מבני נתונים – תרגיל 7 – עץ AVL

שאלה 1

הצע מבנה נתונים עבור קבוצה S של מספרים שלמים התומך בפעולות הבאות:

search(k)
insert(k)
delete(k)

(עבור מספר שלם k) – בזמן $O(\log n)$, ו-opposite-pair() - בזמן $O(1)$.

הפעולה opposite-pair() מחזירה זוג מספרים ב- S שסכומם הוא 0 (אם קיים זוג כזה).

הנח כי בהתחלה S ריקה, וכי אין לאפשר הכנסה של מספר k אם הוא כבר נמצא במבנה.

רמז: מבנה הנתונים החדש יוגדר בעזרת עץ AVL ורשימה מקושרת. שים לב, שאיבר במבנה נתונים יכול להישמר ביותר ממקום אחד.

- יש לפרט את רעיון המימוש (מה יאוחסן בעץ AVL ומה יאוחסן ברשימה המקושרת, ומה הקשר בין שני מבני הנתונים).
- יש לפרט את הפסדו קוד למימוש כל אחת מהפעולות (ניתן להסתמך על פסדו קוד של פעולות שנלמדו בהרצאה, ללא פירוט).

פתרון-

נשמור את כל האיברים בעץ AVL. לכל איבר נוסיף שדה שבו מצביע pair לצומת ברשימה מקושרת [איתחול כ- null].

את הזוגות נשמור ברשימה המקושרת, כאשר את הזוג x,-x נייצג על ידי צומת שבו val=|x|. באיברים x,-x שבעץ, המצביע pair יצביע על הצומת המתאים ברשימה.

פעולות:

א. חיפוש: חיפוש רגיל בעץ AVL.
זמן: $O(\log n)$ [כאורך מסלול בעץ בינרי מאוזן בעל n צמתים.]

ב. הכנסת k:

1. הכנס את k לעץ AVL [אם k כבר נמצא בעץ החזר הודעת שגיאה], ואזן את העץ. שמור מצביע לאיבר זה.
2. בדוק אם האיבר הנגדי k- נמצא כבר בעץ.
אם לא – סיים.
אם כן – זהו זוג חדש בעץ. יש להכניס אותו לרשימת הזוגות. שמור מצביע גם לאיבר זה [הנגדי].
3. צור צומת חדש שבו val=|k|, והכנס אותו לראש רשימת הזוגות.
הכנס למצביע pair של x,-x את הכתובת של הצומת הזה.
זמן: O(log n) [הזמן שלוקח לבצע פעולות הכנסה/חיפוש על עץ AVL.]

ג. מחיקת k:
1. מצא את האיבר המתאים בעץ.
2. אם המצביע pair != null – גם הנגדי של k בעץ וכעת צריך למחוק את הזוג מהרשימה.
(a) בעזרת המצביע עבור לצומת המתאים ברשימה ומחק אותו.
[אם יש חשש שמספר הזוגות במערכת יהיה יותר מ- (log n) צריך לממש בעזרת רשימה דו כיוונית.]
(b) מצא בעץ את k- והכנס למצביע pair שלו את הערך null.
3. מחק את k מהעץ.
זמן: O(log n) [הזמן שלוקח לבצע פעולות חיפוש על עץ AVL.]

ד. החזרת זוג:
מהצומת שבראש הרשימה המקושרת, החזר את val, -val.
זמן: O(1)

<u>שאלה 2</u>

א. כדי לבנות עץ בגובה h, ניעזר בפונקציית העזר הרקורסיבית createCompleteTree (int h, Node_type* root). הפונקציה עוצרת כאשר h=0. אחרת, עבור הצומת שקיבלה היא יוצרת בן שמאלי ובן ימני, ומבצעת קריאה רקורסיבית עם גובה שקטן ב- 1 פעם על הבן השמאלי ופעם על הבן הימני (כדי ליצור את התת עצים שלהם). הפונקציה הראשית createTree(int h) תיצור את שורש העץ (רמה 0), ותקרא לפונקציה הרקורסיבית עם הערך h והמצביע לשורש.

```
void createCompleteTree(int h, Node_type* root)
{
        if (h==0)
                return;
```

```
root->left = createNode();
        createCompleteTree (h-1, root->left);
        root->right = createNode();
        createCompleteTree (h-1, root->right);
}


Node_type* createTree(int h)
{
        Node_type* root = new Node();
        if (h > 0)
                createCompleteTree (h, root);
        return root;
}
```

הוכחות נכונות: הפונקציה הראשית פועלת תמיד באותו אופן, ולכן מספיק להוכיח שפונקציית העזר אכן בונה עץ מלא בגובה h (לא כולל השורש). נראה זאת באינדוקציה על גובה העץ h:

בסיס האינדוקציה: h=0 : הפונקציה הראשית תיצור את השורש ולא תפעיל את פונקציית העזר.
h=1 : הפונקציה הראשית תיצור את השורש. פונקציית העזר תרוץ פעם אחת ליצירת שני הבנים של השורש [עם הערך h=1], ועוד פעם אחת שבה היא תעצור מבלי לעשות כלום כי היא תקבל את הערך h=0 .
צעד האינדוקציה: נניח שהטענה נכונה עבור עץ בעל גובה h-1 ונוכיח נכונות עבור h:
הפונקציה הראשית תיצור את השורש, ופונקציית העזר תרוץ פעם אחת ליצירת שני הבנים של השורש [עם הערך h=1]. כעת יתבצעו שתי קריאות רקורסיביות (אחת לכל בן) שבכל אחת מהן יועבר הערך h-1, ולפי הנחת האינדוקציה כל אחת מהן תבנה עץ בינארי מלא בגובה h-1.
כל עץ כזה הוא תת עץ של אחד הבנים של השורש, ולכן בסה"כ נוצר עץ מלא בגובה h.

חישוב עלות: ידוע כי בעץ מלא מספר הצמתים הוא 2^h-1 ועבור כל צומת שיצרנו קראנו לפונקציה הרקורסיבית שתי פעמים. עלות כל קריאה הייתה $\Theta(1)$, ולכן העלות הכוללת של האלגוריתם היא: $\Theta\left(2(2^h-1)\right)=\Theta(2^h)$.

ב. עלינו למזג בין שני עצי AVL נתונים – T1,T2.
פתרון– נסמן את מספר הצמתים של T1 ב- n1 ואת מספר הצמתים של T2 ב- n2.
נניח בה"כ n1<n2
בכל שלב נמחק צומת מ- T1 ונכניס אותו ל- T2. (פונקציית ההכנסה כוללת כמובן גם את התיקון של העץ במידה והאיזון הופר.)
עלות: עלות של $O(logn_2)$ **לכל הכנסת צומת**, מבצעים n_1 **פעולות הכנסה**, ולכן העלות הכוללת של האלגוריתם היא $\boldsymbol{O(n_1logn_2)}$.
[אמנם בצמתים האחרונים שמכניסים, גודל העץ הוא כבר n_1+n_2, אבל $n_1+n_2=\theta(n_2)$, ולכן זה לא משנה את זמן הריצה].

פתרון יעיל יותר – **טעות! פתרון לא נכון!!!**

1. ~~נעבור על כל עץ בסדר תוכי ונכניס את צמתיו למערך. נקבל שני מערכים ממוינים (כי מעבר בסדר תוכי על עץ AVL נותן מעבר מהקטן לגדול). על הדרך גם נמנה את מספר הצמתים בכל עץ. נסמן ב- n את סכום הצמתים בשני העצים יחד.~~ ~~**עלות:** $O(n)$.~~

~~2. נמזג את שני המערכים למערך ממוין אחד (ע"י פונקציית merge – פירוט בהמשך*). **עלות:** $O(n)$.~~

~~3. נבנה עץ שלם בגובה $logn+1$ תוך שימוש בפונקציה שכתבנו בסעיף א. **עלות:** $O(2^{logn+1}) = O(n)$.~~

~~4. נכניס את נתוני המערך הממוזג שיצרנו בשלב 2 לעץ שיצרנו בשלב 3 לפי הסדר, כלומר – נעבור על העץ בסדר תוכי ועל המערך לפי הסדר ונכניס כל איבר למקומו. **עלות:** $O(1)$ לכל צומת ובסה"כ $O(n)$.~~

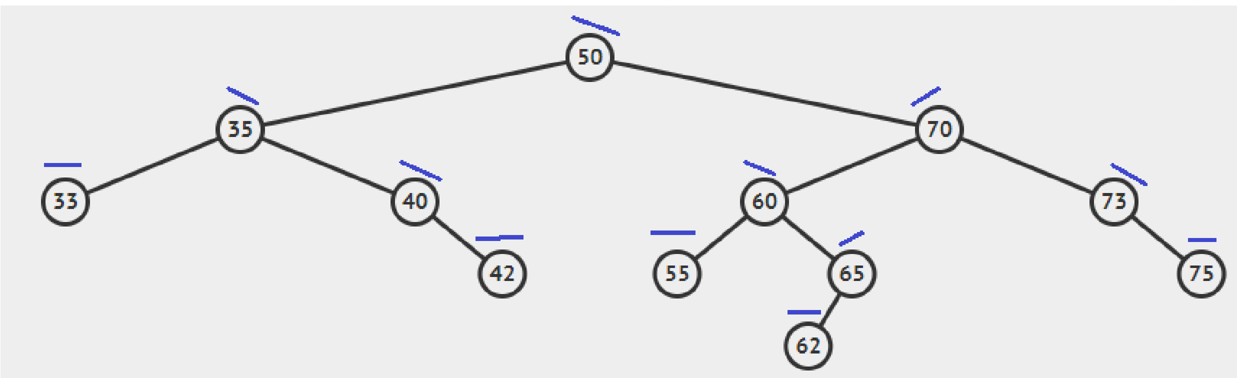
~~**עלות כוללת:** $O(n)$.~~

~~**פונקציית merge**: פונקציה שממזגת שני מערכים ממוינים למערך ממוין אחד.~~

~~בכל שלב נשווה בין האיברים הקטנים ביותר שבשני המערכים (נתחיל משני הראשונים). את הקטן יותר x נכניס למערך החדש, ונתקדם במערך של x אל האיבר הבא. נמשיך כך עד שנגיע לסוף של אחד המערכים, ואז נכניס את כל האיברים שנשארו במערך השני אל המערך החדש (לפי הסדר). נקבל מערך חדש ממוין, שמכיל את כל האיברים שהיו בשני המערכים המקוריים. זמן ריצה עבור n איברים בסה"כ: $O(n)$, כי בכל שלב מעבירים איבר אחד למערך החדש ולא מתעסקים איתו יותר.~~

<u>שאלה 3</u>

ייתכן עץ AVL שיהיו לו עלים בשלוש רמות שונות, להלן דוגמא:



ברמה 2 – 33

ברמה 3 – 42, 55, 75

ברמה 4 – 62

<u>ייתכן עץ AVL שיהיו לו עלים בארבע רמות שונות</u>: ניקח את העץ לעיל, שגובהו 4, ונקרא לו T1. ניקח עץ AVL נוסף בגובה 5 שכל ערכיו גדולים ממש מערכי T1, ונקרא לו T2. נבנה צומת חדש עם מפתח X, כאשר X מוגדר באופן הבא:

$$\max(T1) < X < \min(T2)$$

נגדיר את X כשורש של עץ AVL חדש ש- T1 הוא הבן השמאלי שלו ו- T2 הוא הבן הימני.

קבלנו עץ AVL תקין: תכונת עץ החיפוש הבינארי נשמרת כי ערכי T2 גדולים מ- T1, וכן גובהו של תת העץ הימני גדול ב- 1 מתת העץ השמאלי.

כעת העלים ב- T1 נמצאים ברמות 3,4,5 (עלו ברמה בגלל שהתווסף שורש חדש), וב- T2 קיים עלה בגובה 6 כי הגדרנו שגובהו גדול ב- 1 מ- T1.

הערה – באופן זה ניתן לבנות עץ AVL שיהיו לו עלים בכל מספר רמות שנרצה.