1 הצומת יוכנס כבן ימני של 35. אם נצבע אותו באדום אז כלל 4 יופר כי 35 אדום ויש לו בן אדום. אם הוא יצבע בשחור אז כלל 5 יופר כי במסלול הימני שיוצא מ-35 יש 2 צמתים שחורים, כאשר בשמאלי יש רק 1.

2 נסמן ב-$nbh(k)$ את מספר הצמתים שגובה השחור שלהם הוא k.

ראשית נשים לב שעבור עץ בגבוה h, לכל צומת x מתקיים $bh(x) \leq h$, לכן $nbh(k) = 0$ לכל $k > h$.

נסתכל על עץ שלם בגובה h שבו $n = 2^h - 1$ צמתים פנימיים. אם הרמה האחרונה (לא העלים) כולה שחורה, אז לכל הצמתים מעל רמה זו גובה שחור גדול מ-1. כדי לקבל את מספר הצמתים המקסימלי עם גובה שחור 1, נרצה שהרמה הראשונה שצבועה בשחור תהיה רחוקה כמה שניתן מסוף העץ, כלומר שהרמה האחרונה וזו שלפניה יהיו אדומות, ומעליהן רמה שחורה (אי אפשר יותר מזה, אחרת תהיה לנו שרשרת של 3 צמתים אדומים). את שאר הצמתים אפשר לצבוע בכל צורה תקינה כלשהי (למשל כולם שחורים). במצב הזה, נקבל ש-$nbh(1) = 2^{h-1} + 2^{h-2} + 2^{h-3}$. משיקולים דומים, כדי ש-$nbh(2)$ יהיה מקסימלי, נצבע את הרמה האחרונה בשחור, שתי הרמות מעליה באדום וזו שמעליהן בשחור. מקבלים $nbh(2) = 2^{h-2} + 2^{h-3} + 2^{h-4}$.

טענה: בעץ מושלם בגובה h בעל $n = 2^h - 1$ צמתים, מספר הצמתים הגדול ביותר עם גובה שחור k הוא:

$$nbh(k) = \begin{cases} 2^{h-k} + 2^{h-k-1} + 2^{h-k-2} & k \leq h-2 \\ 2^{h-k} + 2^{h-k-1} = 3 & k = h-1 \\ 2^{h-k} = 1 & k = h \\ 0 & k > h \end{cases}$$

הוכחה: באינדוקציה על h. אם $h = 1$ אז יש צומת אחד בעץ שגובה השחור שלו 1. נניח שהטענה נכונה לכל $g < h$ ונראה שהיא נכונה עבור h. גובה תתי העץ השמאלי והימני של השורש הוא $h-1$ ולכן לפי הנחת האינדוקציה לכל $k \leq h-2$ בכל תת עץ יש $7 \cdot 2^{h-1-k-2}$ וביחד $7 \cdot 2^{h-1-k-2} + 7 \cdot 2^{h-1-k-2} = 7 \cdot 2^{h-k-2}$ כנדרש.

טיפלנו בכל $n = 2^h - 1$ לכל $h \geq 0$. במקרה ש-$2^{h-1} \leq n < 2^h - 1$ יש "להתעלם" מהרמה האחרונה שאינה מלאה (כאשר כל הצמתים שם צבועים באדום).

הטיפול במקרה המינימלי הוא די דומה (ומייגע באותה צורה), רק שאז נרצה כמה שפחות רמות שחורות בתחתית העץ. מקבלים:

$$nbh(k) = \begin{cases} 2^{h-1-3(k-1)} & k \leq \frac{h}{3} \\ 0 & k > \frac{h}{3} \end{cases}$$

3 א לכל צומת בעץ האלגוריתם בודק שהמפתח הנוכחי (בתוספת ערכי ה-$accum$ במסלול לשורש) נמצא בין המפתח הכי גדול בתת העץ השמאלי למפתח הכי קטן בתת העץ הימני. עץ צובר T הוא עץ חיפוש בינארי אם התנאי הנ"ל מתקיים לכל צומת בעץ. כדי שזמן הריצה יהיה לינארי ב-n, השגרה מחזירה בכל שלב אם תת העץ המושרש בצומת z הוא עץ חיפוש בינארי ואת המינימום והמקסימום בתת עץ זה. בצורה כזו אנחנו מבקרים בכל צומת פעם אחת ולכן זמן הריצה הוא $O(n)$.

```
procedure IsBST(z, a)
    if z = nil then
        return True, ∞, −∞
```

$\quad a \leftarrow a + accum[z]$
$\quad lbst, lmin, lmax \leftarrow \text{IsBST}(left[z], a)$
$\quad rbst, rmin, rmax \leftarrow \text{IsBST}(right[z], a)$
$\quad v \leftarrow key[z] + a$
$\quad$ **return** $lbst$ **and** $rbst$ **and** $lmax \leq v \leq rmin,$
$\qquad\qquad \min(lmin, rmin, v),$
$\qquad\qquad \max(lmax, rmax, v)$

וקוראים לשגרה כך: $\text{IsBST}(root[T], 0)$.

3 ב השינוי שצריך להיעשות בשגרת החיפוש הוא פשוט: כשבוחנים את המפתח של הצומת הנוכחי, יש לחבר לו את ערכי ה-$accum$ של כל הצמתים במסלול מהשורש עד לצומת הנוכחי. הקריאה ההתחלתית לשגרה תהיה עם $a = 0$.

procedure $\text{Tree-Search}(x, k, a)$
$\quad$ **if** $x =$ **nil or** $k = (key[x] + accum[x] + a)$ **then**
$\qquad$ **return** x
$\quad a \leftarrow a + accum[x]$
$\quad$ **if** $k < (key[x] + a)$ **then**
$\qquad$ **return** $\text{Tree-Search}(left[x], k, a)$
$\quad$ **else**
$\qquad$ **return** $\text{Tree-Search}(right[x], k, a)$

השינויים בהכנסה דומים לחיפוש: בחלק הראשון של השגרה כשמחפשים את המקום להכניס את הצומת z יש להתחשב בערכי ה-$accum$. לאחר שמצאנו את המקום שבו z יוכנס יש להתאים את ערך המפתח שלו בהתאם ל-$accum$ שצברנו.

procedure $\text{Tree-Insert}(T, z)$
$\quad y \leftarrow$ **nil**
$\quad x \leftarrow root[T]$
$\quad a \leftarrow 0$
$\quad$ **while** $x \neq$ **nil do**
$\qquad a \leftarrow a + accum[x]$
$\qquad y \leftarrow x$
$\qquad$ **if** $key[z] < key[x] + a$ **then**
$\qquad\quad x \leftarrow left[x]$
$\qquad$ **else**
$\qquad\quad x \leftarrow right[x]$
$\quad key[z] \leftarrow key[z] - a$
$\quad p[z] \leftarrow y$
$\quad$ **if** $y =$ **nil then**
$\qquad root[T] \leftarrow z$
$\quad$ **else**
$\qquad$ **if** $key[z] < key[y]$ **then**
$\qquad\quad left[y] \leftarrow z$

```
        else
            right[y] ← z
```

במחיקה מהסוג השני והשלישי, כל צומת בעץ שמושרש בצומת הנמחק בפועל y מאבד מערכו $accum[y]$. עלינו לתקן את העץ כדי שישאר עץ חיפוש תקין כתוצאה מכך. לשם כך ניעזר בשגרה הבאה, שבהינתן צומת כלשהו בעץ מחזירה לנו את סכום ערכי ה-$accum$ במסלול הפשוט מצומת כלשהו לשורש.

```
procedure NODE-ACCUM(z)
    a ← 0
    while z ≠ nil do
        a ← a + accum[z]
        z ← p[z]
    return a
```

השינויים היחידים לשגרה המוצגת בעמ' 221 בספר הם בשורות 12 ו-21-22. בשורה 12 אנחנו לוקחים את הבן היחיד x של הצומת שמוסר y, ומוסיפים לערך ה-$accum$ שלו את הערך של אביו, שכאמור יוסר מהעץ. בצורה הזו אנחנו משמרים על תקינות תת העץ שמושרש בצומת שמוסר y. שורות 21-22 מטפלות במקרה השלישי שבו לצומת המוסר שני ילדים. הצומת y מועתק ל-z ועל כן עלינו לתקן את ערך המפתח שלו.

```
1:  procedure TREE-DELETE(T, z)
2:      if left[z] = nil or right[z] = nil then
3:          y ← z
4:      else
5:          y ← TREE-SUCCESSOR(z)
6:      if left[z] ≠ nil then
7:          x ← left[y]
8:      else
9:          x ← right[y]
10:     if x ≠ nil then
11:         p[x] ← p[y]
12:         accum[x] ← accum[x] + accum[y]
13:     if p[y] = nil then
14:         root[T] ← x
15:     else
16:         if y = left[p[y]] then
17:             left[p[y]] ← x
18:         else
19:             right[p[y]] ← x
20:     if y ≠ z then
21:         v ← key[y]+NODE-ACCUM(y)
22:         key[z] ← v−NODE-ACCUM(z)
23:     return y
```

זמן הריצה נשאר $O(h)$ שכן שתי הקריאות ל-Node-Accum אורכות גם כן $O(h)$.

3 ג כדי ש-A ייצג עץ אדום שחור, אז כמובן צריך ש-IsBST יחזיר **True** עבור A. בנוסף, צריך שכל תכונות האדום-שחור המתוארות בעמ' 230 יתקיימו.

נותר להראות שאפשר לתמוך בפעולת הסיבוב. לשם כך נשתמש בסימונים שבאיור 13.2 בעמ' 234. כשמסובבים שמאלה y הופך להיות אבא של x. לכן נוסיף ל-$accum[y]$ את $accum[x]$ בשורה 19. בתת העץ γ אין מה לתקן. בגלל השינוי שעשינו ל-$accum[y]$ נחסיר מ-$accum[x]$ את $accum[y]$ החדש כדי להחזיר אותו לקדמותו, זה קורה בשורה 20. ב-α אין צורך לעשות שינוי ולשורש של β נוסיף את $accum[y]$ בשורה 8. הטיפול בסיבוב ימני דומה.

```
1:  procedure Left-Rotate(T, x)
2:      y ← right[x]
3:      ax ← accum[x]
4:      ay ← accum[y]
5:      right[x] ← left[y]
6:      if left[y] ≠ nil[T] then
7:          p[left[y]] ← x
8:          accum[left[y]] ← accum[left[y]] + ay
9:      p[y] ← p[x]
10:     if p[x] = nil[T] then
11:         root[T] ← y
12:     else
13:         if x = left[p[x]] then
14:             left[p[x]] ← y
15:         else
16:             right[p[x]] ← y
17:     left[y] ← x
18:     p[x] ← y
19:     accum[y] ← accum[y] + ax
20:     accum[x] ← accum[x] − ax − ay
```

4 בונים עץ אדום שחור ראשי לפי a (כלומר בשדה המפתח נשמור רק את a) לכל $k=(a,b)$. כל צומת x יכיל:

- מצביע $rb[x]$ לעץ אדום שחור משני לפי b שיכיל את כל הזוגות $(key[x], b)$.
- שדה $size[x]$ שמכיל את גודל תת העץ המשני.
- שדה $max-size[x]$ שמכיל את גודל תת העץ המשני המקסימלי בתת העץ המושרש ב-x.

תחזוק השדות עבור צומת כלשהו x: השדות $size, sum$ מתעדכנים בכל הכנסה/מחיקה מתת העץ המשני. את $max-size$ מחשבים לפי המקסימום של $size[x]$, $max-size[left[x]]$ ו-$max-size[right[x]]$ (שלושת השדות המספריים בכל צומת יאותחלו ל-$-\infty$ בעלה של העץ). לפי האמור לעיל, נסיק ממשפט 14.1 שזמני הריצה של הכנסה ומחיקה מהעץ הראשי נשארים $O(\lg n)$.

בנוסף נתחזק עוד עץ אדום שחור שיכיל את כל הסכומים השונים בעץ של זוג מפתחות, כאשר לכל צומת x נשמור גם:

- שדה $count$ שישמור כמה פעמים מופיעים בעץ הראשי והמשני זוג מפתחות מהצורה $x = a + b$.
- שדה $max - count$ ששומר את ה-$count$ המקסימלי בתת העץ המושרש ב-x.

תחזוק השדות הללו דומה לתחזוק בעץ הראשי.

בנוסף נשמור בצד שני מבציעים:

- min יצביע לצומת המינימלי בעץ הראשי.
- maj יצביע לצומת x בעץ הראשי שלו ערך $size[x]$ מקסימלי.
- $maj - sum$ יצביע לצומת x בעץ הסכומים שלו ערך $count[x]$ מקסימלי.

פעולות על המבנה:

- Build - n קריאות ל-Insert.
- Insert - מחפשים את a בעץ הראשי ומכניסים את b לעץ המשני של a (אם b קיים זורקים שגיאה). אם a לא קיים בעץ אז מכניסים אותו ויוצרים עץ משני ריק ומכניסים אליו את b. מגדילים את $size$ ב-1 ואת $max - size$ במסלול לשורש העץ הראשי כפי שהוסבר לפני כן. כל זה לוקח $O(\lg n) + O(1) = O(\lg n)$. מחפשים את $a + b$ לעץ הסכומים, אם קיים מוסיפים ל-$count$ של הצומת שנמצא 1, אחרת מוסיפים צומת חדש ומאתחלים את $count$ ל-1. $max - count$ מעודכן בצורה דומה ל-$max - size$.

  לאחר ההכנסה מעדכנים את min ל-TreeMinimum (למקרה שהצומת שהכנסנו צומת ש-a הוא המינימום החדש). את maj גם מעדכנים ע"י כך שלכל צומת x בודקים אם $size[x] = max - size[x]$, אם כן מחזירים את x. אחרת הולכים רקורסיבית לבן שלו $max - size$ גדול יותר. $maj - sum$ מעודכן בצורה דומה מעץ הסכומים.
- Delete - מחפשים את a בעץ הראשי ומוחקים את b מהעץ המשני של a. מעדכנים את כל השדות של a בדומה ל-Insert. אם ה-$size$ החדש של a הוא 0 אז מוחקים את a מהעץ הראשי. מעדכנים את $min, maj, sum - maj$ כמו ב-Insert. סה"כ זמן הריצה יוצא:

$$\overset{\text{מחיקות}}{\uparrow}O(\lg n) + \underset{\text{עדכון } max - size}{\downarrow}O(\lg n) + \overset{\text{עדכון } min}{\uparrow}O(\lg n) + \underset{\text{עדכון } maj}{\downarrow}O(\lg n) + \overset{\text{עדכון } maj - sum}{\uparrow}O(\lg n) + \underset{\text{עדכון שאר השדות}}{\downarrow}O(1) = O(\lg n)$$

- First-Min - מחזירים את $size[min]$.
- Majority-First - מחזירים את $size[maj]$.
- Majority-Sum - מחזירים את $key[maj - sum]$.

שלושת הפעולות הנ"ל מתבצעות בזמן $O(1)$.