תיעוד פונקציות

Class AVLtree

<u>תיאור המחלקה:</u>

זו מחלקה שמיציגת עץ AVL, **ויש** בה מחלקה פנימית IAVLNode .

לכל עץ יש השדות: Root ,size,min,max,vertualnode

ומחלקה זו תומכת בפעולות:

infoToArray, join, split,insert, delete, size, min, max, keysToArray

ובכל צומת יש את השדות: rson,lson,parent,key,value,height

<u>:empty()</u>

- בודקים אם השורש הוא null אז מחזירים true אחרת מחזירים false .

O(1).

<u>: search(int k)</u>

- עוברים על העץ באופן דומה לחיפוש בינארי : אם המפתח של האיבר הנוכחי שווה ל-k אז מחזירים את הערך שלו, אם הוא קטן מ-k נמשיך לחפש בתת עץ הימני של הנוכחי אחרת נמשיך לחפש בתת עץ השמאלי.
- אם יצאנו מהלולאה בלי להחזיר ערך, אז האיבר איננו נמצא בעץ, נחזיר null .

העץ מאוזן לכן :
O(search)=O(log(n))

<u>: Tree_Position(IAVLNode x ,int k)</u>

- באופן דומה ל-search , אבל הפעם מקבלים איבר שרוצים להתחיל לחפש ממנו ו נחזיק מצביע על האיברים שעוברים עליהם בלולאה.
- בסוף נחזיר מצביע שהוא או מצביע על איבר עם אותו מפתח, או על אבא שלו.

העץ מאוזן לכן :
O(Tree_position)=O(log(n))

:insert(int k, String i)

- משתמשים ב tree_position כדי לדעת איפה להוסיף אותו אם לא קיים בעץ, ולפי מה שראינו זה לוקח O(logn) .
- אם הצומת כבר נמצאה בעץ אז נחזיר 1- .
- אחרת צריך להוסיף אותה לעץ, נוסיף 1 לשדה size של העץ, ונוסיף הצומת ע"י מספר קבוע של שינויי מצביעים שזה לוקח O(1).
- אחר כך משתמשים ב updatemin/max כדי לעדכן האיבר המינימלי/מקסימלי ,וזה לוקח O(logn).
- אם קיבלנו עץ AVL תקין אז אין צורך לעשות איזון לעץ ואז מחזירים 0, אחרת נאזן את העץ ע"י rebalance_insert ומחזירים את המספר שהחזירה, וזה לוקח O(logn) .

סך כל:

3*O(logn)+O(1)=O(logn)

:rebalance_insert(IAVLNode x)

- פונקציה זו מאזנת את העץ ומחזירה את מספר פעולות האיזון שעשינו.
- נשים לב שזו פונקציה רקורסיבית, ותנאי העצירה הוא אם הצומת היא השורש.
- מתחילים מx ובכל קריאה רקורסיבית קוראים לאב של x ולכן לכל היותר יש O(logn) קריאות רקורסיביות.
- נחלק למקרים, ראינו בכיתה שיש 6 מקרים, חלק מהם נעשה rotations וגם promote/demote וזה פותר הביעה אחרת נעשה promote/demote ונקרא רקורסיבית על צומת האב, אבל בשני המקרים הסיבוכיות חסומה ע"י O(logn) .

סך כל:

O(logn)*O(1)=O(logn)

: RotateRight(IAVLNode x)

- נסמן ב- z את אבא של x .
- הבן ימני של x יהפוך להיות הבן שמאלי של z.
- z יהפוך להיות בן ימני של x .

שינוי מספר סופי של מצביעים :

O(RotateRight)=O(1)

:RotateLeft(IAVLNode x)

- נסמן ב- z את אבא של x .
- הבן שמאלי של x יהפוך להיות הבן ימני של z.
- z יהפוך להיות בן שמאלי של x .

שינוי מספר סופי של מצביעים :

O(RotateLeft)=O(1)

:Promote(IAVLNode x)

- מעלים הגובה של x ב-1.

.O(1)

:Demote(IAVLNode x)

- מורידים הגובה של x ב-1.

.O(1)

:delete(int k)

- משתמשים ב tree_position כדי לדעת איפה הצומת נמצאת ,אם לא קיימת נחזיר 1-, ולפי מה שראינו זה לוקח O(logn) .
- אחרת צריך למחק אזי נחסיר 1 מהשדה size של העץ.
- אם הצומת היא השורש: אם היא גם עלה או צומת אונרית אז נשנה מספר קבוע של מצביעות ונעדכן כמה שדות ונחזיר 0, זה לוקח O(1) . אחרת נשים ה successor כRoot ונמחק את הצומת ע"י שינוי מספר קבוע של מצביעים וגם נעדכן שדות, ואחר כך נאזן העץ ע"י rebalance_delete שלוקח O(logn) .
- אם הצומת עלה או אונרית אבל לא שורש: נשנה מספר קבוע של מצביעות ונעדכן כמה שדות, זה לוקח O(1) . אחרת נשים ה successor במקום הצומת ונמחק את הצומת ע"י שינוי מספר קבוע של מצביעים וגם נעדכן שדות, ואחר כך נאזן העץ אם צריך ע"י rebalance_delete שלוקח O(logn), ונחזיר מספר פעולות האיזון שעשינו.

סך כל:

C*O(logn)=O(logn)   C קבוע

:rebalance_delete(IAVLNode z)

- פונקציה זו מאזנת את העץ ומחזירה בסוף את מספר פעולות האיזות שעשינו.
- ראשית אם העץ תקין נחזיר 0.
- אחרת נאזן ע"י מספר קבוע של rotations and promote/demote ואם יש צורך קוראים רקורסיבית לצומת האב, ונעצור כשנגיע לשורש, לכן זה לוקח C*O(logn)=O(logn)

סך כך זה לוקח O(logn).

:successor(IAVLNode x)

- כדי למצוא את ה successor של x עלינו ללכת לבן הימיני ואחר כך נרד שמאלה עד הסוף (נגיע לעלה או צומת אונרי)

O(1).

:min()/max()

- כיוון שיש לנו שדה min/max לעץ שמצביע לצומת המינימלית/מקסימלית אז רק נחזיר אותו לכן זה לוקח O(1) ובכל פעם משנים את העץ אנחנו מעדכנים אותם ע"י updatemin()/updatemax() .

O(1).

: keysToArray()

- אם העץ ריק מחזירים מערך ריק.
- אחרת : מגדירים מערך בגודל העץ וממלאים אותו ע"י שימוש בפונקציה הרקורסיבית keysToArrayRec . O(n)
- מחזירים את המערך.

O(keysToArray)=O(n)

:keysToArrayREC(int i, int[]arr,IAVLNode r)

- עושים in-order-walk על העץ וכל פעם מוסיפים את המפתח של האיבר למערך.
- הפונקציה מחזירה האינדקס.

O(keysToArrayREC)=O(n)

: infoToArray()

- אם העץ ריק מחזירים מערך ריק.
- אחרת : מגדירים מערך בגודל העץ וממלאים אותו ע"י שימוש בפונקציה הרקורסיבית infoToArrayRec . O(n)
- מחזירים את המערך.

O(infoToArray)=O(n)

: infoToArrayREC(int i, String[]arr,IAVLNode r)

- עושים in-order-walk על העץ וכל פעם מוסיפים את הערך של האיבר למערך.
- הפונקציה מחזירה האינדקס.

O(infoToArrayREC)=O(n)

:size()

- יש לנו שדה size שמעדכנים אותו בכל פעולת delete/insert, אז רק מחזירים אותו.

O(1).

:getRoot()

- יש לנו שדה Root שמחזירים אותו.

O(1)

.

:updatemin()/updatemax()

- פונקציות אלו מעדכנים את השדות min/max .
- אנחנו יודעים שהאיבר המקסימלי/המינימלי נמצא הכי ימינה/שמאלה בעץ(אם נטייל מהשורש ימינה/שמאלה עד שנגיע לעלה או צומת אונרית) לכן זה חסום על גובה העץ וכיוון שאנחנו בעץ AVL זה logn (כך ש n הוא מספר הצמתים בעץ)

O(logn).

: Join(IAVLNode x, AVLTree t)

- חשבנו את הערך שאנחנו רוצים להחזיר. (ערך פעולת ה-join)
- אם שתי העצים ריקים : השורש של העץ הנוכחי יהיה ה-x ונחזיר 1.
- אם העץ הנוכחי ריק : נכניס x ל-t ו -t יהיה העץ הנוכחי שלנו.
- אם t ריק : נכניס x לעץ הנוכחי.
- אם אף אחד מהעצים לא ריק : נחלק למקרים, 1. this.keys<x.key<t.keys .2 t.keys<x.key<this.keys , ובכל אחד מהמקרים מחלקים לפי אורכי העצים (קטן,גדול,שווה).
- אם העץ הארוך הוא בעל המפתחות הגדולות עוברים בלולאה ע"י getLeft() עד שנגייע לצומת שהגובה שלו שלו קטן \ שווה הגובה של העץ הקצר ו מחברים. O(log(n))
- אם העץ הארוך הוא בעל המפתחות הקטנות עוברים בלולאה ע"י getRight() עד שנגייע לצומת שהגובה שלו שלו קטן \ שווה הגובה של העץ הקצר ו מחברים. O(log(n))
- בסוף מעדכינים המקסימום והמנימום ועושים rebalance לעץ. O(1) + O(log(n))*3

O(Join)=O(log(n))*4 + O(1) =O(log(n))

: rebalance_join(IAVLNode x)

- בודקים אם מתקיים המקרה : שההפרש בין גבהי האבא של x והבן האחר שלו הוא 2 , וגובה האבא שווה לגובה ה-x .
- אם כן, עושים Promote(x) ו RotationRight או RotationLeft לפי מיקום ה-x ביחס לאבא שלו.

O(rebalance_join)=O(1)

: Split(int k)

- מחפשים את האיבר עם המפתח k נסמן אותו ב-y . O(log(n))
- מגדירים שתי עצים חדשות bigger ו smaller , נשים בהם הבן הימני ו הבן השמאלי בהתאמה.
- אם x שווה למקסימום נעדכן smaller לכל העץ אבל בלי ה-x ו bigger יהיה null .
- אם x שווה למקסימום נעדכן bigger לכל העץ אבל בלי ה-x ו smaller יהיה null .
- אחרת : נעבור על האיברים מ-y עד השורש, וכל פעם בודקים אם האיבר הוא בן שמאלי או בן ימיני. אם הוא ימיני אז נעשה join עם האבא שלו, תת עץ שמאלי של האבא ו smaller . (ראינו בהרצאה שזה O(log(n))).
  אם הוא שמאלי אז נעשה join עם האבא שלו, תת עץ ימיני של האבא ו bigger .
- בסוף מעדכינים מקסימום ו מינימום לעצים, ו מחזירים מערך שהאיבר הראשון בו הוא smaller ו האיבר השני הוא bigger . O(log(n))

O(split)=O(log(n))*3=O(log(n))

Class IAVLNode

:getKey()

- אם הצומת אמיתית מחזירים את השדה key שלה אחרת מחזירים -1

.O(1)

:getValue()

- אם הצומת אמיתית מחזירים את השדה value שלה אחרת מחזירים null.

.O(1)

:setLeft()/setRight()

- משנים הבן השמלי/הימיני (השדה lson/rson) של הצומת.

.O(1)

getLeft()/getRight():

- אם הצומת אמיתית מחזירים את הבן הימיני/שמלי (rson/lson) שלה אחרת מחזירים null .

O(1).

getRight1()/getLeft1():

- מחזירים את הבן הימיני/שמאלי (rson/lson) שלה (כמו getRight/grtLeft() אבל מאפשרים להחזיר vertualnode).

O(1)

setParent():

- משנים האב (השדה parent) של הצומת.

O(1).

getParent():

- מחזירים האב (השדה parent) של הצומת.

O(1)

isRealNode():

- אם הצומת שווה ל-null or virtualnode אז מחזירים false אחרת true.

O(1)

setHeight(int height):

- משנים את הגבה של הצומת (השדה height).

O(1)

getHeight():

- מחזירים הגבה (השדה height) של הצומת.
  O(1)

<u>חלק תיאוריתי</u>

<u>סעיף א</u>

מספר סידורי i	מספר חילופים במערך ממוין הפוך	עלות החיפושים במיון AVL עבור מערך ממוין-הפוך	מספר חילופים במערך מסודר אקראית	עלות החיפושים במיון AVL עבור מערך מסודר אקראי
1	1999000	36884	1022536	31250
2	7998000	81764	4000936	67345
3	31996000	179524	13971354	144562
4	127992000	391044	53219934	303043
5	511984000	846084	245468267	673462

<u>סעיף ב</u>

מספר החילופים במערך ממוין הפוך: קודם כל נעשה n-1 חילופים. אחר כך נעשה n-2 חילופים וכך... אז סכום החילופים שווה ל- n(n-1)/2 , וזה חסום ע"י **O(n^2)** .

עלות החיפושים במערך ממוין הפוך : המערך ממוין הפוך לכן המספר הראשון שמכניסים יהיה המקסימום לכן כל החיפושים יתחילו מהשורש. אז נתחיל במפתח n – 0 בדיקות , n-1 – החיפושים מתחלים בשורש.

n-1 :בדיקה אחת.

n-2 : 2 בדיקות.

n-3 : 2 בדיקות.

n-4 : 3 בדיקות.

.

.

.

n-7 : 3 בדיקות. וכך ממשיכים.. (כל פעם צריך להשוות ערך שלם תחתון (log(n) .

נחסום מלמעלה : גובה העץ יהיה בהכנסה אחרונה לכל היותר (log(n אז נחסום מלמעלה ע"י nlogn. (כל פעם נחסום מספר הבדיקות ע"י logn).

נחסום מלמטה : עץ כמעט מאוזן , לכן חצי האיברים בשורה האחרונה הם כמעט n/2logn .

אז תטא nlogn .