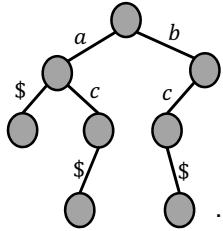


# מילון למחרוזות - Trie

Trie מאפשר חיפוש, הכנסה, הוצאה, ומציאת מינימום (לקסיקוגרפי) של מחרוזות.

המימוש באמצעות עץ. לכל צומת פנימי יש לכל היותר מספר ילדים כגודל האלף-בית + אחד. כל קשת מסומנת בתו. התו \$ (שאינו שייך ל- $\Sigma$ ) מסמן סיום מחרוזת. אנו נתייחס למקרים בהם הגודל של  $\Sigma$  קבוע.



דוגמא: Trie עבור המחרוזות  $ac, a, bc$ , כאשר תו סיום-המחרוזת \$ הוא הקטן ביותר לקסיקוגרפית.

הערה: בכל צומת מוחזק מערך באורך  $|\Sigma| + 1$  של מצביעים לבנים.

כל מחרוזת במבנה מגדירה מסלול מהשורש אל עלה. כל הפעולות מתבצעות ע"י מעקב לאורך המסלול המתאים.

סיבוכיות המימוש:

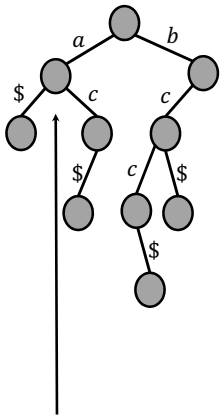
- הזמן הנדרש לביצוע נקבע ע"י אורך המחרוזת  $O(|s|)$  (ולא ע"י מספרן של המחרוזות  $n$ ).
- המקום: לינארי בסכום אורכי המחרוזות במבנה.

# מבני נתונים למחרוזות

## חומר קריאה לשיעור זה

Algorithms on Strings, Trees, and Sequences, Dan Gusfield  
Chapter 5, 7.3, 7.4, 7.17

# מימוש חיפוש, הכנסה והוצאה



הצומת  $v$  עבור הסרת המחרוזת  $ac\$$

חיפוש( $s$ ): נתחיל בשורש.  $i \leftarrow 0$ .

כל עוד  $i \leq |s|$ : אם התו  $i$ -ה במחרוזת הוא התו  $j$ -ה בא"ב, נעקוב אחרי המצביע  $j$ -ה במערך. אם המצביע הזה NULL, נחזיר "לא נמצא". אחרת,  $i \leftarrow i + 1$ . אם  $i > |s|$ , החזר "נמצא".

הכנסה( $s$ ): בדומה לחיפוש, אבל אם ניתקל במצביע NULL במהלך החיפוש, נקצה צומת חדש ונדאג שהמצביע המתאים בצומת הנוכחי יצביע אליו, ונמשיך בחיפוש בצומת החדש.

הוצאה( $s$ ): נחפש את המחרוזת  $s$  ונשמור את הצומת האחרון  $v$  בעל יותר מבחן אחד לאורך מסלול החיפוש ואת האות הבאה של  $s$ . נמחק את תת העץ המתאים של  $s$ .

# מבנה צומת בTrie

הגדרת צומת:



```
#define SIGMA 26;

typedef struct node {
    struct node[ SIGMA + 1 ] *sons ;
} NODE ;

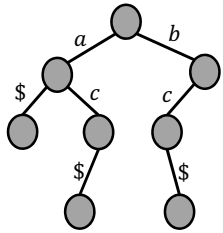
NODE *root ;
```

## מיון מחרוזות באמצעות Trie

ניתן להשתמש בעובדה שלמחרוזות יש מבנה - שרשרת תווים מעל אלף-בית מאורך קבוע  $\Sigma$  - כדי למיין מהר יותר מאשר ע"י השוואות של מחרוזות. נשתמש ב-Trie באופן הבא:

**האלגוריתם**

1. הכנס את  $S_1, \dots, S_n$  ל-Trie.
2. עבור על ה-trie לפי סדר preorder וכתוב לפלט את המסלול לכל עלה. (המסלול נמצא במחסנית הרקורסיה).



**דוגמא:** נתונות המחרוזות  $ac, a, bc$ , כאשר תו סיום-מחרוזת הוא '\$' (הקטן ביותר לקסיקוגרפית). המחרוזות הממוינות הן  $a$, $ac$, $bc$$

## מיון מחרוזות נאיבי

**קלט:** מחרוזות  $S_1, \dots, S_n$  שאורכן הכולל הוא  $m = \sum_{i=1}^n |S_i|$   
**פלט:** הדפסת המחרוזות בסדר לקסיקוגרפי.

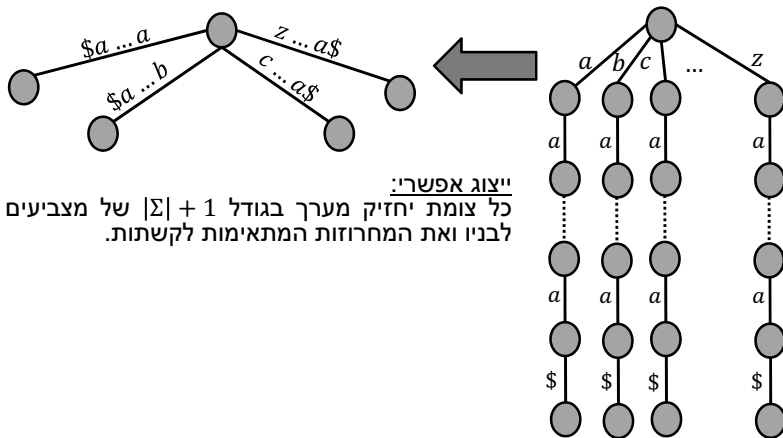
**פתרון באמצעות מיון מבוסס השוואות:**

נניח לרגע (לשם פשטות) שאורך כל המחרוזות אחיד ושווה ל- $m/n$ . השוואת שתי מחרוזות תיקח זמן  $O(|S_i|) = O(m/n)$ . לפתרון באמצעות השוואות נדרשות  $\Theta(n \log n)$  השוואות ולכן סה"כ נדרש זמן  $O((m/n) n \log n) = O(m \log n)$ .

נראה כעת פתרון בזמן  $O(m)$ .

## דחיסת Trie

נסלק מהעץ צמתים בעלי בן אחד ע"י החלפת שרשרת קשתות בקשת בודדת שתסומן בתויות המקודדת את המחרוזת המתאימה.

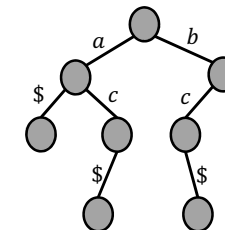


**ייצוג אפשרי:**  
 כל צומת יחזיק מערך בגודל  $|\Sigma| + 1$  של מצביעים לבניו ואת המחרוזות המתאימות לקשתות.

## ניתוח זמנים

**זמן הריצה:**

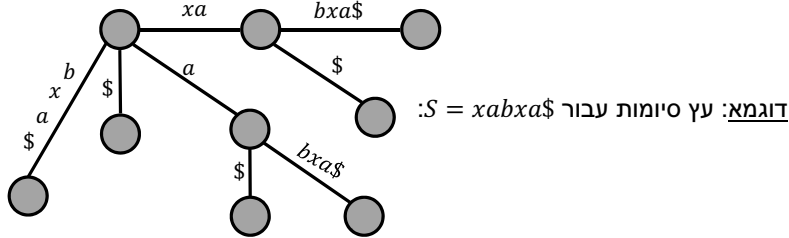
- הכנסת  $s_i$  דורשת זמן  $O(|s_i|)$ .
  - לכן זמן הכנסת כל המחרוזות הוא  $O(m) = O(\sum_i |s_i|)$ .
  - סיור ה-preorder ב-Trie דורש זמן כמספר הצמתים ומספר זה הוא  $O(m)$ .
  - ההדפסות המתבצעות במהלך הסיור דורשות זמן כסכום אורכי המחרוזות  $O(m) = O(\sum_i |s_i|)$ .
- סה"כ זמן ריצת האלגוריתם:  $O(m)$ .



**דוגמא:**

## עץ סיומות (Suffix Tree)

עץ סיומות של מחרוזת  $S$  הוא Trie שבו הוכנסו כל הסיומות של המחרוזת  $S$  עם תו סיום \$.



- לעץ סיומות עשרות שימושים במסגרת אלגוריתמים הפועלים על מחרוזות. אנו נבחן שלושה שימושים (שימושים רבים נוספים מתוארים בספר של Gusfield):
- מציאת תת מחרוזת בתוך מחרוזת נתונה (או בתוך רשימת מחרוזות נתונה).
  - מציאת תת מחרוזת ארוכה ביותר המשותפת לשתיה רשימות נתונות.
  - מימוש אלגוריתם לדחיסת אינפורמציה (Ziv-Lempel compression).

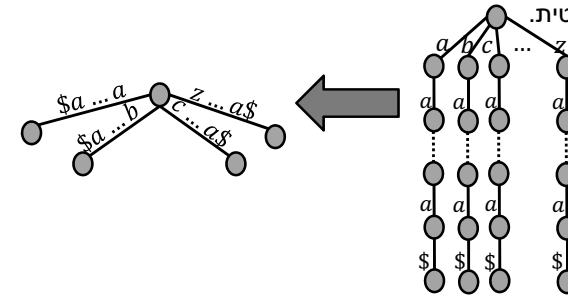
## דחיסת Trie – השפעה על מספר הצמתים

יהא  $M$  מספר הצמתים בעץ ו- $n$  מספר העלים. עקב הדחיסה, לכל אחד מ- $(M - n)$  הצמתים הפנימיים יש לפחות שני בנים.

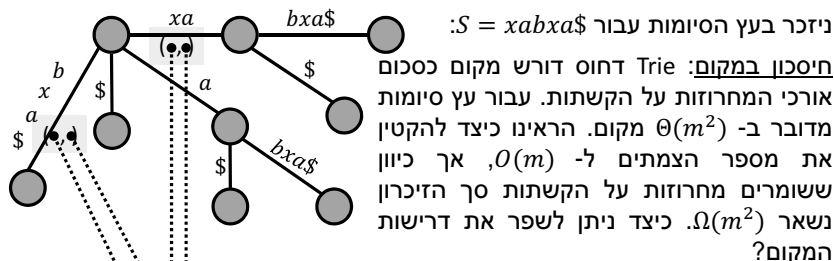
**טענה:** לעץ בו לכל צומת פנימי לפחות 2 בנים, מספר הצמתים מקיים  $2n - 1 \geq M$

הוכחה: שיקולי ספירת קשתות. כיוון שמדובר בעץ, ולפי התכונות הנ"ל, מתקיים:  
 $M - 1 = \sum_v d_{out}(v) \geq 2(M - n)$   
 ומכאן מתקיים  $2n - 1 \geq M$

לכן לאחר דחיסת Trie מספר הצמתים לינארי במספר העלים (מספר המחרוזות) ב-Trie. החסכון יהיה משמעותי יותר בהמשך כאשר תוויות הקשתות יהיו מקודדות בצורה קומפקטית.



## חיסכון הכרחי במקום



**פתרון:** נשמור העתק נפרד של המחרוזת  $S$  וכל תווית תהיה זוג מצביעים המציינים את מיקום התווית במחרוזת  $S$ . כל קשת תדרוש  $O(1)$  מקום. סך המקום הנדרש הוא  $O(m)$ .

למעשה כל אלגוריתם לינארי לבניית עצי סיומות חייב לייצג את תוויות הקשתות בצורה לא-ישירה כיון שקיימות סדרות של מחרוזות  $S_m$  באורך  $m$  כך שסכום האורכים של תוויות הקשתות של  $S_m$  גדול מ- $\Theta(m)$  ולכן זמן כתיבת עץ הסיומות ידרוש יותר מ- $\Theta(m)$  זמן (תרגיל בית).

**רמז:** הסתכלו והכלילו את המחרוזת ((000 001 010 011 100 101 110 111)).

## אלגוריתם לבניית עץ סיומות

נניח לאורך ההרצאה שאורך המחרוזת  $S$  הוא  $m$ .

**אלגוריתם נאיבי לבניית עץ סיומות עבור  $S$ :**

- הכנס את המחרוזות  $S[1 \dots m], S[2 \dots m], \dots, S[m \dots m]$  ל-Trie.
- דחוס את ה-Trie שנוצר.

ניתוח זמנים:  $Time(m) = cm + c(m - 1) + \dots + c = O(m^2)$

קיימים מספר אלגוריתמים מסובכים בהרבה המאפשרים לבנות עץ סיומות בזמן  $O(m)$  (כאשר גודל האלף-בית קבוע). אלגוריתם עם זמן ריצה זה מתואר למשל במאמר:

Esco Ukkonen. "On-line construction of suffix trees." Algorithmica, 14:249-60, 1995

ובספר של Gusfield. אנו נשתמש באלגוריתם זה כ"קופסא שחורה".

## מציאת מחרוזות קצרות בטקסט ארוך

**הבעיה:** נתונה מחרוזת  $T$  מאורך  $m$  הנקראת טקסט. לאחר זמן עיבוד ליניארי  $O(m)$  של הטקסט, יש להיות מוכנים לקבל מחרוזת  $s$  לא ידועה באורך  $n$  ולמצוא מופע של  $s$  בטקסט  $T$  (המופע הראשון) או לקבוע שהמחרוזת  $s$  אינה נמצאת בטקסט.

שימו לב שכל פתרון העובר על הטקסט  $T$  בזמן קבלת המחרוזת  $s$  ללא עיבוד מוקדם של  $T$  יאלץ לבצע לפחות  $\Theta(m)$  פעולות.

### דוגמאות לשימושים:

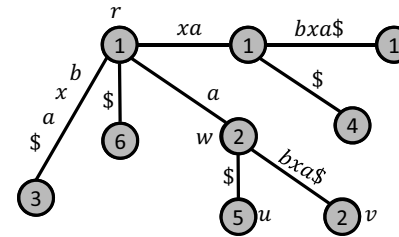
1. הטקסט הוא האנציקלופדיה בריטניקה והמחרוזת  $s$  היא מילה.
2. הטקסט הוא הגנום של אורגניזם כלשהו, כלומר מחרוזת ארוכה של האותיות  $\{A, C, T, G\}$ , והמחרוזת  $s$  היא סדרה של אותיות כאלה המקודדת גן אותו יש למצוא בגנום הנתון.

**הערה:** כמובן שקיימות וריאציות לבעיה זו כגון מציאת כל המופעים של  $s$  בטקסט הנתון, התאמה חלקית של  $s$  לטקסט, או חיפוש  $s$  בתוך אוסף טקסטים.

**הנחה:**  $m \gg n$  כלומר המחרוזת המבוקשת קצרה יחסית לאורך הטקסט.

## אינפורמציה נוספת בעץ סיומות

נבחן שוב את עץ הסיומות עבור  $S = xabxa\$$ :



ע"י סיוור postorder בעץ נוכל בזמן ליניארי  $O(m)$  לחשב לכל צומת  $z$  את המיקום הראשון של תת המחרוזת המיוצגת ע"י המסלול מהשורש ועד  $z$ . נסמן מיקום ראשוני זה ע"י  $c_z$ .

**למשל:** המחרוזת  $abxa\$$  המיוצגת ע"י המסלול  $(r, v)$  מתחילה במקום השני ב- $S$  והמחרוזת המיוצגת ע"י המסלול  $(r, u)$  מתחילה במקום החמישי ב- $S$ .

באופן כללי, עבור עלים מספרים אלה מתקבלים ע"י חיסור מספר התווים המופיעים על המסלול לצומת  $z$  מהאורך הכללי  $m$  ועוד אחד ( $m = 6$  בדוגמא זו).

בצמתים פנימיים יירשם המינימום של ערכי הילדים. למשל 2 בצומת  $w$ , כלומר  $c_w = 2$ . אמנם המיקום השמאלי ביותר של המחרוזת  $a$  ב- $s$  הוא 2.

## דחיסת אינפורמציה

**הבעיה:** טקסט מילולי (ואחר) המקודד בצורה מפורשת הוא לעיתים ארוך מהנחוץ שכן מילים וחלקי מילים חוזרים על עצמם לאורך הטקסט.

**המטרה:** בהינתן מחרוזת  $s$ , לייצר מחרוזת "  $s$  התופסת פחות מקום מ- $s$  והמכילה את אותה האינפורמציה.

**השימוש:** העברה יעילה של קבצי אינפורמציה במדיום אלקטרוני כגון בדיסקטים, ברשתות תקשורת, וכדומה. למשל פקודות הדחיסה המקובלות winzip compress-1 במערכת Windows ובמערכת Unix.

**הרעיון:** נעבור על המחרוזת הנתונה  $s$  משמאל לימין, בכל פעם שעוברים על תת מחרוזת  $z$  שכבר ראינו נחליף את  $z$  עם האינדקס והאורך של המופע השמאלי ביותר של  $z$  במחרוזת  $s$ .

האלגוריתם המתבסס על רעיון זה נקרא Ziv-Lempel compression והוא מתואר במאמרים:

Ziv, Lempel, IEEE Trans on Information Theory, 23:337-43, 1977.

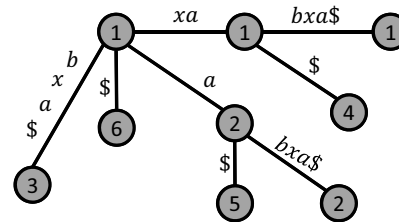
Ziv, Lempel, IEEE Trans on Information Theory, 24:530-368, 1978.

כמו כן מוכח במאמרים אלה שאסימפטוטית, זהו אלגוריתם דחיסה אופטימלי.

## אלגוריתם למציאת מחרוזות בטקסט

בנה בזמן  $O(m)$  עץ סיומות עבור הטקסט  $T$ . הוסף לכל צומת  $v$  בעץ הסיומות את המספר  $c_v$ .

בהינתן מחרוזת  $s$ , עקוב על המסלול מהשורש של עץ הסיומות לפי התווים שבמחרוזת  $s$ . אם נמצאה המחרוזת  $s$ , אזי מקום המחרוזת הוא  $c_v$  כאשר  $v$  הוא הצומת האחרון במסלול החיפוש של  $s$ .



**דוגמא:** ניבחן שוב את עץ הסיומות עבור  $T = xabxa\$$ . המחרוזת  $xa$  נמצאת במקום הראשון (והרביעי) והמחרוזת  $a$  נמצאת במקום השני (והחמישי).

**הערה:** למציאת כל המופעים של  $s$  בטקסט, האלגוריתם מוצא את  $c_u$  לכל עלה  $u$  בתת העץ ששורשו  $v$ . זמן החיפוש הוא  $O(k)$  כאשר  $k$  הוא מספר המופעים של  $s$  בטקסט. החסם נובע מכך שכפי שראינו, מספר הצמתים בתת עץ עם  $k$  עלים קטן מ- $2k$ . בדוגמא, עבור הצומת  $v$  מתקבל  $c_v = 1,4$ .





## מציאת תת מחרוזת ארוכה משותפת ל- $k$ מחרוזות

הבעיה: מצא תת מחרוזת ארוכה ביותר המשותפת ל-  $k$  מחרוזות נתונות  $S_1, S_2, \dots, S_k$  בזמן  $O(|S_1| + |S_2| + \dots + |S_k|)$ .

הרעיון כמו קודם: נבנה עץ סיומות מוכלל המכיל את הסיומות של  $k$  המחרוזות. נסמן את הצמתים ונמצא את המסלול הארוך ביותר מהשורש אשר מסומן בכל אורכו ע"י  $\{1, \dots, k\}$ .

דוגמא לשימוש: המחרוזות הנתונות הם הגנום של מספר אורגניזמים, והמחרוזת הארוכה ביותר המשותפת להם עוזרת למציאת התאמות בן הגנומים השונים.

כמובן שבשימוש זה נצטרך להכניס מגבלות נוספות, כגון מיקום תת המחרוזת בכל גנום, התאמה חלקית לחלק מהגנומים, מחרוזות משותפות נוספות, וכו'.