקובץ.

```
public static LinkedList<String> readPostingLineAtIndex(String path, char c,
List<Integer> indexes, boolean stem)
```

שיטה תקבל נתיב לקובץ Posting ספציפי ותבצע קריאה לפי המיקומים שיינתנו.

חלק 1.2- הסבר על מחלקות חדשות שנוספו עבור חלק ב' בפרויקט:

מחלקת Ranker:

תפקיד המחלקה הינו לדרג את התשובות לשאילתות, כלומר לדרג את המסמכים הרלוונטיים, על פי נוסחת דירוג שאנחנו נפתח. מחלקה זו מסייעת למחלקת ה-Searcher במטרתה.

```
double BM25Algorithm(String word, String documentName, int tf, double idf)
```

אלגוריתם הדירוג BM25 הוא למעשה אלגוריתם אשר מדרג מסמכים מסוימים בהתבסס על ה-terms בשאילתה והופעותיהם בכל אחד מהמסמכים האלו, וללא קשר לסמיכות שלהן אחת לשנייה במסמכים. השיטה שלנו תקבל כפרמטרים מילה מהשאילתה (word-> q_i), שם מסמך (documentName->D), את ה-tf של המילה ואת ה-idf שלה.

הנוסחה עליה נתבסס בחישוב היא זו:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

כאשר את ה-IDF של כל term מתוך השאילתה נחשב ע"י:

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

על פי הנוסחה, k1 ו-b הם פרמטרים חופשיים אשר לאחר מספר בדיקות והרצות שונות בחרנו לקבוע אותם על פי ההמלצות באינטרנט כאשר k1= 1.2 ואילו b=0.75.

לבסוף השיטה תחזיר את התוצאה עבור term אחד מתוך השאילתה בהתייחס למסמך מסויים.

```
void titleAlgorithm(HashMap<String, Double> score, String docName,
Set<String> wordsSet)
```

זהו אלגוריתם הנועד להביא לידי ביטוי את ה-terms המופיעים בכותרת של מסמך מסויים. החלטנו לפתח אלגוריתם זה מתוך מחשבה כי terms אשר מופיעים בכותרת הינם דומיננטיים יותר בעת חיפו מסמך מסויים ועל כן יש למשקל אותם בהתאם.

השיטה תקבל שם של מסמך וסט של כל המילים המופיעות בשאילתה. השיטה תחשב את משקל הכותרת של המסמך בהתייחס לשאילתה ותעדכן את התוצאה.

```
void containingAlgorithm(HashMap<String, Double> score, Set<String>
wordsInQuery)
```

השיטה מקבלת את מפת המסמכים score וסט של כל המילים בשאילתה.

שיטה זו אחראית לעדכן את הדירוג של כל מסמך בהתבסס על הקשר בין השאילתה ל-5 היישויות הדומיננטיות במסמך.

```
private double getDocumentAverageLength()
```

שיטה זו מחזירה ממוצע של מסמך.

מחלקת Searcher:

תפקיד המחלקה הינו לבצע את השאילתות. כלומר, בהינתן שאילתה, באמצעות טקסט חופשי מהמשתמש או מתוך קובץ שאילתות, מחלקה זו תדע להחזיר את 50 המסמכים הרלוונטיים ביותר עבור השאילתה. בנוסף, יתאפשר למשתמש לראות מהן 5 היישויות הדומיננטיות ביותר בכל מסמך מאלו שאוחזרו.

```
public LinkedList<String> call()
```

שיטה מחייבת לצורך מימוש הממשק Callable ופעולת התוכנית ע"י תהליכים. היא מפעילה את השיטה getQueryResults.

```
private LinkedList<String> getQueryResults()
```

זו השיטה העיקרית במחלקה- תפקידה לבצע את כל תהליך אחזור המסמכים, דירוגם על פי האלגוריתמים לדירוג שפיתחנו, והחזרת רשימה המסמכים ממויינים על פי אותו דירוג.

ראשית, נשים במחרוזת את הטקסט עצמו של השאילתה, וברשימה מקושרת נפריד את כל מילות השאילתה למחרוזות. במידה והמשתמש בחר להשתמש במודל הסמנטי אזי נעדכן את הטקסט של השאילתה לכלול גם 2 מילים הקרובות ביותר מבחינה סמנטית עבור כל אחת ממילות השאילתה.

בשלב הבא נעביר את השאילתא בתהליך ה-Parse וניצור לה MiniDictionary. ונייצר HashMap אשר תשמור עבור כל אחת ממילות השאילתא את כמות המופעים שלה בה. נתון

זה ישמש אותו באלגוריתמים של הדירוג. כמו כן, נבדוק האם ה-terms מהשאילתה כבר מופיעים בקבצי ה-Posting וניצור מבנה נתונים אשר יכיל את מילות השאילתה ומיקומם.

מכאן, עבור כל term מהשאילתה נעבור על ה-Posting שלו עם המסמכים הרלוונטיים, נבצע דירוג ונבצע משקול ושקלול עבור כל מסמך בהתייחס לשאילתה. השיטה תחזיר את רשימת התוצאות של המסמכים הרלוונטיים ביותר עבור כל שאילתה- מדורגים.

```
private CaseInsensitiveMap getPosting(Set<String> query)
```

הפונקציה מקבלת סט מחרוזות אשר כל מחרוזת מייצגת term בשאילתה.

עבור כל term מהסט שהתקבל נבדוק האם הינו מילה או שאינו מתחיל ב-ABC ובהתאם ניגש לקובץ ה-Posting המתאים לשורה בה ה-term נמצא.

במידה וה-term לא נמצא במילון שלנו אז נוסיף אותו למבנה שנתונים שנחזיר בסופו של דבר ללא ציון מיקום.

במידה וכן מצאנו אותו במילון אזי נעדכן אותו במבנה הנתונים שנחזיר בסופו של דבר, עם המיקומים הרלוונטיים בהם הוא מופיע במילון.

לבסוף נחזיר מבנה נתונים מסוג CaseInsensitiveMap אשר מכיל את ה-terms ותוכן ה-Posting התואם הרלוונטי לשאילתה.

```
private void addToScore(HashMap<String, Double> score, String docName,
double newScore)
```

השיטה מעדכנת דירוג של מסמך מסויים במבנה הנתונים score.

<u>תת מחלקה CaseInsensitiveMap:</u>

תת מחלקה היורשת מ-HashMap אשר עוטפת את שיטות ה-put ו-get ע"י השמת ה-key באותיות קטנות בלבד, כך גם לגבי ייבוא ה-key.

<u>מחלקת ResultDisplay:</u>

```
private LinkedList<QueryDisplay> toQueryResultList(LinkedList<String>
docNames)
```

השיטה מקבלת רשימה של כל המסמכים שאוחזרו כשתוצאות לשאילתה ספציפית ומחזירה אותם כשרימה של אובייקטים מסוג QueryDisplay.

```
public StringProperty sp_queryIDProperty()
```

השיטה תחזיר StringProperty של מספר השאילתה.

```
public LinkedList<QueryDisplay> getDocNames()
```

getter עבור שדה המחלקה docNames.

תת מחלקה QueryDisplay:

תת מחלקה לצורך יצירת מעטפת להחזרת המחרוזת של שמות המסמכים הרלוונטיים לשאילתה בתצורת StringProperty.

מחלקת Query:

המחלקה מייצגת אובייקט של שאילתה. שדות המחלקה יכילו את המידע אשר ישמש אותנו עבור ניתוח השאילתה כנדרש.

שדות מחלקה זו הינם מספר מזהה של השאילתה, השאילתה עצמה כפי שתופיע בתגית Text, ותיאור השאילתה.

מחלקת SemanticHandler:

מחלקה זו אחראית על כל הטיפול הסמנטי בתהליך אחזור המסמכים. שדות המחלקה הינם אובייקט מסוג WordVectors ומשתנה בוליאני אשר יגדיר האם המשתמש ביקש לבצע את האחזור offline או online.

טיפול סמנטי נועד לשפר את יכולת אחזור המסמכים של מנוע החיפוש. באמצעות שיפור זה, כאשר תוכנס שאילתה ניתן יהיה לאחזר מסמכים רלוונטיים גם אם כוללים מילים דומות ביותר אך לא זהות לאלו הנמצאות בשאילתה.

בהינתן term אשר יש לבדוק עבורו את הוקטור שלו- מחלקה זו תדע להחזיר 2 מילים הקרובות ביותר במשמעות ל-term הנבדק.

```
public String getTwoBestMatches(List<String> originalQueryWords)
```

השיטה היחידה במחלקה- מקבלת רשימה המכילה את כל ה-terms בשאילתה מסוימת כאשר עבור כל אחד מה-terms נחזיר את 2 המילים הדומות ביותר (מילים נרדפות או קרובות במשמעות).

חלק 2- הסבר על כלל האלגוריתמים הפועלים בפרויקט:

אלגוריתם לדירוג מסמכים:

בפרויקט זה ישנה חובה לעבוד על פי האלגוריתם BM25 שהינו אלגוריתם אשר מדרג מסמך ספציפי בהתייחס לכל אחד ואחד מה-terms בשאילתה המוצגת. אלגוריתם זה בא לידי ביטוי בתוך מחלקת Ranker כפי שפורט.

ביצענו מספר הרצות לבדיקת התוצאות שאוחזרו בעת דירוג המסמכים באמצעות השימוש באלגוריתם זה בלבד. ובהתייחס לפרמטרים הפנימיים שנקבעו. לאחר ביצוע בדיקות אלו הגענו למסקנה כי נדרש לשפר את אלגוריתם הדירוג באמצעות הוספת פרמטרים נוספים.

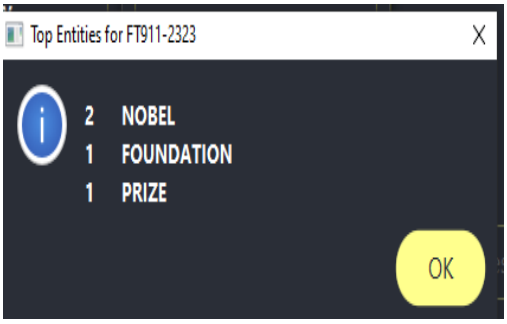
לשם כך הוספנו את ה-titleAlgorithm אשר מוסיף דירוג גבוה יותר במידה והמילה בשאילתה מופיעה בכותרת של המסמך הנבדק, כמו כן נתנו דירוג גם עבור הימצאות המילים באחת מ-5 היישויות הדומיננטיות במסמך הנבדק באמצעות containingAlgorithm.

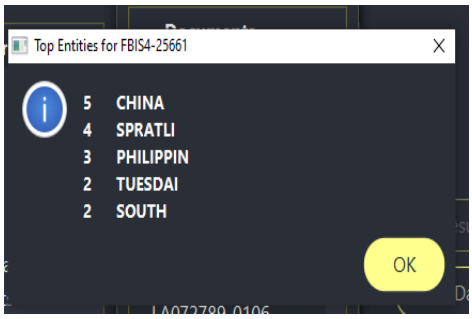
את המשקל הגבוה ביותר נתנו לאלגוריתם הכותרת- 45%, לאלגוריתם BM25 נתנו משקל של 25% ולבסוף לאלגוריתם היישויות הדומיננטיות נתנו משקל של 30%.

משקלים אלו נקבעו מתוך בדיקות והרצות רבות עד להגעה לתוצאות האופטימליות ביותר שהצלחנו להגיע אליהן.

אלגוריתם למציאת 5 יישויות דומיננטיות:

כדי למצוא את היישויות הדומיננטיות ביותר בכל מסמך התהליך שביצענו הוא כזה: ראשית, דירגנו את כל ה-terms שנמצאו במסמך על פי ה-tf שלהם. בשלב השני, בדקנו האם ה-term הינו יישות על פי הגדרה, כלומר האם הוא מופיע בשני מסמכים לפחות. במידה וכן, הוא מוגדר כיישות. כך הכנסנו לתוך מבנה נתונים בתוך MiniDictionary את 5 היישויות הדומיננטיות ביותר. במידה ובמסמך יש פחות מ5 יישויות אזי הוא יציג רק את אלו הקיימות, ובמקרה בו במסמך מסויים אין יישויות כלל תוצג הודעה מתאימה.





אלגוריתם לשיפור סמנטי:

עבור נושא הסמנטיקה החלטנו לפעול בשני מישורים. המישור הראשון הינו ברירת המחדל, כאשר המנוע יעבוד ללא חיבור לאינטרנט (offline). במצב זה, הורדנו מודל מאומן אשר יטפל בנושא השיפור הסמנטי. האלגוריתם בו השתמשנו במצב offline הינו Glove מתוך האתר של stanfordNLP. פירוט נוסף אודות האלגוריתם בחלק 3 של הדוח- הסברים על קוד פתוח.

המישור השני בו פעלנו הינו במצב בו המנוע כן יעבוד עם חיבור לאינטרנט (online). שמנו לב, לאחר בדיקות רבות והרצות שונות כי בעת נתינת אפשרות לחיבור לאינטרנט תוצאות אחזור המסמכים היו טובות בהרבה מאשר ללא חיבור לאינטרנט. בשל כך, הסקנו כי יש לאפשר למשתמש את הבחירה כיצד ירצה לבצע את תהליך אחזור המסמכים. פירוט נוסף על האלגוריתם בו השתמשנו נמצא בחלק 3 של הדוח- הסברים על קוד פתוח.

לדוגמה: עבור המילה isra המודל ימצא גם את המילים Israel, Israeli.

נתונים התומכים באלגוריתמים שמימשנו:

ישנם מספר נתונים אשר שמרנו בעת יצירת המילון וקבצי ה-Posting אשר תומכים בעבודת האלגוריתמים השונים. ראשית, עבור כל MiniDictionary (מילון של מסמך ספציפי) שמרנו את כל ה-terms המופיעים בו ולכל אחד את ה-tf שלו ואת המיקומים שלו בטקסט המסמך. בנוסף, עבור כל מסמך: נשמר אורך המסמך, ה-term השכיח ביותר ושכיחותו, כותרת המסמך, ו-5 היישויות הדומיננטיות ביותר.

בקבצי ה-Posting שמרנו עבור כל term את כל המסמכים הם הוא מופיע, כמות ההופעות שלו בכל מסמך, מיקומים בכל מסמך, וסך הכל כמות ההופעות הכללית שלו במאגר המסמכים.

בקובץ האינדקס ההופכי שמרנו בנוסף גם את כמות המסמכים השונים בהם כל term מופיע.

כל הנתונים האלו אפשרו לנו בסופו של דבר שליפה מהירה של המידע לצורך ביצוע האלגוריתמים.

חלק 3- הסבר על קוד פתוח אשר נעשה בו שימוש במהלך הפרויקט:

השתמשנו בקוד פתוח בתוך המחלקה SemanticHandler לצורך טיפול סמנטי בשאילתות שיוצגו למנוע החיפוש.

תת מחלקה DatamuseAPI:

מחלקה זו תשמש אותנו כאשר המשתמש יבקש לבצע את אחזור המסמכים עם חיבור לאינטרנט, כלומר online. במחלקה זו אנו משתמשים בקוד פתוח.

ה-API בו אנו משתמשים הוא http://api.datamuse.com/words?rel_syn=**term** כאשר term הינה מילה מתוך השאילתה.

```
public JSONObject post(String url) throws IOException
```

שיטה זו תופעל מתוך השיטה synonyms. השיטה תקבל את כתובת אתר האינטרנט אליו יש לגשת. תישלח הודעת request לקבלת המילים הנרדפות ל-term הנוכחי.

השיטה תחזיר אובייקט מסוג JSONObject המכיל את כל המידע הנדרש.

```
public String synonyms(String wordToSyn) throws IOException
```

השיטה תקבל את ה-term מתוך שאילתה עבורו יש לחפש מילים נרדפות ובאמצעות השיטה post תיצור בקשה מאתר האינטרנט להחזיר את המילים הנרדפות המתאימות ביותר עבור המילה שביקשנו. גם במקרה זה נבקש להחזיר את 2 המילים הנרדפות הקרובות ביותר במשמעות, והן יחזרו בתוך מחרוזת.

תת מחלקה GloVe:

מחלקה זו תשמש אותנו כאשר המשתמש יבקש לבצע את אחזור המסמכים ללא חיבור לאינטרנט, כלומר offline.

המודל בו אנו משתמשים הינו מתוך: https://nlp.stanford.edu/projects/glove/

```
public String synonyms (String wordToSyn)
```

שיטה זו תקבל term מתוך השאילתה ובאמצעות השיטה wordsNearest של האובייקט WordVectors תמצא את המילים הדומות ביותר. מכיוון שאנחנו מבצעים את הפעולה הזו עבר כל אחת מהמילים בשאילתה החלטנו כי נחזיר רק 2 מילים דומות על מנת לקבל תוצאות קרובות במשמעות ככל שניתן.

בסופו של דבר, השיטה תחזיר מחרוזת של כל המילים החדשות שנוספו.

חלק 4- ניתוח פלט TREC EVAL:

Stemming

Query Number	Query	Precision	Recall	precision@5	precision@15	precision@30	precision@50	MAP
351	Falkland petroleum exploration	0.4	0.45833	0	0.5333	0.5333	0.4	0.1876
352	British Chunnel impact	0.44	0.08943	0.6	0.5333	0.5	0.44	0.0477
358	blood-alcohol fatalities	0.42	0.41176	0.2	0.4667	0.4	0.42	0.1761
359	mutual fund predictors	0.12	0.21429	0	0.2	0.1667	0.12	0.0349
362	human smuggling	0.14	0.17949	0	0.1333	0.1	0.14	0.0239
367	piracy	0.3	0.08108	0.2	0.1333	0.1667	0.3	0.0224
373	encryption equipment export	0.14	0.4375	0.4	0.4667	0.2333	0.14	0.2435
374	Nobel prize winners	0.32	0.07843	0.6	0.3333	0.2667	0.32	0.0315
377	cigar smoking	0.24	0.33333	0	0.1333	0.2	0.24	0.0699
380	obesity medical treatment	0.1	0.71429	0	0.2667	0.1333	0.1	0.1282
384	space station moon	0.2	0.19608	0.6	0.2667	0.2	0.2	0.0595
385	hybrid fuel cars	0.42	0.24706	0	0.2	0.3	0.42	0.0802
387	radioactive waste	0.22	0.15068	0.2	0.1333	0.1333	0.22	0.0277
388	organic soil enhancement	0.34	0.34	0.6	0.4667	0.4667	0.34	0.1728
390	orphan drugs	0.3	0.12295	0.4	0.2667	0.4	0.3	0.0539
	total relavent from retrival	**205/750**						**0.0924**

Total time for 15 queries (stemming): 1.304 m

No Stemming

Query Number	Query	Precision	Recall	precision@5	precision@15	precision@30	precision@50	MAP
351	Falkland petroleum exploration	0.36	0.375	0	0.4	0.5	0.36	0.1481
352	British Chunnel impact	0.46	0.0935	0.6	0.6	0.5333	0.46	0.0578
358	blood-alcohol fatalities	0.42	0.41176	0.2	0.466	0.4333	0.42	0.182
359	mutual fund predictors	0.16	0.28571	0	0.2	0.233	0.16	0.0557
362	human smuggling	0.16	0.20513	0.4	0.2667	0.2	0.16	0.088
367	piracy	0.26	0.07027	0.4	0.1333	0.1333	0.26	0.0166
373	encryption equipment export	0.14	0.4375	0.2	0.3333	0.2333	0.14	0.1575
374	Nobel prize winners	0.5	0.12255	1	0.5333	0.5333	0.5	0.0814
377	cigar smoking	0.14	0.19444	0	0.0667	0.0667	0.14	0.0226
380	obesity medical treatment	0.08	0.14	0	0.2	0.1	0.08	0.1064
384	space station moon	0.2	0.19608	0.6	0.2667	0.2	0.2	0.0709
385	hybrid fuel cars	0.38	0.22353	0	0.0667	0.233	0.38	0.0619
387	radioactive waste	0.2	0.13699	0.2	0.0667	0.1667	0.2	0.0242
388	organic soil enhancement	0.24	0.24	0.4	0.333	0.3	0.24	0.0881
390	orphan drugs	0.26	0.10656	0.4	0.333	0.3667	0.26	0.0419
	total relavent from retrival	**198/750**						0.0817

Total time for 15 queries (no stemming): 1.76 m

חלק 5- סיכום:

בעיות בהן נתקלנו:

1. בעיה ראשונה שנתקלנו בה היא נושא החיבור לאינטרנט וגודל מאגר המסמכים אשר עמד לרשותנו. במהלך העבודה שמנו לב כי איכות אחזור התוצאות הושפעה מאוד מנושא זה וניתן היה לראות כי מדד ההצלחה עלה משמעותית בעת חיבור לאינטרנט. דבר נוסף שהשקענו בו זמן הוא נושא מאגר המסמכים. עמד לרשותנו מאגר מסמכים גדול מאוד עם מאות אלפי מסמכים, דבר אשר יצר אינדוקס של מעל מיליון terms ייחודים עבור כל המאגר ובהתאם לכך שמירת כלל הנתונים במידה שתאפשר אחזור איכותי ושליפה מהירה של המידע היוותה אתגר משמעותי ונאלצנו לחפש פתרונות יצירתיים. בנוסף, על מנת שנוכל למצוא את האיזון שבין אחזור מסמכים רלוונטיים ביותר לבין אחזורם בזמן סביר למשתמש נאלצנו מצד אחד לוותר על איכות התוצאות (בשל אחזור מספר מוגבל של מסמכים וקביעת פרמטרים מסוימים) ומצד שני להבין כי נצטרך לשאת בזמנים ארוכים מעט על מנת לאפשר בכל זאת תוצאות איכותיות.
2. בעיה נוספת שעלתה במהלך העבודה על חלק ב' בפרויקט היא למעשה נושא שמירת היישויות. דרישה זו הינה הכרחית לצורך אחזור איכותי וכחלק מדרישות הפרויקט בחלק ב'. פעולת שמירת היישויות הדומיננטיות היא פעולה שלא חשבנו עליה מראש בחלק א', ולכן- נאלצנו במהלך העבודה בחלק ב' לבצע שינויים בחלק א'.

האתגרים הגדולים:

1. האתגר העיקרי בחלק ב' הינו למקסם את איות תוצאות אחזור המסמכים תוך זמן מינימלי כך שנוצרת חוויית משתמש איכותית.
2. אתגר אשר רלוונטי בשני חלקי הפרויקט הוא ביצוע תהליך ה-parser בצורה מדויקת. הכוונה היא שגם בתהליך אינדוקס המסמכים נבצע תהליך זה כך שנחזיר כמות terms הגיונית ונכונה ומצד שני בחלק ב' נאפשר את ביצוע התהליך עבור השאילתות כך שנאפשר אחזור מסמכים בצורה מיטבית.
3. אתגר משני אך משמעותי בחלק ב' של הפרויקט הינו אלגוריתם הדירוג אשר משפיע משמעותית על תוצאות אחזור המסמכים- בחלק זה ביצענו ניסיונות רבבים וחשיבה מעמיקה על מנת למצוא את הפרמטרים אותם נרצה לקחת בחשבון בתהליך וכמו כן, גם את המשקלים אשר נדרש לתת לכל אחד מתחומי הדירוג.

המלצות לשיפור/מה היינו עושים אחרת:

ההמלצה העיקרית שלנו לשיפור היא חשיבה מעמיקה ומקדימה מתחילת העבודה על הפרויקט. כלומר, טרם תחילת העבודה על חלק א' אנו ממליצים לתכנן מראש את שני חלקי הפרויקט. בדרך זו ניתן לצפות אתגרים עתידיים בחלק ב' כך שניתן יהיה למזער אותם בעזרת עבודה מקדימה והכנה טובה כבר בחלק א' של העבודה.

תכנון נכון יכול לבוא לידי ביטוי בנושא ביצוע אלגוריתם לשמירת יישויות דומיננטיות במסמכים כבר בחלק א'. כמו כן, תכנון איכותי של אלגוריתמי הדירוג- תשומת לב יתרה לפרטים אשר יכולים לסייע בדירוג מוצלח ושמירת המידע הרלוונטי כבר בחלק א' בשלב האינדוקס.