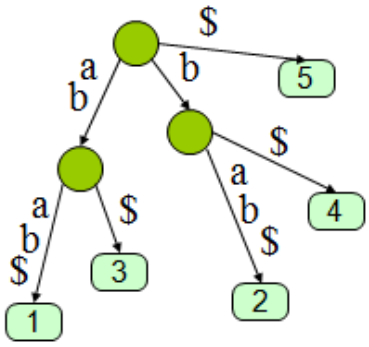
מבני נתונים ואלגוריתמים – תרגול #12

עצי סיפא

נתונה מחרוזת *S* באורך *n*. עץ סיפות של *S* הוא מבנה נתונים:

- עץ מושרש
- מכיל את כל הסיפות של *S* (=מסלולים מהשורש לעלים).
- הקשתות מתאימות לתתי-מחרוזות (לא ריקות).
- לכל צומת פנימי יש לפחות 2 בנים.
- יש בדיוק *n* עלים.
- מסמנים סוף של סיפא ע"י תו מיוחד ($) על מנת להבטיח שהסיפא אינה רישא של סיפא אחרת.
- בעלים שומרים את אינדקס ההתחלה של הסיפא ב-*S*.



<u>האלגוריתם הנאיבי:</u>

- מכניס את כל הסיפות, מהארוכה לקצרה.
- מחפשים את הסיפא בעץ. אם נתקעים, מפצלים את הקשת.
- סיבוכיות: $O(n^2)$.

<u>האלגוריתם של *Ukkonen*:</u>

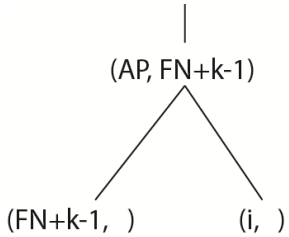
- האלגוריתם בונה עץ סיפות ב-$O(n)$.
- *Suffix links*: בהינתן $X\alpha$, כאשר X הוא תו ו-α מחרוזת, אז *SL* הוא קשת מ-$X\alpha$ ל-α.
- יש עוד כמה טריקים שגורמים לאלגוריתם לרוץ ב-O(n).

<u>פסאודו-קוד:</u>

נתונה מחרוזת S באורך n עם תו $ בסופה.

עוברים על כל התווים S[i]:
- אם AP == root ו-S[i] לא קיים – צור ענף עם S[i] ← (i,).
- אם AP == root ו-S[i] קיים – כנס לענף, ++AP, ++k.
- אחרת:
  - אם S[i]==S[AP+k] – ממשיכים להיכנס – K++
  - אחרת – מפצלים קודקודים כל עוד K≠0.

פיצול:
- תחילה מפצלים את הקודקוד ש-AP מצביע עליו , —K
  * *FN = הקודקוד הראשון ביותר מהשורש בענף מסוים.*
- כל עוד k≠0:
  - מחפשים את S[i-k:i] – לשם יצביע EP.
  - יוצרים SL מ-EP ל-AP (אם לא קיים), ואז AP==EP



- אם S[i-k:i] אינו רישא של ענף – נפצל את AP
- אם k==0 :
  - יוצרים SL מ-AP ל-root.
  - אם S[i] קיים – נכנסים, ++K. אחרת – יוצרים קודקוד חדש עם S[i].

יש דוגמא מלאה באתר .

יישומים:

- התאמת מחרוזות
- Generalized suffix tree – ניתן לבנות עץ סיפות עבור מספר מחרוזות. בסוף הסיפות של כל מחרוזת נוסיף תו מיוחד שונה.

← אפשר למצוא את תת המחרוזת המשתותפ הארוכה ביותר.

← אפשר למצוא פלינדרום – נבנה עץ עם S ו-S^R.

אינפורמציה דחיסה

אנטרופיה: מדד לואריאביליות/חוסר אחידות של התפלגות.

X הוא מ"מ המקבל ערכים $\{x_1, x_2, \dots, x_n\}$ עם הסתברויות$p_1, p_2, \dots, p_3$.

האנרטופיה של X היא:

$$H(X) = -\sum_{i=1}^{n} p_i log\,(p_i)$$

(עבור מ"מ בדיד)

$$H(X) = -\int_{-\infty}^{\infty} f(x)\log\big(f(x)\big)\mathrm{dx}$$

(עבור מ"מ רציף, כאשר f היא פוקנציית הצפיפות של X).

← האנטרופיה לא תלויה בערכים עצמם, אלא רק בהסתברויות!

דוגמא: X מקבל את ערכים 0,1 בהסתברויות $\mathrm{p}_0, \mathrm{p}_1$.

- אם $p_1 = 1$ ו-$p_0 = 0$ אז נקבל $H(X) = 0$ ← סדר מולחט, יודעים בודאות איזה ערך X מקבל.
- אם $p_1 = p_0 = \frac{1}{2}$ (התפלגות אחידה) אז נקבל: $H(X) = -2\left(\frac{1}{2} log \frac{1}{2}\right) = log2$ - זו האנטרופיה המקסמילית: עובר N ערכים אפשריים ש-X יכול לקבל: $H(X) = logN$, כלומר אי אפשר לדעת איזה ערך X מקבל.

← $H(X) \geq 0$ תמיד.

תרגיל:

מטילים מטבע הוגן (0/1) 3 פעמים. מה האנטרופיה של מספר ה-1?

פיתרון:

נחשב את ההסתברויות של 3 הטלות – יש 2^3 אפשרויות:

מספר ה-1	0	1	2	3
p_i	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$

$$H(X) = -\left(2\frac{1}{8}log\frac{1}{8} + 2\frac{3}{8}log\frac{3}{8}\right) = -\left(\frac{1}{4}log\frac{1}{8} + \frac{3}{4}log\frac{3}{8}\right) = \approx 1.81$$

דחיסה - קוד הופמן

- במחשב כל תו מקודד כמחרוזת של ביטים (0/1).
- לא ניתן לקודד ערך של מ"מ *X* בפחות מ-*H(X)* ביטים בממוצע.
- ניתן לקודד כל תו בקוד באורך קבוע או משתנה.

דוגמא – ננית שיש לנו קובץ עם 100,000 תווים עם אלפבית של 6 אותיות.

- אם נקודד באמצעות קוד שהוא באורך קבוע 0 נצטרך 3 ביטים לייצוג כל תו ← 300,000 ביטים.
- באורך משתנה – ננצל את העובדה שיש אותיות שכיחות יותר , אותן נקודד ע"י קוד קצר יותר, על חשבון אותיות פחות שכיחות.

אותיות	f	e	d	c	b	a
שכיחות (אלפים)	5	9	16	12	13	45
קידוד	1100	1101	111	100	101	0

← מספר ביטים לקידוד הקובץ - 224,000 ביטים (חיסכון של 25% במקום).

איך מקודדים ומפענחים קידוד?

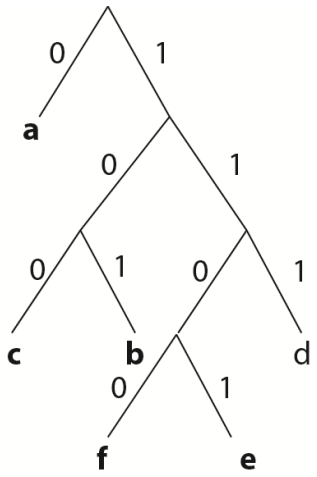
- קידוד – שרשור של הקודים.
- פענוח:
- באורך קבוע – כל n ביטים זה תו.
- באורך משתנה - נשתמש ב*קוד רישות*.

קוד רישות: קוד שבו אין מילה שהיא רישא של מילה אחרת.

בדוגמא הקודמת: 10101010

בודקים האם 1 הוא קידוד – לא. 10 - גם לא. 101 – רק b. ואז 0 – רק a. וכך הלאה.

- דרך נוחה לייצג קוד רישות הוא בעץ בינארי, שבו העלים הם התווים, ומילת קידוד היא מסלול מהשורש לעלה. כאשר פונים שמאלה זה 0 וימינה זה 1. (זה לא עץ חיפוש!)



קוד הופמן - הופמן פיתח אלגוריתם חמדני לבניית קוד רישות אופטימלי.

C – אלפבית, Q – תור קדימויות.

הרעיון – לוקחים בכל איטרציה את 2 התווים בעלי השכיחויות הנמוכות ביותר, מוסיפים לעץ ומכניסים לתור קודקוד פנימי חדש עם שכיחות שהיא סכום השכיחויות שלהם. בסוף מחזירים את השורש של העץ.

Huffman(C):
```
n = |C|
Q.insert(C)
for i=1 to n-1
        new node A
        X = Left(A) = Q.Extract-Min()
        Y = Right(A) = Q.Extract-Min()
        f(A) = f(X) + f(Y)
        Q.Insert(A)
return Q.ExtractMin()
```

סיבוכיות: O(nlogn).