# מבנה נתונים – תרגיל 3

שאלה 1

א. נגדיר $d(h)$ - עומק מינימלי של עלה בעץ avl בגובה h. נשים לב, כי בהכרח קיים תת עץ של השורש בעל גובה h - 1. לתת עץ השני, קיימות 2 אפשרויות לגובהו: h - 1 או h - 2 (נובע ישירות מהתכונה של עץ avl). כמו כן, עבור $d(0) = d(1) = 0$. לכן $d(h) = min(d(h-1), d(h-2)) + 1$ כאשר ה- 1 מתייחס להוספת הקשת מהשורש אל תת העץ המושרש הרלוונטי. לכן נקבל כי הנוסחה לחישוב עומק מינימלי של עלה בעץ avl הינה $d(h) = d(h-2) + 1$. נוכיח באינדוקציה כי עבור עץ avl בגובה h, עומק מינימלי של עלה הוא לכל הפחות $\left\lceil \frac{h}{2} \right\rceil$:

מקרה בסיס: h = 2-

$d(2) = d(0) + 1 = 1$. כמו כן, $\left\lceil \frac{2}{2} \right\rceil = 1$ ונקבל כי $d(2) = 1 \geq 1$.

הנחת האינדוקציה: נניח שעבור כל עץ avl בעל גובה t כך ש- h > t, $d(t) \geq \left\lceil \frac{t}{2} \right\rceil$.

צעד האינדוקציה: נוכיח נכונות עבור t = h:

$d(h) = d(h-2) + 1$. כמו כן, h > h-2 ולכן מהנחת האינדוקציה נובע כי $d(h-2) \geq \left\lceil \frac{h-2}{2} \right\rceil$. כמו כן, לכל n שלם מתקיים כי $\lceil x + n \rceil = \lceil x \rceil + n$. נציב באי שיוויון ונקבל:

$$d(h) = d(h-2) + 1 \geq \left\lceil \frac{h-2}{2} \right\rceil + 1 = \left\lceil \frac{h-2}{2} + 1 \right\rceil = \left\lceil \frac{h}{2} \right\rceil.$$

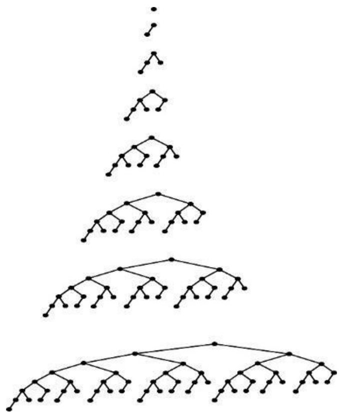
כלומר, קיבלנו כי $d(h) \geq \left\lceil \frac{h}{2} \right\rceil$. החסם הדוק עבור עצים מושלמים. עץ בינארי מושלם הוא עץ בינארי מלא בו כל העלים באותה רמה, כלומר זה עץ בו לכל צומת שאינו עלה יש 2 בנים, ומכיוון שכל העלים באותה רמה אז לכל צומת פנימית x (כולל השורש) $x.left - x.right = 0$ ולכן מדובר בעץ AVL. החסם הוא הדוק כי ראינו שעבור כל עלה בעץ AVL, עומק מינימלי של קודקוד הוא לכל הפחות $\left\lceil \frac{h}{2} \right\rceil$ ומכיוון שכל העלים באותה רמה אז העומק עבור כל עלה הוא לכל היותר $\left\lceil \frac{h}{2} \right\rceil$.

ב.

נסתכל על משפחת עצי ה- AVL המינימליים, כלומר, בהינתן גובה h נסתכל על עץ ה- AVL שנוצר ממינימום קודקודים לפי הנוסחה:

$$n(h) = n(h-1) + n(h-2) + 1,\ h(0) = 1,\ h(1) = 2$$

נשים לב כי העצים הנוצרים בהתאם לנוסחה זו הינם עצי פיבונאצ'י כאשר עבור כל צומת x (פרט לעלים) מתקיים כי $x.left - x.right = 1$:



בסעיף א' הוכחנו כי כל עלה בעץ avl חסום מלמטה ע"י $\left\lceil \frac{h}{2} \right\rceil$, ולכן נרצה לבצע פעולת מחיקה על עלה בעומק מינימלי הנמצא בתת עץ ימני כדי שנצטרך לעשות מספר מקסימלי של רוטציות, הרי מתכונת עץ avl ומבניית משפחת העצים המינימליים, במחיקת עלה בעומק מינימלי בהכרח הופר האיזון ולכן במקרה הגרוע ביותר בעצים מסוג זה נצטרך לבצע רוטציה בכל רמה שבה ירדנו, ומכיוון שמחקנו עלה בעומק מינימלי ירדנו $\left\lceil \frac{\log(|T_n|)}{2} \right\rceil = \left\lceil \frac{h}{2} \right\rceil$ רמות. כמו כן,

$$\frac{\log(|T_n|)}{2} \geq \left\lceil \frac{\log(|T_n|)}{2} \right\rceil$$

ולכן נגדיר כי $c = \frac{1}{2}$ ונקבל כי פעולת המחיקה עבור עלה זה לקחה לכל הפחות $\frac{1}{2} \cdot \log(|T_n|)$ רוטציות.

ג.

בהינתן עץ AVL ריק נתחזק כל קודקוד שמכניסים לעץ או מסירים מהעץ עם עוד 2 שדות next, prev.

next הוא האיבר הבא הגדול מאותו קודקוד ו-prev הוא האיבר הקטן המקסימלי שקיים בעץ.

כאשר מכניסים קודקוד לעץ השדות next ו -prev מאותחלים לnull, ולאחר מכן מכניס אותו בהכנסה רגילה לעץ AVL. בכל שלב של **ההכנסה**(מהשורש) בודקים :

- אם המפתח של הקודקוד הנוכחי גדול יותר מהמפתח שאני מכניס אז נגדיר את next שלו להיות אותו קודקוד.
- אם המפתח של הקודקוד הנוכחי קטן יותר מהמפתח שאני מכניס אז נגדיר את prev שלו להיות אותו קודקוד.

ובכך יורדים בהתאם עד למקום הנכון להכנסה כעלה.

לבסוף מעדכנים את ה-prev של ה-next של אותו עלה שהכנסנו להיות העלה הזה וגם את הnext של ה-prev של הקודקוד שהכנסנו להיות הקודקוד שהכנסנו (כמובן לאחר בדיקה שהוא לא null).

נשים לב שבכל האיטרציה מהשורש ועד לעלה אנו מוסיפים מספר פעולות קבועות של עדכון מצביעים ובמקרה הגרוע ביותר נגיע ל-log(n) פעולות. לכן פעולת insert של העץ נשמרת והיא O(log(n)).

כאשר נרצה **למחוק** קודקוד מהעץ – נחפש את האיבר O(log(n)) ופשוט נעדכן את המצביעים של next ו-prev שלו כמו מחיקה מרשימה מקושרת דו כיוונים(פעולות קבועות ולכן O(1)). לכן זמן הריצה של ההכנסה לעץ AVL לא השתנה כלל.

בפעולת Find(x,k) נרצה למצוא את האיבר עם מפתח x בעץ. לאחר שמצאנו אותו נרוץ באיטרציה כל עוד next של הקודקוד קטן מהערך k.

נשים לב שבדרך זו ניתן להסתכל על המעבר next.key של כל קודקוד כרשימה דו כיוונית ממוינת.
לכן זמן הריצה הכולל הוא O(log(n)) עבור החיפוש ו-O(k) עבור האיטרציה שעוברת על כל k האיברים הגדולים מ-x, כלומר O(log(n)+k).

## שאלה 2

נשתמש במערך A בגודל 10 אשר כל אינדקס במערך מייצג את כל אחד מאברי הקבוצה S {1,2,...,10}.
בנוסף כל תא במערך יצביע לעץ AVL שיכיל את המפתחות המזוהים עם אותו איבר (אינדקס).

**Init ():**

נאתחל מערך בגודל קבוע של 10 – O(1), ועבור כל תא במערך ניצור עץ AVL ריק – O(1), לכן סך הכל נקבל זמן ריצה של O(1).

**Add(k,S) :**

מנתוני השאלה הקבוצה S מיוצגת באמצעות רשימה מקושרת. נשים לב כי הגודל המקסימלי של S יכול להיות 10, ולכן בהכנסת איבר למבנה הנתונים, נעבור על כל הרשימה (גודל קבוע-O(1)), ועבור כל איבר x ברשימה S נעדכן:
אם A[x] = 0 אז לא קיימת קבוצה במבנה הנתונים שהאיבר A[x] שייך אליה ולכן נעדכן A[x] = 1 -O(1), וגם נעדכן שהשדה key של השורש הוא k- כי במקרה זה העץ ריק(זאת אומרת נבצע הכנסה של k לתוך העץ AVL שאליו A[x] מצביע-O(log(n))).
אם A[x] =1 אז קיימת קבוצה במבנה הנתונים שמכילה את האיבר A[x] עם מפתח $k' \neq k$ ולכן נבצע הכנסה של k, הכנסה רגילה לתוך העץ AVL שאליו A[x] מצביע-O(log(n)).
סך הכל זמן הריצה הינו O(log(n)).

**Delete(k):**

נעבור על כל המערך A החל מהמקום הראשון (O(1)), ועבור כל תא נבצע חיפוש רגיל של k בתוך העץ AVL שאליו התא הנוכחי מצביע (O(log(n))).
אם תוצאת החיפוש תחזיר null אז המפתח k לא מזוהה עם האיבר הנוכחי (התא הנוכחי) ולכן לא נצטרך למחוק אותו מהעץ.
אחרת, קיימת קבוצה במבנה הנתונים שמכילה את האיבר הנוכחי(התא הנוכחי) ומזוהה עם המפתח k ולכן נצטרך לבצע מחיקה רגילה של k מהעץ AVL שהתא הנוכחי מצביע אליו-O(log(n)).
נשים לב כי על מנת למחוק את כל הקבוצה שמזוהה עם המפתח k יש לעבור על כל התאים הרלוונטיים במערך A שמייצגים את האיברים בקבוצה S ולמחוק את k מכל אחד מהעצים שלהם במידה וקיים שם.
סך הכל נקבל במקרה הגרוע הוא כאשר הקובצה {1,2,3...10} מזוהה עם k – O(10log(n)) כלומר O(log(n)).

**First(j,k) :**

בהניתן j, ניגש לתא במערך במקום ה- A[j] וכך נגיע לעץ שאליו קיים מצביע בתא הזה ונבצע חיפוש רגיל של k בעץ AVL– O(log(n)) .
מהלך החיפוש יתבצע באופן הבא:
אם מפתח הקודקוד הנוכחי בו אנו נמצאים שווה ל-k נחזיר את k.
אם מפתח הקודקוד הנוכחי בו אנו נמצאים גדול מ-k נבדוק : אם אין לו בן שמאלי אז נחזיר את מפתח הקודקוד הנוכחי, אחרת נמשיך את החיפוש בצורה רקורסיבית בתת העץ השמאלית.
אם מפתח הקודקוד הנוכחי בו אנו נמצאים קטן מ-k נבדוק : אם אין לו בן ימני אז נחזיר את מפתח הקודקוד הנוכחי, אחרת נמשיך את החיפוש בצורה רקורסיבית בתת העץ הימני.
אלגוריתם החיפוש זהה לחיפוש של עץ AVL ולכן סך הכל נקבל זמן ריצה של O(log(n)).

שאלה 3

א.

נתונים 2 עצי AVL: $T_1, T_2$ עם גבהים $h_1, h_2$ כך שכל מפתח בעץ $T_2$ גדול ממש מכל מפתח ב-$T_1$.

בנוסף נתון מפתח x כך שמתקיים : $maxT_1 < x < minT_2$.

בעץ AVL כל קודקוד מכיל שדה height עם ערך הגובה שלו, וגם מספר המייצג אינדיקציה האם הצומת מאוזן או לא.

נחלק ל2 מקרים :

1. $h_1 > h_2$ :

נעבור באיטרציה על כל הקודקודים הנמצאים במסלול הימני ביותר ב- $T_1$ כאשר מתחילים מהשורש. בכל קודקוד נבדוק האם הגובה שלו שווה $h_2$ ($O(h_1 - h_2)$). ברגע שנגיע לקודקוד(נסמנו ב-*y*) הנתון נבצע את הפעולות הבאות :

- נייצר קודקוד עם מפתח *x* הנתון.
- נעדכן מצביעים *O(1)*:

$x.left = y$ – הבן השמאלי של הקודקוד הנתון יהיה הקודקוד שגובהו $h_2$.

$x.right = T_2.root$ – הבן הימני של הקודקוד הנתון יהיה כל העץ $T_2$ (נשים לב שעבור כל צומת ב $T_2$ הוא מאוזן וגובהו $h_2$.

$y.parent.right = x$ – הבן הימני של האבא של הקודקוד שמצאנו שגובהו $h_2$ יצביע על הקודקוד הנתון *x*.

$y.parent = x$ – האבא החדש של הקודקוד שגובהו $h_2$ שמצאנו יהיה *x*.

$T_2.root.parent = x$ – האבא החדש של כל העץ $T_2$ יהיה *x*.

נשים לב כי תת העץ הימני של *x* הוא בעצם $T_2$ , ו$T_2$ כבר עץ *AVL* מאוזן וגובהו $h_2$, ותת העץ השמאלי של *x* הוא גם בגובה $h_2$ מאחר והקודקוד *y* שמצאנו היה בגובה זה, לכן אם יהיו צמתים לא מאוזנים בעץ הם יכולים להיות רק מ-*x.parent* ועד השורש של העץ במקרה הגרוע ביותר.

- עבור כל קודקוד (אב קדמון של *x*) ועד השורש, נבדוק האם הוא מאוזן. אם הגענו לצומת שלא מאוזן נבצע סיבובים(רוטציות-*O(1)*) כפי שנלמד בכיתה לפי ארבעת המקרים הרלוונטיים.

סך הכל קיבלנו פעולות קבועות של *O(1)* וגם פעולת החיפוש של קודקוד *y*, כלומר $O(h_1 - h_2)$.

2. $h_1 \le h_2$:

מקרה זה סימטרי עבור $h_1 < h_2$ , כלומר :

נעבור באיטרציה על כל הקודקודים הנמצאים במסלול השמאלי ביותר ב- $T_2$ כאשר מתחילים מהשורש. בכל קודקוד נבדוק האם הגובה שלו שווה $h_1$ ($O(h_1 - h_2)$). ברגע שנגיע לקודקוד(נסמנו ב-*y*) הנתון נבצע את הפעולות הבאות :

- נייצר קודקוד עם מפתח *x* הנתון.
- נעדכן מצביעים *O(1)*:

$x.left = T_1.root$ – הבן השמאלי של הקודקוד הנתון יהיה כל העץ $T_1$ (נשים לב שעבור כל צומת ב $T_1$ הוא מאוזן וגובהו $h_1$.

$x.right = y$ – הבן הימני של הקודקוד הנתון יהיה הקודקוד שגובהו $h_1$.

$y.parent.left = x$ – הבן השמאלי של האבא של הקודקוד שמצאנו שגובהו $h_1$ יצביע על הקודקוד הנתון *x*.

$y.parent = x$ – האבא החדש של הקודקוד שגובהו $h_1$ שמצאנו יהיה *x*.

$T_1.root.parent = x$ – האבא החדש של כל העץ $T_1$ יהיה *x*.

בדומה למקרה הקודם, תת העץ השמאלי של *x* הוא בעצם $T_1$ והוא מאוזן, ותת העץ הימני של *x* הוא גם בגובה $h_1$ לכן גם כאן המקרה הגרוע ביותר של איזון העץ זהה.

- עבור כל קודקוד (אב קדמון של *x*) ועד השורש, נבדוק האם הוא מאוזן. אם הגענו לצומת שלא מאוזן נבצע סיבובים(רוטציות-*O(1)*) כפי שנלמד בכיתה לפי ארבעת המקרים הרלוונטיים.

סך הכל קיבלנו פעולות קבועות של *O(1)* וגם פעולת החיפוש של קודקוד *y*, כלומר $O(h_2 - h_1)$.

עבור המקרה בו $h_1 = h_2$*:* נייצר קודקוד שהמפתח שלו הוא *x*, ונעדכן מצביעים:

$T_1.root.parent = x$

$x.left = T_1.root$

$T_2.root.parent = x$

$x.right = T_2.root$

כלומר הקודקוד *x* משמש כקודקוד מתווך שגובה תת העץ הימני שלו שווה לתת העץ השמאלי שלו ולכן מאוזן. פעולה זו במקרה זה היא *O(1)*.

עבור כל המקרים ביחד קיבלנו $O(|h_1 - h_2|)$.

ב'

בהינתן עץ AVL ומפתח k תחילה נבדוק האם k נמצא בעץ. אם k לא נמצא בעץ אז נכניס אותו לעץ ובסוף נמחק אותו מהעץ שמכיל את כל האיברים הקטנים או שווים ל-k. סה"כ זמן הריצה $O(3\log(n)) = O(\log(n))$.

האלגוריתם: תוך כדי חיפוש בצורה רקורסיבית בעץ אחר מפתח k, נרצה לפרק את הבעיה לתתי בעיות כאשר כל מה שקטן או שווה ל- k יפורק לעץ אחד וכל מה שגדול מ- k יפורק לעץ שני, ולאחר מכן בחזרה מהרקורסיה נרצה לחבר את כל תתי העצים עם הרלוונטיים להם: עץ אחד יכיל את כל המפתחות הקטנים או שווים ל- k ועץ שני יכיל את כל המפתחות שגדולים ממש מ- k.

בקריאה לפונקציה Split(T,k) המקבלת עץ AVL ומפתח k ומחזירה $T_1, T_2$ תוך כדי שמירה על התכונה בה כל המפתחות ב- $T_1$ קטנים ממש מכל המפתחות ב- $T_2$, נרצה לעבור על העץ T בצורה רקורסיבית עד שנגיע ל-k:

**מקרה 1-** אם העץ ריק, נחזיר שני עצים ריקים. נקבל כי זמן הריצה במקרה זה הוא O(1).

**מקרה 2- המפתח של שורש העץ שווה למפתח k:**

אם לקודקוד זה (השורש) אין בן שמאלי ויש בן ימני אזי נגדיר כי $T_1$ הינו העץ שמכיל את השורש בלבד ונגדיר את $T_2$ להיות תת עץ הימני של השורש. באופן דומה, אם לשורש יש בן שמאלי ואין בן ימני אזי נגדיר כי $T_1$ הינו העץ שמכיל את השורש ואת בנו השמאלי וכי $T_2$ הינו עץ ריק.

אחרת, אם לשורש יש שני בנים נרצה להגדיר כי $T_1$ הינו תת העץ השמאלי של השורש ונבצע הכנסה של השורש לתוך עץ זה, ואילו נגדיר את $T_2$ להיות תת העץ הימני של השורש. נקבל כי זמן הריצה במקרה זה הוא $O(\log(n))$.

**מקרה 3– המפתח בשורש העץ גדול מהמפתח k:**

אם לקודקוד הנוכחי (השורש) אין בן שמאלי נגדיר כי $T_1$ הינו עץ ריק ו- $T_2$ הינו העץ המקורי T. אחרת, נקרא לפונקציה Split(T.left,k) בצורה רקורסיבית עם תת העץ השמאלי של T הנוכחי ונקבל שני עצים: $T_{small}$ (יכיל את כל הערכים הקטנים או שווים ל- k) ו- $T_{big}$ (יכיל את כל הערכים שגדולים ממש מ- k). בחזרה מהקריאה הרקורסיבית נרצה לאחד את העצים כמו שאיחדנו בסעיף א' עם הפונקציה $Merge(T_{big}, T.right, x)$ כאשר $x = T.root.key$. נגדיר כי $T_2$ הינו העץ המתקבל מהאיחוד:

$T_2 = Merge(T_{big}, T.right, x)$, כלומר, נרצה לאחד את תת העץ הימני של העץ הנוכחי T עם העץ שמכיל את המפתחות שגדולים ממש מ- k שחזר מהפונקציה הרקורסיבית, כלומר, $T_{big}$.

לאחר מכן נחזיר את $T_1$ ואת $T_2$.

**מקרה 4- המפתח בשורש העץ קטן מהמפתח k:**

אם לקודקוד הנוכחי(השורש) אין בן ימני נגדיר כי $T_1$ הינו העץ המקורי T וכי $T_2$ הינו עץ ריק. אחרת נקרא לפונקציה Split(T.right,k) בצורה רקורסיבית עם תת-העץ הימני של T הנוכחי ונקבל שני עצים כמו במקרה 3. בחזרה מהקריאה הרקורסיבית נרצה לאחד את העצים כמו שאיחדנו בסעיף א' עם הפונקציה $Merge(T_{small}, T.left, x)$ כאשר $x = T.root.key$. נגדיר כי $T_1$ הינו העץ המתקבל מהאיחוד:

$T_1 = Merge(T_{small}, T.left, x)$, כלומר נרצה לאחד את תת העץ השמאלי של העץ הנוכחי T עם העץ שמכיל את המפתחות הקטנים או שווים ל- k שחזר מהפונקציה הרקורסיבית, כלומר, $T_{small}$.

לאחר מכן נחזיר את $T_1$ ואת $T_2$.

נראה כי זמן הריצה בשני המקרים 3 ו- 4 חסום ע"י גובה העץ:

נגדיר כי h הינו גובה העץ T. נסתכל על תתי העצים עליהם נבצע פעולת merge במהלך האלגוריתם כאשר נפנה משמאל לשורש או מימינו:

$A_1, A_2, ..., A_h$ כאשר $1 \le i \le h$, וכל $A_i$ מיוצג ע"י הגובה שלו. (כלומר, גובה העץ נע בין 1 ל- h וכך גם מספר תתי העצים).

זמן הריצה של פונקציית merge הממזגת בין שני עצים הוא $O(|h_1 - h_2|)$. כמו כן, נשים לב כי זמן הריצה של פונקציה זו חסום מלמעלה ע"י הגובה המקסימלי מבין שני גבהי העצים עליהם נפעיל את הפונקציה. עבור i תתי העצים, תחילה נרצה למזג בין $A_1, A_2$ ולאחר מכן נרצה למזג בין העץ

שהתקבל מהמיזוג של $A_1$ ו$A_2$ (נסמנו $A_2'$) לבין $A_3$, לאחר מכן נרצה למזג בין $A_4, A_3'$ וכך הלאה, כלומר, עבור כל עץ נרצה למזג אותו עם תת העץ שהתקבל ממיזוג קודם. נתבונן זמן הריצה של פעולות המיזוג:

$$\underbrace{O\left(h(A_2) - h(A_1)\right)}_{O\left(max\left(h(A_2),h(A_1)\right)\right) = O\left(h(A_2)\right) = O\left(h(A_2')\right)} + \underbrace{O\left(h(A_3) - h(A_2')\right)}_{O\left(max\left(h(A_3),h(A_2')\right)\right) = O\left(h(A_3)\right) = O\left(h(A_3')\right)} + \underbrace{O\left(h(A_4) - h(A_3)'\right)}_{O\left(max\left(h(A_4),h(A_3')\right)\right) = O\left(h(A_4)\right) = O\left(h(A_4)\right)} + \ldots$$

$$O\left(h(A_h) - h(A_{h-1})'\right) = O\left(h(A_2)\right) + O\left(h(A_3)\right) - O\left(h(A_2)\right) + O\left(h(A_4)\right) - O\left(h(A_3)\right) + \ldots +$$

$$O\left(h(A_h)\right) - O\left(h(A_{h-1})\right) = O\left(h(A_h)\right) = O(h) = O(\log(n))$$

כלומר, נשים לב כי מדובר בטור טלסקופי וכל איבר מתבטל ע"י הקודם לו ולכן נישאר עם העץ בעל גובה *h* מקסימלי שלא קיים איבר לפניו שיבטל אותו.

ג.

נשתמש בשני עצי AVL:

$T_1$ - עץ שיציין שיש להפוך את הצבעים של כל האיברים בו.

$T_2$ - עץ שיציין שאין להפוך את הצבעים של כל האיברים בו.

בנוסף, עבור $T_1$ נתחזק ערך max, ועבור $T_2$ נתחזק ערך min, וכמו כן עבור שני העצים נוסיף שדה change שאם הוא זוגי אזי לא צריך להחליף את הצבע, ואם הוא אי זוגי אזי צריך לשנות את הצבע. נאתחל את השדה הזה בשני העצים ב- 0.

## **ADD(X):**

במקרה בו שני העצים ריקים, **נכניס** את x ל- $T_2$ כאשר האיברים בשני העצים יוכנסו לפי שדה ה-key שלהם. (O(log(n)).

מכיוון שבמהלך האלגוריתם נשתמש בפונקציות מסעיפים א' ב' (merge, split) עלינו לשמור על התכונה כי ברגע שנרצה לאחד שני עצים, כל המפתחות בעץ אחד קטנים ממש מהמפתחות בעץ השני, ולכן אם נרצה להכניס את x למבנה הנתונים וקיימת חלוקה של שני עצים כך שכל האיברים בעץ $T_1$ קטנים ממש מהאיברים בעץ $T_2$ וגם x קטן מכל האיברים בעץ $T_2$, אזי נכניס את x לעץ $T_1$ עם הצבע ההפוך לו, וכך אם נרצה להחזיר את הצבע של האיבר x, נראה כי הוא נמצא בעץ שיש להפוך את הצבעים שלו ולכן נשנה את הצבע שלו וכך נחזיר את צבעו המקורי.

## **COLOR(K):**

בהינתן k, נבצע **חיפוש** (O(log(n))) שלו בשני העצים ונחלק למקרים:

אם k לא נמצא בשני העצים- נחזיר null.

אחרת, בכל קריאה לפונקציה הזו, נבצע את השלבים הבאים:

אם האיבר שאנו מחפשים נמצא ב- $T_1$ ולכן יש להפוך את הצבע של האיבר המזוהה עם המפתח k, ולכן נשתמש בפונקציית עזר שמקבלת קודקוד ומחליפה את השדה color שלו (אם הצבע של הקודקוד הינו שחור, נחליף ללבן וההפך- (O(1)).

אם האיבר שאנו מחפשים נמצא ב- $T_2$, הצבע שלו נשאר כמו שהוא ופשוט נחזיר את השדה color שלו.

## **FlipColor(K):**

בפעם הראשונה בה נפעיל את הפונקציה הנ"ל, כל האיברים שלנו נמצאים בעץ אחד- $T_2$. בהינתן k, נפעיל את הפונקציה **split** מסעיף ב'(O(log(n)). לאחר מכן, קיבלנו שני עצי avl- $T_1$ - המכיל את כל האיברים שקטנים או שווים ל- k ו- $T_2$ - שמכיל את כל האיברים שגדולים ממש מ- k. $T_1$ זה העץ בו צריך להחליף את הצבע לכל האיברים בו, ולכן נשנה את השדה change שלו ל- 1, ואילו $T_2$ זה העץ שלא צריך לעשות בו שינוי ולכן נשאיר את השדה שלו בתור 0. נעדכן את ערכי min ו- max עבור שני העצים. בפעם הבאה שבה נפעיל את הפונקציה, בהינתן k, נרצה לדעת באיזה עץ להפעיל את הפונקציה split ולכן נחלק למקרים:

- אם $k \le T_1.max$, נפעיל את הפונקציה split בעץ $T_1$ ונקבל שני עצי avl: $T_{1_2}$ - מכיל את כל האיברים שקטנים או שווים ל- k, ו- $T_{1_1}$ - מכיל את כל האיברים שגדולים ממש מ- k.
נשים לב כי על העץ $T_{1_2}$ כעת נעשה מספר זוגי של פעולות של שינוי צבע, כלומר, כעת לא צריך לשנות את הצבע של האיברים בעץ זה ולכן נפעיל את פונקציית **merge** $(O(|h_{1_2} - h_2|))$ עם k, $T_2$ ו- $T_{1_2}$. העץ שיתקבל מהאיחוד הוא העץ $T_2$ בעל שדה change 0, ואילו $T_{1_1}$ יהיה העץ $T_1$ בעל שדה change 1.
- אם k גדול ממש מהערך max של $T_1$, נחפש את k בעץ $T_2$:

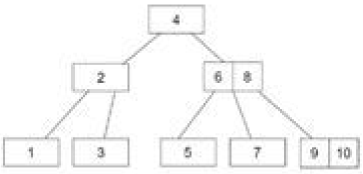
- אם כל האיברים בעץ $T_2$ גדולים מ k -אזי נפעיל את הפונקציה **merge**$(O(|h_1 - h_2|))$ עם $T_1$ k, ו - $T_2$והעץ שנקבל מהאיחוד הינו $T_2$ עם השדה change = 0 ו $T_1$ -יהיה עץ ריק.
- אם k גדול מ- $T_2.,max$ אז נעדכן את השדה change של כל עץ להיות ההופכי ממה שהיה לו ואז $T_2$ יהיה $T_1$ ו-$T_1$ יהיה $T_2$ .
-אם לא כל האיברים בעץ $T_2$ גדולים ממש מ k, -נפעיל את הפונקציה **split** ונקבל שני תתי עצים בעץ $T_2$ : $T_{2_1}$:מכיל את כל האיברים שקטנים או שווים ל k, ו- $T_{2_2}$-מכיל את כל האיברים שגדולים ממש מ k. -נשים לב, כי כל האיברים בעץ $T_{2_2}$ הינם איברים שלא צריך לשנות את צבעם והאיברים בעץ $T_1$ הינם איברים שנעשו עליהם מספר זוגי של שינוי צבע מכיוון שכל האיברים בעץ זה הינם קטנים מ k -ולכן נפעיל את הפונקציה merge עם $T_{2_2}$, k ו $T_1$. -לאחר פעולת ה merge -נגדיר כי העץ שהתקבל הינו $T_2$ עם שדה change = 0, והעץ $T_{2_1}$ הינו העץ $T_1$ עם השדה change = 1.
לאחר מכן, נעדכן את השדות min ו max -עבור שני העצים החדשים שנוצרו.
נשים לב כי זמני הריצה פה הם עבור חיפוש, *split* ו-*merge* כאשר כל אחת מהפונקציות חסומה ע"י זמני ריצה של $O(\log(n))$, $O(\log(n))$, $(O(|h_1 - h_2|))$ בהתאמה, ולכן הפונקציה כולה סדר גודל של ***O(log(n))***.

4.
א'
נשים לב שהאלגוריתם **הלא-עצלני** הינו אלגוריתם שנלמד בכיתה שזמני הריצה שלו **במקרה הגרוע** ביותר למחיקה הוא $\boldsymbol{O((\log_t n)\cdot t)}$ **ובמקרה הטוב** ביותר נרצה לחפש איבר שנמצא ראשון בתוך השורש המכיל לפחות $t$ מפתחות ולכן לפי אלגוריתם זה תמיד נרצה למחוק איבר כעלה ולכן נוריד את האיבר עד שנגיע לעלה, כלומר כגובה העץ (מאחר ואמרו להניח כי קודקוד מיוצג כמערך אז נצטרך להזיז אינדקס את כל האיברים שמאלה ב-1 ולכן $\boldsymbol{O(\log_t n)\cdot t}$.

באלגוריתם העצלני נשים לב שבמהלך המחיקה לא צריכה להתבצע הכנה של הקודקודים בעץ.
במקרה בו $k$ נמצא בעלה המכיל בדיוק *t-1* מפתחות אז לאחר המחיקה נבצע *merge* -*O(t)* עם האבא והאח שלו(שמאלי או ימני), ונבצע פעולה זו באופן רקורסיבי עד לשורש תוך כדי שמירה על תכונות של *B-tree*-(זמן כולל של $O(\log_t n)\cdot t$). אם העלה מכיל לפחות $t$ מפתחות אז רק נמחק את האיבר $k$ מהקודקוד(זמן כולל של חיפוש רגיל בעץ + מחיקה בזמן ריצה קבוע –$O(\log_t n)\cdot t$).
במקרה בו $k$ נמצא בקודקוד פנימי (כולל שורש), נגדיר כי *k=k'* כך ש*k'* הוא העוקב שלו(נגיע לעוקב של $k$ באמצעות המצביע שלו לקודקוד שבוא נמצא העוקב).
*k'* בהכרח יהיה עלה ולכן נבצע מחיקה כמו שמחקנו במקרה לעיל.
**במקרה הגרוע ביותר** נרצה למחוק מעלה שמכיל בדיוק *t-1* מפתחות, ואז נצטרך לבצע *merge* ו-*shift* -*O(t)* מאותו עלה ועד השורש, כלומר סך הכל נקבל זמן ריצה כולל של חיפוש האיבר + תיקונים - $\boldsymbol{O(\log_t n)\cdot t}$.
**במקרה הטוב ביותר** נרצה למחוק מפתח שנמצא כאיבר במקום הראשון בשורש עם לפחות $t$ מפתחות וגם שהעלה שנמצא בו העוקב שלו מכיל לפחות $t$ מפתחות – נגדיר *k=k'* ואז נמחק את העוקב מהעלה, (מאחר ואמרו להניח כי קודקוד מיוצג כמערך אז נצטרך להזיז אינדקס את כל האיברים שמאלה ב-1 ולכן – *O(t)*). לכן זמן הכולל במקרה הטוב ביותר הוא ***O(t)***.

ב'

העץ המתקבל מהכנסת האיברים $1, 2, \ldots, 10$ עם $t = 2$ הוא

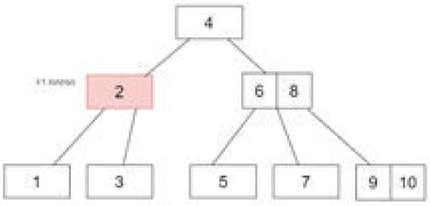



.

נתאר בשלבים את העץ המתקבל ממחיקת האיבר 2 באמצעות המימוש הלא עצלני (כפי שנלמד בכיתה):

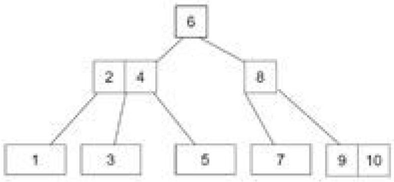
**<u>שלב א':</u>**

נחפש את 2 בעץ. לאחר שמצאנו אותו, נבדוק מה מספר המפתחות בקודקוד בו הוא נמצא ונראה כי מספר המפתחות הוא t-1 ולכן נצטרך לבצע 'תיקון'.



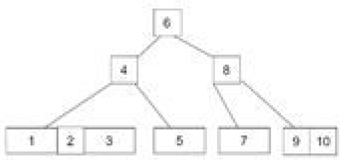

**<u>שלב ב':</u>**

נשים לב כי אי אפשר להלוות מפתחות מהבנים של 2 מכיוון שכל אחד מהם מכיל t-1 מפתחות ולכן נסתכל על האח הימני של 2 ונראה כי הוא מכיל t מפתחות ולכן ניקח משם מפתח (ניקח את 6), נעלה אותו לאבא של 2 ו-6, ונוריד את המפתח 4 שנמצא באבא של 2 לקודקוד שבו 2 נמצא, כלומר, נעשה shift ל-6 וshift ל- 4:



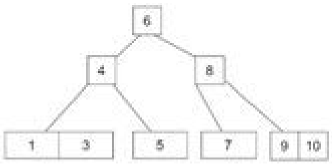

**<u>שלב ג':</u>**

ב- B tree תמיד נרצה למחוק איבר כעלה, ולכן נרצה להוריד את 2 למטה: נעשה shift ל-2 כך שיכנס לקודקוד בו נמצא המפתח 3 ולאחר מכן נעשה merge לקודקוד הזה עם הקודקוד שבו נמצא המפתח 1:

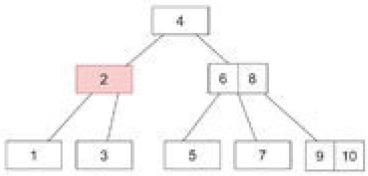



**<u>שלב ד':</u>**

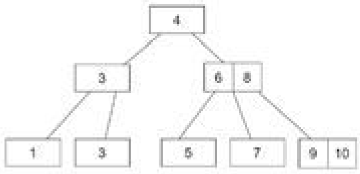
כעת נרצה למחוק את 2 כעלה ומכיוון ש-2 נמצא בקודקוד בו יש לפחות t מפתחות, נוכל למחוק בצורה רגילה ולעדכן כי הבן השמאלי של 4 הינו קודקוד בו מכיל את המפתחות 1 ו- 3 והבן הימני זה הקודקוד המכיל את המפתח 5:



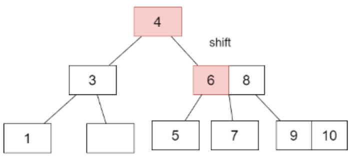
כעת, נתאר את העץ המתקבל ממחיקת האיבר 2 באמצעות המימוש העצלני:
**<u>שלב א':</u>**
נחפש את הקודקוד בו האיבר 2 נמצא בעץ.



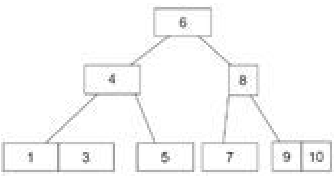

**<u>שלב ב':</u>**
לאחר שמצאנו את 2, נרצה למחוק אותו כמו מחיקה רגילה ב- BST: נחליף את הערך של 2 עם הערך של העוקב שלו, כלומר, עם 3:




**<u>שלב ג':</u>**
כעת, נרצה למחוק את 3 (העוקב של 2) מהעץ ולאחר מחיקה נקבל כי יש קודקוד עם פחות מ- t-1 מפתחות:




**<u>שלב ד':</u>**
נעשה merge בין הקודקוד ללא המפתחות, עם האח השמאלי שלו ועם האבא שלו, כלומר, נעשה merge בין הקודקוד הריק עם 1 ו- 3 ואז נעשה ל- 4 shift שיכנס במקום 3 ול-6 shift שיכנס במקום 4:



ג'

מנתוני השאלה, זמן הקריאה מהדיסק הוא $\alpha + \beta t$. כמו כן, מספר הקריאות מהדיסק הוא כגובה העץ ומכיוון שגובה העץ חסום מלמעלה ע"י $\log_t\left(\frac{n+1}{2}\right)$, זמן החיפוש הכולל במקרה הגרוע ביותר תוך כדי הזנחת זמן המעבד הינו $(\alpha + \beta t)\cdot\left(\log_t\left(\frac{n+1}{2}\right)\right)$. על מנת שנוכל לדעת מה הדרגה t שתמזער את זמן החיפוש הכולל שקיבלנו, נצטרך לגזור את הפונקציה שקיבלנו (פונקציית זמן הריצה של החיפוש הכולל), להשוות את הנגזרת לאפס ולמצוא נקודת מינימום:

$$f(t) = (\alpha + \beta t)\cdot\left(\log_t\left(\frac{n+1}{2}\right)\right)$$

$$f'(t) = ((\alpha + \beta t))'\cdot\left(\log_t\left(\frac{n+1}{2}\right)\right) + (\alpha + \beta t)\cdot\left(\left(\log_t\left(\frac{n+1}{2}\right)\right)\right)'$$

נשתמש בחוקי לוגריתמים:

$$\log_t\left(\frac{n+1}{2}\right) = \frac{\ln\left(\frac{n+1}{2}\right)}{\ln(t)}$$

$$\left(\log_t\left(\frac{n+1}{2}\right)\right)' = \left(\frac{\ln\left(\frac{n+1}{2}\right)}{\ln(t)}\right)' = \left(\ln\left(\frac{n+1}{2}\right)\cdot\frac{1}{\ln(t)}\right)' = \ln\left(\frac{n+1}{2}\right)\cdot\left(\frac{1'\cdot\ln(t) - 1\cdot(\ln(t))'}{\ln^2(t)}\right) =$$

$$\ln\left(\frac{n+1}{2}\right)\cdot\left(\frac{-\frac{1}{t}}{\ln^2(t)}\right) = \frac{-\ln\left(\frac{n+1}{2}\right)}{t\cdot\ln^2(t)} = \frac{\ln(2) - \ln(n+1)}{t\cdot\ln^2 t}$$

$$\left(\log_t\left(\frac{n+1}{2}\right)\right)' = \frac{-\ln\left(\frac{n+1}{2}\right)}{t\cdot\ln^2 t}$$

כמו כן, $(\alpha + \beta t)' = \beta$. נציב ונקבל:

$$f'(t) = ((\alpha + \beta t))'\cdot\left(\log_t\left(\frac{n+1}{2}\right)\right) + (\alpha + \beta t)\cdot\left(\left(\log_t\left(\frac{n+1}{2}\right)\right)\right)'$$

$$f'(t) = \beta\cdot\left(\frac{\ln\left(\frac{n+1}{2}\right)}{\ln(t)}\right) + (\alpha + \beta t)\cdot\left(\frac{-\ln\left(\frac{n+1}{2}\right)}{t\cdot\ln^2(t)}\right)$$

נשווה את הנגזרת לאפס על מנת להביע את t באמצעות $\alpha, \beta$:

$$f'(\mathrm{t}) = 0$$

$$\beta \cdot \left( \frac{\ln\left(\frac{n+1}{2}\right)}{\ln(t)} \right) + (\alpha + \beta t) \cdot \left( \frac{-\ln\left(\frac{n+1}{2}\right)}{t \cdot \ln^2(t)} \right) = 0$$

$$\ln\left(\frac{n+1}{2}\right) \cdot \left( \frac{\beta}{\ln(t)} - \frac{(\alpha + \beta t)}{t \cdot \ln^2(t)} \right) = 0$$

$$\left( \frac{\beta}{\ln(t)} - \frac{(\alpha + \beta t)}{t \cdot \ln^2(t)} \right) = 0$$

$$\frac{\beta t \cdot \ln(t) - \alpha - \beta t}{t \cdot \ln^2(t)} = 0$$

$$\beta t \cdot (\ln(t) - 1) - \alpha = 0$$

$$\beta t \cdot (\ln(t) - 1) = \alpha$$

$$t = \frac{\alpha}{\beta \cdot (\ln(t) - 1)}$$