

מבני נתונים ויעילות אלגוריתמים

(1.3.2015)

1. פתרון נוסחאות נסיגה לינאריות אי-הומוגניות

נכיר משפט נוסף לפתרון נוסחאות נסיגה, אותו נכנה משפט 4, המכליל את משפט 3, ומשתמש גם הוא לפתרון נוסחאות נסיגה לינאריות אי-הומוגניות.

משפט 4

בהינתן נוסחת נסיגה מהצורה

$$f(n) - c_1 \cdot f(n-1) - c_2 \cdot f(n-2) - c_3 \cdot f(n-3) - \dots - c_k \cdot f(n-k) = b_1^n \cdot p_1(n) + b_2^n \cdot p_2(n) + \dots + b_m^n \cdot p_m(n)$$

כאשר b_i הם קבועים כלשהם השונים זה מזה, ו- $p_i(n)$ הם פולינומים השונים מ-0, ובעלי דרגה d_i , אז המשוואה האופיינית היא:

$$(\alpha^k - c_1 \cdot \alpha^{k-1} - c_2 \cdot \alpha^{k-2} - c_3 \cdot \alpha^{k-3} - \dots - c_k)(\alpha - b_1)^{d_1+1} (\alpha - b_2)^{d_2+1} \dots (\alpha - b_m)^{d_m+1} = 0$$

נראה דוגמאות לשימוש במשפט 4.

שאלה

נתונה נוסחת הנסיגה $T(n) = 2T(n-1) + n + 2^n$, עם תנאי ההתחלה $T(0) = 0$, $T(1) = 3$, מצאו את סדר הגודל (אסימפטוטית) של $T(n)$.

תשובה

נרשום את נוסחת הנסיגה בתור: $T(n) - 2T(n-1) = n + 2^n$.

כעת: $p_1(n) = n$, $b_1 = 1$ (פולינום מדרגה $d = 1$)

$p_2(n) = 1$, $b_2 = 2$ (פולינום מדרגה $d = 0$)

ואז, לפי משפט 4, המשוואה האופיינית היא: $(\alpha - 2)(\alpha - 1)^2(\alpha - 2)^1 = 0$

במילים אחרות: $(\alpha - 2)^2(\alpha - 1)^2 = 0$

לכן, שני הפתרונות השונים הם 1 (מריבוי 2), ו-2 (מריבוי 2).

צורת הפתרון היא: $T(n) = A_1 \cdot 2^n + A_2 \cdot n \cdot 2^n + A_3 \cdot 1^n + A_4 \cdot n \cdot 1^n$

לכן, $T(n) = \Theta(n \cdot 2^n)$.

נשים לב שבשאלה האחרונה כלל לא השתמשנו בתנאי ההתחלה, שכן – מה שנדרשנו למצוא הוא את סדר הגודל של נוסחת הנסיגה, ולא את הביטוי המפורש שלה. יתר על כן, גם לו היינו מתבקשים לגלות את הביטוי המפורש שלה, אין לנו כרגע מספיק תנאי התחלה כדי לעשות זאת (אנחנו זקוקים לארבעה תנאי התחלה, כדי שנקבל מערכת של ארבע משוואות בארבעה נעלמים. צריך היה, אם כך, לחשב שני תנאי התחלה נוספים).

שאלה

נתונה נוסחת הנסיגה $T(n) = 2T(n-1) + 2^n - 1$. חשבו את סדר הגודל (אסימפטוטית) של $T(n)$.

שאלה

הדגימו מדוע התנאי, המופיע במשפט 4, הדורש שכל הקבועים b_i יהיו שונים זה מזה, הוא תנאי חשוב.

2. קוד הופמן (Huffman Coding)

קודי הופמן מהווים שיטה יעילה מאוד לדחיסת נתונים. בדרך כלל ניתן לחסוך באמצעותם בין 20% ל-90% ממקום האחסון, תלוי במאפייני הקובץ הנדחס. האלגוריתם של הופמן הוא אלגוריתם חמדן, המשתמש בטבלת שכיחויות של מופעי התווים, כדי לייצג כל תו באופן אופטימלי – תווים שכיחים ייוצגו באמצעות רצף סיביות קצר, ואילו תווים נדירים ייוצגו באמצעות רצף סיביות ארוך.

נניח שברצוננו לאחסן בצורה דחוסה קובץ נתונים, המכיל 100,000 תווים. התווים השונים מופיעים בקובץ בשכיחויות שונות, כמתואר בטבלה הבאה:

	a	b	c	d	e	f
שכיחות (באלפים)	45	13	12	16	9	5
מילת קוד באורך קבוע	000	001	010	011	100	101
מילת קוד באורך משתנה	0	101	100	111	1101	1100

אם נבחר לקדד את התווים השונים באמצעות קוד באורך קבוע (fixed-length code), נזדקק ל-3 סיביות כדי לייצג את ששת התווים השונים. מכיוון שהקובץ מכיל 100,000 תווים, אשר כל אחד מהם ייוצג באמצעות 3 סיביות, הקובץ יתפוס נפח של 300,000 סיביות.

קוד באורך משתנה (variable-length code) יכול להיות יעיל בהרבה מאשר קוד באורך קבוע, על-ידי כך שמייצגים תווים המופיעים בשכיחות גבוהה באמצעות מילות קוד קצרות, ותווים נדירים יותר באמצעות מילות קוד ארוכות.

כמה סיביות יידרשו כדי לאחסן קובץ באורך 100,000 תווים, המקודד בקוד כזה?

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1000 = 224,000$$

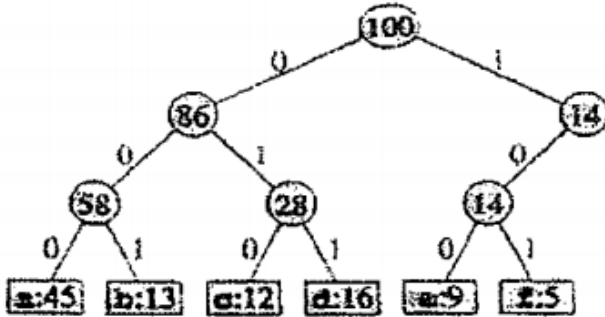
כלומר, הקובץ יתפוס נפח של 224,000 סיביות, שזה חסכון של 25% לעומת הייצוג באמצעות קוד באורך קבוע.

אנו נעסוק רק בקודים שבהם אין מילת קוד שהיא גם תחילית (רישא) של מילת קוד אחרת. קודים כאלה נקראים קודי תחיליות (prefix codes), ולפעמים גם קוד חופשי מתחיליות (prefix free code), או – קוד חופשי (מרישות).

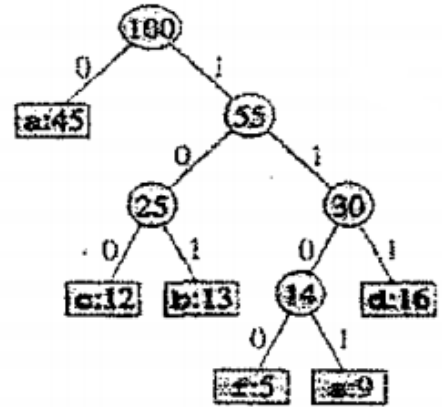
השימוש בקודי תחיליות הוא רצוי, משום שהם מאפשרים את הפענוח: את המחרוזת 001011101 אפשר לפענח רק בדרך אחת 0·0·101·1101, כלומר – aabe.

אחת הדרכים הנוחות לייצג את מילות הקוד, הוא באמצעות עץ בינארי שהעלים שלו הם התווים הנתונים. מילת קוד תיוצג בעץ על-ידי מסלול מן השורש ועד תו, כאשר 0 פירושו "לך אל הבן השמאלי", ו-1 פירושו "לך אל הבן הימני".

כך, למשל, נראים שני העצים עבור שני הקודים שהצגנו לעיל – הן הקוד באורך קבוע, והן הקוד באורך משתנה.



(i)



(ii)

קוד אופטימלי ייוצג תמיד על-ידי עץ בינארי מלא (full), אשר בו לכל צומת שאינו עלה, יש בדיוק שני בנים. הקוד באורך קבוע, בדוגמא שלעיל, איננו אופטימלי, שכן העץ הבינארי המתאים לו איננו מלא לדוגמא: יש מילות קוד המתחילה ב-10...10, אולם אין מילות קוד המתחילות ב-11...11.

נמקד את הדיון שלנו, אם כך, בעצים בינאריים מלאים. מתכוונות שלהם אותן ראינו בעבר, ניתן לקבוע כי אם הקבוצה C היא האלף-בית שממנו נלקחים התווים, אז העץ עבור קוד תחליות אופטימלי יכיל בדיוק $|C|$ עלים, אחד עבור כל אות באלף-בית, ובדיוק $|C|-1$ צמתים פנימיים.



C++



Java/C#



C

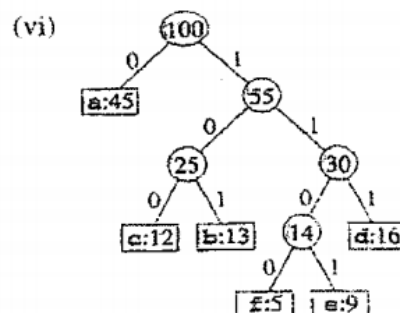
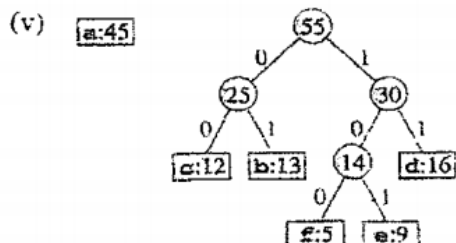
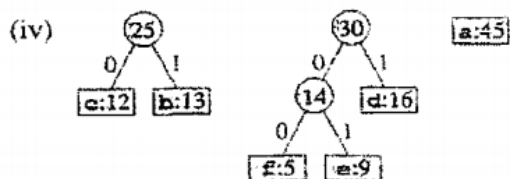
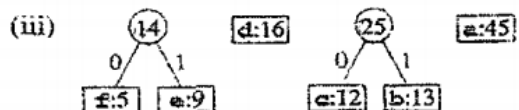
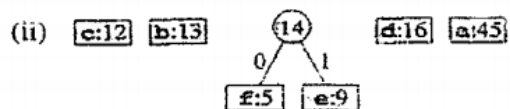
הופמן פיתח אלגוריתם חמדן שבונה קוד תחיליות אופטימלי, הנקרא **קוד הופמן**. האלגוריתם בונה את העץ T המייצג את הקוד האופטימלי "מלמטה למעלה". הוא מתחיל עם קבוצה של $|C|$ עלים, ומבצע סדרה של $|C|-1$ פעולות "מיזוג", עד ליצירת העץ הסופי.

באלגוריתם שלהלן אנחנו מניחים כי C היא קבוצה של n תווים, וכי לכל אות בקבוצה C יש תכונה בשם frequency, המציינת את השכיחות של האות. כמו כן, נניח כי ניתן לבצע על הקבוצה ביעילות פעולות של שליפת איבר בעל שכיחות מינימלית והכנסת איבר. אם, למשל, מיישמים את הקבוצה בעזרת תור קדימויות, הממומש כערימת מינימום, אז ניתן לבצע הוצאת איבר בעל שכיחות מינימלית והכנסת איבר בזמן $\Theta(\log n)$. האלגוריתם מחזיר עץ בינארי המתאים לקוד הופמן המתאים.

```
HUFFMAN-CODING(C)
1   for i ← 1 to |C|-1
2       do allocate memory for z
3           x ← EXTRACT-MIN(C)
4           y ← EXTRACT-MIN(C)
5           z.left ← x
6           z.right ← y
7           z.frequency ← x.frequency + y.frequency
8           INSERT(C, z)
9   return EXTRACT-MIN(C)
```

בכל איטרציה של הלולאה, שולפים מתור הקדימויות את שני הצמתים x ו- y בעלי השכיחויות הנמוכות ביותר, ומכניסים לתור במקומם צומת חדש z , המייצג את מיזוגם. השכיחות של z היא סכום השכיחויות של x ו- y . הצומת x הוא הבן השמאלי של z , והצומת y הוא הבן הימני של z . לאחר $n-1$ פעולות מיזוג, יישאר צומת אחד בתור הקדימויות, והוא יהיה שורשו של העץ הבינארי המתאים לקוד.

(i) f:5 e:9 c:12 b:13 d:16 a:45



נתח את סיבוכיות האלגוריתם, בהנחה שהקבוצה C ממומשת בידי ערימה בינארית. בהינתן קבוצת תווים ושכיחויות, קל לבנות מהם ערימה כזו בזמן לינארי, באמצעות האלגוריתם BUILD-MIN-HEAP.

הלולאה מתבצעת n-1 פעמים, כאשר בכל איטרציה מתבצעת שליפה, שליפה והכנסה – כל אחד מהם בזמן לוגריתמי $\Theta(\log n)$.

בסך הכול, נקבל כי זמן הריצה של אלגוריתם הופמן הוא $\Theta(n \log n)$.

שאלה

הסבירו מדוע העץ הבינארי הנבנה על-ידי אלגוריתם הקידוד של הופמן הוא תמיד עץ בינארי מלא.

שאלה

רוצים לבנות קוד Huffman עבור האלף-בית {a,b,c,d,e,f} בהנחה ששכיחויות אותיות האלף-בית הן:

a:0.1, b:0.3, c:0.22, d:0.08, e:0.1, f:0.2

א. מהו העץ הבינארי המתקבל בסוף ריצת האלגוריתם לבניית הקוד?

ב. כיצד תקודדנה המילים הבאות על פי הקוד שנבנה בסעיף א': dad, bed, feed, cab?

ג. כיצד היו נראות מילים אלו לו היו מקודדות לפי fixed-length coding?