Data Stractures ~ Homework 3 ~ *2025B*

Shahar Perets

May 11, 2025

הערות ומידע אישי

קצת לא הייתי בטוח בתרגיל הבית הזה לגבי האיזון בין פסאדו־קוד לבין הסברים במילים. אם אני מפרט יותר מדי או פחות מדי, אשמח לדעת.

כל שאלה מופיעה בעמוד נפרד. ניתן אישור במודל להגיש את התרגיל שלא על גבי הקובץ המקורי בעבור מי שמקליד על המחשב.

שם: שחר פרץ
ת.ז.: 334558962

המשך בעמוד הבא

סעיף א'

נתאר מבני נתונים התומך בפעולות על קבוצה S מתחום בעל סדר מלא, התומך בהכנסה ומחיקה ב־$O(\log n)$ וכן בפעולות min, max, median בזמן קבוע. לשם כך, ניעזר ב־Finger AVL tree השומר מצביעים לאבות, ל־successor ול־predecessor (תחזוקת מצביעים כאלו ללא שינוי זמן הריצה, תוארה בהרצאה ואפשרית).

- **תיאור מצביע:** מצביע למינימום העץ, הוא האיבר הכי השמאלי בעץ.
תחזוקת המצביע: בעת הוספת איבר, נלך למצביע ונבדוק אם הוא קטן מהמינימום. אם הוא אינו קטן מהמינימום, נמשיך כרגיל. אם הוא קטן מהמינימום, אז נעדכן את המצביע אליו ונמשיך כרגיל. במחיקת איבר, אם לא מוחקים את המינימום נמשיך כרגיל, אחרת נעביר את מצביע המינימום לאביו ונמשיך כרגיל.
- **תיאור המצביע:** מצביע למקסימום העץ, הוא האיבר הימני ביותר בעץ.
תחזוקת המצביע: באופן זהה אך הפוך לתחזוקת המינימום.
- **תיאור המצביע:** מצביע לחציון העץ, האיבר האמצעי בעת מיון האיברים.
תחזוקת המצביע: בעת הוספת איבר, אם גודל העץ המעודכן אי־זוגי לא נעשה דבר. אחרת, נבדוק את הערך במצביע: אם האיבר החדש גדול ממנו, נזיז את המצביע לחציון ל־predecessor של הצומת אליו הוא מפנה. באופן דומה, אם המספר קטן ממנו, נזיז את המצביע לחציון ל־successor של הצומת אליו הוא מפנה. במילים אחרות, אחרי שביצענו הסופה רגילה, נפעיל את הפסאודו־קוד הבא:

```
val ← newNode.value
if val mod 2 = 0 then
    if val >medianPointer.value then
        medianPointer ← medianPointer.predeccessor
    else
        medianPointer ← medianPointer.successor
    end
end
```

סעיף ב'

ניצור מבנה נתונים חדש לחלוטין, שישמור שני עצים – עץ אחד יכיל את הכל עד $\frac{n}{2}$ (עד לכדי עיגול לערך שלם) ועץ שני יכיל הכל אחרי $\frac{n}{2}$. שני העצים הללו יהיו בנויים מ־Finger Trees שמתחזקים מינימום ומקסימום, כמתואר בסעיף הקודם. נתאר את תחזוקם:

- Insert: באופן דומה לסעיף הקודם:
  - אם נוסיף לעץ הראשון עד $\frac{n}{2}$ (נוכל להכריע לאיזה עץ מוסיפים ע"פ בדיקת מינימום העץ השני) אז אם כמות האיברים זוגית נעביר את המקסימום להיות המינימום בעץ השני ונעשה רוטציות (תוספת של $O(1)$ כי יש הפניות להכל).
  - אם נוסיף לעץ השני מ־$\frac{n}{2}$ ויש בו כמות סופית של איברים, אז נעביר את המינימום להיות המקסימום בעץ הראשון ונעשה רוטציות (סה"כ $O(1)$).

  בשני המקרים ההוספה חסומה ב־$\frac{n}{2}$ גודל העץ ולכן $O(\log\frac{n}{2})$ סיבוכיות לוגירתמית.
- Delete: זהה לסעיף הקודם, רק הפוך בשינויים.
- Search: נבצע חיפוש בעץ AVL רגיל, פרט לכך שנחפש בעץ המתאים (ע"י בדיקת המינימום בעץ השני, וההבנה האם האיבר שקיבלנו אמור להיות בעץ החדש או הישן). החיפוש יתבצע מהמינימום בעץ. ראינו בהרצאה שחיפוש בעץ AVL שמתחיל מהמינימום בעץ יארח $\log i$. אך, במקרה הזה, אם $i \geq \frac{n}{2}$ אז הנ"ל יהיה נכון בעבור $i' = i - \frac{n}{2}$ וסיבוכיות $\log i'$, כי זהו עץ נפרד. סה"כ מהגדרת מודולו, וכי החיפוש לא מוגדר בעבור $i > n$, נסיק $\log\left(i \bmod \frac{n}{2}\right)$ כדרוש.

המשך בעמוד הבא

נתכנן מבנה נתונים המכיל מפתחות טבעיים ללא חזות, ותומך בפעולות Insert, Delete, Search בזמן $O(\log n)$ ובפעולה IncreaseAll(k) ב־$O(1)$, וכן בפעולה EvenLees(x) ב־$O(\log n)$ המחזירה את סכום המפתחות הזוגיים שערכם חסום ב־x.

נבנה מבנה נתונים שדה על בסיס עץ AVL. בכל צומת, נשמור שדה בשם addition.

- **Delete(x):** נבצע מחיקה רגילה ב־AVL, פרט לכך שאם $x' = x - \text{offset}$ זוגי, בדרך ל־x' נעדכן את addition בכל אחד מהצמתים בהם נפנה שמאלה להיות גדול יותר ב־$-x'$.
- **Insert(x):** נבצע הוספה רגילה ל־AVL, פרט לכך שאם $x' := x - \text{offset}$ זוגי בדרך ל־x' נעדכן את addition בכל אחד מהצמתים בהם נפנה שמאלי להיות גדול יותר ב־$+x'$.
- **IncreaseAll(k):** נשמור באיפוס המבנה משתנה בשם offset זכרון שמאופס עם ערך 0. בקריאה ל־IncreaseAll(k) נגדיל את המשתנה offeset ב־k.
- **EvenLess(x):** נעשה חיפוש בעץ בינארי למשתנה $x - \text{offset}$, אך בתחילת הרצת הפונקציה נשמור משתנה בשם sum ונאפסו ל־0 בזכרון, ובכל צומת שנעבור דרכו לבן הימני שלו, נוסיף את ה־addition שלו לסכום sum. נחזיר את sum.

פסאודו קוד של חלק מהפונקציות להבהרה, כאשר בירוק השינויים של מהמימוש הרגיל:

Insert

```
input  : T tree, x insertion to the tree
node ← T.root
x ← x − offset
while node ≠ null do
    prevNode ← node
    node.addition ← node.addition + x
    if x < node.value then
        node ← node.left
    else
        node ← node.right
    end
end
Create a new node and assign it to
 prevNode.right/left
```

EvenLess

```
input  : x supremum, T tree
output: the sum of all the even elements up to x in
        the tree T
node ← T.root
sum ← 0
x ← x − offset
while node.value ≠ x do
    if x < node.value then
        node ← node.left
    else
        x ← x + node.additionc
        node ← node.right
    end
end
return sum
```

ניתוח זמן ריצה: מחיקה והוספה נשאר בזמן ריצה זהה, פרט לתחזוק פרמטרים בזמן ריצה קבוע לצומת שבכל מקרה אנו מבקרים בה, ולכן $\Theta(\log n)$. פעולת ה־IncreaseAll תיארך זמן קבוע של חישוב סכום של שני מספרים ושמירתם בזכרון. פעולת ה־EvenLess תיארך זמן של חיפוש בינארי ועוד $\log n$ פעולות אריתמטיות, כלומר $\Theta(\log n)$ גם כן, כדרוש.

המשך בעמוד הבא

נתון עץ AVL בו מאוחסנים מפתחות טבעיים שונים זה מזה. נתון שבצומת v שמור $\text{size}(v)$, גודל תת־העץ שלו.

3.1 סעיף א'

נבחין שאם $k - \text{rank}(k) > 1$ כאשר $\text{rank}(k)$ מספר המפתחות הקטנים או השווים ל־k, משום שיש k מספרים טבעיים לפני k, אז המספר הטבעי המינימלי לא נמצא בתת העץ הזה. נקרא לבדיקה זו check1. ניעזר בה כדי למצוא את הטבעי המינימלי בסיבוכיות לוגריתמית:

```
function check1(N node) is
    if N.left = null then
        return N.value > 1                                  /* Empty case */
    end
    return N.value − N.left.size > 1
end
function FindMinNatural(T tree) is
    currentNode ← T.root
    while currentNode.right ≠ null do                       /* While not at the maximum */
        if check1(currentNode) then
            if currentNode.left = null then
                return currentNode.value − 1                /* Natural since check1 has passed */
            end
            currentNode ← currentNode .left
        end
    end
    return currentNode.value + 1
end
```

משום שבמהלך ריצת האלגוריתם, אנחנו עוברים כלפי מטה בעץ בלבד, זמן הריצה שלו יהיה חסום מלמעלה בגובה העץ הוא עץ AVL, כלומר $O(\log n)$.

3.2 סעיף ב'

עתה נממש את nextMissingAfter(i). לשם כך, נשאר עם קוד זהה לזה של הסעיף הקודם, פרט לשינוי ב־check1:

```
function check1(N node, i minimum) is
    if N.value ≤ i then
        return false
    else if N.left = null then
        return N.value > 1
    end
    return N.value − N.left.size > 1
end
```

אופן המעבר על העץ זהה לסעיף הקודם, ולכן גם כאן הסיבוכיות זהה.

המשך בעמוד הבא

נוכיח מספר טענות בעבור עצי AVL:

(א) יהי עץ AVL מגובה h. נוכיח שכל העלים בעומק של לפחות $\frac{h}{2}$.

הוכחה. נוכיח באינדוקציה על h גובה העץ. בסיס: מקרה בו שורש ובן יחיד או שורש ושני בנים. מתקיים בשניהם.
צעד: נוכיח באינדוקציה מלאה נכונות לכל $k \leq h$, נוכיח ל־$h+1$. יהי עץ T מגובה $h+1$, נתבונן בשורש – בה"כ תת־העץ הימני מגובה h, אזי גובה השני חסום מלמטה ב־$h-1$ כי $|\mathrm{BF}| \leq 1$. נסמנם T_1 ו־T_2 בהתאמה, ונסמן ב־$\mathrm{minL}(P)$ את עומק העלה הנמוך ביותר בעץ P כלשהו. אזי:

$$\mathrm{minL}(T_2) \geq \frac{h-1}{2} \wedge \mathrm{minL}(T_1) \geq \frac{h}{2} \implies \mathrm{minL}(T) \geq \min\{\mathrm{minL}(T_1),\ \mathrm{minL}(T_2)\} + \underbrace{1}_{\text{כי הם במרחק 1 מהשורש}} = \frac{h-1}{2} + 1 = \frac{h+1}{2}$$

סה"כ העלה במרחק מינימלי בעומק $\frac{h+1}{2}$, ובפרט כל העלים בעץ במרחק של לכל בעומק של לפחות $\frac{h+1}{2}$. ■

(ב) נוכיח כי לכל סדרה בת n הכנסות לעץ AVL, גם הטובה ביותר וגם הגרועה ביותר, עלותהְ $\Theta(n \log n)$

הוכחה. נוכיח חסם עליון ותחתון

חסם תחתון: בעבור העץ הסופי, נתבונן בעלי העץ, שמשום שהוא מאוזן יהיו $\Theta(\frac{n}{2})$ מאלו. מאופן ההכנסה לעץ, עבור כל אחד מהם יש צורך להשקיע $\Theta(\log n)$ פעולות בהבאתם לשם (ייתכן שבאמצעות סיבובים, וייתכן שישירות באמצעות הכנסה), זאת כי כל צומת יגיע למיקום שלו באמצעות חיפוש מהשורש או מסיבובים (שישנו עומק ב־1 על כל סיבוב). וסה"כ, נשקיע $\Omega(\frac{n}{2}) \cdot \Omega(\log n) = \Omega(n \log n)$ פעולות.

- **חסם עליון:** ראינו שלוקח זמן קבוע לאזן עץ, וכן ראינו כי החיפוש בעץ מגודל i מבוצע ב־$\log i$ זמן. כמו כן בהכנסה ה־i גודל העץ יהיה i. אזי זמן הריצה חסום מלמעלה ע"י:

$$\mathrm{cost}(op_1 \dots op_n) = O\left(\sum_{i=0}^{n-1} \log i\right) = O(n \log n)$$

■

(ג) יהי עץ AVL ויהי $m \in \mathbb{N}$ ב־AVL. בתרגול 4 הראינו שגובה העץ המתואר הוא $h = O(\log m)$, כחלק מניתוח אותו העץ. לתרגיל זה נוספה תזכורת כי עץ בעל N בגובה h, מקיים $h \leq \log_\Phi N$. הבחנו כי במקרה הזה $h = O(\log m)$, וסה"כ:

$$\log N \leq h \implies N \leq 2^h = 2^{O(\log m)}$$

עבור קבועים c, n_0 כלשהם, מתקיים:

$$\forall n \geq n_0 \colon N \leq 2^{c \log m} = 2^{c \log 2 \log m} = \underbrace{c}_{constant} m = \boldsymbol{O(m)}$$

המשך בעמוד הבא

נציע מימוש למבנה ששומר ישרים מצורת $y = ax + b$ ומכניס אותם למבנה אמ"מ לא קיים ישר במבנה החותך אותו ב־$[0, 1]$.

לצורך כך, נשמור כל צומת בתור $(b, a + b)$, במילון. אפשר להסתכל על ייצוג זה כעל ערך ה־y של הישר ב־$x = 0$ וב־$x = 1$ (שכן ערך ה־x כבר ידוע), מב"תליות הייצוג הזה קיים ויחיד לכל ישר. המילון יהיה מבוסס AVL, המסודר לפי ערכי ה־b. נתאר פעולות:

- `Search(a, b)`, כלומר בדיקה האם $y = ax + b$ נמצא במבנה – נחפש ב־$O(\log n)$ את `Dict(b)`, ואם נמצא שם $a + b$ נחזיר אמת, אחרת נחזיר שקר. הפונקציה תקינה, כי לא ייתכנו שני ישרים שונים בעלי אותו ערך b – כי אז הם נפגשים ב־$0 \in [0, 1]$.
- `Insert(a, b)`, כלומר הוספת $y = ax + b$ – גם כאן נחפש את b במבנה. אם קיים b כזה לא נעשה דבר שכן יש כבר ישר באותו המבנה. אחרת, נוכל לעצור את החיפוש כאשר נגיע ל־$b <$ b_1, ואז גם ה־predeccessor שלו b_2 מקיים $b_1 > b > b_2$. ערכם נסמן בהתאמה $b_1 + a_1, b_2 + a_2$. נבחין כי בין $y = a_1x + b_1$ ו־$y = a_1x + b_2$ אין עוד שום ישרים, משום שאחרת הישרים שבמבנה מתנגשים וזו סתירה לבניית המבנה (פורמלית, באינדוקציה, כאשר הבסיס נכון כי המבנה ריק). לכן, רק נצטרך לוודא ש־$a_2 + b_2 > a + b > a_1 + b_1$ (הערה: מקרי קצה כמו b_2 מקסימום או b_1 מינימום יטופלו ע"י השמת ערכים כגון $-\infty, +\infty$ בהתאם). ראה פירוט בפסאדו קוד.

נכסח בפסאדו־קוד: (הכוונה ב־Node(x), היא הצבת x בפונקציה המיוצגת ע"י הצומת)

```
input  : a, b real numbers
NextNode ← SearchClosest(T, b)    /* If nothing found, SearchClosest() returns the node that the
 loop was in it when it stopped.  */
func(x) ← ax + b
PrevNode ← NextNode.predeccesor
if (PrevNode(1) < func(1) < NextNode(1)) ∧ (func(0) ≠ NextNode(0)) then
 |  Insert (T, func)
end
```

החיפוש להלן בעץ יקח $O(\log n)$ ולכן אנו עומדים בדרישות הסיבוכיות.

המשך בעמוד הבא

(6)

(א) בסדרה יהיו n הכנסות, וכך או אחרת הוכחנו בהרצאה שהכנסה לכל מקום תקין בעץ AVL (שתלויה במיקום ההכנסה, ולא באיפה התחיל החיפוש), תיארך זמן קבוע. הזמן הזה לעולם לא 0 שכן תמיד ישנה עלות בלבדוק את ה־BF והאם יש צורך לתקן. סה"כ נבחין שעלויות התיקון יהיו ב־צ$\Theta(n)$, כי יהיו n הכנסות ועל כן $\Theta(n)$ תיקונים בזמן קבוע.

דרך אחרת לנסח את זה, היא שראינו שעלות תיקון amortized היא $O(1)$. מהגדרת חסם amortized, עבור רצף של n פעולות, עלות התיקון תהיה $n \cdot O(1) = O(n)$ כדרוש.

(ב) נניח שיש d_i איברים שגדולים מ־i. אזי תת־העץ המינימלי שכולל את המקסימום ואת i, יכיל לכל היותר $2d_i + 1$ צמתים (שכן כל ה־d_i־ים ימצאו בו, ועוד $\ge d_i+1$ צמתים הקטנים ממנו, כי $|\text{BF}| \le 1$). בפרט, ההגעה לתת־העץ הזה והחיפוש בו יארכו $2\log(2d_i+1)$ ומתקיים:

$$\mathscr{A}nswer = 2\log(2d_i+1) = O(\log(d_i+1)) = O(\log d_i + \log 1) = \boldsymbol{O(\log d_i)}$$

סה"כ סיבוכיות ההכנסה של i כלשהו, היא $\log d_i$. על כן, הסיבוכיות הכוללת היא:

$$\mathscr{A}nswer = O\left(\sum_{i=1}^{n} \log d_i\right) = O\left(\log \prod_{i=1}^{n} d_i\right) = \boldsymbol{O\left(\log\left(\prod_{i=1}^{n}(d_i+2)\right)\right)}$$

(ג) מא"ש הממוצעים:

$$\frac{I}{n} = \frac{\sum_{i=1}^{n} d_i}{n} = \text{AM}(d_i) \ge \text{GM}(d_i) = \sqrt[n]{\prod_{i=1}^{n} d_i}$$

$$\left(\frac{I}{n}\right)^n \ge \prod_{i=1}^{n} d_i \xleftarrow[()^n]{}$$

נציב ונקבל:

$$\mathscr{A}nswer = \underbrace{O(n)}_{\text{תיקונים}} + \underbrace{O\left(\log \prod_{i=1}^{n} d_i\right)}_{\text{הכנסות}} = O\left(n + \log\left(\frac{I}{n}\right)^n\right) = O\left(n + n\log\left(\frac{I}{n}\right)\right) \overset{(1)}{=} \boldsymbol{O\left(n\log\left(\frac{I}{n}+2\right)\right)}$$

השוויון (1) כי n זניח אסימפטוטית ביחס ל־$n\log\left(\frac{I}{n}+2\right)$ כי $\frac{I}{n}+2 \ge 1$.

שחר פרץ, 2025

קומפל ב־LaTeX ונוצר באמצעות תוכנה חופשית בלבד