**דו"ח – חלק א' בניית מנוע לאחזור מסמכים**

**מגישים :**

**גוני לוין חיימי 308326032**

**גל ויצמן 204328405**

**עיצוב התוכנה :**

בחלק זה של העבודה ביצענו עיבוד מקדים למאגר מסמכים סגור אשר קיבלנו מראש. העיבוד כולל קריאת המסמכים מהקבצים עם מידע רב, ביצוע פירסור תוך יצירת קבצי Inverted Index, Posting ומילון מסמכים המרכזים את הנתונים עבור מאגר המסמכים.

הקלט : מאגר מסמכים הכולל 1816 קבצים.

**מחלקות :**

## **מחלקת ה- ReadFile:**

תפקיד המחלקה הוא לקרוא את מאגר המסמכים ע"י קבלת path מן המשתמש. נתיב זה יכיל את התקייה בה יושבים כלל קבצי המאגר. מחלקה זו בתחילה יוצרת מצביעים לכלל הקבצים במאגר , לאחר מכן מבצעת קריאת 50 קבצים בכל איטרציה תוך ביצוע הפרדת שם המסמך והטקסט עבור כל מסמך ושמירה במבנה נתונים List . בנוסף מחלקה זו מנתחת כל מסמך בהתאם לשפת המסמך, כותרת המסמך, עיר ושומרת נתונים אלו במשתנה stringBuilder שיכתב בהמשך לדיסק.

**ReadFile (String path)**

הבנאי של הפונקציה, מקבל כקלט את הנתיב של מאגר הקבצים .יוצר רשימה של מצביעים( משתנה המחלקה filedInFolder ) לכל הקבצים ב-corpus .

**Read()**

פונקציה זו מבצעת קריאה וניתוח ראשוני של 50 קבצים , הפונקציה נעזרת במשתנה המחלקתי jumping50 על מנת לקרוא בכל איטרציה את 50 הקבצים הרלוונטים. פונקציה זו מנתחת כל מסמך מתוך ה-50 קבצים ומפרקת אותו ל- שם המסמך וטקסט (אשר ישמרו במבנה נתונים List ) , כותרת המסמך, עיר ושפת המסמך (אשר ישמרו במשתנה stringBuilder). בפונקצייה זו אנו משתמשים באובייקט מסוג Jsoup שממומש ב jsoup-1.11.3.jar אשר מאחזר string בין תגיות (למשל בין <DOC> ל </DOC> ) ובכך עוזר לנו לפשט את הקוד וכן לייעל את זמני הריצה .

**public void makeCityListAndLanguageList()**

### תפקיד הפונקציה הוא לעבור על כלל הקבצים ב-corpus ולאתר את שפות המסמכים בכל ה-corpus לפי התגית <F P=105> זאת על מנת ליצור מילון שפות שיוצג למשתמש בהתאם להנחיות העבודה.

#### בנוסף הפונקציה מאתרת את עיר המסמך לפי התגית <F P=104> בכדי שנוכל לאתחל את המילון שיגיע ל-parser עם הערים טרם תחילת תהליך הפירסור על מנת שב-parser תהיה את האפשרות להחליט האם term מסוים הוא עיר . בנוסף, אנחנו משתמשים ב API restcountries שנותן לנו אינפורמציה על ערים בעולם ( למעשה, רק על ערי בירה ) : מדינה, גודל אוכלוסייה ומטבע. נשתמש ב strings שיוחזרו מה API וכתובים בצורה של אובייקט Json , ונמיר אותם לאובייקטים מסוג City , באמצעות שימוש באובייקט מסוג objectMapper שממומש ב Jackson-all-1.9.0.jar.

## **מחלקת ה-Parse :**

תפקיד המחלקה הוא לפרק כל מסמך ל-terms אשר נמצאים בו.

**Public Parse(string path,Boolean isSteming)**

הבנאי של המחלקה, מקבל נתיב של ה- stop word ומעדכן את משתנה המחלקה path להיות הנתיב. מקבל משתה בוליאני IsStemming ומעדכן את משתנה המחלקה בהתאם.

### **startParsing50Files(List<Pair <String, String>> mapOfDocs)**

הפונקציה מקבל List אשר מכיל זוגות סדורים של שם המסמך והטקסט עבור כל מסמך. תפקידה של הפונקציה היא לנתב כל מסמך ( שם וטקסט ) לביצוע הפרסור בפונקציית ה-parsingTextToText.

**public void parsingTextToText(String doc, String docName)**

פונקציה זו תפקידה לעבור על כל הביטויים של הטקסט אשר התקבל כקלט , להסיר stop word ו-delimiters ולאפיין כל term לפי סוגו בהתאם לכללים הנדרשים.עבור כל מסמך אנו מוסיפים למפה docsByTerm שתפקידה להחזיק את שם המסמך וכל ה-terms בתוכו זאת אנו מבצעים ע"י מפה בתוך המפה .בסיום אפיון ה-terms השונים במסמך אנחנו מוסיפים למפה termsInDoc את שם המסמך ותדירות ה-term הנפוץ ביותר במסמך.

### **private void addToterms(String str, String docName, boolean isNumber)**

### תפקיד הפונקציה הוא לנתב את ה-term שהתקבל כקלט (str) לפונקציית directAddingTerm.

### ה str שיישלח לפונקציית directAddingTerm לא ישתנה אלא במקרה הבאים:

1. Str מתחיל עם אות גדולה ולא קיים במילון עם אותיות קטנות, אז יישלח כאשר כולו עם אותיות גדולות
2. Str מתחיל עם אות גדולה אך קיים במילון עם אותיות קטנות ולכן יישלח עם אותיות קטנות. במקרה שכזה גם נעדכן את המילון במידה וקיימת כבר באותיות גדולות ונשנה אותה לאותיות קטנות.

### **private void directAddingTerm(String str, String docName)**

פונקציה זו מקבלת כקלט את ה-term ואת שם המסמך, הפונקציה בודקת האם ה-term קיים במילון docsByTerm במידה וכן, מגדילה ב-1 עבור ה-term במילון לפי שם המסמך את מספר הופעות ה-term במסמך. אחרת, מוסיפה את ה-Term למילון ומאתחלת את מספר ההופעות להיות 1.

**private Double isNumber (String str)**

פונקציה זו מקבל מחרוזת כקלט, ומבצעת בדיקה האם המחרוזת שהתקבלה היא מספר . במידה וכן מחזירה את ערך המחרוזת כמשתנה Double במידה ולא, מחזירה NULL.

**private String dealWithNumbers(String docName,Double number, int i, int length, String nextword, String[]onlyTextFromDoc, boolean isBillionAsWord, boolean isDollar, String percent, String fraction, boolean isBillion, boolean isMillion, boolean betweenAsWord)**

פונקציה זו מקבלת מספר בדמות מחרוזת , בנוסף מקבל אפיון ראשוני של ה- term . פונקציה זו מחזירה אפיון מתקדם יותר של המספר בתצורה של sting לאחר שבוצעו עליו ועל המילים הבאות אחריו (במידה והן חלק מה-term )כל הכללים המפורטים בהנחיות העבודה.

**private String numberToTerm (double number, boolean isDollar, boolean isBillion, boolean isMillion, boolean isTrillion, boolean isThousand, String percent, String fraction , boolean isKilogram, boolean isGram)**

פונקציה זו מקבלת מספר בדמות מחרוזת , בנוסף מקבלת אפיון מתקדם של ה-term לאחר שבוצעו עליו כלל הכללים שמפורטים בהנחיות העבודה. תפקיד הפונקציה הוא לסווג את המספר לתצורתו הסופית לאחר הוספת – Kg, M,B.. בהתאם.

**private void dealWithMakaf (String [] split, boolean isDollar,String [] onlyTextFromDoc, int i, String docName)**

פעולה זו מקבלת מערך של מחרוזת כקלט, במחרוזת קיים התו מקף " – " . תפקידה של הפונקציה הוא לנתח את הביטוי משמאל למקף ומימין למקף בהתאם לחוקים ולכללים אשר נקבעו בהנחיות העבודה ( לדוגמא : טווחים, תאריכים או ביטויים עם מקף).

**public void makePostingForCities()**

#### תפקיד הפונקצייה הוא לכתוב לקובץ ה-posting בשם citiesPosting בעבור כל עיר את שם המסמך וכמות הפעמים שהעיר מופיעה במסמך. כל שפה תהווה שורה בקובץ זה , וההפרדה בין מסמכים תתבצע באמצעות ה- delimiter ~. כל זה יתבצע באמצעות מעבר על המילון cities שמהווה משתנה מחלקתי במחלקת ה-parse.

## **מחלקת ה-Indexer :**

## public class Indexer { Map <String,Integer> numberOfDocsPerTerm = new HashMap<>();

### //counts the appearances of term in different documents Map <String,Integer> frequentOfTermInCorpus = new HashMap<>();

### //counts the a ppearances of term in the corpus int numOfDifferentPosting = 37;

#### // writing to 37 final posting, as described below BufferedWriter [] bufferedWritersArray;//as described below String path; //as described below boolean thereIsNoProblemWithBigLetters = true; //as described below public TreeMap<String,Integer> treeMapForDocsPerTerm;

##### // an alphabetical sort of numberOfDocsPerTerm public TreeMap<String,Integer> treeMapForfrequentOfTermInCorpus;

###### //an alphabetical sort of frequentOfTermInCorpus

**public Indexer(String path)**

הבנאי של ה indexer מקבל את הנתיב (**path**) אליו יכתוב את קבצי ה posting. כמו כן, הוא מאתחל את ה bufferedWriters בצורה הבאה:

**bufferedWritersArray[0]** –

כל הביטויים שמתחילים ב’$’ יכתבו ע״י bufferedWritersArray[0] לקובץ posting 0.txt

**bufferedWritersArray[1]** –

כל הביטויים שמתחילים ב’0’ יכתבו ע״י bufferedWritersArray[1] לקובץ posting 1.txt

**bufferedWritersArray[10]** –

כל הביטויים שמתחילים ב’9’ יכתבו ע״י bufferedWritersArray[10] לקובץ posting 10.txt

**bufferedWritersArray[11]** –

כל הביטויים שמתחילים ב’a’ או ב’A’ יכתבו ע״י bufferedWritersArray[11] לקובץ posting 11.txt

**bufferedWritersArray[36]** –

כל הביטויים שמתחילים ב’z’ או ב’Z’ יכתבו ע״י bufferedWritersArray[36] לקובץ posting 36.txt

שימוש ב bufferedWriters משפר את זמני הריצה מכיוון שה bufferedWriters ״אוגרים״ אצלם כמות מסוימת של מידע ורק אז כותבים אותם לדיסק הקשיח. בכך נחסך גישות לדיסק.

**public void index50Files(Map<String,Map<String,Double>> docsByTerm , int currentFileToWrite)**

שלב 1:

**Map<String,Map<String,Double>> docsByTerm:**

<String = term <String = name of document, \*Double = number of appearances of the term in the doc.the first line which the term in the document>

\*Double – לדוגמא, המספר 3.441: 3 מייצג את מספר המופעים של ה term במסמך. 44 היא השורה הראשונה במסמך בו מופיע ה term. 1 היא הספרה הימנית ביותר וזאת בכדי למנוע מ-0 להיות הספרה הימנית ביותר כדי להימנע מביטול לא רצוי של ספרה זו (למשל, אם השורה הראשונה היא שורה מספר 10, אז נרצה למנוע בוודאות שהספרה 0 ולא תרד)

**currentFileToWrite**: מספר קובץ ה posting הזמני

לאחר שה parser סיים את עבודתו על 50 קבצים, ה indexer מקבל ממנו מילון כמתואר למעלה. דבר ראשון, הוא מעדכן את המילון שסופר בכמה מסמכים שונים הופיע term מסויים (במילון numberOfDocsPerTerm) ואת מספר המופעים הכללי של term בקורפוס (במילון frequentOfTermInCorpus). לאחר מכן, באמצעות treeMap, אנו ממיינים את המילון שקיבלנו על פי סדר אלפביתי.

לאחר מכן, אנו כותבים אל קובץ ה postingהזמני באופן הבא: אם ה term מתחיל בספרה, אות גדולה או סימן כלשהו, נכתוב לקובץ זמני בתוך תיקיית big . לדוגמא:

term = ‘GONI’, currentFileToWrite = 10

נכתוב לקובץ posting בתיקיית big , כלומר לתוך big/10.txt

אחרת, כלומר ה term מתחיל באות קטנה, נכתוב אותו לתיקיית small. לדוגמא:

term = ‘gal’, currentFileToWrite = 10

נכתוב לקובץ posting בתיקיית small , כלומר לתוך small/10.txt

עצם ההפרדה בין small ל big וכתיבה לקבצי posting שונים עבור כל 50 קבצים, תאפשר לנו בשלבים מאוחרים יותר להקל על האיחוד בין הקבצים וכמו כן לטפל במקרים בהם term מופיע עם אותיות קטנות ואם אותיות גדולות.

**public void mergePost(String smallOrBig) {**

שלב 2:

לאחר סיום שלב הפרסינג על כל הקבצים בקורפוס, הפונקצייה מקבלת את שם התיקייה (**smallOrBig**) ומאחדת את כל הקבצים בתיקייה זו.

האיחוד מתבצע באופן הבא:

כל שתי קבצים מתאחדים לקובץ אחד בצורה מקבילית. כלומר אם ישנם 8 קבצים בתוך התיקייה, אז כל זוג יתאחד לקובץ חדש וממויין בסדר אלפביתי. נדגים:

ישנם 8 קבצים: 0.txt, 1.txt, 2.txt, 3.txt, 4.txt, 5.txt, 6.txt, 7.txt

0.txt, 1.txt יתאחדו ל mergedPosting0.txt

2.txt, 3.txt יתאחדו ל mergedPosting1.txt

4.txt, 5.txt יתאחדו ל mergedPosting2.txt

6.txt, 7.txt יתאחדו ל mergedPosting3.txt

לאחר מכן:

mergedPosting0.txt, mergedPosting1.txt יתאחדו ל mergedPosting4.txt

mergedPosting2.txt, mergedPosting3.txt יתאחדו ל mergedPosting5.txt

וכך הלאה, עד להגעה לקובץ מאוחד סופי וממויין (אחד בתיקיית big ואחד בתיקיית small)

כל האיחודים הנ״ל יתרחשו בצורה מקבילית כאמור, באמצעות BufferReader ו BufferWriter אשר יכולים לעבוד במקביל, כלומר מבצעים I/O במקביל. נעשה זאת באמצעות מחלקה פרטית WriteToMergePost שמממשת את מחלקת Runnable , כלומר מממשת את פונקציית run() ומופעלת כ thread.

**public void mergeBigWithSmall(){**

שלב 3:

בשלב זה, לאחר שקיים קובץ אחד מאוחד בתיקיית big וקובץ אחד מאוחד בתיקיית small, נאחד ביניהם לקובץ posting אחד גדול. בנוסף, בשלב זה אנו מבחינים ב terms שכתובים עם אותיות גדולות וגם עם אותיות קטנות, וממירים אותם לאותיות קטנות. אם אכן קרה מקרה שכזה, ישתנה ערכו של המשתנה 'thereIsNoProblemWithBigLetters' ל Fasle וילקח בחשבון בפונקציית כתיבת המילון (writeDictionary).

**public void writeToFinalPosting(){**

שלב 4:

בשלב זה, בהינתן קובץ posting אחד גדול עבור כל ה terms בקורפוס, נפצל אותם לקבצים כפי שהוסבר בבנאי של indexer , כך שכל bufferWriter כותב לקובץ שלו כפי שהוסבר בבנאי של indexer.

**public void writeDictionary (){**

שלב 5:

בשלב זה, אם thereIsNoProblemWithBigLetters==false אז קיימים terms ב 'numberOfDocsPerTerm' וב 'frequentOfTermInCorpus’ אשר כתובים גם עם אותיות גדולות וגם עם אותיות קטנות. נאחד ביניהם ונעדכן את הספירה בכל מילון. לאחר מכן, נמיין את המילונים בסדר אלפביתי באמצעות treeMap כדי לכתוב את ה terms בקלות יותר למילון.

## **מחלקת ה-Stermmer:**

תפקיד המחלקה לבצע stemming בעזרת Porter’s stemmer. השתמשנו במחלקה המצורפת בהנחיות העבודה על מנת לממש מחלקה זו בעבודתנו.

## **מחלקת ה-Term:**

תפקיד המחלקה הוא בניית אובייקט המכיל שני שדות , האחד שם ה-term והשני tf. באובייקט זה אנו משתמשים בעת הצגת המילון ,כאשר אנו מאכלסים את ה- tableView באובייקטים מסוג זה וכך למעשה הערכים מוצגים בצורה אסטתית ונגישה למשתמש.

## **מחלקת ה-MainViewController:**

מחלקה זו מהווה controller עבור המסך הראשי אשר מוצג למשתמש. מחלקה זו תפעיל פונקציות באמצעות ActionEvent שיגיע כטריגר לפעולת המשתמש. מחלקה זו מכילה קישור ישיר לפקדים אשר נמצאים בחלון הראשי שמוצג למשתמש.

**public void loadDicToMemory(ActionEvent actionEvent) throws IOException**

תפקיד הפונקציה הוא טעינת המילון שנמצא בדיסק לזכרון .בעת לחיצה על כפתור “Load dictionary to memory” הפונקציה תבדוק תחילה האם ערך השדה “path to save” הוא לא ריק, במידה והוא ריק הפונקציה תקפיץ הודעת שגיאה בהתאם ותבצע return. במידה וקיים כבר מילון אשר טעון לזכרון הפונקציה תקפיץ הודעת שגיאה בהתאם ותבצע return. הפונקציה בהתאם לנתיב אשר הוזן ובהתאם לכפתור ה- checkBox (האם לבצע stemming ) תשאב מן הדיסק את המילון המתאים לתוך treeMap בזכרון. ותיצור עבור ה-main אובייקט Indexer חדש שבתוכו המילון treeMapForfrequentOfTermInCorpus יצביע למילון אשר שאבנו מן הדיסק.

**public void startBuild(ActionEvent actionEvent)**

תפקיד הפונקציה הוא ניתוב השדות שהתקבלו כקלט מן המשתמש(נתיב corpus ,נתיב לשמירת קבצי ה-posting ומשתנה בוליאני האם לבצע stemming ) לטובת ביצוע readFile, Parser, Indexer ל-corpus באמצעות מחלקת Main . לבסוף הפונקצה מעדכנת בתוצאות יצירת מנוע החיפוש באמצעות שדות יעודיים אשר נמצאים במסך הראשי שמוצג למשתמש ( כמות המסמכים שאוחזרו, כמות ה-terms שאוחזרו, סך הזמן ליצירת מנוע החיפוש, ומילון שפות אשר הופיעו במסמכים).

\*\* הפונקציה מעדכנת את ה-comboBox של השפות בעזרת האובייקט ObservabelArrayList .

**public void browse(ActionEvent actionEvent )**

### תפקיד הפונקצייה הוא לאפשר למשתמש לבחור תקייה לקבל נתיב ה-corpus and stop word באמצעות האובייקט DirectoryChooser.

**public void browse2(ActionEvent actionEvent )**

### תפקיד הפונקצייה הוא לאפשר למשתמש לבחור תקייה לשמירת קבצי ה-posting באמצעות האובייקט DirectoryChooser.

**public void reset(ActionEvent actionEvent )**

תפקיד הפונקציה הוא לאפשר מחיקה רקורסיבית של כלל הקבצים והתקיות (קבצי posting והמילון ) אשר נשמרו בדיסק ,כמו כן מבצע איפוס לזכרון הראשי בתכנית. הנתיב יהיה הנתיב שהוזן לשמירת קבצי ה-posting.

###### **public void showDic(ActionEvent actionEvent) throws IOException**

תפקיד הפונקציה הוא להציג למשתמש את המילון בצורה ממויינת בסדר עולה. הפונקציה יוצרת stage חדש שיוצג בפני המשתמש . במידה והמילון קיים כבר בזיכרון נפעיל את הפונקציה showDic במחלקה ShowDicController ונשלח כפרמטר את main.indexer. אחרת נטען את את המילון ל-TreeMap בזכרון (באופן זהה לפעולה שביצענו בפונקציית ה- loadDicToMemory (וניצור אובייקט Indexer זמני שאותו נשלח לפונקציה ShowDicController.לבסוף נציג את ה-stage שיצרנו.

## **מחלקת ה-ShowDicController:**

תפקיד המחלקה הוא להפעיל Stage חדש שיציג בעזרת TableView את המילון שיצרנו.

**public void showDic(Indexer indexer) throws IOException**

### הפונקציה מקבלת כקלט את ה-indexer ,ומאתחלת את שתי העמודות של ה-tableView להיות הערכים אשר נמצאים בתוך indexer.treeMapForfrequentOfTermInCorpus שהם הערכים אותם אנו רוצים להציג למשתמש. בפונקציה זו אנו עוברים על כל ה-treeMap ולמעשה יוצרים אובייקט חדש בשם Term אשר תפקידו הוא בהכנסת הערכים ל-tableView.

## **מחלקת ה-Main:**

מחלקת ה- Main תפקידה להיות מחקלת ה-Model של העבודה. המחלקה מכילה שלושה משתני מחלקה ReadFile,Indexer,Parser.

**public void start(Stage primaryStage) throws Exception**

תפקיד הפונקציה הוא להיות הפונקציה הראשונה אשר מופעלת בעת תחילת התוכנית, הפונקציה טוענת את ה-primaryStage ומציגה אותו בפני המשתמש.

**public void startBuild(boolean isStemming,String pathOfCorpusAndStopWord , String postingAndDictionary)**

הפונקציה הראשית בעבודה, פונקציה זו מופעלת דרך מחלקת ה-MainViewController. מקבלת כקלט שלושה פמטרים מן המשתמש – נתיב בו נמצא ה-corpus and stop word , נתיב לשמירת קבצי ה-posting ומשתנה בוליאני האם צריך לבצע stemming או לא. הפונקציה יוצרת תקייה חדשה בנתיב לשמירת קבצי ה-posting בהתאם למשתנה הבוליאני isStemming וזאת על מנת למנוע דריסה של אחזור מידע עם/בלי stemming בהתאם להנחיות העבודה. הפונקציה מאתחלת את שלושת משתני המחלקה – ReadFile,Indexer,Parser. בתחילה הפונקציה יוצרת את מילון הערים והשפות ע"י הפעלת פונקציות בתוך המשתנה המחלקתי ReadFile ( מילונים אלו נדרשו כחלק מהנחיות העבודה ).

אנחנו מבצעים את הפעולות הבאות באמצעות לולאת for עבור כל קבוצת מסמכים ) 50 קבצים( מתוך ה-corpus :

1. קריאת 50 קבצים לזכרון דרך הפעלת פונקציית read במחלקת ה-ReadFile.
2. ביצוע תהליך פירסור בעבור 50 הקבצים אותם טענו לזכרון באמצעות פונקציית startParsing50Files במחלקת ה-Parser.
3. הפעלת שני Threads לטובת תכנות במקבילי –

* כתיבה לדיסק את הנתונים הבאים בעבור כל מסמך מקבוצת המסמכים - שם מסמך, עיר , שפה, וכותרת המסמך.
* כתיבה לדיסק את הנתונים הבאים בעבור כל מסמך מקבוצת המסמכים - שם המסמך , מספר ההופעות המקסימלי של term במסמך, כמות המילים השונות במסמך.

1. ביצוע תהליך אינדוקס בעבור 50 הקבצים אותם טענו לזכרון וביצענו עליהם תהליך פירסור , זאת נעשה ע"י שימוש בפונקצייה index50Files שנמצאת במחלקת ה-Indexer.

בסיום מעבר וביצוע ReadFile,Parser,Indexer בעבור כלל הקבצים ב-corpus אנו מבצעים כתיבה של ה-TreeMap של הערים לדיסק אשר כולל את שם העיר, שם המסמך ,מספר מופעים של העיר במסמך.

בנוסף אנו מבצעים פעולות Marge (יוסבר בהמשך על פירוט יצירת הקבציים ההפכיים ) לקבצי ה-posting הזמניים שיצרנו עד לכדי קובץ אחד שממנו אנו פועלים לחלוקה לקבצי לקבצי ה-posting הסופים (הסבר מפורט על סוגי הקבצים יופיע בהמשך בפירוט קבצי ה-posting ). יתר על כן, אנו מבצעים כתיבה של המילון לדיסק (יוסבר בהמשך על פירוט יצירת הקבצים ההופכיים) , כלל הפעולות הללו מבוצעות באמצעת מחלקת ה- Indexer.

## **זמני ריצה ומגבלות זכרון:**

בעבודה זו שמנו דגש על יעילות וזמני ריצה תוך התמודדות עם מגבלת בזיכרון של המחשב היות ומאגר המסמכים מכיל מידע רב . דגשים אלו באו לידי ביטוי באופן הבא :

* קריאת 50 מסמכים בכל איטרציה במחלקת ה-ReadFile ולאחר מכן ביצוע פרסור(מחלקת ה-Parse ) ואינדוקס (מחלקת ה-Indexer) לקבצי פוסטינג זמניים. פעולה זו איפשרה לנו להתמודד עם מגבלות הזכרון של המחשב היות וקראנו בסך הכל 50 קבצים בכל איטרציה מתוך 1816 קבצים.
* JSUP – להפרדת תגיות במסמך איפשר זמני ריצה טובים יותר.
* בעבודה זו נמנענו מיצירת אובייקטים חדשים/מחלקות חדשות(שלא נדרשו), למעט אובייקטי מבני נתונים ואובייקטם פרימטיבים. הימנעות מיצירת אובייקטים חדשים תרמה רבות לשיפור זמני הריצה .
* תכנות מקבילי לטובת ניצול מירבי של משאבים – בחרנו להשתמש ב-Threads בחלק בו כתבנו את קבצי ה-posting הזמניים ובחלק בו ביצענו merging לקבצים אלו. תכנות מקבילי זה איפשר לנו לייעל את תהליך הכתיבה לקבצים באופן מקסימלי .
* בחירת מבני נתונים – בבחירת מבנה הנתונים התייחסנו למספר פרמרטרים: כמות המידע שצריך להחזיק , וסוגי הפעולות שצריך לבצע עבור מבנה הנתונים. לאורך רוב העבודה הבחירה המתאימה ביוצר הייתה מילון (HashMap) בו קיים קשר לוגי של מפתח וערך עבור כל מפתח, גודלו דינאצי ואין צורך לקבוע אותו בעת היצירה ובנוסף זמן ההוספה הממוצע של אובייקט הינו O(1) וכנל לזמן החיפוש. בנוסף על מנת לכתוב לקבצי ה-posing בצורה ממויינת השתמשנו במבנה הנתונים TreeMap שמאפשר מיון של ערכים בזמני ריצה יעילים.
* שימוש ב StringBuilder כאשר יש לבצע שרשור - StringBuilder הינו בעל גודל דינאמי, בניגוד לstring אשר בכל שרשור בעצם מקצה מקום מחדש על פי הגודל החדש שדרש. לכן בכל מקום בו נדרש שרשור של תווים או מחרוזות השתמשנו באובייקט זה.
* שימוש ב-BufferWriter , כאשר אנו מעוניינים לכתוב לדיסק נשתמש באובייקט זה אשר "אוגר" את השורות אותן אנו מעוניינים לכתוב לדיסק ויכתוב אותן בגישה אחת בלבד לדיסק , בניגוד לאובייקטי Writer אחרים כמו FileWriter או PrintWriter אשר כותב אופן ישיר לדיסק וכך מבוצעות גישות רבות לדיסק דבר אשר מעיד על זמני ריצה גבוהים.

## **קבצי ה-posting :**

את קבצי ה-posting שמרנו באופן הבא :

* 26 קבצים בעבור כל אות בשפה האנגלית a-z.
* 10 קבצים בעבור terms אשר מתחילים בספרות 0-9.
* קובץ נוסף בעבור terms אשר מתחילים ב- $.
* קובץ עבור הערים אשר מופיות ב-corpus.

שמרנו את קבצי ה-posting בצורה זו מהסיבה שכאשר נתבקש לאחזר שאילתה מסויימת נוכל באמצעות התו הראשון של כלterm מהשאילתה להגיע בגישה ישירה ומהירה למסמך הפוסטינג הרלוונטי.וכך למעשה לא נצטרך לשמור במילון את שם קובץ ה-posting בעבור כל term אשר קיים במילון. וכן למעשה שמירת הנתונים וקריאתם תתבצע בזמן ריצה המימלי.

כל קובץ posting יכיל את המידע הבא עבור כל ביטוי :



ביטוי

שם המסמך

מספר הופעות הביטוי במסמך

השורה הראשונה במסמך בה הופיע הביטוי

סימן הפרדה בין מסמכים

מסמך

מסמך

עבור קובץ ה-posting של הערים , הקובץ יכיל את המידע הבא בעבור כל ביטוי:

מסמך

מסמך

מסמך

מסמך



שם המסמך

מיקום 2 במסמך

מיקום 1 במסמך

סימן הפרדה בין מסמכים

\*\*אנו נוכל לגשת בגישה ישירה לעיר , מהסיבה שהעיר נמצאת בשורה זהה ביחס למיקום העיר במילון שמתוחזק בזיכרון.

## **הסבר על אופן שמירת ביטוי בתוך קובץ posting :**

כל term יהווה שורה חדשה בקובץ ה-posting , עבור כל term בתחילת השורה ניתן למצוא את הביטוי. לאחריו מופרד בפסיק ניתן לראות את פרטי המסמך הראשון בו ה-term מופיע . בעבור כל מסמך נשמור את שם המסמך, מספר הופעות ה-term במסמך , והשורה הראשונה במסמך בה נתקלנו ב-term ( שמרנו מידע זה כיוון שבעתיד ובמידה ונרצה לאחזר מסמכים ככל שה-term יופיע בתחילת הטקסט כלומר חלק יהיה מפיסקת הפתיחה ניתן עדיפות למסמך זה על פני שאר המסמכים). את ההפרדה בין מסמך למסמך ביצענו באמצעות ~.

## **בחירת גודל קבוצת המסמכים :**

גודל קבוצת המסמכים שלנו הוא 50 קבצים . בחרנו את גודל קבוצת המסמכים לאחר הרצות רבות של גדלים אשר גדולים משמעותית מ-50 וכך למעשה נתקלנו בבעיות זכרון במהלך זמן הריצה – מהסיבה שהזכרון אינו מאפשר טעינה של קבוצת מסמכים כה גדולה .מנגד בעת הרצת גודל קבוצת מסמכים הקטנה משמעותית מ-50 קיבלנו זמני ריצה ארוכים בצורה משמעותית- מהסיבה שעבור כל קבוצת מסמכים אנחנו יוצרים בתחילה שני קבצי posting האחד ל-terms אשר מתחילים באותיות קטנות ואילו השני ל-terms אשר מתחילים באותיות גדולות ושאר התווים והמספרים וכך ישנה כתיבה לכמות רבה מאוד של קבצים וביצוע מיזוג לכמות גדולה של קבצים.

כך שהבחירה בגודל קבוצת המסמכים השווה ל-50 הביא לנו את הפתרון האופטימלי מבחינת זמני הריצה והזכרון.

## **פרטי האינפורמציה הנוספים ששמרנו :**

## שמירת כותרת בעבור כל מסמך - נתחשב בשלב מתקדם יותר כאשר נרצה לאחזר מסמכים, ניתן עדיפות גבוהה יותר למסמכים אשר הביטוי מופיע בכותרת המסמך.

1. שמירת שפת המסמך – נתחשב בשלב מתקדם יותר במידה ונרצה לאחזר מסמכים בשפה מסויימת.
2. שמירת שורה ראשונה שבה הופיע term במסמך. נתחשב בנתון זה בשלב מתקדם יותר בעבודה כאשר נרצה לאחזר מסמכים, ניתן עדיפות גבוהה יותר למסמכים שהביטוי הופיע מוקדם יותר מהסיבה שאם הביטוי הופיע בתחילת הטקסט כלומר בפיסקת הפתיחה תהיה לו חשיבות רבה במהלך המסמך.

## **שני חוקים נוספים למחלקת ה-Parser:**

## טיפול במשקלים:

## מספר כפי שהוגדר בהנחיות העבודה אשר מצורף אליו Kilogram,kg,kilogram,Kg ישמר באופן הבא number kg.

## לדוגמא: : 10 kilogram -> 10 kg, 120 Kg -> 120 kg .

## מספר עפי שהוגדר בהנחיות העבודה אשר מצורף אליו Gram,gram,g ישמר באופן הבא number kg.

## לדוגמא : 1000 gram-> 1 kg , 500 g -> 0.5 kg.

1. טיפול בטווחים ( שדרוג לחוק 2.ז'):

* כל הביטויים שהופיעו כך : between number and number ישמרו באופן הבא : number-number.
* טווחי זמן אשר הופיעו כך: 6-7 July ישמרו באופן הבא : .07-07 ,07-06 ,6-7

**חלק 2 :**

1. ביטויים במאגר ללא stemming: 1,147,825
2. ביטויים במאגר עם stemming: 1,028,495
3. ביטויים שהם מספרים: 240,449
4. מספר המדינות השונות במאגר : 126
5. סך הכל ישנן 465 ערים. 340 מהם אינם ערי בירה
6. שם המסמך עם הכי הרבה מופעים של עיר בודדת: FBIS3-60342

שם העיר: MOSCOW מיקומי העיר במסמך:

18,35,62,89,167,669,694,726,756,757,858,3394,4267,6290,7232,7886,8078

8123,8230,8470,8771,8776,9771,9779,10083,10129,10182,10191,10404, 10452,10495,10553,10577,10667,11301,11326,11596,11714,11725,11732, 11952,12047,12481,12524,12649,12954,13060,13840,14121,14140,14199, 14333,14488,14822,14971,15145,15148,15186,15355,16807,17575,17860, 17927,18211,18293,18515,18536,18808,18857,19069,19249,19408,19428, 19747,19772,19957,19974,19979,20137,20171,20408,20624,20634,20933, 20957,20976,21021,21031,21038,21048,21058,21063,21084,21089,21108, 21113,21839,22157,22387,28491,29378,30134,32415,36768,42812,43519, 43895,44144,44377,44542,44913,44949,45014,45625,45915,46014,46460, 46473,46484,46553,46576,46600,46618,46650,46708,46753,46783,46799,

46807,46910,47387,47399,47456,47603,47636,47637,47640,47654,47691,

47799,47871,47892,47957,47969,47986,48023,48040,48077,48219,48336,

48537,48575,48605,48623,48638,48640,48665,48678,49058,49243,49248,

49315,49369,49390,49407,49427,49441,49475,49480,49535,49540,49547,

49628,49651,49671,49687,49705,49758,49926,49946,50003,50027,50067,

50235,50333,50341,50512,50597,50637,50665,50709,50727,50827,50913,

50921,50969,51288,51362,51369,51554,51689,51798,51865,51892,52075,

52107,52139,52163,52191,52296,52305,52358,52469,52535,52576,52602,

52698,55289,58634,59203,64040,65257,66126,69758,69822,69837,69862,

69927,69970,70150,70219,70275,70320,70427,70654,70690,70732,70858,

70921,70999,71054,71161,71313,71406,71850,72236,79364,80037,89909,

91816,99664,100129,100779,100824,101167,101379,101392,101395,

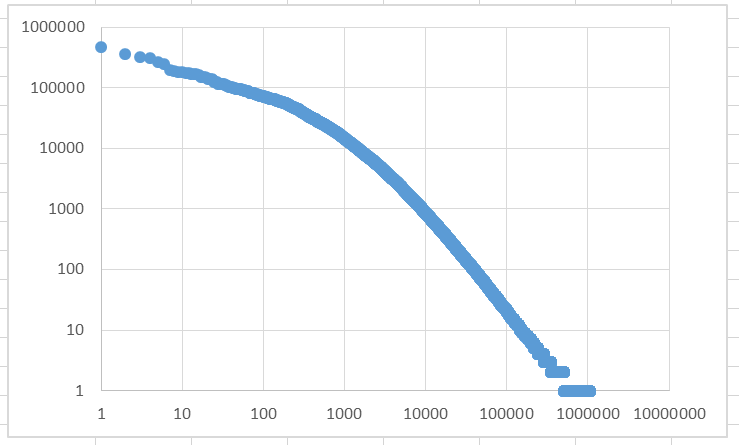
101404,101406,101409,101412,101420,101424,101605,101795,101966,

102157,102534,102919,102944,103383,103554,103606,103711,103870,

104293,104523,104716,104797.

|  |  |
| --- | --- |
| **המילים הכי נפוצות בסדר יורד** | **המילים הכי פחות נפוצות** |
| mr | zzzt |
| Year | zzz’s |
| Cent | zync |
| government | zylonite |
| pounds | zykzym |
| people | zykzyk |
| Years | zyk |
| market | zygotic |
| time | zygote |
| State | zigomatic |

1. ניתן לראות כי העקומה אשר קיבלנו אכן דוגמה ל-Zipf’s Law



ביטוי

שכיחות במאגר

1. פירוט ה terms של מסמך **FBIS3-3366**

03-19,2

1.994K ,1

CPPCC,4

TYPE:BFN,1

XINHUA,1

adopted,2

amended,3

article,1

beijing,1

charter,3

chinese,5

committee,5

conference,4

consultative,4

decided,1

effect,1

eighth,3

language,1

national,4

people's,4

political,4

proposed,1

resolution,1

session,3

standing,1

text,1

today,1

\*ניתן לראות כי רוב הערכים מוצגים באותיות קטנות. במסמך עצמו, המילה “beijing” לדוגמא מופיעה באופן הבא :”Beijing” . אבל מהסיבה שבמסמך מאוחר יותר המילה מופיע כ “beijing” , אז המילה בטקסט הנוכחי הומרה לאותיות קטנות ולכן מוצגת באופן זה.

ההנחה שעבדנו על פיה היא שאת הביטויים אנו מציגים לאחר שביצענו קריאה, פרסור ואינדוקס בעבור ול ה-corpus.

10.

|  |  |  |
| --- | --- | --- |
| **City posting** | **Terms posting** | **גודל הPosting- (ב-KB)** |
| 9420.8 | 1279262.72 | With stemming |
| 9420.8 | 1363148.8 | No stemming |