תיאור הפרויקט:

חלק 1 : בניית inverted index

בחלק זה בניתי מבנה נתונים באופן דומה למה שראינו בכיתה, בטבלת האש שמרתי לכל מילה מיפוי לרשימה של מסמכים שיש בהם את המילה הנ״ל, את כמות המופעים של המילה בכל מסמך, את הidf הרגיל, את הbm25 idf וכו.

יש לציין שעל מנת לחלץ מידע השתמשתי אך ורק בRECORDNUM,TITLE,ABSTRACT,EXTRACT על מנת לחלץ מידע מהמסמכים.

בנוסף שמרתי כמה דברים נוספים על מנת לחסוך זמן מענה לשאילתה אותם חישבתי offline כמו גודל מסמך ממוצע, לכל מסמך את גודלו וכו(מקל על חישוב bm25 ביעלות)

פונקציות בחלק זה:

```
def prologue():
```

פונקציה זו שומרת את הstopwords בתוך מבנה נתונים(אותו מבנה נתונים שמחזיק את הinverted index)

```
def calc_documents_length():
```

פונקציה זו מחשבת לכל מסמך את אורך המסמך

```
def create_index():
```

פונקציה זו מכניסה לinverted index מיפוי ממילה לרשימה של מסמכים שמכילים את אותה המילה וכמה פעמים המילה מופיעה בכל מסמך, בעצם היא מממשת את הפאודו קוד הבא:

Creating an Inverted Index

```
Create empty HashMap, H;
For each document, D:          // i.e. file in an input directory
    Create a HashMapVector,V, for D;
    For each (non-zero) token, T, in V:
        If T is not already in H,
            Create an empty TokenInfo for T and insert it into H;
        Create a TokenOccurence for T in D and
            add it to the occList in the TokenInfo for T;
```

```
def calculate_idf():
```

בפונקציה זו מחושב ציון הidf (לפי tf-idf) של כל המילים במילון

$$\log_2\left(\frac{|D|}{df_i}\right)$$

```
:()def calculate_bm25_idf
```

כמו למעלה, אבל ציון הidf שמתאים לbm25

$$idf(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

```
:(def dic_to_json_file(dic,file_name
```

פונקציה זו מקבלת מילון ושם קובץ כקלט וכותבת את המילון בפורמט json אל תוך קובץ בשם שקיבלה כקלט, משתמשים בפונקציה זו על מנת לשמור את הinverted index ושאר המידע המחושב כדי שיהיה זמין בזמן קבלת שאילתה מהמשתמש

```
:()def handle_creating_index_request
```

פונקציה זו נקראת פעם אחת כאשר התוכנית קיבלה כקלט בקשה לבניית inverted index, היא קוראת לprologe וקוראת לכל הפונקציות הרלוונטיות על מנת לבנות את האינדקס (create index, calculate idf, calculate bm25 idf, calc document lengths, dic to json file)

חלק שני: החזרת המסמכים הרלוונטיים בהינתן שאילתה מהמשתמש

תיאור הפונקציות:

```
:(def create_hash_map_vector_for_query(query
```

פונקציה זו מקבלת שאילתה ומחזירה טבלת האש שממפה כל מילה שמופיע בשאילתה(לאחר ביצוע טוקניזציה וסטמינג) למספר המופעים שלה בשאילתה

```
:()def calc_similarity_tf_idf
```

פונקציה זו מחשבת את הדמיון בין השאילתה ובין כל מסמך בשיטת tf idf באמצעות שימוש בhash שבנינו עבור השאילתה, פונקציה זו מממשת את הקוד הבא:

Retrieval Algorithm using Inverted-Index

```
Create HashMapVector, Q // for the query
Create empty HashMap, R // store retrieved documents with scores
For each token, T, in Q:
    Let I = IDF(T), K = tf(T, Q); // K is the count of T in Q
    Set W = K * I; // weight of token T in query Q
    Let L be the list of TokenOccurences of T from H; // H is the inverted index
    For each TokenOccurence, O, in L:
        Let D = document(O) and C = tf(T, D);
        If D ∉ R: // D was not previously retrieved
            Set R[D] = 0.0;
        Set R[D] += W * I * C; // product of T-weight in Q and D
```

Retrieval Algorithm cont.

```
Compute L = length(Q);  // square-root of the sum of the squares of its weights
For each retrieved document D ∈ R:
    Let S = accumulated score of D; // S is the dot-product of D and Q
    Let Y = length(D); // as stored in its DocumentReference
    Normalize D's final score to S/(L·Y) ;
```


```
:()def bm25
```

פונקציה זו מחשבת עבור כל מסמך את הדמיון שלו לשאילתה ומחזירה את התוצאה

```
:(def retrive_best_documents(sim_func, index_path, query
```

פונקציה זו נקראת פעם יחידה כאשר התוכנית מקבלת כקלט שאילתה, והיא קוראת לכל הפונקציות הרלוונטיות(create hashmap for query, בנוסף היא קוראת לפוקציה שמחשבת את הדמיון בין השאילתה לכל מסמך(tf idf או bm25) וממיינת את המסמכים לפי המסמכים הטובים ביותר בהתחלה, ומחזירה את התוצאה