**עבודת הגשה: קונקורדנציה  
ע"י שי אוהב ציון (318683479) ונתנאל נתי זפקו (302309513)**

בקובץ הקוד עצמו, על יד כל פונקציה רשמנו בהערה את יעילות הקוד, בהילקח בחשבון הפונקציות שמופעלות מאותה הפונקציה בעצמה. כך יהיה קל להבין את יעילות הקוד כולו, בנפרד מהניתוח שמבוצע בקוד הזה.

את הקונקורדנציה בחרנו לבנות באופן הבא:  
  
עבור כל מילה נשמור את מספרי השורות שבה היא מופיעה בעזרת רשימה מקושרת (Linked List). ברשימה מקושרת זו נשמור מצביע לאיבר הראשון, ומצביע לאיבר האחרון. רשימה מקושרת שכזו הופכת את תהליך ההכנסה ליעיל (בדיוק O(1)) ומאפשר לנו בתהליך ההדפסה לעבור עבור כל מילה על כל הפעמים שהיא מופיעה. בגלל האופי של התרגיל, והסדר הכנסה הוא לפי השורות, באופן טבעי המספרים שנכניס אל תוך הרשימה המקושרת יגדלו או יישארו זהים, ולכן אין לנו צורך לבדוק בהכנסה שגודל מספר השורה גדול מהאיבר הקודם שהכנסנו אל הרשימה.

בגלל האופי של התרגיל, לא היה לנו צורך בפונקציית delete או find, אם כי אנחנו יודעים שאילו היינו צריכים למצוא, או למחוק, יעילות התהליך הייתה O(n). ההכנסה היא החלק החשוב ביותר עבורנו, ולכן בחרנו ברשימה שבה יש מצביע לאיבר האחרון, ובתהליך הקריאה, בכל מקרה נרצה לעבור על כל האיברים לפי הסדר, ולכן רשימה מקושרת הייתה הבחירה הטבעית.

לאחר שבנינו רשימות מקושרות לשמירת השורות והגדרנו "מילה", רצינו מבנה נתונים שיוכל לשמור את כל המילים. לשם כך בחרנו בעץ חיפוש בינארי (Binary Search Tree). עץ כזה שאינו מאפשר "כפילויות" ובו תהליך ההכנסה הינו בממוצע O(lg(n)) כאשר n מספר המילים, אך במקרה הגרוע ביותר O(n) אילו כל המילים מסודרות לקסיקוגרפית (לא משנה קטן לגדול, או גדול לקטן, זה ישפיע רק על ה**כיוון** של ה"עץ" שייווצר כמסודר בשורה ארוכה). במידה ומכניסים מילה קטנה לקסיקוגרפית – היא תהיה בת שמאלית, ואם אחרי – בת שמאלית, ובמידה ומצאנו מילה שכבר מופיעה בעץ, אין לנו צורך להוסיף אותה לעץ, אלא להוסיף לרשימה המקושרת עוד הופעה של המילה הרצויה, הנ"ל כמובן פועל בצורה רקורסיבית, במידה והבן כבר "תפוס".

שוב, לא היה לנו צורך בהוצאה / חיפוש בעץ, ולכן היה לנו קל להניח שמרבית המקרים שבהם הקוד יפעיל יהיו המקרה הממוצע, ולכן גם בהם, כמו בעץ AVL או עץ אדום-שחור, היעילות של הוצאה וחיפוש במקרה הממוצע הנם O(lg(n)), אך היעילות של המקרה הגרוע שבמבנים הללו נשארת O(lg(n)) בעוד בעץ BST שבו בחרנו היעילות היא O(n) במקרה הגרוע ביותר.

בהרצת הקוד כולו, עבור קובץ שבו יש n מילים, לכל מילה ההכנסה היא O(lg(n)), לכן הכנסת כל המילים סה"כ הינה O(n\*lg(n)) בממוצע. עם-זאת, בקובץ שמסודר לקסיקוגרפית, יעילות ההכנסה בקוד תהיה O(n2), ובקובץ שבו ישנה רק מילה אחת שחוזרת על עצמה בלבד, שוב ושוב (לא משנה אם בשורות שונות, או לא) יעילות ההכנסה תהיה O(n).  
בתהליך ההדפסה, עלינו לעבור על בדיוק n איברים (**כל מילה** הינה איבר בעץ BST, **או** לחילופין, איבר בשרשרת Linked List) ולכן תהליך ההדפסה הנו O(n).

לכן בפרויקט שלנו, יעילות הקוד **כולו** הנה O(n\*lg(n)).

כפי שהבטחנו, הנה תפוח אדמה מאוד מגניב.

