

אחזור מידע

מבוא

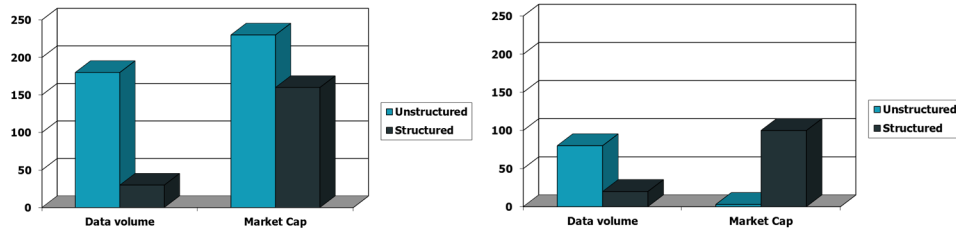
נניח שיש לנו אוסף גדול מאד של מסמכים השמורים על גבי מחשב אחד או יותר (נהוג לכנותם בשם קורפוס). כמו כן, יש נושא לגבי אני מעוניין במידע (צורך מסוים). חלק מהמסמכים בקורפוס מכילים את המידע בו אני מעוניין, כלומר אותם מסמכים הם בעלי עניין, או רלוונטיים, עבורי. אבל איך אני יכול לקבל את המסמכים הללו, או במילים אחרות, איך אני יודע מי מהמסמכים רלוונטי עבורי. בכדי לקבל את המסמכים הרלוונטיים אני צריכים לבטא את הצורך שלנו בצורה של שאילתא, שבעזרתה אנו מתשאלים את אותו מחשב/ים, שבהתאם לכך יודעים לחפש ולהביא את אותם המסמכים.

האופן בו השאילתא נכתבת תלוי באם המסמכים הם בעלי מבנה מוגדר או לא. דוגמא למסמך בעל מבנה מוגדר היא טבלה. בטבלה אנו יודעים בברור מה המשמעות של כל עמודה, ולעיתים, מה המשמעות של כל תא בטבלה. לדוגמא, בבסיס נתונים טבלאי, הנתונים נשמרים בטבלאות. אנו יודעים מה המשמעות של כל רשומה, ואף כל שדה ברשומה. כמו כן, עומדים לרשותנו כלים חזקים, כמו SQL, שמאפשרים לנו לתשאל את בסיס הנתונים ולקבל את המידע מהטבלאות השונות.

למסמך בעל מבנה לא מוגדר, אין, כמו שנרמז, מבנה מוגדר, ואנו לא יכולים לדעת במה ואיך עוסק המסמך. במקרים כאלו אנו לא יכולים לבצע שאילתות מורכבות כמו בבסיסי נתונים. אבל אנו יכולים לשאול אם ביטויים מסוימים נמצאים במסמך.

אחזור מידע (Information Retrieval – IR) הוא מציאת חומר (בדרך כלל מסמכים) בעלי מבנה לא מוגדר (בדרך כלל טקסט) המספק מידע נדרש מתוך אוספים גדולים (בדרך כלל מאוחסנים במחשבים).

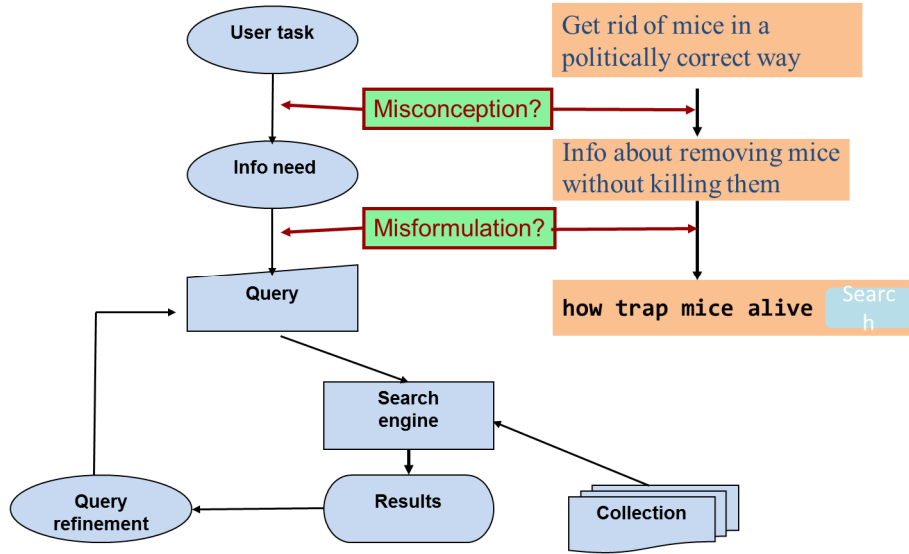
כאשר מדברים על אחזור מידע, הדבר הראשון העולה לראש הוא חיפוש באינטרנט, אבל יש הרבה מקרים אחרים הקשורים לאחזור מידע: (1) חיפוש ב-e-mails, (2) חיפוש במחשבים, (3) חיפוש במערכות אירגוניות ו-(4) חיפוש במסמכים משפטיים.



הגרף מימין מראה שכבר בשנת 1996 כמות המסמכים בעלי המבנה הלא מוגדר הייתה גדולה פי כמה מכמות המסמכים בעלי המבנה המוגדר. ולמרות זאת, מעט מאד חברות התעסקו בניית מסמכים בעלי מבנה לא מוגדר. בשנת 2009 לעומת זאת (הגרף משמאל), כמות החברות שהתעסקו בניית מסמכים בעלי מבנה לא מוגדר גדלה משמעותית ואף עברה את כמות החברות שהתעסקו עם מסמכים מובנים (אנו רואים פיתוח של הרבה מנועי חיפוש למיניהם).

במציאות, גם כאשר אנו מדברים על מסמכים בעלי מבנה לא מוגדר, הם לא באמת כאלו. לדוגמא, למסמך הזה יש כותרת, שיכולה לרמז על תוכן המסמך. מאמרים אקדמאים יש להם מבנה מסוים (כותרת, אבסטרקט, מילות מפתח, ביבליוגרפיה וכו'). אפשר להשתמש במידע החצי מובנה הזה לצורך שליפת מסמכים (למשל, הבא את המסמכים שבכותרת שלהם מופיעות המילים....).

אנו עובדים עם אוסף של מסמכים, כלומר קבוצה של מסמכים, בשלב זה קבוצה קבועה שלא משתנה. לאוסף מסמכים זה, כאמור, נהוג לקרוא קורפוס. המטרה שלנו היא אחזור מסמכים המכילים מידע הרלוונטי למשתמש, והעוזרים למשתמש בביצוע משימה.



תהליך אחזור המידע מתחיל עם בעיה מסוימת של המשתמש, שבעקבותיה נוצרת דרישה מסוימת של המשתמש למידע (צורך). הדרישה הזאת מתורגמת לשאלתא, אשר מועברת למנוע החיפוש. מנוע החיפוש בודק אלו מהמסמכים עונים לשאלתא, ומחזיר אותם למשתמש. המשתמש בודקת את התשובה המתקבלת, ובמידה והיא לא עונה על הצורך (נגיד, מס' רב של מסמכים לא רלוונטיים), המשתמש חוזר ומנסח את השאלתא שלו מחדש, בצורה מדויקת יותר.

בכדי לדעת עד כמה טוב הפתרון שלנו, כלומר עד כמה קבוצת המסמכים שהחזרנו עונה לצרכי המשתמש אנו נגדיר שני מדדים: בהינתן שאלתא, יהי R קבוצת המסמכים הרלוונטיים ו-A קבוצת המסמכים שאוחזרו. המדד הראשון הוא דיוק - קבוצת המסמכים המאוחזרים הרלוונטיים לצורכי המידע של המשתמש מתוך כלל המסמכים שאוחזרו $\frac{R \cap A}{A}$ - המדד השני הוא כיסוי - קבוצת המסמכים המאוחזרים הרלוונטיים לצורכי המידע של המשתמש מתוך כלל המסמכים הרלוונטיים למשתמש $\frac{R \cap A}{R}$.

מטריצת מפגשים לביטויים במסמכים

נניח שאנו מעוניינים לדעת אלו מהמחזות של שייקספיר (הקורפוס שלנו) מכילים את המילים Brutus וגם Caesar אבל לא את Calpurnia. נתחיל עם פתרון נאיבי. בכדי להגיע לפתרון אפשר לחפש (למשל באמצעות grep), בכל המחזות, את השורות שמכילות Brutus ו/או Caesar, ומהתוצאה להסיר את כל השורות שמכילות את המילה Calpurnia. לצורת צורת חיפש מספר חסרונות: (1) איטי - כל פעם שנבצע שאלתא אנו נצטרך לסרוק את הקבצים מחדש על מנת לדעת אם ביטוי מסוים נמצא בהם או לא (במקרה של קורפוס מאד גדול לא פתרון מעשי), (2) שיטה זו לא נותנת פתרון ל-"לא את Calpurnia", (3) אופרטורים אחרים (למשל המילה Romans נמצאת ליד המילה countrymen) אינם ניתנים למימוש בשיטה זו, ו-(4) אי אפשר לדרג את הפתרונות.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

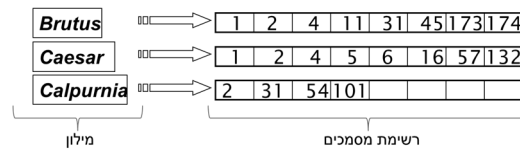
אנו יכולים להשתמש במטריצת מפגשים בכדי לציין אלו ביטויים נמצאים בכל מסמך. את מטריצת המפגשים אנו בנוים מראש. לכל מסמך יש ווקטור מפגשים המתאים לו, בו כל ביטוי יכול לקבל את ערך 0 (כשהביטוי לא נמצא במסמך) או 1 (כשהביטוי נמצא במסמך). באופן דומה, לכל ביטוי יש ווקטור מפגשים, בו כל מסמך יכול לקבל את ערך 0 (כשהביטוי לא נמצא במסמך) או 1 (כשהביטוי נמצא במסמך). מטריצת המפגשים אינה שומרת את מס'

הפעמים שכל ביטוי מופיע בכל מסמך, וכן, אנו לא יודעים היכן במסמך (מיקום) נמצא כל ביטוי. בכדי לענות על השאלתא שלנו, אנו לוקחים את הווקטורים של הביטויים Brutus, Caesar והמשלים של Calpurnia (כי אנו רוצים שלא יכיל את Calpurnia), ונבצע עליהם פעולת bitwise AND. כלומר, 110100 and 110111 and $101111 = 100100$. ומכאן שהתשובה היא Antony and Cleopatra ו-Hamlet.

נניח שיש לנו מיליון מסמכים, שבכל אחד מהם כ-1000 מילים, כשבממוצע מילה מורכבת משישה תווים (כולל רווחים ותווי פיסוק), כלומר 6GB של מידע. כמו כן, נניח שיש 500,000 מושגים שונים במסמכים האלו. במקרה כזו, אנו מדברים על מטריצת מפגשים בגודל $500K \times 1M$ של אפסים ואחדות (אם אנו משתמשים בבית אחד עבור כל תא, גודל המטריצה בזיכרון הוא 500GB). ברוב במקרים המטריצות האלו הן ממש דלילות, כלומר, מספר האחדות הוא נמוך משמעותית מסך התאים שבמטריצה (במקרה שלנו $1K \times 1M$, שזה 0.2% מסך התאים במטריצה). אם כך, אנו צריכים למצוא שיטה טובה יותר לייצג את המידע, שתשמור רק את המידע של האחדות.

Inverted Index - אינדקס מסמכים

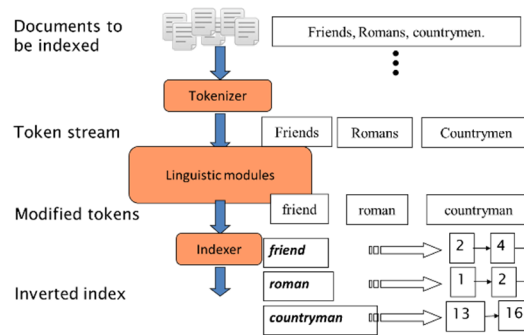
עבור כל מושג, t, אנו צריכים לשמור רשימה של כל המסמכים בהם הוא נמצא. נניח שכל מסמך מיוצג על-ידי מספר סידורי, docID, אנו יכולים לייצר את הרשימה הבאה.



למבנה הזה אנו קוראים Inverted Index, כאשר רשימת הביטויים נקראת מילון (dictionary) ורשימת המסמכים נקראת posting list.

מכוון שאנו רוצים לחסוך במקום / זיכרון, אנו נשתמש ברשימות דינמיות (כלומר, מבנה נתונים שהגודל שלו אינו קבוע, ויכול להשתנות בהתאם לכמות המידע הנשמר).

נתבונן בתהליך הכללי של בניית אינדקסים.



- (1) Tokenization - עבור כל מסמך, אנו חותכים את רצף התווים למילים (tokens). כמו כן, אנו, בין היתר, מורידים סימני פיסוק. (2) Normalization - העברת מילים וביטויים הנכתבים בצורה שונה למצב זהה (למשל USA ו-U.S.A.). (3) Stemming - מציאת השורש של המילים והביטויים השונים (למשל authorize, authorization). (4) Stop words - הסרה (או לא) של מילים נפוצות כגון the, a, to, of.

בכדי ליצור את ה-inverted index אנו (1) חותכים המסמכים למילים, כל מילה עוברת דרך ה-Linguistic modules, כשאת הביטוי המתקבל (term) משייכים למסמך (docID). בשלב הזה אנו מקבלים רשימה של זוגות, <ביטוי, מס' מסמך>. (2) מיון הרשימה. אנו ממיינים את רשימת הזוגות שלנו לפי הביטויים, בסדר עולה. (3) הורדת כפילויות. מכוון שאותו ביטוי יכול להופיע מס' פעמים באותו הטקסט, או במספר מסמכים שונים, אנו מבצעים פעולה של הורדת כפילויות. כתוצאה מפעולה זה אנו מקבלים שני תוצרים. האחד, רשימה של ביטויים ומספר המופעים שלהם באוסף המסמכים שלנו (מס' מופעים באותו מסמך נספרים פעם אחת בלבד), וכן, עבור כל ביטוי, רשימה מקושרת של מספרי המסמכים בהם הוא נמצא (posting list).

ביצוע שאלות בעזרת Inverted Index

נבחן את השאלתא הבאה: Brutus AND Caesar. תחילה, יש לאתר את Brutus בקובץ המילון, ובהתאם לקבל את רשימת הקבצים בהם הוא נמצא. שנית, יש לאתר את Caesar בקובץ המילון, ובהתאם לקבל את רשימת הקבצים בהם הוא נמצא. בכדי לדעת באיזה מסמכים נמצאות המילים Brutus וגם Caesar, נעבור על שתי הרשימות ונחפש מסמכים המשותפים לשני הביטויים. אם ה-posting lists ממוינים, אפשר לעבור על שתיהן במקביל (בדומה ל-marge sort), ואז שסיבוכיות התהליך היא $O(x + y)$, כאשר x הוא מספר האיברים ברשימה הראשונה, ו-y הוא מספר האיברים ברשימה השנייה.

```

INTERSECT( $p_1, p_2$ )
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then ADD(answer,  $\text{docID}(p_1)$ )
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7  else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8      then  $p_1 \leftarrow \text{next}(p_1)$ 
9  else  $p_2 \leftarrow \text{next}(p_2)$ 
10 return answer
    
```

האלגוריתם הנ"ל מתאר את אופן המעבר על שתי הרשימות במקביל. אנו מתחילים עם תשובה ריקה (שורה 1). p_1 ו- p_2 הם מצביעים ל-docID בכל אחת מהרשימות. אם p_1 או p_2 שווים ל-NIL סיימנו. אם לא, אז אנו בודקים $\text{docID}(p_1)$ ו- $\text{docID}(p_2)$, אם כן, הרי שמצאנו מסמך משותף, ואנו נוסף אותו לתשובה שלנו (שורה 4). במקרה זה אנו גם נקדם את p_1 ו- p_2 . אם הם לא שווים, הרי שלא מצאנו מסמך משותף. במקרה הזה אנו לא נקדם את שני המצביעים, אלא רק את המצביע שמצביע ל-docID קטן יותר.

שאלות של OR, כגון Brutus OR Caesar, עובדות באופן דומה. אנו עוברים על שתי הרשימות, ומוסיפים את כל אחד מהאיברים. אם איבר מסוים נמצא בשתי הרשימות, נוסף אותו פעם אחת בלבד.

שאלות בוליאניות

שאלתא בוליאנית היא שאלתא הכוללת בתוכה אופרטורים בוליאניים כגון "וגם", "או" ו-"לא" על מנת לחבר תוצאות של שאלות על ביטויים פשוטים (זאת לא צורת השאלתא במערכות IR חדשות כמו גוגל). לדוגמא: WestLaw, החברה הגדולה ביותר המספקת שרותי חיפוש בתיקים משפטיים שונים, עם בערך 700,000 משתמשים. השרות שלהם התחיל ב-1975. הם מספקים שרות חיפוש מבוסס שאלות בוליאניות, שעדיין נמצא בשימוש ע"י מספר רב של משתמשים, וזאת למרות שכבר ב-1992 הם אפשרו שיטות חיפוש אחרות.

נניח שאנו רוצים לענות על השאלה הבאה:

What is the statute of limitations in cases involving the federal tort claims act?

השאלתא, בפורמט של WestLaw תהיה

LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM

כאשר עומדות ברשותנו מס' אפשרויות נוספות, כגון /3 = within 3 words, /S = in same sentence

השאלתא הזאת אומרת, תביא את כל המסמכים שכוללים מילה המתחילה ב-Limit, וכן את המילים Statute, Action, Federal, Tort ו-Claim. כאשר המילים Limit ו-Statute צריכים להיות במרחק של לא יותר מ-3 מילים במסמך. המילים Federal ו-Tort צריכים להופיע במרחק של לא יותר מ-2 מילים במסמך. והמילים Tort ו-Claim צריכים להופיע במרחק של לא יותר מ-3 מילים במסמך. המילים Action ו-Federal צריכים להופיע באותו המשפט.

האם ניתן להשתמש inverted index (עם שינויים קלים) על מנת לבצע את השאלות הבאות: (1) Brutus AND NOT Caesar. פעולת NOT משמעותה על ה-docID שלא נמצאים ב-posting list. (2) Brutus OR NOT Caesar. האם זמן הסיבוכיות היא עדיין $O(x + y)$? פעולת NOT, במקרה שה-posting list קצר והקורפוס מאד גדול לא ניתנת לביצוע בזמן ריצה לינארי. מה לגבי שאלות בוליאניות כלליות, כגון: Brutus OR Caesar) AND NOT (Antony OR Cleopatra). האם זמן הריצה הוא תמיד לינארי ?

מהו הסדר הטוב ביותר לעיבוד השאלות. נניח שיש שאלות AND של n ביטויים. עבור כל אחד מהביטויים צריך לקבל את רשימת המסמכים בו הוא נמצא, ולבצע AND על כל הרשימות. כיצד נבצע את הפעולה הזאת בצורה הטובה ביותר, כלומר, בזמן ריצה קצר ביותר וכמה שפחות זיכרון. בכדי לבצע כמה שפחות פעולות, יש להתחיל עם הביטויים שיש להם הכי פחות מופעים. בדוגמא שלנו נבצע תחילה Calpurnia AND Brutus, ועל התוצאה המתקבלת נבצע AND Caesar. עבור שאלות שמכילות ביטוי OR, למשל ignoble AND (madding OR crowd) (OR strife), יש לקבל את מספר המופעים של כל ביטוי. להעריך את הגודל של כל תוצאת OR, ע"י סכימת המופעים שלהם. לבצע את השאלות בהתאם לגודל המוערך שהתקבל. זאת הסיבה שאנו שומרים את מספר המופעים (ששווה לאורך ה-posting list) עבור כל ביטוי.

שאלות של ביטויים

אנו מעוניינים לענות על שאלות הכוללות ביטויים כגון "מכללה האקדמית תל-אביב - יפו". למשל, המשפט "אני לומד במכללה האקדמית אשר נמצאת בתל-אביב - יפו", אינה עונה על השאלתא שלנו.

פתרון אפשרי הוא בעזרת Biwords. יש לאנדקס כל שתי מילים סמוכות בטקסט כביטוי. לדוגמא, עבור הטקסט "Friends, Romans, Countrymen" לקבל את ה-Biword-ים הבאים: (1) friends romans ו-(2) romans countrymen. כל אחד מה-Biword-ים היא ביטוי עבור המילון שלנו. באופן זה, אנו יכולים לעשות חיפושים מידיים על כל זוגות המילים בטקסט.

שאלות על ביטויים ארוכים ניתן לבצע ע"י חלוקתם לביטויים קצרים יותר. את הביטוי "Stanford university Palo alto" ניתן לאחזר ע"י שימוש בשאלתא הבוליאנית הבאה: "Stanford university" AND "university Palo" AND "Palo alto". שאלתא מסוג זה עלולה להחזיר תוצאות שגויות. במקרה כזה, אנו צריכים את המסמכים עצמם בכדי לבדוק אם קיבלנו את התשובות הנכונות. חסרונות השיטה: (1) תוצאות שגויות. (2) קובץ אינדקס גדול כתוצאה מהגדלת המילון. (3) אינדקס Biword אינו הפתרון במקרים כגון אלו, אך יכול להוות חלק מפתרון כולל יותר.

ב-positional index, לכל ביטוי, אנו שומרים את מס' הקבצים שמכילים את הביטוי וכן את המיקום של הביטוי בכל קובץ.

```
<term, number of docs containing term;
doc1: position1, position2 ... ;
doc2: position1, position2 ... ;
etc.>
```

```
<be: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, ...>
```

בכדי לבצע שאלות על ביטויים, אני משתמשים באלגוריתם המיוזג באופן רקורסיבי על גבי המסמכים. אולם במקרים כאלו, אנו צריכים להתמודד לא רק עם שווינויים.

נניח שאנו מחפשים את הביטוי "to be or not to be". עבור כל מילה אנו שולפים את רשימת המיקומים שלה בכל אחד מהמסמכים.

```
to: 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
```

```
be: 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...
```

כעת, לאחר זיהוינו מסמכים משותפים, אנו מחפשים סמיכות בין שתי מילים (למשל 16 ו-17, 190 ו-191 ו-433 ו-434). אנו יכולים גם לפסול מסמכים מסוימים על סמך האינדקס הזה, באופן פשוט יחסית. למשל בביטוי שלנו, המילים to ו-be מופיעים פעמיים. המרחק בין to ל-to הוא 5 מילים, וכנ"ל לגבי be. מסמך מס' 2, למשל, מכיל שני מופעים בלבד של המילה be, שהמרחק ביניהם הוא 146 מילים, הרבה מעבר ל-5 המילים שאנו דורשים, ולכן המסמך הזה הוא לא חלק מפתרון השאלתא שלנו.

שימוש באינדקס מיקומים מגדיל באופן משמעותי את שטח האחסון הנדרש (למרות שניתן לכווץ את הקובץ הנ"ל). בכל זאת, למרות חיסרון זה, שימוש באינדקס מיקומים נמצא בשימוש באופן סטנדרטי בגלל הכוח והתועלת המתקבלים בעת ביצוע שאילתות על ביטוי. עבור כל ביטוי אנו צריכים לשמור את כל המיקומים שלו במסמך, בניגוד לאינדיקציה אם הוא נמצא או לא. לפיכך, גודלו של אינדקס מיקומים תלוי בגודלם של המסמכים שלנו. בהנחה שהשכיחות של כל מושג במסמך היא 0.1% מתקבל

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

גודלו של אינדקס מיקומים הוא פי 2 עד פי 4 יותר מאשר אינדקס רגיל. עבור מסמך נתון, גודלו של אינדקס המיקומים מהווה בין 35% ל-50% מגודלו של המסמך המקורי.