דחיסת נתונים א'

מחברת מתוך ההרצאות של פרופ' דנה שפירא
אוניברסיטת אריאל, 2022

דחיסת נתונים - מחברת הרצאות

נערך ונכתב על-ידי דור עזריה

2022

ספר זה לא נבדק על ידי מרצה, יתכן שימצאו טעויות.

לסיכומים נוספים שלי במדעי המחשב ומתמטיקה:
https://dorazaria.github.io/

מוזמנים לעקוב אחרי 😊:
LinkedIn: https://www.linkedin.com/in/dor-azaria/
GitHub: https://github.com/DorAzaria

תוכן עניינים

מבוא לדחיסת נתונים	3
מידול	14
קודים סטטים	19
קוד שאנון-פאנו - Shannon	30
קוד הופמן - Huffman	35
קוד הופמן קנוני	42
קוד הופמן D-Ary	49
קוד הופמן דינמי	53
עץ שלד הופמן - Skeleton	61
קוד אריתמטי	70
דחיסה מילונית	81
דחיסת LZ77	83
דחיסת LZSS	85
דחיסת LZS	89
דחיסת LZ78	90
דחיסת LZW	92
אלגוריתמי RLE, BWT, MTF	98
אלגוריתם PPM	110
דחיסה דקדוקית (חסרת-הקשר)	130
קודי Tunstall	132

מבוא לדחיסת נתונים

אפשרויות לדחיסת זיכרון

- בעולם התקשורת יש הגבלה פיזית להעברת נתונים "כבדים" ולכן דוחסים את הנתונים כדי להעביר אותם באותו רוחב פס - תמיד נרצה להעביר יותר מהתשתית עצמה.
- דרישה הולכת וגוברת של שמירה אונליין.
- שיפור זמן תגובה של אפליקציות מסוימות.

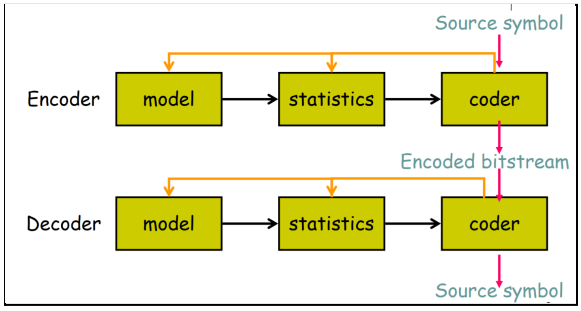
פעולות מערכת הדחיסה

כל מערכת דחיסה מורכבת מ-3 גורמים:

1. **מידול** - יצירת הנחה מראש על הקובץ לפני דחיסה כמו למשל:
   - אם מדובר בטקסט באנגלית אז נצמצם את ASCII לפחות תווים.
   - תלות מותנית כלומר אם יש תו T אז H אז סביר להניח כי אחריו יגיע E.
2. **איסוף סטטיסטיקות**
3. **קידוד**
   - אלפבית המקור
   - אלפבית הערוץ (לרוב נדבר על אלפבית בינארי 0 או 1).
     - למשל קוד אונרי: 0,10,110,....

המקודד והמפענח

- המקודד (Encoder) - אמור לשלוח את המודל שהוא בחר.
- איסוף סטטיסטיקות יעזור לכתיבת הקוד.
- המפענח (Decoder) - לוקח את הביטים ויודע מה ה-א"ב שלו.
- נרצה התאמה מסונכרנת בין המקודד והמפענח באופן חח"ע ועל.
- ככה יהיה אפשר ליצור סנכרון תקין ללא תקלות בין ה-2.
- דחיסת **Lossless** מתאימה לטקסט - נרצה שהקובץ המפוענח יהיה זהה למקורי.
- דחיסת **Lossy** מתאימה למדיה - בשלב הדחיסה ניתן לאבד מידע בלי לפגוע במידע שעובר.



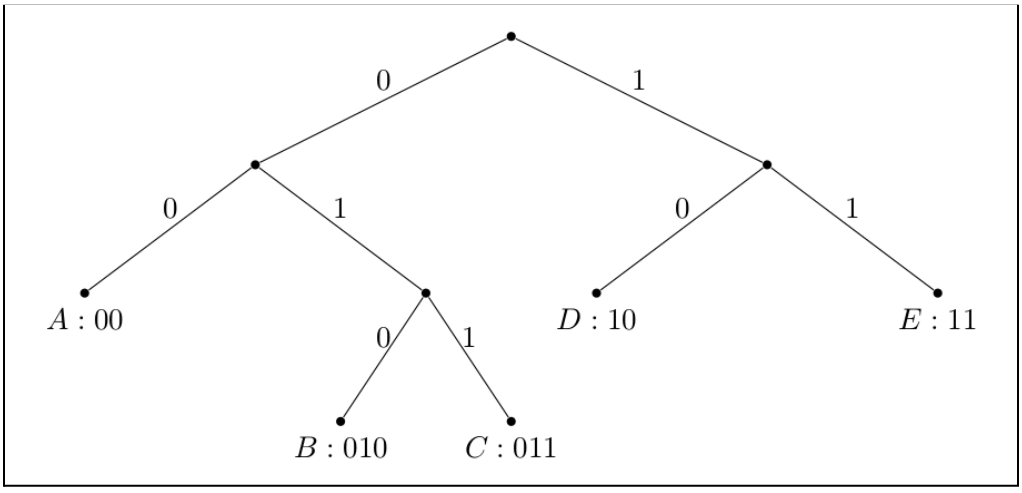
טרמינולוגיה

כלי למדידת יעילות בין אלגוריתמים שונים.

- **אלפבית** של קובץ המקור נסמן כ- $\Sigma = S = [s_1, s_2, ..., s_n]$
- **הסתברות** (התפלגות מסוימת) נסמן $P = [p_1, p_2, ..., p_n]$
  - כאשר $\sum_{i=1}^{n} p_i = 1$ (נניח שסכום ההסתברויות הוא 1).
  - ונניח ש- $p_1 \geq p_2 \geq .... \geq p_n$ (כלומר נניח שהם בסדר יורד).
- נרצה להתאים **מילות קוד**, ונסמן את הקבוצה שלה כך: $C = [c_1, c_2, ..., c_n]$
- **אורך מילת הקוד** נסמן $|C| = [|c_1|, |c_2|, ..., |c_n|]$
- **אורך מילת הקוד הממוצעת** נסמן $E(C, P) = \sum_{i=1}^{n} p_i \cdot |c_i|$.
- **המטרה** היא שאורך מילת הקוד הממוצעת יהיה כמה שיותר קטן כדי שנוכל לדחוס לקובץ יותר קטן.

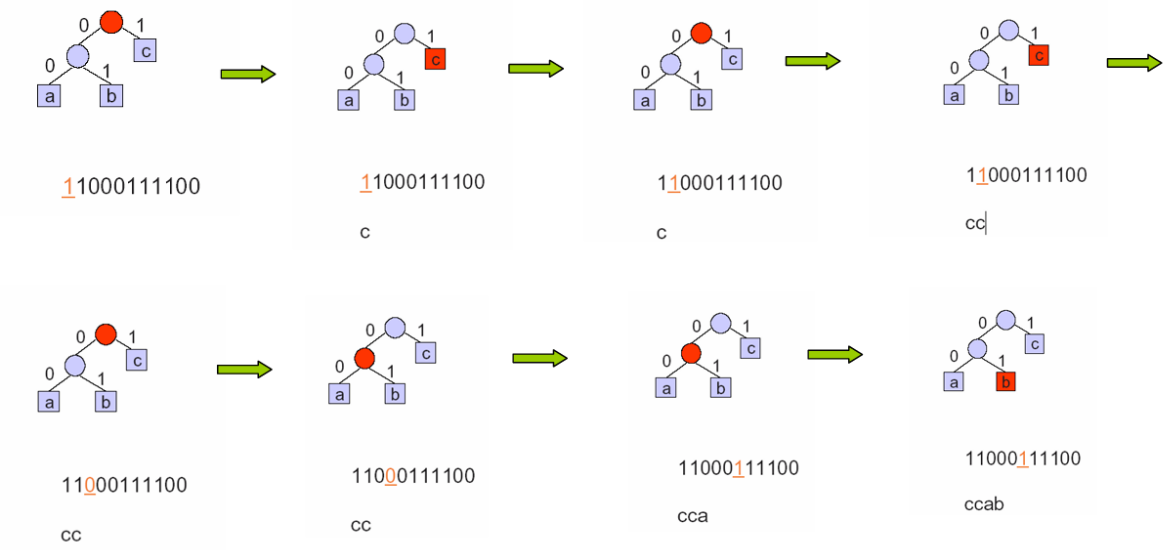
Prefix-Free Codewords (קוד פרפיקסי / קוד חסר רישות)

- אף מילת קוד היא לא רישא של מילת קוד אחרת.
- פענוח של קוד פרפיקסי יותר מהיר.
- למשל עבור מילת הקוד $c_1 = 1011$ אז כל הרישות שלו הם $\epsilon, 1, 10, 101, 1011$.



בעץ מלמעלה אפשר לראות את מילות הקוד בעלים, ואף מילת קוד היא לא רישא של מילת קוד אחרת.

דוגמה לפענוח של מילת קוד Prefix-Free



UD Code Uniquely Decipherable:

- כל הודעה ניתנת לפענוח בצורה יחידה, אנחנו לא רוצים שההודעה הדחוסה שלנו תינתן לפענוח יותר מצורה אחת.
- נשים לב כי $PrefixFree \rightarrow UD$ כלומר כל קוד שהוא Prefix Free הוא גם UD.
  ברגע שהגענו למילת קוד אנחנו יודעים שמילת קוד היא לא הרישא של מילת קוד אחרת ולכן אנחנו יכולים "לשבור" ולפענח אותה בצורה יחידה.
- בנוסף UD **לא גורר** Prefix-Free.
- קוד חסר רישות ⇐ ניתן לבנות עץ שמילות הקוד בעלים ⇐ ניתן לפענוח UD.
- עץ מלא ⇐ עץ שלם.

Complete Code (קוד שלם)

- כל מחרוזת אינסופית למחצה (מחרוזת שהולכת עד אינסוף) ניתנת לפענוח חד-ערכי.
- למשל הקוד ....11 הוא לא ניתן לפענוח כיוון שרצף אחדות אינסופי לא ניתנות למחצה (אין אפס שיכול להגדיר לי "חוצצים" בין כל רצף של אחדות).
- אם ניתן לייצג קוד בתור עץ, אז כל עץ מלא הוא Complete קוד.

תזכורת עצים

- **עץ שלם** הוא עץ שבו לכל צומת מלבד העלים יש שני ילדים.
- **עץ מלא** זה עץ שבו כל רמה, למעט אולי האחרונה, מלאה לחלוטין.

דוגמה

נוסחת אורך מילת הקוד הממוצעת: $E(C,P) = \sum_{i=1}^{n} p_i \cdot |c_i|$

s_i	p_i	C_1	C_2
a	0.67	000	00
b	0.11	001	01
c	0.07	010	100
d	0.06	011	110
e	0.05	100	110
f	0.04	101	111
Expected length		3.0	2.22

- האם קוד 1 ו-2 בטבלה הם prefix-free?

שניהם prefix-free (חסרי רישות) כיוון שאף מילת קוד היא לא תחילית של מילת קוד אחרת.

למשל ב-C_1 אז 000 הוא לא רישא של אף מילת קוד אחרת ב-C_1.

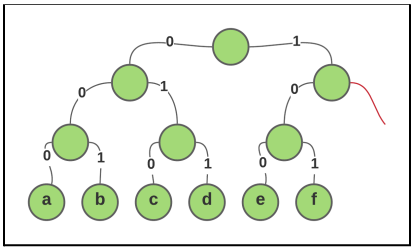
- האם קוד 1 ו-2 בטבלה הם UD?

כן, ראינו כבר שכל קוד פרפיקסי הוא גם UD.

ניתן לדעת זאת גם באמצעות ציור עץ בינארי, אם לכל מילה יש מסלול ייחודי משלו אז הקוד UD.

- האם קוד 1 ו-2 בטבלה הם Complete?

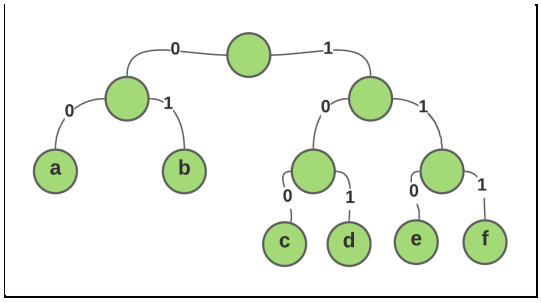
ראינו כבר שכל קוד שניתן לייצג אותו בעץ מלא הוא קוד שלם Complete.



נצייר עבור C_1 עץ בינארי ואכן ניתן לראות כי לא קיבלנו עץ מלא.

הקודקוד שמסומן בצלע באדום אין המשך של 1 וזה אומר שמצאנו מחרוזת אינסופית למחצה שלא ניתנת לפענוח בכלל ובפרט בצורה חד-ערכית.

לכן קוד 1 הוא **לא קוד מלא** כי העץ לא מלא.



נצייר עבור C_2 עץ בינארי ואכן ניתן לראות כי קיבלנו עץ מלא.

בקוד הזה כל רצף ניתן לפענוח ולכן **קוד 2 מלא**.

דוגמה

איזה מהקודים הבאים הוא UD?

$$E(C,P) = \sum_{i=1}^{n} p_i \cdot |c_i|$$

s_i	p_i	C_1	C_2	C_3	C_4
a	0.5	0	0	0	0
b	0.25	0	1	10	01
c	0.125	1	00	110	011
d	0.125	10	11	111	0111
Expected length		1.125	1.25	1.75	1.875

אם קוד הוא לא UD, צריך להראות מחרוזת עם 2 פירושים או יותר.

- נבחן את C_1:

  נשים לב כי a ו-b יש את אותו הקידוד וזה סותר את ההגדרה של UD, לכן C_1 **לא** UD.

- נבחן את C_2:

  נשים לב שהוא לא קוד פרפיקסי, אבל זה לא בהכרח אומר שהוא לא UD!

  נשים לב כי את c אפשר לפענח עם פעמיים a ולכן אין צורת פענוח ייחודית ל-c ולכן C_2 קוד **לא** UD.

- נבחן את C_3:

  הקוד הוא **כן** UD כיוון שהוא גם פרפיקסי.

- נבחן את C_4:

  נשים לב שהוא לא קוד פרפיקסי, אבל זה לא בהכרח אומר שהוא לא UD!

  במקרה זה, כל תו ניתן לפענח באופן ייחודי משלו, כלומר לא קיימת אות x כלשהי, שניתן לפענח אותה על סמך y כלשהו ולכן C_4 הוא **כן** קוד UD.

קוד מיידי

- נגיד שקוד הוא **מיידי** אם המפענח יודע את הפיענוח של אותה מילה נוכחית באופן שלם ומיידי.
- כלומר, כשאני נמצא במילת הקוד אני כבר יודע את הפענוח שלו.
- קוד prefix-free ⇔ קוד מיידי.
- קוד UD לא בהכרח מיידי כיוון שיכול להיות שקיים קוד UD שלא בהכרח פרפיקסי ולכן הוא לא מיידי.
- למשל הקוד 01111111 הוא לא מיידי כי אנחנו לא יודעים מתי "לשבור" את המחרוזת בגלל רצף ה-1.
- הקוד C_3 בטבלה הקודמת הוא מיידי כיוון ש: $\underline{0}1\underline{0}11\underline{0}111$ ניתן לשבירה באפסים.

UD Test

- נועד כדי לבדוק האם קוד הוא UD.
- נניח שיש לנו 2 מילות קוד בינאריות a ו-b כאשר $|a| = k$ ביטים וגם $|b| = n$ ביטים כאשר $k < n$.
  אם ה-k ביטים הראשונים של b הם זהים ל-a אז a נקרא ה-**רישא** של b.
  ושאר ה- (n-k) ביטים נקראים ה-**סיפא (dangling suffix)** של b.
- כלומר, אם ה-prefix-ים שווים, אז מה שנשאר נקרא **dangling suffix**.
- לדוגמה: $a = 010$ ו-$b = 01011\underline{11}$, אז ה-suffix dangling הוא 11 (לאחר השמטת הרישא מ-b).

אלגוריתם - UD Test Algorithm	
נגדיר רשימה חדשה L ונכניס לתוכה את כל מילות הקוד הקיימות ב-C.	1:
עוברים על כל זוגות מילות קוד ב-L ובודקים האם אחת היא הרישא של האחרת.	2:
כל עוד a היא רישא של b, אנחנו מוסיפים את ה-dangling לרשימה L עד אשר מתקיים אחד מהתנאים הבאים:	3:
1. אם ה-dangling שהוספנו ל-L היא כבר מילת קוד מקורית ב-C - אז הקוד לא UD ונעצור.	4:
2. אם ה-dangling שהוספנו ל-L לא מילת קוד מקורית אז נמשיך הלאה ונעצור את האלגוריתם אם עברנו על כל צירופי הזוגות האפשריים ונחזיר כי הקוד הוא כן UD.	5:

לדוגמה

מילות הקוד $C = \{0,\ 01,\ 11\}$.

נשמור ברשימה חדשה L בצד את כל מילות הקוד ונוסיף אליה כל פעם ה-dangling שמצאנו.

נעבור על כל זוגות מילות הקוד, נתחיל עם '0' ונראה כי היא רישא של '01' ולכן ה-dangling הוא 1.

לכן נוסיף ל-L את '1' כלומר $L = \{0,\ 01,\ 11,\ 1\}$, מאחר ו-'1' הוא לא חלק מ-C אז נמשיך...

נשים לב כי '1' הוא רישא של '11' לכן נוסיף את '1' ל-L אך כיוון שהוא כבר קיים והוא לא מילה מקורית מתוך C אז נמשיך לחפש.. לא נמצאו עוד dangling suffix מלבד '1' ולכן Cהוא **כן** קוד UD.

לדוגמה

מילות הקוד $C = \{0,\ 01,\ 10\}$.

נשמור ברשימה חדשה L בצד את כל מילות הקוד ונוסיף אליה כל פעם ה-dangling שמצאנו.

נעבור על כל זוגות מילות הקוד, נתחיל עם '0' ונראה כי היא רישא של '01' ולכן ה-dangling הוא 1.

לכן נוסיף ל-L את '1' כלומר $L = \{0,\ 01,\ 10,\ 1\}$, מאחר ו-'1' הוא לא חלק מ-Cאז נמשיך...

נשים לב כי '1' הוא רישא של '10' ולכן נוסיף את '0' שהוא dangling לתוך הרשימה של L.

מאחר וה-'0' שהוספנו ל-L היא מילה מקורית ב-C אז נעצור ונגיד כי C הוא **לא** קוד UD.

אלגוריתם Sardinas-Patterson

מדובר באותו אלגוריתם ממקודם רק באופן פורמלי (אלגוריתם לבדיקת UD).
בהינתן מחרוזות S וגם T.
מילות הקוד שנמצאות ב-T ויש להם רישא ב-S נסמן כך:

$$S^{-1}T = \{ d \mid ad \in T,\ a \in S \}$$

Sardinas-Patterson Algorithm

```
1:  i ⇐ 1
2:  S_1 ⇐ C^{-1}C − {ε}
3:  while true:
4:        S_{i+1} ← C^{-1}S_i ∪ S_i^{-1}C
5:        i = i +1
6:        if  ε ∈ S_i or c ∈ S_i:
7:              for c in C:
8:                    print "not UD"
9:                    exit
10:       else if ∃j < i such that S_i = S_j:
11:             print "UD"
12:             exit
```

קודי יתרות מינימליים

בהינתן התפלגות מסוימת, לא קיים קוד שייתן אורך קוד יותר קטן ממה שנגיע אליו.
המטרה שלנו היא למצוא קוד שבו אורך הממוצע שלו הוא מינימלי.

<u>פורמלי:</u>
יהי $E(C,P)$ אורך קוד ממוצע עבור C, אזי C הוא קוד בעל יתירות מינימלי עבור ההסתברות P אם לכל קידוד C' מתקיים:

$$E(C,P) \leq E(C',P)$$

כמות האינפורמציה

האינפורמציה זה רמת האי-וודאות.

כמות האינפורמציה היא בעצם כמות הביטים המינימלית שניתן לייצג עבור s_i.

אנחנו נרצה להשיג את הדחיסה הטובה ביותר ונעשה זאת על ידי הסרה של מידע מיותר.

כלומר, ככל שכמות האינפורמציה קטנה יותר ככה יהיה לנו פחות אי-וודאות כי אנו פחות נדרוש מידע לפענוח.

משפט הקידוד של Shannon (שאנון)

כמות המידע המוכלת בסימן s_i בהסתברות p_i היא:

$$I(s_i) = -\log_2(p_i)$$

כאשר $-\log_2(p_i)$ היא כמות הביטים המינימלית כדי לייצג את s_i.

הפונקציה $I(s_i)$ מציגה לנו את רמת חוסר הוודאות שלנו:

- אם $p_i = 1$ אז $I(s_i) = 0$ כלומר שאין לנו חוסר ודאות.
- אם $p_i = 0$ אז $I(s_i) = \infty$ כלומר יש לנו חוסר ודאות מוחלט.

כלומר, אם מאורע הוא וודאי, כלומר ההסתברות שלו שיקרה זה $p_i = 1$, אז כמות האינפורמציה היא 0 כלומר ש- $I(s_i) = 0$ והכוונה, שכבר הכל ידוע לנו, אז גודל ההפתעה הוא אפסי.

מאחר ו- $0 \le p \le 1$ אז מכאן ה-"מינוס" בנוסחה של הלוג, זאת כדי לקבל תוצאה חיובית.

לדוגמה: $p = \frac{1}{2}$ אז נציב ונקבל:

$$-\log_2(\tfrac{1}{2}) = -\log_2(2^{-1}) =$$
$$= -(-1)\cdot\log_2(2) = 1$$

התוצאה "1" אומרת לנו שאנו צריכים רק ביט 1 כדי לקודד סימן בהסתברות $p = \frac{1}{2}$.

תכונות שאנון

1. אם $p_i = 1$ אז $I(s_i) = 0$ - כלומר, מאורע ודאי ⇐כמות האינפורמציה = 0.
2. אם הסדרה $s_i s_j$ מתקיימת בהסתברות $p_i p_j$ (מודל בלתי תלוי) אז:

$$I(s_i s_j) = I(s_i) + I(s_j)$$

כלומר:

$$I(s_i s_j) = -\log p_i + \left(-\log p_j\right) = I(s_i) + I(s_j)$$

3. אורך מילת הקוד = כמות האינפורמציה.
4. ככל שההסתברות יותר גדולה אז כמות האינפורמציה קטנה יותר.

אנטרופיה

- אנטרופיה זה בעצם ממוצע משוקלל של כמות האינפורמציה.
- אנטרופיה היא מדד לאי-חיזוי.
- האנטרופיה של מסר היא במובן מסוים מדד לכמות האינפורמציה שהוא באמת מכיל.
- $H(P)$ מייצג את המספר הממוצע של סיביות מידע לכל סמל מהמקור.

בהינתן הסתברות בהתפלגות P, נגדיר:

$$H(P) = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i$$

לכל קוד C מתקיים:

$$H(P) \leq E(C,P)$$

כלומר, נראה שהאנטרופיה מהווה חסם תחתון ל-אורך קוד ממוצע.

- האנטרופיה מהווה חסם תחתון כלומר *אי אפשר לדחוס יותר טוב מהאנטרופיה.*

- אם נרצה לקודד מחרוזות ארוכות יותר ויותר של סמלים כלשהם, נוכל למצוא קודים שהביצועים שלהם (מספר סיביות ממוצע לסמל) מתקרבים ל-$H(P)$ - שזה בדיוק מה שמשפט הקידוד של שאנון עושה.

אי שוויון Kraft-McMillan

כמה קצר יכול להיות קוד שהוא UD?

אי-שוויון קראפט מתאר תנאי מספיק והכרחי לשיוך קבוצת מילים לצמתי עץ, כך שלא תשויך יותר ממילה אחת לאורך כל מסלול היוצא מהראש.

כאשר מילים משויכות לעצים, אפשר לבנות קוד למילים. בהינתן מילה, נמצא את הצומת המתאים למילה, ונתאר את המילה בעזרת המסלול המוביל מראש העץ למילה.

משפט אי שוויון Kraft-McMillan

יהא $C = [c_1, c_2, ..., c_n]$ להיות קוד עם n מילות קוד באורך: $|C| = \left[\left|c_1\right|, \left|c_2\right|, ..., \left|c_n\right|\right] = n$.

אם C הוא UD אז:

$$K(C) = \sum_{i=1}^{n} 2^{-\left|c_i\right|} \leq 1$$

זה בעצם מדד איכות עבור קידוד, ככל ש- $K(C)$ קרוב יותר ל-1, ככה הוא איכותי יותר.

אם $K \leq 1$ אז ניתן לבנות קוד חסר רישות עם אותם אורכים (UD).

משפט

אם מתקיים $K(C) \leq 1$ עבור קוד C כלשהו (לא בהכרח UD) אזי קיים C' אחר, כך ש:

1. $E(C,P) = E(C',P)$
2. $|C'| = |C|$
3. C' הוא קוד חסר רישות (prefix-free).

במילים אחרות, קיים ***קוד רגעי*** (חסר רישות) אופטימלי (בפרט או ייחודי).

משפט

אם קוד C הוא ייחודי אז $K(C) \leq 1$ (זה תנאי הכרחי לייחודיות).

משפט

אם עבור קוד C כלשהו מתקיים:

$$K(C) = \sum_{i=1}^{n} 2^{-|c_i|} > 1$$

אזי הוא לא קוד חסר רישות ובפרט לא ייחודי.

דוגמה לאי-שוויון Kraft

דוגמה שלא הוצגה בהרצאה אך היא עוזרת להבין את היתרון לשימוש באי-שוויון זה.

כאשר מילים *משויכות* לעצים, אפשר לבנות קוד למילים.

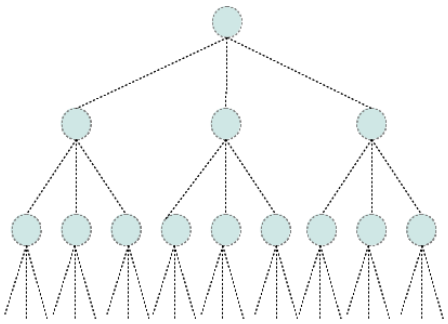
בהינתן מילה, נמצא את הצומת המתאים למילה, ונתאר את המילה בעזרת המסלול המוביל מראש העץ למילה.

נניח שיש לנו אלפבית בעל שלוש אותיות, לדוגמה $\Sigma = \{0,1,2\}$.

נניח שיש לנו קבוצה בת חמש מילים מעל הא"ב הזה, שאורכיהן הם $|C| = \{2,1,2,2,2\}$.

עלינו למצוא שיוך לעץ המוצג מלמטה כך ש:

- ארבעה צמתים מגובה 2 ישויכו
- צומת אחת בגובה 1 ישוייך
- אף צומת לא יהיה צאצא של צומת משויך אחר

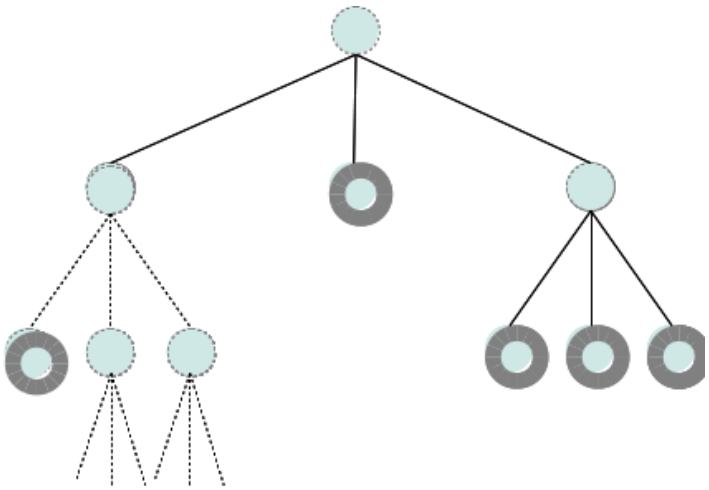


לכן, במקרה זה נקבל לפי אי-שוויון Kraft ש:

$$K(C) = 3^{-2} + 3^{-1} + 3^{-2} + 3^{-2} + 3^{-2} = 0.77777 < 1$$

ולכן אי-שוויון קראפט מבטיח שיש שיוך מתאים.

כאשר ה-3 זה בעצם נוסחה של ה-Kraft שלמדנו רק שהפעם $|\Sigma| = 3$ כלומר עבור עץ טרינרי ולא בינארי כאשר $K(C) = \sum_{i=1}^{n} |\Sigma|^{-|c_i|} \leq 1$.



מידול

נרצה למצוא אלגוריתמי דחיסה ופרישה של נתונים.
זאת כדי לשמור מקום (כלומר הדחיסה תחסוך לנו בגודל הזיכרון ששומר את הנתונים).

- דחיסת **Lossless** מתאימה לטקסט - נרצה שהקובץ המפוענח יהיה זהה למקורי.
- דחיסת **Lossy** מתאימה למדיה - בשלב הדחיסה ניתן לעבד מידע בלי לפגוע במידע שעובר.

משפט 1

<u>קיים</u> קוד מיידי עם אורכי מילות קוד $l_1 \leq l_2 \leq ... \leq l_n$ אמ"מ $\sum_{i=1}^{n} 2^{-l_i} \leq 1$.

הוכחה 1

כיוון א': עבור כל מילת קוד באורך l_i אנחנו מורידים $2^{l_n - l_i}$ עלים.

לכן עבור כל מילות הקוד נקבל:

$$\sum_{i=1}^{n} 2^{l_n - l_i} \leq 2^{l_n} \quad \backslash : 2^{l_n}$$

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1$$

וקיבלנו:

$$K(C) \leq 1$$

כיוון ב': נבחר צומת ברמה l_1,

$$2^{l_n - l_1} < 2^{l_n}$$

$$2^{l_n - l_1} + 2^{l_n - l_2} < 2^{l_n} \Rightarrow ...$$

∎

משפט 2

תנאי הכרחי לקוד עם $l_1 \leq l_2 \leq ... \leq l_n$ אורכי מילות קוד להיות קוד UD הוא כאשר $\sum_{i=1}^{n} 2^{-l_i} \leq 1$.

נסמן d_i להיות מספר מילות הקוד באורך i ומכאן $\sum_{i=1}^{n} 2^{-l_i} = \sum_{i=1}^{m} d_i \cdot 2^{-i}$

- תנאי הכרחי שהקוד יהיה UD הוא כאשר $K(C) \leq 1$.

הוכחה 2

$$\left(\sum_{i=1}^{m} d_i \cdot 2^{-i}\right)^k = \left(d_1 \cdot 2^{-1} + \ldots + d_m \cdot 2^{-m}\right)^k = \sum_{j=k}^{k \cdot m} N_j \cdot 2^{-j}$$

נסמן N_j זה מספר ההודעות שיש לי באורך j.

$$N_j \leq 2^j$$

$$\left(\sum_{i=1}^{m} d_i \cdot 2^{-i}\right)^k \leq \sum_{j=k}^{k \cdot m} 1 \leq k \cdot m \quad \backslash \cdot ()^{1/k}$$

$$\left(\sum_{i=1}^{m} d_i \cdot 2^{-i}\right) \leq \left(\sqrt[k]{k \cdot m}\right)_{k \to \infty} = 1$$

∎

מה זה מודל?

למידה או הנחות לגבי מבנה הנתונים הנדחסים.

מודל Zero Order

נניח לשם הפשטות שמדובר במודל מרקוב מסדר אפס ונציג כאן שלושה מודלים:

1. **מערכת Static Compression**
   - המודל כי פשוט.
   - מודל סטטי, אין לנו שום הנחות מקדימות, לא צריך להעביר שום מידע מקדים למפענח.
   - נתבסס על טבלת ה-**ASCII** שבה יש 256 כניסות, המפענח יודע את הטבלה **מראש**.
   - כל מילת קוד היא מצוינת ב-8 סיביות.
   - <u>האנטרופיה</u>:
   - נניח שההסתברות של כל תו בטבלת ASCII היא הסתברות זהה וההתפלגות היא אחידה וההסתברות היא $p_i = \frac{1}{256}$, אז האנטרופיה היא:

$$H(P) = -\sum_{i=1}^{256} \frac{1}{256} \cdot \log_2\left(\frac{1}{256}\right) = 8$$

האנטרופיה זה למעשה ממוצע משוקלל של כמויות המידע והתוצאה היא 8 ביטים.
ואכן כל מילת קוד מצוינת ב-8 ביטים לפי ASCII.

2. מערכת Semi-Static Compression (מודל סטטי למחצה)

- אנחנו תלויים בהודעה אותה אנחנו הולכים לקודד.

Bring me my bow of burning gold!
Bring me my arrows of desire!
Bring me my spear!
O clouds unfold!
Bring me my chariot of fire!

אם נסתכל על השיר הזה נקבל שיש לו בסה"כ 25 תווים בשימוש.

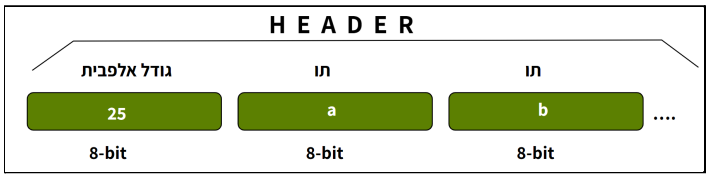
- המודל הזה אומר - למה לי להשתמש בקוד הASCII שמכיל 256 תווים, בואו נתמקד רק באלפבית שממנו בנוייה ההודעה.
- כלומר, יש פה Preprocessing להכנה של מספר התווים (בשונה מהקודם).
- <u>האנטרופיה</u>:

כל תו בעל אותה ההסתברות $p_i = \frac{1}{25}$ - למרות שאנחנו רואים למשל שה-m מופיע הרבה יותר פעמים משאר תווים שמופיעים פחות. לכן, ההסתברות לא נכונה בהודעה הזאת אבל אנחנו נסתפק בה בכל זאת.

$$H(P) = -\sum_{i=1}^{25} \frac{1}{25} \cdot \log_2\left(\frac{1}{25}\right) = 4.64$$

זה אומר שקיבלנו 4.64 ביטים (זה אומר שכמות המידע בכל ביט היא קצת פחות מ-5 סיביות)

- מה אנחנו צריכים להעביר למפענח? :
  - אנחנו צריכים להגיד לו מי הם התווים שמשתתפים ומה הסדר שלהם כדי שהמפענח יוכל לעשות את ההשמה של התו באלפבית לפי הקוד.
  - האלפבית יתפוס לכל היותר 8 סיביות כי אנחנו מניחים שהאלפבית שלנו הוא לכל היותר 256 תווים.



  - בתמונה אפשר לראות כי ה-Header שומר בראש את גודל האלפבית כדי שהמפענח ידע את זה והוא תופס 8 ביטים.
  - תיאור התווים הוא $25 \cdot 8 = 200$ ביטים.
  - לכן סה"כ נקבל $200 + 8 = 208$.
  - אם נעבור על כל ההודעה בדוגמה, אורכה כולל שורה חדשה וכולל רווחים הוא 128 תווים (זה גודל ההודעה).
  - לכן אורך מילת הקוד הממוצעת יהיה: $4.64 + \frac{208}{128} = 6.27$

3. מודל סטטי למחצה עם הסתברויות עצמאיות

הפעם ההסתברות לא אחידה, ותחושב ע"י כמות הפעמים שהתו הופיע חלקי גודל ההודעה.

- מודל שנכנס עמוק יותר להודעה.
- נגדיר v_i כמות ההופעות של התו ה-s_i.
- נגדיר m להיות גודל ההודעה.
- ההסתברות למופע של כל תו יהיה $p_i = \frac{v_i}{m}$.
- אם אנחנו רואים שתו מופיע הרבה אנחנו נרצה לתת לו מילת קוד קצרה יותר.
- אנחנו אוספים את ההסתברויות כדי לתת מילות קוד קצרות יותר לתווים תדירים יותר ואם ניקח את ההודעה הבאה:

Bring me my bow of burning gold!
Bring me my arrows of desire!
Bring me my spear!
O clouds unfold!
Bring me my chariot of fire!

s_i	p_i	s_i	p_i	s_i	p_i
'\n'	4/128	f	5/128	s	4/128
' '	22/128	g	6/128	t	1/128
!	5/128	h	1/128	u	3/128
B	4/128	i	8/128	w	2/128
O	1/128	l	3/128	y	4/128
a	3/128	m	8/128		
b	2/128	n	7/128		
c	2/128	o	9/128		
d	4/128	p	1/128		
e	8/128	r	11/128		

- האנטרופיה לפי הסימונים שלנו: $H(P) = -\sum_{i=1}^{n} \frac{v_i}{m} \cdot \log(\frac{v_i}{m}) = 4.22$.
- אם נכפיל ב-m את האנטרופיה נקבל כמה תעלה לנו ההודעה (כמה הקידוד יתפוס) במקרה הטוב ביותר.
- מה המפענח צריך במקרה זה?
  - 8 ביטים עבור גודל האלפבית.
  - 25*8 מהם התווים עצמם.
  - 25*4 מה התדירות/ההסתברות של כל תו.
  - לכן ה-Header יעלה לנו $8 + 25 \cdot 8 + 25 \cdot 4 = 308\ bit$.
- לכן אורך מילת הקוד הממוצעת היא: $4.22 + \frac{308}{128} = 6.63$
- כלומר, בהודעה הקצרה הספציפית הזאת הוא **גרוע** בהשוואה למודל סטטי למחצה. זה כי אנחנו נתנו מודל יותר מדי מורכב להודעה קצרה מדי.

עקרון אורך הודעה מינימלי

כשאנחנו בודקים את הקובץ הדחוס אנחנו צריכים לבדוק את 2 הרכיבים יחד:

- התיאור של המודל
- הקידוד עצמו

(אסור לנו לשכוח את ה header).

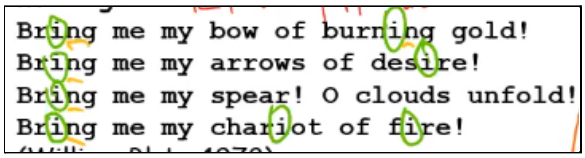
מסקנה - ככל שה-Header יהיה מפורט יותר ככה הקידוד יהיה יעיל יותר.

מודל First Order

מודל מרקוב מסדר ראשון - אנחנו קובעים את ההסתברות של התו הבא על סמך התו הנוכחי.

- במודל זה מניחים תלות בין התווים.

לדוגמה:



- אחרי התו 'i' מגיע 'n' בהסתברות ⅝.
- אחרי התו 'i' מגיע 'o' בהסתברות ⅛.
- אחרי התו 'i' מגיע 'r' בהסתברות 2/8.

סיכום הפרק

- **מודל סטטי** - נקבעת לפני הקידוד ע"פ ASCII ונגדיר הסתברות אחידה לכל תו (256 תווים).
  המערכת הזאת לא צריכה Preprocessing כי היא יודעת כבר את טבלת ה-ASCII מראש.
  בהסתברות אחידה $p_i = \frac{1}{256}$ ו- $H(P) = -\sum_{i=1}^{256} \frac{1}{256} \cdot \log_2\left(\frac{1}{256}\right) = 8.0$
  אז יעלה לנו להעביר: $E(C,P) = H(P)$.

- **מודל סטטי למחצה** - דורש Preprocessing.
  נספור כמה תווים (אותיות) שונים יש (נסמן ב-N) ונניח שההסתברות אחידה $p_i = \frac{1}{N}$ וראינו ש:
  $$H(P) = -\sum_{i=1}^{N} \frac{1}{N} \cdot \log_2\left(\frac{1}{N}\right).$$
  אז יעלה לנו להעביר: $E(C,P) = H(P) + \frac{8 \cdot |\Sigma| + 8}{|Text\ Size|}$.

- **מודל סטטי למחצה עם הסתברויות עצמאיות** - דורש Preprocessing.
  הפעם ההסתברות לא אחידה, ותחושב ע"י כמות המופעים v_i של התו s_i חלקי גודל ההודעה m וראינו ש: $p_i = \frac{v_i}{m}$. וגם ראינו ש: $H(P) = -\sum_{i=1}^{n} \frac{v_i}{m} \cdot \log(\frac{v_i}{m})$.
  אז יעלה לנו להעביר: $E(C,P) = H(P) + \frac{8 \cdot |\Sigma| + 8 \cdot |p| + 8}{|Text\ Size|}$.

קודים סטטים

- שימוש הקטן ביותר בהסתברויות
- נשתמש בקוד קבוע כדי שהוא יהיה מהיר
- דחיסות פחות טובות

אנחנו לא נעשה עיבוד מוקדם של כל הקובץ ומעבר עליו לפני הדחיסה *ולכן ה-header יהיה ריק* כי אנחנו נשתמש בקודים שהם ידועים ונראה למעשה שנעשה שימוש קטן של ההסתברויות כלומר במקרה שהקובץ כן ידוע לנו מראש אנחנו נמיין את התווים באלפבית לפי התדירות שלהם מהגבוהה לנמוכה אבל מלבד זה אנחנו לא נתאר את ההסתברויות עצמן.

אלפבית אינסופי

הרעיון: תחשבו שאנחנו מקבלים קובץ ואנחנו מתחילים לדחוס אותו ופתאום יש איזה תו שלא חשבנו שהוא יופיע אז פשוט נתן לו את המילת הקוד הפנויה הבאה.

- בניגוד למה שראינו עד עכשיו שהאלפבית מסתיים ב-S_n נעבוד הפעם באינסוף.
- נסמן את האלפבית $S = [s_1, s_2, \ldots]$, כלומר לא צריך לדעת מראש את גודל הא"ב שלנו.
- ההסתברויות בצורה יורדת $p_1 \geq p_2 \geq \ldots > 0$.

קוד אונרי (Unary)

- מילת הקוד שמופיעה במקום x תקודד עם x-1 אחדות (ביטים דלוקים), ואז "0" שמייצג את סוף מילת הקוד.
- לדוגמה: $0,\ 10,\ 110,\ 1110, \ldots$.
- קוד אינסופי - לא צריכים לדעת מראש מה גודל האלפבית.
- אם אנחנו יודעים מראש שהאלפבית הוא סופי, אנחנו יכולים לחסוך קצת בקידוד.
- מתי הקוד בעל יתירות מינימלית? - כלומר, מתי קוד אונרי הוא קוד יעיל כך שכל קוד C' יקיים:

$$E(C_{Unary}, P) \leq E(C', P)$$

זה כאשר ההסתברויות הן חזקות של ½ או ¼ אנחנו מקבלים שהקוד האינסופי הוא בעל יתירות מינימלית.

- בקוד אינסופי, כאשר כמות האינפורמציה (ביטים מינימלית לייצוג תו) תהיה שווה לאורך מילת הקוד - הקוד האונרי יהיה ללא יתירות (אורך מילה ממוצעת מינימלי).
- בקוד סופי, פשוט נקצץ את הביט האחרון, ה-0 החוצץ בסוף, בכדי שיהיה קוד ללא יתירות.

קוד בינארי

קוד בינארי פשוט

- קוד בעל אורך קבוע.
- עבור א"ב בגודל n, אורך מילת הקוד יהיה:

$$\lceil \log_2 n \rceil$$

- נשים לב שאם יש לנו $n = 2^k$ אז נקבל שאורך מילות הקוד הוא $k = \lceil \log_2 2^k \rceil = \lceil \log_2 n \rceil$.
- לדוגמה, $n = 5$ אז $2^2 < 5 < 2^3$ לכן אנחנו צריכים 3 סיביות לכל היותר.

קוד בינארי מינימלי

- לא מחייבים קוד בעל אורך קבוע.
- אפשר אורכים שונים במילות הקוד וכך לייעל את הקידוד.
- מילות הקוד הארוכות יהיו באורך:

$$\lceil \log_2 n \rceil$$

- מילות הקוד הקצרות יהיו באורך:

$$\lceil \log_2 n \rceil - 1$$

  - כלומר, נאפשר מקסימום הפרש של סיבית אחת בין מילות הקוד הקצרות למילות הקוד הארוכות.
- לדוגמה, $n = 5$ אז $2^2 < 5 < 2^3$ יהיו לנו מילות קוד ארוכות באורך $\lceil \log_2 5 \rceil = 3$ ומילות קוד קצרות באורך $\lfloor \log_2 5 \rfloor = 2$.
  - כמה מילות קצרות יהיו לנו?: $2^{\lceil \log_2 n \rceil} - n = 2^3 - 5 = 8 - 5 = 3$
    לכן יהיו לנו 3 מילות קוד קצרות באורך 2.
  - כמה מילות קוד ארוכות יהיו לנו? - $5 - 3 = 2$
    לכן יהיו לנו 2 מילות קוד ארוכות באורך 3.

אבחנה

אם יש לנו $n = 2^k$ (חזקה של 2) אז נקבל שאורך מילת הקוד הוא k. כאשר n הוא מספר הסיביות.

קוד אליאס Elias

שתי שיטות קידוד (C_γ - גמא, C_δ - דלתא) כך שהביטים יהיו בסדר גודל לוגריתמי של ההודעה.

קוד אליאס הוא מעין מיזוג של בינארי ואונרי.

מילת הקוד במקום ה-x היא באורך ביטים של:

$$O(\log(x))$$

<u>הקוד הראשון: C_γ גמא:</u>

- חלק 1:
  - כותבים בקוד אונרי את מספר הסיביות, בייצוג הבינארי של x.
  - חלק זה יקח לנו $\lfloor \log_2(x) \rfloor + 1$.
- חלק 2:
  - כותבים את הייצוג הבינארי של x, ללא ה-1 המוביל (מורידים את ה"1" המוביל).
  - לכן חלק זה יקח לנו $\lfloor \log_2(x) \rfloor$
- ומכאן סה"כ אם נחבר את 2 החלקים נקבל $2\lfloor \log_2(x) \rfloor + 1$ סיביות שזה $O\left(\log_2(x)\right)$

Elias Code C_γ Algorithm

```
1: Elias_Code_Cγ(x) {
2:       w = Binary(x);
3:       out1 = Unary(|w|);
4:       out2 = w - (first 1); // remove first '1'
5:       return out1*out2; // merge two strings
6: }
```

<u>דוגמה</u>: נתון לנו $x = 25$.

```
2:       w = Binary(25)=11001;
3:       out1 = Unary(|w|)=Unary(5)=11110;
4:       out2 = w - (first 1) = 1001 = 1001;
5:       return out1*out2 = 11110*1001 = 111101001
```

דוגמה: נתון לנו $x = 50$.

```
2:      w = Binary(50)=110010;
3:      out1 = Unary(|w|)=Unary(6)=111110;
4:      out2 = w - (first 1) = 110010 - 10010 = 10010;
5:      return out1*out2 = 111110*10010 = 11111010010
```

- בהינתן מספר לא חסום, איך נמצא את המספר הזה?
  - כשנרצה למצוא תחום שבו הוא חסום נשתמש בקוד אונרי.
  - אם זה קטן מ-2^k אם לא אז נוסיף "1" לייצוג האונרי משמאל.
  - אם התשובה היא כן, אז נניח זה בין 4 ל-7 (באיור משמאל) אז נמשיך בחיפוש בינארי (כי 0 או 1 זה כמו קטן או גדול).
  - האקספוננציאלי מתאים לקוד האונרי והחיפוש הבינארי מתאים לקוד הבינארי.

x	Elias C_γ Code
1	0
2	10 0
3	10 1
4	110 00
5	110 01
6	110 10
7	110 11
8	1110 000
9	1110 001

Also "expo Bentl

- **הקוד השני C_δ דלתא:**

בצד שמאל נכתוב את מספר הביטים כייצוג בינארי אך הפעם כ- c_γ

ומצד ימין שוב את הבינארי בלי ה-1 המוביל.

  - חלק ראשון:
    - מקודדים את מספר הסיביות ב-x על ידי C_γ.
    - כלומר $1 + 2 \cdot \lfloor \log_2\left(\log_2(2x) \right) \rfloor$
  - חלק שני:
    - קוד בינארי ב-x בטווח שנקבע על ידי C_γ.
  - סה"כ נקבל $\lfloor \log_2(x) \rfloor + \lfloor \left(\log_2(2x) \right) \rfloor + 1 + 2 \cdot \lfloor \log_2\left(\log_2(2x) \right) \rfloor$.

x	Elias C_γ Code	Elias C_δ Code
1	0	0
2	10 0	100 0
3	10 1	100 1
4	110 00	101 00
5	110 01	101 01
6	110 10	101 10
7	110 11	101 11
8	1110 000	11000 000
9	1110 001	11000 001

- יש לנו מספרים ש-C_δ גדול יותר מ-C_γ
  - למשל עבור 1 תופסים אותו מספר סיביות.
  - עבור 2,3 ה-C_δ יותר ארוך מ-C_γ
  - עבור 8,9 פתאום ה-C_δ יותר גרוע מ-C_γ

- יש מספר בודדים שהם C_δ גרוע יותר C_γ

אנחנו צריכים לשאול את עצמנו מתי C_γ גרוע יותר מהקוד האונרי.

לכן נשאל - מתי הקוד האונארי קצר יותר מ-C_γ?

יש 2 מספרים שהם הייצוג האונארי דווקא יותר טוב מהייצוג של C_γ, למשל הייצוג האונארי של 2 זה 10

ומצאנו דוגמה כזאת, גם ב-4 הייצוג האונארי יתפוס 4 סיביות אבל ב-C_γ הוא תופס 5 סיביות

מסקנה - עבור ערכים גדולים קודי אליאס הם טובים יותר באופן אקספוננציאלי מקודים אונארים.

Elias Code C_δ Algorithm

```
1: Elias_Code_C_δ(x) {
2:       w = Binary(x);
3:       out1 = Elias_Code_C_γ(|w|);
4:       out2 = w - (first 1); // remove first '1'
5:       return out1*out2; // merge two strings
6: }
```

דוגמה: נתון $x = 25$.

```
2:       w = Binary(25)=11001;
3:       out1 = Elias_Code_C_γ(5)=110 ⊥01 = 11001;
4:       out2 = w - (first 1)= ⊥1001;
5:       return out1*out2 = 11001*1001= 110011001;
```

שאלה חשובה - למה שנעצור כאן ולא נמשיך לקוד C_ε נוסף?

אם נעבוד עם קוד C_ε שתלוי ב-C_δ אנחנו נראה שהיא לא חוסכת לנו בכלום אפילו לא למספר ממש גדול.

לכן הקוד הזה לא יהיה בשימוש ולכן לא הגדרנו אותו, נראה לדוגמה עבור $x = 10^9 \approx 2^{30}$.

שיטת קוד	מספר סיביות לחלק ראשון	מס' סיביות חלק שני	סה"כ סיביות
$C_\gamma(10^9) =$	$\lvert Unary(30)\rvert = 30\ bits$	$29\ bits$	$59\ bits$
$C_\delta(10^9) =$	$\lvert C_\gamma(30)\rvert = 9\ bits$	$29\ bits$	$38\ bits$
$C_\varepsilon(10^9) =$	$\lvert C_\delta(30)\rvert = 9\ bits$	$29\ bits$	$38\ bits$

קוד אוניברסלי

נניח אוסף א"ב בסדרה יורדת של הסתברויות $p_1 \geq p_2 \geq ... \geq p_n$

ובגלל שהסדרה יורדת אז לכל $1 \leq x \leq n$ נגיד שההסתברות: $p_x \leq \frac{1}{x}$

נרצה להוכיח את ההסתברות הזאת, כאשר x זה האינדקס של המספר ואנחנו רוצים לדעת את ההסתברות.

<u>הוכחה</u>

נניח בשלילה שקיים x שעבורו $p_x > \frac{1}{x}$ לכן נקח את הסכום של כל ההסתברויות מההסתברות הראשונה ועד x:

$$\sum_{i=1}^{x} p_i > \sum_{i=1}^{x} \frac{1}{x} = x \cdot \frac{1}{x} = 1$$

קיבלנו שסכום ההסתברויות גדול מ-1 ולכן זאת הסתירה ולכן:

$$p_x \leq \frac{1}{x}$$

לכל $1 \leq x \leq n$.

∎

למה זה מעניין אותנו? - אם ניקח את כמות המידע שיושב במקום ה-x ולפי ההגדרה:

$$I(S_x) = -\log(p_i) \leq -\log(\frac{1}{x}) = \log(x)$$

כלומר כמות המידע היא בסדר גודל של $\log(x)$.

אנחנו הראינו בקודי אליאס שאנחנו רחוקים מכמות המידע ב-:

$$C_\gamma: \log_2(x) + \theta(\log(x))$$

$$C_\delta: \log_2(x) + o(\log(x))$$

קוד גולומב Golomb

קידוד גולומב הוא פשוט שיטה/אלגוריתם לדחיסת נתונים, זה קוד עם בלוקים באורך קבוע - Fixed size.
במקום לדבר על בלוקים בגודל שעולה אקספוננציאלי כמו שהיה לנו ב- C_γ ו-C_δ אנחנו נדבר על מקרה בו כל הבלוקים הם *<u>באותו הגודל</u>*. למעשה יהיה לנו פרמטר לקוד הזה שנקרא לו b והוא יהיה הגודל של כל בלוק.
גודל הא"ב הוא b וגם $0 < r \leq b$.

Golomb Encode Algorithm (הצפנה)

```
1: Golomb_encode(x,b) {
2:       q ⇐ (x-1) div b; // החלק השלם
3:       r ⇐ x-q*b; // השארית
4:       Unary_encode(q+1); // מקודדים באונרי את השארית
5:       Minimal_binary_encode(r,b); // מקודדים את השאר
6: }
```

Golomb Decode Algorithm (פענוח)

```
1: Golomb_decode(b) {
2:       q ⇐ Unary_decode()-1;
3:       r ⇐ Minimal_binary_decode(b);
4:       return r+q*b;
5: }
```

בדוגמה הבאה, נרצה לקודד את $x = 9$, יש לנו $b = 5$ מילות קוד. כלומר נרצה לקודד את מילת הקוד התשיעית כאשר x הוא האינדקס של מילת הקוד שרוצים לקודד.

Golomb Encode Algorithm - $x = 9$ $b = 5$ **דוגמה**

```
1: Golomb_encode(x=9,b=5) {
2:       q ⇐ (9-1) div 5; // q = 1
3:       r ⇐ 9-1*5; // r = 4
4:       Unary_encode(1+1); // unary(2) = 10
5:       Minimal_binary_encode(r,b); // MBE(4,5) = 110
6: } // so we get 9_golomb = 10110
```

קוד רייס Rice

קוד רייס הוא מקרה פרטי של קוד גולומב. נסתכל על בלוקים b בגודל של חזקות של 2 כלומר עבור איזה k מסוים מתקיים $b = 2^k$. מה ההיתרון? - הפעולות רובן הם פעולות של shifting שזה פעולה מאוד חסכונית לעומת כפל וחילוק וכו'.. כלומר *השימוש של רייס חוסך באנרגיה*.

דוגמה - נועד כדי להשתכנע שרייס הוא מקרה פרטי של גולומב.

עבור גולומב:

נחשב בדומה לדוגמה הקודמת רק שהפעם $b = 4$ שזה חזקה של 2. כלומר $Golomb(9, 4)$.

$$q = 2$$

$$r = 9 - 2 \cdot 4 = 1$$

$$Unary(3) = 110$$

במקרה שהא"ב שלנו הוא בחזקה של 2 יהיה לנו מאוד קל לעבוד עם MBE לכן b=4 אז אנחנו מדברים על מילות קוד באורך *קבוע* ומילת הקוד הראשונה זה "00"

$$MBE(1, 4) = 00$$

ומכאן:

$$Golomb(9, 4) = 110\ 00$$

עבור רייס:

נתון $x = 9$ וגם $b = 4$ אז $k = 2$

$$(x - 1)_2 = 1000$$

לאחר right-shift נקבל ~~00~~10.

$$Unary(2 + 1) = 110$$

את אותם ביטים שמחקנו, נסתכל עליהם ונקבל:

$$Rice(9, 4) = 110\ 00$$

כלומר קיבלנו אותו הדבר, לכן הרייס הוא מקרה פרטי של גולומב ונעשה את זה בפעולות של shift שהיא פעולה שחוסכת אנרגיה ולכן נשתמש בה במקרים שיש לנו b מחזקה של 2.

תכונות של קוד גולומב

- קוד גולומב עם פרמטר b עבור x וגם $x + b$ שונה באורכו ב-1.
- $\left|C_{x+b}\right| = \left|C_x\right| + 1$
- $p_x = (1 - p)^{x-1} \cdot p$
- $p_{x+b} = (1 - p)^{x+b-1} \cdot p$
- $\frac{p_x}{p_{x+b}} = (1 - p)^b$
- פרמטר b נבחר כדי לספק את $\frac{p_x}{p_{x+b}} = 0.5$

מספרי פיבונצ'י

תזכורת:

בסיס: $F_0 = 0, \quad F_1 = 1$

צעד: $F_i = F_{i-2} + F_{i-1}$

הגדרה: ניתן לייצג כל מספר בעזרת מספרי פיבונאצ'י: $n = \sum_{i=1}^{k} n_i \cdot F_{i+1}$.

שיטת Fib1

Fibonacci Binary Encode Algorithm (הצפנה)

```
1:  Fibonacci_Binary_Encode(n) {
2:        find the largest Fibonacci number ≤ n.
3:        Repeat recursively with n − F_k until n − F_k = 0.
4:  }
```

לדוגמה:

עבור $n = 73$:

$binary$		128	64	32	16	8	4	2	1
73		0	1	0	0	1	0	0	1
$fibo$	55	34	21	13	8	5	3	2	1
73	1	0	0	1	0	1	0	0	0

בעצם מה שעשינו בייצוג בינארי שאלנו מה החזקה של 2 הכי גבוהה שנכנסת לאותו מספר והצבנו 1.

בפיבונאצ'י ניקח את המספר הכי גבוהה שנכנס ל-n ונחזור רקורסיבית על התהליך עם n פחות אותו מספר פיבונאצ'י עד שנגיע ל-0.

i	Fibonacci Binary Representation	Fibonacci Code
1	1	11
2	10	011
3	100	0011
4	101	1011
5	1000	00011
6	1001	10011
7	1010	01011
8	10000	000011
9	10001	100011
10	10010	010011

Data Compression Course - Dana Shapira

סידור קוד פיבונאצ'י לצורה של קוד

1. תמיד נוסיף 1 למוביל בהתחלה.
2. נהפוך כל מילה.

- תמיד במספר פיבונאצ'י מתחיל ב-1 ולכן אחרי ההוספה כל מילת קוד תתחיל ב-11.
- בעיה - הקוד לא מיידי (צריך לקרוא 2 אחדות - לחזור ולשבור).
- תיקון - נהפוך כל מילה, וה-11 ישב בסוף.

<u>תכונות:</u>

- ה-F_k הן מילות קוד באורך $k + 1$ (כולל ה-1 שמוסיפים בסוף) כך ש: $F_k = F_{k+2} - F_{k+1}$.
- קוד פרפיקסי
- קוד מיידי
- קוד אחיד - יש השמטה/החלפה של ביט.

שיטת Fib2

החיסרון ב-Fib1 הוא שאין מילת קוד באורך 1 (כי הראשונה היא 11) - ואם יש לנו תו עם הסתברות מעל חצי זה יחסר לנו - לכן הוסיפו את 1 ושאר המילים לפי החוקיות הבאה:

- דרך ראשונה:
  - מורידים את ה-1 הכי ימני "שהוספנו" בסוף ב-Fib1.
  - מורידים את כל מילות הקוד שמתחילות ב-0.
- דרך שניה:
  - מורידים את ה-1 שהוספנו מימין בסוף כל מילת קוד.

Fib_1	Fib_2
11	1
011	01
0011	001
1011	101
00011	0001

  - נוסיף 10 לכל מילת קוד בהתחלה

Fib_1	Fib_2
11	101
011	1001
0011	10001
1011	10101
00011	100001

  - נוסיף את 1 כמילת קוד.

Fib_1	Fib_2
11	1
011	101
0011	1001
1011	10001
00011	10101

<u>תכונות:</u>

1. מכיל את מילת הקוד '1'
2. יש F_{n-2} מילות קוד באורך k.
3. ניתן להוכיח שתמיד יותר טוב מ-C_γ ופחות טוב מ-C_δ.

קוד שאנון-פאנו - Shannon

מבוא

מכיוון שכמות האינפורמציה היא לא בהכרח מספר שלם והיא חוסמת את אורך מילת הקוד מלמטה אז שאנון הציע פתרון כך שכל מילת קוד שנבנה יהיה הערך העליון של כמות האינפורמציה.

עשינו זאת כך:

$$I(S_i) = -\log_2(p_i) = \log_2(\frac{1}{p_i})$$

מאחר ו-:

$$\log_2(\frac{1}{p_i}) \leq \lceil \log_2(\frac{1}{p_i}) \rceil < \log_2(\frac{1}{p_i}) + 1$$

אז לפי האנטרופיה נקבל:

$$H(P) = -\sum_{i=1}^{n} p_i \log_2 p_i = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} \leq$$

$$\leq \sum_{i=1}^{n} p_i \lceil \log_2(\frac{1}{p_i}) \rceil < \sum_{i=1}^{n} p_i (\log_2 \frac{1}{p_i} + 1)$$

והראנו שלמעשה אם ניקח את האורך הממוצע של מילות הקוד:

$$L = \sum_{i=1}^{n} p_i \cdot \lceil \log_2(\frac{1}{p_i}) \rceil$$

אז נקבל:

$$H(P) \leq L < H(P) + 1$$

כלומר אורך מילת הקוד רחוק מהאנטרופיה רק בסיבית 1.

והסברנו שזה הקוד הממוצע לכל מילת קוד ואם אנחנו מאבדים סיבית אחת על כל אורך קלט n אז זה אומר שבסה"כ נאבד n סיביות.

שאנון - אלפבית של זוגות

שאנון הציע להגדיל את האלפבית שלנו - m. נסתכל על כל הזוגות של התווים a, b, c כלומר $ab,, bc, ca$ וכו'...
לכן במקום n איברים יהיה לנו n^2 איברים, מה הרווח שלנו מזה? -
נניח שההסתברויות <u>*בלתי תלויות*</u>, נסתכל על האנטרופיה של כל הזוגות שלנו.
נחשיב את הסמלים של האלפבית ככל הזוגות שלנו (s_1, s_2) עם ההסתברויות $p(s_1, s_2) = p(s_1) \cdot p(s_2)$.

$$H\left(p(s_i, s_j)\right) = -\sum_{(i,j)} p(s_i,\ s_j) \cdot \log_2 p(s_i\ ,\ s_j) =$$

$$= -\sum_i \sum_j p(s_i) \cdot p(s_j) \cdot \log_2\left(p(s_i) \cdot p(s_j)\right) =$$

$$= -\sum_i \sum_j p(s_i) \cdot p(s_j) \cdot \left(\log_2\left(p(s_i)\right) + \log_2\left(p(s_j)\right)\right) =$$

$$= -\sum_i p(s_i) \cdot \sum_j p(s_j) \cdot \log_2(p(s_j)) - \sum_j p(s_j) \cdot \sum_i p(s_i) \cdot \log_2(p(s_i)) =$$

$$= H(P(s)) \cdot \sum_i p(s_i) + H(P(s)) \cdot \sum_j p(s_j) = 2H(P(s))$$

קיבלנו שהאנטרופיה של הזוגות של האלפבית בגודל n^2 זה פשוט פעמיים האנטרופיה על תווים בודדים.
אם נחזור לקוד שאנון אמרנו שהקוד שלו רחוק מהאנטרופיה בסיבית אחת, לכן אם נעשה זאת על האלפבית של זוגות במקום על אלפבית מקורי אז נהיה רחוקים עדיין בסיבית אחת אבל אם נחשב את האורך הממוצע פר-תו בודד ולא פר-זוג אז אנחנו מקבלים שאנחנו רחוקים מהאנטרופיה רק בחצי סיבית כלומר התקרבנו יותר לאנטרופיה.

לכן, אורך קוד שאנון של האלפבית החדש חסום בצורה הבאה:

$$L\left((s_1, s_2)\right) < H\left(P(s_1, s_2)\right) + 1 = 2H\left(\ P(s)\ \right) + 1$$

מספר הסיביות בכל סימן עבור זוגות יהיה חסום בצורה הבאה:

$$\frac{1}{2} \cdot L\left((s_1, s_2)\right) < H(P(s)) + \frac{1}{2}$$

ומספר הסיביות בכל סימן עבור שלשות, רביעיות... ו-n-יות יהיה חסום בצורה הבאה:

$$\frac{1}{n} \cdot L\left((s_1, s_2, \ldots, s_n)\right) < H(P(s)) + \frac{1}{n}$$

קידוד עץ בינארי מלא

לכל צומת פנימית נציב '1'.

לכל עלה נציב '0'.

טענה

לעץ בינארי מלא עם n עלים יש $n - 1$ צמתים פנימיים.

טענה

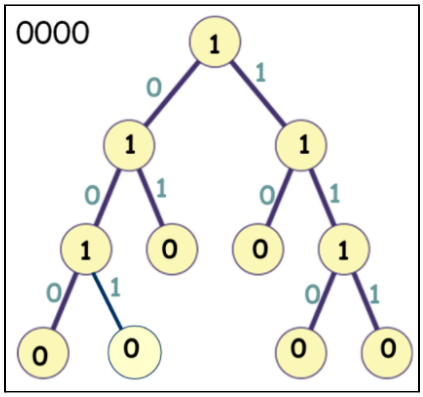
מספר הביטים הנחוצים לקידוד עץ בינארי מלא עם n עלים הוא $2n - 1$.

דוגמה

- המחרוזת הבינארית שמתארת את העץ היא:

  1 11 1001 0000

- יש $n = 6$ עלים וגם $2n - 1 = (11)bits$ בסדרה.



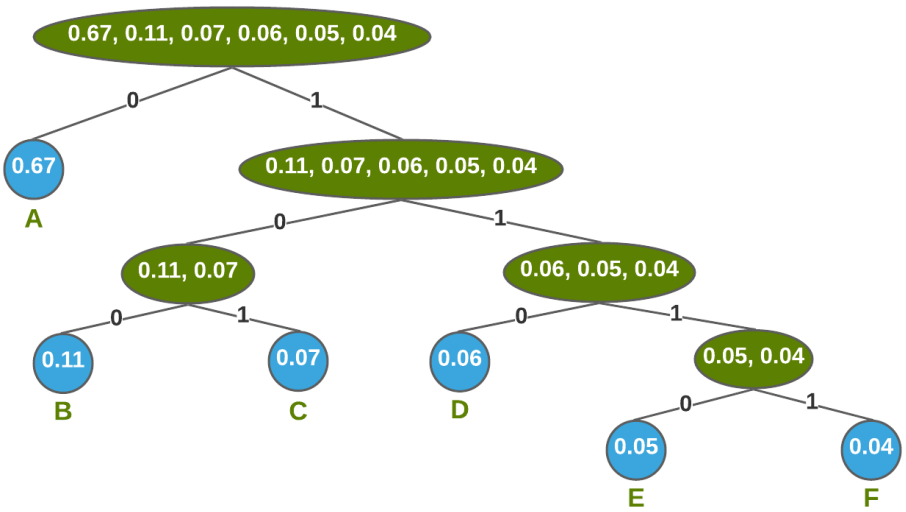
אלגוריתם שאנון-פאנו

- בקוד שאנון-פאנו, הסמלים מסודרים לפי ההסתברויות שהם נמצאים, מהגדול לקטן ואז הם מחולקים לשתי קבוצות כך שסכום ההסתברויות בכל קבוצה שווה כמה שאפשר לקבוצה השנייה.
  כל סמל מקבל את הספרה הראשונה של קוד התחיליות שלו, הסמלים בקבוצה הראשונה מקבלים "0" והסמלים בקבוצה השנייה מקבלים "1".
  עבור כל קבוצה שנשאר בה יותר מסמל אחד, התהליך חוזר על עצמו וכל סמל יקבל ספרה נוספת.
  כאשר בקבוצה נשאר רק סמל אחד - הקוד עבור אותו סמל הושלם ואותו קוד לא יופיע בתחילת קוד של סמל אחר.
- קוד שאנון-פאנו תמיד יוצר עץ מלא אך הוא לא השיטה הכי אופטימלית.

Shannon-Fano Algorithm

1. עבור רשימה נתונה של סמלים, יש להתאים עבור כל סמל את ההסתברות שלו בקטע המקור.
2. נמיין את רשימת הסמלים לפי ההסתברות שלהם כאשר הגבוהה ביותר נמצאת בשמאל והנמוכה ביותר בימין.
3. נפצל את הרשימה לשתי קבוצות כך שסכום ההסתברויות בכל קבוצה שווה כמה שאפשר לקבוצה השנייה.
4. החלק השמאלי של הרשימה מקבל את הספרה הבינארית "0" והימני מקבל "1".
   הכוונה היא שכל קוד עבור סמל שנמצא בחלק השמאלי יתחיל ב-"0" ובימני יתחיל ב-"1".
5. חזור באופן רקורסיבי על חלקים 3 ו-4 עבור כל אחד משני החלקים - חלק בשנית כל חלק והוסף ביטים לקוד, עד שכל סמל הופך לעלה בעץ ויש לו קוד מתאים.

דוגמה: נתון סדרת הסתברויות מונוטונית יורדת: $P = \{0.67,\ 0.11,\ 0.07,\ 0.06,\ 0.05,\ 0.04\}$
וגם $\Sigma = \{A, B, C, D, E, F\}$.



הערה: נשים לב שבכל שלב פיצלנו ל-2 ולכן נקבל עץ מלא, כלומר קוד שלם.

<u>דוגמה של האלגוריתם</u>

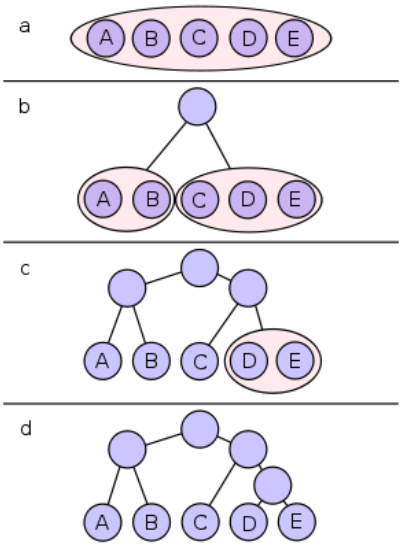
הדוגמה מראה בנייה של קוד שאנון-פאנו עבור אלפבית קטן.

ננית שיש חמישה סמלים באלפבית עם התדירויות הבאות:

סמל	E	D	C	B	A
כמות	5	6	6	7	15
הסתברות	0.12820513	0.15384615	0.15384615	0.17948718	0.38461538

a) כל הסמלים ממוינים לפי תדירות משמאל לימין כמתואר בטבלה.

b) לפי שלב 3 של האלגוריתם, נשים קו מפריד בין הסמלים B ו-C כך שסכום הקבוצה השמאלית יהיה 22 וסכום הימנית יהיה 17 - החלוקה שנותנת את ההפרש הקטן ביותר שניתן בין החלקים.



c) על פי החלוקה הזאת, עבור כל אחד מ-A ו-B, ישויך קוד שמתחיל בביט שערכו "0" ועבור C, D ו-E קוד שמתחיל ב-"1" כפי שמתואר בטבלה מלמטה. לאחר מכן (שלב 4) החלק השמאלי של העץ מקבל חלוקה חדשה בין A ו-B, מה שמשאיר את A בתור עלה עם קוד 00 ואת B בתור עלה עם קוד 01.

d) אחרי ארבעה תהליכי חלוקה, נוצר העץ של הקוד. בעץ הסופי, שלושת הסמלים עם התדירויות הגבוהות, קיבלו קוד של שני ביטים כל אחד ושני הסמלים עם הכמות הקטנה יותר, שלושה ביטים כל אחד, כמו שמתואר בטבלה למטה:

סמל	E	D	C	B	A
קוד	111	110	10	01	00

- נשים לב שכל פעם פיצלנו ל-2 ולכן נקבל עץ מלא, כלומר קוד שלם.
- שיטת הקוד הזו לא תמיד אופטימלית.
- אנחנו בנינו את הקוד מלמעלה למטה.

קוד הופמן - Huffman

מבוא

- אלגוריתם שאנון-פאנו, לא תמיד יוצר קוד תחיליות אופטימלי.
נציג אלגוריתם אחר, שיוצר עץ אופטימלי עבור כל קבוצת סמלים עם הסתברות נתונה.
- העץ של האפמן נבנה מהעלים לכיוון השורש - מלמטה למעלה, בניגוד לעץ שאנון-פאנו שנבנה מהשורש לעלים - מלמעלה למטה.

- קוד האפמן הוא שיטה לקידוד סימנים, כגון תווי טקסט, ללא אובדן נתונים.

קוד האפמן הוא קוד פרפיקסי, כלומר מחרוזת ביטים שמייצגת אות לעולם אינה מהווה תחילית של מחרוזת המייצגת אות אחרת. קוד כזה מבטיח אפשרות יחידה לפיענוח, ויתרה מזאת, הפיענוח מהיר, שכן מספיק לעבור על הרצף המקודד פעם אחת מההתחלה ועד הסוף תוך שמירת מעט מידע.

קוד מסוג זה ניתן לייצג על ידי עץ בינארי, כאשר עלי העץ מייצגים את האותיות המקודדות, וצמתי העץ מסומנים ב-0 או 1. כאשר רוצים לפענח רצף ביטים כלשהו, הולכים על העץ על פי הביטים שנקלטים עד אשר מגיעים לעלה. האות המאוחסנת בעלה היא האות המפוענחת.

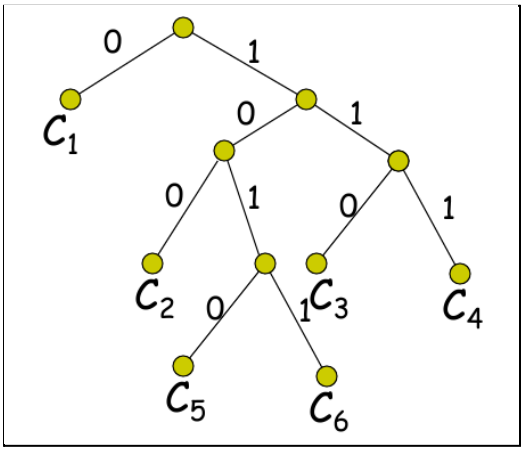
כאשר הקוד הוא אופטימלי, העץ יהיה עץ מלא. כלומר, לכל צומת שאינו עלה יהיו בדיוק שני בנים.

שלבי האלגוריתם

1. צור תור עדיפות Q שיכיל צמתים המייצגים את כל הסמלים, ממוינים על פי הסתברות הופעתם.
2. כל עוד בתור Q יש יותר מצומת אחת, בצע:
   a. צור צומת חדש z.
   b. הוצא מתור העדיפויות את שני האיברים העליונים, x, y - בעלי ההסתברות הנמוכה ביותר.
   c. הפוך את x, y לבנים הימני והשמאלי של z.
   d. קבע את ההסתברות של z להיות סכום ההסתברויות של x ו-y.
   e. הכנס את z לתור העדיפויות.
3. הצומת הבודד שנותר בתור העדיפויות הוא שורש עץ הקוד המבוקש.
4. הקוד עבור כל סמל יקבע לפי המסלול מהשורש לעלה שמייצג את הסמל. אם הצלע ה-nית מובילה לבן שמאלי, ערך הביט ה-nי יהיה "0". אם היא מובילה לבן ימני, ערכו יהיה "1".

עצי קוד

- כל קוד פרפיקסי יכול להיחשב כקוד-עץ.
- סמלי המקור הם העלים בעץ.
- מסלול ייחודי מהשורש אל העלה מייצג את הקוד עצמו עבור הסמל.
- פענוח והצפנה - נטייל על עץ-הקוד מצלע לצלע.
- חסרון - דרישה לנפח מקום ואיטי.



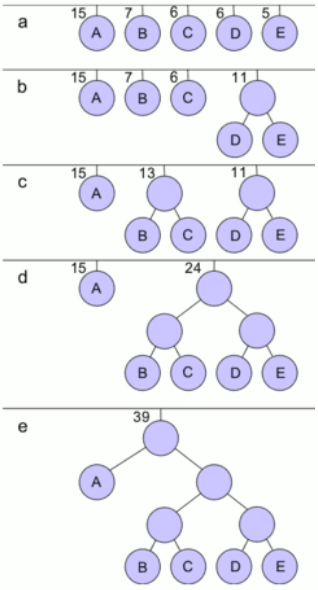
Huffman Algorithm

```
1:  Huffman(Σ){
2:        n ⇐ |Σ|
3:        Q ⇐ Σ
4:        for i ⇐ 1 to n-1 do:
5:              do ALLOCATE-NODE(z)
6:              left[z] ⇐ x ⇐ EXTRACT-MIN(Q)
7:              right[z] ⇐ y ⇐ EXTRACT-MIN(Q)
8:              w[z] ⇐ w[x] + w[y]
9:              INSERT(Q, z)
10:       end-for
11: return EXTRACT-MIN(Q)
12: }
```

דוגמה:

נשתמש באותם תדירויות כמו שהשתמשנו בדוגמה של שאנון-פאנו, כלומר:

סמל	E	D	C	B	A
כמות	5	6	6	7	15
הסתברות	0.12820513	0.15384615	0.15384615	0.17948718	0.38461538



a) נציג את הסדרה.

b) על פי האלגוריתם של האפמן, ל-D ו-E יש את ההסתברות הנמוכה ביותר ולכן יהיו בנים לצומת בעלת ההסתברות של D ו-E ביחד, כלומר 0.28205128.

c) הזוג הבא בתור הוא B ו C ויהיו בנים לצומת בעלת ההסתברות של B ו C ביחד, כלומר 0.33333333.

d) זה משאיר את BC ו-DE עם ההסתברות הנמוכה ביותר בתור ולכן גם הם יהיו בנים לצומת BCDE.

e) נשארנו עם A, שיהיה בן לצומת אחרונה.

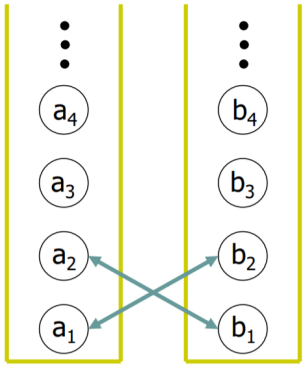
סמל	E	D	C	B	A
קוד	111	110	101	100	0

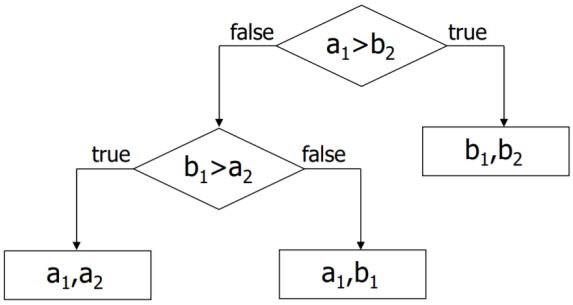
זמן הריצה של האלגוריתם

- יצירת ערימה $O(n)$
- רצים על הלולאה $n - 1$ פעמים.
  - הקצאה של צומת $O(\log n)$
  - הוצאה של המינימלי ביותר מהערימה $O(\log n)$
  - השמה לבן של z היא $O(\log n)$
  - הכנסה של z לערימה $O(\log n)$
- סה"כ $O(n \log n)$

קוד האפמן עם שני תורים

- עבור הסתברויות נתונות <u>*בסדרה שהיא כבר ממוינת*</u>, אפשר ליעל יותר טוב מהאלגוריתם הרגיל.
- נשים את ההסתברויות המקוריות בתור הראשון, ואת הסכומים בתור השני.
- בכל שלב נבחר את 2 המינימליים ביותר.
- אנחנו נקבל זמן ריצה של $O(n)$ במקום $O(n\log n)$ - כלומר יעיל יותר!.
- נשים לב שאפשר להשתמש באלגוריתם זה רק אם קיבלנו את סדרת המשקלים ממוינת מראש!





משפט

אלגוריתם הופמן נותן קוד בעל יתירות מינימלית.

בהינתן סדרת משקלים $w_1,\ldots,w_n$.

אלגוריתם הופמן מקצה קוד באורכים $l_1,\ldots,l_n$ כך ש $\sum_{i=1}^{n} w_i l_i$ הוא מינימלי.

נוכיח משפט זה בעזרת הוכחת 3 הלמות הבאות:

למה 1

עץ אופטימלי (עץ המתאים לקוד בעל יתירות מינימלית) הוא עץ מלא.

<u>*הוכחה:*</u>

נניח בשלילה שהעץ האופטימלי T אינו מלא, קיים צומת x בעל ילד יחיד y.

נבנה עץ חדש $\bar{T}$ כך ש-x ו-y מתלכדים בסתירה לאופטימליות של T.

∎

<u>*הסבר*</u>:

לקחנו את כל מילות הקוד שלא מושרשות ב-y ונשארות אותו הדבר וכל אלה שכן מורשות ב-y קיצרנו אותם בסיבית אחת ולכן שיפרנו את הסכום $\sum_{i=1}^{n} w_i l_i$ בסתירה לאופטימליות של T. (כלומר מצאנו אופטימלי יותר).

למה 2

בעץ אופטימלי המשקלים הנמוכים ביותר w_n ו-w_{n-1} נמצאים ברמה התחתונה ביותר, שנתנו להם את מילות הקוד הארוכות ביותר.

הוכחה:

נניח בשלילה ש-w_{n-1} בלי הגבלת הכלליות אינו נמצא ברמה הנמוכה ביותר, כלומר במקומו יש את w_x ו-w_{n-1} נמצא ברמה גבוהה יותר מעליו.

כלומר $w_x > w_{n-1}$ וגם $l_x > l_{n-1}$.

ולכן:

$$\left(w_x - w_{n-1}\right)\left(l_x - l_{n-1}\right) > 0$$

נכפיל ונקבל:

$$w_x l_x - w_x l_{n-1} - w_{n-1} l_x + w_{n-1} l_{n-1} > 0$$
$$w_x l_x + w_{n-1} l_{n-1} > w_x l_{n-1} + w_{n-1} l_x$$

אך לא קיים עץ $\bar{T}$ כזה.

∎

הסבר:

הנחנו ששני המשקלים הנמוכים ביותר לא נמצאים ברמה הנמוכה בעץ, כלומר קיים איזה x שמשקלו גבוה יותר מאחד מהמשקלים הנמוכים יותר, ונתנו ל-x אורך גדול יותר.

l_x היא מילת קוד ארוכה יותר מ-l_{n-1} בה"כ. בנינו עץ $\bar{T}$ ששם הסכום $w_i l_i$ יהיה טוב יותר מהעץ T שהיה האופטימלי ביותר וזו הסתירה שהגענו כי לא ייתכן שקיים $\bar{T}$ שכזה.

למה 3

בעץ אופטימלי המתאים לקוד אופטימלי אנחנו יכולים להניח שהמשקלים הנמוכים ביותר w_n, w_{n-1} הם אחים.

- בעץ הופמן תמיד איחדנו את הנמוכים ביותר להיות אחים.

הוכחה:

נסתכל על עץ אופטימלי T.

לפי למה 1 הוא עץ מלא ולפי למה 2 אנו יודעים כי w_n, w_{n-1}הם ברמה התחתונה אבל הם לא אחים.

אם העץ הוא מלא אז לשניהם חייב להיות אחים ל-w_n יש את האח w_x ול-w_{n-1} את האח w_y.

אנו יודעים שכולם נמצאים ברמה התחתונה כלומר האורך של מילת הקוד: $l_n = l_x = l_y = l_{n-1}$.

למעשה, אם נחליף את w_{n-1} ב-w_x נקבל את העץ $\bar{T}$ כך שלא שינינו את הסכומים $w_i l_i$ והאורכים.

∎

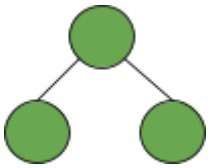
משפט

עץ הופמן הוא עץ אופטימלי.

(זה אותו משפט קודם שהוצג לפני הלמות, היינו צריכים להוכיח את הלמות כדי להוכיח את המשפט הזה).

הוכחה

נוכיח באינדוקציה על n עלים.



- <u>*בסיס האינדוקציה*</u>: עבור $n = 2$ עלים.
  עץ זה הוא עץ אופטימלי יחיד עם 2 העלים וזה עץ זהה לעץ הופמן המתקבל על ידי 2 עלים.
- <u>*הנחת האינדוקציה*</u>: נניח עבור $n - 1$.
- <u>*צעד האינדוקציה*</u>: נוכיח עבור n.

יהי T_1 *עץ אופטימלי* עבור המשקולות הבאות: $\{w_1,\dots,w_n\}$ עם מילות קוד עם אורכים $l_1,\dots,l_n$

כך ש: $\Sigma_{i=1}^{n} w_i l_i$ הוא מינימלי.

אנו יודעים כי $l_n = l_{n-1} = l_{x+1}$ אז w_n ו-w_{n-1} נמצאים ברמה התחתונה ואחים והאבא שלהם זה w_x.

נבנה עץ T_2 מ-T_1 עם $(n-1)$ ע"י השמטת w_{n-1}, w_n כך ש-w_x כעת הוא עלה.

נוכיח את הטענה הבאה לסיום ההוכחה:

טענה

העץ T_2 הוא עץ אופטימלי עבור $n - 1$ משקלים $\{w_1,\dots,w_x\}$ כאשר $w_x = w_{n-1} + w_n$.

הוכחה

נסמן $M_1 = \Sigma_{i=1}^{n} w_i l_i$.

נשים לב כי:

$$M_2 = M_1 - (w_{n-1} + w_n) \cdot 1$$

ננית בשלילה כי T_2 לא אופטימלי ולכן קיים T_3 עבור אותם משקולות $\{w_1,\dots,w_x\}$ כך ש: $M_3 < M_2$.

נבנה עץ T_4 אשר הינו עץ זהה, עם משקולות $w_{n-1},\ w_n$ ונקבל M_4.

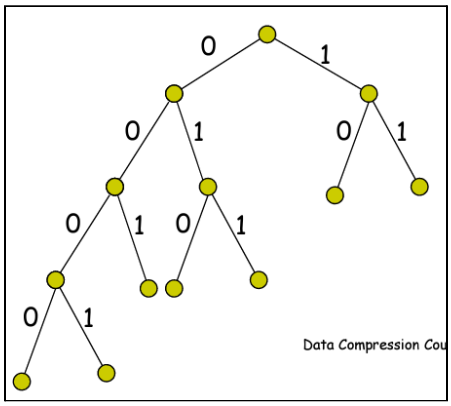
ידוע כי $M_3 < M_2$ לפי ההנחה ולכן:

$$M_4 = M_3 + \left(w_{n-1} + w_n\right) < M_2 + \left(w_{n-1} + w_n\right) = M_1$$

בסתירה לאופטימליות של M_1 ולכן T_2 הוא עץ אופטימלי, נוסיף ל-T_1 שני עלים על ידי הופמן ונקבל כי T_1 הוא עץ הופמן.

- יש כמה עצים אופטימליים ולא רק עץ אופטימלי אחד ויחיד.

קוד הפמן קנוני



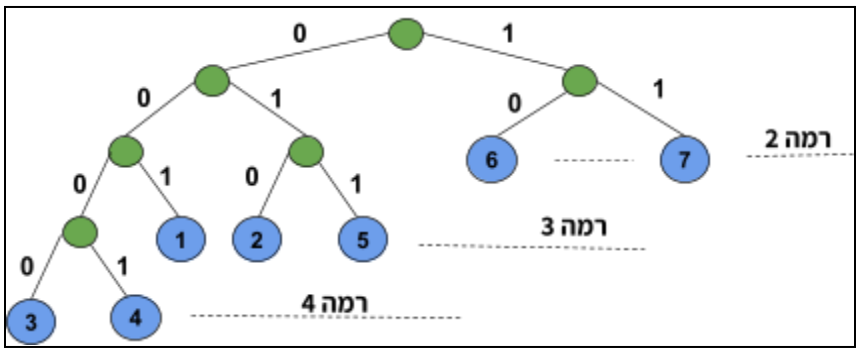
בקוד הפמן קנוני לוקחים את האורכים של מילות הקוד שמקבלים מהפמן ומסדרים בעץ בצורה כזאת שמילות הקוד הארוכות ביותר יהיו לאורך המסלול השמאלי ביותר בעץ.
למעשה, אנחנו מסדרים את העץ לפי אורך המילים, הארוכות ביותר במסלולים השמאליים והקצרים מימין.
בהפמן קנוני אנחנו יוצרים בלוקים לפי האורכים של מילות הקוד, כלומר כל בלוק מייצג אורך ומכיל מספרים בינאריים עוקבים.

היתרון

קוד הפמן קנוני עובר בצורה מכווצת יותר למפענח, נעביר למפענח את:

- הא"ב לפי הסדר שלהם בעלים של העץ משמאל לימין.
- מילת הקוד הראשונה בכל בלוק.
- כדי להפריד בין כל בלוק נתן גם את התו של כל מילת קוד ראשונה בבלוק.

S_i	*Canonical*	*Length*
3	0000	4
4	0001	
1	001	3
2	010	
5	011	
6	10	2
7	11	



אלגוריתם לבניית קוד הפמן קנוני

האלגוריתם הבא מתאים גם ל-decoder וגם ל-encoder .

הרעיון הוא לבנות קוד האפמן שהוא יחיד, אבל הוא יחיד כתלות באורכים.

בהינתן הסימנים $\{i\}$ באורכים אופטימליים $\{l_i\}$.

אנחנו מניחים שבאמת האורכים האלה התקבלו באמצעות אלגוריתם האפמן.

אנחנו צריכים לקבל את האורכים כדי להגיע לאותו קוד הופמן קנוני.

Huffman Assigning Codewords Algorithm

```
1:  HuffmanAssigningCodewords(Σ, {ℓ_i}) {
2:  maxlength ⇐ max({ℓ_i})
3:  // find the number of codewords of each length
4:  for ℓ ⇐ 1 to maxlength do:
5:        num[ℓ] ⇐ 0
6:  for i ⇐ 1 to n do:
7:        num[ℓ_i]++
8:  // store the first codeword
9:  firstcode[maxlength] ⇐ 0
10: for ℓ ⇐ maxlength-1 down to 1 do:
11:       firstcode[ℓ] ⇐ (firstcode[ℓ+1] + num[ℓ+1])/2
12: for ℓ ⇐ 1 to maxlength do:
13:       nextcode[ℓ] ⇐ firstcode[ℓ]
14: for i ⇐ 1 to n do:
15:       codeword[i] ⇐ nextcode[ℓ_i]
16:       symbol[ℓ_i , nextcode[ℓ_i]-firstcode[ℓ_i]] ⇐ i
17:       nextcode[ℓ_i]++
18: }
```

Huffman Assigning Codewords Algorithm - הסבר קוד	
1:	בהינתן א"ב וגם האורכים (אחרי הרצת הפמן), נרצה לבנות את עץ הפמן קנוני
2:	נבחר את אורך מילת הקוד הארוכה ביותר להיות maxlength.
3:	נרצה למצוא את מספר מילות הקוד בכל אורך.
4:	נעבור על כל האורכים מ-1 עד המקסימלי ביותר.
5:	המערך num מייצג את מספר מילות הקוד בכל בלוק (כל אינדקס זה אורך) -נציב 0.
6:	נעבור על כל אות ב-א"ב שלנו כאשר $n = \|\Sigma\|$.
7:	נתון לנו האורך ℓ_i של כל אות בא"ב ולכן לכל אורך ℓ נספור כמה פעמים יש כזה.
8:	נשמור את מילת הקוד הראשונה.
9:	המערך firstcode מייצג את מילת הקוד הראשונה בכל בלוק.בנוסף הוא מייצג את מספר הצמתים הפנימיים בעץ. בקוד אנחנו נאתחל ב-0 את התא של האורך המקסימלי כי תמיד מילת הקוד הראשונה זה 0 - אמרנו שמילת הקוד הארוכה ביותר תהיה בצד השמאלי ביותר בעץ ולכן הוא יקבל 0000... שזה המסלול השמאלי ביותר בעץ קנוני.
10:	נלך אחורה עד ל-1.
11:	כיוון שהמספרים הם בינאריים עוקבים אז אם ניקח את firstcode ונוסיף לו את ה-num אז אנחנו נדע כמה יש לנו באותו הבלוק ונחלק ל-2 כדי לעשות shift ימינה, כלומר לוותר על סיבית - אנחנו נלך ממילת הקוד הארוכה ביותר, נחשב בצורה הזאת מה מילת הקוד הראשונה בבלוק הבא. בצורה הזאת נדע לסדר נכון לפי האורכים בסדר יורד. כלומר, אם בשורה 9 אמרנו שהוספנו תחילה 0000.. אז במילות הקוד עם האורכים הארוכים ביותר (פחות אחד) יקבלו בהתאם 0001.. כלומר עדיין מצד שמאל בהתאם.
12:	נרצה לדעת את ה-nextcode שאומר מהי מילת הקוד לכל תו בבלוק ולא רק לראשון. נעבור על כל אחד מהבלוקים.
13:	נאתחל את nextcode להיות כל אחד מה-firstcode תחילה כדי לזכור לאיזה מצב הגענו עד עכשיו.
14:	נעבור על כל אות בא"ב.
15:	ניתן את מילת הקוד הראשונה, לדוגמה: האיבר הראשון למשל הוא האות a עם אורך של 3, ניקח ל-a את ה-nextcode ב-ℓ_i שלו כלומר $\ell_1 = 3$ ובתא של $nextcode[3] = 1$ לכן $codeword[1] = nextcode[3] = 1$. הכוונה היא שנכתוב את המספר 1 עם שלוש סיביות הכוונה היא 001. כלומר מילת הקוד של a היא 001.
16:	ה-decoder יכול לקבל את האורכים ולבנות את העץ כך שהוא יודע כמה מילות קוד יש בכל אורך. ה-symbol הוא מערך דו-מימדי. החיסור שם הוא למעשה ההיסט של אותו symbol באותו בלוק, כלומר כמה הוא רחוק ממילת הקוד הראשונה. הכוונה היא שאם יש לנו כמה מילות קוד באותו אורך נוכל לדעת לפי ההפרש מהי המילת קוד הראשונה באותו אורך ואז להציב את האינדקס של אותו אות בתא המתאים לו ב-symbol וככה ה-decoder ידע את כל מה שצריך לגבי כל אות ואורך.
17:	נקדם למילת הקוד הבאה בתור, כדי שלא ניתן ל-2 תווים את אותו מילת קוד.
18:	סיום

דוגמה פרקטית

לעבור על הדוגמה רק אחרי מעבר על ההסבר של הקוד.

נתונים:

	1	2	3	4	5	6	7
$\sum$ =	a	b	c	d	e	f	g

	1	2	3	4	5	6	7
$\{\ell_i\}$ =	3	3	4	4	3	2	2

נעבור לפי שורות הקוד.

שורה 2: האורך הגדול ביותר הוא 4 ולכן $maxlength = 4$.

שורה 3-5: ניצור מערך חדש num בגודל $maxlength$:

	1	2	3	4
num[] =	0	0	0	0

שורה 6-7: המטרה של num היא לספור את מספר מילות הקוד באותו אורך, כל אינדקס מייצג אורך.

לכן, נעבור בלולאה ולכל מילת קוד באורך ℓ_i נספור בתא המתאים מערך (+1):

	1	2	3	4
num[] =	0	2	3	2

שורה 8-11:

המערך $firstcode$ מייצג את מילת הקוד הראשונה בכל בלוק.

בגלל שאנחנו עוברים בסדר יורד וכל פעם רוצים לדעת מהי מילת הקוד בבלוק הבא אז פשוט נחלק ב-2.

זאת מאחר ואמרנו כי המספרים מיוצגים באופן בינארי אז זה כמו לעשות Shifting:

	1	2	3	4
firstcode[] =	2	2	1	0

שורה 12-13:

המערך $nextcode$ אומר מהי מילת הקוד לכל תו בבלוק.

נאתחל אותו להיות firstcode שמגדיר לנו מהי מילת הקוד הראשונה בבלוק.

	1	2	3	4
nextcode[] =	2	2	1	0

שורות 14-17:

נעבור בלולאה על כל תו בא"ב, נמצא את מילת הקוד הנכונה לו, נבנה את symbol עבור ה-decoder.

המערך $codeword$ מייצג לנו את מילת הקוד של כל תו i.

בהמרה לקוד בינארי של כל $codeword[i]$, הקוד הבינארי שנמיר צריך להיות באותו אורך של תו ה-i והוא ייצג את המסלול בעץ הקנוני:

	1	2	3	4	5	6	7
$codeword[]$ =	0	0	0	0	0	0	0

אומנם לא מוצג בקוד האתחול של $symbol[][]$ אך נאתחל אותו לטובת הדוגמה.
העמודות יהיו לפי גודל ההפרש שיש בחישוב בשורה 16.
השורות יהיו לפי אורך מילות הקוד.

symbol[][]=	0	1	2
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

נציג את איטרציות הלולאה:

i=1 , 'a'

	1	2	3	4	5	6	7
codeword[] =	1	0	0	0	0	0	0

	1	2	3	4
nextcode[] =	2	2	2	0

symbol[][]=	0	1	2
1	0	0	0
2	0	0	0
3	1	0	0
4	0	0	0

i=2, 'b'

	1	2	3	4	5	6	7
codeword[] =	1	2	0	0	0	0	0

	1	2	3	4
nextcode[] =	2	2	3	0

symbol[][]=	0	1	2
1	0	0	0
2	0	0	0
3	1	2	0
4	0	0	0

i=3 , 'c'

	1	2	3	4	5	6	7
codeword[] =	1	2	0	0	0	0	0

	1	2	3	4
nextcode[] =	2	2	3	1

symbol[][]=	0	1	2
1	0	0	0
2	0	0	0
3	1	2	0
4	3	0	0

i=4 , 'd'

	1	2	3	4	5	6	7
codeword[] =	1	2	0	1	0	0	0

	1	2	3	4
nextcode[] =	2	2	3	2

symbol[][]=	0	1	2
1	0	0	0
2	0	0	0
3	1	2	0
4	3	4	0

i=5 , 'e'

	1	2	3	4	5	6	7
codeword[] =	1	2	0	1	3	0	0

	1	2	3	4
nextcode[] =	2	2	4	2

symbol[][]=	0	1	2
1	0	0	0
2	0	0	0
3	1	2	5
4	3	4	0

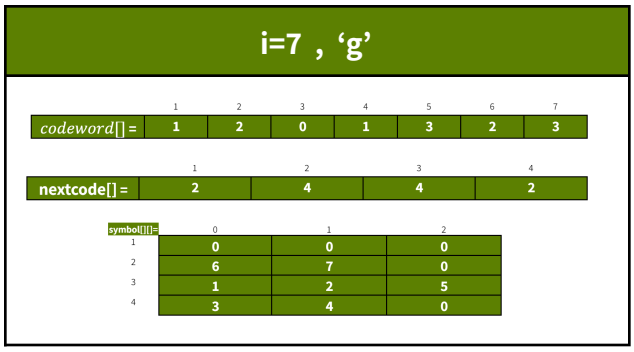
i=6, 'f'

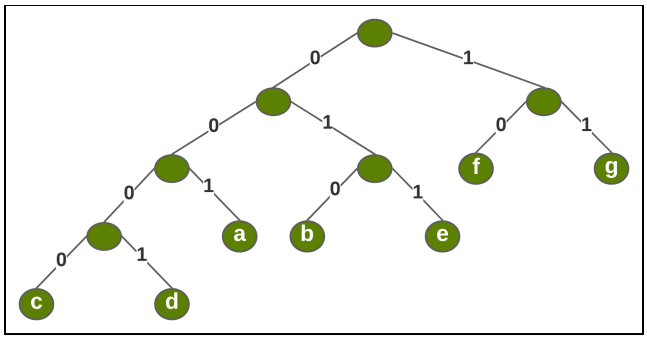
	1	2	3	4	5	6	7
codeword[] =	1	2	0	1	3	2	0

	1	2	3	4
nextcode[] =	2	3	4	2

symbol[][]=	0	1	2
1	0	0	0
2	6	0	0
3	1	2	5
4	3	4	0

ונקבל את העץ הקנוני הבאה, כל מסלול הוא בעצם מילת קוד מתוך מערך $codeword[i]$ שנמיר לבינארי באורך של אותו מילה ℓ_i:





פענוח של הפמן קנוני

המשתנה v מצביע בכל שלב על הסיבית הבאה.

המשתנה l סופר מה אורך מילת הקוד הנוכחית.

Canonical Huffman Decoding Algorithm

```
1:  CanonicalHuffmanDecoding(){
2:        v ⇐ nextInputBit()
3:        l ⇐ 1
4:        while v < firstcode[l] do:
5:              v ⇐ 2v + nextInputBit()
6:              l++
7:        end-while
8:        // the integer v is a valid codeword of l bits
9:  return symbol[l , v - firstcode[l]
10: // return the index of the decoded symbol
11: }
```

הרעיון הוא עבור כל אות נפעיל את הקוד הזה.
אם הערך המספרי v גדול מהfirstcode אז אנחנו עוצרים את הלולאה ומפענחים באמצעות הטבלה symbol ואז נחזיר תו x כלשהו.
אם v קטן ממנו אז זה אומר שלא הגענו לאורך המתאים ואנחנו ממשיכים וקוראים את הסיבית הבאה.

Canonical Huffman Decoding Algorithm - **הסבר קוד**	
1:	
2:	נקבל את הסיבית הראשונה v.
3:	הביט הראשון שאנחנו קוראים הערך שלו הוא 1 כי בהתחלה קוראים רק ביט אחד.
4:	תמיד אם v הוא סיבית אחת אז הוא יהיה קטן מה- $firstcode[1]$ ולכן זה תמיד יכריח אותנו לקרוא את הסיבית הבאה.
5:	כל פעם אצביע על הסיבית הבאה בתור בקלט. 2v זה למעשה לתרגם לייצוג עשרוני כדי לעשות shift ימינה, ונוסיף nextInputBit כדי לקבל אותו ערך עשרוני שמתאים לסיביות האלה.
6:	כדי להגיד שקראנו סיבית נוספת.
7:	
8:	
9:	נחזיר את האות הרלוונטית.
10:	
11:	

כלומר ב-symbol זו טבלה דו מימדית שתעזור לנו לפענוח, אינדקס השורה מתאים לאורכים ואינדקס העמודה כותב את מילת הקוד שמתאימה לאותו בלוק אחד אחרי השני כלומר 0,1,2 אלו הם האינדקסים.
כלומר, מילת הקוד הראשונה תהיה בעמודה 0 ואז מילת הקוד הבאה ב-1 וכו'...

s_i	ℓ_i	codeword[i]	Code
1 (a)	2	1	01
2 (b)	5	0	00000
3 (c)	5	1	00001
4 (d)	3	1	001
5 (e)	2	2	10
6 (f)	5	2	00010
7 (g)	5	3	00011
8 (h)	2	3	11

num	0	3	1	0	4
firstcode	2	1	1	2	0

symbol[ℓ_i ,j]

	0	1	2	3
1				
2	1(a)	5(e)	8(h)	
3	4(d)			
4				
5	2(b)	3(c)	6(f)	7(g)

קוד הופמן D-Ary

משפט

בעץ הופמן D-Ary מלא עם k צמתים פנימיים יש $k(D-1)+1$ עלים.

הוכחה

נוכיח באינדוקציה על k. (לא נוכיח בקורס).

משפט

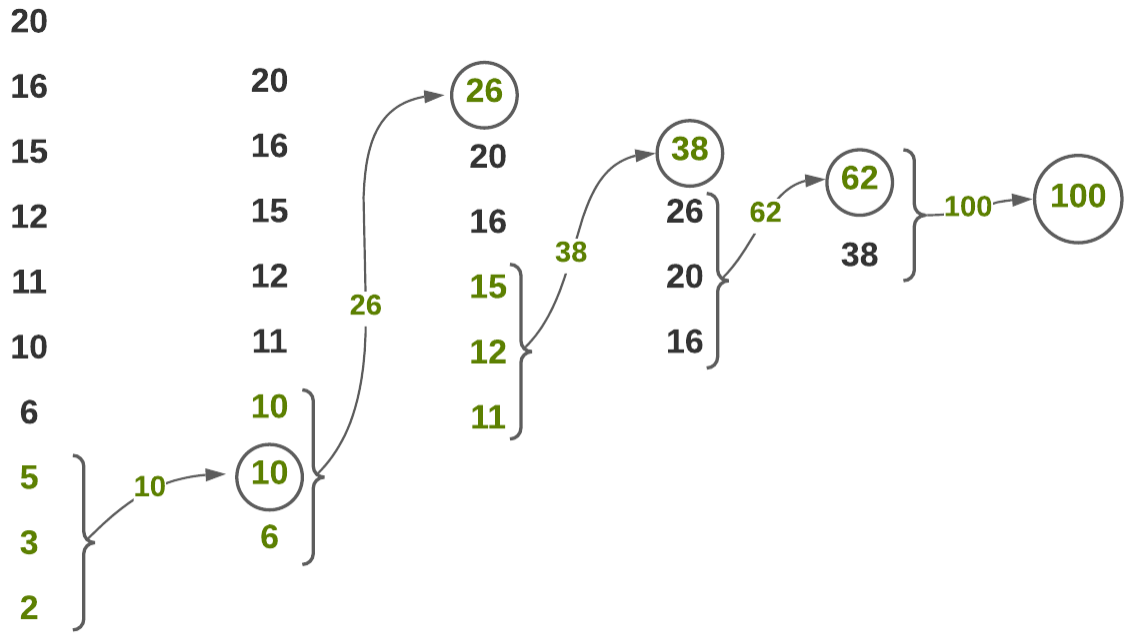
ללא הוכחה.

בעץ אופטימלי:

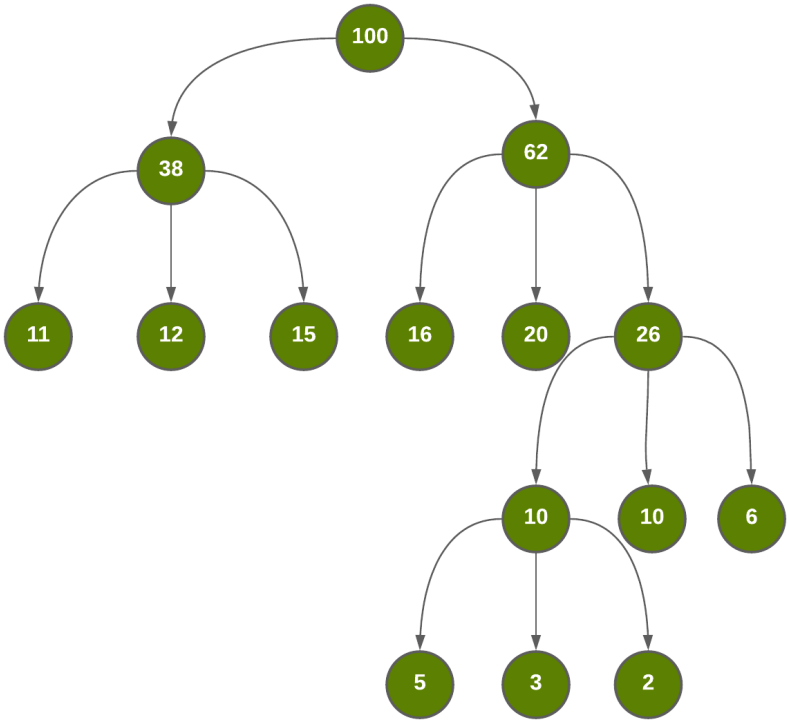
- יש לפחות 2 עלים ברמה התחתונה.
- שני הצמתים עם המשקלים הנמוכים ביותר נמצאים ברמה התחתונה בעץ.
- שני המשקלים הנמוכים ביותר יכולים להיות אחים.

דוגמה

נסתכל על עץ $ary-3$ ונרצה לבנות עץ הופמן אופטימלי עבור השכיחויות הבאות:



ונקבל את העץ הבא:



מה הבעיה כאן?

העץ שהצגנו לא אופטימלי, כי לשורש רק 2 בנים ועדיף שמספר הבנים של כל צומת יהיה קטן יותר ברמה הנמוכה ולא בגבוהה.

בנוסף ראינו לפי ההגדרה לאופטימליות של עץ D-Ary ש:

- יש לפחות 2 עלים ברמה התחתונה.
- שני הצמתים עם המשקלים הנמוכים ביותר נמצאים ברמה התחתונה בעץ.
- שני המשקלים הנמוכים ביותר יכולים להיות אחים.

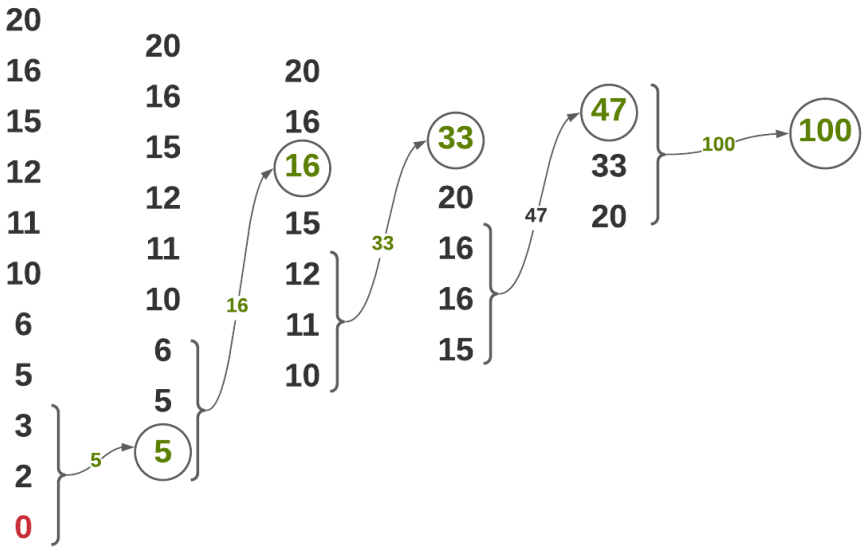
לכן, אם נתונים מספר שכיחויות זוגי, אנחנו נוסיף צומת פיקטיבית עם משקל 0 על מנת לקבל עץ אופטימלי.

כלומר אנחנו נוסיף צומת פיקטיבית כדי להשלים למספר העלים הרצוי ואז נפעיל את האלגוריתם.

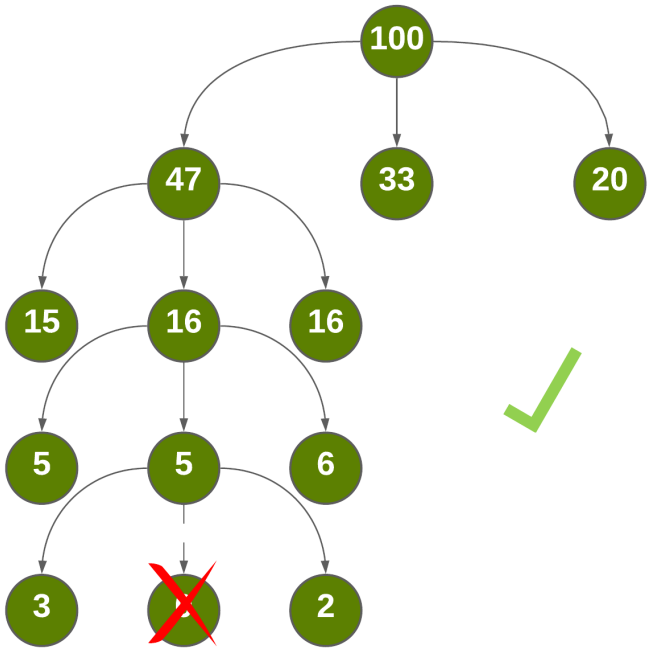
למשל, עבור הדוגמה מקודם ראינו שיש לנו 10 שכיחויות, נוסיף עוד שכיחות שהיא '0' (הוא בטוח יהיה הכי נמוך בעץ) ואז נתחיל לאחד את שלושת התחתונים כמו שעשינו בעבר.

במקרה הזה, אנחנו באמת נקבל עץ אופטימלי.

כלומר, נוסיף את הצומת הפיקטיבית '0' ונחשב מחדש ונקבל:



העץ שנקבל הוא לא מלא, אך הוא אופטימלי, אנחנו יכולים להוסיף את האפס, אבל מכיוון שהיא פיקטיבית נשמיט אותה מהעץ ונקבל עץ אופטימלי:



אלגוריתם לאופטימליות של עץ D-Ary

1. תשלים עם n_0 צמתים עם משקל 0 לרשימת הצמתים ההתחלתית n כך ש:

$$n + n_0 = n' \cdot (D - 1) + 1$$

כאשר:

- n זה מספר השכיחויות ההתחלתי.
- n_0 זה מספר הצמתים שעלינו להוסיף.
- n' זה מספר הצמתים הפנימיים.
- $n + n_0$ זה מספר העלים בעץ

בדוגמה שלנו $10 + n_0 = 2n' + 1$ והוספנו צומת פיקטיבית אחת ולכן $n_0 = 1$.

2. בכל שלב נאחד את ה-D שכיחויות הכי נמוכות.
3. הקצאה שרירותית $0, 1, \ldots, D - 1$ לצלעות מהצומת החדש לצמתים שאוחדו.
4. ניצור את מילות הקוד בעלים.

קוד הופמן דינמי

מבוא

- באלגוריתם הפמן דינמי אנחנו לא מקבלים עץ קבוע מראש, אלא, מילות הקוד שלי משתנות ככל שנקרא עוד ועוד...
- השכיחויות שלו עולה, נקדם אותו במעלה העץ וניתן לו את מילת הקוד הקצרה ביותר על מנת לחסוך בסיביות.
- **המטרה שלנו** - לקודד את התו שמופיע ב-x_t ונרצה להסתמך על ה-t תווים הראשונים.

ניצור את מילת הקוד ל-x_t על פי $x_1, \ldots, x_{t-1}$ כלומר, על סמך הסימנים שכבר קראתי.

יתרונות

- נעבור פעם אחת בלבד על הקובץ ובמהלך המעבר גם נלמד את הסטטיסטיקות וגם נדחוס את הקובץ.
- לא צריך לפענח את הנתונים של עץ קנוני.
- מתחילים עם התפלגות אחידה וכל פעם שרואים תו a חדש, נקדם את השכיחות של a ואם מקדמים אותו מספיק אז ניתן לו מילת קוד קצרה יותר (כלומר אנחנו נעלה אותו בעץ למעלה).
- אין צורך להעביר את העץ לצד השני ולכן אין צורך ב-Prelude.

חסרונות

- הרבה פחות יעיל ולכן לא באמת משתמשים בזה בחיים האמיתיים.

המפענח בונה בדיוק את העץ שה-encoder היה בונה, הוא מפענח כל תו ומתקן את העץ בהתאם.
לצורך זה, נגדיר את **Sibling Property - תכונת האחים.**

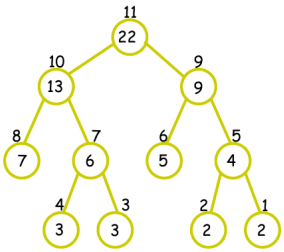
Sibling Property - תכונת האחים

עץ מקיים את תכונת האחים ⇔ הוא עץ האפמן.
ניתן למיין את האחים בסדרה עולה כך שנוכל למספר 2 אחים במספרים עוקבים.
בתכונת האחים נרצה שיתקיים:

1. $weight(v) = weight(left(v)) + weight(right(v))$
2. הצמתים יכולים להיות בסדר עולה לפי משקלם כך ש: $2j - 1$ וגם $2j$ הם אחים.

דוגמה

כאמור, מדובר בעץ האפמן וכפי שניתן לראות, כל זוג אחים הם מספרים עוקבים.
מכאן, אם מדובר בעץ האפמן, אז דרישה (2) של תכונת האחים מתקיימת.



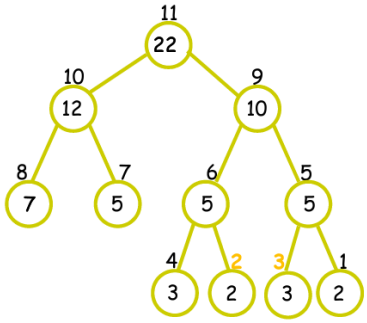
7 5 3 3 2 2
7 5 4 3 3
7 6 5 4
9 7 6
13 9
22

אם עץ האפמן ⇐ תכונת האחים - דיי אינטואיטיבי.
אך מה קורה בכיוון ההפוך לגרירה?

דוגמה

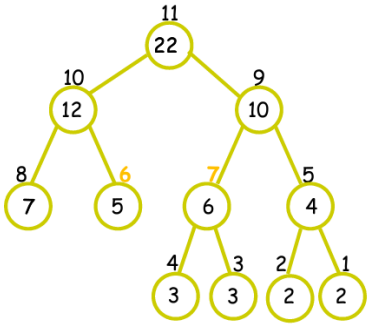
בדוגמה זו נציג מקרה בו העץ לא האפמן ולא מתקיימת בו תכונת האחים.

לפי התכונות של עץ אופטימלי הוכחנו ששני המשקלים הכי נמוכים יכולים להיות אחים, אך הם לא חייבים.
נבצע את השינוי על מנת להראות שאם העץ הוא לא האפמן אז התכונה של המספרים העוקבים לאחים לא מתקיימת.



7 5 3 2 3 2
7 5 5 3 2
7 5 5 5
10 7 5
12 10
22

לא ניתן למספר לפי אחים עוקבים ולכן זה לא עץ האפמן.
נראה שזה נכון גם לגבי הרצה פנימית בעץ:



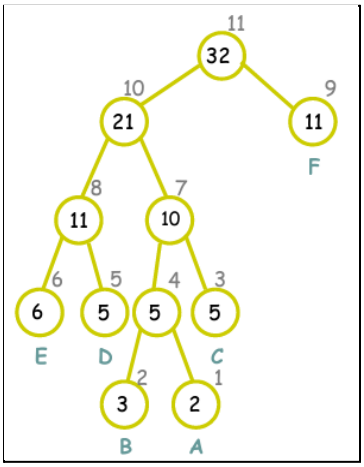
7 5 3 3 2 2
7 5 4 3 3
7 5 6 4
10 7 5
12 10
22

חזרה להאפמן דינמי

אז אמרנו שבהאפמן דינמי נעשה:

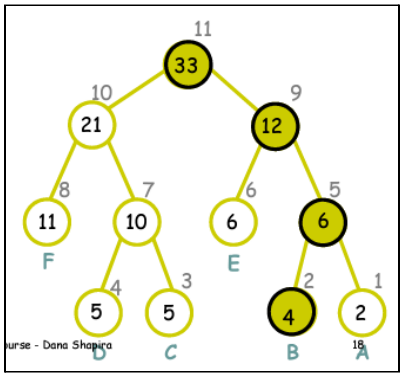
- נשנה את העץ במהלך הקידוד.
- נאסוף סטטיסטיקות במהלך הקידוד.
- ככל שהשכיחות עולה, נקדם במעלה העץ וניתן לו קידוד קצר יותר.

המטרה שלנו - ליצור עץ בינארי שמתעדכן.



נניח שזה עץ האפמן שנמצא בשלב ביניים כלשהו וגם אנו יודעים איך להגיע אליו.
למשל, עבור הצומת F זה אומר שעד עכשיו ראינו 11 פעמים את האות הזאת.

כעת, נניח שהתו הבא שאנו קוראים הוא B, עד עכשיו ראינו אותו 3 פעמים ועכשיו נראה את הרביעי.
המקודד והמפענח יבנו את אותו עץ האפמן במקביל כדי שהמקודד לא יצטרך להעביר עץ למפענח.
ייתכן שהמקודד תמיד יהיה שלב אחד לפני המפענח כי הוא יודע מה הוא הולך לקרוא.
תחילה, נעביר את המילה B לפי המודל הקודם 0100 ונעדכן את השכיחות.
אנחנו לא יכולים פשוט להפוך ל-4 את הצומת של B כי אז יווצר מצב שבו לאחים **אין** מספרים עוקבים ואז לא תתקיים תכונת האחים.
לאחר קידוד B העץ יראה כך כעת קיבלנו עץ האפמן שגם מקיים את תכונת האחים:



<u>התהליך הוא:</u>

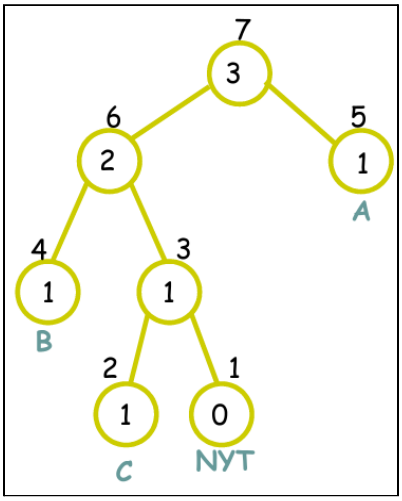
1. עדכון משקלי האבות הקדמונים.
2. אם יש משקלים זהים, נעבור לצומת עם האינדקס המקסימלי.

צומת-0

- כאשר לא רוצים לשלוח ב-Prelude את הא"ב אז צריך להשתמש בצומת-0 לתווים החדשים שנתקל בהם בכל פעם.
- צומת-0 תקרא **NYT** או **N**ot **Y**et **T**ransmitted שמשקלו יהיה 0.

שלבים: בכל פעם שנתקלים בתו חדש בפעם הראשונה:

1. נעביר את מילת הקוד של התו ל-NYT
2. נעביר את קוד ה-ASCII של התו
3. נקצה לתו הזה עלה בעץ
4. נעדכן את כל העץ בהתאם

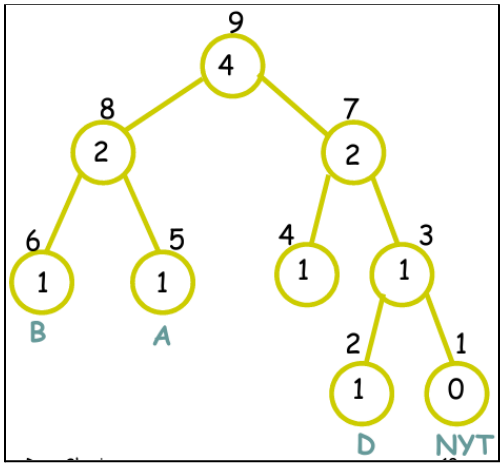


כאשר אנחנו מתחילים לקודד, אנו לא יודעים מי הא"ב שמשתתף בקובץ.
כיצד נודיע ל-decoder (המפענח) שעלינו להוסיף עלה חדש עם תו חדש?
נסמן NYT = את מילת הקוד שאנחנו נתקלים בה בפעם הראשונה.
ננית שעד עכשיו ראינו A, B, C וזה העץ שלנו:

כעת, ראינו לראשונה את התו החדש D.

1. תחילה נעביר את הקידוד של ה-NYT בהתאם למבנה העץ.
2. עלינו ליצור ל-NYT שני ילדים שאחד מהם יהיה ה-NYT החדש והשני יהיה D שראינו לראשונה.
3. נעדכן משקלים בשיטה שהוסברה על מנת לשמור על תכונת האחים.

נקבל את העץ הבא:



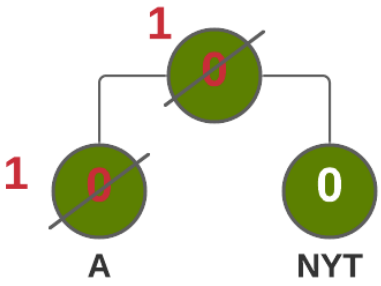
אתחול של עץ האפמן דינמי

מתחילים מעץ עם NYT ונניח שצריך לקרוא את ABC:

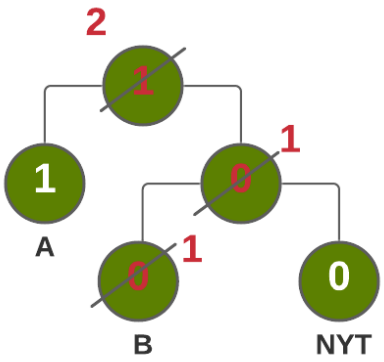
ברור למפענח שהתו הראשון הוא תו חדש ולא צריך להעביר את הקידוד של A כי הוא פשוט תו ריק ε.

כלומר, עץ שהוא ריק זה בעצם $\varepsilon = NYT$.

לכן, נעביר רק את קוד ה-ASCII של A למפענח ונפצל את העץ לפי האלגוריתם:



נוסיף את B :



וככה הלאה גם עבור C...

Dynamic Huffman Algorithm

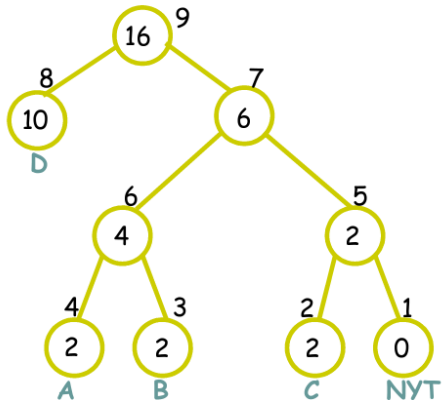
```
1:  Update()
2:        q ⇇ leaf(x_t)
3:        if (q is the 0-node)
4:              replace q by a parent 0-node with two 0-node children;
5:              q ⇇ left child;
6:        if (q is a sibling of a 0-node)
7:              interchange q with the highest numbered leaf of the same
                weight.
8:              increment q's weight by 1;
9:              q ⇇ parent of q
10:       while (q ≠ root)
11:             interchange q with the highest numbered node of the same
                weight.
12:             increment q's weight by 1;
13:             q ⇇ parent of q
14:       increment q's weight by 1;
```

Dynamic Huffman Algorithm - הסבר קוד

1:	
2:	אנחנו מקודדים את x_t על סמך מה שראינו עד עכשיו ומחפשים את העלים שמתאים ל-x_t.
3:	אם q הוא NYT אז:
4:	עלינו ליצור 2 ילדים עם משקלים 0.
5:	ונשים את q כבן השמאלי.
6:	אם q הוא אח של NYT אז:
7:	נחליף את q עם העלה בעל האינדקס הכי גבוהה עם אותו משקל.
8:	נגדיל את המשקל של q ב-1.
9:	כעת ניקח את ההורה של q ונשווה אותו ל-q.
10:	כל עוד לא הגענו לשורש אז:
11:	נחליף את ההורה עם ההורה עם האינדקס הכי גבוהה באותו משקל.
12:	נגדיל את המשקל של q ב-1.
13:	כעת ניקח את ההורה של q ונשווה אותו ל-q.
14:	נצא מהלולאה ונגדיל שוב את המשקל של q ב-1.

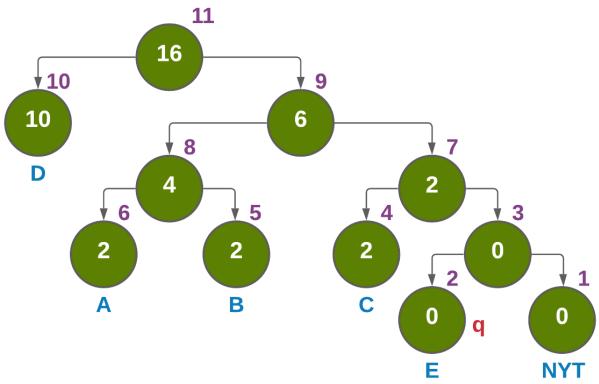
תרגיל

נציג תרגיל כדוגמה לאלגוריתם עדכון עץ האפמן דינמי וגם נציג את כל השלבים לצורך הבנה.
ננית שבשלב מסוים באלגוריתם עץ הופמן דינאמי קיבלנו את העץ הבא:



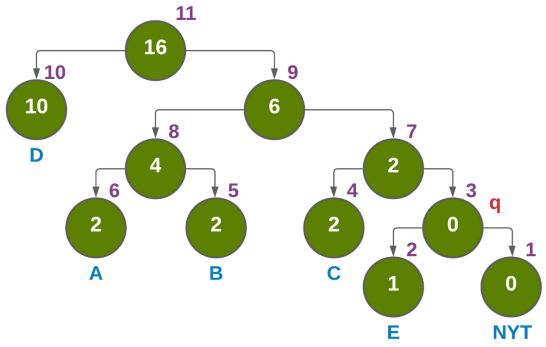
עלינו להכניס $EEEE$:

1. נתקלנו ב-E בפעם הראשונה, עלינו להעביר ל-Encoder את 111 (זה ה-NYT) וגם להעביר את ה-ASCII של האות E.
2. נסמן ב-q את ה-NYT ואז ניצור לו שני ילדים עם משקל 0 ונשים את q בבן השמאלי.



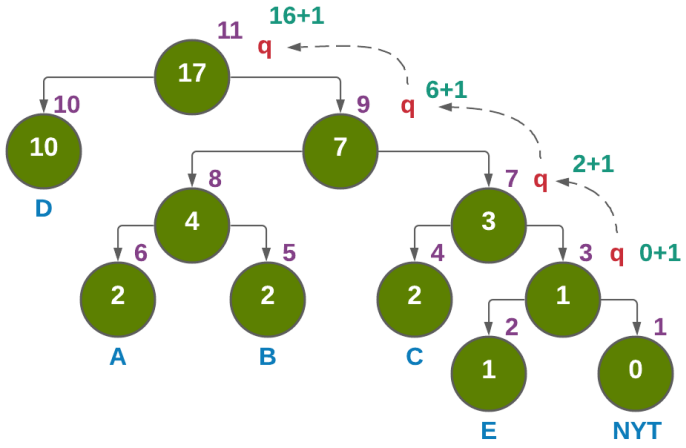
3. מכיוון ש-q הוא עכשיו הבן השמאלי הוא תמיד יהיה העלה עם האינדקס הכי גבוהה ולכן השאלה הזאת מיותרת (העלה הכי גבוהה במשקל 0).
לכן, נעלה את המשקל של q ב-1 ונעלה אותו לאבא.

נקבל:



4 אין לנו עוד צומת עם משקל 0 מאינדקס גבוה יותר ולכן נעבור לפקודה הבאה:

נעלה את הערך של q ב-1 ונקדם את q בהתאם ככה עד לשורש:



סיימנו להכניס E ראשון, נכניס עוד E:

1 נעביר את הקידוד של E שהוא 1110.

2 ה-q אינו NYT ולכן התנאי לא מתקיים.

3 q הוא אח של NYT ולכן התנאי מתקיים - אין עוד עלה עם משקל 1 ולכן אין צורך לבצע החלפה.

נקדם את הערך של q ל-2 ונעלה אותו לאביו.

4 נבצע את ה-while כלומר נעדכן את האבות הקדמונים ומשקלים שלהם ונבצע החלפות במידת הצורך.

וככה הלאה לכל ה-E הנוספים...

עץ שלד הופמן - Skeleton

לפני שנדבר על עץ שלד, בהגדרה של עץ קנוני ראינו לא רק עץ שמשוך שמאלה אלא גם לימינה.
נסתכל על עץ שמשוך ימינה ונראה איך שם מטמינים את מילות הקוד.
אם נזכר, היה לנו את firstcode, nextcode והיה לנו את הטבלה symbol בשביל הפענוח.

הגדרה

- כאשר אנחנו סורקים את העלים משמאל לימין, אם נסתכל על העומקים שלהם, יש לנו סדרה מונוטונית עולה של אורכי מילות הקוד.
- ההגדרה הזאת היא בניגוד להגדרה הקודמת בה העץ היה משוך שמאלה.
- כשאנחנו בונים עץ משוך ימינה, מילת הקוד הארוכה ביותר היא רצף של אחדות.

אלגוריתם לעצים קנונים

1. מחפשים את עץ ההאפמן עם אורכים $l_1,....,l_n$ עבור האיברים.
2. ממיינים את האיברים בהתאם לאורכים.
3. הקצה לכל איבר את ה-l_i ביטים הראשונים אחרי הנקודה הבינארית של: $\sum_{j=1}^{i-1} 2^{-l_j}$

דוגמה:

אות	A	B	C	D	E	F	G
אורך	3	2	4	2	3	3	4

אנחנו נמיין אותם אך שהפעם הם יהיו מהקטן לגדול באופן הבא:

Item	l_i	2^{-l_i}	$\Sigma_{j=1}^{i-1} 2^{-l_j}$
B	2	0.01	0.**00**000
D	2	0.01	0.**01**000
A	3	0.001	0.**100**00
E	3	0.001	0.**101**00
F	3	0.001	0.**110**00
C	4	0.0001	0.**1110**0
G	4	0.0001	0.**1111**0

אם נסתכל על B שהוא 2^{-2} כאשר $l_i = 2$ אז בבסיס של 2 אנחנו נקבל 0.01 .

למה קיבלנו מספר כזה כשבר (נקודה)? כי לפי בינארי מתקיים לדוגמה כללית:

2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
4	2	1	½	¼	⅛
100	10	01	0.1	0.01	0.001

וכעת, נחבר את כל השלבים הקודים שהיינו בהם ועד לשלב הנוכחי.

- כאשר נרצה לחשב את מילת הקוד של B אז ה-i הולך עד 1-1 כלומר עד 0 אז הטור הוא ריק והסכום הריק הוא 0.
- כאשר נרצה לחשב את מילת הקוד של D אנו מחשבים למעשה את 0.01 כלומר:

$$\sum_{j=1}^{2-1=1} 2^{-l_j} = 2^{-l_1} = 2^{-2} = 0.01000$$

- כאשר נרצה לחשב את מילת הקוד של A אנו מחשבים כך:

$$\sum_{j=1}^{3-1=2} 2^{-l_j} = 2^{-l_1} + 2^{-l_2} = 2^{-2} + 2^{-2} = 0.01 + 0.01 = 0.10000$$

מוטיבציה - עץ שלד

כאשר קראנו חלק ממילת הקוד והגענו לבלוק שבו כל אורכי המילים הוא באורך קבוע מסוים אז נדע ישר כמה תווים נשאר לנו לקרוא, ונוכל לקרוא אותם בבת אחת.

- אנחנו לא צריכים לשמור את כל העץ.
- לדוגמה, אם מילת הקוד מתחילה ב-1101 אז זה צריך להיות באורך של 9 ביטים. לכן, אנחנו יכולים לקרוא את ה-5 ביטים הבאים בתוך הבלוק.

כלומר, במקום לתפוס סיבית, סיבית, אנחנו יכולים לתפוס את כל ה-5 סיביות בפעם אחת ולפענח.

- אנחנו נזהה את אותו הפתיח המינימלי שיאפשר לי לראות את אורך מילת הקוד.

```
    ...
 62 11001101
 63 110011100
 64 110011101
    ...
    1101...
125 111011010
126 1110110110
127 1110110111
    ...
199 1111111111
```

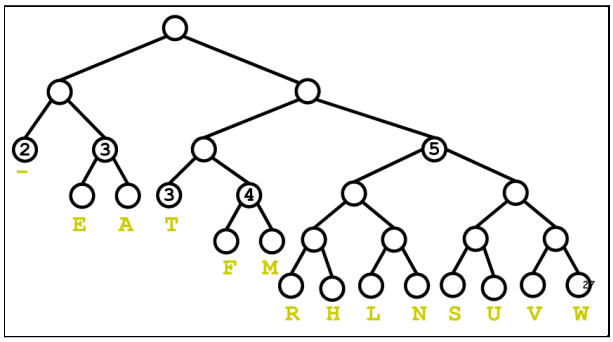
מה זה עץ שלד?

- זה עץ האפמן קנוני שמשוך ימינה שבו נתייחס לתתי-עצים שלמים ואנחנו מקצצים אותם אם הגובה שלהם $h \geq 1$.
- הפריפיקס הוא הביטוי הקצר ביותר הנחוץ כדי לזהות את אורך המילים.
- העלים v , מכילים את האורכים של מילות הקוד המתאימים ונסמן אותם בתור $value(v)$.

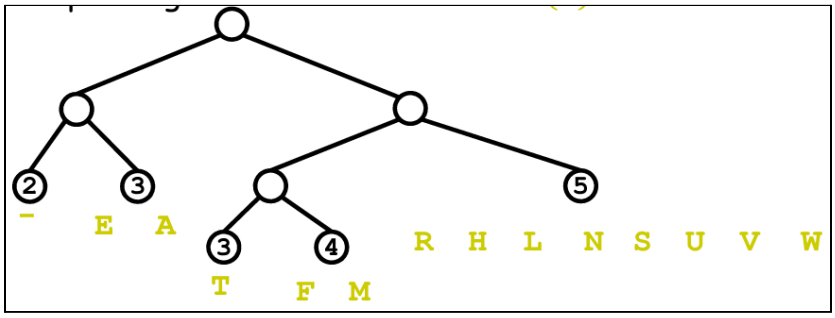
דוגמה:

ניקח את העץ הקנוני בציור, העץ משוך ימינה ונתייחס לתתי-עצים שלמים.

- למשל הצומת ה-(5) הוא שורש של תת-עץ שהוא שלם ובעץ שלד אנחנו נקצץ את המספרים האלה.



- נקצץ כל אותם תתי-עצים מגובה של $h \geq 1$, אותם תתי-העצים שמייצגים את האורכים של מילות הקוד. לאחר הקיצוץ אנחנו נקבל את העץ בציור מלמטה.
- ניקח את העלים v, לכל אחד ה-$value(v)$ יחזיר את אורך מילות הקוד.
- אם v היא צומת פנימית נגדיר מראש כי $value(v) = 0$ ואם הוא עלה, אז הוא יחזיר את הערך בו.



- בהרצאה הראו דוגמה להשוואה בין עץ האפמן רגיל לבין עץ האפמן שלד. לפי הדוגמה אפשר לראות כמה השלד יותר יעיל מהאפן הרגיל כי הוא חוסך לנו משמעותית במספר "הצמתים בעץ"
  - בעץ האפמן רגיל - 399 צמתים.
  - בעץ האפמן שלד - 49 צמתים.

אנחנו נציג כאן כמות גדולה של הגדרות שאמורות להיות פשוטות בדוגמאות אחר כך:

הגדרות

- n_i זה מספר מילות הקוד באורך i.

- m מוגדר להיות $m = min\{i \mid n_i > 0\}$.
כלומר נרצה לדעת מאיפה להתחיל, לכן נצטרך לדעת מה אורך מילות הקוד המינימלי שלנו.

- $base(i)$ - מתייחס למילת הקוד הראשונה בבלוק ה-i והוא יהיה הערך *העשרוני* של מילת הקוד.
  - באופן ישיר $base(m) = 0$.
  - $base(i) = 2 \cdot (base(i-1) + n_{i-1})$

- $B_s(k)$ - מסתכלים על רצף בינארי עם s-סיביות של שלם k, מרפדים באפסים אם צריך.

- מילות הקוד ה-j^{th} עם אורך i עבור $\left(j = 0,\ 1,\ \ldots.\ ,\ n_i - 1\right)$ הם $B_i(base(i) + j)$.

- $seq(i)$ - האינדקס של מילת הקוד הראשונה בבלוק ה-i.
  - תמיד מתקיים כאשר $seq(m) = 0$.
  - $seq(i) = seq(i-1) + n_{i-1}$.

גם כאן, מספיק האינדקס של מילת הקוד הראשונה בבלוק ואין צורך לדעת את האינדקסים של כל מילות הקוד.

- $I(w)$ - הערך המספרי של מחרוזת בינארית w.
  - אם w באורך l אז נקבל ש: $B_l(I(w)) = w$.

- $I(w) - base(l)$: אינדקס יחסי של מילת קוד w בבלוק של מילת הקוד באורך l, כאשר w באורך l סיביות.

- $seq(l) + I(w) - base(l)$: אינדקס יחסי של מילת-קוד w ביחס לכל הרשימה של מילות הקוד, כאשר w באורך l סיביות.
  - ניתן גם לרשום: $diff(l) = base(l) - seq(l)$ עבור $I(w) - diff(l)$.

- לכן כל אחד צריך רשימה של $diff(l)$.

דוגמה להגדרות שהצגנו

נציג בעזרת טבלת הנתונים הבאה (בסוף העמוד) ונבין את הקשר ביניהם להגדרות שהצגנו עד כה:

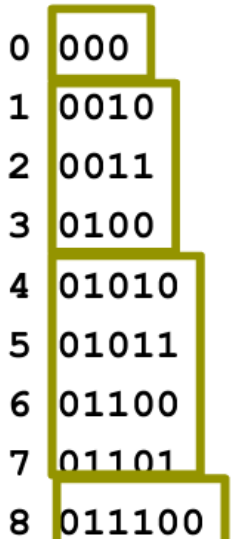
עבור שורה הראשונה:

- $l = 3$ כלומר אורך של 3.
- $n_3 = 1$ כלומר יש רק מילה אחת באורך 3 ואפשר לראות את זה בציור מימין:
- $base(3) = 0$ כי הגדרנו $base(m) = 0$ ובאמת $m = 3$ כי הוא האורך הקצר מכולם.
- $seq(3) = 0$ כי הגדרנו $seq(m) = 0$.
- $diff(3) = 0$ כי הגדרנו: $diff(l) = base(l) - seq(l)$.

עבור שורה השניה:

- $l = 4$ כלומר אורך של 4.
- $n_4 = 3$ כלומר יש 3 מילים באורך 4 ואפשר לראות את זה בציור מימין:
- $base(4) = 2$ כי $base(4) = 2(base(4-1) + n_{4-1}) = 2(0+1) = 2$
- $seq(4) = 1$ כי הגדרנו $seq(4) = seq(4-1) + n_{4-1} = 0 + 1 = 1$.
- $diff(4) = 1$ כי הגדרנו: $diff(4) = base(4) - seq(4) = 2 - 1 = 1$.

ℓ	n_l	$base(\ell)$	$seq(\ell)$	$diff(\ell)$
3	1	0	0	0
4	3	2	1	1
5	4	10	4	6
6	8	28	8	20
7	15	72	16	56
8	32	174	31	143
9	63	412	63	349
10	74	950	126	824



אלגוריתם פענוח

- אם $value(v) = 0$ אז v הוא צומת פנימית של העץ, אחרת, נותנים את האורך של מילת הקוד ומקצצים את תת-העץ השלם המיוצג בעזרת v.
- עבור $1 \leq j \leq n$ הסימון $table(j)$ אומר שלכל מילת קוד j קיים תרגום שאומר מה הא"ב הזה.

Skeleton Huffman Trees Decoding Algorithm

```
 1:  tree_pointer ⇐ root
 2:  i ⇐ 1
 3:  start ⇐ 1
 4:  while i < length_of_string do:
 5:        if string[i]=0 then:
 6:              tree_pointer ⇐ left(tree_pointer)
 7:        else:
 8:              tree_pointer ⇐ right(tree_pointer)
 9:        if value(tree_pointer) > 0 then:
10:              codeword ⇐ string[start… (start+value(tree_pointer)-1)]
11:              output ⇐ table[ I(codeword) - diff[value(tree_pointer)] ]
12:              tree_pointer ⇐ root
13:              start ⇐ start + value(tree_pointer)
14:              i ⇐ start
15:        else:
16:              i++
```

Skeleton Huffman Trees Decoding Algorithm - **הסבר קוד**	
1:	מתחילים מהשורש, הפוינטר יקבל את השורש.
2:	ה-i מצביע על האינדקס במחרוזת - מצביע כרגע על התו הראשון במחרוזת.
3:	ה-start מקבל 1 הוא פרמטר שיגיד לנו מה ההתחלה של כל מילת קוד שניפגש בעץ.
4:	כל עוד לא סיימנו לעבור על כל המחרוזת אז:
5:	אנחנו נעשה טיול בעץ, הטיול הזה זה אותו טיול שעשינו לעץ הפמן המקורי. כלומר אם התו הנוכחי הוא 0 אז שמאלה ואם הוא 1 הוא ימינה. לכן, בשורת הקוד הזאת אם הביט הנוכחי הוא 0 אז נרצה ללכת שמאלה לכן:
6:	הפוינטר יצביע על לבן השמאלי.
7:	אחרת אנחנו הולכים ימינה כלומר:
8:	הפוינטר יצביע על לבן הימני.
9:	ה-if הזה דרוש בגלל השלד. אם הערך של הפוינטר גדול מאפס, כלומר אם הגענו לעלה אז:
10:	אפשר לקבל מנקודה זאת את מילת הקוד כי הגענו במסלול הזה מהשורש ועד לעלה הנוכחי. לכן, אז נדע בדיוק מה לשלוף, מילת הקוד שלנו codeword ילך מ-start עד כל אורך מילת הקוד. והפחות אחד זה כי אם אנחנו קוראים למשל מילת קוד באורך של 5 אז צריך לקרוא מ-start עוד 4 מקומות כי אנחנו כבר עומדים על המקום הראשון שלנו.
11:	אנחנו נכנס לטבלה עם המיקום של מילת הקוד בהמרה מבינארי ל-integer (בגלל זה מסומן לנו ו) ובטבלה הזאת של כל הא"ב שלנו נרצה למצוא את האינדקס כדי לדעת באיזה מקום בטבלה להיכנס על מנת לפענח ולכן חיסרנו כי זה בדיוק ההיסט הזה מראש הטבלה כאשר ה-diff זה בעצם לפי ההגדרה $diff(l) = base(l) - seq(l)$.
12:	נחזור לשורש.
13:	את ה-start נקפיץ בבת אחת עם אותו ערך כלומר נתקדם מכל הביטים שעברנו עליהם באיטרציה הנוכחית לעבר ההתחלה של מילת הקוד הבאה.
14:	נתקדם למילת הקוד הבאה, כלומר ה-i ימשיך למילת הקוד הבאה.
15:	אחרת, כלומר אם עדיין לא הגענו לעלה אז:
16:	נקדם את ה-i, כלומר נמשיך לטייל בעץ עד שנמצא עלה.

עץ שלד מצומצם

- לכל צומת v נגדיר את עץ השלד הבא:
  - אם v הוא עלה אז:

$$lower(v) = upper(v) = value(v)$$

  - אם v היא צומת פנימית אז:

$$lower(v) = lower(left(v))$$
$$upper(v) = upper(right(v))$$

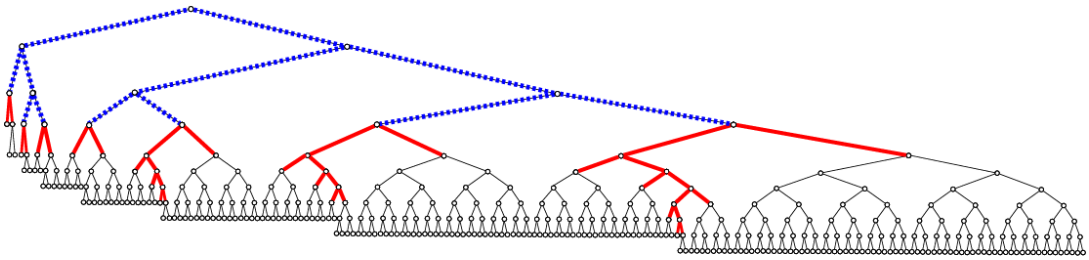
- עץ שלד מצומצם הוא תת העץ הקטן ביותר מתוך עץ השלד המקורי עבור כל העלים w מתקיים:

$$upper(w) \leq lower(w) + 1$$

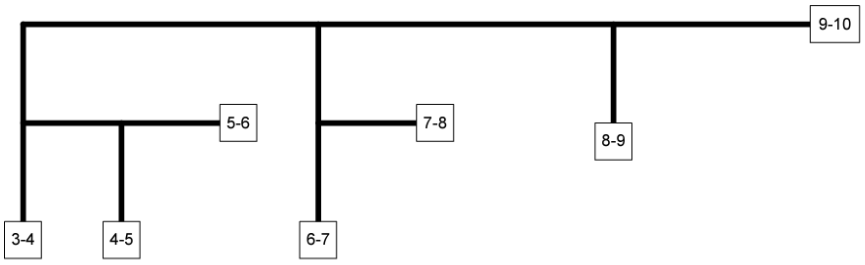
- עץ השלד המצומצם גוזם את עץ שלד האפמן בצומת פנימי שבו אורך מילות הקוד הנוכחיות שייך לקבוצה של גודל לכל היותר 2.
- למשל בציור הבא מלמטה, האדום זה עץ שלד והכחול זה עץ שלד מצומצם.

נשים לב בכחול, לכל עלה w ה-$upper(w)$ זה הימני וה-$lower(w)$ זה השמאלי ואכן מתקיים התנאי.

- לדוגמה, עבור $w = 111$ אז נלך על הכחול ימינה, ובאמת $lower(w) = 9$ וגם $upper(w) = 10$ ובאמת מתקיים $upper(w) \leq lower(w) + 1$.



- ואם נסתכל על אותו העץ בדוגמה מלמעלה, נציג אותו בצורה יותר מינימלית (מצומצמת):
וגם כאן עבור $w = 111$ נקבל את אותו הדבר.



קוד אלגוריתם עץ שלד מצומצם:

Skeleton Huffman Reduced Trees Algorithm

```
 1:  tree_pointer ⇐ root
 2:  i ⇐ 1
 3:  start ⇐ 1
 4:  while i < length_of_string do:
 5:        if string[i]=0 then:
 6:              tree_pointer ⇐ left(tree_pointer)
 7:        else:
 8:              tree_pointer ⇐ right(tree_pointer)
 9:        if value(tree_pointer) > 0 then:
10:              len ⇐ value(tree_pointer)
11:              codeword ⇐ string[start… (start+len-1)]
12:              if flag(tree_pointer)==1 and 2*I(codeword)>=base(len+1):
13:                    codeword ⇐ string[start…(start+len)
14:                    len++
15:              output ⇐ table[ I(codeword) - diff[len] ]
16:              tree_pointer ⇐ root
17:              start ⇐ start + value(tree_pointer)
18:              i ⇐ start
19:        else:
20:              i++
```

קוד אריתמטי

נציג מה זה קוד אריתמטי לפי ההשוואה בינו לבין קוד האפמן:

קוד האפמן	קוד אריתמטי
כל תו בא"ב מקבל מילת קוד	מחליפים את ההודעה כולה במספר יחיד של floating-point בטווח מסוים (הרבה סיביות אחרי הנקודה).
הוא צריך התפלגויות (חוץ מדינאמי שלומד בתהליך)	לא צריכים התפלגויות (חוץ מאריתמטי סטטי).
כבד לשינוי (לעדכן התפלגות)	קוד אדפטיבי קל
טבלה או עץ שיגיד את המיפוי בין מילות הקוד לא"ב	לא צריך לשלוח את המודל בשביל הפענוח כי המפענח מתחזק את כל המודל בעצמו
יש לנו הגבלה - משתמשים במספר שלם של ביטים כדי לקודד כל מילת קוד	משתמשים בשברי סיביות בשביל מילת קוד אחת.
הוכחנו שהאפמן אופטימלי אבל ההבדל בינו לבין האנטרופיה נמדדת בסיבית אחת לכל אות וזה יכול להצטבר כלומר בטקסט בגודל n יצבר לנו כמות של n סיביות וזה פחות טוב.	בקוד אריתמטי נשפר יותר טוב מהאפמן. כלומר נראה שהוא אופטימלי.

דוגמה

character	probability	Huffman Code
A	0.95	0
B	0.02	11
C	0.03	10

- האנטרופיה במקרה הזה הוא $0.335\ bps$ שזה bits per symbol.
- האורך הממוצע הוא $1.05\ bps$.

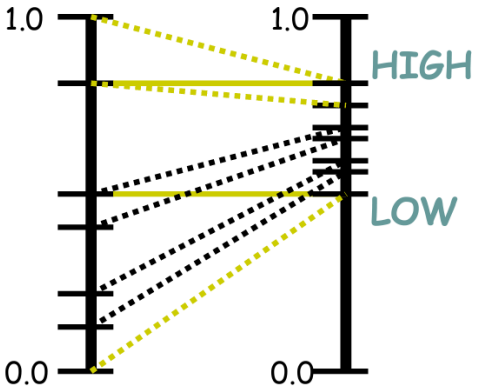
אפשרות חדשה לשיפור

נעבור לזוגות של תווים ונבנה עץ האפמן לפי ההתפלגות החדשה.

character	probability	Huffman Code
AA	0.9025	0
AB	0.019	111
AC	0.0285	100
BA	0.019	1101
BB	0.0004	110011
BC	0.0006	110001
CA	0.0285	101
CB	0.0006	110010
CC	0.0009	110000

- קיבלנו את אותו האנטרופיה כמו מקודם $0.335\ bps$.
- ממוצע האורכים הוא $1.222\ bps$ על כל זוג תווים ולכן עבור כל תו נחלק ב-2 ונקבל $0.611\ bps$.

קוד אריתמטי סטטי



1. מתייחסים תמיד לקטע החצי פתוח $[0, 1)$.
2. מחלקים לתתי-קטעים לפי ההתפלגות שיש לנו לכל תו.
3. כל תת-קטע הוא תו והקטע שלו חצי פתוח.
4. ברגע שנרצה לקודד תו מסוים נתמקד בקטע של אותו תו.
5. את תת-הקטע (**אינטרוול**) נסמן ב- $[LOW,\ HIGH)$.

דוגמה

S_i	p_i	low_bound	high_bound
A	0.67	0.0	0.67
B	0.11	0.67	0.78
C	0.07	0.78	0.85
D	0.06	0.85	0.91
E	0.05	0.91	0.96
F	0.04	0.96	1.0

$$low\_bound(s_i) \leftarrow \sum_{j=1}^{i-1} p_j$$

$$high\_bound(s_i) \leftarrow \sum_{j=1}^{i} p_j$$

אלגוריתם קידוד אריתמטי

Encoding Arithmetic Code Algorithm

```
1:  low ⇐ 0.0
2:  high ⇐ 1.0
3:  while input symbols remain do:
4:          range ⇐ high - low
5:          get symbol
6:          high ⇐ low + high_bound(symbol)*range
7:          low ⇐ low + low_bound(symbol)*range
8:  end-while
9:  output any value in [low, high)
```

דוגמה לקידוד

נתמקד בקטעי הקוד הבאים:

```
6:          high ⇐ low + high_bound(symbol)*range
7:          low ⇐ low + low_bound(symbol)*range
```

- עבור A נחשב:
  - $high = 0 + (0.67) \cdot 1 = 0.67$
  - $low = 0 + 0.0 \cdot 1 = 0.0000$
- עבור B נחשב:
  - $high = 0 + (0.78) \cdot 0.67 = 0.5226$
  - $low = 0 + (0.67) \cdot 0.67 = 0.4489$
- עבור A נוסף נחשב:
  - $high = 0.4489 + (0.67) \cdot 0.0737 = 0.0493$

M[i]	Low	High	Range
-	0.0000	1.0000	1.0000
A	0.0000	0.67	0.67
B	0.4489	0.5226	0.0737
A	0.4489	0.498279	0.049379
A	0.4489	0.48198393	0.03308393
A	0.4489	0.47106623	0.02216623
E	0.46907127	0.47017958	0.00110831
A	0.46907127	0.46981384	0.00074257
A	0.46907127	0.46956879	0.00049752
B	0.46940461	0.46945934	0.00005473
A	0.46940461	0.46944128	0.00003667

S_i	Low bound	High bound
A	0.0	0.67
B	0.67	0.78
C	0.78	0.85
D	0.85	0.91
E	0.91	0.96
F	0.96	1.0

8

הצגה בינארית

בסוף התהליך של האלגוריתם צריך איזשהו מספר בין ה-low לבין ה-high, לא משנה איזה העיקר שהוא בתת-הקטע. אך מכיוון שאנחנו בתוך הקורס של דחיסת נתונים, דווקא מעניין אותנו לבחור גם את המספר הכי קצר בתוך הקטע הזה ולכן נציג את הרעיון הבא:

- low = **0.0111100000101**010111010
- high = **0.0111100000101**101010011

- אם נסתכל על הבינארי, אנחנו נרצה לקחת מספר בין ה-low וה-high שהוא הקטן ביותר שאפשר להציג באותו תת-קטע, למשל בדוגמה שלנו לקחנו את כל הרישא שצבועה בכחול והוספנו עוד מספרים כך שניהיה בתוך תת-הקטע עצמו.
- לכן, אנחנו נוסיף 3 סיביות לסוף שהם 100 כלומר:

0.0111100000101100

והמספר שהוספנו מייצג בעצם את מספר השבר הבינארי הנמוך ביותר בין ה-low וה-high.

סוף קלט למחרוזות סופיות

מתי נדע שסיימנו את ההודעה?

למשל, אם המפענח מקבל '0' ונניח ואנחנו לפי סדר לקסיקוגרפי, איך נדע שזה מייצג את ההודעה:

$$a, aa, aaa, aaaa, ...$$

הוא לא יודע עד מתי צריך לפענח הרי ה-0 נמצא גם בקטע של a וגם בקטע של aa וכו'...

ולכן יש לנו 2 שיטות כדי להגיד ל decoder להפסיק לפענח:

1. באמצעות $, כלומר נוסיף תו חדש שלא נמצא בא"ב המקורי עם הסתברות מאוד נמוכה שברגע שהוא מגיע לפענח את ה-$ הוא ידע לעצור.
2. אומרים לו מה אורך ההודעה ואז ברגע שהוא מגיע לאורך ההודעה הוא ידע לעצור.

דוגמה

- נניח שההסתברות של A היא 0.9 וה-$ שלנו זה עשירית (הסתברות המשלים של 0.9) אז כדי להגיע ל-A הראשון אנחנו בקטע $[0.0\ ,\ 0.9]$ וה-$ יכנס בשארית הקטנה ממש בין $[0.9,\ 1.0)$.
- ועכשיו שוב עבור ה-A הבא נחלק לפי אותו יחס, אנחנו ניתן 0.09 ל-$ ולתת הקטע כלומר ניתן $[0.81,\ 0.9)$ ועבור ה-A ניתן את תת הקטע $[0.0\ ,\ 0.81)$.
- וככה הלאה...

	high	low
Initial state	1.0	0.0
A	0.9	0.0
A	0.81	0.0
A	0.729	0.0
A	0.6561	0.0
A	0.59049	0.0
A	0.531441	0.0
A	0.4782969	0
$	0.4782969	0.43046721
output	0.45	

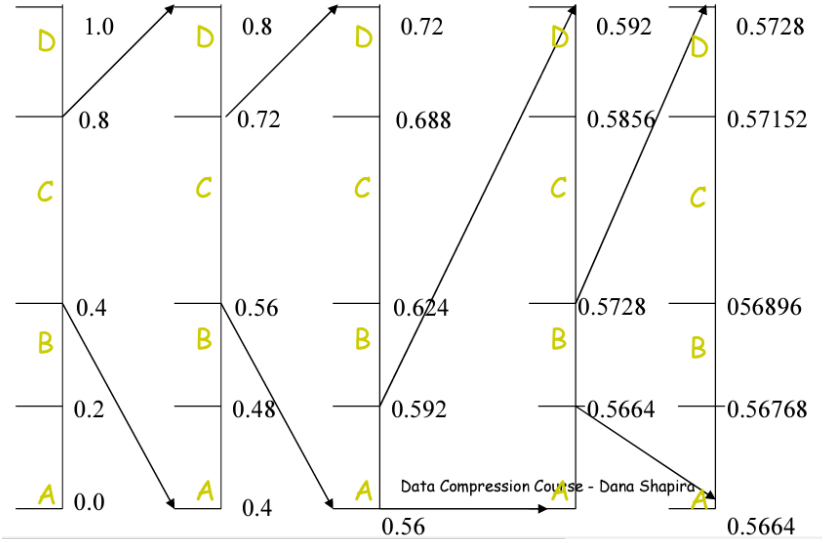
- בסופו של דבר צריך לתת מספר שנמצא בתוך הקטע, תמיד 0 יהיה בתוך תת-הקטע של כל A שעברנו עליו.
- ה-$ הוא כמו EOF שאנחנו מכירים בתכנות.
- אחרי שעשינו 7 פעמים A והצטמצמנו, נלך לתת-הקטע של ה-$ ונראה את גודל הקטע שלו.

שיטות לפענוח

מה שחסר לנו זה איך אנחנו מפענחים את זה?

<u>שיטה 1</u>

אנחנו מקבלים את המספר $E(M) = 0.572$ ואנחנו צריכים לדעת מה ההודעה שהובילה למספר הזה.
אז אנחנו אמורים לדעת מה ההתפלגות ומה הסדר של התווים בתתי-הקטעים מ-0 עד ל-1.



נסביר בשלבים:

- התו הראשון בהודעה הוא C כי 0.572 יצטמצם לתת-הקטע של C (כי הוא נמצא בתוכו) ולכן נעשה "זום-אין" לתוך C כלומר נסתכל לתוך $[0.4,\ 0.8]$ כלומר פענחנו עד כה "C".
- נשאל שוב, איפה יושב עכשיו 0.572 והוא יושב שוב על C כלומר פיענחנו עד כה "CC".
- נעשה שוב "זום-אין" , לקחנו את תת הקטע ושוב חילקנו לאותם פרופורציות ונשאל שוב איפה נופל עכשיו 0.572 והוא נופל ב-A ולכן פיענחנו עד כה "CCA".
- וככה הלאה עד שנפענח את "$CCABD$".

נשים לב שבשיטה הזו עשינו המון חישובים כי היינו צריכים לחשב את הגודל של תתי-הקטעים.
בשיטה השניה אנחנו נחסוך את זה:

שיטה 2

יש לנו decoding שמנטרל את ההשפעה של התו האחרון.

1. מקבלים את המספר שאותו אנחנו רוצים לפענח.
2. מחפשים את אותו תת הקטע שהוא נמצא שם.
3. נחשב את ה-range לפי $high - low$ של ה-symbol הנוכחי.
4. ה-encoded יקבל $(encoded - low[symbol])/range$.

לצורך השוואה

בשיטה הקודמת לקחנו $low \cdot range$ או $high \cdot range$.

כאן אנחנו עושים את הפעולה ההפוכה ולכן פה יש לנו פחות חישובים כי במקום לחשב כל פעם את תתי הקטעים החדשים אנחנו משנים את המספר שאותו אנחנו מפענחים וכך חוסכים בחישובים נוספים.

Decoding Arithmetic Code Algorithm

```
1:  encoded ⇐ get(encoded number)
2:  do {
3:          Find symbol whose range contains encoded
4:          Output the symbol
5:          range ⇐ high(symbol) - low(symbol)
6:          encoded ⇐ (encoded - low(symbol))/range
7:  } until (EOF)
```

דוגמה לפענוח לפי שיטה 2

Symbol	Low	High
B	0	0.25
I	0.25	0.50
L	0.50	1.00

נרצה לקודד את $E = 0.109375$, במקום לחשב כל פעם את תתי הקטעים אנחנו משנים את E ונניח שהקטעים שלנו הם כפי שמתואר בטבלה משמאל:

1. המספר נופל בקטע של B ולכן נחשב:
   $E = (0.1093 - 0.0)/0.25 = 0.4375$.
2. המספר 0.4375 נופל בקטע של I ולכן נחשב: $E = (0.4375 - 0.25)/0.25 = 0.75$.
3. המספר 0.75 נופל בקטע של L ולכן נחשב: $E = (0.75 - 0.5)/0.5 = 0.5$.
4. המספר 0.5 נופל בקטע של L ולכן נחשב: $E = (0.5 - 0.5)/0.5 = 0.0$ וכיוון שקיבלנו כעת את המספר $E = 0$ אז נעצור.

וקיבלנו את הפענוח: $BILL$.

דוגמה קיצונית ומסקנה חשובה

נתונים לנו

- $\Sigma = \{a, b\}$ ונסמן $N = 2 = |\Sigma|$.
- ההסתברות ל-a היא 0.999 ול-b היא 0.001.
- נניח שההודעה M יש לנו 999 פעמים a ו-b פעם אחת.

אם נבדוק את גודל תת-הקטע בסוף התהליך:

1. בפעם הראשונה גודל הקטע יהיה 1
2. בפעם השניה גודל תת-הקטע יהיה 0.999
3. בפעמים השלישית $(0.999)^2$
4. וכו'...

- אם נחשב את גודל הrange האחרון של הקידוד אנחנו נקבל מספר $3.7 \cdot 10^{-4}$
- אם נחשב את כמות האינפורמציה נקבל 12
- אם צריך לכתוב את ההודעה הזאת בקוד אריתמטי אז נחשב ב-13 סיביות.

שאלה

אם נבנה את זה בקוד האפמן כמה סיביות אנחנו צריכים? - 1000 סיביות.

כלומר, במקום 1000 סיביות בקוד אריתמטי אנחנו צריכים 13 סיביות שזה 0.013 bps בממוצע.

מסקנה

תמיד הקוד האריתמטי יהיה לא גרוע מהאפמן.

חישוב גודל של קטע

ניקח גודל תת-הקטע האחרון ונסמן אותו במספר חדש: $[low, high) = [l,\ l + r)$ כאשר r גודל הקטע.

יש לנו הודעה M עם k תווים שאותה נרצה לתפוס: $M = m_1 m_2 \ldots m_k$.

למעשה, כל תת קטע כזה, הגודל שלו מצטמצם לפי ההסתברות.

צריך $\lceil \log_2\left(\frac{1}{r}\right) \rceil$ ביטים כדי לייצג את r כאשר מוגדר:

$$r = \prod_{i=1}^{k} p(m_i)$$

- גודל האינטרוול (תת-הקטע) האחרון יהיה זהה למכפלת ההסתברויות כולה.

דוגמה

נסתכל על הטבלה משמאל ונתחיל לקודד את $BILL$.

Symbol	Low	High
B	0	0.25
I	0.25	0.50
L	0.50	1.00

	low	high	range
	0	1	1
B	0	$0.25 = \frac{1}{4}$	$0.25 = \frac{1}{4}$
I	$0 + \frac{1}{4} \cdot \frac{1}{4} = \frac{1}{16}$	$0 + \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}$	$\frac{1}{8} - \frac{1}{16} = \frac{1}{16}$
L	$\frac{1}{16} + \frac{1}{16} \cdot \frac{1}{2} = \frac{3}{32}$	$\frac{1}{16} + 1 \cdot \frac{1}{16} = \frac{1}{8}$	$\frac{1}{8} - \frac{3}{32} = \frac{1}{32}$
L	$\frac{3}{32} + \frac{1}{2} \cdot \frac{1}{32} = \frac{7}{64}$	$\frac{8}{64}$	$\frac{8}{64} - \frac{7}{64} = \frac{1}{64}$

מצד אחד אם אנחנו מכפילים את ההסתברויות אנחנו מקבלים את המספר $\frac{1}{64}$.

מצד שני לפי האלגוריתם, הגודל של תת-הקטע האחרון זה $\frac{1}{64} = \frac{7}{64} - \frac{1}{8}$.

כלומר, גודל תת-הקטע האחרון יהיה זהה למכפלת ההסתברויות.

מספר הסיביות

את תת הקטע האחרון נסמן $[l,\ l + r)$ כך ש-r מייצג לנו את גודל תת הקטע הזה (האינטרוול הזה).

בדוגמה שלנו $r = \frac{1}{64}$, אנחנו צריכים להעביר בסופו של דבר את המספר בתת-הקטע האחרון, ככל שתת-הקטע יותר קטן, אנחנו נצטרך יותר סיביות כדי לייצג את המספר הזה.

למעשה, *מספר הסיביות תלוי בגודל תת-הקטע שלנו*.

ראינו שגודל תת-הקטע האחרון הוא מכפלת ההסתברויות בהודעה שלנו.

לכן באמת מתקיים:

$$\Pr(B) \cdot \Pr(I) \cdot \Pr(L) \cdot \Pr(L) = \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{64}$$

באופן כללי, נרצה לקחת את כמות האינפורמציה של תת-הקטע האחרון כלומר $\lceil \log_2\left(\frac{1}{r}\right) \rceil$ ולראות כמה ביטים יקח לנו לייצג מספר בתוך תת-הקטע האחרון.

דוגמה

נגדיר:

- k זה אורך ההודעה.
- n - זה גודל הא"ב.

נתונים לנו:

$$n = |\Sigma| = 3 \quad , \quad k = |M| = 9 \quad , \quad M = abacbbaac \quad , \quad \Sigma = \{a, b, c\}$$

נסמן את השכיחות של כל תו:

$$w_a = 4, \; w_b = 3, \; w_c = 2$$

מה אנחנו מנסים להראות פה? - למעשה, אנחנו רוצים לחשב את כמות האינפורמציה של האינטרבל האחרון.
במקרה שלנו:

$$-\log_2(r) = -\log_2(\Pr(a)\Pr(b)\Pr(a)\Pr(c)\cdot.....\cdot\Pr(a)\Pr(c)) =$$

לפי חוקי לוגריתמים: $\log_2(xy) = \log_2(x) + \log_2(y)$ ולכן במקרה שלנו:

$$= -\log_2(\Pr(a)) - \log_2(\Pr(b)) - \log_2(\Pr(a)) - ..... - \log_2(\Pr(c)) =$$

כעת נשאל לדוגמה כמה פעמים יש לנו את הביטוי $-\log_2(\Pr(a))$? , יש לנו 4 פעמים כלומר w_a פעמים.
לכן נקבל:

$$= w_a\left(-\log_2(\Pr(a))\right) + w_b\left(-\log_2(\Pr(b))\right) + w_c\left(-\log_2(\Pr(c))\right) =$$

נסמן: $p_i = \frac{w_i}{k}$ ונחזור למקרה שלנו:

$$= -\sum_{j=1}^{3} w_j \log_2(p_j) = -9 \cdot \sum_{j=1}^{3} \frac{w_j}{9} \log_2(p_j) = 9 \cdot \left(-\sum_{j=1}^{3} p_j \cdot \log_2(p_j)\right) = 9H$$

כלומר קיבלנו שגודל ההודעה הדחוסה זה מספר התווים k כפול האנטרופיה H.

בנוסחה כללית

$$-\log_2(r) = -\log_2\left(\prod_{i=1}^{k} p(m_i)\right) = k \cdot H$$

קוד אריתמטי אדפטיבי

במקום שאנחנו נעבוד ונעשה preprocessing כדי ללמוד את השכיחויות ואז נדחוס, אנחנו למעשה לומדים את ההסתברויות והסטטיסטיקות במעבר אחד. במהלך המעבר על הקובץ, אנחנו גם לומדים את הקובץ הנתון וגם דוחסים באותו שלב. נכון שבהתחלה הדחיסה שלנו לא כל כך טובה (בגלל זמן הלמידה) ולפעמים זה לא נכון לחשוב בשיטה שמה שלמדנו בעבר יתקיים גם בהווה אך זה מאפשר לנו הערכה טובה להמשך.
המקודד צריך לשלוח למפענח את הא"ב הסדור, את ההודעה הדחוסה ואת אורך ההודעה.

לדוגמה

נניח שהא"ב שלנו הוא $\Sigma = \{a, b, c\}$
וההודעה שלנו היא $bccb$.
נסמן ב-$N(x)$ להיות מספר המופעים של כל סימן x שראינו עד הנקודה הנוכחית במעבר על ההודעה.
אנחנו מדברים על אלגוריתם אדפטיבי אז כל פעם העבר מאפשר לנו הערכה לקידוד התו הנוכחי.
לכן, $N(x)$ מדבר על התו שראינו עד הרגע.
למשל:

$$p(a) = \frac{N(a)+1}{N(a)+N(b)+N(c)+3}$$
$$p(b) = \frac{N(b)+1}{N(a)+N(b)+N(c)+3}$$
$$p(c) = \frac{N(c)+1}{N(a)+N(b)+N(c)+3}$$

ההסתברות הראשונית של כל תו היא $\frac{1}{3}$ ואז על כל תו נוסף אנחנו הולכים ולומדים יותר בהסתברות שונה בהתאם לצורת ההודעה.

דוגמה

נשתמש בדוגמה הקודמת שהצגנו ונציג שלבי אינטרוולים:
נניח שהא"ב שלנו הוא $\Sigma = \{a, b, c\}$
וההודעה שלנו היא $bccb$.

1. *<u>אינטרוול ראשון</u>* $[0, 1)$ *<u>:</u>*

$$p(a) = p(b) = p(c) = \frac{1}{3}$$
$$N(a) = N(b) = N(c) = 0$$

2. *<u>קידקוד תו ראשון בהודעה **b** שזה אינטרוול</u>* $[0.333,\ 0.6667)$ *<u>:</u>*

$$p(a) = p(c) = \frac{1}{4}\ ,\ p(b) = \frac{1}{2}$$
$$N(a) = N(c) = 0,\ \ N(b) = 1$$

3. *קידוד תו שני בהודעה **c** שזה אינטרוול* $[0.583,\ 0.6667)$ *:*

$$p(a) = \frac{N(a)+1}{N(a)+N(b)+N(c)+3} = \frac{1}{5}\ ,\ \ p(b) = p(c) = \frac{2}{5}$$
$$N(a) = 0,\ \ N(b) = N(c) = 1$$

4. *קידוד תו שלישי בהודעה **c** שזה אינטרוול* $[0.6334,\ 0.6667)$ *:*

$$p(a) = \frac{0+1}{0+1+2+3} = \frac{1}{6},\ \ p(b) = \frac{1+1}{0+1+2+3} = \frac{2}{6},\ \ p(c) = \frac{2+1}{0+1+2+3} = \frac{3}{6}$$
$$N(a) = 0,\ \ N(b) = 1,\ N(c) = 2$$

5. *קידוד תו רביעי ואחרון בהודעה **b** שזה אינטרוול* $[0.639583,\ 0.6501)$ *:*

$$p(a) = \frac{1}{0+2+2+3} = \frac{1}{7},\ \ p(b) = p(c) = \frac{2+1}{0+2+2+3} = \frac{3}{7}$$
$$N(a) = 0,\ \ N(b) = 2,\ \ N(c) = 2$$

לכן, קידוד ההודעה הוא בכל מספר מתוך האיטרוול $[0.639583,\ 0.6501)$ למשל המספר 0.64.

חסרונות של קוד אריתמטי אדפטיבי

- **דיוק** - יכולת הדיוק מוגבלת ולכן צריך לדמות עם Integer.
- **זה לא "On the fly"** - אי אפשר לעשות בצורה של String אבל אפשר לפתור את זה.
- **איטי יותר** מ-Huffman במיוחד עם סטטי וסמי-סטטי.
- **גישה ישירה** - למשל אם יש קובץ שדחוס עם האפמן ואנו מעוניינים במשהו שנמצא בחצי השני של הקובץ, אפשר להתחיל מהחצי של הקובץ הדחוס ולהתחיל לפרוש משם. מצד שני, בקוד אריתמטי אם נרצה לגשת לחצי השני של הקובץ אנחנו נקבל ג'יבריש.

קוד אריתמטי מול קוד האפמן

בטקסט ארוך כלשהו לא קיימת אפילו מילה אחת עם האות e.
האות e היא האות הכי נפוצה בשפה האנגלית.
אם ננסה לדחוס את הפסקה הזאת לפי ההתפלגות הידועה של האנגלית אז לא נדחוס את זה בצורה טובה, אנחנו צריכים לעבור על הטקסט הזה ששם ההתפלגות היא שונה כי ראינו מקרה ש-e לא קיים בכלל.
אז נרצה להשתמש בהסתברויות ידועות כדי לדחוס קובץ אחר.

לסיכום, היתרונות של קוד אריתמטי הם:

- הוא מגיע לאנטרופיה
- אפשר להפוך אותו לאדפטיבי

אבל מצד שני, החסרונות של קוד אריתמטי הם:

- לוקח לו יותר זמן לדחיסה ופענוח
- כוח דחיסה קטן

דחיסה מילונית

לדחיסה מילונית יש כמה גישות:

1. **הגישה הסטטית**
   a. המילון נבנה לפני הקידוד
   b. דרוש ידע קודם על הקובץ שמקודדים (מילות קוד, סטטיסטיקות וכו'...).
2. **הגישה האדפטיבית**
   a. המילון נבנה תוך כדי הקידוד.
3. **הגישה הסטטיסטית**
   a. מנסה לחזות את הסמל הבא בכל שלב
4. **הגישה ההחלפתית**
   a. במקום להמציא את הגלגל מחדש, רואים את המחרוזת והיא הופיעה כבר אז במקום להמציא אותה שוב, פשוט נעתיק אותה.

מה זה מילון סטטי? - זה מילון שמישהו כתב מראש ונתן לנו.

דוגמה למילון סטטי VS מילון אדפטיבי

בהינתן מילון ומחרוזת - איך שוברים אותה לבלוקים?

1. **השיטה החמדנית** - מכל מקום ניקח את המחרוזת הארוכה ביותר שנמצאת במילון.
   נעשה זאת כך: היא תתחיל במיקום הנוכחי וכל פעם היא תיקח את המחרוזת הארוכה ביותר שניתן לתפוס במקום הנוכחי, ההתקדמות היא פר-תו.
   כלומר, a - נמצא במילון, aa נמצא במילון אבל aaa לא נמצא במילון אז נעצור פה וניקח את הכניסה שמתאימה ל-aa.
2. **שיטת Longest Fragment** - ניקח את המחרוזת הארוכה ביותר שמופיעה במילון ולכן פה לא עוברים משמאל לימין, אלא צריך לעבור על המחרוזת באופן רקורסיבי ולמצוא את המחרוזת הארוכה ביותר - שזה חסרון כי זמן הריצה פה הוא אקספוננציאלי.
3. **שיטת Min Words** - נרצה למצוא את מספר מילות הקוד המינימליות שניתן לפרק אותן. נחפש חלוקה כך שאנחנו נבצע חלוקה של המחרוזות לכמה שפחות מילים, כלומר כמה שפחות כניסות למילון.
4. **השיטה האופטימלית** - יש לנו מידע חיצוני שנותן לנו את הפתרון הטוב ביותר.

ניתוח אופטימלי עבור מילון נתון

נתון:

D מילון.

λ פונקציה שממפה אותנו למילות הקוד.

T הטקסט.

המטרה:

למצוא חלוקה של $T = w_1 w_2 .... w_k$ למילות-קוד, כל מילה נמצאת במילון $w_i \in D$, ושהחלוקה תביא לכך שהסכום של מספר הביטים בכל קידוד כזה יהיה מינימלי. כלומר, נרצה את הסכום הבא שיהיה מינימלי:

$$\sum_{i=1}^{k} \left|\lambda(w_i)\right|$$

דחיסת LZ77

ב-LZ77 הטקסט מחולק ל-2 חלקים, הנקודה הנוכחית היא בדיוק בחוצץ בין 2 החלקים.

1. Window - זה חלון למה שכבר קודדנו.
2. Look ahead buffer - מה שעומדים לקודד.

הרעיון הוא שמאיפה שאנחנו נמצאים בכל נקודה נתחיל לקודד לפי מה שראינו קודם לכן.
כלומר, אם באים לקודד את מה שיש ב- Look ahead buffer ויש שם מחרוזת חוזרת שמופיעה ב-Window אז פשוט נצביע אליה - נרשום כמה תווים ללכת אחורה וכמה תווים להעתיק.
גם המפענח באותו שלב יודע מה ה-Window והוא יוכל להשתמש בו.

שלשת המצביע ב-LZ77

בשיטה הזו, כל מצביע Pointer יהיה מיוצג כשלשה:

- ההיסט Offset - שהוא יגיד כמה תווים נצטרך ללכת אחורה כדי להגיע למחרוזת הקודמת שהיא כמו המחרוזת הנוכחית.
- האורך Length - האורך של המחרוזת המועתקת.
- התו Symbol - תו נוסף שמוסיפים אחרי אותה מחרוזת חוזרת.

אלגוריתם קידוד של LZ77

LZ77 Encoding Algorithm - קידוד

```
1:  p ⇐ 1 // the next char to be coded
2:  while there is text remaining to be coded do:
3:        search for the longest match for S[p..] in S[p-W…p-1]
          suppose that the match occurs at position m with length l
4:        output the triple(p-m, l , S[p+l])
5:        p ⇐ p+l+1
```

1. מתחילים מהמקום הראשון בקובץ $p = 1$.
2. כל עוד יש טקסט שעומדים לדחוס:

3. תחפש את המחרוזת הארוכה ביותר שפותחת במקום הנוכחי כלומר מ-$S[p...]$ ממקום p והולך... בתוך החלון $S[p-W...\ p-1]$.
4. תוציא לפלט שלשה (ההיסט, האורך והתו).
5. נתקדם לתו הבא בתור.

אלגוריתם פענוח של LZ77

LZ77 Decoding Algorithm - פענוח

```
1:  p ⇐ 1 // the next char to be decoded
2:  for each triple (f, l , c) in the input do:
3:        S[p…p+l-1] ⇐ S[p-f … p-f+l-1]
4:        S[p+l] ⇐ c
5:        p ⇐ p+l+1
```

1. גם כאן מתחילים מהמקום הראשון לכן $p = 1$.
2. עבור כל שלשה (f, l, c) בקלט:
3. פשוט נמלא את המחרוזת שלנו, נמלא את המקום מאיפה שהיה המופע הקודם, שהוא נמצא ב : $S[p-f\ ...\ p-f+l-1]$.
4. התו הבא בתור הוא c.
5. מקדמים את המצביע p.

הצבעה עצמית

בהצבעה עצמית מצביעים על מחרוזת שטרם יש לנו אותה.

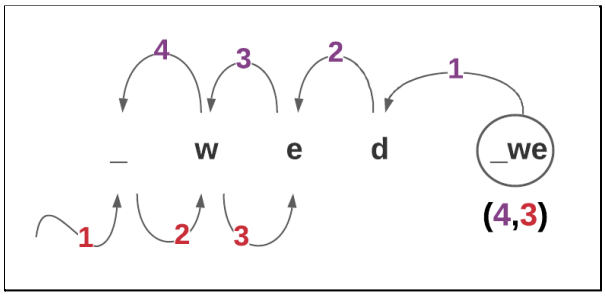
אנחנו נזהה הצבעה עצמית אם $length > offset$.

זה בעצם אומר שאנחנו מעתיקים מהצבעה שאנחנו טרם יודעים מה היא.

חסרונות עם LZ77

- משתמשים ב-Sliding Window , כלומר ככל שאנחנו מתקדמים וברגע שהגענו לחלון המקסימלי אנחנו למעשה מזרימים את החלון איתנו.
- השלשה "עולה" הרבה ביטים, אם היה לנו תו אחד שבמקור תפס 8 ביטים (ASCII) עכשיו הוא יתפוס גם את $length + offset$ + תו נוסף.
- לא נאמר איך מוצאים את המחרוזת הארוכה ביותר.

דחיסת LZSS



זה גרסה משופרת של LZ77.

- חיפוש המחרוזת הארוכה ביותר יהיה באמצעות עץ בינארי מאוזן.
- במקום להשתמש בכל פעם בשלשה, נשתמש לפעמים בתווים בודדים, ולפעמים במצביע שהמצביע יהיה זוג סדור של $(offset,\ length)$. כלומר כמה נלך אחורה, וכמה תווים להעתיק.
- נבדיל בין Char לבין זוג סדור על ידי ביט מוביל 0 או 1.

כאשר נקודד את האלמנטים האלה, נוכל לקודד את כל הסיביות המובילות בהתחלה וזה יתן לנו מעין הדרכה איך לקודד את כל הקובץ.

- נשים לב שגודל כל תו יהיה תמיד 8 סיביות -ASCII אבל קידוד של $(offset,\ length)$ יהיה שונה, כי זה יהיה תלוי כמה סיביות נגדיר ל-length וכמה ל-offset וזה יהיה תלוי בגודל החלון.
- לכן, משום שגודל הקידוד של זוג סדור זה משהו משתנה, אזי לפעמים ישתלם לנו להעתיק 2 תווים כזוג סדור או יכול גם לא להשתלם, כי זוג סדור "יעלה" יותר מ- $8 + 8 = 16$ סיביות ויהיה עדיף פשוט לרשום את 2 התווים וכן לגבי 3 תווים ויותר...
- לגבי תו אחד, סביר להניח שתמיד יהיה עדיף פשוט לרשום אותו מאשר לעשות עבורו זוג סדור.

איך נבדיל בין Char לבין זוג סדור?

אנחנו נבדיל ביניהם בעזרת ביט מוביל של 0 או 1 כלומר:

- $(0,\ char)$ - במקרה כזה, גודל התו תמיד יהיה 8 סיביות (ASCII) יחד עם ביט נוסף שיסמן זאת. כלומר סה"כ יהיה לנו במקרה זה $9\ bits$.
- $(1,\ (offset,\ length))$ - קידוד כזה יהיה תלוי במספר הסיביות שהגדרנו עבור $len,\ off$ וראינו שפרמטרים אלה תלויים בגודל החלון שלנו.

<u>לדוגמה</u>:

נתון חלון של $2^{12} = 4096$ וגם מחרוזת בגודל $2^4 = 16$.

לכן, אנחנו צריכים 17 ביטים כדי לקודד במקרה זה.

זאת כיוון ש: $17 = 12 + 4 + 1$ כאשר 4 זה len, ה-12 זה off ו-1 זה הסימן $sign$.

<u>לדוגמה</u>:

אם נתון לנו ש-$offset$ עם 3 סיביות אז טווח ההיסט שלו יכול להיות לכל היותר $2^3 = 8$.

כלומר, אנחנו יכולים "לקפוץ אחורה" לכל היותר 8 צעדים.

דוגמה לשימוש בחיפוש

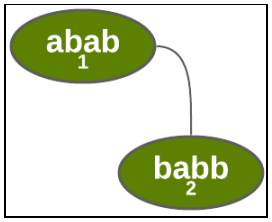
קודדו את המחרוזת הבאה בעזרת עץ חיפוש מאוזן כדי למצוא את ההתאמות הקודמות.
ננית שקיים חלון בגודל 5.

ababbababaa

כיוון שקיים חלון בגודל 5 אז $offset \leq 5$ וננית ש: $length \leq 4$.
נבנה את עץ חיפוש בינארי מאוזן לפי סדר לקסיקוגרפי.
נעבור על המחרוזת משמאל לימין, כאשר תחילה החלון ריק:

- התו הראשון הוא a, נדחוס אותו לתו בודד ונכניס אותו לחלון כלומר $(0, a)$.
כעת, כיוון שגודל המחרוזת שלנו הוא לכל היותר 4 אז נכניס לעץ חיפוש בינארי מאוזן את המחרוזת באורך 4 החל מהמיקום הנוכחי, כלומר נבחר את **abab**ababaa.
כלומר, אנחנו מכינים אותנו לחיפושים אחורה לכן ניקח את גודל המחרוזת הארוכה ביותר שאפשר להעתיק. ניצור עץ חדש כשהשורש זה המחרוזת הנוכחית והמספר מייצג את מספר התו שיצר את המחרוזת הזאת (נקרא לכל מספר בעץ "הכתובת").

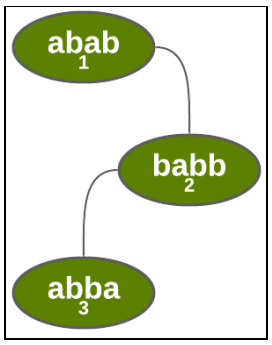
- נדחוס את התו הבא b אבל בחלון שלנו יש רק את a לכן נסתכל על b בהשוואה לשורש הזה ונראה שאין לנו התאמה ולכן נכתוב את b בפני עצמו. הכוונה היא שנכתוב $(0, b)$.
נכניס לעץ את המחרוזת באורך 4 החל מהמקום הנוכחי כלומר את a**babb**ababaa.
כיוון שהמחרוזת הנוכחית יותר גדולה מבחינה לקסיקוגרפית אז נכניס אותו מימין לשורש.



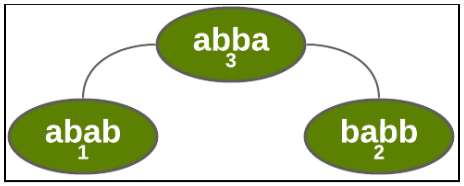
- נדחוס את התו הבא a (התו השלישי), וניקח את המחרוזת ab**abba**baba.
המחרוזת הנוכחית גדולה יותר מהשורש ולכן אנחנו הולכים ימינה, אפשר לראות שאין מחרוזת המשותפת עם abba ולכן מצאנו את ההתאמה הארוכה ביותר אז נרצה לחשב את ה-$offset$.
מצאנו את ההתאמה הארוכה ביותר בכתובת (1) ואנחנו נמצאים בכתובת (3), לכן נחשב בצורה הבאה:

$$offset = 3 - 1 = 2$$

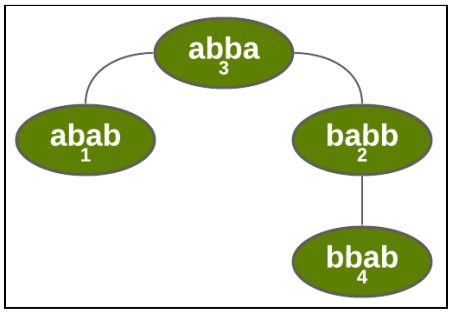
נכניס את המחרוזת הנוכחית לעץ המאוזן בהתאם לסדר הלקסיקוגרפי.



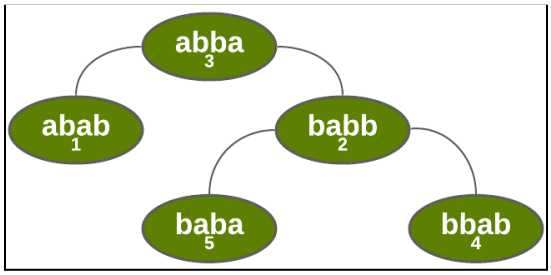
כעת נשים לב שהעץ הזה לא מאוזן, לכן נשתמש ב-Rotation (כמו שנלמד בקורסי קדם)



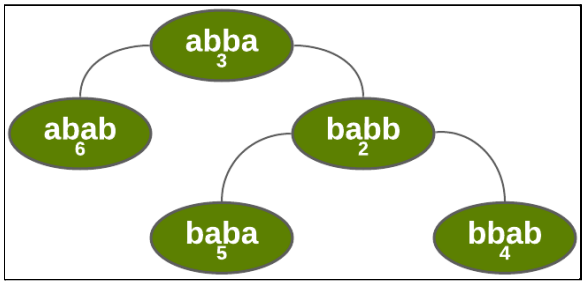
- הגענו לתו 4, ניקח את המחרוזת ababbababaa ונראה שאין לו התאמות, נכניס אותו לעץ.



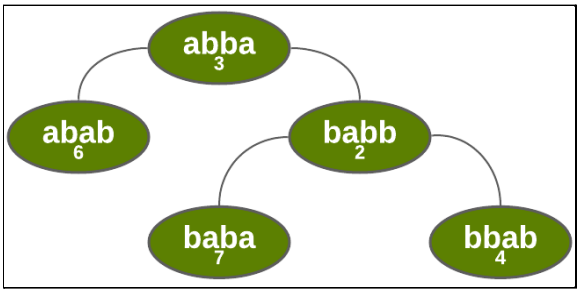
- עבור תו 5, ניקח את המחרוזת ababbabaa ונשווה אותו עם השורש ונראה שאין התאמה.
נעבור לבן הימני (כיוון שהמחרוזת הנוכחית גדולה מהשורש).
נשווה אותו עם מחרוזת בכתובת (2) ונראה שיש לנו שלוש תווים משותפים ביניהם.
לכן נרצה לחשב את ה-$offset$ אז נחשב $offset = 5 - 2 = 3$ והוא ייצג את הזוג הסדור $(3, 3)$.
נכניס את המחרוזת לעץ:



- עבור התו ה-6 נשים לב שהוא ממש כמו בכתובת (1) כלומר המחרוזת ababb**abab**aa.
  לכן, נכניס אותו לעץ, אין צורך בבדיקה של התאמה כי היא נבדקה כבר לפני.
  מאחר ויש 2 מחרוזות זהות אז נדרוס את הכתובת הקודמת (1) ונציב (6).



- עבור התו ה-7 כלומר המחרוזת ababba**baba** שוב, הגענו למחרוזת שכבר קיימת ולכן נדרוס את הכתובת (5) ונציב במקומו את (7).



- וכך הלאה...

איך טוענים את המופע הקודם?

ידוע לנו כי Hashing מבצע פעולות יעילות ביותר בגודל $O(1)$.
נשמור את האינדקסים (כמו בעץ בינארי מאוזן) בטבלת Hash כדי לבדוק האם זה בגבולות החלון או לא.
הרעיון הוא לעשות פונקציה שמפזרת באופן אחיד על פני הטבלה כך בממוצע זה יקח לנו בזמן ריצה קבוע.
עד עכשיו אמרנו שנמצא את המחרוזת הארוכה ביותר על ידי עץ בינארי מאוזן, אבל נשים לב שלא חייב לקחת את המחרוזת הארוכה ביותר מבחינה מעשית, אולי לפעמים זה יפגע בדחיסה עצמה, אבל הרבה פעמים זה יקח מהר יותר. אנחנו נשתמש בפונקצית Hash.

- אם נמצאים במקום הנוכחי ונניח שכדאי להעתיק רק מחרוזת מאורך 3 ומעלה אז מה שנעשה זה Hashing על שלושת התווים הראשונים שרואים וה-Hashing הזה יוביל אותנו לאיזה מופע מסוים של שלושת התווים הללו שהופיעו קודם בחלון.
- יכול להיות שיש התאמה ארוכה יותר ולא תמיד נפגע דווקא בהתאמה הזו אבל מבחינה מעשית, ה-Hash הזה נותן דחיסה דיי טובה ויעילה יותר מבחינת זמנים.
- אנחנו רק צריכים לבדוק שהפלט של הפונקציה הוא בטווח של גודל החלון שלנו, כלומר האם ניתן לגשת אליו או שהחלון כבר עבר אותו.

דחיסת LZS

אלגוריתם LZS שמבוסס על LZ77.
נתונים $Char,\ Offset,\ Length$.
בשונה מ-LZ77, אנחנו נרצה להגדיל את החלון, כלומר להגדיל את הטווח שה-$Offset$ יכול להגיע אליו.

הגדרת הפונקציה λ

- עבור תו בודד, נשתמש בסיבית מובילה 0 וגם 8 סיביות בשביל (ASCII).
- עבור זוג סדור, כלומר $(Offset,\ Length)$, נשתמש ב-2 סיביות מובילות בצורה פרפיקסית ונחלק את המקרה ל-2 סוגים שונים של חלונות:
  1. *חלון קרוב* - חלון שמוגדר עד 128 תווים, לכן נצטרך רק 7 סיביות כדי לייצג אותו.
  2. *חלון רחוק* - חלון שמוגדר עד 2048 תווים, לכן נצטרך 11 סיביות כדי לייצג אותו.

ההגיון הוא שלא נצטרך תמיד להשתמש בחלון רחוק אלא לעיתים יהיה אפשרות להשתמש בחלון קרוב וכך נוכל לחסוך בהרבה ביטים.

הגדרת הפונקציה λ בשיטת DoubleSpace

ההגדרה דומה לפונקציה הקודמת רק שבשיטה זאת יש לנו 3 סוגים שונים של חלונות:

1. *חלון ממש קרוב* - חלון שמוגדר עד 64 תווים, לכן נצטרך רק 6 סיביות. (2 סיביות מובילות).
2. *חלון קרוב* - חלון שמוגדר עד 320 תווים, לכן נצטרך 8 סיביות. (3 סיביות מובילות).
3. *חלון רחוק* - חלון שמוגדר עד 2368 תווים, לכן נצטרך 11 סיביות. (3 סיביות מובילות).

מה שטוב פה זה שהם עשו $Offset$ קצרים ואז עם 8 סיביות אפשר יהיה להתעסק באופן טוב יותר.

דחיסת LZ78

ב-LZ77 יש לנו את ה-Fixed Window שזה חלון באורך קבוע.
היינו צריכים להגיד שצריך ללכת x צעדים אחורה כדי לקבל את האורך של המחרוזת.
מצד שני, ב-LZ78 אנחנו נראה מילון מפורש.
כלומר, יהיה לנו אינדקס שהוא בכניסה למילון וכך נדע מיד באיזה מחרוזת מדובר.

מבחינה טכנית

בונים מילון D ונתון לנו טקסט T שאותו נרצה לדחוס.
עבור הטקסט T נחלק אותו לתתי מחרוזות $T_1, T_2, \dots, T_Z$ כאשר כל תת-מחרוזת נמצאת במילון $T_i \in D$.
למעשה, אנחנו נעשה חלוקה לטקסט, כלומר נקבל סדרה שנקראת ה-$Parsing$ של T עבור המילון D.

הגדרה פורמלית

נבנה מילון D שנכניס אליו מחרוזות: $D = \{Z_0, Z_1, \dots\}$.
כל מחרוזת נוספת שנכניס תקבל את האינדקס הבא בתור שפנוי.
המילון ההתחלתי יהיה המילון שמכיל רק את המילה הריקה כלומר $Z_0 = \varepsilon$.
נחלק את הטקסט לתתי-מחרוזות (נעשה $Parsing$):
נניח ואנחנו נמצאים באמצע תהליך כלשהו של חלוקה: $T_1, \dots, T_{j-1}$ עבור $T[0 \dots i)$
וכעת נרצה ליצור את תת-המחרוזת T_j שמתחיל באינדקס i.
נמצא את המחרוזת הארוכה ביותר שמופיעה במילון ונסמן אותה ב-$Z_k \in D$ שפותחת במקום הנוכחי שלנו בו אנחנו נמצאים כלומר $T[i \dots n-1)$.
תת המחרוזת הבאה בתור T_j תהיה:

$$T_j = Z_j = T\left[i \dots \left(i + \left|Z_k\right|\right)\right] = Z_k\, t_{\left(i+\left|Z_k\right|\right)}$$

וכך הוא נכנס למילון.

תרגיל דוגמה לחלוקה של תתי-מחרוזות לפי אלגוריתם LZ78

נשתמש בהגדרה:

$$T_j = Z_j = T\left[i \ldots \left(i + \left|Z_k\right|\right)\right] = Z_k\, t_{\left(i+\left|Z_k\right|\right)}$$

נתון הטקסט הבא:

$$T = badadadabaab$$

ונתבונן בטבלה הבאה:

אינדקס i - (T_i)	תת המחרוזת	קידוד	מספר סיביות	בינארי
0	ε			
1	b	$(0,\ b)$	$0 + 2$	01
2	a	$(0,\ a)$	$1 + 2$	0 00
3	d	$(0,\ d)$	$2 + 2$	00 10
4	ad	$(2,\ d)$	$2 + 2$	10 10
5	ada	$(4,\ a)$	$3 + 2$	100 00
6	ba	$(1,\ a)$	$3 + 2$	001 00
7	ab	$(2,\ b)$	$3 + 2$	010 01

נרצה לחפש את המחרוזת הארוכה ביותר במופיעה במילון ותואמת לרישא של המיקום הנוכחי.

- תחילה אין לנו שום רישא שחופפת כי ε ולכן נוסיף למילון את Z_i כלומר:

$$Z_0 = T[0 \ldots (0 + |\varepsilon|)] = T[0 \ldots (0 + 0)] = T[0 \ldots 0] = \varepsilon$$

- כעת נכניס את b כלומר $i = 1$.
- כעת אנחנו ב-$i = 2$ ונסתכל על a.

נחפש את המחרוזת הארוכה ביותר שהופיעה קודם במילון אך יש לנו רק את $\varepsilon,\ b$.

לכן, המחרוזת הארוכה ביותר היא ε ומכאן:

$$T_2 = Z_2 = T[2 \ldots (2 + |\varepsilon|)] = T[2 \ldots (2 + 0)] = \varepsilon \cdot a$$

- עבור d נשאל מה המחרוזת הארוכה ביותר שהופיעה קודם במילון וזה עדיין ε ולכן

$$T_3 = Z_3 = T[3 \ldots (3 + |\varepsilon|)] = T[3 \ldots (3 + 0)] = \varepsilon \cdot d$$

כלומר נוסיף את $\varepsilon \cdot d$ למילון.

- כעת נשאל מהי המחרוזת הארוכה ביותר שפותחת במקום הנוכחי ad שהופיעה קודם במילון.

כלומר כל מחרוזת שנוסיף כעת למילון מורכבת ממחרוזת אחרת במילון + תו חדש.

למשל ב $i = 4$ יש לנו את ad ולפי הקידוד אפשר לראות $(2, d)$ כלומר באינדקס 2 יש לנו את התו a ונוסיף אליה את התו d.

דחיסת LZW

זה בעצם שיפור של LZ78.
משתמשים במבנה נתונים Trie (או עץ Prefix).

- המילון מאותחל עם תווים בודדים.
- כל תת-מחרוזת שבמילון נכניס לתוך ה-Trie ובכל פעם נטייל על ה-Trie וברגע שנגיע לעלה זה אומר שהגענו למחרוזת הארוכה ביותר שתואמת את הרישא הנוכחית ואנחנו נרחיב אותו עם התו הבא בתור.

אלגוריתם קידוד LZW

LZW Encoding Algorithm - קידוד

```
1:  Dictionary ⇐ single characters
2:  w ⇐ first char of input
3:  repeat do:
4:        k ⇐ next char
5:        if (EOF) then:
6:              output code(w)
7:        else if (w·k) ∈ Dictionary then:
8:              w ⇐ w·k
9:        else: // if w·k is not in the dictionary then add it.
10:             output code(w)
11:             Dictionary ⇐ w·k
12:             w ⇐ k
13:       end-if-else
14: end-repeat
```

יהיו $w,\ k$ וכל פעם נחפש האם $w \cdot k$ נמצא במילון?
כל עוד הוא נמצא במילון - נרחיב את זה לתו הבא במילון.
אם הוא לא נמצא במילון אז נעדכן את המילון ונעבור להמשך הקובץ.

- נשים לב שמי שיוצר את המילון זה גם המקודד וגם המפענח, כל אחד בצד שלו.
- מספר הסיביות שהקובץ מכיל הוא $|Number\ of\ insertions\ to\ D| \cdot \log_2(|D|)$

דוגמה לקידוד

1. נתון הטקסט הבא:

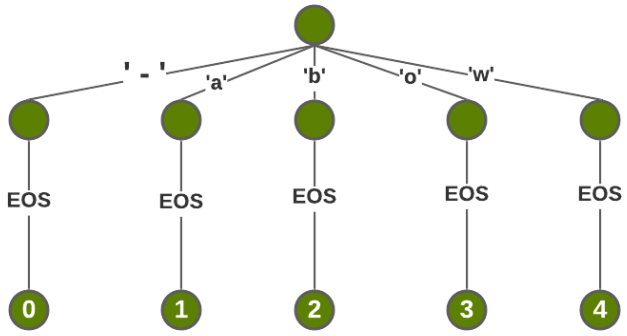
$$T = wabba\_wabba\_wabba\_wabba\_woo\_woo$$

2. נאתחל את המילון D שלנו כאשר ה-Code הוא האינדקס שנשלח לפלט.
גודל המילון $|D|$ תלוי בגודל $|\Sigma| = 5$, הוא בעצם יהיה החזקה הקרובה ביותר של 2 אבל גדולה ממש מגודל הא"ב. כלומר במקרה שלנו: $2^2 < |\Sigma| < 2^3 = |D| = 8$.
אנחנו עושים את זה כדי לשמור מקום "ספייר" לכניסות הבאות.

Code	Phrase
0	-
1	a
2	b
3	o
4	w
5	
6	
7	

- ברגע שהמילון מתמלא נכפיל את המילון כפול 2 ואז זה אומר שנוסיף עוד סיבית.
  כלומר ממילון בגודל 8 (3 סיביות) למילון בגודל 16 (4 סיביות).

3. נבנה עץ $TRIE$ התחלתי:



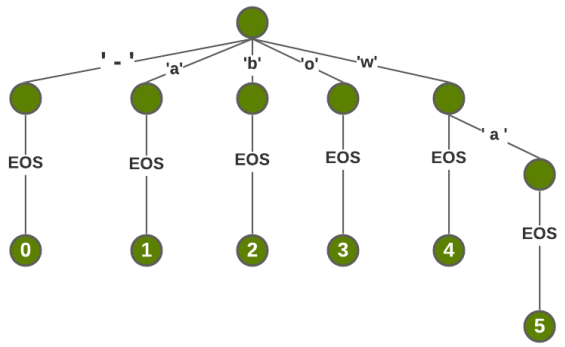
4. נתחיל לדחוס את הטקסט: $T = wabba\_wabba\_wabba\_wabba\_woo\_woo$
נסתכל באלגוריתם עצמו עבור הפרמטרים הרלוונטיים.
בהתחלה w מקבל את האות הראשונה ו-k מקבל את התו הבא.
כלומר $w = 'w'$ וגם $k = 'a'$ ואנחנו שואלים האם $w \cdot k$ נמצא במילון (כלומר 'wa').
אנחנו מסתכלים על העץ שיצרנו ושואלים האם יש לנו יציאה מקודקוד 'w' לקודקוד 'a' והתשובה היא לא.
לכן, לפי האורך של המחרוזת אפשר להגיד שהיא לא נמצאת במילון ומכאן הגענו לפלט של הקוד w.
הקוד של 'w' הוא 4 וצריך להעשיר את המילון שלנו בכניסה נוספת, לכן:

a) נוסיף לטבלה שלנו כניסה חדשה של 'wa' .

b) נוסיף ענף חדש לעץ בהתאם.

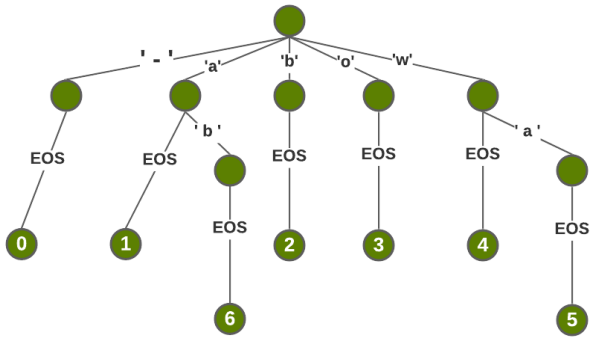
c) הפלט של הקוד של w הוא 4 ולכן הפלט הראשוני יהיה **100**.

Code	Phrase
0	-
1	a
2	b
3	o
4	w
5	wa
6	
7	



5. ניקח את התו הבא $w = 'a'$ ו- $k = 'b'$.
נשאל האם 'ab' נמצא במילון? - נבדוק את זה בעזרת העץ - לא נמצא.
לכן נוסיף אותו לטבלה, לעץ והפלט יהיה הקוד של w שהוא 1 ולכן הפלט יהיה **100 001**.

Code	Phrase
0	-
1	a
2	b
3	o
4	w
5	wa
6	ab
7	

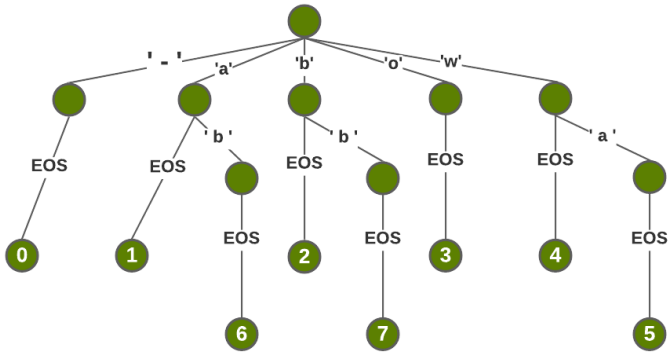


6. ניקח את הבא $w = 'b'$ ו- $k = 'b'$.

נשאל האם $'bb'$ נמצא במילון? - נבדוק את זה בעזרת העץ - לא נמצא.

לכן נוסיף אותו לטבלה, לעץ והפלט יהיה הקוד של w שהוא 2 ולכן הפלט יהיה **010 001 100**.

Code	Phrase
0	-
1	a
2	b
3	o
4	w
5	wa
6	ab
7	bb



7. ניקח את הבא $w = 'b'$ ו- $k = 'a'$.

נשאל האם $'ba'$ נמצא במילון? - נבדוק את זה בעזרת העץ - לא נמצא.

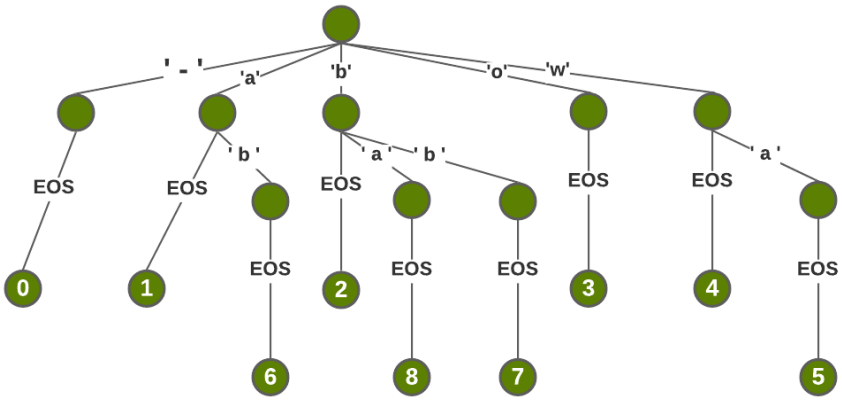
הפלט יהיה הקוד של w שהוא 2 ולכן הפלט יהיה **010 010 001 100**.

לכן נוסיף אותו לטבלה, אך נשים לב שהיא כבר מלאה ולכן נגדיל את המילון פי 2 כלומר לגודל 16.

אך זה אומר שאנחנו עכשיו מדברים על 4 סיביות במקום 3 סיביות.

לכן, מעכשיו כל הוספה שלנו לפלט יהיה באורך 4 סיביות (לא כולל השלב הנוכחי שעדיין 3 סיביות).

Code	Phrase
0	-
1	a
2	b
3	o
4	w
5	wa
6	ab
7	bb
8	ba
…	
…	
15	



8. נמשיך ככה הלאה עם שאר התווים בטקסט..

אלגוריתם פענוח LZW

בפענוח במקום להשתמש ב- w, k כמו שהשתמשנו בקידוד, הפעם נשתמש ב- $OLD,\ NEW$ אבל זה בעצם אותם תפקידים.

אנחנו נתמקד יותר בשלב השני שבו ה-Encoder משתמש בכניסה שזה עתה הוא הכניס למילון.

וכאשר ה-Decoder מגיע לשלב הזה הוא עדיין לא רואה את הכניסה הזו בטבלה שלו, אבל הוא יכול לשחזור אליה בדיוק כי הוא יודע שזה מה שהוכנס על ידי ה-Encoder.

LZW Decoding Algorithm

```
 1:  Initialize table with single character strings
 2:  OLD = first input code
 3:  output translation of OLD
 4:  while not end of input stream do:
 5:        NEW = next input code
 6:        if NEW is not in the string table then:
 7:              S = translation of OLD
 8:              S = S · C
 9:        else:
10:              S = translation of NEW
11:        output S
12:        C = first character of S
13:        Translation(OLD) · C to the string table
14:        OLD = NEW
15:  end-while
```

דוגמה לפענוח

הפענוח הולך בכיוון ההפוך אבל כדי לא לבלבל במקום לקרוא להם w, k נקרא להם $OLD,\ NEW$.

נתון לנו $0, 1, 3, 2, 4, 7, 0, 9, 10, 0$. אנחנו צריכים גם את S מחרוזת ו-C זה התו הראשון של S.

וגם נתון לנו המילון ההתחלתי:

Code	Symbol
0	a
1	b
2	c

נפעל לפי האלגוריתם:

Old	New	S	C	Output
0	1	b	b	a, b
1	3	ab	a	ab
3	2	c	c	c
2	4	ba	b	ba
4	7	bab	b	bab

והמילון:

Code	Symbol
0	a
1	b
2	c
3	ab
4	ba
5	abc
6	cb
7	bab

LZ78/LZW VS LZ77/LZSS

ה-LZ77 נוטה לעבוד טוב יותר מאשר LZ78/LZW מהסיבות הבאות:

1. ל-LZW יש כניסות במילון שאנחנו לא משתמשים בהם (לא כל תת-מחרוזת נכנסה)
2. ל-LZW יש תתי-מחרוזות שלא הכנסנו למילון ואולי דווקא אותם נצטרך.
3. ל-LZ77 יש מילון מרומז (ה-Window) ושם אנחנו מכניסים כל תת-מחרוזת שראינו בהיסטוריה.

אלגוריתמי RLE, BWT, MTF

אלגוריתם קודי Run-Length - RLE

- המושג Run אומר שאם יש לנו תו שהוא זהה (נניח 'aaa' אז נגיד שזה Run של a עם 3 פעמים).
- הרעיון באלגוריתם זה הוא שאנחנו מקודדים רצפים שכאלה.
- השתמשו בשיטה הזאת כדי להעביר פקסים.
- למשל עבור הטקסט הבא:

$$a\ c\ c\ c\ b\ b\ a\ a\ a\ b\ b$$

אנחנו נקודד בצורה הבאה:

$$(a,1)(c,3)(b,2)(a,3)(b,2)$$

קודי Binary Run-Length

- מה קורה במקרים בינארים?
- למשל עבור המחרוזת הבאה:

$$0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1$$

בגלל שמדובר כאן על א"ב בינארי אז לא צריך להבין כמה פעמים כל תו מופיע כי יש רק 2 שכאלה.
אז כל מספר בקידוד מפריד ומגדיר את רצף האחדות או אפסים שיש לנו.
לכן נקבל:

$$1\ 4\ 2\ 3\ 5\ 1\ 1\ 1\ 1\ 2$$

כלומר, יש 0 פעם אחת ואח"כ יש 4 פעמים אחדות ואז פעמיים אפסים וכו'..

- השאלה האמיתית פה היא איך אנחנו מקודדים את אותו Length?
  נניח שמשתמשים ב-$k > 0$ סיביות כדי לקודד את אותו Length.
  כלומר האורכים שלנו הם בין 0 לבין $2^k - 1$.

- מה יקרה אם יש לנו אורך יותר גדול ממה שאפשר?
  אנחנו נטפל באורכים חריגים באופן מיוחד - נשתמש בשיטה שנקראת **Escape Code**:
  הוא בעצם אומר לנו לא להחליף צד (בין 0 ל-1)
  אם ה-Run הוא בדיוק $2^k - 1$ אז נשרשר אחריו '0'.

- **דוגמה:**

נניח $k = 3$ (כלומר אפשר לאורכים עד 7, כי אמרנו שברגע ש: $2^k - 1$ אז מציבים '0').

- יש לנו רצף של **שמונה** אפסים. איך נקודד ביטוי שכזה אם $k = 3$?

נסתכל על המקרה הכי גדול $2^3 - 1 = 7$ כלומר 111 ואז נחשב $8 - 7 = 1$ כלומר 001.
לכן סה"כ נקודד 111 001.

- יש לנו רצף של **שבעה** אפסים. איך נקודד ביטוי שכזה אם $k = 3$?

נסתכל על המקרה הכי גדול $2^3 - 1 = 7$ כלומר 111 ואז נחשב $7 - 7 = 0$ כלומר 000.
לכן סה"כ נקודד 111 000.

- יש לנו רצף של **14** אפסים. איך נקודד ביטוי שכזה אם $k = 3$?

נסתכל על המקרה הכי גדול $2^3 - 1 = 7$ כלומר 111 ואז נחשב $14 - 7 = 7$
כלומר 111 ואז $7 - 7 = 0$ כלומר 000 לכן סה"כ נקודד 000 111 111.

באופן כללי נקבל: $m \cdot \left(2^k - 1\right)$.

<u>דוגמה שיש מקום ל-Escape Code</u>:

נתון לנו בלוק מקסימלי $k = 3$, כל בלוק יכול להחזיק עד: $2^k - 1 = 8 - 1 = 7$ רצפים.
קודדו את המחרוזת הבאה: 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1

⇩

1, 4, 2, 3, 8, 1

נשים לב שאנחנו בבעיה כי גודל כל בלוק הוא לכל היותר 3 סיביות (עד המספר 7) אבל יש לנו כאן 8.
לכן, עבור נסתכל על המקרה הכי גדול $2^3 - 1 = 7$ כלומר 111 ואז נחשב $8 - 7 = 1$ כלומר 001.
לכן סה"כ נקודד 111 001. ⇩

001 100 010 011 111 001 001

<u>דוגמה שהמקום ל-Escape Code כבר תפוס עם הספרה 7</u>:

נתון לנו בלוק מקסימלי $k = 3$, כל בלוק יכול להחזיק עד: $2^k - 1 = 8 - 1 = 7$ רצפים.
קודדו את המחרוזת הבאה: 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0

⇩

1, 4, 2, 3, 7, 8, 1

נשים לב שאנחנו בבעיה כי גודל כל בלוק הוא לכל היותר 3 סיביות (עד המספר 7) אבל יש לנו כאן 8.
בנוסף, המספר 7 כאן ותופס את המקום של ה-Escape code, כלומר האלגוריתם יחשוב שאנחנו קולטים מספר גדול מהבלוק כי עשינו 111 אבל בעצם בסה"כ רצינו לקודד את המספר 7.
לכן, עבור הספרה 8 נקודד 111 001 כלומר ה-escape code וגם $8 - 7 = 1 \Rightarrow 001$
ועבור הספרה 7 נקודד 111 000 כלומר "נקודד 7 ככה 111" ובשביל להגיד שזה לא אקסייפ אז $7 - 7 = 0$

⇩

001 100 010 011 111000 111001 001

Binary Run-Length Encoding Algorithm

```
1:  MaxCount ⇐ 2^k − 1 ; count ⇐ 0 ;  PrevBit ⇐ "0";
2:  while input remains do:
3:        CurrentBit ⇐ next input bit
4:        if PrevBit == CurrentBit and count < MaxCount then:
5:              count++
6:        else then:
7:              output count using k bits
8:              if count == MaxCount and PrevBit ≠ CurrentBit then:
9:                    output 0 using k bits
10:             count ⇐ 1
11:       PrevBit ⇐ CurrentBit
12: output count using k bits
```

Binary Run-Length Encoding Algorithm - **הסבר שורות קוד**

1:	ה-MaxCount סופר כמה מופעים יש לנו (הוא החסם שלנו). ה-count סופר רצף של אותם סיביות (אם אפס סימן ששינינו סוג של סיבית). ה-PrevBit שומר את הביט הקודם שעמדנו בו.
2:	כל עוד יש לנו עוד ביטים לקודד אז:
3:	הביט הנוכחי מקבל את הביט הבא
4:	אם זה אותה הסיבית כמו הקודם וגם עוד לא הגענו לחסם שלנו אז:
5:	נקדם את count + 1
6:	אחרת (אם הגענו לאורך מקסימלי או שנפגשנו בסיבית מסוג אחר מהקודם) אז:
7:	תחזיר את ה-count בצורה של k סיביות
8:	אם הגענו למצב של אורך חריג והביט הקודם שונה מהקיים :
9:	נחזיר אפסים בצורה של k סיביות
10:	נתחיל ספירה מההתחלה
11:	מעדכנים את הביט הקודם להיות הביט הנוכחי
12:	בסוף התהליך נוציא את ה-count ב-k ביטים

Binary Run-Length Decoding Algorithm

```
1:  MaxCount ⇐ 2^k − 1
2:  parity ⇐ 0
3:  while input remains do:
4:        read k bits from the input stream to get the integer x.
5:        if parity == 0 then:
6:              output x times "0" bits
7:        else then:
8:              output x times "1" bits
9:        if x < MaxCount then:
10:             parity := 1 - parity
11: end-while
```

Binary Run-Length Decoding Algorithm - **הסבר שורות קוד**

1:	ה-MaxCount זה חסם לאורך
2:	מתחילים מהסיבית 0
3:	כל עוד יש עוד בקלט שלנו אז
4:	קוראים יחידות של k ביטים
5:	אם הביט הוא מסוג 0 אז:
6:	אז נוציא x פעמים את "0"
7:	אחרת, אם הביט מסוג 1 אז:
8:	אז נוציא x פעמים את "1"
9:	אם x קטן מהאורך המקסימלי האפשרי אז:
10:	אז מחליפים תפקיד (כלומר מ-1 ל-0 או להפך)
11:	סוף לולאה וקוד

כלליות

אנחנו יכולים לעשות את השיטה שהצגנו (עם ה-k) באופן אדפטיבי או להשתמש בהאפמן כדי לקודד את האורכים שלנו.

אלגוריתם BWT

- זה בעצם טרנספורמציה.
- היא לא דוחסת אלא לוקחת את הטקסט, מערבבת אותו קצת ומעבירה אותו לטקסט שהוא דחיס יותר.
- היא לא תעשה כלום אם הטרנספורמציה הזאת לא הפיכה.
- אנחנו נרצה להמיר את הטקסט למשהו יותר דחיס, לדחוס אותו ולהמיר אותו בחזרה לצורה המקורית במקרה הצורך.

צעדי האלגוריתם

1. נבנה מטריצה עצומה nxn כאשר גודל הטקסט הוא $|T| = n$.
2. לוקחים את כל הטרנספורמציות הציקליות של הטקסט ושמים אותם בשורה.
3. ממיינים אותם בסדר לקסיקוגרפי
4. מה שנרצה בסוף התהליך זה:
   a. לקבל את העמודה האחרונה של המטריצה.
   b. אינדקס שאומר לנו באיזה שורה נמצא הטקסט המקורי בסדרה הממוינת.

דוגמה

נתון לנו הטקסט: $T = mississippi$.

ניקח את כל הטרנספורמציות שלה בצורה ציקלית ונמיין בסדר לקסיקוגרפי:

אנחנו נעביר בתור הטרנספורמציה את העמודה האחרונה (תו אחרון בכל שורה) ואת האינדקס של הטקסט המקורי כלומר שורה 4.

פורמלי:

$$BWT(T) = (pssmipissii,\ 4)$$

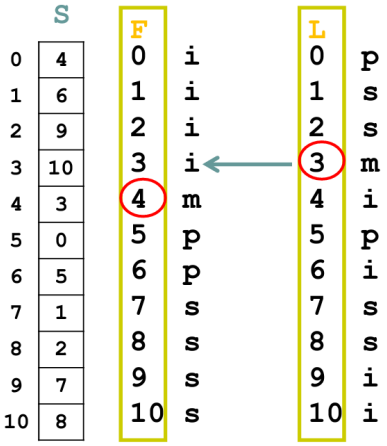
F	L
imississipp	
ippimississ	
issippimiss	
ississippim	
mississippi	
pimississip	
ppimississi	
sippimissis	
sissippimis	
ssippimissi	
ssissippimi	

- אנחנו רוצים להראות שבעזרת התו האחרון אפשר לחזור לטקסט המקורי.
  איך בעזרת L ו-F והאינדקס אפשר לשחזר את הטקסט המקורי?
  ניקח את העמודה הראשונה והאחרונה ונצמיד אותם ביחד: (מלמטה):
  ה-A' הוא פרמוטציה ציקלית של A ו-A' ממויין לפי העמודה השניה.

L	F	A'
p	i	mississip
s	i	ppimissis
s	i	ssippimis
m	i	ssissippi
i	m	ississipp
p	p	imississi
i	p	pimississ
s	s	ippimissi
s	s	issippimi
i	s	sippimiss
i	s	sissippim

למה

אם יש לו את אותו תו אז הוא מופיע באותו הסדר גם ב-L וגם ב-F.



איך זה עוזר לנו עם הפענוח?

כדי לעשות את ההופכיות של הטרנספורמציה נעשה את הדבר הבא:

נבנה מיפוי בין ה-F לבין ה-L.

כלומר $S: F \rightarrow L$.

BWT Reconstructing T Algorithm

```
1:  BWT(L, index) {
2:          F ⇐ sort(L)
3:          I ⇐ index
4:          for (i=0; i<n; i++) {
5:                  T[i] ⇐ F[I];
6:                  I ⇐ S[I]
7:          }
8:  }
```

BWT Reconstructing T Algorithm - הסבר קוד

```
1:
2:                                          F-ה מקבל את המיון של L
3:                                  I-ה מקבל את האינדקס אותו אנחנו רוצים לפענח
4:                                          אנחנו יודעים מה גודל הטקסט (n)
5:                                  T-ה מקבל את האות במיקום ה-I לפי המערך F.
6:          האינדקס I מקבל את האינדקס שמיצג את האות ב-L (לפי הדוגמה בציור למעלה)
7:
8:
```

תרגיל

נתון: $BWT(T) = (nnbaaa,\ 3)$ מצאו T.

	0	1	2	3	4	5
L =	n	n	b	a	a	a

ממוין לקסיקוגרפי:

	0	1	2	3	4	5
F =	a	a	a	b	n	n

ה-S זה המערך שמעביר אותנו מ-F ל-L.

	0	1	2	3	4	5
S =	3	4	5	2	0	1

כעת, כדי לפענח אנחנו צריכים את האינדקס ההתחלתי שזה 3 ונתחיל ממנו: $b\ a\ n\ a\ n\ a$

אלגוריתם קוד Move To Find - MTF

- כל תו נכתוב את מספר התווים השונים שהופיעו מאז המופע האחרון שלו.
- נשתמש ברשימה כדי לדחוס את זה.

דוגמה להבנת הרעיון

- נתונים לנו א"ב הבאים $\Sigma = \{d, e, h, l, o, r, w\}$.
- נתון הטקסט הבא: $T = helloworld$
- ניצור רשימה באופן ההתחלתי הבא: $MTF - list = \{d, e, h, l, o, r, w\}$.
- כל תו שהוא מקודד עובר לראש הרשימה.
- $MTF(T)$ יהיה שווה לסדרה מספרית שתייצג לנו את הקוד.

שלבים:

1. נתחיל עם האות הראשונה בטקסט h.
2. האות h מופיע באינדקס 2 ברשימה $list$ ולכן $MTF(T) = 2$.
3. נעביר את h לראש הרשימה ונקבל $MTF - list = \{h, d, e, l, o, r, w\}$.
4. האות השניה בטקסט e.

5. האות e מופיע באינדקס 2 ברשימה $list$ ולכן $MTF(T) = 2\ 2$.
6. נעביר את e לראש הרשימה ונקבל $MTF - list = \{e, h, d, l, o, r, w\}$.
7. האות השלישית בטקסט l.
8. האות l מופיע באינדקס 3 ברשימה $list$ ולכן $MTF(T) = 2\ 2\ 3$.
9. נעביר את l לראש הרשימה ונקבל $MTF - list = \{l, e, h, d, o, r, w\}$.
10. האות הרביעית בטקסט l.
11. האות l מופיע באינדקס 0 ברשימה $list$ ולכן $MTF(T) = 2\ 2\ 3\ 0$.
12. נעביר את l לראש הרשימה ונקבל $MTF - list = \{l, e, h, d, o, r, w\}$.
13. האות החמישית בטקסט l.
14. האות o מופיע באינדקס 4 ברשימה $list$ ולכן $MTF(T) = 2\ 2\ 3\ 0\ 4$.
15. נעביר את o לראש הרשימה ונקבל $MTF - list = \{o, l, e, h, d, r, w\}$.
16. וכך הלאה עד שנקבל ... $MTF(T) = 2\ 2\ 3\ 0\ 4\ 6\ 1$

הרשימה ההתחלתית יכולה להיות או א"ב לפי סדר לקסיקוגרפי או לפי סדר שמופיע בטקסט.

חיבור כל האלגוריתמים בפרק

1. ניקח את $BWT(T)$ ונפעיל על הטקסט.
2. על התוצאה נפעיל $MTF(T)$
3. אנחנו נקבל מלא מספרים קטנים שעליהם נפעיל את $Huffman$.

שאלת מטלה

נניח כי נתונה ההודעה T הבאה: $T = miss\_missouri$

א. חשב/י את $S = BWT(T)$

ב. חשב/י את $R = MTF(S)$ כאשר $\Sigma = \{\_, i, m, o, r, s, u\}$.

ג. הפעל/הפעילי את אלגוריתם הפמן על התוצאה.

<u>פתרון א'</u> - יוצרים מטריצה בגודל nxn ובמקרה שלנו בגודל $13x13$.

לוקחים את כל הטרנספורמציות הציקליות של הטקסט ושמים אותם בשורה:

m	i	s	s	_	m	i	s	s	o	u	r	i
i	s	s	_	m	i	s	s	o	u	r	i	m
s	s	_	m	i	s	s	o	u	r	i	m	i
s	_	m	i	s	s	o	u	r	i	m	i	s
_	m	i	s	s	o	u	r	i	m	i	s	s
m	i	s	s	o	u	r	i	m	i	s	s	_
i	s	s	o	u	r	i	m	i	s	s	_	m
s	s	o	u	r	i	m	i	s	s	_	m	i
s	o	u	r	i	m	i	s	s	_	m	i	s
o	u	r	i	m	i	s	s	_	m	i	s	s
u	r	i	m	i	s	s	_	m	i	s	s	o
r	i	m	i	s	s	_	m	i	s	s	o	u
i	m	i	s	s	_	m	i	s	s	o	u	r

<u>ממיינים אותם בסדר לקסיקוגרפי</u>:

F												L
_	m	i	s	s	o	u	r	i	m	i	s	s
i	m	i	s	s	_	m	i	s	s	o	u	r
i	s	s	_	m	i	s	s	o	u	r	i	m
i	s	s	o	u	r	i	m	i	s	s	_	m
m	i	s	s	_	m	i	s	s	o	u	r	i
m	i	s	s	o	u	r	i	m	i	s	s	_
o	u	r	i	m	i	s	s	_	m	i	s	s
r	i	m	i	s	s	_	m	i	s	s	o	u
s	_	m	i	s	s	o	u	r	i	m	i	s
s	o	u	r	i	m	i	s	s	_	m	i	s
s	s	_	m	i	s	s	o	u	r	i	m	i
s	s	o	u	r	i	m	i	s	s	_	m	i
u	r	i	m	i	s	s	_	m	i	s	s	o

מה שנרצה בסוף התהליך זה:

1. לקבל את העמודה האחרונה של המטריצה.
2. אינדקס שאומר לנו באיזה שורה נמצא הטקסט המקורי בסדרה הממוינת.

המטריצה-'A היא פרמוטציה ציקלית של A ו-'A ממוינת לפי העמודה השניה.

אנחנו ניקח את העמודה הראשונה F והאחרונה L ונרכיב בעזרתם את הטקסט חזרה.

הם שומרים על הסדר, כלומר ה-i הראשון ב-F זה יהיה ה-i הראשון ב-L וכן הלאה, כלומר יש התאמה בין העמודות.

<u>נציג את המיפוי</u>: נבנה מיפוי בין ה-F לבין ה-L. כלומר $S: F \to L$.

כלומר, נעבור כל כל תו ב-F לפי הסדר החל מ-0, לכל תו ב-F נשאל איפה התו הזה ב-L לפי הסדר שלו.

- ב-$F[1]$ יש את i בפעם הראשונה, אז ה-i הראשון נמצא ב-$L[4] = i$, לכן נציב ב-$S[1] = 4$
- ב-$F[2]$ יש את i בפעם השניה, אז ה-i השני נמצא ב-$L[10] = i$, לכן נציב ב-$S[1] = 10$.

	F
0	_
1	i
2	i
3	i
4	m
5	m
6	o
7	r
8	s
9	s
10	s
11	s
12	u

⇒

	L
0	s
1	r
2	m
3	m
4	i
5	_
6	s
7	u
8	s
9	s
10	i
11	i
12	o

⇒

	S
0	5
1	4
2	10
3	11
4	2
5	3
6	12
7	1
8	0
9	6
10	8
11	9
12	7

לכן, $S = BWT(T) = (srmmi\_sussiio,\ 4)$ אינדקס 4 כי ב-$F[4] = m$ שזה התו הראשון בהודעה שלנו: $miss\_missouri$.

<u>פתרון ב'</u>

חשב/י את $R = MTF(S)$ כאשר $\Sigma = \{\_,\ i,\ m,\ o,\ r,\ s,\ u\}$.

כלומר צריך לחשב עבור $S = srmmi\_sussiio$.

ניצור רשימה באופן ההתחלתי הבא:

0	1	2	3	4	5	6
_	i	m	o	r	s	u

כל תו שהוא מקודד עובר לראש הרשימה ו-$MTF(T)$ יהיה שווה לסדרה מספרית שתייצג לנו את הקוד.

נתחיל עם האות הראשונה בטקסט: s, האות הזאת מופיעה באינדקס **5**, נעביר את s לראש הרשימה ונקבל:

0	1	2	3	4	5	6
s	**_**	**i**	**m**	**o**	**r**	**u**

האות השניה בטקסט: r, האות הזאת מופיעה באינדקס **5**, נעביר את r לראש הרשימה ונקבל:

0	1	2	3	4	5	6
r	**s**	**_**	**i**	**m**	**o**	**u**

האות השלישית בטקסט: m, האות הזאת מופיעה באינדקס **4**, נעביר את m לראש הרשימה ונקבל:

0	1	2	3	4	5	6
m	**r**	**s**	**_**	**i**	**o**	**u**

האות הבאה בטקסט: m, האות הזאת מופיעה באינדקס **0**, נעביר את m לראש הרשימה ונקבל:

0	1	2	3	4	5	6
m	**r**	**s**	**_**	**i**	**o**	**u**

האות הבאה בטקסט: i, האות הזאת מופיעה באינדקס **4**, נעביר את i לראש הרשימה ונקבל:

0	1	2	3	4	5	6
i	**m**	**r**	**s**	**_**	**o**	**u**

האות הבאה בטקסט: _, האות הזאת מופיעה באינדקס **4**, נעביר את _ לראש הרשימה ונקבל:

0	1	2	3	4	5	6
_	**i**	**m**	**r**	**s**	**o**	**u**

האות הבאה בטקסט: s, האות הזאת מופיעה באינדקס **4**, נעביר את s לראש הרשימה ונקבל:

0	1	2	3	4	5	6
s	**_**	**i**	**m**	**r**	**o**	**u**

האות הבאה בטקסט: u, האות הזאת מופיעה באינדקס **6**, נעביר את u לראש הרשימה ונקבל:

0	1	2	3	4	5	6
u	**s**	**_**	**i**	**m**	**r**	**o**

האות הבאה בטקסט: s, האות הזאת מופיעה באינדקס **1**, נעביר את s לראש הרשימה ונקבל:

0	1	2	3	4	5	6
s	**u**	**_**	**i**	**m**	**r**	**o**

האות הבאה בטקסט: s, האות הזאת מופיעה באינדקס **0**, נעביר את s לראש הרשימה ונקבל:

0	1	2	3	4	5	6
s	**u**	**_**	**i**	**m**	**r**	**o**

האות הבאה בטקסט: i, האות הזאת מופיעה באינדקס **3**, נעביר את i לראש הרשימה ונקבל:

0	1	2	3	4	5	6
i	**s**	**u**	**_**	**m**	**r**	**o**

האות הבאה בטקסט: i, האות הזאת מופיעה באינדקס **0**, נעביר את i לראש הרשימה ונקבל:

0	1	2	3	4	5	6
i	**s**	**u**	**_**	**m**	**r**	**o**

האות הבאה בטקסט: o, האות הזאת מופיעה באינדקס **6**, נעביר את o לראש הרשימה ונקבל:

0	1	2	3	4	5	6
i	s	u	_	m	r	o

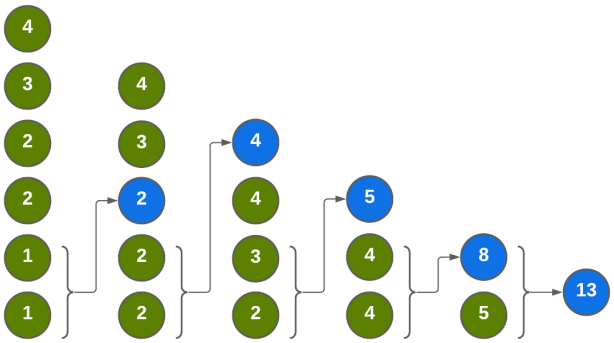
נקבל את $R = 5540444610306$.

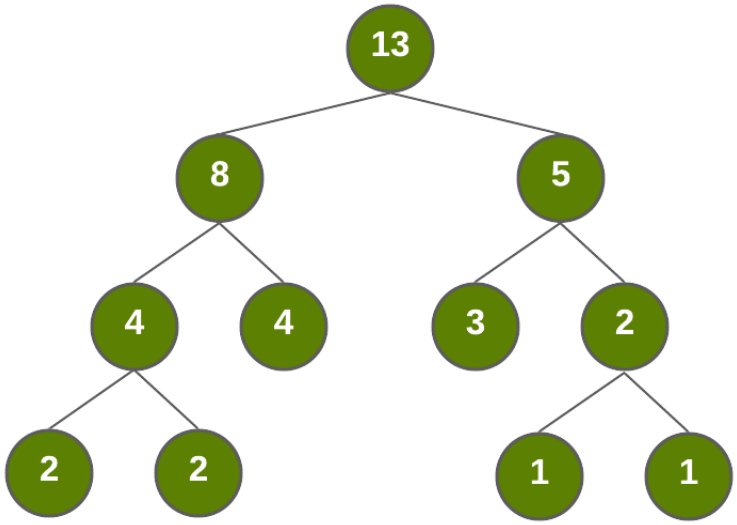
<u>פתרון ג'</u>

על מנת להפעיל את אלגוריתם הפמן נחשב את השכיחות של כל "תו" ונפעיל על השכיחויות את אלגוריתם הפמן, הא"ב שלנו לפי R הוא: $\Sigma = \{0, 1, 3, 4, 5, 6\}$ והשכיחות של כל "תו" הוא:

$$\Pr(0) = \frac{3}{13}\ ,\ \Pr(1) = \frac{1}{13}\ ,\ \Pr(3) = \frac{1}{13}\ ,\ \Pr(4) = \frac{4}{13}\ ,\ \Pr(5) = \frac{2}{13}\ ,\ \Pr(6) = \frac{2}{13}$$

נבנה את עץ הפמן לפי סדר ההסתברויות מהקטן לגדול באופן הבא:





אלגוריתם PPM

- אלגוריתם שמשתמש בקופסה שחורה (אריתמטי או הפמן).
- זה דחיסה סטטיסטית-אדפטיבית.
- עובד עם מודל מרקוב אך בסדר גבוהה יותר ממה שנלמד עד עכשיו בקורס.
- ביצועי דחיסה טובים ביותר היום אך הבעיה שלה זה שהיא לוקחת יותר זמן מה-LZSS.

השיטה

הסתברות גבוהה של תווים ← כמות נמוכה של אינפורמציה ← דחיסה טובה יותר.

- איך נעלה את ההסתברויות? - נדחוס כל תו לפי ההקשר שלו.
  עדיף לנו לקודד את X בהקשר Y מאשר לקודד את X עצמו.
  למשל, בשפה האנגלית ב-99% אחרי האות q תופיע התו u, לכן ההסתברות שסתם יופיע u בטקסט הרבה יותר נמוכה מאשר ההסתברות ש-u יופיע בהקשר של q.
- אנחנו למעשה משפרים את ההסתברויות ואחר כך מפעילים על זה "קופסה שחורה" כדי לקודד את התו בהקשר מסוים. כלומר, לכל הקשר יהיה מודל משלו.
- נעדיף להשתמש בקידוד אריתמטי כי הוא יותר טוב בהסתברויות שהם מאוד נמוכות (יכול לקודד בשברי ביטים).

האנטרופיה המותנית

נרצה להקטין את האנטרופיה המותנית שהיא $H(X \mid Y = y)$.
מוגדרת כך:

$$H(X|Y) = \sum_{y \in Y} \Pr(y) \cdot H(X|Y = y)$$

אם נוריד את האנטרופיה המותנית, נוכל לדחוס יותר טוב.
כלומר, המטרה שלנו בסופו של דבר זה להראות ש: $H(X|Y) \leq H(X)$.

$$
\begin{aligned}
&H\left(X\middle|Y\right)=\sum_{y\in Y}p\left(y\right)H\left(X\middle|Y=y\right)=\\
&=-\sum_{y\in Y}p\left(y\right)\sum_{x\in X}p(x\mid y)\log p(x\mid y)=\\
&=-\sum_{x\in X}\sum_{y\in Y}p(x,y)\log p(x\mid y)=\\
&=-\sum_{x\in X,y\in Y}p(x,y)\log p(x\mid y)=\\
&=-\sum_{x\in X,y\in Y}p(x,y)\log\frac{p(x,y)}{p(y)}=\\
&=\sum_{x\in X,y\in Y}p(x,y)\log\frac{p(y)}{p(x,y)}
\end{aligned}
$$

אנטרופיה משותפת ומותנית

לכל משתנים מקריים X, Y, האנטרופיה המשותפת של X ושל Y מוגדרת כך:

$$H(X,Y) = -\sum_{x\in X,\ y\in Y}\Pr(x,y)\cdot\log(\Pr(x,y))$$

נרצה להראות ש: $H(X|Y) = H(X,Y) - H(Y)$.

אז:

$$
\begin{aligned}
H(X|Y) &= \sum_{x\in X,\ y\in Y}\Pr(x,y)\cdot\log\left(\frac{\Pr(y)}{\Pr(x,y)}\right)=\\
&=\left[-\sum_{x\in X,\ y\in Y}\Pr(x,y)\cdot\log(\Pr(x,y))\right]+\left[\sum_{x\in X,\ y\in Y}\Pr(x,y)\cdot\log(\Pr(y))\right]=\\
&=[\,H(X,Y)\,]\ +\ [Something]=\\
&=H(X,Y)-H(Y)
\end{aligned}
$$

נותר להראות ש: $[Something] = -H(Y)$.

$$\sum_{x\in X,\ y\in Y}\Pr(x,y)\cdot\log(\Pr(y))=\sum_{y\in Y}\log(\Pr(y))\cdot\sum_{x\in X}\Pr(x,y)=$$

$$=\sum_{y\in Y}\log(\Pr(y))\cdot\sum_{x\in X}\Pr(y|x)\cdot\Pr(x)=\sum_{y\in Y}\log(\Pr(y))\cdot\Pr(y)=-H(Y)$$

קידוד הסמל הבא ברצף, בהתחשב בהקשר של הסמלים הקודמים ברצף.
על ידי שימוש בהקשר, אנו מצמצמים את האנטרופיה של הסמל הבא, וזה מאפשר לנו לקודד את הסמל הבא באמצעות פחות ביטים. נזכור שהמטרה שלנו היא להראות ש: $H(X|Y) \leq H(X)$.
כלומר המטרה היא להראות שבהסתברות מותנית המצב לא יותר גרוע - אלא משתפר.

Fact: $\sum_{x\in X} p(x)f(x) = \sum_{x\in X, y\in Y} p(x,y)f(x)$

$$H(X|Y) - H(X) = \sum_{x\in X, y\in Y} p(x,y)\log\frac{p(y)}{p(x,y)} + \sum_{x\in X} p(x)\log p(x) =$$

$$= \sum_{x\in X, y\in Y} p(x,y)\log\frac{p(y)}{p(x,y)} + \sum_{x\in X, y\in Y} p(x,y)\log p(x) =$$

$$= \sum_{x\in X}\sum_{y\in Y} p(x,y)\left(\log\frac{p(y)}{p(x,y)} + \log p(x)\right) =$$

$$= \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\frac{p(y)p(x)}{p(x,y)} \leq$$

$$\leq \log\left(\sum_{x\in X}\sum_{y\in Y} p(x,y)\frac{p(y)p(x)}{p(x,y)}\right) = 0$$

ולפי אי שוויון ג'נסן קיבלנו ש: $H(X|Y) \leq H(X)$.

הסתברות מותנית

נשתמש ב-k התווים הקודמים בתור ההקשר שלנו.
נעשה זאת בעזרת שימוש בהסתברות מותנית: $\Pr(x|s) = \frac{C(x|s)}{C(x)}$.
כאשר $C(x|s)$ הוא מספר הפעמים ש-x מופיע בהקשר של s ונחלק ב- $C(x)$ שזה מספר המופעים של x.

דוגמה
נניח ש: **e** מופיע **12** פעמים, וגם **e** מופיע **7** פעמים בהקשר של **th**.
אז ההסתברות המותנית היא: $\Pr(e \mid th) = \frac{C(e \mid th)}{C(e)} = \frac{7}{12}$.

- בדחיסת PPM ככל שנגדיל את ה-k_{max} נקבל דחיסה יותר טובה אבל אי אפשר לתחזק טבלאות גדולות מדי.
  לכן, אנחנו נרצה להשתמש ב-k קטן כדי שהמילון לא יגדל יותר מדי, בדר"כ נשתמש ב-$k \leq 8$.
- אם ניקח קובץ באנגלית ונתייחס לגודל כל המחרוזות בגודל 8 אז נקבל 2.3 ביטים לכל תו.
  אך עם ה-PPM אנחנו נוכל להגיע ל-1.7 ביטים לכל תו.

בעיה

- אנחנו מנסים לקודד תו מסוים בהקשרים שכבר ראינו - כי אנחנו רוצים שהמפענח ידע על מה מדובר (הוא יודע הקשרים שהוא כבר מכיר).
- אם אנחנו רואים תו בהקשר שלא ראינו קודם אז ההסתברות שלו תהיה אפס.
- אנחנו נרצה להימנע ממצב בו ההסתברות שווה ל-0.

למה? - כי אם נחשב את כמות האינפורמציה בהסתברות של 0 אז נקבל $\log(\frac{1}{\infty})$ ואם כמות האינפורמציה היא אינסופית אז זה אומר שצריך להשתמש באינסוף ביטים כדי לקודד את זה.

נרצה טכניקה שתדע להתמודד עם הבעיה הזאת - כלומר שתקודד לנו תו בהקשר שלא הופיע קודם לכן.

אלגוריתם PPM

- נתחיל מהקשר גבוהה - נקרא לו k_{max}.
- נבדוק האם התו הופיע בגודל ההקשר הזה:
  - אם הוא הופיע שם - אז נקודד לפי ההקשר הזה.
  - אחרת, נגיד למפענח שאנחנו יורדים בגודל של ההקשר (להקשר יותר נמוך).
- נשמור טבלה בשביל כל ההקשרים הקיימים.

למשל אם הגדרנו $k_{max} = 3$ אז נשמור טבלה עבור כל ההקשרים עד גודל 3. הכוונה לכל המחרוזות בגודל $0 \leq k \leq 3$ שהופיעו בטקסט.

דוגמה

בהינתן מחרוזת שנרצה לקודד: $S = ACCBACCACBA$.

שלב 1: ניצור טבלה על גודל כל ההקשרים, נגדיר $k_{max} = 2$.

k = ?	הקשר	תדירות	הסבר
0	ריק ε	• ההקשר ε נקבל: $A = 4,\ B = 2,\ C = 5$	$k = 0$ אז זה אומר שהמחרוזת שלנו ריקה כלומר ההקשר ריק. אם ההקשר ריק, זה בדיוק השכיחויות שאספנו במחרוזת מכל תו.
1	A B C	• ההקשר A נקבל $C = 3$. • ההקשר B נקבל $A = 2$. • ההקשר C נקבל $A = 1,\ B = 2,\ C = 2$	ההקשר הוא בעצם לקחת כל איבר בהקשר הנוכחי ולבדוק מי מופיע אחריו ונכתוב את התדירות שלו.
2	AC BA CA CB CC	• הקשר AC נקבל: $B = 1,\ C = 2$ • הקשר BA נקבל: $C = 1$ • הקשר CA נקבל: $C = 1$ • הקשר CB נקבל: $A = 2$ • הקשר CC נקבל: $A = 1,\ B = 1$	

שלב 2: נקודד את התווים A, B, C.

- נעבור על כל ההקשרים ונשאל עליהם את התווים.
- נקצה תו מיוחד (**$**) בטבלה שיגיד לנו שאנחנו עומדים לקודד תו שלא מופיע בקשר הנוכחי.

את התו **$** שיהיה בתדירות של מספר התווים שמתקיימים בכל הקשר, נציב עבור כל הקשר בטבלה וכך הוא יגיד למפענח שצריך לרדת להקשר נמוך יותר. אם בהקשר מסוים קיים כבר.

נקודד לדוגמה את התו A:

1. סיימנו את המחרוזת עם ההקשר BA שידוע גם למפענח וגם למקודד.
2. נבדוק אם A מופיע בהקשר BA, הוא לא מופיע לכן נעביר $ ונרד להקשר נמוך יותר.
3. נבדוק אם A מופיע בקשר A, הוא לא מופיע לכן נעביר $ ונרד להקשר נמוך יותר.
4. נבדוק אם A מופיע בהקשר ε (ריק) הוא כן מופיע ולכן נקודד את A עם ההסתברות $\frac{4}{14} = \frac{4}{4+2+5+|\$|}$.
5. לכן כדי לקודד את A היינו צריכים להעביר $\$, \$, \$$.

- ככה נקודד גם את B וגם את C.

שלב 3: עדכון הטבלה לאחר כל קידוד של תו שעשינו בשלב 2.

אחרי שקודדנו את C צריך לעדכן את הטבלה כדי להגיד לה שנוסף לנו תו חדש למחרוזת (התו C).

כעת המחרוזת היא $S = ACCBACCACBA\mathbf{C}$.

כלומר, צריך לעדכן את התדירויות של C, נכתוב בטבלה:

k = ?	הקשר	תדירות
0	ריק	C=~~5~~ **6** , $A = 4, B = 2$
1	A B C	• ההקשר A נקבל C=~~3~~ **4** • ההקשר B נקבל $A = 2$. • ההקשר C נקבל $A = 1, B = 2, C = 2$
2	AC BA CA CB CC	• הקשר AC נקבל: $B = 1, C = 2$ • הקשר BA נקבל: C=~~1~~ **2** • הקשר CA נקבל: $C = 1$ • הקשר CB נקבל: $A = 2$ • הקשר CC נקבל: $A = 1, B = 1$

נקודה למחשבה:

נניח ונרצה לקודד תו שלא קיים בא"ב שהוא D.

נשאל עבור ההקשר BA ונעביר $\$, נשאל עבור ההקשר A ונעביר $ ונשאל עבור ההקשר ε ונעביר שוב $\$.

כעת הגענו למצב בו $k = -1$, במקרה שכזה, אנחנו נגדיר התדירויות יהיו ה-א"ב וההסתברות תהיה *אחידה*.

- מתי צריך לקודד $ בהקשר של $k = -1$? כאשר נגיע לסוף הקובץ נעשה EOF.

נקודות על אלגוריתם PPM

- K_{max} זה ההקשר הכי גדול - כלומר מפעילים את PPM עם K_{max} שמועבר למפענח.
- יש גרסה (PPM *) ששם אין K_{max} ותמיד נעלה שם להקשר גבוהה יותר.
  - חיסרון: יותר זיכרון ואיטי.
  - יתרון: מדויק יותר.
- אם אנחנו לא יכולים לתת הערכה לתו הבא אז צריך להגיע עד להקשר של $K = -1$ כי בא"ב אנחנו אמורים לדעת איך לקודד אותו, אם אנחנו משתמשים ב-ASCII עם 8 סיביות בהסתברות אחידה שהיא תהיה $\frac{1}{|\Sigma\ ASCII|} = \frac{1}{256}$.
- ממשיכים בתהליך עד שמוצאים את התו בהקשר מסוים או שהגענו להקשר $k = -1$ ואז נעביר את התו הרלוונטי.

הסבר על החלפות בין הקשרים ב-PPM

איך אנחנו אומרים למפענח שצריך כעת להשתמש בהקשר קטן יותר?
ה-**$** הוא בעצם ה-**Escape Message** שנשלח למפענח כדי ליידע שמורידים את ההקשר ב-1.
גרסת **PPMC**, שזה הגרסה שאנחנו לומדים בקורס הזה, אומר ל-$ שגודל התדירות שלו הוא כגודל ה-א"ב.

חישוב ההקשרים

חישובי פונקציית ההסתברות המותנית לפי האלגוריתם מתואר כך:

$$\Pr\left(next\ symbol \mid previous\ (K_{max})\ symbols\right)$$
$$\Pr\left(next\ symbol \mid previous\ (K_{max} - 1)\ symbols\right)$$
$$\ldots\ldots$$
$$\Pr(next\ symbol \mid previous\ symbol)$$
$$\Pr(next\ symbol)$$

ההסתברויות הנ"ל מתארות תהליך בו התו הנוכחי ($next\ symbol$) לא מוצא הקשר עם K_{max} ולכן נוריד את גודל ההקשר ל- $K_{max} - 1$ וכך הלאה עד למקרה האחרון כאשר הוא לא נמצא באף הקשר ולכן הוא תלוי בעצמו בהסתברות אחידה.

מקרה ה-Escape Code

ה-Escape Code בעצם אומר למפענח לרדת בגודל ההקשר ($k\ --$), ראינו שסימנו אותו בסימן $.
על מנת לבחור את ההסתברות של התו הבא שאנחנו מקודדים נשתמש בהקשר הגדול ביותר שיכולים למצוא שהוא בעצם k_{max} ובמידה והוא לא נמצא באורך הזה אז נקרא ל-Escape Code, נעביר למפענח $ והוא יבין שצריך לרדת בגודל ההקשר.

דוגמה לקידוד

בהינתן מחרוזת שנרצה לקודד: $S = ACCBACCACBA$ וגם $k_{max} = 2$ ובהינתן הטבלה הבאה.

נרצה לקודד את התו A.

k = ?	הקשר	תדירות
0	ריק	$A = 4,\ B = 2,\ C = 5,\ \$ = \|\Sigma\| = 3$
1	A B C	• ההקשר A נקבל: $C = 3,\ \$ = 1$ • ההקשר B נקבל $A = 2,\ \$ = 1$. • ההקשר C נקבל $A = 1,\ B = 2,\ C = 2,\ \$ = 3$
2	AC BA CA CB CC	• הקשר AC נקבל: $B = 1,\ C = 2,\ \$ = 2$ • הקשר BA נקבל: $C = 1,\ \$ = 1$ • הקשר CA נקבל: $C = 1,\ \$ = 1$ • הקשר CB נקבל: $A = 2,\ \$ = 1$ • הקשר CC נקבל: $A = 1,\ B = 1,\ \$ = 2$

<u>שלב 1</u>:

עבור ההקשר BA נראה כי A לא מוכר לו ולכן נקודד את $ בהסתברות $\Pr(\$) = \frac{|\$|}{|C|+|\$|} = \frac{1}{1+1} = \frac{1}{2}$.

<u>שלב 2</u>:

עבור ההקשר A נראה כי A לא מוכר לו ולכן נקודד את $ בהסתברות $\Pr(\$) = \frac{|\$|}{|C|+|\$|} = \frac{1}{3+1} = \frac{1}{4}$.

<u>שלב 3</u>:

עבור ההקשר ε נראה כי A מוכר לו ולכן נקודד בהסתברות $\Pr(A) = \frac{|A|}{4+2+5+3} = \frac{4}{14}$.

מה אומר לנו ההסתברויות בקידוד?

- אנחנו מכניסים ב-"קופסה שחורה" לקוד אריתמטי.
- מספר הסיביות שיקח לנו לקודד את $\Pr(\$) = \frac{1}{2}$ הוא $\lceil -\log_2\left(\frac{1}{2}\right) \rceil = 1$ סיביות.
- מספר הסיביות שיקח לנו לקודד את $\Pr(\$) = \frac{1}{4}$ הוא $\lceil -\log_2\left(\frac{1}{4}\right) \rceil = 2$ סיביות.
- מספר הסיביות שיקח לנו לקודד את $\Pr(A) = \frac{4}{14}$ הוא $\lceil -\log_2\left(\frac{4}{14}\right) \rceil = 2$ סיביות.

כלומר מספר הסיביות שיקח לנו לקודד את A יהיה $5 = 2 + 2 + 1$ סיביות.

דוגמה נוספת

בהינתן מחרוזת שנרצה לקודד: $S = ABRACADABRA$ וגם $K_{max} = 2$.

אנחנו נרצה לקודד ובהדרגתיות, כלומר לפי הסדר של המחרוזת משמאל לימין.

הטבלה הבאה תתאר לנו את השלבים עבור כל הכנסה של אות יחד עם התהליך של Escape בסימן $.

סדר הקשר	תו למפענח
2	$
1	$
0	$
-1	a
2	$
1	$
0	$
-1	b
2	$
1	$
0	$

⇒

סדר הקשר	תו למפענח
-1	r
2	$
1	$
0	a
2	$
1	$
0	$
-1	c
2	$
1	$
0	a

⇒

סדר הקשר	תו למפענח
2	$
1	$
0	$
-1	d
2	$
1	$
0	a
2	$
1	b
2	r
2	a

נקודה חשובה

נשים לב שעבור הפעם הראשונה שהכנסנו את A קודדנו $A$$$.

זה אמנם נראה פעולה ארוכה וכבדה אך למעשה אנחנו מעבירים את A בהקשר $K = -1$ כי אנחנו נתקלים בתו בפעם הראשונה ולכן כל הדולרים האלה זה הסתברות בגודל "1" ולכן אנחנו צריכים להעביר 0 ביטים.

אם זה היה טבלת ASCII היינו צריכים להעביר 8 סיביות ולא יותר מזה.

נשים לב

- אם תו מופיע בפעם הראשונה אז נשתמש בהקשר $K = -1$ כדי להעביר את התו למפענח.
- הרבה מהסימנים של $ ההסתברות שלהם היא 1 ולכן ההעברה לא עולה לנו כלום (0 סיביות העברה).

שלבי הפתרון של הדוגמה הקודמת

בהינתן מחרוזת שנרצה לקודד: $S = ABRACADABRA$ וגם $K_{max} = 2$.

אנחנו נרצה לקודד ובהדרגתיות, כלומר לפי הסדר של המחרוזת משמאל לימין.

בתרגיל זה, נראה איך אנחנו מקודדים ומה אנחנו מעבירים.

המקודד (Encoder) והמפענח (Decoder) בונים את הטבלה ביחד עבור כל תו שנעבור עליו.

נציב את התרגיל בשלבים קטנים בשביל ההבנה אך הפתרון של התרגיל לא אמור להיות מפורט בשלבים האלה.

שלב אתחול

k=0	k=1	k=2

שלב 1- ABRACADABRA

- *העברה*: נרצה להעביר את התו הראשון A
  - עבור $k = 2$ אין לו הקשר, כי הכל ריק, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ אין לו הקשר, כי הכל ריק, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 0$ אין לו הקשר, כי הכל ריק, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = -1$, אם הגענו עד לכאן, אז נעביר את **A** בהסתברות $\Pr(A) = \frac{1}{|\Sigma|} = \frac{1}{6}$.
  - לכן נעביר **A$$$**.
- *סה"כ סיביות*: $3 \cdot \lceil -\log_2(\Pr(\$)) \rceil + \lceil -\log_2\left(\frac{1}{6}\right) \rceil = 3 \cdot 0 + 3 = 3$.
- *עדכון*: גם המפענח והמקודד בונים את הטבלה בשביל A ונקבל:

k=0	k=1	k=2
$A = 1, \$ = 1$		

שלב 2- ABRACADABRA

- *העברה*: נרצה להעביר את התו השני B
  - עבור $k = 2$ אין לו הקשר, כי הכל ריק, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ אין לו הקשר עם A, כי הכל ריק, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 0$ אין לו הקשר עם ε, לכן נעביר **$** בהסתברות $\Pr(\$) = \frac{1}{1+1} = \frac{1}{2}$.
  - עבור $k = -1$, אם הגענו עד לכאן, אז נעביר את **B** בהסתברות $\Pr(B) = \frac{1}{|\Sigma|} = \frac{1}{6}$.
  - לכן נעביר **B$$$**.
- *סה"כ סיביות*: $2 \cdot 0 + 1 + 3 = 4$.
- *עדכון*: גם המפענח והמקודד בונים את הטבלה בשביל B ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A = 1, B = 1, \$ = 2$	$A \mid B = 1, \$ = 1$	

שלב 3- ABRACADABRA

- *<u>העברה</u>*: נרצה להעביר את התו השלישי R
  - עבור $k = 2$ אין לו הקשר עם AB, כי הכל ריק, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ אין לו הקשר עם B, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 0$ אין לו הקשר עם ε, לכן נעביר **$** בהסתברות $\Pr(\$) = \frac{2}{1+1+2} = \frac{1}{2}$.
  - עבור $k = -1$, אם הגענו עד לכאן, אז נעביר את **R** בהסתברות $\Pr(R) = \frac{1}{|\Sigma|} = \frac{1}{6}$.
  - לכן נעביר **$$$R**.
- *<u>סה"כ סיביות</u>*: $2 \cdot 0 + 1 + 3 = 4$.
- *<u>עדכון</u>*: גם המפענח והמקודד בונים את הטבלה בשביל R ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A = 1, B = 1, \$ = 3, R = 1$	$A \mid B = 1, \$ = 1$ $B \mid R = 1, \$ = 1$	$AB \mid R = 1, \$ = 1$

שלב 4- ABRACADABRA

- *<u>העברה</u>*: נרצה להעביר את התו הרביעי A
  - עבור $k = 2$ אין לו הקשר עם BR, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ אין לו הקשר עם R, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 0$ יש לו הקשר עם ε, לכן נעביר **A** בהסתברות $\Pr(\$) = \frac{|A|=1}{1+1+1+3} = \frac{1}{6}$.
  - לכן נעביר **$$A**.
- *<u>סה"כ סיביות</u>*: $2 \cdot 0 + 3 = 3$.
- *<u>עדכון</u>*: גם המפענח והמקודד בונים את הטבלה בשביל A ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A = 2, B = 1, \$ = 3, R = 1$	$A \mid B = 1, \$ = 1$ $B \mid R = 1, \$ = 1$ $R \mid A = 1, \$ = 1$	$AB \mid R = 1, \$ = 1$ $BR \mid A = 1, \$ = 1$

שלב 5- ABRACADABRA

- *<u>העברה</u>*: נרצה להעביר את התו החמישי C
  - עבור $k = 2$ אין לו הקשר עם RA, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ אין לו הקשר עם A, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 0$ אין לו הקשר עם ε, לכן נעביר **$** בהסתברות $\Pr(\$) = \frac{|\$|=3}{2+1+1+3} = \frac{3}{7}$.
  - עבור $k = -1$, אם הגענו עד לכאן, אז נעביר את **C** בהסתברות $\Pr(C) = \frac{1}{|\Sigma|} = \frac{1}{6}$.
  - לכן נעביר **$$$C**.
- *<u>סה"כ סיביות</u>*: $3 \cdot 0 + 2 + 3 = 5$.

- *עדכון*: גם המפענח והמקודד בונים את הטבלה בשביל C ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A = 2, B = 1, \$ = 4,$ $, R = 1, C = 1$	$A \mid B = 1, C = 1, \$ = 2$ $B \mid R = 1, \$ = 1$ $R \mid A = 1, \$ = 1$	$AB \mid R = 1, \$ = 1$ $BR \mid A = 1, \$ = 1$ $RA \mid C = 1, \$ = 1$

שלב 6- ABRACADABRA

- *העברה*: נרצה להעביר את התו השישי A
  - עבור $k = 2$ אין לו הקשר עם AC, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ אין לו הקשר עם C, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 0$ יש לו הקשר עם ε, לכן נעביר **$** בהסתברות $\Pr(A) = \frac{|A|=2}{2+1+1+1+4} = \frac{2}{9}$.
  - לכן נעביר **$$A**.
- *סה"כ סיביות*: $2 \cdot 0 + 3 = 3$.
- *עדכון*: גם המפענח והמקודד בונים את הטבלה בשביל A ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A = 3, B = 1, \$ = 4,$ $, R = 1, C = 1$	$A \mid B = 1, C = 1, \$ = 2$ $B \mid R = 1, \$ = 1$ $R \mid A = 1, \$ = 1$ $C \mid A = 1, \$ = 1$	$AB \mid R = 1, \$ = 1$ $BR \mid A = 1, \$ = 1$ $RA \mid C = 1, \$ = 1$ $AC \mid A = 1, \$ = 1$

שלב 7- ABRACADABRA

- *העברה*: נרצה להעביר את התו D
  - עבור $k = 2$ אין לו הקשר עם CA, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ אין לו הקשר עם A לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 0$ אין לו הקשר עם ε, לכן נעביר **$** בהסתברות $\Pr(\$) = \frac{|\$|=4}{3+1+1+1+4} = \frac{4}{10}$.
  - עבור $k = -1$, אם הגענו עד לכאן, אז נעביר את **D** בהסתברות $\Pr(D) = \frac{1}{|\Sigma|} = \frac{1}{6}$.
  - לכן נעביר **$$$D**.
- *סה"כ סיביות*: $2 \cdot 0 + 2 + 3 = 5$.
- *עדכון*: גם המפענח והמקודד בונים את הטבלה בשביל D ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A = 3, B = 1, \$ = 5,$ $, R = 1, C = 1, D = 1$	$A \mid B = 1, C = 1, D = 1, \$ = 3$ $B \mid R = 1, \$ = 1$ $R \mid A = 1, \$ = 1$ $C \mid A = 1, \$ = 1$	$AB \mid R = 1, \$ = 1$ $BR \mid A = 1, \$ = 1$ $RA \mid C = 1, \$ = 1$ $AC \mid A = 1, \$ = 1$ $CA \mid D = 1, \$ = 1$

שלב 8- ABRACADABRA

- *<u>העברה</u>*: נרצה להעביר את התו A
  - עבור $k = 2$ אין לו הקשר עם AD, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ אין לו הקשר עם D, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 0$ יש לו הקשר עם ε, לכן נעביר **A** בהסתברות $\Pr(\$) = \frac{|\$|=5}{12} = \frac{5}{12}$.
  - לכן נעביר **A$$**.
- *<u>סה"כ סיביות</u>*: $2 \cdot 0 + 2 = 2$.
- *<u>עדכון</u>*: גם המפענח והמקודד בונים את הטבלה בשביל A ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A = 4, B = 1, \$ = 5, R = 1, C = 1, D = 1$	$A \mid B = 1, C = 1, D = 1, \$ = 3$ $B \mid R = 1, \$ = 1$ $R \mid A = 1, \$ = 1$ $C \mid A = 1, \$ = 1$ $D \mid A = 1, \$ = 1$	$AB \mid R = 1, \$ = 1$ $BR \mid A = 1, \$ = 1$ $RA \mid C = 1, \$ = 1$ $AC \mid A = 1, \$ = 1$ $CA \mid D = 1, \$ = 1$ $AD \mid A = 1, \$ = 1$

שלב 9- ABRACADABRA

- *<u>העברה</u>*: נרצה להעביר את התו B
  - עבור $k = 2$ אין לו הקשר עם DA, לכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
  - עבור $k = 1$ יש לו הקשר עם A, לכן נעביר **B** בהסתברות $\Pr(B) = \frac{1}{1+1+1+3} = \frac{1}{6}$.
  - לכן נעביר **B$**.
- *<u>סה"כ סיביות</u>*: $1 \cdot 0 + 3 = 3$.
- *<u>עדכון</u>*: גם המפענח והמקודד בונים את הטבלה בשביל B ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A = 4, B = 2, \$ = 5, R = 1, C = 1, D = 1$	$A \mid B = 2, C = 1, D = 1, \$ = 3$ $B \mid R = 1, \$ = 1$ $R \mid A = 1, \$ = 1$ $C \mid A = 1, \$ = 1$ $D \mid A = 1, \$ = 1$	$AB \mid R = 1, \$ = 1$ $BR \mid A = 1, \$ = 1$ $RA \mid C = 1, \$ = 1$ $AC \mid A = 1, \$ = 1$ $CA \mid D = 1, \$ = 1$ $AD \mid A = 1, \$ = 1$ $DA \mid B = 1, \$ = 1$

שלב 10- ABRACADABRA

- *<u>העברה</u>*: נרצה להעביר את התו R
  - עבור $k = 2$ יש לו הקשר עם AB, לכן נעביר **R** בהסתברות $\Pr(R) = \frac{1}{1+1} = \frac{1}{2}$.
  - לכן נעביר **R**.

- *סה"כ סיביות*: $\lceil -\log_2\left(\frac{1}{2}\right) \rceil = 1$.
- *עדכון*: גם המפענח והמקודד בונים את הטבלה בשביל R ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A=4, B=2, \$=5, R=2, C=1, D=1$	$A \mid B=2, C=1, D=1, \$=3$ $B \mid R=2, \$=1$ $R \mid A=1, \$=1$ $C \mid A=1, \$=1$ $D \mid A=1, \$=1$	$AB \mid R=2, \$=1$ $BR \mid A=1, \$=1$ $RA \mid C=1, \$=1$ $AC \mid A=1, \$=1$ $CA \mid D=1, \$=1$ $AD \mid A=1, \$=1$ $DA \mid B=1, \$=1$

שלב 11- ABRACADABRA

- *העברה*: נרצה להעביר את התו A
  - עבור $k=2$ יש לו הקשר עם BR, לכן נעביר **A** בהסתברות $\Pr(A)=\frac{1}{1+1}=\frac{1}{2}$.
  - לכן נעביר **A**.
- *סה"כ סיביות*: $\lceil -\log_2\left(\frac{1}{2}\right) \rceil = 1$.
- *עדכון*: גם המפענח והמקודד בונים את הטבלה בשביל A ונקבל:

k=0	k=1	k=2
$\varepsilon \mid A=5, B=2, \$=5, R=2, C=1, D=1$	$A \mid B=2, C=1, D=1, \$=3$ $B \mid R=2, \$=1$ $R \mid A=2, \$=1$ $C \mid A=1, \$=1$ $D \mid A=1, \$=1$	$AB \mid R=2, \$=1$ $BR \mid A=2, \$=1$ $RA \mid C=1, \$=1$ $AC \mid A=1, \$=1$ $CA \mid D=1, \$=1$ $AD \mid A=1, \$=1$ $DA \mid B=1, \$=1$

סך הכל העברנו בכל השלבים

תו	קידוד	מספר סיביות
A	$$$A	3
B	$$$B	4
R	$$$R	4
A	$$A	3
C	$$$C	5
A	$$A	3

תו	קידוד	מספר סיביות
D	$$$D	5
A	$$A	2
B	$B	3
R	R	1
A	A	1

דוגמה נוספת

בהינתן מחרוזת שנרצה לקודד: $S = aabbaabb$..... וגם $K_{max} = 2$.

סדר הקשר	תו למפענח
2	$
1	$
0	$
-1	a
2	$
1	$
0	a

⇒

סדר הקשר	תו למפענח
2	$
1	$
0	$
-1	b
2	$
1	$
0	b

⇒

סדר הקשר	תו למפענח
2	$
1	$
0	a
2	$
1	a
2	b
...	...

תרגיל PPMC

בהינתן מחרוזת: $S = ABRACADABRA$ וגם $K_{max} = 2$. הא"ב הוא: $\Sigma = \{A, B, C, D, R, \$\}$.

Context (K=0)	Counts 0	Context (K=1)	Counts 1	Context (K=2)	Counts 2
ε	$A = 5$ $B = 2$ $R = 2$ $C = 1$ $D = 1$ $\$ = 5$	A B R	$C = 1$ $B = 1, \$ = 2$ $R = 1, \$ = 1$ $A = 1, \$ = 1$	AB BR RA	$R = 1, \$ = 1$ $A = 1, \$ = 1$ $C = 1, \$ = 1$ .....

תו	העברה	הסתברות	סיביות
A	$\$\$\$A$	$\Pr(\$_2) = \Pr(\$_1) = \Pr(\$_0) = 1,\ \Pr(\$_{-1}) = \frac{1}{\|\Sigma\|} = \frac{1}{6}$	3
B	$\$\$\$B$	$\Pr(\$_2) = \Pr(\$_1) = 1,\ \Pr(\$_0) = \frac{\|\$\|}{\|\$\|+\|A\|} = \frac{1}{2},\ \Pr(\$_{-1}) = \frac{1}{\|\Sigma\|} = \frac{1}{6}$	4
R	$\$\$\$R$	$\Pr(\$_2) = \Pr(\$_1) = 1,\ \Pr(\$_0) = \frac{\|\$\|}{\|\$\|+\|A\|+\|B\|} = \frac{2}{4} = \frac{1}{2},\ \Pr(\$_{-1}) = \frac{1}{\|\Sigma\|} = \frac{1}{6}$	4
A	$\$\$A$	$\Pr(\$_2) = \Pr(\$_1) = 1,\ \Pr(\$_0) = \frac{1}{6}$	3
C	$\$\$\$C$	$\Pr(\$_2) = 1,\ \Pr(\$_1) = \frac{1}{\$+B} = \frac{1}{2},\ \Pr(\$_0) = \frac{3}{3+2+1+1} = \frac{3}{7},\ \Pr(\$_{-1}) = \frac{1}{\|\Sigma\|} = \frac{1}{6}$	6
A	....	—	....
D	....	—	....
A	....	—	....
B	....	—	....
R	....	—	....
A	....	—	....

עקרון ההרחקה - EP

אם תו הופיע בהקשר גבוה יותר ולא השתמשנו בו אז אנחנו גם לא נשתמש בו בהקשר נמוך יותר ולכן אנחנו יכולים להוריד אותו מהמודל שלנו. המשמעות להוריד מהמודל זה להפוך את השכיחות שלו ל-0 וכך להעלות את ההסתברויות של שאר התווים ולכן כמות האינפורמציה יורדת וכך האיכות של הדחיסה משתפרת.

<u>לדוגמה</u>:

בהינתן מחרוזת .....$S = aabb$ וגם $K_{max} = 1$ נקבל את הטבלה:

תו שמועבר	סדר גודל ההקשר	$\Pr(a)$	$\Pr(b)$	$\Pr(\$)$
$	1	0	0	1
$	0	0	0	1
a	**-1**	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
$	1	0	0	1
a	**0**	$\frac{1}{1+1} = \frac{1}{2}$	**0**	$\frac{1}{2}$
$	1	$\frac{1}{2}$	0	$\frac{1}{2}$
$	0	**0**	0	1
b	**-1**	**0**	$\frac{1}{2}$	$\frac{1}{2}$
$	1	0	0	1
b	**0**	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{2}{5}$

כשהגענו למצב המסומן באדום, עברנו כבר את aab.

אנחנו יודעים שאם היינו צריכים לקודד פה את a במקום את b אז היינו מקודדים אותו בהקשר של $k = 1$.

אך לא קודדנו אותו, כלומר במקרה זה אנחנו יודעים ש-a לא מופיע כרגע, כלומר אין הסתברות שבכלל נקודד אותו ולכן את ה-$ אנחנו יכולים לקודד ב-1 ולכן במקרה הזה $\Pr(a) = 0$.

בגלל שלא קודדנו אותו בהקשר יותר גבוה אז אנחנו יודעים בוודאות שאנחנו לא מקודדים את a ולכן $\Pr(a) = 0$.

נשים לב שזה נכון גם על הקשר של $k = -1$, כלומר גם בהקשר זה אין הסתברות בכלל שנקודד את a בהקשר הזה ולכן במקום הסתברות $\frac{1}{3}$ אפשר לתת את ההסתברות $\frac{1}{2}$.

כלומר, היתרון פה זה שאנחנו מגדילים את ההסתברות וכך מורידים את כמות האינפורמציה וכך שיפרנו את הדחיסה שלנו.

עקרון ההרחקה EP - פורמלי

אם לא ראינו עדיין את ההקשר הנוכחי, אנחנו נשתמש ב-$. אנחנו יכולים **להוריד** תווים שהופיעו בהקשרים גבוהים יותר (כי יכולנו לקודד אותם כבר במקרה הגבוה). אם לא קודדנו אותם זה אומר שהם לא התו הבא ולכן אנחנו נוריד אותם להקשר נמוך יותר. השיטה הזאת חוסכת את הדחיסה ב-20%.

כאשר אנחנו מנסים להעריך את: $\Pr(encode\ a_i \mid previous\ k\ symbols)$ כאשר $k < k_{max}$.

אם: $count[a_i \mid previous\ (k+1)\ symbols] > 0$ כלומר שהוא הופיע בהקשר גדול יותר.

אז בהקשר הנוכחי ניתן לו: $\Pr(encode\ a_i) = 0$.

1. בעיקרון ההרחקה אנחנו לא נעשה את זה עבור $k = k_{max}$.
2. ההסתברות לקודד את a_i אנחנו שואלים מה ההסתברות שאנחנו נראה אותו בהקשר הנוכחי ולא מה ההסתברות שהוא התו הבא בקובץ.

Exclusion Principle (EP) Algorithm

```
1:  estimate_prob (k) {
2:        sum ⇐ count['$'];
3:        for(i = 1; i <= |Σ|; i++) {
4:              if(k == k_max) {
5:                    count[i] ⇐ count[a_i | prev k symbols]
6:              }
7:              else if(k >= 0) {
8:                    count[i] ⇐ (count[a_i | prev k+1 symbols]==0)*
                                  (count[a_i | prev k symbols])
9:              }
10:             else {
11:                   count[i] ⇐ (count[a_i]==0)
12:             }
13:             sum += count[i];
14:       }
15:       for(i=1 i <= |Σ|; i++) {
16:             p(encode a_i) ⇐ count[i]/sum;
17:       }
18:       p(encode '$') ⇐ count['$']/sum;
19: }
```

Exclusion Principle (EP) Algorithm - הסבר קוד	
1:	
2:	כדי לחשב את ההסתברות צריך לדעת מה סכום הכולל של השכיחויות. כיוון ש-$ לא משתתף בעיקרון ההרחקה אז נוסיף אותו ל-sum לפני שמתחילים את האלגוריתם.
3:	עוברים על כל התווים ב-א"ב.
4:	במקרה כזה אי אפשר להפעיל את עקרון ההרחקה:
5:	ולכן ה-count של i יהיה לפי ההסתברות המותנית הרגילה שלמדנו על PPM עבור כל תו.
6:	
7:	במקרה כזה אנחנו יכולים להפעיל את עיקרון ההרחקה:
8:	ה-count של התו ה-i יפעל לפי הבדיקה בה (זה בעצם אינדיקטור): אם ה-a_i הופיע בהקשר גבוהה יותר אז נקבל == 1. אם ה-a_i לא הופיע בהקשר גבוהה יותר אז נקבל == 0. במידה והוא קיבל 0 אז בגלל הכפל נחזיר ל-count פשוט 0, אחרת נחזיר את ההסתברות.
9:	
10:	אחרת, כלומר כאשר k=-1 אז:
11:	אנחנו נותנים ל-count את 0. כלומר אם הוא הופיע קודם בטקסט אז אנחנו יודעים שהיינו צריכים לעצור בהקשר גבוהה יותר ולכן מוחקים אותו בהקשר של k=-1.
12:	
13:	נצבור ב-sum את סכום השכיחויות.
14:	
15:	עוברים על כל התווים ב-א"ב:
16:	ההסתברות של a_i זה פשוט מספר המופעים שלו חלקי מספר כל השכיחויות.
17:	
18:	בגלל ש-$ לא משתתף באלגוריתם אז צריך גם לתת לו את ההסתברות בסוף האלגוריתם
19:	

תרגיל למימוש עקרון ההרחקה

בהינתן המחרוזת $S = aabb$ צריך למלא את הטבלה הבאה כאשר $K_{max} = 1$.

k=1			k=0		
הסתברות	תדירות	הקשר	הסתברות	תדירות	הקשר
$\Pr(a) = 1/4$ $\Pr(b) = 1/4$ $\Pr(\$) = 1/2$	$a = 1$ $b = 1$ $\$ = 2$	a	$\Pr(a) = 1/3$ $\Pr(b) = 1/3$ $\Pr(\$) = 1/3$	$a = 2$ $b = 2$ $\$ = 2$	ε
$\Pr(b) = 1/2$ $\Pr(\$) = 1/2$	$b = 1$ $\$ = 1$	b			

כעת, נניח שה-א"ב הוא $\Sigma = \{a, b, c, \$\}$ והמחרוזת $aabb$ כבר עברה קידוד (כלומר מילאנו את הטבלה).
איך נקודד את c כעת?

- בגלל ש: $K_{max} = 1$ אז צריך להסתכל על רצף התווים האחרונים במחרוזת במקרה זה, זה רק התו b.
- נשים לב שהתו c לא מופיע בתדירויות שלו ולכן נשתמש ב-$ בהסתברות של ½ .
- נעבור להקשר נמוך יותר, גם כאן c לא מופיע ונשתמש ב-$ בהסתברות של ⅓.
- הגענו להקשר של $k = -1$, נקודד את c עם ההסתברות ¼ .
- כלומר קודדנו את c עם 5 סיביות.

k=1			k=0		
הסתברות	תדירות	הקשר	הסתברות	תדירות	הקשר
$\Pr(a) = 1/4$ $\Pr(b) = 1/4$ $\Pr(\$) = 1/2$	$a = 1$ $b = 1$ $\$ = 2$	a	⅓	$a = 2$	ε
			⅓	$b = 2$	
$\Pr(b) = 1/4$ $\Pr(c) = 1/4$ $\Pr(\$) = 1/2$	$b = 1$ $c = 1$ $\$ = 2$	b	⅛	$c = 1$	
			⅜	$\$ = 3$	

האם עקרון ההרחקה יעזור לנו לחסוך בסיביות? - עד עכשיו לא השתמשנו בעקרון ההרחקה.
נרצה לקודד את c בעזרת העקרון (הטבלה הקרובה פה מלמעלה לא רלוונטית - להסתכל על תחילת העמוד).

- עבור ההקשר $k = 1$ נסתכל על c בהקשר של b, נזכור כי אי אפשר להפעיל את העיקרון כי $k = k_{max}$ אז אנחנו מקודדים את ה-$ בהסתברות של ½ . אנחנו יודעים שזה בוודאי לא התו b כי הוא כבר מופיע בהקשר הזה.
- עבור ההקשר $k = 0$, התדירות של התו b הופך להיות 0 כיוון שהוא כבר הופיע. נקודד את $ אבל ההסתברות עכשיו תהיה ½.

- עבור ההקשר $k = -1$ עכשיו נקודד את c, נשים לב שהורדנו את b, מאחר שגם ראינו כבר את התו a בשלב $k = 0$ אז גם אותו נוריד. נקודד את התו c בהסתברות $\frac{1}{2}$ כי זה מצב בו $\Sigma = \{c, \$\}$.
  כי מחקנו גם את a בשלב 0 וגם את b בשלב 1.

כלומר כעת קודדנו את c עם 3 ביטים במקום 5 ביטים.
לכן שימוש בעקרון ההרחקה עזר לנו לחסוך במספר הסיביות וכך שיפר את הקידוד לאיכותי.

תרגיל למימוש עקרון ההרחקה

מתוך מבחן 2022 מועד א' שאלה 5.

א. בנו את הטבלה של המצב הנוכחי "$ATGACGA$" עם $k_{max} = 3$.

ב. הוסיפו את התו T בסוף הקלט, מה ההסתברות ל-T הזה ללא EP.

ג. מה ההסתברות ל-T הזה עם EP.

<u>פתרון א'</u>

k=3			k=2			k=1			k=0		
הסתברות	תדירות	הקשר	הסתברות	תדירות	הקשר	הסתברות	תדירות	הקשר	הסתברות	תדירות	הקשר
$\frac{1}{2}$	$A = 1$	ATG	$\frac{1}{2}$	$G = 1$	AT	$\frac{1}{4}$	$T = 1$	A	$\frac{3}{11}$	$A = 3$	ε
$\frac{1}{2}$	$\$ = 1$		$\frac{1}{2}$	$\$ = 1$		$\frac{1}{4}$	$C = 1$				
$\frac{1}{2}$	$C = 1$	TGA	$\frac{1}{2}$	$A = 1$	TG	$\frac{2}{4}$	$\$ = 2$		$\frac{1}{11}$	$T = 1$	
$\frac{1}{2}$	$\$ = 1$		$\frac{1}{2}$	$\$ = 1$		$\frac{1}{2}$	$G = 1$	T	$\frac{2}{11}$	$G = 2$	
$\frac{1}{2}$	$G = 1$	GAC	$\frac{1}{2}$	$C = 1$	GA	$\frac{1}{2}$	$\$ = 1$				
$\frac{1}{2}$	$\$ = 1$		$\frac{1}{2}$	$\$ = 1$		$\frac{2}{3}$	$A = 2$	G	$\frac{1}{11}$	$C = 1$	
$\frac{1}{2}$	$A = 1$	ACG	$\frac{1}{2}$	$G = 1$	AC	$\frac{1}{3}$	$\$ = 1$		$\frac{4}{11}$	$\$ = 4$	
$\frac{1}{2}$	$\$ = 1$		$\frac{1}{2}$	$\$ = 1$		$\frac{1}{2}$	$G = 1$	C			
			$\frac{1}{2}$	$A = 1$	CG	$\frac{1}{2}$	$\$ = 1$				
			$\frac{1}{2}$	$\$ = 1$							

<u>פתרון ב'</u>

אנחנו נרצה להכניס את T בלי עקרון ההרחקה כלומר עבור המצב $\underline{T}ATGACGAT$.

- בהקשר $k = 3$ אין לנו את CGA ולכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
- בהקשר $k = 2$ אז עבור GA נשים לב ש T לא שייך לשם ולכן נעביר **$** בהסתברות $\Pr(\$) = \frac{1}{2}$.
- בהקשר $k = 1$ אז עבור A נשים לב ש T שייך לשם ולכן נעביר **T** בהסתברות $\Pr(T) = \frac{1}{4}$.

לכן סה"כ נקבל:

$$\$(1),\ \$(\tfrac{1}{2}),\ T(\tfrac{1}{4}) = \left\lceil -\log_2(1) - \log_2\left(\tfrac{1}{2}\right) - \log_2\left(\tfrac{1}{4}\right)\right\rceil = 0 + 1 + 2 = 3$$

<u>פתרון ג'</u>

עקרון ההרחקה אומר שאם התו נמצא בהקשר גבוהה יותר ולא השתמשנו בו אז בטוח גם לא נשתמש בו בהקשר נמוך יותר ולכן נוכל להוריד אותו וההסתברות שלו תהיה פשוט 0.

בנוסף אנו יודעים כי לא ניתן להפעיל את עקרון ההרחקה על k_{max}.

אנחנו נרצה להכניס את T עם עקרון ההרחקה כלומר עבור המצב $\underline{T}ATGACGAT$.

- בהקשר $k = 3$ אין לנו את CGA ולכן נעביר **$** בהסתברות $\Pr(\$) = 1$.
- בהקשר $k = 2$ אז עבור GA נשים לב ש T לא שייך לשם ולכן נעביר **$** בהסתברות $\Pr(\$) = \frac{1}{2}$.

אך בגלל שלא השתמשנו ב-C אחרי GA אז בהקשר הנמוך הבא אנחנו בוודאי גם לא נשתמש בו.

- בהקשר $k = 1$ אז עבור A נשים לב ש T שייך לשם ולכן נעביר **T** בהסתברות $\Pr(T) = \frac{1}{3}$.

נשים לב כי בהקשר $k = 2$ לא השתמשנו ב-C ולכן בהקשר $k = 1$ אז $C = 0$.

כלומר:

k=1		
הקשר	**תדירות**	**הסתברות**
A	$T = 1$	$\frac{1}{1+0+2} = \frac{1}{3}$
	$C = 0$	$\frac{0}{1+0+2} = 0$
	$\$ = 2$	$\frac{2}{1+0+2} = \frac{2}{3}$

לכן סה"כ נקבל:

$$\$(1),\ \$(\tfrac{1}{2}),\ T(\tfrac{1}{3}) = \left\lceil -\log_2(1) - \log_2\left(\tfrac{1}{2}\right) - \log_2\left(\tfrac{1}{3}\right)\right\rceil \approx 0 + 1 + 1.584 \approx 2.584$$

אפשר לראות שעקרון ההרחקה יותר טוב כי הוא שיפר את הקידוד עם פחות סיביות $2.584 < 3$.

דחיסה דקדוקית (חסרת-הקשר)

כאשר מדברים על שפה - היא מורכבת מתווים וסימנים - פיסוק, ומהם אנו יוצרים מילים שיוצרים משפטים.
באופן רעיוני מספר המשפטים הוא אינסופי וכך גם מספר המילים.
כדי שזו תהיה שפה פורמלית, צריך לבנות כללים שצריך לעקוף, שהם כללי דקדוק.
את הכללים ניתן לייצג כך:

1. Terminal symbols - אבני היסוד / התווים
2. Production rules - כללי היצירה $A \to a$

נשתמש באותם כללי גזירה בשביל לדחוס.

חסר הקשר - לא מתייחסים לתווים שהיו לפני התו הנוכחי.

עקרון Straight Line Grammar

העיקרון הזה הוא מקרה פרטי שבו יוצרים מחרוזת אחת בלבד כדי שזה יהיה UD.

1. ניצור את אותו דקדוק שמתאים למחרוזת הנוכחית.
2. נעביר את הדקדוק למפענח.
3. נדחס את הדקדוק עצמו.

המפענח יפרוס את הדקדוק ואז באמצעותו הוא יגיע למחרוזת המקורית.

אלגוריתם Sequiter

זה בעצם דחיסה, הרעיון של האלגוריתם הוא לקרוא את התווים אחד אחד משמאל לימין וכדי למצוא את המחרוזת כולה זה יקח לו $O(n)$.
אנחנו נקרא כל תו ונעמוד ב-2 חוקים, ברגע שנפר איזשהו כלל נעשה תיקון ונמשיך הלאה.
החוקים:

1. לא נשאיר 2 תווים סמוכים שיחזרו על עצמם יותר מפעם אחת.
2. חוק צריך להופיע לפחות פעמיים.

האלגוריתם

כל עוד אנחנו לא ב-EOF, אז נבדוק:

1. אם יש 2 תווים צמודים שחוזרים על עצמם 0 ניצור חוק דקדוק חדש שמתקן את זה.
2. אם יש חוק שמשתמשים בו רק פעם אחת - נסיר אותו ונציג במקומו את הצורה המוכלת שלו.

דוגמה

לדוגמה $abcdbcabcdbc$.

המחרוזת ההתחלתית $S \to AA$ וגם יש לנו $A \to abcdbc$

יש כאן הפרה של חוק (1) כיוון ש: a**bc**d**bc** כלומר ה-bc הם שכנים ומופיעים יותר מפעמיים.

הפתרון: ליצור תלות חדשה $B \to bc$ ואז נקבל $aBdB$ וזה נכון כלפי החוק.

דוגמה

נתונה מחרוזת התחלתית $S \to CC$ וגם יש לנו $C \to BdA$ וגם $B \to aA$ ו- $A \to bc$.

אז נשים לב שיש כאן הפרה של החוק (2) כיוון ש-B מופיע רק פעם אחת ולכן נמחק את התלות הזאת.

אלגוריתם Re-Pair

1. נמצא את הזוג הכי שכיח ab בטקסט S.
2. ברגע שמצאנו אותו, פשוט ניצור חוק דקדוק $A \to ab$
3. נחליף את כל המופעים של ab בטקסט S על ידי A.
4. נפעל כך שוב ושוב עד שכל זוג יופיע רק פעם אחת.

- T= singing_do_wah_diddy_diddy_dum_diddy_do

$A \to \_d$	singingAo_wahAiddyAiddyAumAiddyAo
$B \to dd$	singingAo_wahAiByAiByAumAiByAo
$C \to Ai$	singingAo_wahCByCByAumCByAo
$D \to By$	singingAo_wahCDCDAumCDAo
$E \to CD$	singingAo_wahEEAumEAo
$F \to in$	sFgFgAo_wahEEAumEAo
$G \to Ao$	sFgFgG_wahEEAumEG
$H \to Fg$	sHHG_wahEEAumEG

קודי Tunstall

המטרה פה היא לעשות $Variable - to - fixed$.
כלומר יש לנו מחרוזת באורך משתנה ואנחנו אמורים ליצור מילות-קוד באורך קבוע.

בעיה הפוכה להפמן

זה בעיה בשם $Inverse\ Huffman\ Problem$.
כלומר קודי Tunstall עובדים ההפך מהפמן.
בהפמן הדגש היה על הקידוד - דחיסה כמה שיותר טובה.
פה הדגש הוא על הפענוח שהוא יהיה *יותר מהיר* ועבור מילות קוד באורך קבוע, המפענח פשוט שובר לבלוקים.

	הפמן	הופכי להפמן
מה נתון	תתי-מחרוזות בטקסט (תווים)	מילות-קוד
מה נחפש	מילת קוד באורך משתנה	תתי-מחרוזות בטקסט באורך משתנה
על מה הדגש	קידוד	פיענוח

- חשוב לציין כי האלגוריתם עמיד כאשר יש טעויות של החלפה או השמטה של סיביות.

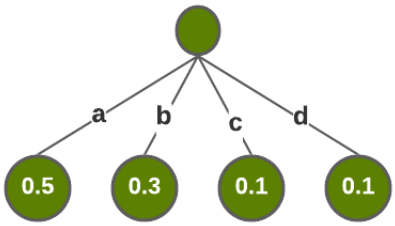
Tunstall Algorithm

```
1:  Tunstall(n){
2:        D ⇐ Σ
3:        while |D| ≤ 2^n − |Σ| {
4:              let d ∈ D be with highest probability
5:              D ⇐ D ∪ { ∪_{σ∈Σ} dσ }
6:              if d ∉ Σ then:
7:                    D ⇐ D - {d}
8:        }
9:  }
```

Tunstall Algorithm - הסבר קוד	
1:	ה-n זה מספר הסיביות שהקצנו לכל מילת-קוד.
2:	יש לנו מעין מילון D שמאותחל בא"ב שלנו.
3:	כל עוד הגודל של המילון קטן או שווה לביטוי אז נרצה להשלים את השימוש ב-2^n אופציות למילות-קוד. (כלומר כל עוד יש לי מקום להכניס מילים למילון).
4:	מפצלים - לוקחים את המילה במילון עם ההסתברות הכי גבוהה.
5:	מוסיפים את כל הפיצולים האפשריים.
6:	אם $d \notin D$ כלומר אם d אינו תו מקורי בא"ב אז:
7:	אז נמחק אותו משום שאין צורך באותה מחרוזת במילון כי יצרנו את כל האופציות.

דוגמה פרקטית

נתונים לנו הא"ב $\Sigma = \{a,\ b,\ c,\ d\}$ וגם סדרת ההסתברויות בהתאם $P = \{0.5,\ 0.3,\ 0.1,\ 0.1\}$.
וגם $n = 4$. נסדר בעץ את הנתונים ההתחלתיים:



תחילה $D = \Sigma$ ולכן $|D| = 4$.

נחשב $2^n - |\Sigma| = 2^4 - 4 = 16 - 4 = 12$.

כעת נתחיל את הלולאה, כל עוד $|D| \leq 12$ אז:

<u>איטרציה 1</u>:

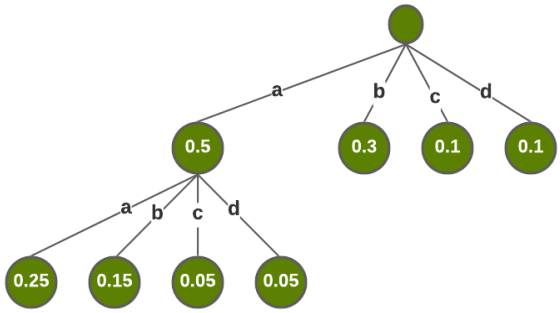
1. בדיקת כניסה ללולאה, אכן $4 \leq 12$ אז נרוץ:
2. נבחר את 'a' כיוון שהוא עם ההסתברות הגבוהה ביותר, כלומר $d = $ 'a'.
3. נוסיף למילון D את כל המילים שנבנו מהמילה d עם שרשור כל תו σ מתוך הא"ב, כלומר:

$$D \Leftarrow D \cup \{aa,\ ab,\ ac,\ ad\}$$

$$D = \{a, b, c, d, aa, ab, ac, ad\}$$

4. נבדוק האם $d = $ 'a' $\in \Sigma$ כלומר הוא כן שייך לא"ב ולכן לא נפעיל את ה-if.

כעת נצייר את העץ עם הנתונים החדשים במילון, כל ערך הסתברותי של כניסה חדשה נוצר לפי מכפלה של הסתברויות למשל עבור ab ההסתברות היא 0.15 בגלל ש: $0.3 \cdot 0.5 = 0.15$.



<u>איטרציה 2</u>:

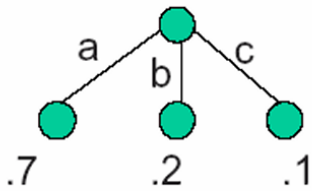
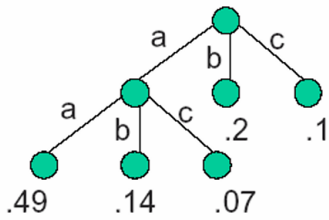
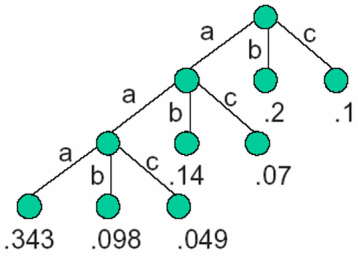
1. בדיקת כניסה ללולאה, אכן $8 \leq 12$ אז נרוץ:

להשלים את הדוגמה הזאת ולשאול לגביה בהרצאה

דוגמה

עבור $n = 3$ וגם ההסתברויות $\Pr(c) = 0.1$, $\Pr(b) = 0.2$, $\Pr(a) = 0.7$

הוא אכן עצר אחרי איטרציה שניה כיוון ש: $2^n - |\Sigma| = 2^3 - 3 = 8 - 3 = 5$

שלב 1	שלב 2	שלב 3
		

חלאס.