**מנוע אחזור – דו"ח חלק ב'**

**הסבר מפורט על אופן פעולת המנוע:**

ראשית נציין אילו שינויים ביצענו במחלקות מחלק א' של הפרויקט:

* כדי לא להכביד על הזיכרון ולעזור בשיפור זמן הריצה החלטנו במחלקה Parse לשנות את שלב ביצוע הורדת פרטי המסמכים לדיסק. במקום שהמסמכים ישארו עד סוף האינדוקס במבנה נתונים בParse ורק אז יכתבו לדיסק נבצע כתיבה לדיסק אחרי כל איטרציה (כלומר, אחרי קריאה ופירסור של 1000 מסמכים).
* בגלל שבחלק ב' של הפרויקט הודרכנו לצרף את קבצי האינדקס שלנו על מנת לחסוך זמן בבדיקות הפרונטליות. נאלצנו לשמור יחד עם קבצי הPosting והמילון גם את קובץ ה Stop Words, כדי שברגע שנטען מהנתיב את קבצי הPosting נוכל גם לטעון את קובץ הStop Words ולהיעזר במילים אלו בשלב של ביצוע Parse על שאילתה. בחלק א' כאשר טוענים את קובץ המילים האלו נדאג גם להעתיק את אותו הקובץ אל הנתיב שבוא נשמרים קבצי הindex שלנו.
* במחלקה Parse אחרי שסיימנו לעבור על כל המילים במסמך מסוים כתבנו לדיסק את כל הישויות המוכלות במסמך, על מנת שנוכל בחלק ב' להוציא עבור מסמך, המוחזר כרלוונטי לשאילתה מסוימת, את חמשת הישויות הכי דומיננטיות ע"פ אלגוריתם שנפרט בהמשך.
* בנוסף, בשלב של הוצאת הישויות הבנו שנשמרות לנו ישויות שהן לא בהכרח ישויות לכן שיפרנו את אופן זיהוי הישות במסמך מה שגרר להקטנה משמעותית של קבצי הפוסטינג שלנו וגודל המילון.

מחלקות חדשות שהוספנו:

בחלק זה התבקשנו להוסיף שתי מחלקות Searcher ו- Ranker.

בנוסף לשתי מחלקות אלו החלטנו להוסיף מחלקת עזר בשם TermQueryואת המחלקה Semantic.

TermQuery

מחלקה זו תשמור לנו מידע על הterm בשאילתה שמופיע במסמך מסוים. בעצם המחלקה תעזור לנו בשליפת הנתונים הרלוונטיים ברגע שנחשב את דירוג המסמך.

משתנים במחלקה:

**name**- מחזיק את שם הterm.  
**numInDoc**- מספר המופעים של הterm במסמך.

**numOfDocs**- מספר המסמכים שבהם הterm מופיע.

**numInQuery**- מספר המופעים של הterm בשאילתה.

**isLoc**- שווה ל-1 אם הterm נמצא ב100 מילים הראשונות של המסמך, אחרת 0.

**isEntity** – שווה ל-1 אם הterm הוא ישות, אחרת 0.

**isMaxTerm**- שווה ל-1 אם הterm הוא המילה שמופיעה הכי הרבה פעמים במסמך, אחרת 0.

המחלקה מכילה Constructor וGetter and Setter.

Ranker

מחלקה שתפקידה לדרג את המסמכים הרלוונטיים לשאילתות המתקבלות.

משתנים במחלקה:

**docsAfterRank**- משתנה מסוג hash שבkey מחזיק string של שם המסמך ובvalue מחזיק את הדירוג של המסמך עבור השאילתה.

**termsQueryInDoc**- משתנה מסוג hash שבkey מחזיק אובייקט מסוג Doc המכיל את שם המסמך ופרטים רלוונטיים על המסמך שיעזרו לנו בחישובים, בvalue מחזיק רשימה מקושרת של אובייקטים מסוג TermQuery. בעצם עבור כל מסמך שומר את הפרטים על הterms שמופיעים גם בשאילתה וגם באותו מסמך.

**postingPath**- מחזיק את הנתיב שבוא נמצאים קבצי הפוסטינג שלנו.

**Indexer**- אובייקט של המחלקה Indexer.

**docs**- משתנה מסוג hash שטוען אליו את הפרטים על כל המסמכים בקורפוס, פרטים אלה נשמרו במהלך הרצת חלק א' ונכתבו ביחד עם קבצי הפוסטינג לקובץ בדיסק.

**avgDocs** – משתנה המכיל את ממוצע אורך המסמכים בקורפוס.

**k,b** – פרמטרים קבועים המשמשים לחישוב נוסחת BM25.

בנאי המחלקה מאתחל את כל המשתנים וקורא לפונ' readDocsPosting() שעוברת שורה, שורה על קובץ הפוסטינג שמכיל מידע על המסמכים בקורפוס ומכניסה את כל המידע למבנה הנתונים docs.

שיטות המחלקה:

**rateDocs** – הפונקציה הראשית של המחלקה, מקבלת מהמחלקה Searcher מבנה נתונים המחזיק את הterms של השאילתה כולל מספר המופעים שלה בשאילתה. הפונקציה נעזרת בפונקציות נוספות של המחלקה ולבסוף מחזירה לSearcher מבנה נתונים שמכיל את כל המסמכים הרלוונטיים לשאילתה ואת דירוגם.

**finalRank**- הפונקציה סוכמת עבור כל מסמך שנמצא כרלוונטי את הדירוג שלו לפי הדירוג של כל terms המופיעים במסמך.

**getRankTerm**- הפונקציה מבצעת דירוג עבור כל term במסמך לפי נוסחת הBM25 (שנפרט עליה בהמשך) בתוספת של אינפורמציה שעוזרת לדירוג טוב יותר.

**readDocsPosting** – פונקציה שמכניסה את כל הנתונים על המסמכים ששמרנו בחלק א' למבנה נתונים.

**lineFromPosting**- הפונקציה מקבלת term ומחזירה את השורה שלו עם כל המידע עליו מקובץ הפוסטינג ששמרנו בחלק א'.

Searcher

מחלקה שתפקידה לבצע שאילתות. המחלקה תקבל שאילתה בודדת או קובץ של שאילתות, תשלח אותן למחלקה Parse על מנת שיעבור פרסור בהתאם לניתוח הטקסט שנעשה על כל המסמכים בקורפוס.

משתנים במחלקה:

**parser**- אובייקט של המחלקה Parser.

**indexer**- אובייקט של המחלקה Indexer.

**ranker**- אובייקט של המחלקה Ranker.

**semantic**- אובייקט של המחלקה Semantic.

- **isSemanticOn** משתנה בוליאני המחזיק true אם המשתמש בחר הרצה עם .semantic  
- **isStemmerOn** משתנה בוליאני המחזיק true אם המשתמש בחר הרצה עם stemmer.  
**docsForQueries** – מבנה נתונים ממוין המחזיק את מספר השאילתה ואת המסמכים הרלוונטיים עבורה לפי דירוגם (רק 50 מסמכים).  
-**pathToSave** נתיב שבו נשמור את התוצאות המתקבלות.  
 - **relevantDoc** מבנה נתונים המחזיק את שם המסמך ואת חמשת הישויות הדומיננטיות עבור אותו מסמך.

מטודות המחלקה:

**readQuery**- הפונקציה קוראת שאילתה בודדת או קובץ של שאילתות דואגת להעביר אותם פרסור ולשלוח לדירוג וקבלת רשימת מסמכים הרלוונטיים לכל שאילתה.

**semanticFunc**- במידה והמשתמש בחר באופציה של semantic הפונקציה מקבלת את תוכן השאילתה לאחר פירסור ושולחת למחלקה Semantic המילים שיחזרו יתווספו לתוכן השאילתה.

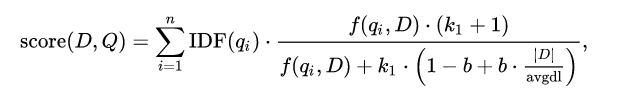
**relevantDocForQuery** – מקבלת את הterms הרלוונטיים עבור שאילתה דואגת לשלוח לדירוג במחלקת Ranker ולהכניס למבנה docsForQueries את החמישים המסמכים המדורגים הכי גבוה עבור השאילתה.

**writeToDisk** – כותבת את התוצאות המתקבלות לקובץ בנתיב הנבחר.

**insertDominantEntities** – הפונקציה עוברת על הקובץ ששמרנו בחלק א' המכיל לכל מסמך את הישויות שלו, מוודאת שהישות היא באמת ישות (מופיעה בשני מסמכים או יותר בקורפוס, כלומר נמצאת במילון). עבור כל מסמך שאוחזר ונמצא כרלוונטי הפונקציה מפעילה נוסחה לחישוב הדומיננטיות של ישות במסמך (עליה נפרט בהמשך) ושומרת במבנה נתונים עבור כל מסמך את חמשת הישויות הכי דומיננטיות.

**פירוט אלגוריתם הדירוג:**

אלגוריתם זה מתבצע עבור כל מסמך שמכיל את הterms של השאילתה (לפחות term אחד). כפי שהוגדר לנו במטלה, נבצע דירוג למילה במסמך לפי נוסחת BM25 (לכל מילה מהשאילתה עם המסמכים בהם הופיעה) ולפי נתונים נוספים שנפרט בהמשך. בכדי לקבל דירוג סופי למסמך ,עבור כל מסמך סכמנו את הדירוג של כל המילים מהשאילתה שהופיעו במסמך זה. עבור BM25 השתמשנו בנוסחה מויקיפדיה :





N – מספר המסמכים בקורפוס.

– מספר המסמכים בקורפוס המכילים את המילה.

– תדירות המילה במסמך (מספר הפעמים שמילה הופיעה בשאילתה כפול מספר הפעמים שהמילה הופיעה במסמך).

|D| -אורך המסמך.

avgdl- אורך ממוצע של המסמכים בקורפוס.

K =1.2

b=0.75

הדירוג של מילה במסמך מוגדר כך: משקל של 0.7 לנוסחת הBM25 + 0.05 אם מילה מופיעה ב100 מילים הראשונות של המסמך + 0.2 אם מילה היא ישות במסמך + 0.05 אם המילה היא המילה שמופיעה הכי הרבה במסמך. לאחר כמה הרצות של שאילות במנוע משקלים אלו נמצאו הכי מתאימים לאחזור מסמכים רלוונטיים עבור שאילתות.

בדקנו ומצאנו שברגע שהמילה היא ישות במסמך המסמך הוא רלוונטי יותר לשאילתה ולכן הוא יקבל דירוג גבוה יותר. בנוסף, הבחנו שכאשר יש מספרים בשאילתה חוזרים לנו המון מסמכים שלא בהכרח רלוונטיים לשאילתה לכן החלטנו להתעלם ממספרים בשלב הדירוג, דבר שגם הוכח כנכון בשלב בדיקת ביצועי המנוע.

**דגש נוסף**- בשלב הסכימה של דירוג המילים במסמך, על מנת לקבל דירוג סופי למסמך הנמצא כרלוונטי לשאילתה, בחרנו לתת משקל שונה לדירוג של מילים שהגיעו מאלגוריתם הסמנטיקה (במידה והמשתמש בחר באופציה הזאת). מילים אלה לא בהכרח המילים שהמשתמש התכוון אליהם בזמן כתיבת השאילתה והם לא חלק מהשאילתה המקורית ולכן יקבלו אחוז קטן יותר בשלב הדירוג הסופי.

**אלגוריתם למציאת 5 הישויות הדומיננטיות במסמך:**

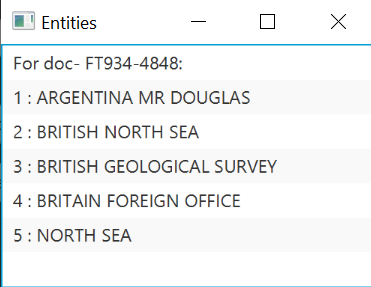
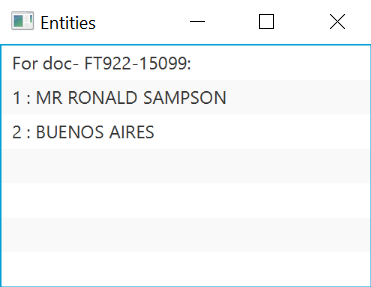
ביצענו שינוי בחלק א' כך שאנחנו שומרות את הישויות שמצאנו עבור כל מסמך ואת מס' הפעמים שהישות מופיעה במסמך ממוינות בסדר יורד (מהישות שמופיעה הכי הרבה במסמך עד לזאת שמופיעה הכי פחות). עבור כל מסמך שאוחזר אנחנו עוברות על השורה שלו בקובץ הישויות ששמרנו, מוודאות שאכן הישות היא ישות, כלומר הופיעה בשני מסמכים או יותר בקורפוס בעזרת בדיקה האם היא נמצאת במילון.

אנחנו מוציאות את מס' הפעמים שהישות מופיעה במסמך ומחלקות במס' המסמכים שהישות מופיעה בהם (מידע ששמור במילון) ושומרות במבנה נתונים. לאחר מעבר על כל הישויות הרלוונטיות עבור מסמך אנחנו שומרות עבור המסמך את ה-5 ישויות שמדורגות גבוהה ביותר, כלומר הדומיננטיות ביותר.

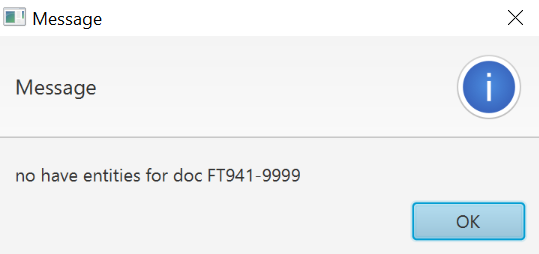
**נוסחה:**

*דוגמאות:*

*במידה וקיימות ישויות דומיננטיות עבור המסמך:*



*במידה ולא קיימות ישויות דומיננטיות עבור המסמך:*



**אלגוריתם לשיפור סמנטי:**

האלגוריתם מופעל רק כאשר המשתמש מעוניין בחיפוש סמנטי. האלגוריתם מוסיף ל-terms שחזרו מה-parse עבור השאילתה את המילים הסמנטיות.

החיפוש הסמנטי מתבצע ב-offline מימוש האלגוריתם פועל באופן הבא:

טוענים קובץ טקסט לצורך "למידת" המילים הסמנטיות.

שימוש בספריית "Medalia" לצורך מימוש האלגוריתם, ובעזרתה מצאנו מילה נרדפת או דומה למילה מסוימת מהשאילתה. נוסיף את המילה שחזרה למבנה הנתונים המחזיק את ה-terms מהשאילתה כדי לבצע עבורה את תהליך הדירוג חוץ מהדירוג הסופי.

למילה שחזרה מהטיפול הסמנטי נוסיף '\*' על מנת שנוכל לזהות אותה בדירוג הסופי של המסמך וניתן לה משקל של 0.02 מאחר והיא מילה שלא הופיעה בחיפוש המקורי של המשתמש ונרצה שיהיה לה השפעה נמוכה ביחס למילים המקוריות בשאילתה.

***כיצד הנתונים במילון ובפוסטינג תומכים באלגוריתם שממשנו:***

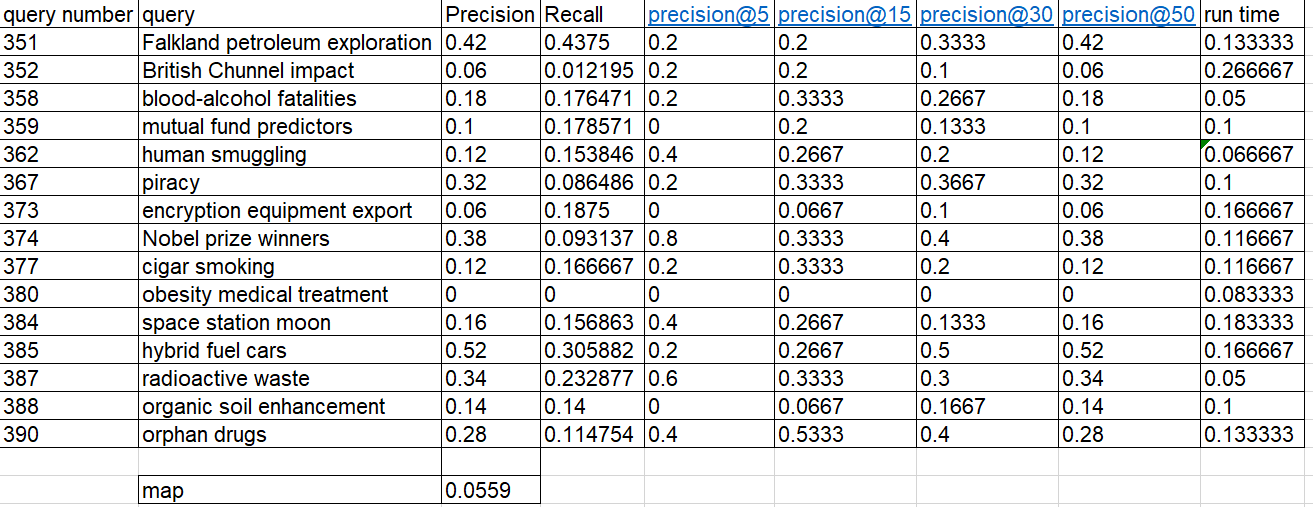
נתונים מהמילון- במילון שמרנו את הterm ומספר המופעים שלו בקורפוס. השתמשנו בנתון של מספר המופעים של מילה בקורפוס על מנת לדרג את המילה במסמך.

נתונים מקבצי הפוסטינג- בקבצי הפוסטינג עבור כל term שמרנו את כמות המסמכים שבו הוא מופיע, באלו מסמכים הוא הופיע וכמות המופעים באותו מסמך נתונים אלו עזרו לנו בשלב חישוב הדירוג של מילה במסמך. בנוסף שמרנו בחלק א' גם את המיקומים של מילה במסמך ואם המילה היא ישות במסמך הנתונים האלה עזרו לנו להוסיף משקל לדירוג הסופי ולקבוע אם המסמך רלוונטי יותר עבור השאילתה.

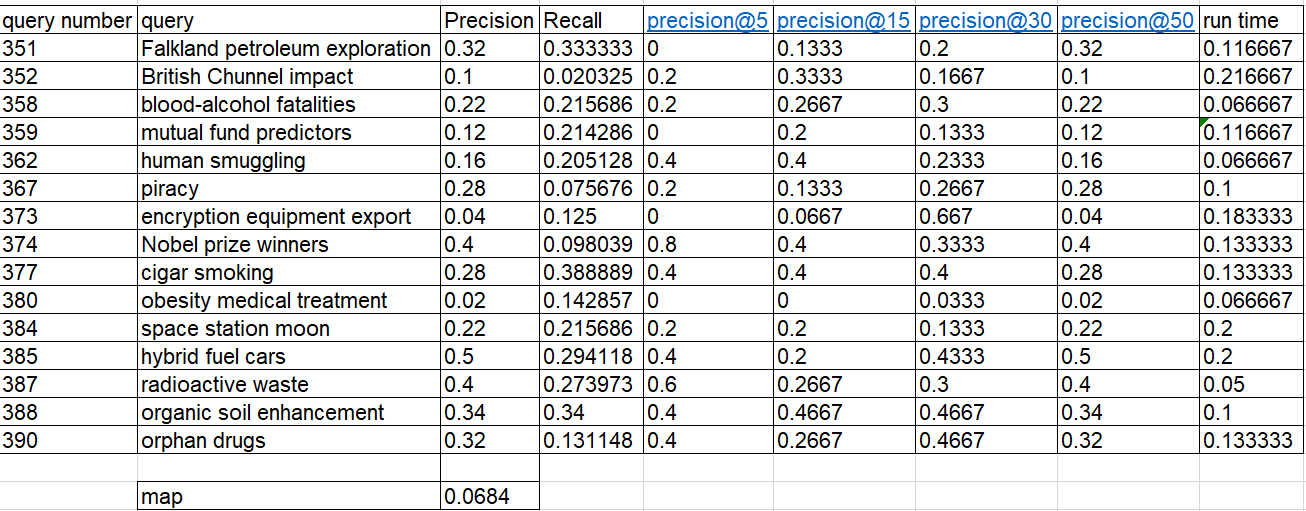
נתונים מקובץ המסמכים- בחלק א' שמרנו קובץ פוסטינג עם נתונים על כל מסמך בקורפוס, מקובץ זה שלפנו בשלב הדירוג את אורך המסמך המבוקש ואת הterm שמופיע הכי הרבה פעמים במסמך.

**הערכה של המנוע:**

**Without stemming**



**With stemming**



**סיכום:**

בעיות שנתקלנו בהם:

בכל שלבי בניית המנוע נתקלנו בבעיות של חוסר זיכרון וזמני ריצה. בחלק זה של העבודה היינו צריכים לשלוף מידע רב על המון מסמכים מהקורפוס, לכן במקום לגשת כל פעם לקרוא מהדיסק את נתונים אלו דאגנו לקרוא את כולו פעם אחת ולשמור בזיכרון את הנתונים על כל המסמכים בקורפוס.

בעיה נוספת היא כשהבנו שהדירוג BM25 אינו מספיק לאחזור המסמכים הרלוונטיים ולכן היינו צריכות לחשוב איזה נתונים להוסיף לדירוג על מנת לשפר את האיחזור, לאחר מציאת נתונים אלה (האם המילה היא ישות, מופיעה בתחילת המסמך...) ביצענו כמה וכמה הרצות על מנת להחליט מה הם המשקלים המתאימים לכל חלק בדירוג. ובנוסף, ברגע שהוספנו לדירוג את תיאור השאילתה (description) הדבר עזר לדירוג והחזיר מסמכים רלוונטיים יותר עבור השאילתה.

עוד בעיה שנגררה איתנו עוד משלב א' של העבודה וגילינו עליה בשלב הזה היא שמירת הישויות במסמכים. ברגע ששלפנו עבור כל מסמך את הישויות הדומיננטיות עבורו הבחנו בהמון ישויות שהן לא ענייניות ולא היינו מגדירות אותן כ"ישות", החלטנו לגשת לחלק א' של העבודה ולשפר את האלגוריתם שמזהה לנו ישויות דבר הגרר אחזור טוב יותר של מסמכים והקטנה משמעותית של קבצי הפוסטינג וקובץ המילון שלנו.

*האתגרים הגדולים בפרוייקט:*

* *להביא מקסימום תוצאות במינימום זמן. (זמני הריצה בחלק ב' פחות באים לידי ביטוי כי המסה הגדולה של הנתונים היא בחלק א').*
* *לבצע parser במידה מדויקת ככל הניתן, כך שלא מורידים יותר מידי terms ומצד שני גם לא "מזבלים" את הindex במונחים לא מדויקים.*
* *למצוא אלגוריתם דירוג מדויק שיאחזר כמה שיותר מסמכים רלוונטיים לשאילתה.*

*מה היינו עושות אחרת? מאמינות שעם חשיבה מקדימה ותכנון נכון יכולנו לפרסר את המסמכים בצורה טובה יותר. במידה והיה לנו יותר זמן היינו מנסות למצוא אלגוריתם דירוג מדויק יותר המאחזר מסמכים רלוונטיים יותר לשאילתה. בנוסף, בחלק א' של העבודה יכולנו לשמור נתונים נוספים על מסמך שיתאר יותר את המסמך ויעזור בשלב האחזור.*