**מנוע חיפוש חלק ב'- דו"ח**

עילי פרידמן 305709248

אורן שור 204076053

**חלק א':**

תיאור אופן פעולת מנוע החיפוש:

בחלק א' של העבודה נוצר אינדקס הופכי וזהו אופן שמירת קבצי Postings בדיסק:

1. קובץ Posting בשם finalIndex שהוא קובץ טקסט (.txt) המכיל את כל המידע על כל הביטויים הקיימים במאגר המסמכים ברצף, ללא הפרדה בין ביטויים.
2. מילון הביטויים הסופי בשם baseDict שהוא קובץ מסוג Ujson (.ujson) המכיל את הנתונים על המידע והמיקום שלו בקובץ ה- Posting. המילון בנוי כך:

Term: [offset, size, num of occurrences in all the corpus, num of docs]

1. מילון המסמכים בשם fileIndex שהוא קובץ מסוג Ujson (.ujson) המכיל את כל המידע עבור קובץ מסוים.

המילון בנוי כך:

Doc name: [max\_tf, num of distinct terms, city of doc, language, doc length, entities list]

1. מילון הערים בשם citiesIndex שהוא קובץ מסוג Ujson (.ujson) המכיל את כל המידע על הערים המופיעות במאגר המסמכים. המילון בנוי כך:

City: [Country, currencies, population, docs containing the city, num of occurrences and locations]

1. רשימת שפות בשם languageList השמורה בקובץ טקסט (.txt) המכיל את כל השפות המשויכות למסמכים במאגר.

על מנת להתחיל בתהליך אחזור המידע על המשתמש לטעון את הקבצים הדרושים לזיכרון התוכנית ע"י בחירת הנתיב בו נמצאים קבצי ה- Postingsבמיקום ה- "Saving path" ולחיצה על כפתור ה- "Load Files from path". לאחר טעינת הקבצים לזיכרון ניתן יהיה להזין שאילתה יחידה או להזין מיקום בו נמצא קובץ של שאילתות.  
בנוסף, ניתן לבחור מהאפשרויות: חיפוש סמנטי, מיון תוצאות החיפוש לפי ערים נבחרות והצגת הישויות הדומיננטיות בכל מסמך שחזר.

המחלקות שהוספנו בחלק זה:

1. **מחלקת Searcher:**

הפרמטרים:

* stopWords- רשימה ה- Stop words שמהן עלינו להתעלם.
* postPath- הנתיב בו נמצאים קבצי ה- Postings.
* doStem- משתנה בוליאני המורה אם לבצע Stemming או לא.
* baseDict- מילון הביטויים מבין קבצי ה- Postings.
* fileIndex- מילון הקבצים מבין קבצי ה- Postings.
* showEntities- משתנה בוליאני המורה אם להציג את הישויות הדומיננטיות או לא.
* doSemantics- משתנה בוליאני המורה אם לבצע חיפוש סמנטי או לא.
* citiesList- רשימת הערים שלפיה נמיין את התוצאות החוזרות מהשאילתה, במידה ולא נרצה למיין תהיה NULL.

השדות:

* renker: משתנה מהטיפוס Renker.
* similarityDict: מילון בו נשתמש בעת חיפוש סמנטי.

המטודות:

* **singleQueryCalc(query)-** בפונקציה זו מתבצע אחזור מידע בעבור שאילתה יחידה. תחילה אנו מפרסרים את שאילתה ע"מ לקבל רשימה של ביטויי השאילתה, ולאחר מכן נשלחת רשימה זו ל- renker לחישוב ציון ההתאמה של כל מסמך במאגר לשאילתה הנוכחית.
* **multiQueryCalc(queryFile)-** במטודה זו נבצע אחזור מידע בעבור כל שאילתה מתוך קובץ שאילתות נתון. בדומה למטודה הקודמת.
* **addEntities(docList)-** בעבור כל מסמך בתוצאות פונקציה זו מוסיפה את רשימת הישויות הדומיננטיות שלו.

1. **מחלקת Renker:**

הפרמטרים:

* baseIndex- מילון הביטויים מבין קבצי ה- Postings.
* docIndex- מילון הקבצים מבין קבצי ה- Postings.
* postPath- הנתיב בו נמצאים קבצי ה- Postings.
* doStem- משתנה בוליאני המורה אם לבצע Stemming או לא.

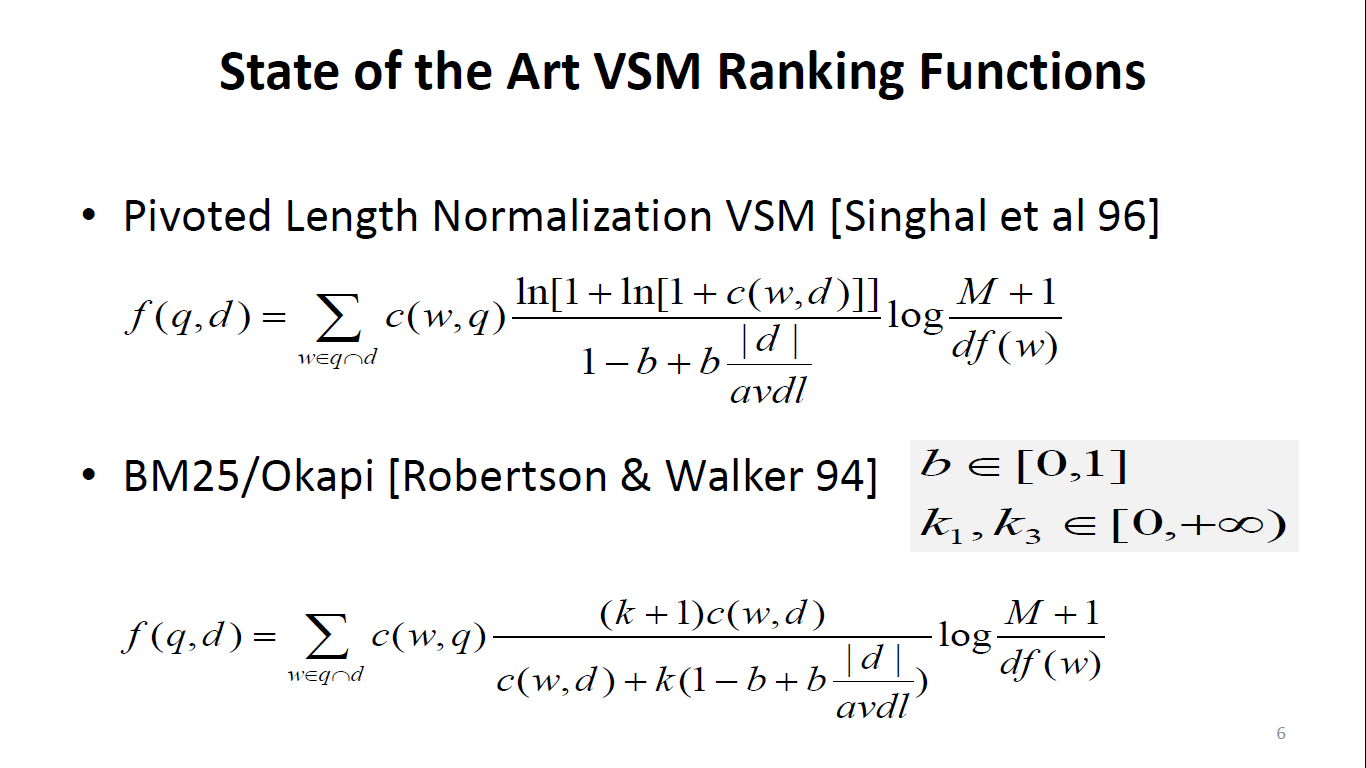
השדות:

* Avdl- האורך מסמך ממוצע בכל המאגר.
* docNum- כמות המסמכים בכל המאגר.

המטודות:

* **calculateRate(query)-** בעבור כל ביטוי מהשאילתה נקרא המידע השמור עליו וזאת על מנת לחשב את הדירוג של כלל המסמכים במאגר בהתאם לשאילתה הנוכחית בעזרת האלגוריתם BM25.
* **bmCalc(cWQ,cWD.docLen.df)-** פונקציה המחשבת את האלגוריתם BM25 בעבור כל ביטוי בשאילתה התאם למסמך הנבדק הנוכחי.
* **readFromFile(token)-** פונקציית קריאה מהקובץ finalIndex שהינו קובץ ה – Posting של המנוע. בקובץ זה ישנו מידע בעבור כל ביטוי במאגר.

האלגוריתמים הכלולים במנוע:

* אלגוריתם הדירוג: באחזור המידע במנוע זה השתמשנו באלגוריתם BM25 על מנת לחשב את ציון הדירוג בעבור כל מסמך. את חישוב הציון ביצענו באמצעות הנוסחה הבאה:

כאשר:

* b=0.75 – בדומה לשימוש בתרגולים
* בחרנו לחשב באמצעות
* c(w.d)- מספר הפעמים שמופיע הביטוי w במסמך d.
* c(w.q)- כמות הפעמים שמופיע הביטוי w בשאילתה q.
* |d|- אורך המסמך d (ללא stop words).
* avdl- אורך המסמך הממוצע במאגר המסמכים.
* M- כמות המסמכים במאגר.
* df(w)- מספר המסמכים בהם מופיע הביטוי w במאגר המסמכים.

תהליך הדירוג: בעבור כל שאילתה יצרנו רשימה של כל מילות השאילתה ובנוסף הוספנו את המילים הרלוונטיות בלבד מחלק ה- Description ומחלק ה- Narrative בעבור השאילתה, ובמידה ונבחר חיפוש סמנטי הוספנו גם את המילים הדומות למילות בשאילתה. לאחר מכן חישבנו את מדד ה- BM25:

* בעבור המילים המקוריות מהשאילתה (Q) נחשב עם הפרמטרים: b=0.75, k=1.9.
* בעבור המילים שנוספו מחלק ה- Description ומחלק ה- Narrative (DN) נחשב עם הפרמטרים: b=0.75, k=1.85 ואת התוצאה נכפיל ב- 0.6.
* בעבור המילים הדומות בחיפוש הסמנטי (SIM) נחשב עם הפרמטרים: b=0.75, k=1.2, ואת התוצאה נכפיל ב- 0.05.

המשקולות של החלקים השונים נקבעו על בסיס ניסוי וטעייה, עד לקבלת פרמטרים המחזירים תוצאה אופטימלית. תוצאת הדירוג של מסמך הינה סכום החישובים הנ"ל:

* מציאת ישויות דומיננטיות: בעת יצירת האינדקס בחלק א' אנו שומרים בעבור כל מסמך את רשימת הישויות המלאה שלו. לאחר מכן בעת אחזור המידע במידה ונבחר ע"י המשתמש להציג את רשימת הישויות הדומיננטיות בעבור התוצאות, נעבור שוב על רשימת הישויות בעבור כל מסמך הרלוונטי לשאילתה ונסדוק מול מילון הביטויים של המאגר כולו ע"מ לבדוק אם הישות הנבדקת אכן ישות בכל המאגר. נחשב את הדירוג של הישויות הדומיננטיות של המסמך לפי: , ועל פי חישוב זה נציג את חמשת הישויות הדומיננטיות.
* אלגוריתם לשיפור סמנטי: השתמשנו במאגר מילים דומות שהורדנו מהאתר <http://paraphrase.org>, יצרנו ממאגר זה מילון (word: list of similar words) שבו מופיעות בעבור כל מילה מהשפה האנגלית רשימה של המילים הדומות לה ע"פ סדר עולה מהמילה הדומה ביותר לדומה פחות. בעבור כל מילה מהשאילתה הוספנו את המילים הדומות לה ע"פ המילון הנ"ל לשאילתה ובכך הגדלנו את האפשרויות למציאת מסמכים רלוונטיים.

**חלק ב':**

* ביצועי המנוע ותוצאותיו על קובץ השאילתות ללא Stemming וללא חיפוש סמנטי:



* ביצועי המנוע ותוצאותיו על קובץ השאילתות עם Stemming וללא חיפוש סמנטי:



* ביצועי המנוע ותוצאותיו על קובץ השאילתות עם Stemming ועם חיפוש סמנטי:



**חלק ג':   
סיכום:**   
**במהלך הפרויקט התמודדנו 2 אתגרים מרכזיים שונים:**

* זמני ריצה: האתגר המשמעותי של התוכנית. כאשר התחלנו לממש את מחלקת Parse נוכחנו להבין כמה חשוב לבצע שימוש נכון באופרטורים שונים וספריות שונות על זמני הריצה. מכיוון שהתעסקנו עם מאגר מידע גדול – ספריות שונות אשר הכרנו כמהירות מאוד (דוגמת re) התגלו כ'צוואר בקבוק צר' אשר גוזל זמן ריצה יקר (על כן לדוגמא, בחרנו לוותר על שימוש בregex-ים ובפונקציות נוספות מספריית re שמאוד נוחות אך גוזלות זמן). זאת ועוד , הבנו כי ישנה חשיבות גדולה בהבנת האופן בו מאגר המידע עצמו בנוי וחיזוי הצפי של הקלט. לדוגמא – סידור התנאים השונים בביצוע parse קיצר משמעותית זמני ריצה – כאשר הבנו וצמצמנו את מספר התנאים שכל token נאלץ לעבור למינימום והפכנו את התנאים להיות ייחודיים ומדויקים יותר הצלחנו לשפר זמני ריצה.  
  בנוסף גילינו, את המשמעות להעתקות וחוסר ניקוי הזיכרון מבחינת זמן ריצה: רשימות שהצטברו לארוכות מאוד, או משתנים שלא אופסו או נמחקו מזיכרון התוכנית בסוף כל שימוש האטו מאוד את הקצב.  
  לבסוף כתבנו את מחלקת parse וindexer פעמיים – כאשר בפעם השנייה מימשנו את כל הכתוב לעיל.  
  נציין כי על אף החשש ששפת התכנות (python) תהווה אתגר משמעותי מבחינת זמני ריצה (לעומת שאר השפות) – גילינו שהיא אינה מהווה בעיה גדולה מדי ועמדנו ביעדי הריצה לחלוטין.  
  מעבר לכך , כידוע זמני הריצה של כתיבה וקריאה מהדיסק לקחו את מירב הזמן. בעקבות בעיות זיכרון (יפורט בהמשך) נאלצנו לחשוב על פתרון אשר יבצע זאת במהירות . הבנו כי בשוק משתמשים במחלקת json אשר יודעת לבצע קריאה ושליפה מהירה של הנתונים ואף נחשפנו למחלקה המשופרת שלה ultra json אשר הביאה תוצאות מהירות אף יותר. ללא ספק היה פתרון שהועיל מאוד בכל חלק החל מהשלב הראשוני ועד האחרון בריצת התוכנית.
* זיכרון התוכנית: בשל מגבלות הפרויקט וסביבת הפיתוח – לא היה ביכולתנו לשמור את כלל המידע שנאסף בתהליך יצירת קבצי ה-Postings בזיכרון התוכנית. לכן פיצלנו את תהליך זה למספר שלבים בכל אחד מהם נכתב חלק מן המידע לדיסק ולבסוף שמרנו בזיכרון התוכנית את המילון שנאסף רק עם המפתחות והמצביעים לקבצי ה-Postings- שנכתב גם הוא לבסוף לדיסק. כאמור השתמשנו גם במחלקת UJSON אשר תרמה מאוד לשליפה והקריאה מהדיסק.   
  התמודדנו גם עם שיפור בעת מתן מענה על שאילתות – במקום לעבור על כלל המאגר , נעזרנו במילון שמסודר על פי שמות מסמכים ובדקנו רק את המסמכים הרלוונטיים. בכך שיפרנו את זמן ההמתנה ואת הזיכרון שבהמתנה למענה על שאילתות.

אנחנו מאמינים שהאתגר המרכזי בפרויקט היה השילוב בין 2 האתגרים שהצגנו : זמן ריצה והזיכרון של התוכנית. במהלך כלל כתיבת הקוד למדנו להבין את הרווח וההפסד בהרצה בשתי אפשריות: אפשרות אחת היא **הרצה באופן מהיר יותר אך זיכרון התוכנית מלא יותר** לעומת האפשרות ההפוכה של **הרצה איטית יותר אך זיכרון התוכנית פחות מלא**. המשחק בין 2 אלמנטים אלה ומציאת האיזון האופטימלי ליווה אותנו לאורך כל כתיבת הקוד. בנוסף, אנו מאמינים כי יש חשיבות גדולה למתן דיוק בדרישות של המנוע. הרי יכולות האחזור וניהול המידע כל כך רחבות שניתן לקבל תוצרים דומים מאוד אך שונים מאוד. מצאנו שבמהלך כתיבת הפרויקט איננו יודעים תמיד מה בדיוק היה נדרש (או לחילופין : מהו היקף המידע שעבורו דרישה מסוימת תקפה). מה שהנחה אותנו הוא שהאחזור נמדד בהתאמה סופית של תוצאות השאילתות ולא בשלבים שבדרך.

**שיפורים לתוכנית:**

* לדעתנו ניתן היה לשפר את האחזור שלנו בעזרת שמירת מידע נוסף על ה-term-ים והמסמכים במאגר (לדוגמא: שמירת הכותרת של כל מסמך באופן יותר מפורט ונתינת משקל גדול יותר למילים שלו, או לחילופין שמירת מיקומים יותר פרטנית של המילים ונתינת משקל גדול יותר למילים שנמצאות באותו אזור בטקסט).
* הוספת פונקציות אחזור נוספות עבור מחלקת Ranker, לדוגמא CosSim (למרות שניסינו להוסיף וזה לא שיפר את האחזור אלא גרע ממנו).
* ניתן היה לבצע Parse בתדירות גדולה יותר (כיום מבוצע עבור כל מסמך) ובכך לשפר את זמני הריצה (ובתמורה להעמיס יותר על זיכרון התוכנית כאמור..) ,
* ניתן לשפר את יכולת הסמנטיקה שלנו בכך שהיינו מגדירים ממדים נוספים עבור כל וקטור, או משתמשים במאגר מילים רחב יותר המכיל יותר מילים דומות בעבור כל מילה.