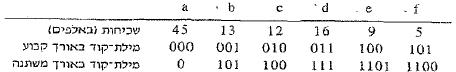
**Huffman code**

**קוד הופמן**

זו שיטה יעילה מאוד לדחיסת נתונים הזוכה לשימוש נרחב. בדרך כלל ניתן לחסוך באמצעותה בין 20% ל-90% ממקום האחסון והדבר תלוי במאפייני הקובץ הנדחס**. האלגוריתם החמדן** של הופמן משתמש בטבלת שכיחויות/תדירויות של מופעי התווים לבניית דרך אופטימלית לייצוג כל תו כמחזורת בינארית.

נניח אנו רוצים לאחסן בצורה דחוסה קובץ הנתונים המכיל 100,000 תווים. התווים השונים מופיעים בקובץ בשכיחויות הנתונות להלן. כלומר, בקובץ מופיעים 6 תווים שונים בלבד והתו a מופיע 45,000 פעמים.



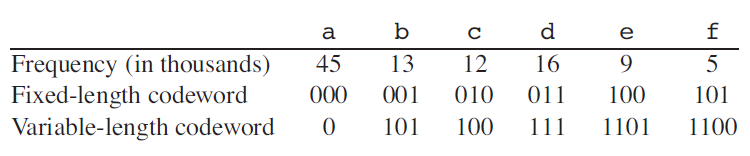
קיימות דרכים רבות ליצוג קובץ מידע כזה. אנו נראה את בניית **קוד תווים בינארי (binary character code)** או בקיצור **קוד**. בקוד זה כל תו מיוצג ע"י מחרוזת בינארית יחידה. אם נשתמש **בקוד באורך קבוע (fixed-length code**), יידרשו 3 סיביות לייצוג 6 תווים: a=000, b=001,…, f=101. בשיטה זו נדרשות 300,000 סיביות לקידוד הקובץ כולו. האם ניתן לשפר את התוצאה הזו?

**קוד באורך משתנה (variable-length code)** יכול להיות יעיל בהרבה מאשר קוד באורך קבוע ע"י כך שמייצגים תווים המופיעים בשכיחות גבוהה באמצעות מילות קוד קצרות ותווים נדירים יותר באמצעות מילות קוד ארוכות. למשל, התו a מיוצג ע"י מחרוזת 0 שאורכה 1, התו f מיוצג ע"י המחרוזת 1100 שאורכה 4. מספר הסיביות הנדרשות לייצוג הקובץ בקוד זה הוא: (45\*1+13\*3+12\*3+16\*3+9\*4+5\*4)\*1000=224,000

דהיינו, חיסכון של כ-25%. למעשה, זה קוד תווים אופטימאלי עבור קובץ זה.

**קודי תחיליות (prefix codes)** הם קודים שבהם אין מילת-קוד שהיא גם תחילית של מילת-קוד אחרת. דחיסת הנתונים האופטימלית המושגת ע"י קוד תווים, ניתן תמיד להשיג באמצעות קוד תחיליות. השימוש בקודים תחיליות רצוי כי הם מפשטים את הקידוד (הדחיסה) ואת הפענוח. הקידוד הוא תמיד קל עבור כל קוד תווים בינארי, כי פשוט משרשרים את מילות הקוד המייצגות כל אחד מהתווים בקובץ.

לדוגמה,



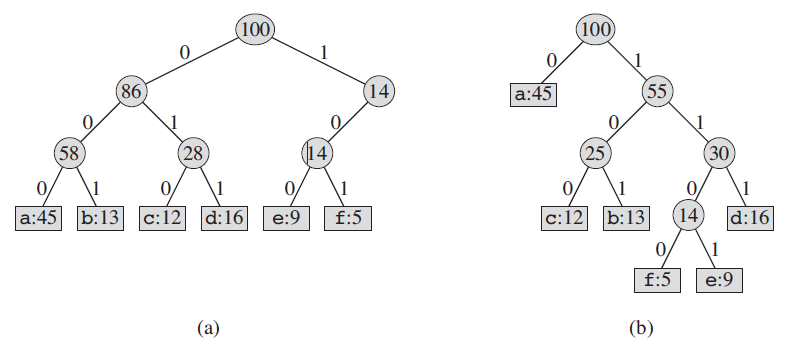
באמצעות קוד התחיליות באורך משתנה, ניתן לקודד את הקובץ בו שלושת התווים abc באופן הבא, כאשר "." מסמן שרשור:

פענוחו של קוד התחיליות גם פשוט: מאחר שאף מילת-קוד אינה תחילית של מילת-קוד אחרת, מילת הקוד הראשונה בקובץ המקודד היא חד-משמעית.

לדוגמה, ניתן לפענח מחרוזת 001011101 באופן ייחודי לתווי המקור:

תהליך הפענוח דורש ייצוג נוח של קוד התחיליות על מנת לזהות בקלות את מילת הקוד הראשונה, אחד הייצוגים האפשריים הוא ע"י **עץ בינארי** שהעלים שלו הם התווים הנתונים. מילת הקוד הבינארית של תו מיוצגת בעץ ע"י מסלול מהשורש עד אותו תו, כאשר 0 – בן שמאלי, 1 –בן ימני. קוד אופטימלי עבור קובץ מיוצג תמיד ע"י **עץ מלא**, אשר לכל צומת שאינו עלה יש שני בנים.

דוגמה לעצים המתאמים לשתי שיטות הקידוד (i - עץ המתאים לקוד באורך הקבוע, ii - עץ המתאים לקוד התחיליות האופטימלי), בכל עלה מופיעה תווית ובה תו ושכיחות הופעתו. בכל צומת פנימי מופיעה תווית ובה סכום המשקלות של העלים בתת-עץ שלו.:



בעצים בינאריים מלאים כאשר ∑ הוא אלפבית שממנו נלקחים התווים, בקידוד תחיליות אופטימלי, יש בדיוק |∑| עלים, אחד עבור כל אות באלפבית ובדיוק |∑|-1 צמתים פנימיים.

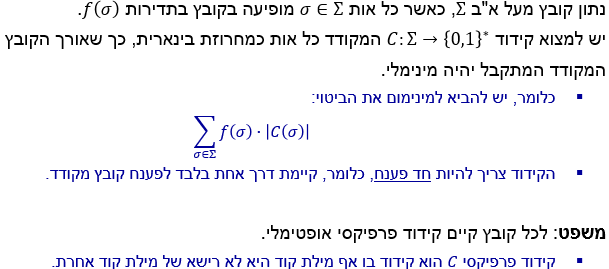
בהינתן עץ T המייצג קוד תחיליות. מספר הסיביות הנדרשות לקידוד קובץ עבור כל תו σ השייך לאלפבית ∑ כאשר f(σ)הוא שכיחות (תדירות) הופעתו של σ בקובץ ו-|C(σ)| הוא עומק של העלה של σ בעץ וגם אורכה של מילת הקוד עבור התו σ, נחשב לפי הנוסחה:

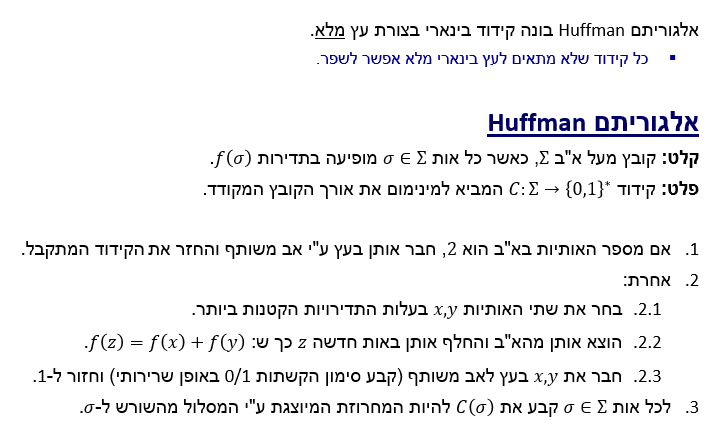
מספר זה מוגדר בעלות (cost) של העץ T.

**בניית קוד הופמן**

הופמן פיתח **אלגוריתם חמדן** שבונה קוד תחיליות אופטימלי הנקרא **קוד הופמן** (**Huffman code)**. האלגוריתם בונה את העץ T המייצג את הקוד האופטימלי "מלמטה למעלה" הוא מתחיל עם קבוצה של |∑| עלים ומבצע סדרה של |∑|-1 פעולות "מיזוג" ליצירת העץ הסופי.

**רעיון האלגוריתם:**





נראה **דוגמה של הרצת האלגוריתם**:

**String:** "this is an example of a huffman tree"

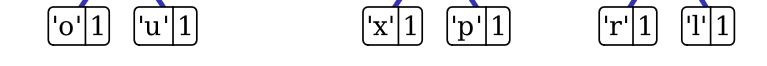
16 symbols

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Char** | **o** | **u** | **x** | **p** | **r** | **l** | **n** | **t** | **m** | **i** | **h** | **s** | **f** | **e** | **a** | **‘ ‘** |
| **Frequency** | **1** | **1** | **1** | **1** | **1** | **1** | **2** | **2** | **2** | **2** | **2** | **2** | **3** | **4** | **4** | **7** |

**Define that right child gets code 1 and left child gets code 0**

במחרוזת 16 תווים, לבניית העץ נדרשים 15 שלבי מיזוג. העץ הסופי מייצג את קוד התחיליות האופטימאלי. מילת הקוד היא סדרת התוויות על הקשתות המרכיבות את המסלול מן השורש עד לאות זה.

**Step 1**

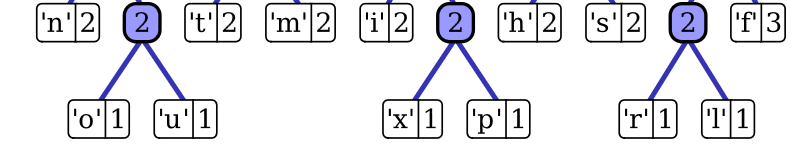
****

נעזר בטבלה (שקולה לעץ) למימוש האלגוריתם כאשר שורות בטבלה הן 16 (עלים של תווים) ועוד 15 עבור צמתים/אבות מדומים:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **##** | **Son-L-0** | **Son-R-1** | **father** | **frequency** | **char** | **code** |
| **1** |  |  |  | **1** | **o** |  |
| **2** |  |  |  | **1** | **u** |  |
| **3** |  |  |  | **1** | **x** |  |
| **4** |  |  |  | **1** | **p** |  |
| **5** |  |  |  | **1** | **r** |  |
| **6** |  |  |  | **1** | **l** |  |
| **7** |  |  |  | **2** | **n** |  |
| **8** |  |  |  | **2** | **t** |  |
| **9** |  |  |  | **2** | **m** |  |
| **10** |  |  |  | **2** | **i** |  |
| **11** |  |  |  | **2** | **h** |  |
| **12** |  |  |  | **2** | **s** |  |
| **13** |  |  |  | **3** | **f** |  |
| **14** |  |  |  | **4** | **e** |  |
| **15** |  |  |  | **4** | **a** |  |
| **16** |  |  |  | **7** | **‘ ‘** |  |
| **1** |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |
| **6** |  |  |  |  |  |  |
| **7** |  |  |  |  |  |  |
| **8** |  |  |  |  |  |  |
| **9** |  |  |  |  |  |  |
| **10** |  |  |  |  |  |  |
| **11** |  |  |  |  |  |  |
| **12** |  |  |  |  |  |  |
| **13** |  |  |  |  |  |  |
| **14** |  |  |  |  |  |  |
| **15** |  |  |  |  |  |  |

**Step 2**

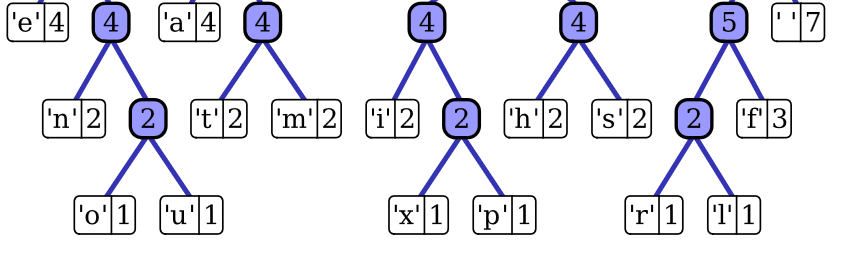
**1 2 3**

****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **##** | **son** | **son** | **father** | **frequency** | **char** | **code** |
| **1** |  |  | **2 (1)** | **1** | **o** |  |
| **2** |  |  | **2 (1)** | **1** | **u** |  |
| **3** |  |  | **2 (2)** | **1** | **x** |  |
| **4** |  |  | **2 (2)** | **1** | **p** |  |
| **5** |  |  | **2 (3)** | **1** | **r** |  |
| **6** |  |  | **2 (3)** | **1** | **l** |  |
| **7** |  |  |  | **2** | **n** |  |
| **8** |  |  |  | **2** | **t** |  |
| **9** |  |  |  | **2** | **m** |  |
| **10** |  |  |  | **2** | **i** |  |
| **11** |  |  |  | **2** | **h** |  |
| **12** |  |  |  | **2** | **s** |  |
| **13** |  |  |  | **3** | **f** |  |
| **14** |  |  |  | **4** | **e** |  |
| **15** |  |  |  | **4** | **a** |  |
| **16** |  |  |  | **7** | **‘ ‘** |  |
| **1** | **o** | **u** | **2 (1)** |  |  |  |
| **2** | **x** | **p** | **2 (2)** |  |  |  |
| **3** | **r** | **l** | **2 (3)** |  |  |  |
| **4** |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |
| **6** |  |  |  |  |  |  |
| **7** |  |  |  |  |  |  |
| **8** |  |  |  |  |  |  |
| **9** |  |  |  |  |  |  |
| **10** |  |  |  |  |  |  |
| **11** |  |  |  |  |  |  |
| **12** |  |  |  |  |  |  |
| **13** |  |  |  |  |  |  |
| **14** |  |  |  |  |  |  |
| **15** |  |  |  |  |  |  |

**Step 3**

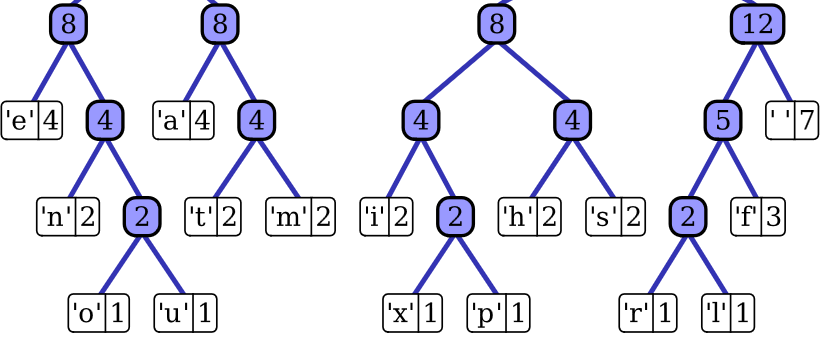
**1 2 3 4**

****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **##** | **son** | **son** | **father** | **frequency** | **char** | **code** |
| **1** |  |  | **2 (1)** | **1** | **o** |  |
| **2** |  |  | **2 (1)** | **1** | **u** |  |
| **3** |  |  | **2 (2)** | **1** | **x** |  |
| **4** |  |  | **2 (2)** | **1** | **p** |  |
| **5** |  |  | **2 (3)** | **1** | **r** |  |
| **6** |  |  | **2 (3)** | **1** | **l** |  |
| **7** |  |  | **4 (1)** | **2** | **n** |  |
| **8** |  |  | **4 (2)** | **2** | **t** |  |
| **9** |  |  | **4 (2)** | **2** | **m** |  |
| **10** |  |  | **4 (3)** | **2** | **i** |  |
| **11** |  |  | **4 (4)** | **2** | **h** |  |
| **12** |  |  | **4 (4)** | **2** | **s** |  |
| **13** |  |  | **5** | **3** | **f** |  |
| **14** |  |  |  | **4** | **e** |  |
| **15** |  |  |  | **4** | **a** |  |
| **16** |  |  |  | **7** | **‘ ‘** |  |
| **1** | **o** | **u** | **2 (1)** |  |  |  |
| **2** | **x** | **p** | **2 (2)** |  |  |  |
| **3** | **r** | **l** | **2 (3)** |  |  |  |
| **4** | **n** | **2 (1)** | **4 (1)** |  |  |  |
| **5** | **t** | **m** | **4 (2)** |  |  |  |
| **6** | **i** | **2 (2)** | **4 (3)** |  |  |  |
| **7** | **h** | **s** | **4 (4)** |  |  |  |
| **8** | **2 (3)** | **f** | **5** |  |  |  |
| **9** |  |  |  |  |  |  |
| **10** |  |  |  |  |  |  |
| **11** |  |  |  |  |  |  |
| **12** |  |  |  |  |  |  |
| **13** |  |  |  |  |  |  |
| **14** |  |  |  |  |  |  |
| **15** |  |  |  |  |  |  |

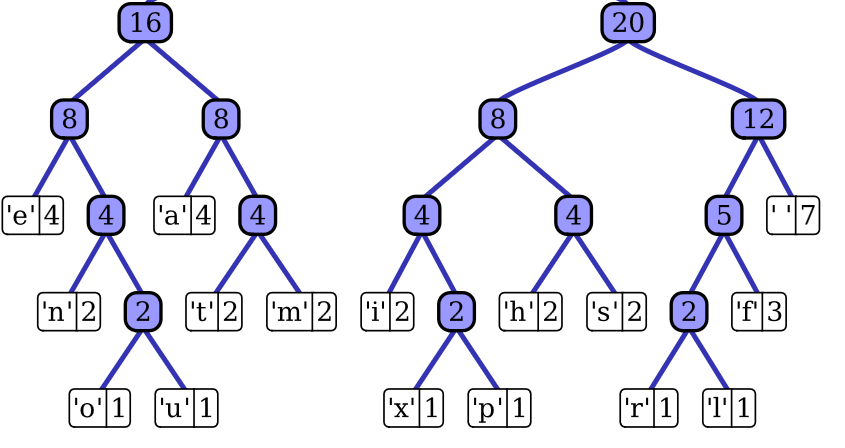
**Step 4**

**1 2 3**

****

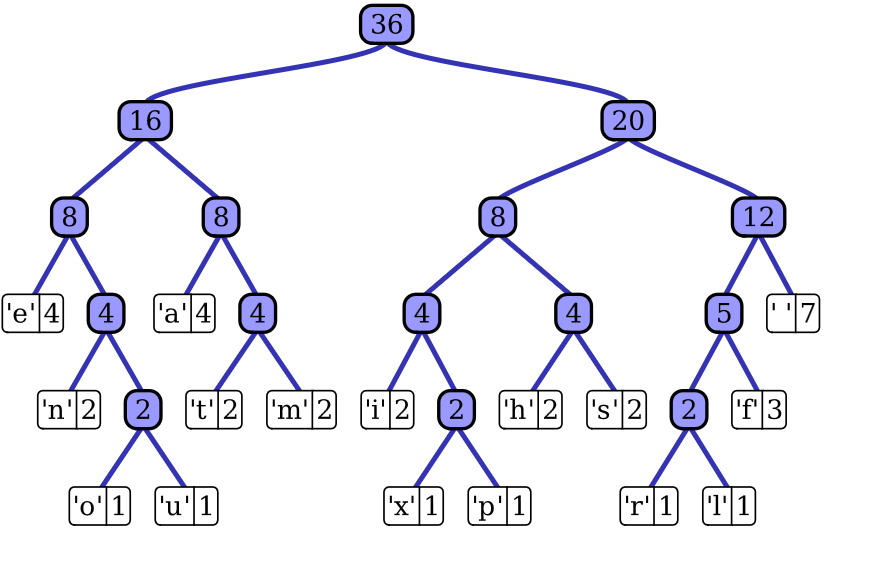
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **##** | | **son** | | **son** | **father** | **frequency** | **char** | **code** | |
| **1** | |  | |  | **2 (1)** | **1** | **o** |  | |
| **2** | |  | |  | **2 (1)** | **1** | **u** |  | |
| **3** | |  | |  | **2 (2)** | **1** | **x** |  | |
| **4** | |  | |  | **2 (2)** | **1** | **p** |  | |
| **5** | |  | |  | **2 (3)** | **1** | **r** |  | |
| **6** | |  | |  | **2 (3)** | **1** | **l** |  | |
| **7** | |  | |  | **4 (1)** | **2** | **n** |  | |
| **8** | |  | |  | **4 (2)** | **2** | **t** |  | |
| **9** | |  | |  | **4 (2)** | **2** | **m** |  | |
| **10** | |  | |  | **4 (3)** | **2** | **i** |  | |
| **11** | |  | |  | **4 (4)** | **2** | **h** |  | |
| **12** | |  | |  | **4 (4)** | **2** | **s** |  | |
| **13** | |  | |  | **5** | **3** | **f** |  | |
| **14** | |  | |  | **8 (1)** | **4** | **e** |  | |
| **15** | |  | |  | **8 (2)** | **4** | **a** |  | |
| **16** | |  | |  | **12** | **7** | **‘ ‘** |  | |
| **1** | **o** | | **u** | | **2 (1)** |  |  | |  |
| **2** | **x** | | **p** | | **2 (2)** |  |  | |  |
| **3** | **r** | | **l** | | **2 (3)** |  |  | |  |
| **4** | **n** | | **2 (1)** | | **4 (1)** |  |  | |  |
| **5** | **t** | | **m** | | **4 (2)** |  |  | |  |
| **6** | **i** | | **2 (2)** | | **4 (3)** |  |  | |  |
| **7** | **h** | | **s** | | **4 (4)** |  |  | |  |
| **8** | **2 (3)** | | **f** | | **5** |  |  | |  |
| **9** | **e** | | **4 (1)** | | **8 (1)** |  |  | |  |
| **10** | **a** | | **4 (2)** | | **8 (2)** |  |  | |  |
| **11** | **4 (3)** | | **4 (4)** | | **8 (3)** |  |  | |  |
| **12** | **5** | | **‘ ‘** | | **12** |  |  | |  |
| **13** |  | |  | |  |  |  | |  |
| **14** |  | |  | |  |  |  | |  |
| **15** |  | |  | |  |  |  | |  |

**Step 5**

****

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **##** | **son** | | **son** | **father** | **frequency** | **char** | **code** | |
| **1** |  | |  | **2 (1)** | **1** | **o** |  | |
| **2** |  | |  | **2 (1)** | **1** | **u** |  | |
| **3** |  | |  | **2 (2)** | **1** | **x** |  | |
| **4** |  | |  | **2 (2)** | **1** | **p** |  | |
| **5** |  | |  | **2 (3)** | **1** | **r** |  | |
| **6** |  | |  | **2 (3)** | **1** | **l** |  | |
| **7** |  | |  | **4 (1)** | **2** | **n** |  | |
| **8** |  | |  | **4 (2)** | **2** | **t** |  | |
| **9** |  | |  | **4 (2)** | **2** | **m** |  | |
| **10** |  | |  | **4 (3)** | **2** | **i** |  | |
| **11** |  | |  | **4 (4)** | **2** | **h** |  | |
| **12** |  | |  | **4 (4)** | **2** | **s** |  | |
| **13** |  | |  | **5** | **3** | **f** |  | |
| **14** |  | |  | **8 (1)** | **4** | **e** |  | |
| **15** |  | |  | **8 (2)** | **4** | **a** |  | |
| **16** |  | |  | **12** | **7** | **‘ ‘** |  | |
| **1** | **o** | **u** | | **2 (1)** |  |  | |  |
| **2** | **x** | **p** | | **2 (2)** |  |  | |  |
| **3** | **r** | **l** | | **2 (3)** |  |  | |  |
| **4** | **n** | **2 (1)** | | **4 (1)** |  |  | |  |
| **5** | **t** | **m** | | **4 (2)** |  |  | |  |
| **6** | **i** | **2 (2)** | | **4 (3)** |  |  | |  |
| **7** | **h** | **s** | | **4 (4)** |  |  | |  |
| **8** | **2 (3)** | **f** | | **5** |  |  | |  |
| **9** | **e** | **4 (1)** | | **8 (1)** |  |  | |  |
| **10** | **a** | **4 (2)** | | **8 (2)** |  |  | |  |
| **11** | **4 (3)** | **4 (4)** | | **8 (3)** |  |  | |  |
| **12** | **5** | **‘ ‘** | | **12** |  |  | |  |
| **13** | **8 (1)** | **8 (2)** | | **16** |  |  | |  |
| **14** | **8 (3)** | **12** | | **20** |  |  | |  |
| **15** |  |  | |  |  |  | |  |

**Step 6**

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **##** | **son** | **son** | **father** | **frequency** | **char** | | **code** |
| **1** |  |  | **2 (1)** | **1** | **o** | | **00110** |
| **2** |  |  | **2 (1)** | **1** | **u** | | **00111** |
| **3** |  |  | **2 (2)** | **1** | **x** | | **10010** |
| **4** |  |  | **2 (2)** | **1** | **p** | | **10011** |
| **5** |  |  | **2 (3)** | **1** | **r** | | **11000** |
| **6** |  |  | **2 (3)** | **1** | **l** | | **11001** |
| **7** |  |  | **4 (1)** | **2** | **n** | | **0010** |
| **8** |  |  | **4 (2)** | **2** | **t** | | **0110** |
| **9** |  |  | **4 (2)** | **2** | **m** | | **0111** |
| **10** |  |  | **4 (3)** | **2** | **i** | | **1000** |
| **11** |  |  | **4 (4)** | **2** | **h** | | **1010** |
| **12** |  |  | **4 (4)** | **2** | **s** | | **1011** |
| **13** |  |  | **5** | **3** | **f** | | **1101** |
| **14** |  |  | **8 (1)** | **4** | **e** | | **000** |
| **15** |  |  | **8 (2)** | **4** | **a** | | **010** |
| **16** |  |  | **12** | **7** | **‘ ‘** | | **111** |
| **1** | **o** | **u** | **2 (1)** |  |  |  | |
| **2** | **x** | **p** | **2 (2)** |  |  |  | |
| **3** | **r** | **l** | **2 (3)** |  |  |  | |
| **4** | **n** | **2 (1)** | **4 (1)** |  |  |  | |
| **5** | **t** | **m** | **4 (2)** |  |  |  | |
| **6** | **i** | **2 (2)** | **4 (3)** |  |  |  | |
| **7** | **h** | **s** | **4 (4)** |  |  |  | |
| **8** | **2 (3)** | **f** | **5** |  |  |  | |
| **9** | **e** | **4 (1)** | **8 (1)** |  |  |  | |
| **10** | **a** | **4 (2)** | **8 (2)** |  |  |  | |
| **11** | **4 (3)** | **4 (4)** | **8 (3)** |  |  |  | |
| **12** | **5** | **‘ ‘** | **12** |  |  |  | |
| **13** | **8 (1)** | **8 (2)** | **16** |  |  |  | |
| **14** | **8 (3)** | **12** | **20** |  |  |  | |
| **15** | **16** | **20** | **36** |  |  |  | |

**מימוש האלגוריתם:**

1. **באמצעות טבלה**

**Pseudocode**

**int** \_n = freq.length;

**int** [] \_freq = **new** **int**[2\*\_n-1];

/\* table: \_mat[][0]- parent index,

\_mat[][1]- left child index (0),

mat[][2]- right child index (1) \*/

**int** [][] \_mat = **new** **int**[2\*\_n-1][3];

String \_code[];// Huffman Code

//initialization

**for** (**int** i=0; i<\_n; i++){ // O(n)

\_freq[i] = freq[i];

\_code[i] = "";

}

// create the table

**HuffmanTable()**

**int** [] min12 = **new** **int**[2];

**int** ind1, ind2;

**int** parent = \_n;

**while** (parent < \_freq.length){ // O(n)

// find indexes of two smallest unused frequencies

min12 = **twoMinIndexes**(parent); // O(n)

ind1 = min12[0];

ind2 = min12[1];

\_mat[parent][1] = ind1; //left child (0)

\_mat[parent][2] = ind2; //right child (1)

\_mat[ind1][0] = parent; // parent

\_mat[ind2][0] = parent; // parent

\_freq[parent] = \_freq[ind1] +\_freq[ind2];

\_freq[ind1] = - \_freq[ind1];// not in use

\_freq[ind2] = - \_freq[ind2];// not in use

parent++;

}

// build the Huffman's Code for all letters

**HuffmanCode()**

**for** (**int** i=0; i<\_n; i++){

**int** child = i;

**int** parent = \_mat[child][0];

**while**(parent!=0){

**if** (\_mat[parent][1]==child) \_code[i]=\_code[i]+"0";

**else** \_code[i]=\_code[i]+"1";

child = parent;

parent = \_mat[child][0];

}

}

**סיבוכיות האלגוריתם** היא {\displaystyle \ O(n\log n)} פעולות.

1. **באמצעות ערימת מינימום**

לצורך הפעלת האלגוריתם צריך מנגנון שיאפשר למצוא במהירות את שתי האותיות בעלות המופעים המינימאליים (השכיחויות הנמוכה ביותר) כדי למזגן. לצורך כך ניתן להיעזר במבנה הנתונים ערימה שממומשת ע"י [תור קדימויות/עדיפויות](https://he.wikipedia.org/wiki/%D7%AA%D7%95%D7%A8_%D7%A2%D7%93%D7%99%D7%A4%D7%95%D7%99%D7%95%D7%AA) עם השכיחויות/תדירות כמפתחות, (PriorityQueue ב-java). תוצאת מיזוגן של שתי האותיות היא אות/עצם חדשה אשר שכיחותה היא סכום השכיחויות של שתי האותיות שמוזגו.

**Pseudocode**

C is a set of n characters; c є C and f(c) is frequency of c

|C| - set of leaves

Q – Priority Queue where key is f(c)

**HUFFMAN(C)**

1 n = |C|

2 Q = C //initialization – O(n)

3 **for** i = 1 **to** n – 1 // O(n)

4 allocate a new node z //O()

5 z.*left* = x = EXTRACT-MIN(Q) //O()

6 z.*right* = y = EXTRACT-MIN(Q) //O()

7 z.*freq* = x.*freq* + y.*freq*

8 INSERT(Q,z) //O()

9 **return** EXTRACT-MIN(Q) // return the root of the tree

**סיבוכיות האלגוריתם** כאשר תור ממומש ע"י ערימה בינארית היא {\displaystyle \ O(n\log n)} פעולות.

1. **שיפור סיבוכיות האלגוריתם לסיבוכיות של O(n)**

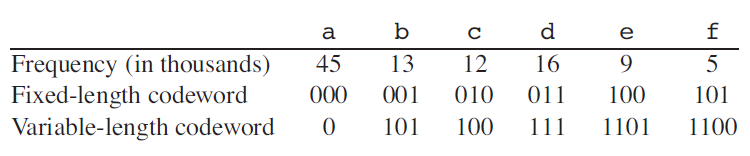
באלגוריתם של Huffman יש מקום לשיפור היעילות: זה חיפוש שתי תדירויות קטנות ביותר במערך של תדירויות. חיפוש רגיל (מילוי מטריצה: n-1 איטרציות, בכל איטרציה חיפוש של 2 מינימליים: ) נותן סיבוכיות של O(n2), חיפוש באמצעות ערימה (min heap) נותן סיבוכיות של .

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

**תיאור האלגוריתם של בניית העץ (בניית הטבלה):**

1. מגדירים שני תורים ריקים: Q1 ו- 2Q.
2. מכניסים (פקודתinsert ) את כל התדירויות (עלי העץ) ל- Q1 כך שהאיבר (העלה)   
   הקטן ביותר (התדירות הקטנה ביותר) יימצא בחזית (front) של התור - O(n).
3. כל עוד בשני התורים יש יותר מאיבר אחד:
   1. מוציאים (פקודתremove ) שני איברים קטנים ביותר x1, x2 מחזית   
      (front) של שני התורים.
   2. יוצרים צומת ( Node ) חדש שהצמתים x1, x2 הופכים לבניו וערך (התדירות) שלו שווה לסכום הערכים (התדירויות) של בניו ומכניסים (פקודתinsert ) את הצומת החדש לתור השני 2Q.
   3. חוזרים לסעיף a..
4. הצומת האחרון שנשאר באחד מהתורים הוא ראש העץ. סיימנו לבנות את הטבלה.

**דוגמה להרצת האלגוריתם:**



1. Q1: 5, 9, 12, 13, 16, 45; Q2:
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: Q2: 100

**פסדו-קוד:**

**public void huffmanON()**Queue<Node> q1, q2 // ArrayBlockingQueue

while (q1.size() + q2.size() > 1 )

Node n1 = nextMin() //O(1)

Node n2 = nextMin()

Node newNode = new Node(place, n1.freq + n2.freq, n1.letterNumber, n2.letterNumber)

n1.setParent(place)

n2.setParent(place) **//O(1)**

q2.add(newNode) **//**O(1)

nodes[place++] = newNode

end-while

// build the Huffman's Code for all letters

for (int i=0; i<numOfLeaves; i++) //O(2n-1)

Node child = nodes[i]

Node parent = nodes[child.parent]

while(child.parent != nil)

if (parent.left==child.letterNumber)

codes[i] = "0" + codes[i]

else codes[i] = "1" + codes[i]

child = parent;

if (child.parent != nil) parent = nodes[child.parent]

end-while

end-for  
**end-huffmanON**

**Node nextMin()**

Node x, y

if (q1.isEmpty())

x = q2.poll() //retrieves and removes the head of the queue

else if (q2.isEmpty())

x = q1.poll()

else

x = q1.peek**()** //retrieves, but doesn’t remove, the head of the queue

y = q2.peek()

if (x.freq > y.freq)

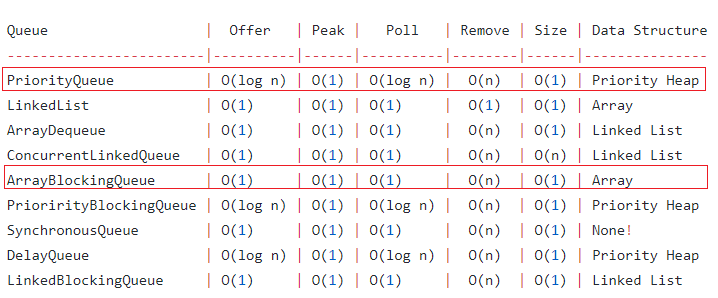
x = q2.poll()

else

x = q1.poll()

end-if

return x  
**end-nextMin**

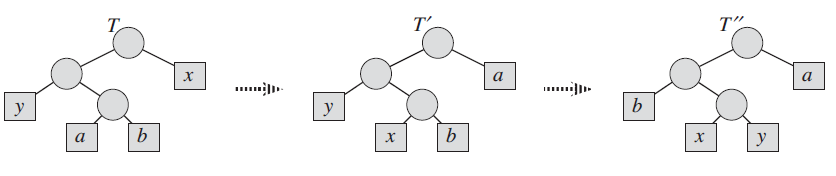


**נכונות האלגוריתם של הופמן**

**למה 1**   
יהיה אלפבית, לכל מוגדרת תדירות . יהיה x,y שני תווים בעלי תדירות מינימאלית. אזי קיים prefix code אופטימאלי ל- *C* כך שקודים של x,y הם בעלי עומק מקסימאלי ,שווי אורך ונבדלים רק בסיבית אחרון, כלומר עלים סמוכים.

**הוכחה.** ניקח Tעץ שמיצג prefix code אופטימאלי ונשנה את T כך שנקבל prefix code אופטימאלי אחר שבו תווים x,y יהיו עלים בעלי עומק מקסימאלי בעץ חדש.  
יהיו a,b שני תווים שהם עלים סמוכים בעלי עומק (מרחק עד השורש) מקסימאלי בעץ T. ללא אובדן של הכלליות, אנו מניחים כי ו-. בגלל ש- x,הם בעלי תדירות מינימאלית אז וגם .

נניח כי , כלומר (אם הלמה טריוויאלית) . נחליף את   
 aו- xכמו שרואים באיור 2:

נקבל עץ חדש , נחליף את y ו- b, נקבל שבו x,y קדקודים סמוכים בעלי עומק מקסימאלי. ההפרש בין העלויות:

*=*

*כאן כי* -הוא קדקוד בעל תדירות מינימאלית, ו-  
 *כי הוא בעל עומק מקסימאל.*

*באופן דומה נחליף* y *ו-* b *ונקבל , לכן   
בגלל ש-T הוא אופטימאלי אז ערכך של פונקציה המטרה שלו קטן מערך של כל פונקציה אחרת, כלומר ומכאן נובע כי . לכן ו- הוא אופטימאלי* ובו x,y הם קדקודים סמוכים בעלי עומק מקסימאלי הנבדלים רק בסיבית אחרון. מש"ל.

**למה 2** יהיה אלפבית, לכל מוגדרת תדירות . יהיה x,y שני תווים בעלי תדירות מינימאלית.נמחק מ- את קדקודים x,y ונוסף קדקוד Z כך ש-  
 , קבלנו - אלפבית חדש. יהיה עץ אופטימלי עבור .  
אז

***הוכחה****: לכל תו מתקיים , לכן   
 . מכוון ש-  
 יש לנו*

*מכאן מקבלים כי*

,   
*או*

*מש"ל.*

***משפט*** *עץ T שהתקבל ע"י אלגוריתם האפמן הוא אופטימלי.*

***הוכחה****: באינדוקציה.*

1. *בסיס האינדוקציה: , אלפבית . האלגוריתם נותן קוד 0 ל-ו-1 ל-,  
   או הפוך – תלוי בתדירות של האותיות. ברור שקוד המורכב מסיבית אחת הוא אופטימלי.*
2. *הנחת אינדוקציה: נניח שקוד של האפמן אופטימאלי עבור תווים. נוכיח שהוא אופטימאלי עבור תווים.*
3. *נבנה , כאשר , וללא אובדן של כלליות ניתן להניח כי הם עלים סמוכים בעלי עומק מקסימאלי (למה 1). בגלל   
   ש- מכיל תווים אלגוריתם של האפמן נותן עץ אופטימלי עבור אלפבית (הנחת אינדוקציה).  
   יהיה T הוא עץ הבנוי לפי האלגוריתם עבור תווים.   
   נניח בדרך השלילה שהוא לא מייצג קוד אופטימאלי עבור אלפבית . לכן קיים עץ אופטימאלי כך ש- . בגלל ש- הוא עץ אופטימלי ,ניתן להניח   
    כי הם עלים סמוכים ב-. יהיה עץ שבנוי מ- כך שקדקוד האב של   
   הוחלף בעלה , כך ש- . אז*

*סתירה לעובדה ש- אופטימאלי. מש"ל.*