# שעור 6 – קוד האפמן – Huffman coding

**קוד האפמן** הוא שיטה לקידוד תווי טקסט ללא אובדן נתונים. הקוד המספק דחיסת נתונים מרבית, כלומר מאחסן את התווים במספר מזערי של סיביות. השיטה מתבססת על הקצאת אורך הקוד לתווים על פי שכיחותם, כך שתו נפוץ יוצג באמצעות מספר קטן של סיביות. לרוב ניתן לחסוך באמצעות שיטה זו בין 20% ל-90% משטח האחסון.
נניח שיש לנו קובץ נתונים של 100,000 תווים שאנחנו רוצים לאחסן בצורה קומפקטית. אנו רואים כי התווים בקובץ מופיעים עם התדרים שניתנים על ידי איור 1. כלומר, רק 6 תווים שונים מופיעים, ותו a מתרחשת 45,000 פעמים.

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| Frequency (in thousands) | 45 | 13 | 12 | 16 | 9 | 5 |
| Fixed-length codeword | 000 | 001 | 010 | 011 | 100 | 101 |
| Variable-length codeword | 0 | 101 | 100 | 111 | 1101 | 1100 |

איור 1

יש מספר שיטות לקדד את הנתונים. אנו נתמקד בשיטה שלכל תו מתאימה את מחרוזת בינארית של תווים שנקרא ***codeword***. כאשר להשתמש קוד **באורך קבוע** (***fixed-length code***) נצטרך 3 סיביות להצגת 6 תווים: a = 000, b = 001, . . . , f = 101.
שיטה זו דורשת 300000 סיביות לקידוד כל הקובץ.
שימוש **בקוד שארכו שונה** (***variable-length code***) ניתן להשתמש בפחות זכרון. קוד כזה מיוצג באיור 1 ודורש

$$(1 \cdot 45 + 3 \cdot 13 + 3 \cdot 12 + 3 \cdot 16 + 9 \cdot 4 + 5 \cdot 4) \cdot 100000 = 224000$$ סיביות

חסכנו בערך 25%. נראה בהמשך שהקוד הזה הוא אופטימאלי ונראה איך מקבלים קוד כזה.

### ***קודי תחיליות (prefix codes) .***

*קודי תחיליות*, כלומר מחרוזת סיביות 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. בדוגמה שלנו ניתן לפענח מחרוזת 001011101 באופן ייחודי:

$$0 \cdot 0 \cdot 101 \cdot 1101 = aabe$$

כאן קודה היא סימן הפרדה.
נגדיר פונקציית המטרה שמחשבת את עלות הקוד, כלומר מספר סיביות הנדרשות לשמירת הנתונים:

$$B(T) = \sum_{c \in C} c.freq \cdot d_t(c)$$

כאשר $C$ הוא אלפבית - אוסף של כל התווים האפשריים,
$c \in C$ תו מסוים, $c.freq$ מספר פעמים (תדר) ש-$c$ מופיע בטקסט, ולכל $c$- מתקיים $c.freq > 0$.
$d_t(c)$ – הוא אורך של קוד של תו $c$.
$B(T)$ נקראת **עלות הקידוד**.

האלגוריתם של האפמן הוא דוגמה לאלגוריתם **חמדן**. הוא מבצע בכל שלב את הפעולה שנראית, נקודתית, כפעולה המשתלמת ביותר

**תיאור האלגוריתם:**

הרעיון באלגוריתם הוא כזה:

- נבנה את העץ הבינארי של הקוד "מלמטה למעלה".
- ראשית, נמצא את שתי האותיות שמספר המופעים שלהן מינימלי, וניצור צומת חדש, כך ששתי האותיות הללו יהיו בניו.
- כעת נתייחס לצומת החדש בתור אות חדשה, שמספר המופעים שלה הוא סכום מספרי המופעים של שתי האותיות שהן בניה.
- כך קיבלנו צמצום של הבעיה מבעיה ב n אותיות לבעיה ב n-1 אותיות. נחזור על התהליך עד שנישאר עם צומת אחד בלבד - שורש העץ.

לצורך פעולתו, האלגוריתם זקוק למנגנון שיאפשר לו למצוא במהירות את שתי האותיות בעלות המופעים המינימאליים. לצורך כך ניתן להיעזר בתור עדיפויות , (`PriorityQueue` ב-java).

מבחינה פורמלית:

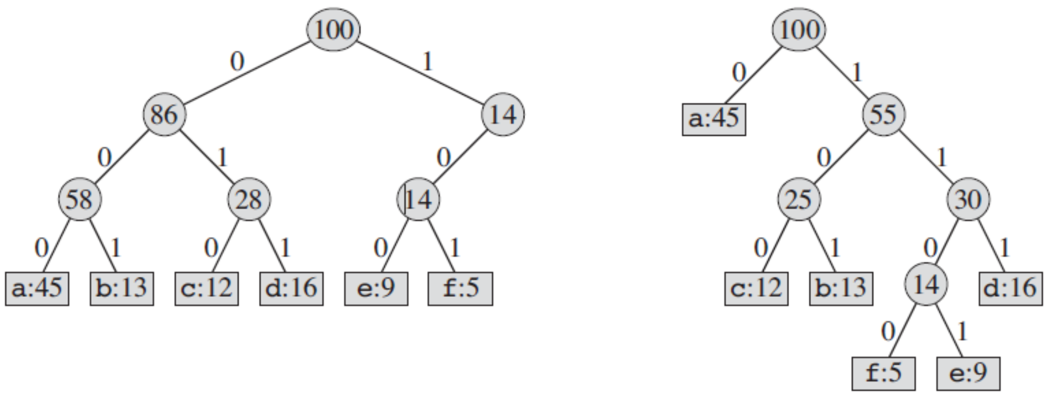
1. צור תור עדיפויות Q שיכיל צמתים המייצגים את כל האותיות, כאשר העדיפות בתור ניתנת לצומת שמייצגת את האות בעלת מספר המופעים הקטן ביותר.
2. כל עוד בתור העדיפויות יש יותר מצומת אחד, בצע:
   2.1. צור צומת חדש z.
   2.2. הוצא מתור העדיפויות את שני האיברים העליונים x,y.
   2.3. הפוך את x,y לבנים הימני והשמאלי של z.
   2.4. קבע את מספר המופעים של z להיות סכום מספרי המופעים של x ו-y.
   2.5. הכנס את z לתור העדיפויות.
3. הצומת הבודד שנותר בתור העדיפויות הוא שורש עץ הקוד המבוקש.
4. קבע את מילת הקוד עבור כל אות, לפי המסלול מהשורש, לעלה שמייצג את האות. אם הצלע מובילה לבן שמאלי, ערך הסיבית יהיה "0". אם היא מובילה לבן ימני, ערכו יהיה "1".

**פסדו-קוד**

```
Huffman(C)  // O(nlog_2 n)
     n = |C|
     Q ← C
     for i=1 to n-1 // O(n)
          allocate a new node z
          x = Q.extractMin() //O(log_2 n)
          y = Q.extractMin() //O(log_2 n)
          z.left = x;    z.right = y
          z.freq = x.freq + y.freq
          x.parent = z;  y.parent = z
          Q.insert(z)
     end-for
 retutn Q.extractMin() // return the root of the tree
```

**סיבוכיות האלגוריתם** היא $O(nlog_2n)$ פעולות. אם האותיות נתונות בצורה ממוינת, ידוע אלגוריתם עם זמן ריצה ליניארי למציאת קוד האפמן.

דוגמה לעץ בהתאם לקודים שנתונים באיור 1:

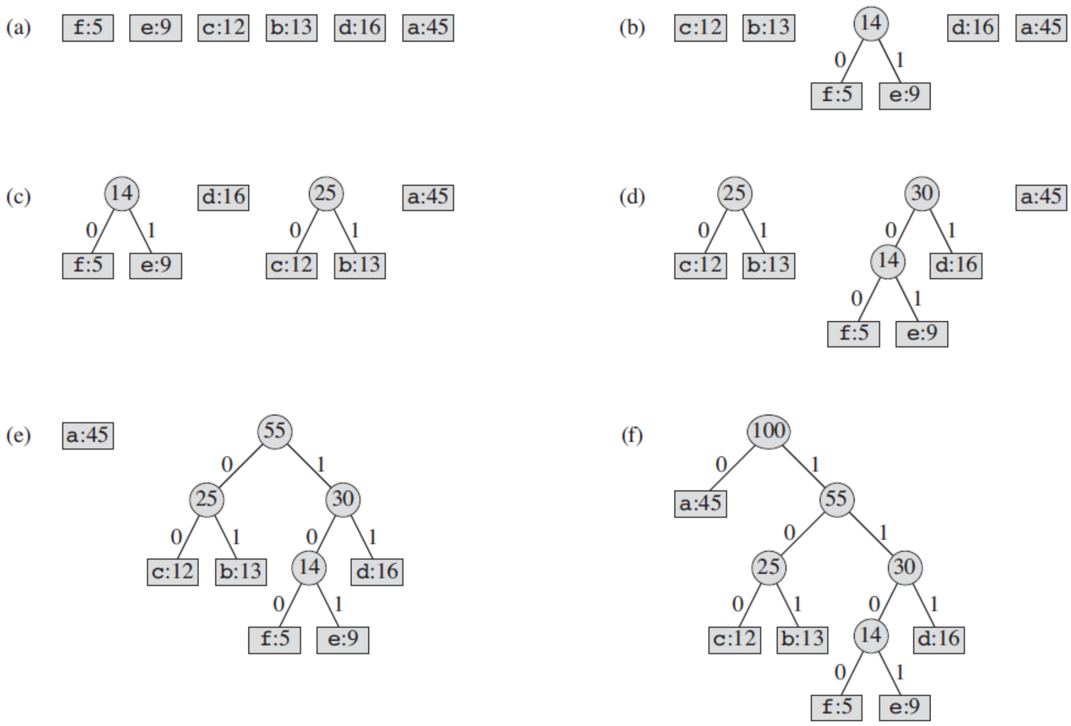



(a) (b)

(a) – דוגמה לקוד שווי אורך (*fixed-length code*)

(b) - דוגמה לקוד שארכו שונה (*variable-length code*)

שלבי אלגוריתם של האפמן:




```
Codes: a→0, b→101, c→100, d→111, e→1101, f→1100
```

טבלה למימוש האלגוריתם:

| letter | index | freq | left index | right index | parent index |
|---|---|---|---|---|---|
| a | 1 | 45 | | | 11 |
| b | 2 | 13 | | | 8 |
| c | 3 | 12 | | | 8 |
| d | 4 | 16 | | | 9 |
| e | 5 | 9 | | | 7 |
| f | 6 | 5 | | | 7 |
| | 7 | 14 | 6 | 5 | 9 |
| | 8 | 25 | 3 | 2 | 10 |
| | 9 | 30 | 7 | 4 | 10 |
| | 10 | 55 | 8 | 9 | 11 |
| | 11 | 100 | 1 | 10 | -1 (שורש) |

נבנה, לדוגמה קוד לאות a: קדקוד האב של a הוא קדקוד 11 שהוא שורש ו-a הוא הבן השמאלי שלו, לכן קוד של a הוא 0.

קוד לאות f: אתחול code(f) = "" – מחרוזת ריקה.
קדקוד האב שלו הוא קדקוד 7 ו- f הוא הבן השמאלי שלו: code(f) = "0".
קדקוד האב של 7 הוא 9 ה-7 הוא הבן השמאלי שלו, לכן code(f) = "0"+ code(f) = "00".
קדקוד האב של 9 הוא 10 ו-9 הוא הבן הימני שלו, לכן code(f) = "1"+ code(f) = "100".
קדקוד האב של 10 הוא 11 ו-10 הוא הבן הימני שלו, לכן code(f) = "1"+ code(f) = "1100".

**psedo-code :**

```
public void huffmanCode(){
   for i=0; to n
      child = i
      parent = mat[child][3]
      while(parent != -1)
            if (mat[parent][1]==child) code[i]="0" + _code[i]
            else code[i]="1" + code[i]
            child = parent
            parent = mat[child][3]
      end-while
   end-for
end-huffmanCode
```

## מימוש אלגוריתם של Huffman בסיבוכיות של O(n)

http://en.wikipedia.org/wiki/Huffman_coding

באלגוריתם של Huffman יש מקום לשיפור היעילות: זה חיפוש שתי תדירויות קטנות ביותר במערך של תדירויות. חיפוש רגיל נותן סיבוכיות של $O(n^2)$, חיפוש באמצעות ערמה (min heap) נותן של $O(n \cdot \log_2 n)$.

כאשר מערך של תדירויות כבר **ממוין בסדר עולה** (מקטן לגדול) , שימוש בשני תורים נותן סיבוכיות של $O(n)$.

תיאור האלגוריתם שלביית העץ (בניית הטבלה):

1. מגדירים שני תורים ריקים: $Q_1$ ו- $Q_2$.
2. מכניסים (פקודת insert ) את כל התדירויות (עלי העץ) ל- $Q_1$ כך שהאיבר (העלה) הקטן ביותר יימצא בחזית (front) של התור $O(n)$.
3. כל עוד בשני התורים יש יותר מאיבר אחד:
   a. מוציאים (פקודת remove ) שני איברים קטנים ביותר $m_1$, $m_2$ מחזית (front) של שני התורים
   b. יוצרים צומת ( Node ) חדש שהצמתים $m_1$, $m_2$ הופכים לבניו וערך (התדירות) שווה לסכום הערכים (התדירויות) שלו ומכניסים (פקודת insert ) את הצומת החדש לתור השני $Q_2$.
   c. חוזרים לסעיף a.
4. הצומת האחרון שנשאר באחד מהתורים הוא ראש העץ. סיימנו לבנות את הטבלה.

**דוגמה:**

| | | | | |
|---|---|---|---|---|
| 1) | Q1: 5, 9, 12, 13, 16, 45; | | | Q2: Ø |
| 2) | x1 = 5, | x2 = 9, | Q1: 12, 13, 16, 45; | Q2: 5 + 9 = 14 |
| 3) | x1 = 12, | x2 = 13; | Q1: 16, 45; | Q2: 14, 25 |
| 4) | x1 = 16, | x2 = 14 ; | Q1: 45; | Q2: 25, 30 |
| 5) | x1 = 25, | x2 = 30; | Q1: 45; | Q2: 55 |
| 6) | x1 = 45, | x2 = 55; | Q1: Ø | Q2: 100 |