הוכן ע"י אמיר רובינשטיין

מבני נתונים ומבוא לאלגוריתמים

נושא 12

מחרוזות
Strings

בתוכנית

- נכיר מבנה נתונים יעיל למילון של מחרוזות - trie.

- נכיר אלגוריתם למציאת תת-סדרה משותפת ארוכה ביותר.

  זהו אלגוריתם הפועל בשיטת תכנון דינאמי (dynamic programming).

  (ביבליוגרפיה לנושא זה: מבוא לאלגוריתמים, פרק 16 במהדורה הראשונה)

מבנה הנתונים trie

מוטיבציה

<u>הגדרה</u>: <u>מחרוזת</u> (string) מעל קבוצה נתונה □ היא סדרה של איברים מתוך □.

את □ נכנה <u>א"ב</u>, ואת איבריה <u>אותיות</u>.

על מחרוזות אפשר להגדיר יחס סדר, שנקרא <u>סדר לקסיקוגרפי</u> (lexicographic order):

נאמר שהמחרוזת $a = a_0a_1\ldots a_p$ קטנה לקסיקוגרפית מהמחרוזת $b = b_0b_1\ldots b_q$

אם מתקיים אחד משני התנאים:

1. קיים שלם j כך ש- $a_i=b_i$ עבור $i=0,1,\ldots,j-1$ **וגם** $a_j < b_j$.

או:

2. $p<q$ ו- $a_i = b_i$ עבור $i=0,1,\ldots,p$.

מוטיבציה

<u>המשימה</u> - לממש ביעילות מילון (כולל מינימום, עוקב, וכו') של מחרוזות.

אפשרויות מוכרות:
- AVL (שבכל צומת שלו יש מחרוזת או מצביע למחרוזת)
- hash table (שבכל איבר שלה יש מחרוזת או מצביע למחרוזת)

אפשר למשל להשתמש בטבלת גיבוב עם שרשור. נניח שבידנו פונקציה גיבוב טובה למחרוזות.

נתחו את יעילותם של המימושים הללו.

<u>זיכרו</u>: השוואה בין שתי מחרוזות דורשת זמן ליניארי באורך המחרוזת הקצרה מהשתיים.

- הניחו לשם פשטות שכל המחרוזות באותו אורך l.
- מספר המחרוזות יסומן כרגיל ב- n.

<u>מוטיבציה</u>

<u>AVL</u>

- <u>הכנסה / חיפוש</u>: בכל צומת תתבצע השוואה בזמן $\Theta(l)$, ומספר ההשוואות הנ"ל הוא $\Theta(\log n)$.
  לכן כל הפעולות יבוצעו בזמן $\Theta(l \cdot \log n)$.
- <u>מחיקה / מינימום / עוקב</u>: $\Theta(\log n)$.

<u>hash table</u>

- <u>הכנסה</u>: $\Theta(l)$ במקרה הגרוע, בהנחה שחישוב ערך הגיבוב למחרוזת דורש $\Theta(l)$ (אחרת $\Theta(1)$).
- <u>מחיקה</u>: בזמן קבוע במקרה הגרוע, אם הרשימות דו-כיווניות ונתון מצביע.
- <u>חיפוש</u>: $\Theta(l)$ בממוצע, $\Theta(l \cdot n)$ במקרה הגרוע.
- <u>מינימום / עוקב</u>: $\Theta(l \cdot n)$ בכל המקרים

<u>מוטיבציה</u>

נראה כעת מימוש יעיל יותר, שבו סיבוכיות הפעולות כלל לא תלויה <u>במספר</u> המחרוזות שבמבנה, אלא רק ב<u>אורכן</u>.

מבנה הנתונים trie

מקור השם trie הוא המילה retrieval (אחזור).

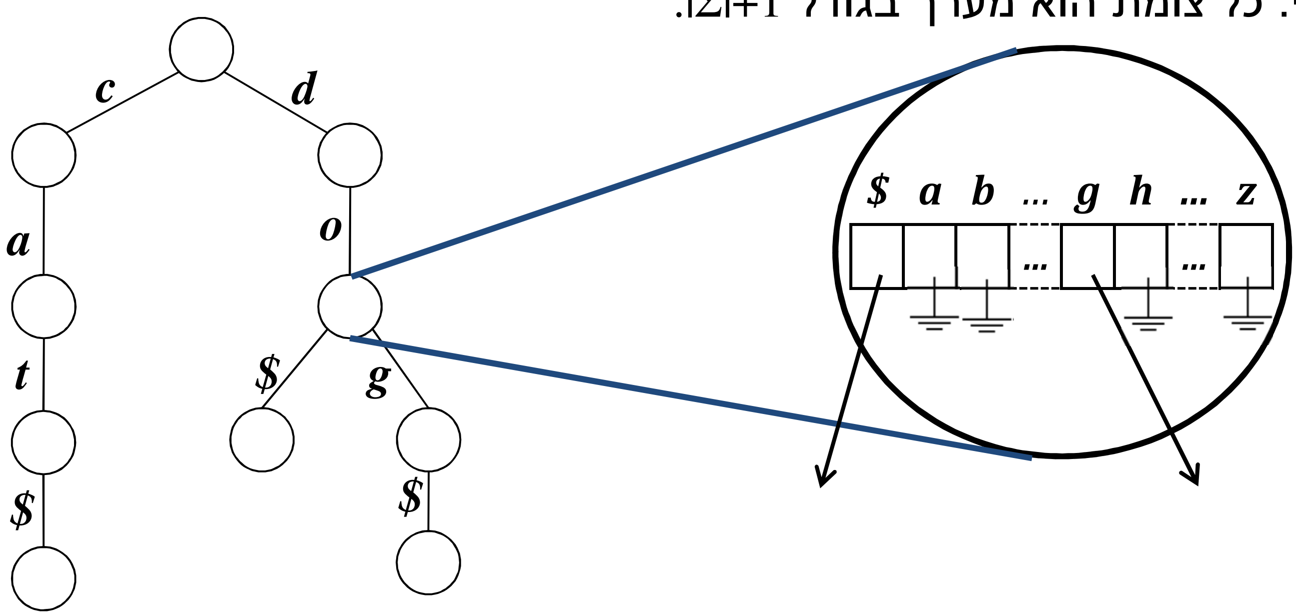
<u>שתי הנחות</u>:

1. לכל אורך ההרצאה נניח כי גודלו של □ קבוע לצורכי ניתוח סיבוכיות.
2. קיים תו *$* אינו שייך לא"ב □. תו זה ישמש כקטן ביותר לקסיקוגרפית, וייצג סוף מחרוזת.

trie הוא עץ, שבו לכל צומת לכל היותר $|\Sigma|+1$ בנים, עבור כל אחת מאותיות הא"ב ו- *$*.

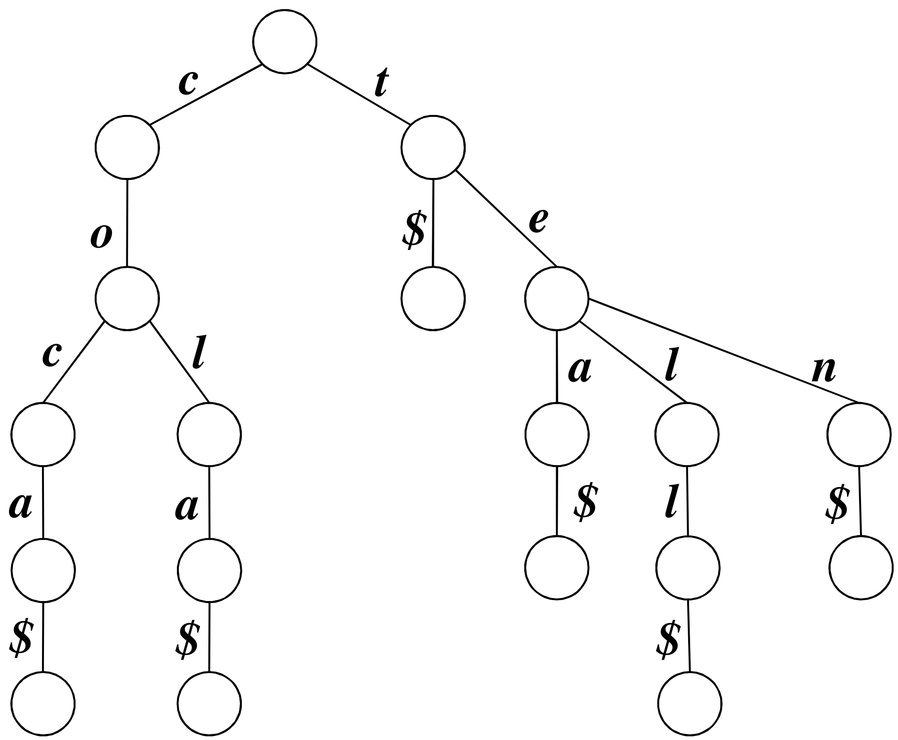
מימוש אפשרי: כל צומת הוא מערך בגודל $|\Sigma|+1$.

לדוגמה:



trie המכיל 3 מחרוזות: *cat*, *do*, *dog*

מבנה הנתונים trie



עוד דוגמה:

אילו מחרוזות נמצאות ב- trie הבא?

כל מחרוזת מיוצגת ע"י מסלול מהשורש לעלה (מספר העלים הוא כמספר המחרוזות).

נתאר כעת כיצד יבוצעו פעולות המילון השונות על trie.

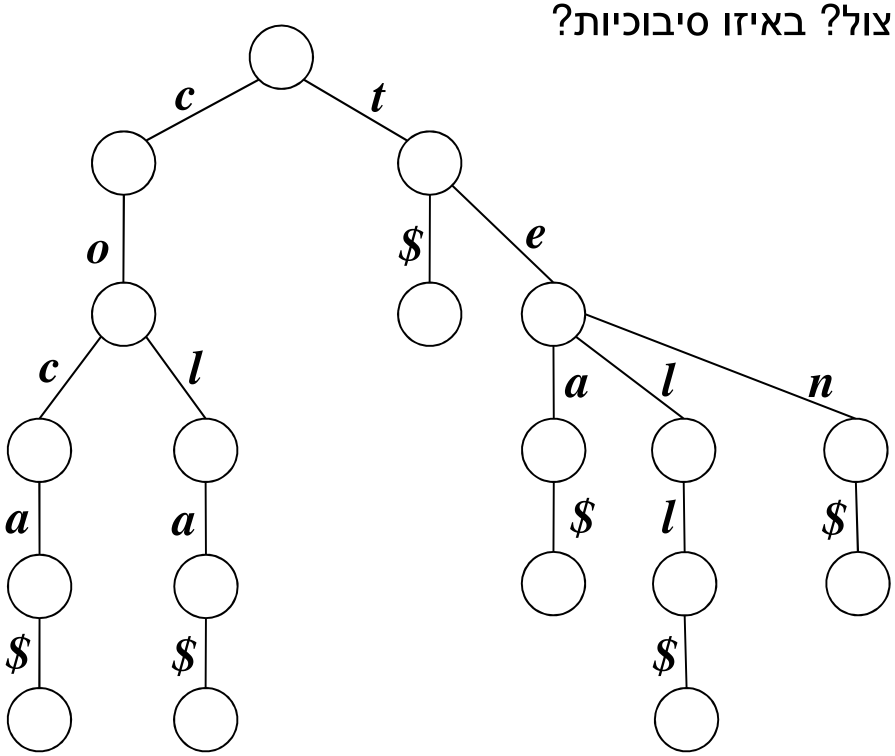
פעולות על trie

- Search(s) – עוקבים אחר המסלול של s ב- trie. s נמצאת אם"ם אפשר להגיע לעלה ($\$$).

- Insert(s) – עוקבים אחר המסלול של s, עד שנתקלים ב- Nil, ואז ביתרת המסלול מקצים צמתים חדשים ומצביעים בהתאם. בסוף מוסיפים $\$$.

- Delete(s) – יש למחוק את חלק המסלול של s שאחרי נקודת הפיצול* האחרונה שלו.

* איך בודקים אם צומת הוא פיצול? באיזו סיבוכיות?



<u>הפעולות הנ"ל רצות בזמן $\Theta(|s|)$.</u>

פעולות על trie

- Minimum() – עוקבים אחר המסלול השמאלי ביותר, ו"רושמים את האותיות דרכן עברנו".

סיבוכיות: בכל צומת יש לסרוק את המערך כדי לגלות מיהו המצביע השמאלי ביותר שאינו Nil.

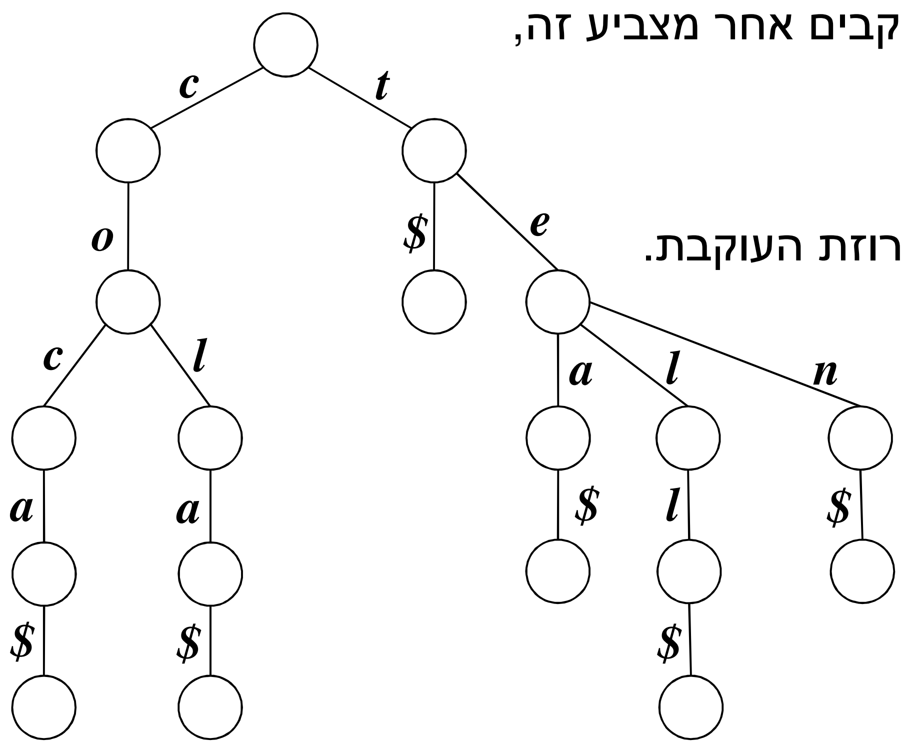
פעולה זו דורשת $\Theta(|\Sigma|) = \Theta(1)$ בכל צומת במסלול.

סה"כ $\Theta(|s_{min}|)$ כאשר s_{min} היא המחרוזת המינימלית.

- Successor(s) – עוקבים אחר המסלול של s עד לעלה. משם חוזרים לנקודת הפיצול האחרונה שיש בה מצביע ימני יותר שאינו Nil, עוקבים אחר מצביע זה,

ומשם ממשיכים כמו ב- Minimum.

סיבוכיות: $\Theta(|s|+|s'|)$ כאשר s' היא המחרוזת העוקבת.



סיבוכיות מקום

נתון trie המכיל n מחרוזות שאורכן הכולל m.

שאלה: מהי סיבוכיות הזיכרון הדרושה?

תשובה: מספר הצמתים כפול כמות הזיכרון שדורש כל צומת.

- כל צומת מכיל מערך בגודל $|\Sigma|+1 = \Theta(1)$.

- כמות הצמתים: ישנו שורש אחד, ובנוסף מחרוזת s_i מיוצגת ע"י $|s_i|+1$ צמתים.
  במקרה הגרוע, אם אין צמתים משותפים כלל, כמות הצמתים הכוללת היא:

$$\#\text{צמתים} = 1 + \sum_{i=1}^{n} (|s_i|+1) = 1 + m + n = \Theta(m)$$

סיבוכיות הזיכרון הדרושה היא אם כן ליניארית באורך הכולל m, ולא תלויה כלל במספר המחרוזות.

מיון מחרוזות

נתונות n מחרוזות, שאורכן הכולל הוא m.

אנו מחפשים אלגוריתם יעיל שממיין את המחרוזות לפי סדר לקסיקוגרפי.

<u>פתרון אפשרי (לא יעיל)</u>:

להכניס את המחרוזות ל- AVL (בכל צומת מחרוזת או מצביע למחרוזת), ולסייר בעץ inorder.

<u>סיבוכיות</u>:

נניח לשם פשטות שכל מחרוזת באורך אחיד m/n.

הכנסת n המחרוזות לעץ דורשת $\Theta(\frac{m}{n}\sum_{i=1}^{n}\log i) = \Theta(m\log n)$

הסיור כולל ההדפסות $\Theta(n+m) = \Theta(m)$.

סה"כ: $\Theta(m\log n)$

נראה כעת כיצד שימוש ב- trie מוריד את הסיבוכיות ל- $\Theta(m)$.

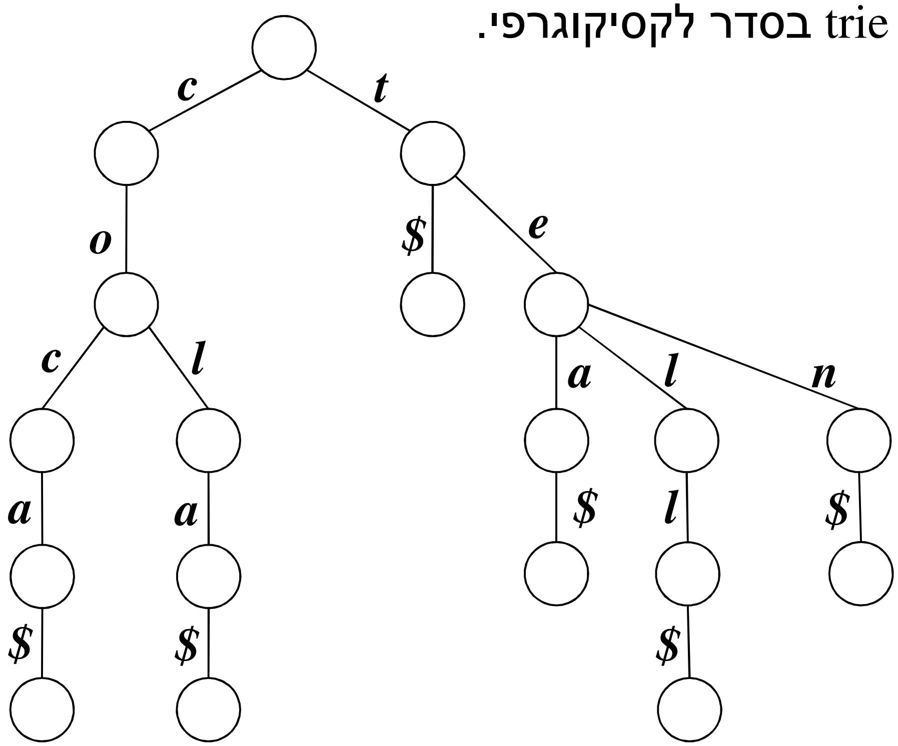
מיון מחרוזות בעזרת trie

<u>תרגיל</u>

נתון trie כלשהו, ובו n מחרוזות שאורכן הכולל הוא m.

תארו אלגוריתם שמדפיס את המחרוזות שב- trie בסדר לקסיקוגרפי.

מה סיבוכיות הפתרון שלכם?



מיון מחרוזות בעזרת trie

בהמשך לתרגיל הקודם, להלן אלגוריתם למיון n מחרוזות, שאורכן הכולל הוא m.

1. מכניסים ל- trie את המחרוזות בזו אחר זו.
2. מבצעים סיור pre-order ב- trie באופן הבא:

   2.1 מאתחלים רשימה מקושרת ריקה

   2.2 בכל ירידה בעץ מוסיפים את האות "דרכה עברנו" לסוף הרשימה

   2.3 בכל עלייה בעץ מוציאים מהרשימה את האות שבסופה

   2.4 בכל פעם שמגיעים לעלה, מדפיסים את תוכן הרשימה מתחילתה עד סופה

סיבוכיות

1. $\Theta(\sum_{i=1}^{n} |s_i|) = \Theta(m)$
2. מספר הצמתים בעץ הוא כאמור במקרה הגרוע $\Theta(m)$, וסיור בעץ מתבצע כזכור בזמן ליניארי במספר הצמתים (שורות 2.2 ו- 2.3 משפיעות רק על הקבועים).

שורה 2.4 לא בהכרח רצה בזמן קבוע כל פעם, אבל סך כל זמן הריצה שלה הוא $\Theta(m)$ (מדוע?)

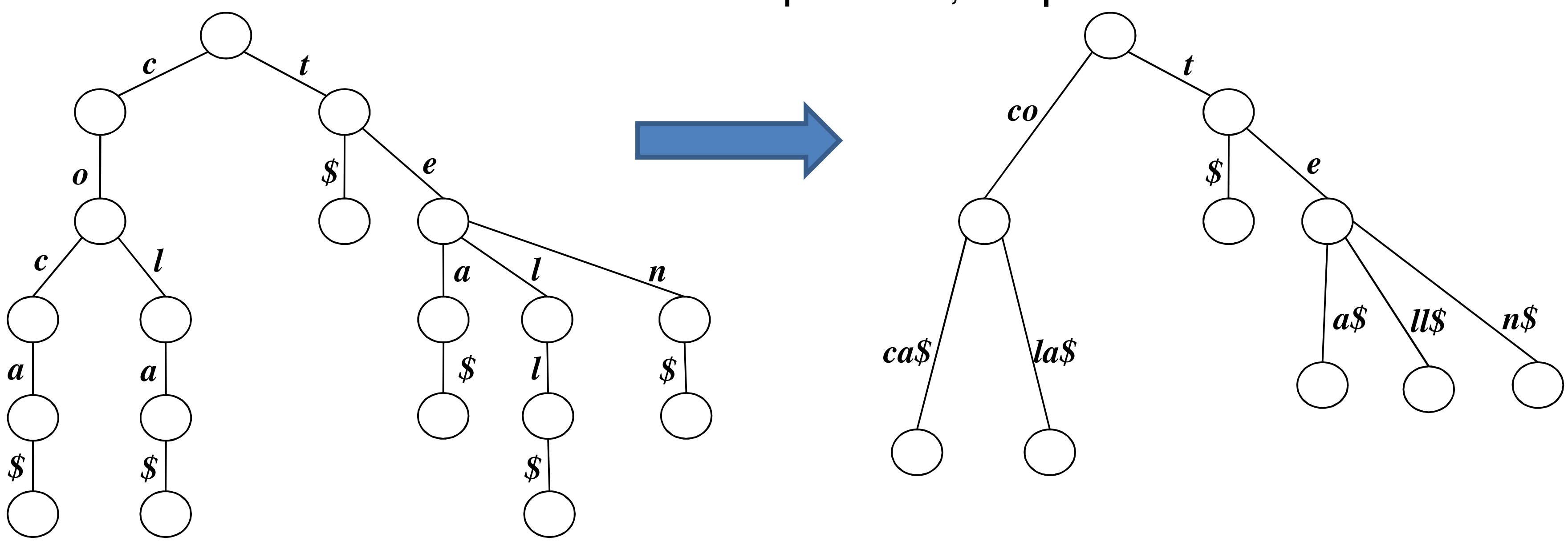
סה"כ $\Theta(m)$.

שיפור סיבוכיות מקום - דחיסה

נתון trie המכיל n מחרוזות שאורכן הכולל m.

<u>כזכור, סיבוכיות הזיכרון הדרושה היא ליניארית באורך הכולל m, ולא תלויה כלל במספר המחרוזות</u>.
האם ניתן לחסוך בזיכרון?

דחיסת trie - ביטול צמתים בעלי בן יחיד, כדי לחסוך בצמתים.



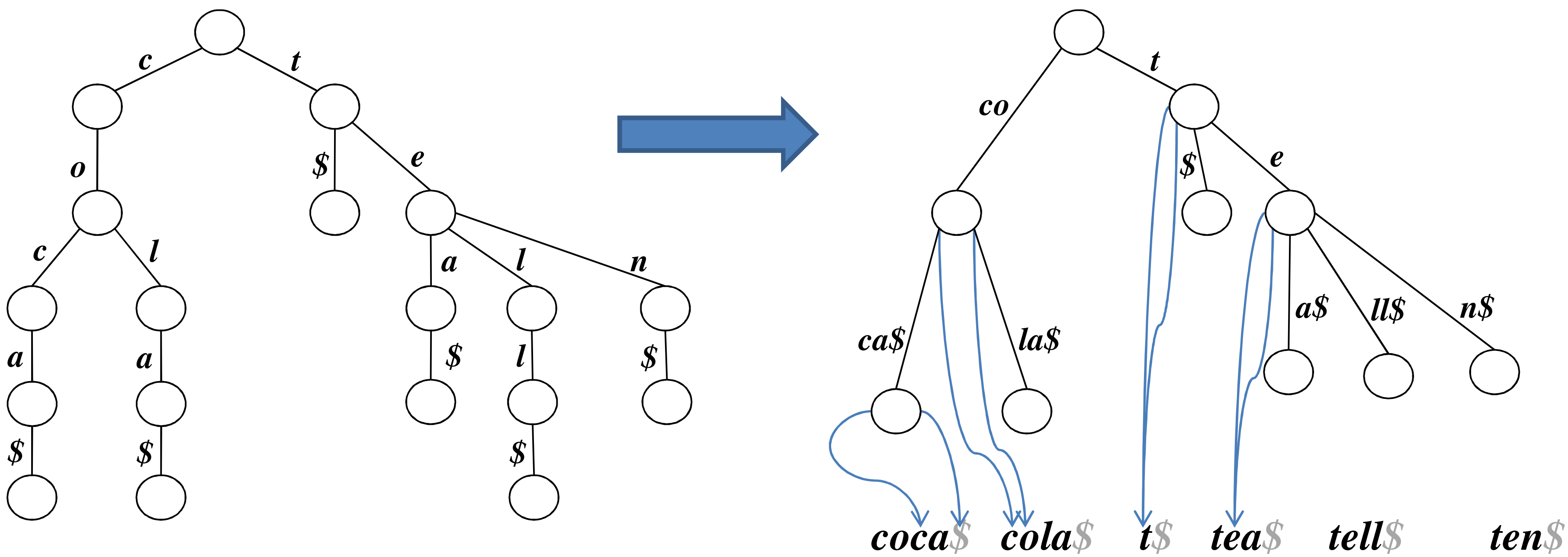
לאחר הדחיסה מספר הצמתים קטן מ- $\Theta(m)$ ל- $\Theta(n)$:

- יש n עלים (כמספר המחרוזות), לכל היותר $n/2$ אבות, לכל היותר $n/4$ סבים, וכו'.
- לכן מספר הצמתים לא גדול מ- $n + n/2 + n/4 + \ldots < 2n$.

שיפור סיבוכיות מקום - דחיסה

נוצרה בעיה – לא מספיק להחזיק מערך בכל צומת, כי צמתים מייצגים תתי-מחרוזות ולא אותיות.

פתרון: בכל צומת נשמור שני מצביעים – לתחילת תת-המחרוזת שמייצג הצומת ולסופה.



כך, בכל צומת דרוש עדיין $\Theta(1)$ זיכרון, וכאמור כמות הצמתים ירדה ל- $\Theta(n)$.

אבל צריך כעת לשמור עותקים מהמחרוזות, אליהם יופנו המצביעים מהצמתים – סה"כ $\Theta(m)$.

מתי אם כן דחיסת trie היא בעלת ערך בחיסכון בזיכרון?

שיפור סיבוכיות מקום בעץ סיומות

השיפור הנ"ל בסיבוכיות הזיכרון רלוונטי יותר בגרסה חשובה ושימושית של trie שנקראת "עץ סיומות" (suffix tree).

זהו trie אליו הוכנסו כל הסיומות של מחרוזת מסוימת.

לדוגמא, עץ סיומות עבור המחרוזת "s = "banana:

המחרוזות בעץ:

מחרוזת ריקה

a

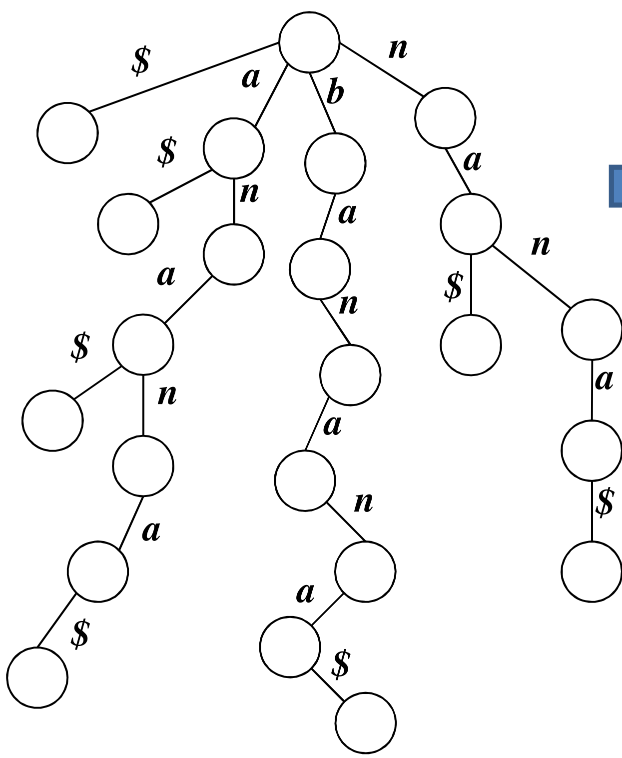
na

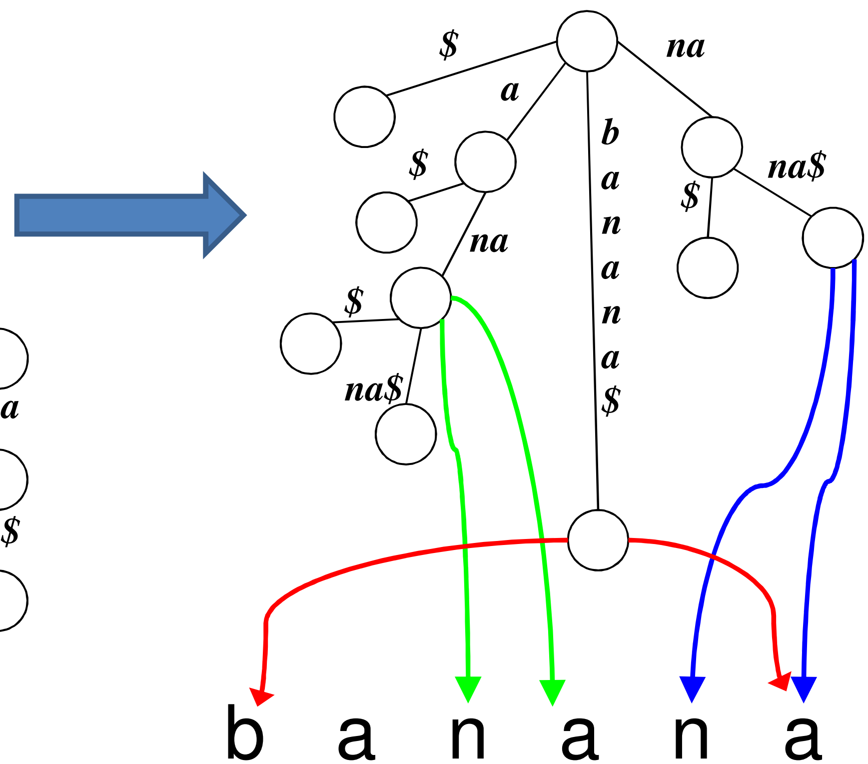
ana

nana

anana

s = bananas





$\Theta(|s|^2)$ זיכרון ב- trie הלא דחוס, במקרה הגרוע

$\Theta(|s|)$ זיכרון ב- trie הדחוס במקרה הגרוע

עץ סיומות

לעצי סיומות מספר שימושים חשובים, ביניהם:

1. מציאת תת-מחרוזת בתוך מחרוזת נתונה (מנוע חיפוש, ביואינפורמטיקה).
2. דחיסת אינפורמציה (למשל אלגוריתם ziv-lempel compression).
3. עבור שתי מחרוזות נתונות, מציאת תת-מחרוזת (רצופה) משותפת ארוכה ביותר.

ניתן לקרוא על כך למשל ב-

Algorithms on strings, Trees and sequences, Dan Gustfield

Chapter 5, 7.3, 7.4, 7.17

<u>תת-סדרה משותפת ארוכה ביותר</u>

<u>Longest common subseries (LCS)</u>

תת-סדרה משותפת ארוכה ביותר

הגדרת הבעיה

קלט: שתי מחרוזות: X באורך m ו- Y באורך n.

פלט: אורך תת-סדרה משותפת ארוכה ביותר.

הערה: תת-סדרה משותפת לא חייבת להיות רצופה (בניגוד ל"תת-מחרוזת").

דוגמה: $Y = acabc$ $X = bacaacb$

תתי סדרות משותפות מקסימליות: $acac$ או $acab$

פתרון נאיבי: לבדוק כל תת-סדרה אפשרית של X: $\Theta(n2^m)$. (כמה תת-סדרות יש לסדרה באורך m?)

סימונים

- תת-סדרה משותפת ארוכה ביותר של מחרוזות X ו- Y תסומן $\mathrm{LCS}(X,Y)$.
- האורך של $\mathrm{LCS}(X,Y)$ יסומן $\mathrm{L}(X,Y)$.

עבור מחרוזת X:

- נסמן ב- x_i את התו ה- i של X
- נסמן ב- $X[i..j]$ את תת-הסדרה מ- x_i עד x_j.

תת-סדרה משותפת ארוכה ביותר

3 אבחנות, שיעזרו לגיבוש פתרון

עבור שתי מחרוזות: X באורך m ו- Y באורך n מתקיים:

אבחנה 1: אם אחת הסדרות ריקה אז $\mathbf{L}(X,Y) = 0$ ו- $\mathbf{LCS}(X,Y) = \square$

אבחנה 2: אם $x_m = y_n$ אז $\mathbf{L}(X,Y) = \mathbf{L}(\ X[1..m\text{-}1],\ Y[1..n\text{-}1]\ ) + 1$.

$$\mathbf{LCS}(X,Y) = \mathbf{LCS}(\ X[1..m\text{-}1],\ Y[1..n\text{-}1]\ ) \ \square\ x_m$$

אבחנה 3: אם $x_m \neq y_n$ אז $\mathbf{L}(X,Y) = \max\{\ \mathbf{L}(\ X[1..m\text{-}1],\ Y\ )\ ,\ \mathbf{L}(\ X,\ Y[1..n\text{-}1]\ )\ \}$.

$$\mathbf{LCS}(X,Y) = \mathbf{LCS}(\ X[1..m\text{-}1],\ Y\ ) \quad OR \quad \mathbf{LCS}(\ X,\ Y[1..n\text{-}1]\ )$$

פתרון רקורסיבי פשוט

לאור האבחנות הללו, להלן אלגוריתם רקורסיבי לחישוב $L(X,Y)$

```
L(X, Y)
1.  m ← length[X],  n ← length[Y]
2.  if  n=0  or  m=0
3.          return  0
4.  if  x_m = y_n
5.          return  L(X[1..m-1], Y[1..n-1])  + 1
6.  else
7.          return  max {  L(X[1..m-1], Y)  , L(X, Y[1..n-1])  }
```

סיבוכיות

נסמן ב- s את סכום האורכים $n+m$.

במקרה הגרוע נכנסים לשורה 7 בכל פעם:

$$T(s) = 2T(s-1) + 1 = \Theta(2^s)$$

סיבוכיות אקספוננציאלית בסכום האורכים!

ממה זה נובע?

<u>פתרון משופר בשיטת "תכנון דינאמי"</u>

בפתרונות רקורסיביים לבעיות, אם ישנן תת-בעיות חופפות זו לזו (כלומר יש להן תת-תת-בעיות משותפות), פעמים רבות הדבר גורר סיבוכיות זמן אקספוננציאלית. זאת מכיוון שהאלגוריתם פותר פעמים רבות את אותן תת-בעיות.

(בהקשר זה, היזכרו גם בפתרון הרקורסיבי לחישוב האיבר ה- n בסדרת פיבונאצ'י).

<u>תכנון דינאמי</u> (dynamic programming) היא שיטה שמאפשרת חסכון בזמן ע"י כך שכל תת-בעיה נפתרת רק פעם אחת: מאחסנים בטבלה את הפתרונות שחושבו, ושולפים אותם מהטבלה בעת הצורך.

הדגמת הפתרון בשיטת "תכנון דינאמי"

	$Y=$	a	c	a	b	c
$X=$	0	0	0	0	0	0
b	0					
a	0					
c	0					
a	0					
c	0					
b	0					

בטבלה הבאה, $C[i, j]$ יכיל את $\mathbf{L}(X[1..i], Y[1..j])$.

<u>אתחול</u>: שורה ראשונה ועמודה ראשונה יכילו 0.

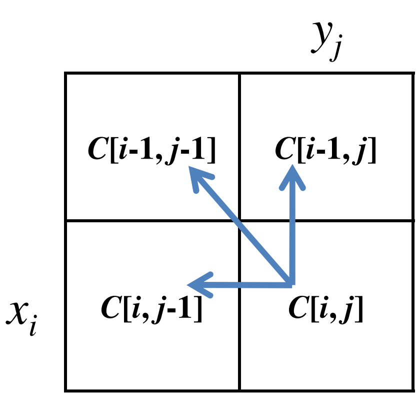
<u>סיום</u>: בסיום האלגוריתם הפלט יהיה רשום ב- $C[m, n]$.

כלומר $\mathrm{L}(X, Y) = C[m, n]$.

<u>צעדי האלגוריתם</u>:

נחשב את הערכים בטבלה שורה אחר שורה.

אפשר לחשב כל תא מתוך 3 תאים סמוכים בלבד:



אם $x_i = y_j$ אז $C[i, j] = C[i\text{-}1, j\text{-}1] + 1$.

אחרת - $x_i \neq y_j$ אז

אם $C[i\text{-}1, j] \geq C[i, j\text{-}1]$ אז $C[i, j] = C[i\text{-}1, j]$

אחרת $C[i, j] = C[i, j\text{-}1]$

הדגמת הפתרון בשיטת "תכנון דינאמי"

אם $x_i = y_j$ אז $C[i, j] = C[i\text{-}1, j\text{-}1] + 1$.

אחרת - $x_i \neq y_j$ אז

אם $C[i\text{-}1, j] \geq C[i, j\text{-}1]$ אז $C[i, j] = C[i\text{-}1, j]$

אחרת $C[i, j] = C[i, j\text{-}1]$

	$Y=$	a	c	a	b	c
$X=$	0	0	0	0	0	0
b	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1
a	0	↖ 1	← 1	↖ 1	↑ 1	↑ 1
c	0	↑ 1	↖ 2	← 2	← 2	↖ 2
a	0	↖ 1	↑ 2	↖ 3	← 3	← 3
c	0	↑ 1	↖ 2	↑ 3	↑ 3	↖ 4
b	0	↑ 1	↑ 2	↑ 3	↖ 4	↑ 4

האלגוריתם

לשם נוחות נמספר שורות ועמודות החל ב- 0.

```
L (X, Y)
m ← length[X],  n ← length[Y]
for  i ← 0  to  m            C[i, 0] ← 0          //leftmost column
for  j ← 0  to  n            C[0, j] ← 0          //top row
for   i ← 1 to m
          for   j ← 1 to n
                         if  x_i = y_j
                              C[i, j] ← C[i-1, j-1] + 1
                         else   if   C[i-1, j]  ≥  C[i, j-1]
                                                 C[i, j] ← C[i-1, j]
                               else              C[i, j] ← C[i, j-1]
return C[m,n]
```

סיבוכיות $\Theta(mn)$.

שאלה: מה צריך להוסיף כדי שנוכל גם לשחזר את ה- LCS עצמה?

שחזור תת-סדרה משותפת ארוכה ביותר

נשמור בטבלה נוספת B את ה"כיוונים".

בסיום האלגוריתם:

- נאתחל מחרוזת ריקה Z
- נעקוב אחר המסלול החל ב- $B[m, n]$:
  - בכל פעם ש- $B[i, j] =$ "↖" נוסיף את x_i לתחילת המחרוזת Z.

שימו לב שבסיום $Z = \mathrm{LCS}(X,Y)$, אבל ייתכן שיש עוד תת-סדרות ארוכות ביותר שונות מ- Z.

X = \ Y=		***a***	***c***	***a***	*b*	***c***
	0	0	0	0	0	0
b	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1
a	0	**↖** 1	← 1	↖ 1	↑ 1	↑ 1
c	0	↑ 1	**↖** 2	← 2	← 2	↖ 2
a	0	↖ 1	↑ 2	**↖** 3	**←** 3	← 3
c	0	↑ 1	↖ 2	↑ 3	↑ 3	**↖** 4
b	0	↑ 1	↑ 2	↑ 3	↖ 4	**↑** 4

$\mathrm{LCS}(X, Y) =$ "*acac*"

סיבוכיות השחזור: $\Theta(m+n)$

שחזור תת-סדרה משותפת ארוכה ביותר

בנית הטבלה B:

```
L (X, Y)
m ← length[X],  n ← length[Y]
for  i ← 0  to  m          C[i, 0] ← 0          //leftmost column
for  j ← 0  to  n          C[0, j] ← 0          //top row
for   i ← 1 to m
          for   j ← 1 to n
                    if  x_i = y_j
                         C[i, j] ← C[i-1, j-1] + 1                B[i, j] ← “↖“
                    else if     C[i-1, j]  ≥  C[i, j-1]
                                   C[i, j] ← C[i-1, j]          B[i, j] ← “↑“
                         else      C[i, j] ← C[i, j-1]          B[i, j] ← “←“
return C[m,n]
```

למעשה, היה אפשר לוותר על הטבלה B. כיצד?

שאלות חזרה

1. מצאו תת-סדרה משותפת ארוכה ביותר של המחרוזות:

$X=BABBDAC$
$Y=BBADCA$

ע"י הרצת האלגוריתם LCS.
מיצאו גם תת-סדרות משותפות ארוכות ביותר אחרות מזו שהאלגוריתם מחזיר.

<u>תשובות לשאלות חזרה</u>

1. $BADA$, $BADC$, $BBDA$, $BBDC$, $BBAC$

תרגילים

תרגילים מומלצים מהספר

פרק 12 במהדורה השנייה (2008)
בעיה 12-2

פרק 16 במהדורה הראשונה (1998)
16.3-1
16.3-2
16.3-5

תרגילים נוספים

1. הציעו אלגוריתם שזמן ריצתו $O(n^2)$ אשר מוצא תת-סדרה לא יורדת ארוכה ביותר של סדרה בת n מספרים.

2. דרוש מבנה נתונים למימוש הפעולות:
$\text{Init}(\{s_1, s_2, \ldots, s_n\})$ – אתחול המבנה בהינתן קבוצה של n מחרוזות.
סיבוכיות - $O(m)$ כאשר m הוא האורך הכולל של כל המחרוזות.
$\text{Find-Permute}(s)$ – בדיקה האם s היא פרמוטציה של אחת המחרוזות $s_1,\ldots,s_n$.
סיבוכיות - $O(|s|)$.

כלומר, בהינתן קבוצת מחרוזות ידועה מראש, יש לענות על השאלה הבאה:
האם מחרוזת חדשה כלשהי היא פרמוטציה של אחת המחרוזות הנתונות?

3. עץ סיומות (suffix tree) של מחרוזת נתונה s הוא trie אליו הוכנסו כל הסיומות של s.
(סיומת של מחרוזת היא תת-מחרוזת שלה החל ממקום מסוים ועד סופה).

בהינתן עץ סיומות של מחרוזת s, הראו כיצד ניתן למצוא בעזרתו האם מחרוזת s' היא תת-מחרוזת (רצופה) של s.

<u>פתרון 1</u>

עבור סדרה X, ניצור עותק שלה X' ונמיין אותו. כעת נקרא ל- $LCS(X, X')$.
התוצאה היא תת-סדרה ארוכה ביותר המשותפת לסדרה המקורית ולעותק הממוין שלה,
וזוהי גם תת-סדרה מונוטונית ארוכה ביותר של הסדרה המקורית.

זמן ריצה: $\Theta(n\log n)$ עבור המיון, ועוד $\Theta(n^2)$ עבור האלגוריתם LCS. סה"כ $\Theta(n^2)$.

<u>פתרון 2</u>

:Init

- נמיין כל מחרוזת s_i באמצעות Counting-Sort.
  סיבוכיות מיון מחרוזת בודדת s_i : $\Theta(n+k) = \Theta(|s_i|+ |\Sigma|) = \Theta(|s_i|)$
  סה"כ – $\Theta(m)$
- נכניס את כל המחרוזות הממוינות ל- trie בזו אחר זו.
  סיבוכיות הכנסת מחרוזת בודדת $\Theta(|s_i|)$, ובסה"כ $\Theta(m)$.

:Find-Permute

- נמיין את s באותו אופן, ונבצע חיפוש ב- trie של s הממוינת. סיבוכיות המיון + החיפוש $\Theta(|s|)$.

<u>פתרון 3</u>

תת-מחרוזת של מחרוזת נתונה היא התחלה (רישא) של סיומת (סיפא) כלשהי שלה.
לכן s' היא תת-מחרוזת של s אם"ם קיים מסלול של s' שמתחיל בשורש של עץ הסיומות של s.