תוכן עניינים

להשלים:

- הוכחות מחלק ראשון -קראפט - מיידי וכד'
- הפמן דינאמי ו $D-ary$
- טבלה ארימתטי הפמן
- מדוע צריך EOF הדוגמה עם הרצפים של a ?
- ארימתטי להבין עד הסוף
- $trie$

להשלים 2 :

- סקלטון
- מימוש ארימתיטי עשרוני עם אנדפלוו
- LZSS עם עץ AVL סרטון 10 $2:00$

1 תובנות

- קוד שלם אינו בהכרח אופטימלי
  - דוגמה נגדית: נקח עץ הפמן ונחליף בין יקר בזול
  - הקרפאט ישמר + בנינו עץ מלא
- קוד מיידי אינו בהכרח שלם
- קוד שלם בהכרח מיידי : שלם $\Leftarrow$ עץ מלא $\Leftarrow$ $prefix-free$ $\iff$ מיידי
- קוד בינארי עם 2^k מילים מביא יתירות מינמלית
- C_γ למעשה חיפוש בבלוקים מגדול חזקות של 2 (החלק האונארי - האורך), ואז חיפוש בינארי בקוד (החלק הבינארי ללא ה1)
- קל להפוך אריתמטי לאדפיטבי

האפמן מול ארימתטי:

- האפמן:
  - הקלט הופך למילות קוד
  - צריך את ההסבתרויות
  - קשה להפוך לאדפטבי
  - צריך לשמור את מילות הקוד בטבלה
  - אורך מילת קוד מינמלי הוא 1
- אריתמטי:
  - כל הקלט הופך למספר בודד

- אדיפטיבי קל
- לא צריך את ההתפלגויות
- לא צריך לשלוח מילות הקוד
- "ניתן ליצג בשברי ביטים"

2 משפטים והגדרות

שיעור 1

- שני סוגי דחיסה:
  - $Lossles\ compression$: דחיסה חסרת הפסדים
  זה מתאים לקבצי טקסט או מספרים. נדחוס וכאשר נפרש נרצה להגיע בדיוק לקובץ המקורי. כלומר נבחר פונקציות הפיכות
  - $Lossy\ compression$ - דחיה בעלת הפסדים
  כלומר מפסידה מידע במהלך הדחיסה, מתאים יותר לתמונות. הקובץ המקורי שונה מהקובץ שנפרש. לדוג' $JPEG$
- קוד - אוסף של מילות קוד
- $fixed\ length$ - מילות קוד באורך קבוע . דוג' $Ascii$
- $variable\ length$ - מילות הקוד באורך משתנה
- א"ב של קובץ המקור $S := [s_1, s_2, ..., s_n]$
- הסתברות $P = [p_1, p_2, ..., P_n]$, כך ש: $\sum_{i=1}^{n} p_i = 1$, ונניח להניח כי $p_1 \geq p_2 \geq .. \geq p_n$
- מילות קוד $C = [c_1, c_2, ...c_n]$
- אורך מילות קוד $|C| = [|c_1|, ..., |c_n|]$
- אורך מילת קוד ממוצע : $E(C,P) = \sum_{i=1}^{n} p_i \cdot |c_i|$
- $prefix - free\ codeword$ - קוד חסר רישות - אף מילת קוד אינה רישא של מילת קוד אחרת, נאמר על קוד אשר מקיים תכונה זאת ככקוד פרפקסי
- קוד UD - קוד הניתן לפענוח בצורה יחידה.
  - נרצה שכל רצף של תוים נוכל לפעמח בצורה יחידה

$$Prefix - free \Rightarrow UD$$
$$\neg Prefix - free \nRightarrow \neg UD$$

- קוד שלם - $complete\ code$ - כל מחרוזת אינסופית למחצה ניתנת לפענוח בצורה יחודית - (נותן עץ מלא)
- אינסופית למחצה - אינסופית בכיוון (יש התחלה)
- קוד מיידי $Instanteous$ - המפענח מזהה את המילה מיד בסיום קריאתה
- $dangling\ suffix$ - סיפא מתנדנת. (נשתמש למבחן UD)
  - ניקח 2 מחרוזות בינאריות a ו b , כך שכל אחת מילת קוד בפני עצמה , $|a| = k$, $|b| = n$, ו $k < n$.

– אם k הביטים הראשונים של b הם a , אז a היא הרישא של b וה $(n-k)$ ביטים האחרונים של b נקראים סיפא מתנדנת = $dangling\ suffix$.

- $left\ qoutient$ - מנה שמאלית (נשתמש למבחן UD)

– עבור מחרוזות נתונות S ו T , המנה השמאלית היא כל ה $dangling\ suffix$ שקיימים.

$$S^{-1}T=\left\{\underbrace{d}_{dangling\ suffox}\ \mid ad\in T,\underbrace{a}_{prefix}\in S\right\}$$

- קוד $Minumum\,redundancy\ codes$ - בהינתן הסתברויות מסוימות אין עוד אפשרות של קידודים אחרים שיכולים להפחית את אורך מילת קוד הממוצעת מעבר למה שנקבל ב $minimul$

– **פורמלית:** יהי $E\left(C,P\right)$ אורך קוד ממוצע עבור C , אז C הוא קוד בעל יתירות מינמלי עבור ההסתברות P אם $E\left(C<P\right)\leq E\left(C',P\right)$ לכל קידוד C'

- **אינפורציה:** בהינתן סמל s_i והסתברות p_i אז כמות האינפורציה של s_i היא $-\log_2 p_i=$ כמות הביטים המינמלית כדי לייצג את s_i

תכונות (נובע מהתנהגות הlog):

– אם $p_i=1$ אז $l(s_i)=0$

– אם $p_i=0$ אז $l(s_i)=1$

– אם יש רצף תווים s_is_j שהסתברויותהם $p_i\cdot p_j$ מכך שבת"ל:

$$I\left(s_is_j\right)=I\left(s_i\right)+I\left(s_j\right)=-\log_2\left(p_i\right)+\left(-\log_2 p_j\right)=-\log_2\left(p_ip_J\right)$$

- **אנטרופיה:** כיון שחייב להקצות ביטים שלמים שנון הגדיר: $H(P)=-\sum\limits_{i=1}^{n}p_i\cdot\log_2 p_i=-\sum\limits_{i=1}^{n}P_i\cdot I\left(s_i\right)$

– האנטרופיה מהווה חסם תחתון לאורך מילת הקוד הממוצעת $HP\leq E\left(C,P\right)$

- א"ש קראפט: יהיה $C=\left[c_1,c_2,...,c_n\right]$ להיות קוד עם n מילות קוד עם אורכים $|C|=\left[\left|c_1\right|,...,\left|c_n\right|\right]$ אז אם C הוא UD אז:

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

- אם $K(C)=\sum\limits_{i=1}^{n}2^{-|c_i|}\leq 1$ עבור קוד C כשלהו אז קיים קוד C' אחר, כך ש:

– $E\left(C,P\right)=E\left(C',P\right)$

– $|C'|=|C|$

– C' קוד חסר רישיות

כלומר קיים קוד חסר רישיות אופטימלי

- קוד אוניברסלי: קוד שבו מספר הסיביות לתן הוא $O\left(\log x\right)$

– אם נניח בשלילה ש $p_x>\frac{1}{x}\Leftarrow\sum\limits_{i=1}^{x}p_i>\sum\limits_{i=1}^{x}\frac{1}{x}=1$

– לכן ה $I\left(s_x\right)=-\log p_x\geq\log x^{-1}=\log x$

כללי:

(* חישוב מספרים לא שלמים בבינארית

$$\begin{matrix} 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \\ \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel \\ 8 & 4 & 2 & 1 & \frac{1}{2} & \frac{1}{4} & \frac{1}{8} \end{matrix}$$

- כך יהיה לנו אותו עיקרון כמו מספרים שלמים שאנחנו מכירים

דוגמא -

$$\frac{7}{16} = \underline{\underline{0}}\cdot\frac{1}{2} + \underline{\underline{1}}\cdot\frac{1}{4} + \underline{\underline{1}}\cdot\frac{1}{8} + \underline{\underline{1}}\cdot\frac{1}{16} \Rightarrow {\frac{7}{16}}_2 = 0.0\underline{111}$$

(*) כמו שבמספרים שלמים יש לנו אינסוף אפסים משמאלה, כאן יהיה אינסוף אפסים ימינה.

משפטים

2.0.1 קוד מיידי $\iff$ קוד חסר־רישות

הוכחה **להשלים**

מי

- משפט: לא כל קוד בעל פענוח יחיד הוא קוד מיידי
- אלגוריתם לזיהוי UD
- אלגוריתם $Sardinas–Patterson$
- משפט הקידוד של שנון

משפט: קיים קוד מיידי C עם מילות קוד באורכים $l_1 \le l_2 \le ... \le l_n \iff K(c) = \sum_{i=1}^{n} 2^{-l_i} \le 1$

- במילים: בהנתן קוד c נאמר שהוא קוד מיידי אסם מקיים את קראפט

<u>קוד מיידי $\Leftarrow K(c) \le 1$</u>

- נבנה עץ מלא כאורך המילה הכי ארוכה נסמנה ב l_n (כאילו יש n מילות־קוד) $\Leftarrow$ בעץ יש 2^{l_n} עלים
- נבנה עץ שהוא מתאים לקוד חסר־רישות $\Leftarrow$ עבור כל מילת קוד באורך l_i - אנחנו מורידים $2^{l_n - l_i}$ עלים
- כלומר לכל בחירת מילים "זורקים" עלים (= זורקים מילים)
- נשים לב שלא נוכל לזרוק יותר מ 2^{l_n} , ולכן האי־שיוון מוגדר היטב:

$$2^{l_n}\sum_{i=1}^{n} 2^{-l_i} = \sum_{i=1}^{n} 2^{l_n - l_i} \le 2^{l_n}$$
$$\Downarrow$$
$$\sum_{i=1}^{n} 2^{-l_i} \le 1 = K(C)$$

$K\left(C\right)\leq1 \Leftarrow$ קוד מיידי

- על ידי בניית העץ
- נבחר את מילות הקוד הקצרה ביותר l_1 , אוטומטית היא "זוקת" את העלים שמתחתיה שזה: $2^{l_n-l_1}$
  - ומתקיים ש: $2^{l_n-l_1}\leq2^{l_n}$
- נבחר את מילות הקוד l_2 אוטמטית היא זורקית את העלים שמתחתיה שזה : $2^{l_n-l_1}$
  - ומתקיים ש:$2^{l_n-l_1}+2^{l_n-l_2}=2^{l_n}\left(2^{-l_1}+2^{-l_2}\right)\leq2^{l_n}$
- ...
- לבסוף עם העץ מלא נבחר את המילה ה l_n ונקבל ש: $2^{l_n}\left(2^{-l_1}+2^{-l_2}+...+2^{-l_n}\right)\leq2^{l_n}$
- כעת כבר לא נוכל להוסיף מילים כי $K\left(C\right)\leq1$ כבר יהיה גדול מ1
- הראנו$K\left(c\right)\Leftarrow$בנינו עץ חסר רישות $\Leftrightarrow$ קוד מיידי

קוד

שיעור 2

קוד $UD\iff K\left(c\right)\leq1$

הוכחה:

כיון ראשון: $K\left(c\right)\leq1\Leftarrow$ מיידי $\Leftarrow UD$

כיון שני $UD\Rightarrow K\left(c\right)\leq1$:

- יהיה קוד UD עם אורכים $l_1,...,l_n$ צ"ל $\sum_{i=1}^{n}2^{-l_i}\leq1$
- נצמצם את הסכום לפי האורכים הזהים, כלומר נסמן ב d_j את מספר המילים באורך j (אם האורך לא קיים המקדם 0)

  ויתקיים ש: $\sum_{i=1}^{n}2^{-l_i}=\sum_{j=1}^{\boldsymbol{m}}d_j\cdot2^{-\boldsymbol{j}}$, $m\leq n$
- נעלה את הסכום בk ,

$$\left(\sum_{j=1}^{m}d_j\cdot2^{-j}\right)^k=\left(d_i2^{-1}+....+d_m2^{-m}\right)^k=\sum_{j=k}^{mk}N_j\cdot2^{-j}$$

- ויהיה N_jמספר האפשרויות להרכיב הודעה באורך j (צרוף מילים) , כלומר: $N_j=\sum_{i_1+....i_k=j}d_{i_1},d_{i_2},..d_{i_k}$
- אבל הקוד $UD\Rightarrow N_j\leq2^j$ ולכן $\left(\sum d_i2^{-i}\right)^k\leq\sum_{j=k}^{km}1\leq km$
- ולכן $\left(\sum d_i2^{-i}\right)\leq\sqrt[k]{km}\underset{k\rightarrow\infty}{\longrightarrow}1$

המשך

- $Fixed\ length$ - קוד שבו כל מילת קוד היא באורך קבוע, לכומר, כל מילות הקוד באותו האורך
  - דוגמה: $ascii$
- $varialbe\ length$ - קוד שבו המילות קוד הן באורך משתנה
- משפט: מיידי $\iff$ קוד חסר רישות

2.0.2 האנטרופיה $H\left(P\right)$ מהווה חסם תחתון לאורך מילת קוד ממומצעת $E(C,P)$

הוכחה:

נניח בשלילה שקיים קידוד C עם אורך מילה ממוצעת נמוך מהאנטרופיה , כלומר

מתקיים ש: $E\left(C,p\right)<H\left(P\right)\iff H\left(P\right)-E\left(C,P\right)>0$

כעת:

$$H\left(P\right)-E\left(C,P\right)=-\sum_{i=1}^{n}p_{i}\cdot\log\left(p_{i}\right)-\sum_{i=1}^{n}p_{i}\cdot\left|c_{i}\right|\overset{1}{=}\sum_{i=1}^{n}p_{i}\cdot\left(\log\left(\frac{1}{p_{i}}\right)-\left|c_{i}\right|\right)$$

$$\overset{2}{=}\sum_{i=1}^{n}p_{i}\cdot\left(\log\left(\frac{1}{p_{i}}\right)-\log_{2}2^{\left|c_{i}\right|}\right)=\sum_{i=1}^{n}p_{i}\cdot\left(\log\left(\frac{1}{p_{i}}\right)-\log_{2}2^{\left|c_{i}\right|}\right)\overset{2}{=}\sum_{i=1}^{n}p_{i}\cdot\left(\log\left(\frac{1}{p_{i}}\right)+\log_{2}2^{-\left|c_{i}\right|}\right)$$

1. הכנסנו מינוס ללוג, ואיחדנו סיגמות . 2. חוקי לוגים.

$$\overset{2}{=}\sum_{i=1}^{n}p_{i}\cdot\left(\log\left(\frac{2^{-\left|c_{i}\right|}}{p_{i}}\right)\right)\overset{3}{\leq}\sum_{i=1}^{n}p_{i}\cdot\left(\frac{2^{-\left|c_{i}\right|}}{p_{i}}-1\right)=\sum_{i=1}^{n}p_{i}\cdot\frac{2^{-\left|c_{i}\right|}}{p_{i}}-p_{i}$$

3. $ln(x)\leq x-1$

$$=\sum_{i=1}^{n}2^{-\left|c_{i}\right|}-\sum_{i=1}^{n}p_{i}\overset{3}{\leq}1-1=0$$

4. שמאלי - קרפאט $K\left(C\right)\leq1$. ימני - סך ההסתברויות במרחב המדגם

סה"כ קיבלנו $H\left(P\right)-E\left(C,P\right)\leq0$ בסתירה להנחה, ומכאן שלא קיים קוד כזה.

2.1 שיטת מידול סטטית

- מניחים שההסתברויות בלתי תלויות
- בשיטה הזו לא משנה איזה קובץ יש לנו, אנו מניחים שכל אחד מהתווים מופיע בהסתברות $\frac{1}{256}$ ולכן משתמשים בקוד $ascii$. במקרה הזה אין $prelude$ (פתיח / $header$)
- מתקיים ש $\forall i\in\left[n\right]\ p_{i}=\frac{1}{256}$

$$H\left(P\right)=-\sum_{i=1}^{256}\frac{1}{256}\cdot\log_{2}\left(\frac{1}{256}\right)=8$$

כלומר 8 סיביות $\Leftarrow$ כל מילת קוד מקבלת 8 סיביות

- כלומר אם ההסתברות אחידה 8 סיביות זה האופטימלי (הקטע שההסתברויות לא אחידות...)

2.2 שיטת מידול סטטית למחצה - $semi-static$

- בשיטה הזו עושים מעבר זריז על הקובץ כדי לראות כמה תווים יש לנו (ללא ספירה על מופעים) ואז מניחים שההסתברות בינהם אחידה.
  - במקרה זה יש $prelude$ שאומר למפענח אילו תווים יש בטקסט , ב $prelude$ נשתמש ב $ascii$, וגם צריך להגיד למפענח כמה תווים שונים מעבירים לו.
  - ה$prelude$ יכיל:
    - גודל א"ב - מספר n - $|n| = 8\ bits$
    - התווים עצמם ב $ascii$ - $8 \cdot n\ bits$
    - מילות הקוד באורך קבוע
- דוגמה , נניח גילנו שיש 25 תווים שונים אז $p_i = \frac{1}{25}$
- דוגמה: $H(P) = -\sum_{i=1}^{25} \frac{1}{25} \log_2 \left(\frac{1}{25}\right) = 4.64 \Rightarrow 5$ (צריך לעגל כי א"א לקחת 4.64 ביט)

המפענח:

- מקבל:
  - 8 ביטים לתאור 25_2
  - $8 \cdot 25 = 200$ ביטים לתאור ב $ascii$ את הא"ב

2.2.1 שיטת מידול סטטית למחצה עם הסתברויות עצמיות

- בשיטה הזו עושים מעבר על הקובץ כדי לראות כמה תווים יש נלו וגם סופרים מופעים לכל תו על מנת לחשב הסתברות.
  - הסתברות של תו: $p_i = \frac{v_i}{m} = \frac{\text{appreances number}}{\text{msg size}}$
  - ב$prelude$ נתאר:
    - גודל א"ב - מספר n - $|n| = 8\ bits$
    - התווים עצמם ב $ascii$ - $8 \cdot n\ bits$
    - תאור שכיחויות לכל תו - $n \cdot \underbrace{\text{description}}$

2.3 עקרון הודעה בגודל מינימלי

- ישנם 2 מרכיבים:
  - תאור המודל
  - הקוד ביחס למודל
- $trade-off$ בין המודל לבין ההודעה ביחס למדול.
  - בכל פעם שנהיה יותר ספיצפים להודעה הקידוד יעשה יעיל יותר , אבל גודל של ה $prelude$ בהודעות קצרות כבר לא יהיה זניח - וזה לא טוב

2.4 מודל מרקוב מסדר ראשון - $first\ order$

- במודל זה מניחים תלות בין התווים.
  - אם אנו יודעים שתו מסוים מופיע ואחריו יש לנו הסתברות גבוהה יותר לקודד תויים מסוימים, אנו מעלים את ההסתברות של אותו תו ואז כמות האינפורמציה יורדת, ולכן גם האנטרופיה יורדת
  - במודל הזה הא"ב שלנו יהי כל הזוגות האפשרים מהא"ב המקורי של התויים הבודדים.
- קוד דיאדי - קוד שבו ההסתברויות של התויים הן חזקות של $\frac{1}{2}$
  - כמות האינפורציה של כל תו תהיה שווה בדיוק לאורך של מילת הקוד
- ויזואליזציה מבוססת עץ
  - אם יוצרים קוד פרפיקסי, אז ניתן להסתכל על העץ ומילות הקוד יהיו בעלים.
  - יהיה מסלול יחיד מהשורש עד לעלים וזאת תהיה מילת הקוד.
  - הקידוד והפענוח יהיה באמצעות מעבר על העץ, המעבר על העץ הוא די איטי

2.5 שאנון מקיים את הקרפאט:

- שאנון הגדיר שאורך כל מילה יהיה $|c_i| = \left\lceil \log_2 \frac{1}{p_i} \right\rceil$ כעת:

$$\sum_{i=1}^{n} 2^{-|c_i|} = \sum_{i=1}^{n} 2^{-\left\lceil \log_2 \frac{1}{p_i} \right\rceil} \leq \sum_{i=1}^{n} 2^{-\log_2 \frac{1}{p_i}} = \sum_{i=1}^{n} 2^{\log_2 p_i} = \sum_{i=1}^{n} p_i = 1$$

2.6 אנטרופיה לזוגות

ההסבתרות לזוג היא $p(s_1, s_2) = p(s_1) \cdot p(s_2)$ (בלתי תלויים), מהי האנטרופיה?

אז:

$$H(P(s_1, s_2)) = -\sum_{i,j} P(s_i, s_j) \log_2 p(s_i, s_j) = -\sum_i \sum_j p(s_i) \cdot p(s_j) \log_2 (p(s_i) \cdot p(s_j))$$

$$(1)$$

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

$$(2)$$

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

$$(3)$$

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

1. הנחה שבלתי תלוים. 2.חוקי לוגים ופיצול הסכום 3. האנטרופיה

- מסקנה: $L(s_1, s_2) \leq H(s_1, s_2) + 1 = 2H(P(s)) + 1$
  - $\Leftarrow$ מספר הביטים למילה יהיה: $\frac{1}{2} L(s_1, s_2) < H(P(s)) + \frac{1}{2}$
  - ולכאורה: $\frac{1}{n} L(s_1, s_2, ..., s_n) < H(P(s)) + \frac{1}{n}$ אבל זה ידרוש ממנו $prelide$ מאוד ארוך שיתאר את כל הקומבינציות

2.7 לעץ מלא בעל n עלים יש $n-1$ צמתים פנימיים

הוכחה:

בסיס $n=1$ קודקוד יחיד יש $n-1$ מצתים פנימים ✓

צעד: נניח לעץ עם $n-1$ ונוכיח ל n

- יהיה עץ עם n עלים, העץ מלא ולכן יש קודקוד כלשהו שיש לו אח, נבחר אותו
- נוריד את האחים, נקבל את הנחת האינדוקציה, נחזיר ונחשב את מספר העלים , ואת מספר הצמתים הפנימיים
- ונקבל את הדרוש.

מסקנה: תאור עץ דורש $2n-1$ ביטים. צומת פנימי יסומן ב 1 ועלה יסומן ב 0

2.8 האפמן

עץ אופטימלי = עץ השייך לקוד האופטימלי

2.8.1 למה 1 : עץ אופטימלי הוא עץ מלא

הוכחה

- נניח בשלילה שהעץ לא מלא $\Leftarrow$ קיים צומת פנימי עם בן יחיד
- נשים לב שאם:
  - ננתק את תת העץ הנפרש מילדיו של אותו בן
  - נמחק את ההבן ואת הצלע המקשרת אליו
  - נחבר את תת העץ
- קיבלנו עץ חדש בו קיצרנו בסיבית אחת את הקוד לכל הבנים בתת העץ
- $\Leftarrow$ קיבלנו עץ עם $\sum w_i l_i$ קטן יותר $\Leftarrow$ סתירה לאופיטמליות של העץ $\Leftarrow$ העץ מלא

2.8.2 למה 2 : בהנתן משקולות $w_1 \geq ... \geq w_{n-1} \geq w_n$ בעץ אופיטמלי , המשקולות הנמוכים ביותר נמצאים בשכבה הנמוכה ביותר

הוכחה:

- נניח בשלילה בה"כ w_n אינו נמצא ברמה התחתונה $\Leftarrow$ ישנו w_x המקיים:
  - $l_x > l_n \wedge w_x > w_n, w_{n-1}$
  - לw_x יש אח , היות והעץ מלא
- נבנה T' חדש על ידי החלפה בין w_x ל w_n ונתבונן ב $\sum w_i l_i$
- בשאר העץ לא נגענו לכן לכל $w_i \neq w_x, x_n$ הסכום זהה, ולכן נתבונן על w_x, w_n
  - לפני ההחלפה $w_x l_x + w_n l_n$
  - אחרי ההחלפה $w_x l_n + w_n l_x$

- ובאופן כללי:
  * $w_x > w_n \Leftrightarrow w_x - w_n > 0$ מהנחה בשלילה
  * $l_x > l_n \Leftrightarrow l_x - l_n > 0$ גם מההנחה בשלילה

- כעת:

$$(w_x - w_n)(l_x - l_n) > 0$$

$$w_x l_x - w_x l_n - w_n l_x + w_n l_n > 0$$

$$w_x l_x + w_n l_n > w_x l_n + w_n l_x$$

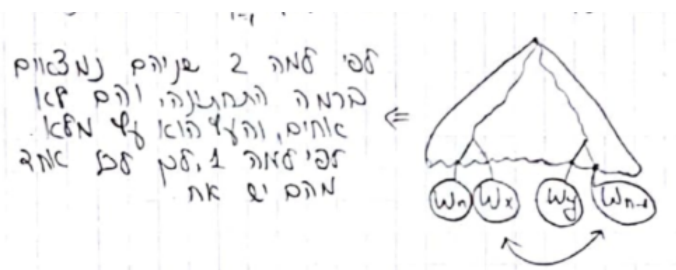
קיבלנו ש $\sum$ ב T' קטן מ $\sum w_i l_i$ בסתירה לכך שT אופיטמלי

- לכן w_x, w_n בהכרח בשכבה התחתונה, כנדרש.

למה 3 : w_{n-1}, w_n משקולות נמוכים ביותר בעץ אופיטמלי $T \Leftarrow$ w_{n-1}, w_n יכולים להיות אחים.

<u>הוכחה:</u>

- לפי למה 2 , w_{n-1}, w_n נמצאים ברמה תחתונה
- אם הם אחים, סיימנו. אחרת נקבל מלמה 1 שלכ"א יש אח:



- נבצע החלפה, והיות שהם באותו רמה האורכים זהים ולכן $\sum$ בT' זהה ל$\sum w_i l_i$ ב T כי:
  – עבור שאר האיברים לא השתנה דבר.
  – עבור המוחלפים יתקיים ש:

$$w_n l_n + w_{n-1} l_x + w_y l_y + w_x l_n = w_n l_n + w_{n-1} l_n + w_y l_y + w_x l_x$$

כי כל האורכים שווים

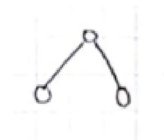
2.8.3 משפט (האפמן אופיטמלי): בהנתן משקולות $w_1, ..., w_n$ אלגוריתם הפמן בונה קוד עם אורכים $l_1, ..l_n$ כך ש $\sum_{i=1}^{n} w_i l_i$ הוא מינמלי

<u>הוכחה - באינדוקציה על מספר המשקולות n</u>

<u>בסיס:</u>

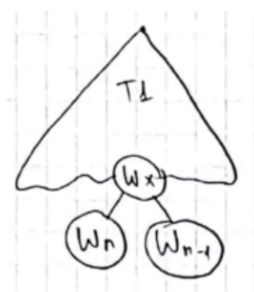
- $n = 1$ - נכון באופן ריק (יש משקולת אחת)

- $n=2$ ישנו עץ יחיד, ובפרט הפמן יתן אותו:



<u>צעד:</u> נניח ל $k<n$ הפמן בונה קוד אופיטמלי ונוכיח ל n

- יהיה T_1 עץ אופיטמלי עבור המשקלים $\{w_1,...,w_n\}$
- מלמות $3-1$ כול להניח שבתחת העץ המלא יש שני אחים עם משקולים מינמלים
  - נגדיר $w_b=w_{n-1}+w_n$
  - נסמן $M_1=\sum\limits_{i=1}^{n}w_il_i=\sum\limits_{i=1}^{n-2}w_il_i+w_{n-1}l_n+w_nl_n$



- נבנה T_2 ללא w_{n-1},w_n : נשמיט את w_{n-1},w_n אבל נשמור על $w_b=w_{n-1}+w_n$:

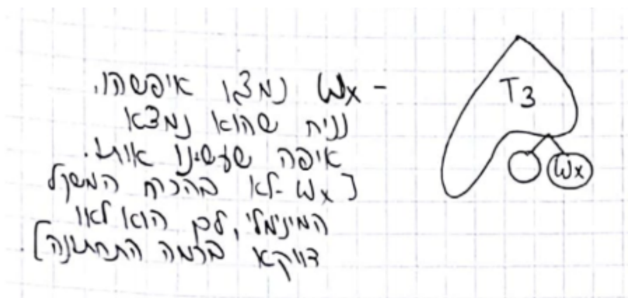
- נתבונן בסכום עבור $\{w_1,...,w_{n-2},w_b\}$:

$$l_b=l_n-1$$
$$w_bl_b=(w_{n-1}+w_n)(l_n-1)=(w_{n-1}+w_n)l_n-(w_{n-1}+w_n)$$

- כלומר $(l_{n-1}=l_n)$:

$$\boldsymbol{M_2}=\sum_{i=1}^{n-2}w_il_i+w_bl_b=\sum_{i=1}^{n-2}w_il_i+(w_{n-1}+w_n)l_n-(w_{n-1}+w_n)$$
$$=\boldsymbol{M_1-(w_{n-1}+w_n)}$$

- כעת אם נראה ש T_2 עץ אופטימלי, מהנחת האינדוקציה עבור $n-1$ משקולות, הפמן יתן $\sum\limits_{i=1}^{n-1}w_il_i$ מינימלי.
- <u>טענה: T_2 אופטימלי עבור $n-1$ עלים בעלי המשקלים: $\{w_1,...,w_{n-2},w_b\}$</u>
  - נניח בשלילה שיש עץ אופטימלי ביחס ל T_2 , כלומר ישנו T_3 עם M_3 המקיים $M_3<M_2$
  - יהיה w_b ־ לא בהכרח ברמה התחתונה ־ בT_3 (אבל כן עלה)



- ”נגדל“ חזרה את w_{n-1}, w_n על w_b (לשמור על התכונה של הפמן) $\Leftarrow$ קיבלנו T_4 ונסמן $M_4 = \sum_i^n w_i l_i$ (הl_i בM_4לא בהכרח זהים לl_i בM_1)

- מתקיים ש:

$$M_4 = M_3 + w_{n-1} + w_n < M_2 + w_{n-1} + w_n = M_1$$

- וזו סתירה לאופטימליות של T_1
- לכן $\boldsymbol{T_2}$ אופטימלי

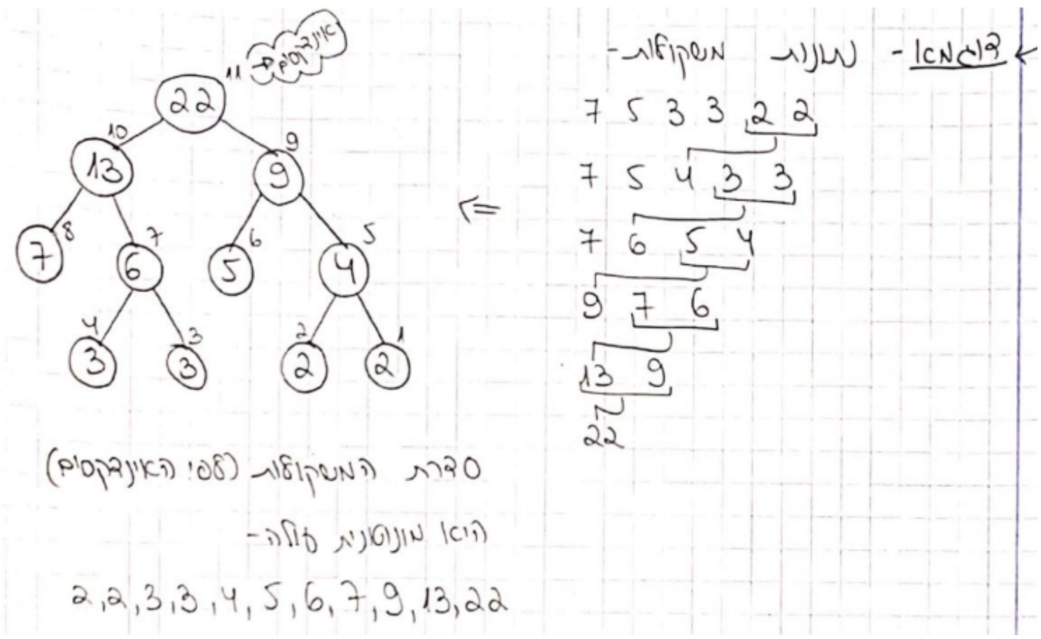
- לסיכום T_2 אופטימלי. כעת ע“פ קוד הפמן יתקיים ש $w_x = w_{n-1} + w_n$ (ואורך l_n בהתאמה) $\Leftarrow$ T_1 האופטימלי שקול להפמן , כנדרש

המשך

- עץ $D - ary$
  - עץ שבו יכולים להיות יותר מ2 בנים. מספר הבנים יהיה D
  - למשל בעץ $8 - ary$ לכל צומת יש לכל היותר 8 בנים
  - עץ $D - ary$ מלא ־ לכל צומת יש D בנים בדיוק
  - לעץ $D - ary$ מלא עם k צמתים פנימיים יש $(D-1)\cdot k + 1$ עלים
    * (עבור $D = 2$ $\Leftarrow$ $n - 1$ צמתים פנימים $\Leftarrow$ $(2-1)\, n + 1 = n + 1$ עלים

- $Sibling\, propery$

  - זו תכונה שקוראת בכל עץ הפמן, שהיא אומרת שהמשקל של כל צומת הוא סכום המשקולות של הבן השמאלי והבן הימני שלו וניתן לסדר את המשקלים בסדרה מונוטונית יורדת ועולה, ולמספר את הצמתים מלמטה עד למעלה, והאינדקסים של 2 אחים יהיו המס' האי־זוי והמס' הזוגי הבא בתור, כלומר שני אח,ים הם מספרים עוקבים
  - משפט: לעץ יש $Sibiling\ property$ $\iff$ זה עץ הפמן (נוצר ע“י אלג' הפמן)



2.8.4 ארימתטי מביא את האנטרופיה

מהו גודל ה$interval$?

- נסתכל על ה$interval$ האחרון נסמן את $l=low$,$range=r$ אז $intreval=[l,l+r)$
- ואם נסתכל על הודעה m שמכילה k תווים: $M=m_1\cdot\ldots\cdot m_k$ אז גודל r הוא:$r=\prod_{i=1}^{k}p(m_i)$ (מכפלת ה $rang$ים)
  - בדוגמה של $BILL$: $r=p(B)\cdot p(I)\cdot p(L)\cdot p(L)=\frac{1}{4}\cdot\frac{1}{4}\cdot\frac{1}{2}\cdot\frac{1}{2}$
- מספר הביטים ליצוג r ־ כפי שהסברנו בהצגה בינארית למספר עשרוני: $\left\lceil\log_2\frac{1}{r}\right\rceil$ כאשר $r=\prod_{i=1}^{k}p(m_i)$ (שבר)
  - מתקיים ש: $\lceil-\log_2(r)\rceil=\left\lceil\log_2\left(\frac{1}{r}\right)\right\rceil$
- אבחנה: אם נסמן בw_i את השכיחות ל s_i אז ההסתברות היא : $p_i=\frac{w_i}{\sum w_j}=\frac{w_i}{k}$ כאשר $M=m_1\cdot\ldots\cdot m_k$ (k גודל ההודעה)
- אז:

$$\lceil-\log_2(r)\rceil=-\log_2\left(\prod_{i=1}^{k}p(m_i)\right)\overset{1}{=}-\sum_{i=1}^{k}\log_2(p(m_i))\overset{2}{=}-\sum_{j=1}^{n}w_j\cdot\log_2 p_j$$

1. חוקי לוגים (כפל). 2. נאחד כל $\log(p_j)$ לפי מספר המופעים שלו.

$$\overset{3}{=}-k\sum_{j=1}^{n}\frac{w_j}{k}\cdot\log_2 p_j\overset{4}{=}k\sum_{j=1}^{n}p_j\cdot\log_2 p_j=kH$$

3. נכפיל ב $\frac{k}{k}$. 4. זוהי ההסתברות מהאבחנה.

3 קודים

3.1 Unique Decodability Test

- Examine all pairs of codewords:
  1. Construct a list of all codewords.
  2. If there exist a codeword, a, which is a prefix of another codeword, b, add the dangling suffix to the list (if it is not there already), until:
     (a) You get a dangling suffix that is an original codeword the code is not UD
     (b) There are no more unique dangling suffixes the code is UD

- $left\ qoutient$ - מנה שמאלית (נשתמש למבחן UD)
  - עבור מחרוזות נתונות S ו T , המנה השמאלית היא כל ה $dangling\ suffix$ שקיימים.

$$S^{-1}T = \left\{ \underbrace{d}_{dangling\ suffox} \mid ad \in T, \underbrace{a}_{prefix} \in S \right\}$$

3.1.1 Sardinas–Patterson algorithm

```
Sardinas-Patterson algorithm :
                                                        // איטרציה ראשונה //
i = 1
                    // מגדירים קבוצה אחת שלקוחת את כל dangling suffix של הקוד עצמו.
                               בכל פעם נגדיר קבוצה חדשה בעזרת הקבוצה הקודמת //
S_i = C^{-1}C - {ε}
whiel true
                          // הsuffix יכול לבוא משני הכיוונים (מההתחלה או מהסוף)
   S_{i+1} = C^{-1}S_i ∪ S_i^{-1}C
   i++
                                                  // c,ε הן מילות קוד
   if ε ∈ S_i  or c ∈ S_i for c in C :
      print not UD end exit
                    // אם הגענו למצב שהקב' נשארות אותו דבר, ולא מצאנו //
                      // dangling suffix שהיא מילת קוד מקורית אז UD //
   elss if ∃j < i such that  s_i = s_j
      print UD and exit
```

<u>דוגמת הרצה 1 :</u>

- נניח $codeword = \{0, 01, 11\}$
- בודקים האם מילה היא רישא של אחרת, ואם כן מוסיפים את ה $dangling\ suffix$ ל$codeword$
  - אצלנו: 0 רישא ל 01 $\Leftarrow$ נוסיף את 1 נקבל $S_1 = \{0, 01, 11, 1\}$

- נמשיך לבדוק $(while)$
  * 1 רישא של 11 $\Leftarrow$ ננסה להוסיף את 1 , אבל $1 \in S_1$ ולכן $S_2 = \{0, 01, 11, 1\} = S_1$
  * כמו $i++ \Leftarrow i = 2$
  * סה"כ $S_1 = S_2, 1 < 2 \Leftarrow$ הקוד UD

דוגמת הרצה 2:

- נניח $codeword = \{0, 01, 10\}$
- 0 רישא ל 01 $\Leftarrow$ נוסיף את 1 נקבל $S_1 = \{0, 01, 10, 1\}$
- 1 רישא של 10 $\Leftarrow$ נוסיף את 0 נקבל $S_2 = \{0, 01, 10, 1, 0\}$
- 0 נמצא במילות הקוד המקוריות $(C) \Leftarrow$ הקוד אינו UD

3.2 קוד אונארי $Unary$

- קוד קבוע מראש. עבור א"ב סופי או אינסופי מילת הקוד באינדקס i תהי $x_i = 1^{1-i}0$
- הסיבית 0 משמשת כהפרדה בין מילות קוד
- דוגמאות:

$$\begin{array}{ll} word\ 1 & w_1 = '0' \\ word\ 2 & w_2 = '10' \end{array}$$

- אם ידוע לנו שהא"ב סופי להוריד את ה 0 במילת הקוד האחרונה ואז נקבל עץ מלא- קוד שלם
  - לא יפריע כי לאחר קריאת 1^{n-1} בתו ה n נוכל לבדוק אם הוא 0 זוx_{n-1}אחרת $x_n = 1^{n-1}$
  - נקבל ש $K(C) = 1$
- מתי הקוד בעל יתרות מינמלית? פורמלית - מתי האונארי הוא קוד יעיל כך שכל קוד C' יקיים ש: $E(C_{unary}, P) \le E(C', P)$

x	c_i	$\lvert c_i \rvert$	p_i
1	0	1	$\frac{1}{2}$
2	10	2	$\frac{1}{4}$
$\vdots$			
$n-1$	1..10		
n	$\underbrace{1...1}_{n-1}$		$\left(\frac{1}{2}\right)^n$

כאשר ההסתברויות הן חזקות של $\frac{1}{2}$ אנחנו מקבלים שהקוד האינסופי אופטימלי/בעל יתרות מיני'

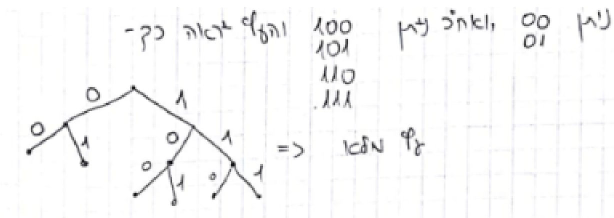
3.3 קוד בינארי

3.3.1 $Simple\ binary\ ocde$

- עושים קוד בעל אורך קבוע, כמו ב $ascii$.
- עבור א"ב בגודל n , ניתן מילות קוד באורך $\lceil \log_2(n) \rceil$
- נשים לב שאם יש לנו $n = 2^k$ (חזקה של 2) אז נקבל שאורך מילות הקוד הוא $\lceil \log_2(n) \rceil = \lceil \log_2(2^k) \rceil = k$

3.3.2 $Minimal\ binary\ ocde$

- לא מחייבים קוד בעל אורך קבוע
- נעשה שחלק ממילות הקוד יהיו באורך $\lceil\log_2(n)\rceil$
- השאר יהיו באורך $\lceil\log_2(n)\rceil-1$
  - כלומר נאפשר מקסימום הפרש של סיבית אחת בין מילות הקוד הקצרות למילות הקוד הארוכות
- בחירת המילים בעזרת דוגמה - נניח $n=6$ אז $\log_2(n)=2.6$:
  - יהיה לנו $2^{\lceil\log_2(n)\rceil}-n=2^3-6=2$ באורך $\lfloor\log_2(n)\rfloor=2$
  - ו $2n-2^{\lceil\log_2(n)\rceil}=12-8=4$, באורך $\lceil\log_2(n)\rceil=3$
  - נבנה עץ מלא:



- נניח n חזקה של 2 , מתי הקוד אופטימלי (MRC)?
  - נרצה שכמות האינפורציה תהיה שווה למספר הביטים שמקצים בקוד לכל מילת קוד
  - אם $n=2^k$ אז בשביל כמות אינפורציה אופטימלית צריך שהאינפורציה תהיה k , וההסתברות $p_i=\frac{1}{2^n}$ (אחידה) ונקצה k סיביות לכל מילת קוד

3.4 קוד $Elias$ - לתקן

- מעין מיזוג של בינארי ואונארי
  - האורך של מילת הקוד ה x יהיה $O(\log_2 x)$

3.4.1 C_γ

- בחלק הראשון כותבים **בקוד אונארי את מס' הסיביות בייצוג הבינארי של** x - יקח לנו $1+\log_2(x)$
- בחלק השני כותבים את **הייצוג הבינארי של** x **, ללא ה** 1 **המוביל** - יקח לנו $\lfloor\log_2(x)\rfloor \Leftarrow \lfloor\log_2(x)\rfloor$ $\left|x_{c_\gamma}\right|=1+2\cdot\lfloor\log_2(x)\rfloor$

  1. ניצג את x בבינארי , $w=x_2$
  2. נסמן $out1=|w|_1$
  3. נסמן $out2=w-(first\ 1)$
  4. נפלוט: $out1\cdot out2$ (שרשור)

- דוגמה:

$$25_2 \overset{1}{=} 11001 \Rightarrow |25_2|=5 \Rightarrow 5_1 \overset{2}{=} 11110$$
$$out2 \overset{3}{=} \cancel{1}1001$$
$$\overset{4}{\Rightarrow} 25_{c_\gamma}=\underbrace{11110}_{out1}\underbrace{1001}_{out2}$$

3.4.2 C_δ

- בחלק הראשון כותבים את **מס' הביטים בייצוג הבינארי של** x **, על ידי** C_γ . יקח לנו : $1+2\cdot\lfloor\log_2(\log_2(2x))\rfloor$
- בחלק השני כותבים את הייצוג **הבינארי** של x , ללא ה 1 המוביל - יקח לנו $\lfloor\log_2(x)\rfloor$
- סה"כ $\Leftarrow$ $1+2\cdot\lfloor\log_2(\log_2(2x))\rfloor+\lfloor\log_2(x)\rfloor$ ביטים

1. ניצג את x בבינארי , $w=x_2$
2. נסמן $out1=|w|_{c_\gamma}$
3. נסמן $out2=w-(first\ 1)$
4. נפלוט: $out1\cdot out2$ (שרשור)

- דוגמה:

$$25_2\overset{1}{=}11001\Rightarrow|25_2|=5\Rightarrow|5|_{c_\gamma}\overset{2}{=}\underbrace{110}\underbrace{01}=out1$$
$$out2\overset{3}{=}\cancel{1}1001$$
$$\Rightarrow25_{c_\delta}=\underbrace{11001}_{out1}\underbrace{1001}_{out2}$$

- אם נמשיך ל C_ε לא ייעל - גם כאן השיפור רק במספרים גדולים
- מתי C_γ עדיף על C_δ ? כאשר היצוג האונארי יותר קטן מ C_γ , מקיים עובר אורכים $x=2,4$ ולכן C_δ ארוך יותר במילים $X=2,3,8...15$ ובשאר C_δ עדיף

קוד $Golomb$

- קוד עם "דלי" בגדול קבוע b- כלומר $fiexed\ size$
- כל ה"דליים יהיו בגודל b
- קוד - x האינדקס של מילת הקוד שרוצים לקודד. b הגדול של <u>הקבוע</u> של הדלי

```
Golom_encode(x,b):
    q = (x-1)÷ b;
    r = x-q*b;
    p1 = Unary_encode(q+1)
    p2 = Minimal_binary_endoce(r,b)
    return p1·p2
```

q החלק השלם , r השארית

נרצה לקודד את $x=8$ והדלי בגודל $b=5$

- $q=(8-1)\div5=1$
- $r=8-1\cdot5=3$
- $unary\,(q+1)=unary\,(2)=10$
- $MBE\,(3,5)=10$

– הסבר : $\begin{pmatrix} 0 \\ 01 \\ 10 \\ 110 \\ 111 \end{pmatrix} \rightarrow$ מילת הקוד השלישית בא"ב שגודלו 5 - לשים לב למילים ש MBE בוחר

- $C_8 = 1010$

מפענח:

- b גודל הדלי, נעשה -1 משום שבקידוד עשינו $+1$

```
Golomob_decode(b) {
    q = Unary_decode() - 1
    r = Minimal_binary_decode(b)
    return r+q*b;
```

x	Golomb code b=5	Rice K=2
1	0 00	0 00
2	0 01	0 01
3	0 10	0 10
4	0 110	0 11
5	0 111	10 00
6	10 00	10 01
7	10 01	10 10
8	10 10	10 11
9	10 110	110 00

•Rice code v
Golomb cod
•How would
unary code
codes?

דוגמה: נניח קראנו $x = 1001$

- נקרא את החלק האונארי (נדע לפי האפס) $\Leftarrow 10 \Leftarrow q = 2 - 1 = 1$
- נקרא את השאר: $01 \Leftarrow$ המילה השניה בא"ב מגודל $5 \Leftarrow r = 2$
- סה"כ $x = 2 + 5 \cdot 1 = 7$

3.4.3 קוד $Rice$

- זה מקרה פרטי של $golomb$, שהוא מתייחס ל"דליים" שהם בגודל של חזקה של $2 \leftarrow b = 2^k$ (עבור $k \in \mathbb{N}$)
- אם גודל הדלי הוא חזקה של 2 , ניתן להגיד על החלק השני של ה MBE, שהוא מתלכד עם ה $Simple\ Binary$
- בחלק הראשון פשוט נעשה $shift\ right$ ל $(x-1)$ ב k ביטים, ומוסיפים 1 (חיבור), ואז מקודדים באונארי
- דוגמה:

– לפי $Golomb$:

$$x = 8 \qquad\qquad b = 4 = 2^2$$
$$q = \tfrac{8-1}{4} = 1$$
$$r = 8 - 1 \cdot 4$$
$$Unary\,(1+1) = 2 = 10$$
$$MBE\,(4,4) = 11$$
$$x_{Gol-4} = 1011$$

– לפי $Rice$:

* החלק האונארי:

$$x = 8 \xrightarrow{(-1)} 7 \xrightarrow{7_2} 111 \overset{\text{SR of k=2 bit}}{\longrightarrow} 1\ \cancel{1}\ \cancel{1} \xrightarrow{\text{add 1}} 2 \xrightarrow{\text{Unary}} 10$$

* בחלק של ה MB

· במקום לעשות קידוד בינארי, ניקח את $k = 2$ הביטים הנמוכים ביותר של $x - 1$ - למעשה אותם ביטים ש"זרקנו" ב SR בחלק של האונרי ← 11 ⇐ התוצאה זה השרשור שלהם

- עוד דוגמה - להשלים

3.5 קוד $shannon$

3.5.1 קידוד של עץ בינארי

- כדי להעביר את הקוד למפענח, לא תמיד נצטרך להעביר את ההסתברויות אפשר להעביר את העץ עצמו.

  – מניחים שיש עץ עם n - n גודל הא"ב

    * נייצג ב1 כל צומת פנימי וב 0 את העלים

- נשים לב שבעץ מלא עם n עלים יהיו לנו $n-1$ צמתים פנימיים

- מספר הביטים שנצטרך כדי לקוד עץ בינארי עם n עלים הוא $2n-1$ ביטים $= 2n-1 \quad \underbrace{n}_{\text{leaves}} + \underbrace{n-1}_{\text{interior nodes}}$

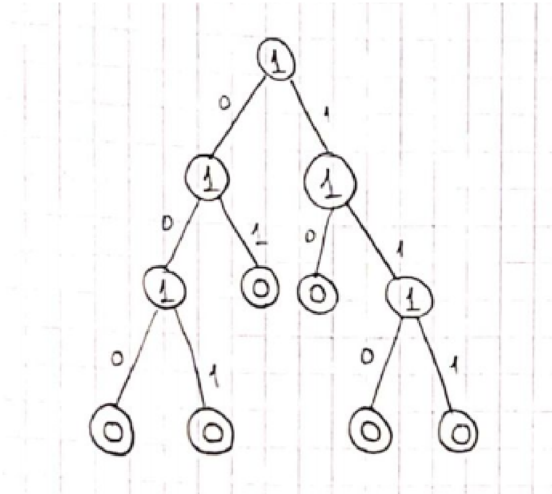
- המפענח יעבוד בצורה הבאה:

  – 8 סיביות לגודל הא"ב נסמנו ב n

  – תיאור הא"ב ב$ascii$ = $8 \cdot n$

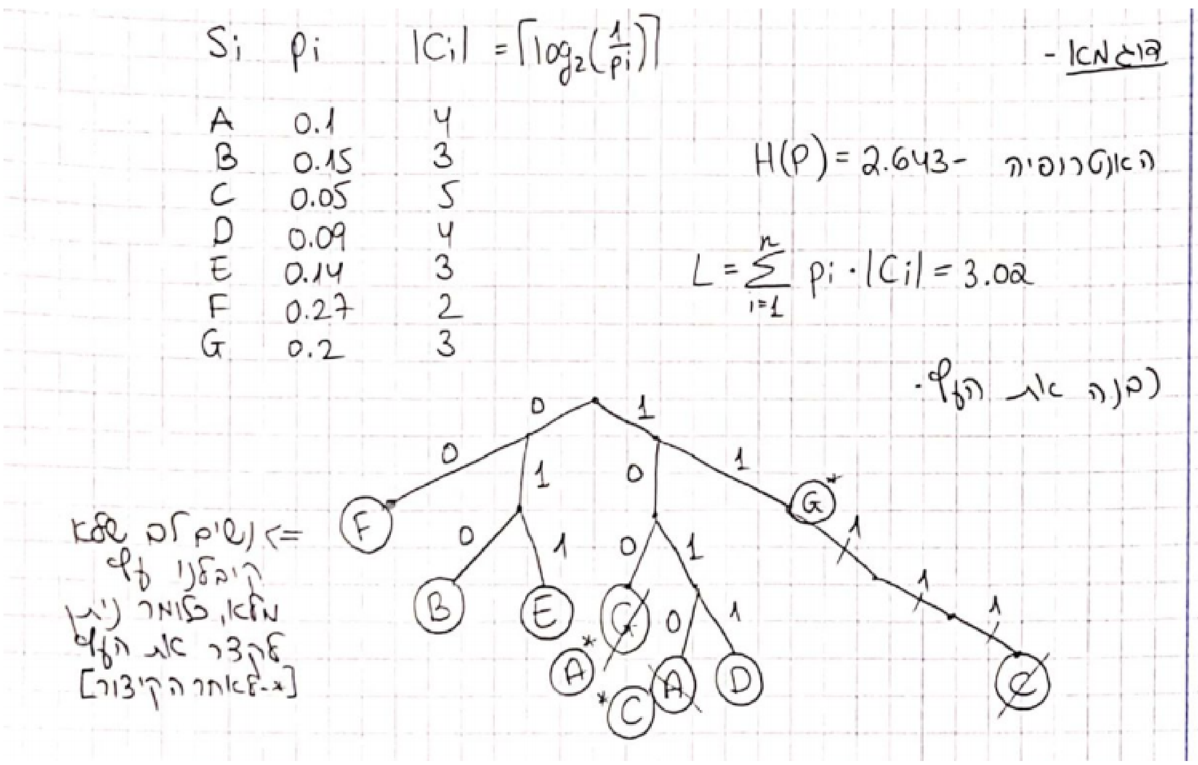
  – נניח שהא"ב מגיע משמאל לימין לפי העלים ונרצה לפענח את המחרוזת 11110010000

  – המפענח, יגדיר רמות:$\underbrace{1}_{\text{root}}\ \underbrace{11}_{\text{level 1}}\ \underbrace{1001}_{\text{leve 2}}\ \underbrace{0000}_{\text{level 3}}$



3.5.2 קוד $Shannon$

- נרצה למצוא קוד בעל יתירות מינמלית ולהתקרב לאנטרופיה כמה שיותר (החסם התחתון) , לכן הוא עיגל למספר הקרוב ביותר (הצעה ראשונה)
- עבור תו c_i והסתברות p_i אורך מילת הקוד של c_i יהיה $= \left\lceil \log\left(\frac{1}{p_i}\right) \right\rceil$, ונבנה את הקוד בעזרת עץ עם האורכים שקיבלנו



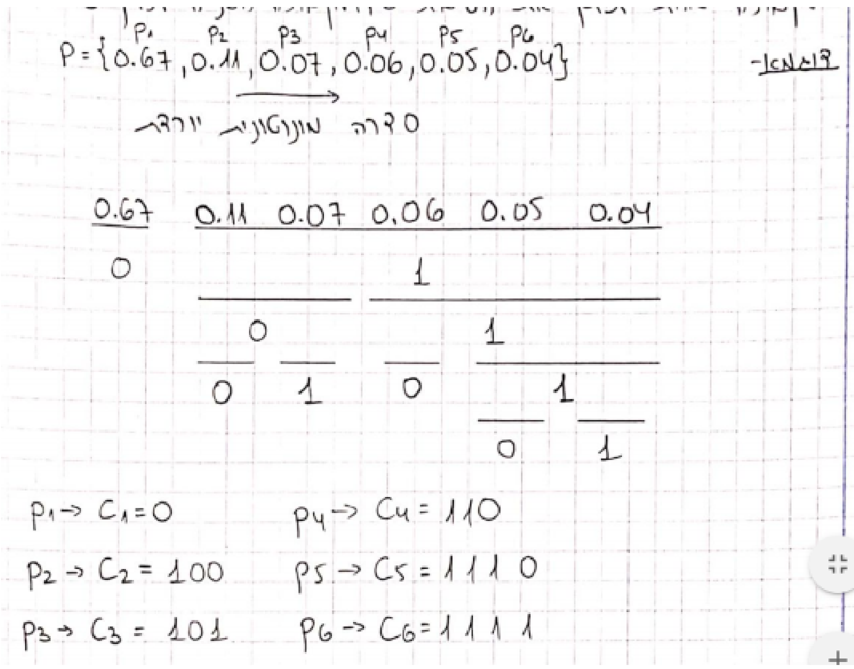
- נשים לב שאם היינו רוצים $fixed\ length$, האורך היה 3 עבור 7 מילות קוד ואז ה $E(C,P) = 3$ שזה כבר יותר טוב מההצעה הראשונה של שאנון

3.6 $shannon - Fano$

- קוד $shannon$ לא כל כך יעיל. לכן שאנון עשה קוד יעיל יותר:

- מיין את התווים לפי הסתברויות בסדרה מונטומית יורדת
- על עוד בקבוצת ההסתברויות יש יותר מאיבר אחד
  - חלק את הקבוצה לשני חלקם פחות או יותר שוים על פי ההסבתרויות
  - לקבוצה אחת תן את הסיבית 0
  - לקבוצה השניה תן סיבית 1

- דוגמה: $P = \{0.67, 0.11, 0.07, 0.06, 0.05, 0.04\}$



- נשים לב שכל פעם שפיצלנו ־ פיצלנו ל2 , לכן נקבל עץ מלא, כלומר קוד שלם
- הקוד $shannon - fano$ לא תמיד אופטימלי
- המפענח ישלח:
  - את הא"ב לפי סדר העלים ב$ascii, C_\delta$
  - ואת העץ

3.7 קוד הפמן

הפמן בנה את הקוד הפוך. מלמעלה למטה. בצורה הבאה:

- מיין את ההסתברויות בסדרה מונטונית יורדת
- כל עוד יש יותר מאיבר אחד בקבצה:
  - קח את שתי ההסתברויות הנמוכות ביותר
  - אחד אתם לאב משותף, לאחד תן 0 , לשני תן 1

הקידוד של כל מילה יהיה המסלול מהשורש למילה

דוגמה - עם טבלה + ציור העץ

נשים לב :

- קיבלנו קוד שלם (עץ מלא - חיברנו שתיים בכל שלב)
- קיבלנו קוד פרפיקסי
- קיבלנו של $K(C)=1$

הקוד:

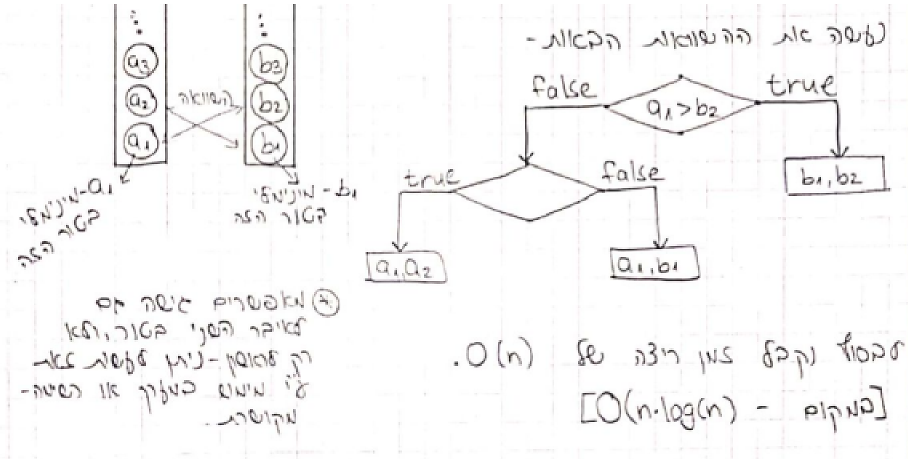
```
huffman (∑):
    n = |∑|
    Q1 = ∑ // min heap (priority queue)
    for i = 1 to n-1:
        do ALLOCATE-NODE(z) // new node
        z.left = x = Extract_min(Q1)
        z.right =y = Extract_min(Q1)
        z.weight = x.weight + y.weight
        Insert(Q1,z)
    return Exract-min(Q1) //
```

זמן ריצה:

- יצירת ערמה ראשונית $O(n)$
- רצים על לולאה $n-1$ פעמים
  - הקצאה של צומת $O(\log n)$
  - הוצאה של מינימלי מערימה $O(\log n)$
  - השמה לבן של z - $O(\log(n))$
  - הכנסה של z לערימה $O(\log n)$
- סך הכל $O(n\cdot\log n)$

3.7.1 האפמן שני תורים:

- עבור שכיחויות נתונות בסדרה ממונית , ניתן ליעל
- נשים את השכיחויות המקרויות בטור/ערמה אחד
- נשים את הסכומים בטור שני
- בכל פעם לבחור את שני המינימלי



3.7.2 קוד הפמן קנוני

- לוקחים את האורכים של מילת הקוד שמקבלים מהפמן
- מסדרים בעץ כך שאם נעבור על העץ משמאל לימין מילות הקוד מתקצרות, כלומר:
  - בצד שמאל מילות הקוד יהיו הארוכות ביותר
  - בצד ימין הקצרות ביותר
- למעשה זהו סידור לפי גודל המילים
- העץ שנקבל יהיה עץ הפמן קנוני

דוגמא -

עבור הפמן קיבלנו -

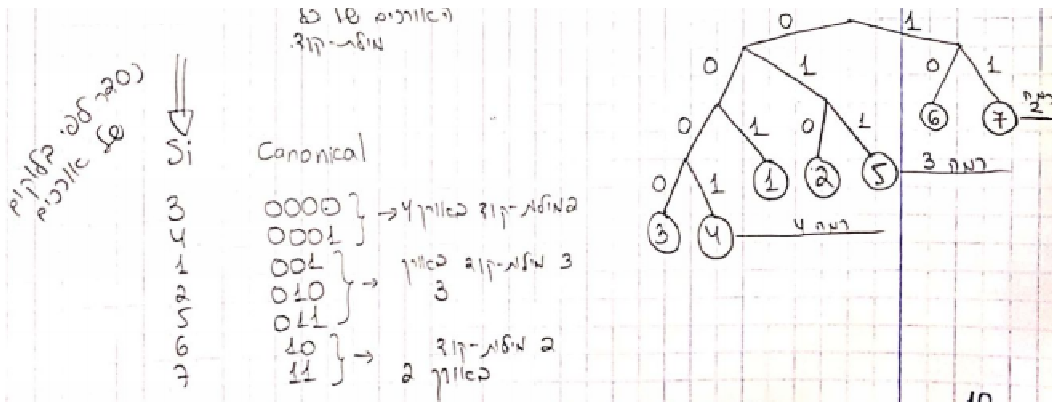
Si	pi	Code	length
1	0.1	100	3
2	0.15	001	3
3	0.05	0000	4
4	0.09	0001	4
5	0.14	101	3
6	0.27	01	2
7	0.2	11	2

$\Leftarrow$

נמיר להפמן קנוני

Si	pi	Canonical	length
1	0.1	001	3
2	0.15	010	3
3	0.05	0000	4
4	0.09	0001	4
5	0.14	011	3
6	0.27	10	2
7	0.2	11	2

נשים לב ששמרנו על כל האורכים של מילות-קוד



- למעשה בהפמן קנוני יוצרים בלוקים לפי האורכים של מילות הקוד
- ומקבלים שבכל בלוק של אורך כלשהו הבלוק מכיל מספרים בינארים עוקבים
- <u>כעת נוכל להעביר למפענח את הקוד בצורה מקוצרת:</u>
  - א"ב חייב לעבור בכל מקרה, אך נעביר אותו על פי סדר העץ משמאל לימין
  - ניתן מילת הקוד הראשונה בכל בלוק + התו המתאים
- נמשיך את הדוגמה:

S_i	canonical
3	0000
1	001
6	10

  - את הטבלה נעבר קודם את התו לפי *ascii* - 8 סיביות
  - טבלת מילות הקוד:
    - אורך מילת הקוד - אפשר ב C_φ
    - מילת הקוד עצמה

3.7.3 אלגוריתם הפמן קנוני

```
canonical_huffman();

    max_len = max{l_i}
                                        // 1. מציאת מספר מילות הקוד בכל אורך //
    for l = 1 to max_len
        num[l] = 0
    for i =1 to n
        num[i]++
                                        // 2. הכנסה של מילת־הקוד הראשונה //
    firstcode[max_len] = 0
    for l = max_len downto 1:
        firstcode[l] = (firstcode[l+1]+ num[l+1] )/ 2
                                        // 3. הכנסה של מילת־הקוד הראשונה //
    for l =1 to max_len:
```

```
        nextcode[l] = firstcode[l]
    for i =1 to n:
        codeword[i] = nextcode[l_i ?]
        symbol[l_i ,nexcode[l_i]-firstcode[l_i] = i
        nextcode[l_i]++
```

- שלב 1 : num אורכי מילה לפי מיקום בתא , כלומר $num[3]=4$ ישנן 4 מילים באורך 3
- שלב 2 : $firstcode$- מילת הקוד הראשונה בכל בלוק (נחשב לפי האורך והערך המספרי באותו האורך)
- שלב 3 : $nextcode$ - מילת הקוד הבאה עבור כל אורך ובלוק

3.7.4 אלגוריתם לפענוח

```
v = nextInputBit()
l = 1
while v < firstcode[l] do:
    v = 2v + nextInputBit()
    l++
```

המספר v הוא מילת הקוד באורך l סיביות

```
return symbol[l,v-firstcode[l]
```

3.7.5 קוד הפמן $D-ary$

- אחת מהכללות להפמן - נכליל את המשפטים: בעץ אופטימלי:
  - לפחות 2 צמתים נמצאים ברמה התחתונה
  - 2 הצמתים עם המשקולות המינמלים הם הרמה התחתונה
  - ניתן להניח ש 2 המשקולות המינמלים הם אחים
- עד כה דיברנו על הפמן בינארי - לכל צומת 2 בנים
- בהכלל זו יכולים להיות יותר מ2 בנים, נסמן את מספר הבנים ב D

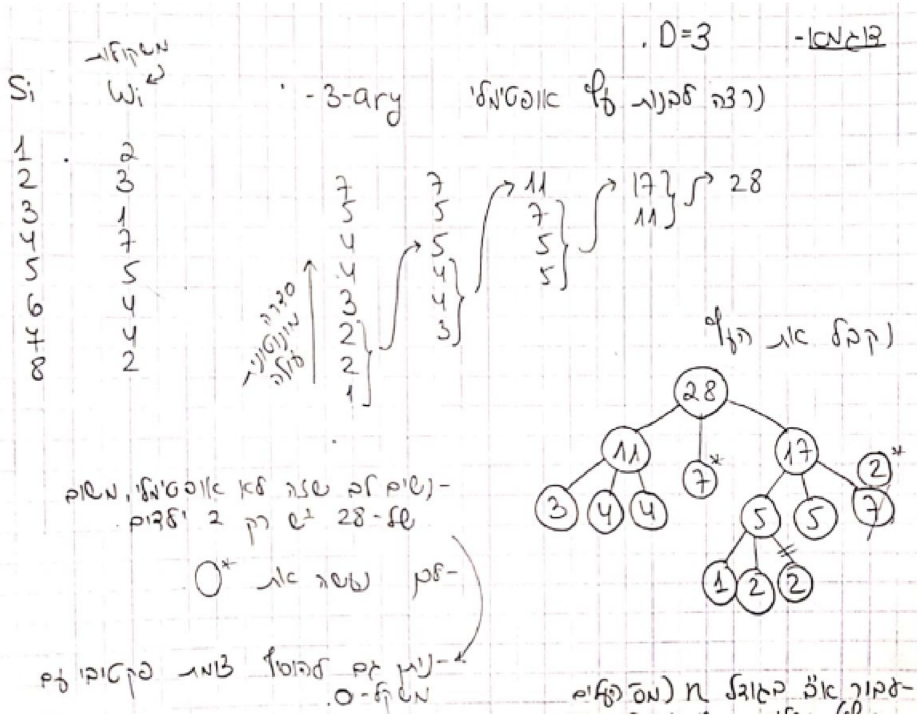
אלגוריתם:

- חשב מה מספר העלים n_0 שצריך להוסיף כך ש: $n+n_0=(D-1)\,n'+1$ ותן להם את הערך 0
- הפעל הפמן , עבור D עלים בכל שלב
- תן ערך $0,..D-1$ לכל קשת
- התאם עלה להסתברות שלו (וזרוק את העלים עם ה0

הסבר:

- הרעיון שמילות הקוד שנזרוק יהיו הכי "יקרות", ובכך לא בזבנו מילים "זולות".

- הערה: הערך שנותנים לקשת בטוח $[0, D-1]$ הוא חשוב (כי ישנו תרגום נוסף לבינארי) ויש לודא פרטנית מה הערך המועדף



- צורך להוסיף צמתים פקטיבים
- בעץ $D-ary$ בעל n צמתים ישנם $(D-1)\cdot n+1$ עלים

3.7.6 הפמן דינאמי

- מעוניינים לקודד את התו שנמצא במקום t על סמך התווים שנמצאים ב $t-1$ המקומות הראשוניים
- היתרון זה שלאט לאט לומדים את שכיחויות של הקובץ, תוך כדי מעבר על הקובץ
- מתחיל מהתפלגות אחידה
- כל פעם שרואים תו a חדש נקדם את השכיחות של a(אם נקדם נקדם אותו מספיק ניתן לו מילת קוד קצרה יותר = נעלה אותו בעץ)
- אין צורך להעביר את העץ לצד השני ⇐ אין צורך ב $prelude$
- במקה של הפמן דינאמי זה הרבה פחות יעיל, לכן לא באמת משתמשים
- צומת 0 לתו החדש ????
  - כאשר לא רוצים לשלוח ב $prelude$ את הא"ב אז נצטרך להשתמש בצומת 0 לתוים חדשים שניתקל בהם בכל פעם
  - צומת 0 תקרא NYT (not yet transmoted) שמשקלו יהיה 0
  - בכל פעם שנתקלים בתו הפעם הראשונה נעבר את מילת הקוד ל NYT ואחריה את ה $ascii$ של התו
  - לאחר מכן נקצה לתו עלה בעץ ונעדכן את כל העץ
- אלגוריתם:

algo

```
q = leaf(x_t)
                                         // מקרה שנתקלים בתו בפעם הראשונה //
if (q  is the 0-node)
    replace q by a parent 0-node with two 0-node children
    q = left child;
if (q is a sibling of a 0-node)
    interchange q with the highest numbered leaf of the same weight
    incremment q’s weight by 1;
    q = parent of q;
                                     // עולים עד השורש, ומעדכנים את כל העץ //
whil (q!=root)
    interchange q with the highest numbered leaf of the same weight
    incremment q’s weight by 1;
    q = parent of q;
increment q’s weight by 1
```

3.7.7 הגדרה שקולה לעצי הפמן קנוניים

- ראינו עץ הפמן קנוני שמשוך שמאלה, כלומר מילת הקו הארוכה ביותר הייתה רצף של אפסים
- ניתן גם לבנות עץ הפמן קנוני משוך ימינה, ואז מילת הקוד הארוכה ביותר היא רצף של אחדות
- באלגוריתם הזה כמו בקודם, ראשית נתחיל מהאפמן, כדי לקבל את הארוכים של מיולת הוקד
  - נמיין אותם מונוטונית עולה (את האורכים)
  - לכל איבר ניתן l_i בסדרה מונוטנית עולה (את האורכים)
  - לכל איבר ניתן l_i סיביות, אבל בהתאם ש l_i מספרים אחרי הנקודה בטור : $\sum_{i=1}^{i-1} 2^{l_i}$
- לדגומה - נניח שקיבלנו מהפמן את האורכים כמו בטבלה :

דוגמא - נניח שקיבלנו מהפמן את האורכים כמו בטבלה -

i	S_i	l_i	2^{-l_i}	$\sum_{j=1}^{i-1} 2^{-l_j}$
1	B	2	0.01	0.00000
2	D	2	0.01	0.01000
3	A	3	0.001	0.10000
4	E	3	0.001	0.10100
5	F	3	0.001	0.11000
6	C	4	0.0001	0.11100
7	G	4	0.0001	0.11110

סכום ההסתברויות

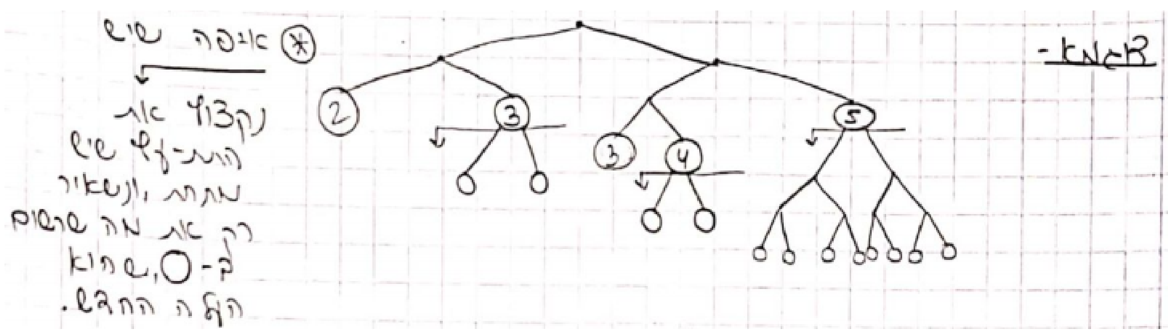
3.7.8 Skeleton Trees עצי שלד

מוטביציה: כאשר קראנו חלק ממילת והגענו לבלוק שבו כל אורכי המלים הוא באורך קבוע מסוים אז נדע ישר כמה תווים נשאר לנו לקרוא, ונוכל לקרוא אותם בבת אח

3.7.9 skeletion Huffman trees

אלגוריתם:

- ניקח עץ קנוני לפי ההגדרה השקולה (שהעץ משוך ימינה)
- וניקח תת עצים <u>שלמים</u> שגובהם ≤ 1
- נהפוך את השורש של התת־עץ הזה **לעלה**
- נשים בתוכן של העלה את אורך מילות הקודד הכולל (מהשורש עד העלים) של מילות־הקוד בתת עץ זה
- דוגמה:



- למפענח נשלח את הא"ב לפי הסדר של העלים בעץ המקורי (לפני הקיצוץ)
  - (בכל מקרה בקנוני צריך לשלוח את הא"ב לפי סדר העלים משמאל לימין)

פענוח:

- נגדיר:
  - n_i ־ מס' מילות הקוד בבלוק ה i , בלוק i אומר מילות קוד באורך i
  - m ־ נצטרך לדעת מאיפה להתחיל, לכן נצטרך לדעת מה אורך־מילות הקוד המינמלי לכן נגדיר:

$$m = \underbrace{\min}_{\text{min block}} \left\{ i \mid \underbrace{n_i > 0}_{\text{blocks that contain words}} \right\}$$

  - $base(i)$ ־ מתייחס למילת ־ קוד הראשונה בבבלוק ה i (כמו ה $firstcode()$ שהיה בקנוני) וזה יהיה הערך <u>העשרוני</u> של מילת הקוד
  - $B_s(k)$ ־ מסתכלים על רצף בינארי עם s־סיביות של שלם k , מרפדים באפסים אם צריך $(padding)$
  - מילות הקוד ה j עם אורך i : $j = 0,1_{100},n_i-1$: jירוץ על כל מילות הקוד בכל בלוק.
    - * מכיון שמדובר על מספרים בינאריים נוקבים, מספיק לדעת מה מילת הוקד הראשונה בבלוק.

$$B_i\left(\underbrace{base(i)}_{\text{first word at i'th block}} + \underbrace{j}_{\text{block offset}}\right)$$

- $seq\,(i)$ ־ האינדקס של מילת הקוד הראשונה בבלוק ה i .
  * תמיד מתקיים ש : $seq\,(m) = 0$
  * $seq\,(i) = seq\,(i-1) + n - 1$

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

- הערה: נשים לב שאם נחסר את הייצוג הבינארי של מילת־קוד נוכחית עם הייצוג הבינארי של המילת־קוד הראשונה בבלוק, נקבל את ההיסט של מילה באותו הבלוק, משום שאנו שאנו מדברים על מספרים בינאריים עוקבים. כלומר:

$$x_2 - (\text{first word at x block}) = \text{x'th offeset}$$

- $I\,(w)$: הערך המספרי של מחרוזת בינארית w . אם w באורך l 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: $I\,(w) - diff\,(l)$
  - כאשר $dif\,(l) = base(l) - seq(l)$
- $value(v) = 0$:אם v הוא צומת פניני של העץ.
  - אחרת, נותנים את האורך שך מילת־הקוד ומקצצים עצים שלמים
- $table\,(j)\,,\ \text{for}\ 1 \le j \le n$ לכל מילת קוד j , ילך מ 1 עד n , ויש תרוגם שאומר מה הא"ב הזה

אלגוריתתם פענוח של $skeleton$:

```
decode:

    tree_pointer = root
    i = 1
    start = i
    while i < length_of_string // len of encoded string
        if string[i] = 0
            tree_pointer = left(tree_pointer)
        else
            tree_pointer = right(tree_pointer)
        if value(tree_pointer) > 0
            codeword = string [start...(start+value(tree_pointer) -1]
            output = table[I(codeword)-diff[value(tree_pointer)]]
            tree_pointer = root
            start = start + value(tree_pointer)
            i = start
        else
            i++
```

3.7.10 reduced skeleton tree

- עבור עץ שלד אמרנו שנקצץ תתי־עצים שלמים
- אם נרצה לקצץ עוד, נוכל לקצץ איפה שההפרש בין מילות־הקוד מהשורש הוא לכל היותר 1 בין המילות־קוד הארוכות לקצרות
  - כלומר, בכל מקום שיש תת־עץ בו יש מילות קוד באורך l או $l+1$ השורש של תת העץ הזה עלה עץ בעץ שלד מוקטן
- דגוגמה

פענוח

- לא נוכל לפענח כמו ב $skeleton$, נצטרך להוסיף עוד בדיקה
- עבור עץ שלד מוקטן, אנו יודעים "בערך כמה עוד נשאר לנו לקרוא, **אבל** זה לא ודאי כמו ב $skeleton$, פה אנחנו רק ידועים שהאורים עם הפרש 1
- מה נעשה?
  - נוכל לקרוא עוד כמה תווים בודדים ואז להבדיל
  - או לקחת את הערך של עד איפה שקראנו ולהשוות עם האינדקסים של הבלוקים, ונוכל להחליט האם זה מילת־קוד באורך l או $l+1$
- בשביל האלגוריתם נגדיר:
  - $lower(v), upper(v)$ לכל צומת v, נבדוק:
    - אם v עלה: ־ אנחנו נותנים ל $lower\left(v\right), upper(v)$ את הערך של $value(v)$ ($value\left(v\right)$ היה אורך מילת־הקוד)
    - אם v צומת פנימי:
      $$lower\left(v\right) = lower\left(left\left(v\right)\right)$$
      $$upper\left(v\right) = upper\left(right\left(v\right)\right)$$
  - עבור $lower$ נלך שמאלה, עבור $upper$ נלך ימינה. <u>משום שאמרנו שאנו מדברים על עץ קנוני משוך ימינה</u>
- ה reduced skeleton הוא תת העץ הקטן ביותר בתוך ה $skeleton$ שההפרש בין ה $upper$ וה $lower$ הוא לכל היותר 1
- אם ההפרש שווה ל 0 , זה אומר שכל תת העץ שלם, וב$skeleton$ רגיל הוא היה עלה
- דוגמה
- אלגוריתם פענוח

```
decode reduced:
    tree_pointer = root
    i = 1 = start
    while i < length_of_string // len of encoded string
        if string[i] = 0
            tree_pointer = left(tree_pointer)
        else
            tree_pointer = right(tree_pointer)
        if value(tree_pointer) > 0
```

```
    len = value(tree_pointer)
    codeword = string [start...(start + len -1]
    if flag(tree_pointer) = 1 && 2*I(codeword)≥base(len+1)
      codeword = string [start...(start + len]
      len++
    output = table[I(codeword)-diff[len]]
    tree_pointer = root
    start = start + len
    i = start
else
    i++
```

3.8 קוד אריתמטי

- קוד אריתמטי נייצג את כל הטקסט הגדול במספר בודד אחד
- בקוד אריתמטי אדפיטבי לא צרכיים לשלוח למפענח את הא"ב וההתפלוגיות - המפענח קורא ותוך כדי מעבד את הנתונים
- בקוד ארימתטי נוכל להשתמש ב"שברי ביטים" ⇐ נותן את האנטרופיה

3.8.1 קידוד ארימתטי סטטי

- הרעיון:
  - מתחילים מ$intrval$ חצי פתוח $[0,1)$
    * נחלק את $[0,1)$ בצורה פרופצינואלית לפי ההסתבוריות של אותו תו ⇐ נעדכן את ה $interval$ כך: $[low, high)$
    * נעשה מעין $zoom - in$ ונחזור על התהליך
- שולחים $interval$ ים אחד אחרי השני, לפי הא"ב וההסתבוריות:

Si	Pi		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

כל ה-interval-ים האלה - הם חצי פתוחים - [,).

דוגמא-

לוקחים את ה-interval ומחלקים אותו לתתי-interval-ים, שכל תת עם low-bound ו-high-bound שמתאים לו לפי ההסתברות.

אלגוריתם קידוד:

נגדיר:

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

```
Arithmetic_encode:

    low  = 0.0
    high = 1.0
    while input symbols remain {
        range = high - low
        get symbol
        hig = low + high_bound(symbol)*range
        low = low + low_bound(symbol)*range
    }
    output any value in [low,high)
```

דוגמה - המחרוזת $ABAAAEAABA$:

Encoding Example

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

$$\begin{array}{ccccc}
\underline{A} & & \underline{B} & & \underline{A} \\
range = high - low = 1 & & range = high - low = 0.67 & & \\
\boldsymbol{low} = low + low_bnd(A) * range & \Rightarrow & \boldsymbol{low} = low + low_bnd(A) * range & \Rightarrow & \ldots \\
= 0.0 + 0.0 * 1 = \mathbf{0.0} & & = 0.0 + 0.67 * 0.67 = \mathbf{0.4489} & & \\
\boldsymbol{high} = low + high_bnd(A) * range & & \boldsymbol{high} = low + high_bnd(A) * range & & \\
= 0.0 + 0.67 * 1 = \mathbf{0.67} & & = 0.0 + 0.78 * 67 = \mathbf{0.5229} & &
\end{array}$$

```
Arithmetic_decode(endoce_number n):
do {

    Find symbol whose range contains n
    Output the symbol
    range = high(symbol) - low(symbol)
    encoded =  (encoded - low(symbol))/range
```

דוגמה:

E = .109375
between 0.0, 0.25; output 'B'
E = (.109375 - 0.0) / 0.25 = .4375
.4375 between 0.25, 0.5; output 'I'
E = (.4375 - 0.25) / 0.25 = 0.75
0.75 between 0.5, 1.0; output 'L'
E = (0.75 - 0.5) / 0.5 = 0.5
0.5 between 0.5, 1.0; output 'L'
E = (0.5 - 0.5) / 0.5 = 0.0 -> STOP

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

- גודל הודעה היא $r = \prod_{i=1}^{k} p(m_i)$ ביצוג בינארי: $\lceil -\log_2(r) \rceil = \lceil \log_2\left(\frac{1}{r}\right) \rceil$
- המפענח ידע מתי לסיים:
  - המקודד ישלח לו מספר איטרציות
  - נגדיר תו מסוים ל EOF

3.8.2 קידוד אריתמטי אדפטיבי (דינאמי)

- בקוד אריתמטי אדפטיבי אין צורך ב $prelude$ של ההסתבוריות של הא"ב
- נסמן ב $N(x)$ את מספר המופעים של x עד לאותה נקודה שנמצאים
- נחשב את ההסתברויות של כל תו בצורה הבאה - עבור א"ב בגודל 3 - נניח $\{a,b,c\}$:

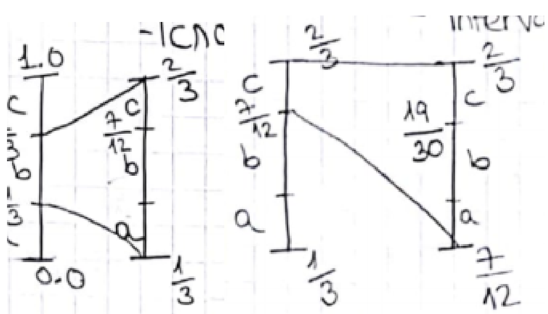
$$p(a) = \frac{N(a)+1}{N(a)+N(b)+N(c)+3}$$
$$p(b) = \frac{N(b)+1}{N(a)+N(b)+N(c)+3}$$
$$p(c) = \frac{N(c)+1}{N(a)+N(b)+N(c)+3}$$

בהתחלה $N(x) = 0$ - כלומר מתחילים מהסתברות אחידה

- דוגמה - חלקית: המחרוזת $T = bccb$ אז בגלל ש $N(x) = 0$ נקבל: $p(a) = p(b) = p(c) = \frac{1}{3}$
  - נקרא אות ראשונה b נתכנס לקטע $[\frac{1}{3}, \frac{2}{3})$ ונקבל ש:

$$p(a) = \frac{0+1}{0+1+0+3} = \frac{1}{4} \Rightarrow [0, \frac{1}{4})$$
$$p(b) = \frac{1+1}{0+1+0+3} = \frac{1}{2} \Rightarrow [\frac{1}{4}, \frac{3}{4})$$
$$p(c) = \frac{0+1}{0+1+0+3} = \frac{1}{4} \Rightarrow [\frac{3}{4}, 1)$$

  - האות הבאה הבאה c , לכן נתכנס לקטע $[\frac{1}{3} + \frac{1}{3} \cdot \frac{3}{4}, \frac{1}{3} + \frac{1}{3} \cdot 1) = [\frac{7}{12}, \frac{2}{3})$. (עדכון $low, high$ קלאסי)



– וכאן הלאה..

הערה:

- מהירות הקידוד לעומת מהירות הפענוח. נבדוק את הפעולות:
  - בקידוד: חיבור וכפל
  - בפענוח: חיסור וחילוק
    - חילוק איטי ביחס לכפל
- קיבלנו שהפענוח איטי ביחס לקידוד ואנחנו מעדפים להשקיע יותר בקידוד לקבל קובץ יותר קטן שהמפענח יעבוד פחות
- חסרונות:
  - עבור גודל קובץ $125kB$ צריך: $125KB \approx 2^7 \cdot 2^{10} \cdot 2^3 = 2^{20} \approx 10^6 b$ כלומר צריך בערך מיליון סיביות ורוב המחשבים לא תומכים בזה
  - א"א $stream$ כלומר לעבוד עם משהו תוך כדי ריצה (כמו במימוש הראשון של ארית?)⇐ חייבים את המספר כולו
    - (במימוש הראשון כן יכולנו להבין בשלבים את ההתכנסות שלנו ולהתחיל לשחרר מידע)
  - איטי ביחס להפמן (אם נשווה קודים אדפטיבים ארימתטי ינצח)
  - אין גישה ישירה = כל שינוי משפיע ⇐ משתמשים בהצפנות

ישום של קוד אריתמטי

- נסתכל על הקטעים $[0, 0.999)$ או $.1111$ בבינארית
- נניח ש$register$ בעל 5 ספרות אז: $low = 0 = 0000$ ו $high = 99999$ (ויתרנו גם על הנקודה)
- נסמן $range = 1$ כי $0.999... - 0.000... = 1$
  - high=low+high_bound(symbol)*range-1

1. אם הMSB תואם:
   (א) פלוט את הסיפרה התואמת על ידי הזזת כל הספרות שמאלה ב $high$ וב low
2. אחרת:
   (א) חשב מחדש את $high$ ואת ה low וחזור ל 1

Example:

	high	low	range	output
Initial state	99999	00000	100000	
Encode B (0.2-0.3)	29999	20000		
Shift out 2	99999	00000	100000	.2
Encode I (0.5-0.6)	59999	50000		.2
Shift out 5	99999	00000	100000	.25
Encode L (0.6-0.8)	79999	60000	20000	.25
Encode L (0.6-0.8)	75999	72000		.25
Shift out 7	59999	20000	40000	.257
Encode SPACE (0.0-0.1)	23999	20000		.257
Shift out 2	39999	00000	40000	.2572

Data Compression Course - Dana Shapira 5

Example (cont.):

	high	low	range	output
Encode G (0.4-0.5)	19999	16000		.2572
Shift out 1	99999	60000	40000	.25721
Encode A (0.1-0.2)	67999	64000		.25721
Shift out 6	79999	40000	40000	.257216
Encode T (0.9-1.0)	79999	76000		.257216
Shift out 7	99999	60000	40000	.2572167
Encode E (0.3-0.4)	75999	72000		.2572167
Shift out 7	59999	20000	40000	.25721677
Encode S (0.8-0.9)	55999	52000		.25721677
Shift out 5	59999	20000		.257216775
Shift out 2				.2572167752
Shift out 0				.25721677520

Data Compression Course - Dana Shapira

נשים לב לחישוב של $high, low$ עם ה $range$ <u>החדש</u>

נשים לב לסוף - ממשיך ב $shift$ +פליטה עד שחוזרים ל$range$ של 1

בעית $underflow$

- נמשיך עם הדוגמה של ה 5 אוגרים ונניח שקיבלנו $\begin{Bmatrix} low = 39999 \\ high = 4000 \end{Bmatrix}$ אז משלב זה כל מספר שיגיע יכנס ל$interval$ הזה ונכנס ללולאה אינסופית.
- במילים אחרות מה נעשה כאשר $high$ וה low קרובים מדי? אין לנו פתרון, אבל כן נמנע מראש. איך?
  - בכל שלב נגדיר underflow_counter=0
  - נסתכל בכל שלב על הסיפרה השניה אם ב<u>low יש 9</u> וב<u>$high$ </u>0 - יש חשש שנכנס ל$underflow$
  - נמחק את הספרות השניות $(0,9)$ בהתאמה) ונעדכן ++underflow_counter
  - וננסה שוב לפעול רגיל (נחשב מחדש את ה $low, high$ בעזרת ה $range$)
  - כאשר יתלכד נבדוק:
    * אם ה<u>$high$ התלכד עם low</u> נפלוט את הסיפרה שהתלכדה <u>בתוספת 0</u> כמספר ה$counter$
    * אם ה<u>low התלכד עם $high$</u> נפלוט את הסיפרה שהתלכדה <u>בתוספת 9</u> כמספר ה$counter$

יצוג מספר עשרוני בבינארי עבור $x \in [0,1)$

```
L= 0 ; H = 1 ; i = 1 ;
while x > L:
    if x <(L+H)/2
        b_i = 0
        H = (L+H) / 2
    else if x >= (L+H)/2
        b_i= 1
        L = (L+H)/2
    i = i + 1
end-while
b_j=0 for all j>=i //0's padding
```

דוגמה:

$x=\frac{5}{7}$	L	H	$\frac{L+H}{2}$	x big?	b_i
	0	1	$\frac{1}{2}$	$>$	1
	$\frac{1}{2}$	1	$\frac{3}{4}$	$<$	0
	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{5}{8}$	$>$	1
	$\frac{5}{8}$	$\frac{3}{4}$	$\frac{11}{16}$	$>$	1

$\Rightarrow \frac{5}{7} = 0.1011$

$scaling$

```
y = x; i = 0;
while y > 0:
    i++;
    if y <1/2
        b_i = 0
        y = 2y
    else if y >= (1)/2
        b_i= 1
        y = 2y-1
end-while
b_j=0 for all j>=i //0's padding
```

$$
\begin{array}{c|c|c|c|c}
\underline{y=\frac{1}{3}} & \underline{y=\frac{2}{3}} & \underline{y=\frac{1}{3}} & \underline{y=\frac{2}{3}} & \underline{y=\frac{2}{3}} \\
i=1 & i=2 & i=3 & i=2 & \\
< & > & < & > & \\
b_1=0 & b_1=1 & b_1=0 & b_1=1 & ...... \\
y=\frac{2}{3} & y=\frac{2\cdot 2}{3}-1=\frac{1}{3} & y=\frac{2}{3} & y=\frac{1}{3} &
\end{array}
$$

אינווריאנטה: $x=b_1,b_2....b_i+\frac{y}{2^i}$

בסיס: $i=0 \Leftarrow x=0+\frac{y}{2^0}=y$

צעד: נניח ל i ונוכיח ל $i+1$:

- אם $y<\frac{1}{2}$ אז מהגדרת האלגו' $b_{i+1}=0$ ונגדיר $y'=2y$
- ויתקיים ש:

$$b_1b_2...b_ib_{i+1}+\frac{y'}{2^{i+1}}=b_1b_2...b_i0+\frac{\cancel{2}y}{2^{\cancel{i+1}}}=b_1b_2...b_i+\frac{y}{2^i}=x$$

ניתן להתעלם מה 0 בסוף (לא משפיע על המספר)

- אם $y\geq\frac{1}{2}$ אז מהגדרת האלגו' $b_{i+1}=1$ ונגדיר $y'=2y-1$ ויתקיים ש:

$$b_1b_2...b_ib_{i+1}+\frac{y'}{2^{i+1}}=b_1b_2...b_i\underbrace{1}+\frac{2y-1}{2^{i+1}}=b_1b_2...b_i+\underbrace{\cancel{\frac{1}{2^{i+1}}}}_{\text{1 repr' and value}}+\frac{2y}{2^{i+1}}-\cancel{\frac{1}{2^{i+1}}}=b_1b_2...b_i+\frac{2y}{2^{i+1}}=x$$

למה למדנו את זה? בשביל:

3.8.3 incremental coding

- משתמש ברעיון של $scaling$
- פולטים רק כאשר ה $high,low$ מתלכדים

- מתמודד עם ה $underflow$

אלגוריתם נגדיר את המקטעים: חצי עליון : $[\frac{1}{2}, 1)$ ־ חצי אמצעי : $[\frac{1}{4}, \frac{3}{4})$ ־ חצי תחתון : $[0, \frac{1}{2})$

1. נבדוק האם $interval$ בעייתי
   - כלומר נבדוק אם ה$interval$ שלנו נופל בחצי עליון/אמצעי/תחתון
2. אם נופלים בתוך אחד החצאים הללו ־ נכנס לבדיקה $underfow$ (שלבים 4־6)
3. אחרת = לא נופלים על אחד החצאים האלו . אז:
   (א) נצא החוצה ונגיד שאין לנו סיביות להוציא ל $output$ ונמשיך לקריאת הסיבית הבאה.
4. אם נופלים בחצי תחתון ־ מוצאים 0 ובודקים האם $0 < \text{underflow\_counter}$ אם כן נעשה $1^{counter}$, ונכפיל את ה $inteval$ ב2
5. אם נופלים בחצי עליון ־ מוצאים 1 ובודקים האם $0 < counter$, אם כן נעשה $0^{counter}$ ונכפיל את ה $inteval$ ב 2 ונחסיר 1
6. אם נופלים בחצי אמצעי ־ נשמור את מספר הפעמים שאנו באו־ודאות ב$counter$, נוכפיל את ה$interval$ בצורה הבאה:
   (א) את ה low שקטן מ $\frac{1}{2}$ נכפיל ב 2
   (ב) את ה $high$שגדול נכפיל ב 2 ונחסיר 1

ארימתטי מול הפמן

- יתרונות:
  - מקבלים את האנטרופיה
  - ניתן להפוך לדינאמי בצורה קלה
- חסרונות
  - הזמן שלוקח לדחוס גבוה מהפמן
  - היעול של ארימתטי לעומת הפמן לא שווה את ההשקעה בדחיסה
- $Sychorniztion$ ־ העובדה שאם ביט הלך לאיבוד הארימתטי לא ניתן לשחזר את הקובץ המלא ובהפמן כן, לכאורה חסרון, הפכו ליתרון בעולם ההצפנות.

3.9 מילון סטטי מול אדפטיבי

- בשיטה הסטטית המילון נבנה לפני הקידוד , דורש ידע קודם על הקובץ שמקודדים (מילות קוד, הסתברויות וכו'
- בשיטה האדפטיבית המילון נבנה תוך כדי
- שיטיות נוספות
  - שיטה סטטסיטית ־ מנסים לחזות את התו סטטיסטית
  - שיטת ההחלפה ־ מחליפים מחרוזות במצביע לקטע מקודד

בהנתן מילון סטטי ־ומחרוזת איך נשבור אותה לבלוקים שנקודד?

נעיר שבמילון יש תווים בודדים ולכן באף שיטה לא נתקע

- ה$Greedy$ - נבדוק את התו הנוכחי אם ניתן לצרף לו עוד תו ועדיין יש רשומה במילון נעשה זאת, אחרת נתחיל מילה חדשה $\Leftarrow$ הקידוד לא בהכרח אופטימלי
- $longest\ fragment$ - נבדוק מה המחרוזת הכי ארוכה שניתן לעשות בכל שלב $\Leftarrow$ זמן ריצה אקס'
- $Min\ word$ - נבדוק מהי החלוקה הכי קטנה
- $Optimal$ - נבדוק מה הקובץ הכי שנקבל = מספר הסיבויות הנמוך ביותר

הגדרה (ניתוח אופטימלי למילון נתון):

בהנתן מילון D , פונקצית λ - שממפה מילה לקידוד, וטקסט T החלוקה $T = w_1.w_2...w_k$ כאשר $w_i \in D$ מקיימת $\sum\limits_{i=1}^{k} |\lambda(w_i)|$ מינימלי.

ברידוקציה לגרפים - מציאת המסלול הקצר בגרף עם משקלים - אפשר לראות שזה פולינומי (דיקטסרה)

3.10 $LZ77$

אדפטיבי

window	loog ahead buffer

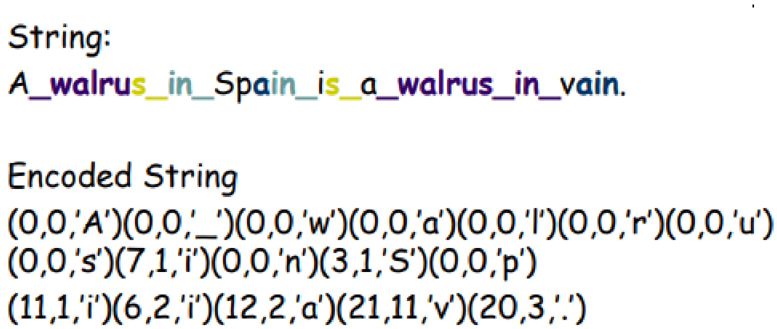
כל $pointer$ מיוצג כשלשה:

- $offset$ - יגיד כמה תוי צריך ללכת אחורה כדי להגיע למחרוזת קודמת שהיא כמו מחרוזת נוכחית
- $length$ - מה אורך המחרוזת המועתקת
- $symbol$ - תו נוסף שמוספים אחרי אותה מחרוזת חוזרת

אלגוריתם קידוד:

```
LZ77_encode:
    p=1 // next char to be coded
    while (there is text to be coded):
        1.search for the longest match for S[p...] in s[p-w...p-1]
            // suppose match at pos m with len = l
        2. output the triple (p-m,l,s[p+1])
        3. p = p+l+1
```

דוגמה:


String:
A_walrus_in_Spain_is_a_walrus_in_vain.

Encoded String
(0,0,'A')(0,0,'_')(0,0,'w')(0,0,'a')(0,0,'l')(0,0,'r')(0,0,'u')
(0,0,'s')(7,1,'i')(0,0,'n')(3,1,'S')(0,0,'p')
(11,1,'i')(6,2,'i')(12,2,'a')(21,11,'v')(20,3,'.')

```
LZ77_decode:
    p = 1  // next char to be coded
    for each triple (f,l,c) in the input :
       1. S[p...p+l-1]=s[p-f..p-f+l-1]
       2. s[p+l] = c
       3. p = p+l+1
```

דוגמה:

Encoded String
(0,0,'A')(0,0,'_')(0,0,'w')(0,0,'a')(0,0,'l')(0,0,'r')(0,0,'u')
(0,0,'s')(7,1,'i')(0,0,'n')(3,1,'S')(0,0,'p')
(11,1,'i')(6,2,'i')(12,2,'a')(21,11,'v')(20,3,'.')

String:
A_walrus_in_Spain_is_a_walrus_in_vain.

הצבעה חוזרת

עבור a^{100} נקודד $(0,0,'a')\,(1,98,'a')$ נזהה כאשר $offset < length$

ב$encdoe$ נשנה את: s[p..] in s[p-w...p-1] $\Leftarrow$ s[p..] in s[p-w...]

חסרונות:

- משתמשים ב $sliding\,window$ כלומר לאורך כל הקידוד צריך להחזיק את כל ה $window$
  – לא ברור איך מוצאים מחרוזת הכי ארוכה (לשפר את את הקידוד)
- השלשה (f,l,c) עולה הרבה ביטים
  – ב $ascii$ אנחנו נותנים לתו 8 ביטים ופה תו יתפוס $\left(\underbrace{12}_{off},\underbrace{4}_{len},\underbrace{8}_{c}\right)$ סה"כ 24 ביטים

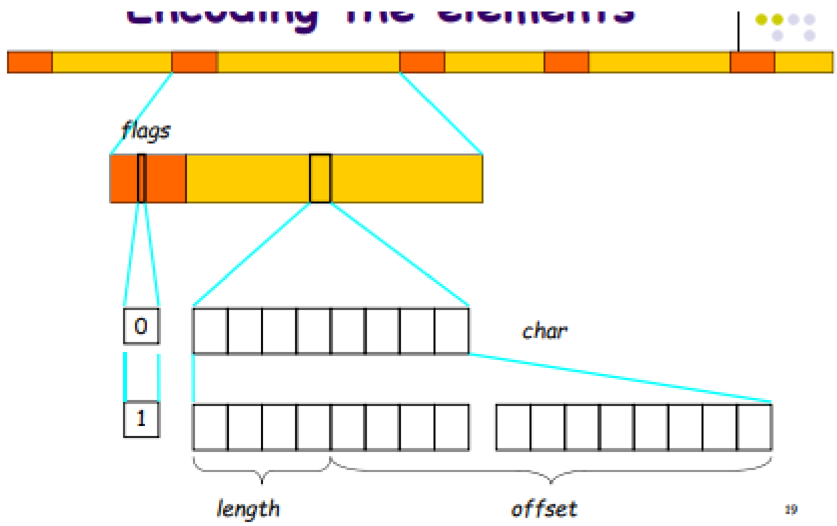
3.11 $LZSS$

- שיפור ל $LZ77$
- חיפוש המחרוזת הארוכה ביותר יהיה באמצועץ עץ בינארי מאוזן
- במקום להשתמש בשלשה , נשתמש לפעמים בתווים בודדים ולפעמים במצביע שיהיה זוג סדור $(offset, length)$ - כמה נלך אחורה וכמה תווים להעתיק
- נבדיל בין $char$ לזוג סדור על ידי ביט מוביל 0 או 1 שיסמן
  – $(0, charcther)$ $\Leftarrow$ גודל של תו יהיה תמיד 8 סיביות $ascii$- + ביט שיסמן זאת = $9bit$
  – קידוד של $(1, off, len)$ יהיה שונה - תלוי בכמה סיביות הגדרנו ל len ו לoff וזה תלוי בגדול החלון

– דוגמה: עם חלון של $2^{12}=4096$, $look-ahead$ של $2^{4}=16$ $(character)$

אז: $\left(\underbrace{1}_{sign},\underbrace{12}_{off},\underbrace{4}_{len}\right)=17$ ביטים לקודד . כלומר עבור מחרוזת באורך 2 צריך 18 ביטים ועבור זוג סדור צריך 17 לכן שווה, אבל יש פעמים שגם זה לא יהיה שווה לנו ונעדיף להעתיק מחרוזות באורך 2 או 3 כבודדים מאשר בזוגות סדורים

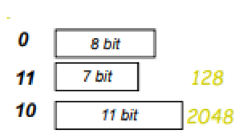
- בחיים האמיתים מחשבים עובדים בבתים - כלומר יחידות של 8 ביטים ולכן נרצה לאגד שמיניות קידוד ומכל שמינייה לקחת את הביט המוביל שמדריך איך לקרוא אותה - ובכך חזרנו ליחידות של 8 ביט וזה מקל עלינו מאוד - בכתום כל $flag$ בצהוב כל קידוד של תווים



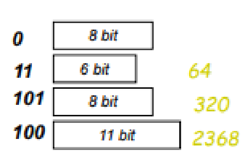
- בחיים האמיתים בודקים מחרוזת ארוכה בעזרת $hashing$ ולא בעזרת עץ בינארי - יותר מהיר (לא בהכרח נקבל דחיסה מקסימלית אבל מהיר הרבה היותר)

3.12 LZS

- אותו רעיון כמו ב $LZSS$ אבל נרצה להגדיל את החלון כלומר להגדיל את הטווח ש$offset$ יכול להגיע אליו
- לדוגמה:
  – ראשית עבור $char$ בודד ניתן 0 מוביל ו 8 ביטים
  – עבור (off, len) ניתן 2 סיביות מובילות (פרפקסיות אחת לשניה)
  – חלון קרוב $0-128$ תווים נצטרך רק 7 סיביות
  – חלון רחוק $129-2048$ תווים נצטרך 11 סיביות



חלוקה נוספת:



3.13 $LZ78$

- ב$LZ77$ יש מילון מרומז וזה החלון שלנו
- ב$LZ78$ המילון יהיה מפורש (נבנה מילון)
- שניהם אדפיטבים
- משתמשים במבנה נתונים $TRIE$:

אלגוריתם (מהאינטרנט):

1. מתחילים ממילון ריק
2. מצא את ההתאמה הארוכה ביותר בין ה$look-ahead$ לבין ה$entry$ הארוך ביותר הקיים במילון
3. פלוט את הזוג (id,c) כאשר $c=next-char$
4. אם $c=eof$ עצור
5. אחרת הוסף למילון: $(max\_id+1, T(id)\cdot c)$
6. חזור ל2

דוגמה:

הא"ב שלנו $\{00,01,10,11\} \overset{binary}{=} \sum=\{a,b,c,d\}$

$$T=\underbrace{b}\,\underbrace{a}\,\underbrace{d}\,\underbrace{ad}\,\underbrace{ada}\,\underbrace{ba}\,\underbrace{ab}$$

idx	$Phrase$	$encoding$	num of bits
0	ε		0
1	b	$(0,b)$	$0+2$
2	a	$(0,b)$	$1+2$
3	d	$(0,d)$	$2+2$
4	ad	$(2,d)$	$2+2$
5	ada	$(4,a)$	$3+2$
6	ba	$(1,a)$	$3+2$
7	ab	$(2,b)$	$3+2$

$$\Rightarrow \left\{ \begin{array}{c} \underline{encoding:} \\ \underbrace{(0,b)}_{b}\underbrace{(0,a)}_{a}\underbrace{(0,d)}_{d}\underbrace{(2,d)}_{ad}\underbrace{(4,a)}_{ada}\underbrace{(1,a)}_{ba}\underbrace{(2,b)}_{ab} \\ \underline{in\ bits:} \\ \underbrace{(\varepsilon^*,01)}_{b}\underbrace{(0,00)}_{a}\underbrace{(00,11)}_{d}\underbrace{(10,11)}_{ad}\underbrace{(100,00)}_{ada}\underbrace{(001,00)}_{ba}\underbrace{(010,01)}_{ab} \end{array} \right\}$$

$*$ עבור התו <u>הראשון </u>אין צורך להפנות לשורה כי ברור שלא קיימת שורה מתאימה, לכן נחסוך את הביטים הללו. מכאן ואילך הביטים שייצינו את האינדקס יהיו כמספר הביטים הנדרשים לתאר את השורה המקיסמלית במילון

פענוח:

- עובדים בדיוק הפוך :
  - $(0,b)$ יהפוך לרשומה במילון במקום 1 עם b ונפלוט b
  - ...
  - $(2,d)$ נלך לשורה 2 נכתוב את הערך ונוסיף d , נפלוט זאת ונוסיף למלון.

3.14 LZW

- משתמשים במבנה נתונים $TRIE$: מימוש בפועל:
  - מערך של הא"ב שמצביע על מערך עם הא"ב שמצביע וכו' : חיפוש $O\left(|w|+1\right)$
  - אפשרות אחרת מערך שכ"א מצביע על רשימה מקושרת: חפוש $O\left(\left(|w|+1\right)\sum\right)$

הערה: נראה עם מספרים עשרוניים בפועל ישנו תרגום נוסף בין עשרוני לבינארי

קידוד:

```
LZW_encode:

    1. Dictionary = single Characters
    2. w = first char of input
    3. repeat{
        1. k = next char
        2. if(EOF)
            1. output code(w)
        3. else if (w·k)∈ Dictionary
            1. w = w·k
        4. else
            1. output code(w)
            2. Dictionary = w·k
            3. w = k
```

דוגמה

T=wabba_wabba_wabba_wabba_woo_woo

$init \Rightarrow$

code	*phrase*
0	_
1	*a*
2	*b*
3	*o*
4	*w*

$\Rightarrow$

w	*k*	$\in$	$update\,(n, wk)$	*out*
w	*a*	$\times$	$(5, wa)$	4
a	*b*	$\times$	$(6, ab)$	1
b	*b*	$\times$	$(7, bb)$	2
b	*a*	$\times$	$(8, ba)$	2
a	_	$\times$	$(9, a\_)$	1
_	*w*	$\times$	$(10, \_w)$	0
w	*a*	✓	–	
wa	*b*	$\times$	$(11, wab)$	5
b	*b*	✓	–	
bb	*a*	$\times$	$(12, bba)$	7
$\vdots$				

$\Rightarrow$

code	*phrase*
0	_
1	*a*
2	*b*
3	*o*
4	*w*
5	*wa*
6	*ab*
7	*bb*
8	*ba*
9	*a_*
10	*_w*
11	*wab*
12	*bba*
$\vdots$	
15	

קראנו: T=**wabba_wabba**_wabba_wabba_woo_woo ופלטנו: 41221057

פענוח

```
LZW_decode:

    1. Initialize table with single character strings
    2. OLD = first input code
    3. output translation of OLD
    4. while not end of input stream{
        1. NEW = next input code
        2. if NEW is not in the string table
            1. S = translation of OLD
            2. S = S·C
        3. else
            1. S = translation of NEW
        4. output S
        5. C = first character of S
        6. Translation(OLD)·C to the string table
        7. OLD = NEW
    }
```

נניח קיבלנו את הרצף: $0, 1, 3, 2, 4, 7, 0, 9, 10, 0$ ואת המילון:

$code$	$phrase$
0	a
1	b
2	c

$\Rightarrow$

old	new	$\in$	s	out	c	$update\,(n, T(old)\cdot c)$
0	$-$	$\checkmark$	$-$	a	$-$	$-$
0	1	$\checkmark$	b	b	b	$(4, T\,(0)\cdot b) = (3, ab)$
1	3	$\checkmark$	ab	ab	a	$(4, T\,(1)\cdot a) = (4, ba)$
3	2	$\checkmark$	c	c	c	$(5, T\,(3)\,c) = (5, abc)$
2	4	$\checkmark$	ba	ba	b	$(6, T\,(2)\,b) = (6, cb)$
4	7	$\times$	$ba\cdot b$	bab	b	$(7, T\,(4)\,b) = (7, bab)$
7	0	$\checkmark$	a	a	a	$(8, T\,(7)\,b) = (8, baba)$
0	9	$\times$	$a\cdot a$	aa	a	$(9, T\,(0)\,a) = (9, aa)$
9	10	$\times$	$aa\cdot a$	aaa	a	$(10, aaa)$
10	0	$\checkmark$	a	a	a	$(11, aaaa)$

$\Rightarrow$

$code$	$phrase$
0	a
1	b
2	c
3	ab
4	ba
5	abc
6	cb
7	bab
8	$baba$
9	aa
10	aaa
11	$aaaa$

deocded = a b ab c ba bab a aa aaa a

השוואה:

- $LZ77$ בד"כ יותר טוב בגלל ש:
  - ב LZW יש כניסות במילון שמעולם לא נשתמש
  - ב LZW כדי לקבל מחרוזת ארוכה, כל הרישות שלה צריכות להיות קודם במילון
  - ב$LZ77$ ברגע שיש מחרוזת ארוכה חוזרת, נשתמש ישירות בכולה

3.15 דקדוקים חסרי הקשר

- מתקשר לדחיסות מילוניות
- נגדיר כללים כמו באוטומטים:
  - $Terminal\ syumbol$ - אבני היסוד/ התווים
  - $Producation\ rules$ - כללי יצירה

$$A \to a$$

  - דקדוק: הדקדוק צריך ליצר מחרוזת יחידה:
    1. אין אפשרות להסתעפויות בגזירה
    2. אין אפשרות למשהו מעגלי

3.16 $Squiter$

- קורא תווים משמאל לימין + עיבוד $\Leftarrow O(n)$
- חוקים:
  - תווים עוקבים לא יופיע יותר מפעם אחת
  - חוק צריך להופיע לפחות פעמיים

אלגוריתם:

1. כל עוד אנחנו לא ב EOF , בדוק:
   (א) אם ישנן 2 תווים צמודים שחוזרים על עצמם - צור כלל/חוק דקדוק חדש
   (ב) אם חוק שמשתמשים בו פעם אחת - הסר: הכנס את התוכן למופע היחיד שלו

דוגמה:

$$T = abcdbcabcdbc$$

1. S מקבל את $a, ab, abc, ...$ לבסוף יש לו $a\boldsymbol{bc}d\boldsymbol{bc}$ - החוק הראשון הופר (שני תווים עוקבים)
   (א) מגדירים: $\begin{Bmatrix} S \to aAdA \\ A \to bc \end{Bmatrix}$ (משתמשים ב A פעמיים ✓)
2. קולטים את ab : $\begin{Bmatrix} S \to aAdAab \\ A \to bc \end{Bmatrix}$ ✓

$$T = \boldsymbol{abcdbcab}cdbc$$

3. קולטים את c : $\begin{Bmatrix} S \to aAdAabc \\ A \to bc \end{Bmatrix}$ × - bc מופיע פעמיים $\Leftarrow$ נמיר ל $A \Leftarrow$ $\begin{Bmatrix} S \to aAdAaA \\ A \to bc \end{Bmatrix}$ × - aA חוזר על עצמו $\Leftarrow$
   $\begin{Bmatrix} S \to aAdAaA \\ A \to bc \end{Bmatrix} \Leftarrow$ נגדיר: $\begin{Bmatrix} S \to BdAB \\ A \to bc \\ B \to aA \end{Bmatrix}$ ✓

4. קולטים את d : $\begin{Bmatrix} S\to BdABd \\ A\to bc \\ B\to aA \end{Bmatrix}\times$ ־ Bd מופיע פעמיים $\Leftarrow$ נמיר ל C $\Leftarrow\begin{Bmatrix} S\to BdABd \\ A\to bc \\ B\to aA \\ C\to Bd \end{Bmatrix}\times$ ־ B פעם אחת = מיותר ונסיר.

- איך מסירים? לוקחים את הגזירה שלו ומכניסים למופע היחיד שלו: $\begin{Bmatrix} S\to CAC \\ A\to bc \\ C\to aAd \end{Bmatrix}\checkmark$

5. נקרא bc (עד הסוף): $\begin{Bmatrix} S\to CACbc \\ A\to bc \\ C\to aAd \end{Bmatrix}\times$ bc חוזר $\begin{Bmatrix} S\to CACA \\ A\to bc \\ C\to aAd \end{Bmatrix}\times$ CA חוזר $\Leftarrow\begin{Bmatrix} S\to DD \\ A\to bc \\ C\to aAd \\ D\to CA \end{Bmatrix}\times$ C מופיע פעם אחת $\Leftarrow\begin{Bmatrix} S\to DD \\ A\to bc \\ D\to aAdA \end{Bmatrix}\checkmark$

3.17 $Re-Pair$

אלגוריתם

1. מצא זוג תווים הכי נפוץ ־ לדוגמה ab
2. צור כלל ־ לדוגמה $A\to ab$
3. החלף את כל מופעי הזוג בכלל ־ לדוגמה את כל הab ב A
4. חזור ל1 כל עוד מצאת זוג הכי נפוץ נוסף

3.18 $Tunstal$

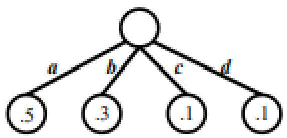
- קוד מסוג $variable-to-fiexed$ ־ ממילים באורך משתנה למילות קוד באורך קבוע
- נחשב ההפוך להפמן
- הדגש על הפענו שיהיה יותר מהיר, כי מילות־קוד באורך קבוע המעפנח פשוט שובר לבלוקים

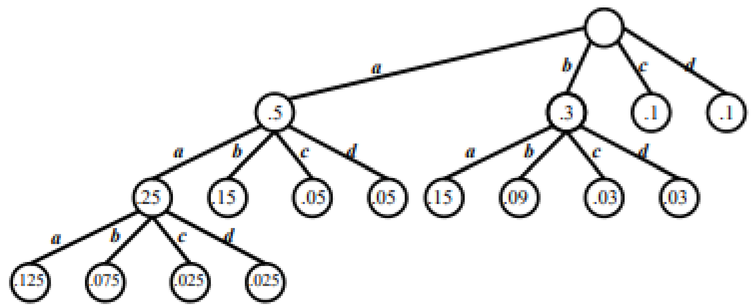
אלגוריתם:

```
Tunstall(n){
    D = ∑
    while (|D| ≤ 2^n -|∑|){
        Let d∈D be with highest probability
            D = D∪ ( ⋃_{σ∈∑} dσ )
        if d ∉ ∑
            D = D\{d}
```

```
        }
    }
```

דוגמה $n = 4$:





Dictionary word	Codeword	Dictionary word	Codeword
aaa	0000	*bb*	1000
aab	0001	*bc*	1001
aac	0010	*bd*	1010
aad	0011	*a*	1011
ab	0100	*b*	1100
ac	0101	*c*	1101
ad	0110	*d*	1110
ba	0111		

Data Compression Course - Dana Shapira

3.19 *Fibonacci*

- ניתן לייצג כל מספר בעזרת מספרי פיבונאצ׳י $n = \sum_{i=1}^{k} n_i \cdot F_{i+1}$
- דוגמה: $n = 73$

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

- בפיבו׳ לא ניקח מספר פעמים (נניח בשלילה, נקבל חזרה בסדרה)
- אין 2 אחדות סמוכים (נניח בשלילה נקבל שהיינו לוקחים את המספר הבא -מהגדרת הסדרה)
  – ננצל זאת ונוסיף לקוד 1 מוביל - כנפגוש זוג נדע שזו התחלה חדשה

- – <u>בעיה:</u> לא קוד מיידי (צריך לקרוא 2 אחדות - לחזור ולשבור)
  - – <u>תיקון:</u> נהפוך כל מילה ו11 יהיה בסוף
  - – <u>דרך מהירה:</u> נעבור על כל המספרים הבינארים שמסתיימות ב11 ואין להם 2 11 סמוכים
- תכונות:
  - – הוכחנו פיבונאצי קוד שלם (בעזרת הקרפאט)
  - – יש F_k מילות קוד באורך $k+1$ ($k+1$ כולל ה1 שמוסיפים בסוף) כך ש $F_k = F_{k+2} - F_{k+1}$
  - – קדו פרפיקסי $\Leftarrow$ מיידי
  - – קוד אחיד = עמיד = יש השמטה/החלפה של ביט אנחנו חוזרים די מהר לקריאה המקורית

$Fib2$

- חסרון ב $Fib1$ - שאין מילת קוד באורך 1 (הראשונה 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 $right\ most(1)$ - ה1 ש"הוספנו" בסוף $fib1$
- מורדים את כל מילות הקוד שמתחילות ב 0

דרך שניה:

- נורד את ה1 שהוספנו בסוף כל מילת קוד מימין
- נוסיף 10 לכל מילת קוד בהתחלה
- נוסיף את 1 כמילת קוד

תכונות:

- מכיל מילה קוד $'1'$
- יש F_{k-2} מילות קוד באורך k
- ניתן להוכיח שתמיד יותר טוב מC_γ ופחות מ C_δ