מגישות:

יערה פדרמן, ת"ז 205946312, שם משתמש yaaraf

שירי קוליוק, ת"ז 312530686, שם משתמש shirikullock

תיאור המחלקה

המחלקה AVLTree מכילה את השדות הבאים:

- Root – מצביע לצומת שמשמש בתור שורש העץ.
- Min – מצביע לצומת בעל המפתח המינימלי בעץ.
- Max – מצביע לצומת בעל המפתח המקסימלי בעץ.
- Size – מספר הצמתים בעץ.
- EXTERNAL – צומת שישמש בתור העלה החיצוני של העץ (גובהו יהיה 1-).

המחלקה תומכת בכל הפעולות הנדרשות.

למחלקה AVLTree יש מחלקה פנימית AVLNode שמממשת את המנשק IAVLNode.

למחלקה AVLNode השדות הבאים:

- Key – מספר שמייצג את המפתח של הצומת.
- Info – מחרוזת שמייצגת את המידע השמור בצומת.
- Right – מצביע לבן הימני של הצומת.
- Left – מצביע לבן השמאלי של הצומת.
- Parent – מצביע להורה של הצומת.
- Height – גובה הצומת (שווה לדרגת הצומת בעץ AVL).
- Size – מספר הצמתים בתת העץ של הצומת (כולל עצמו).

מחלקת AVLNode תומכת רק בפעולות - get, set, constructor וisRealNode שמחזירה אם הצומת היא צומת חיצוני או לא.

<u>תיעוד הפונקציות</u>

הערות כלליות:

1. נשים לב כי גובה עץ AVL הוא O(log(n)), ולכן כל פונקציה שרצה בO(height) תרוץ בO(log(n)) זמן.
2. נסמן את הפרשי הדרגות בין צומת מסוים לשורש O(depth).
3. נסמן את הפרש הדרגות של שני עצים שונים ב O(diff).

<u>המחלקה AVLNode</u>

כל הפונקציות של המחלקה AVLNode פועלות בO(1) זמן, שכן הן מחזירות/ משנות ערך בשדה כלשהו של המחלקה, או יוצרות מופע חדש שדורש אתחול של מספר קבוע של שדות.

<u>Constructor</u>

הבנאי לא מקבל ארגומנטים, הוא אחראי לאתחל את שדות המחלקה לערכם הdefault. לוקח O(1) זמן.

<u>Empty</u>-

בודק אם השורש של העץ הוא null, דורש O(1) זמן.

<u>Search</u> –

מחפש צומת עם מפתח מסוים. לשם כך עובר מסלול מהשורש עד אחד הצמתים בעץ, מסלול שחסום על ידי גובה העץ, ולכן רץ בO(log(n)).

<u>Insert</u>

פונקציית insert מבצעת את הפעולות הבאות –

- במידה והעץ ריק: אתחול הצומת כשורש העץ – דורש O(1) זמן. במקרה זה הפונקציה מסיימת לאחר עדכון כל השדות בעץ.
- קריאה לwhereToInsert – לוקח O(log(n)) זמן.
- קריאה לפונקציית fixMinMaxInsert – לוקח O(1) זמן.
- הכנסה של עלה חדש לעץ, דורש שינוי במספר קבוע של מצביעים ולכן לוקח O(1) זמן.
- קריאה לfixSizeTree, כאשר הצומת שנשלחת היא עלה, ולכן O(depth) = O(log(n)).
- עדכון גודל העץ – דורש O(1) זמן.
- קריאה לפונקציית rebalanceInsert - לוקח O(log(n)) זמן.

בסה"כ נקבל שסיבוכיות הפונקציה insert היא O(log(n)) זמן כנדרש.

<u>הפונקציות להן קוראת insert:</u>

WhereToInsert

הפונקציה מחפשת איפה להכניס צומת חדש. היא רצה על העץ כלפי מטה עד שהיא מוצאת את המיקום הרלוונטי.

לאחר מכן היא עולה במעלה העץ באותו המסלול ומגדילה את גדלי הצמתים ב1.

מכיוון שהיא מבצעת 2 מעברים על העץ מהשורש עד עלה כלשהו שלו, היא רצה ב(O(log(n זמן.

fixMinMaxInsert

מתקנת מצביעים לשדות min,max לאחר מספר קבוע של השוואת מפתחות. פועלת ב(O(1 זמן.

fixSizeTree

הפונקציה מטפסת מהצומת לשורש ומוסיפה i לsize של כל הצמתים לאורך הדרך ולכן פועלת ב(O(depth.

rebalanceInsert

הפונקציה מאזנת את העץ לאחר הכנסה של צומת חדש. בפונקציה ישנה לולאה שרצה לכל היותר (log(n פעמים – היא מתחילה מעלה ובכל שלב עולה במעלה העץ, ולכל היותר תרוץ עד השורש.

בלולאה יש מספר קבוע של קריאות לפונקציות fixHeight, rotateLeft, rotateRight, diagnoseInsert – כולן פונקציות שלוקחות (O(1 זמן ולכן הלולאה רצה בסה"כ ב(O(log(n זמן.

<u>הפונקציות להן RebalanceInsert קוראת:</u>

diagnoseInsert

הפונקציה מבצעת מספר קבוע של השוואות הפרשי גבהים – בין הצומת לילדיו, ובחלק מהמקרים גם רמה אחת מטה ומחזירה מחרוזת.

בסה"כ הפונקציה פועלת ב(O(1 זמן.

rotateLeft, rotateRIght

הפונקציות משנות מספר קבוע של מצביעים, מה שלוקח (O(1 זמן. לאחר מכן קוראות קריאה ל fixSizeNode שגם כן רצה ב(O(1 זמן, לכן בסה"כ לוקח (O(1 זמן.

fixSizeNode

מעדכנת גודל של צומת לפי גודל הבנים שלו. החישוב לוקח (O(1 זמן.

fixHeight

מעדכנת גובה של צומת לפי גובה הבנים שלו. החישוב לוקח O(1) זמן.

delete

פונקציית delete מבצעת את הפעולות הבאות –

- בדיקה האם העץ ריק או האם הצומת הנמחק הוא היחיד בעץ – אם כן הטיפול דורש עדכון של שדות העץ ולאחר מכן הפונקציה מסיימת, דורש O(1) זמן.
- קריאה אחת לsearchNode – לוקח O(log(n)) זמן.
- קריאה לfixMinMaxDelete – לוקחת O(log(n)) זמן.
- קריאה לdeleteForReal – לוקחת לכל היותר O(log(n)) זמן.
- קריאה לfixSizeTree, כאשר הצומת שנשלח הוא מגובה לכל היותר 1, ולכן O(diff) = O(log(n)).
- עדכון גודל העץ – דורש O(1) זמן.
- קריאה לפונקציית rebalanceDelete - לוקחת O(log(n)) זמן.

בסה"כ נקבל שסיבוכיות הפונקציה delete היא O(log(n)) זמן כנדרש.

<u>הפונקציות להן קוראת delete:</u>

searchNode

הפונקציה מחפשת איפה נמצא הצומת המבוקש. היא רצה על העץ כלפי מטה עד שהיא מוצאת את הצומת הרלוונטי. כיוון שהיא מבצעת מעבר אחד על העץ מהשורש עד עלה כלשהו שלו, היא תרוץ בO(height) זמן, כלומר O(log(n)) זמן.

fixMinMaxDelete

מתקנת מצביעים לmin,max ע"י קריאה אחת לכל היותר לsuccessor או לpredecessor ולכן פועלת בO(log(n)) זמן.

deleteForReal

הפונקציה מוחקת צומת מבוקש מהעץ. נחלק למקרים –

- אם הצומת המבוקש הוא עלה או צומת אונרי – נשנה מספר קבוע של מצביעים ולכן ייקח O(1) זמן.
- אם לצומת המבוקש יש 2 ילדים – נבצע קריאה לפונקציית successor שלוקחת O(log(n)) זמן.

בסה"כ נקבל שסיבוכיות הפונקציה היא O(log(n)) זמן.

Successor

הפונקציה מחפשת את הsuccessor של צומת. הפונקציה עושה אחת משתי פעולות –

- מחפשת את הצומת המינימלית בתת העץ הימני של הצומת. לשם כך היא רק יורדת בתת עץ שגובהו חסום ע"י גובה העץ ולכן פעולה זו תיקח ((O(log(n זמן.
- אם לצומת אין בן ימני, הפונקציה תעלה במעלה העץ כל עוד הצומת עליו הפעלנו את הפונקציה נמצא בתת העץ הימני של הצומת הבא. בסה"כ נעלה לכל היותר עד השורש ולכן פעולה זו תיקח ((O(log(n זמן.

בסה"כ נקבל סיבוכיות ((O(log(n זמן.

Predecessor

הפונקציה מחפשת את הpredecessor של צומת. הפונקציה עושה אחת משתי פעולות –

- מחפשת את הצומת המקסימלי בתת העץ השמאלי של הצומת. לשם כך היא רק יורדת בתת עץ שגובהו חסום ע"י גובה העץ ולכן פעולה זו תיקח ((O(log(n זמן.
- אם לצומת אין בן שמאלי, הפונקציה תעלה במעלה העץ כל עוד הצומת עליו הפעלנו את הפונקציה נמצא בתת העץ השמאלי של הצומת הבא. בסה"כ נעלה לכל היותר עד השורש ולכן פעולה זו תיקח ((O(log(n זמן.

בסה"כ נקבל סיבוכיות ((O(log(n זמן.

rebalanceDelete

הפונקציה מאזנת את העץ לאחר מחיקה של צומת. בפונקציה ישנה לולאה שרצה לכל היותר log(n) פעמים – היא מתחילה מעלה ובכל שלב עולה במעלה העץ, לכל היותר תרוץ עד השורש.

בלולאה יש מספר קבוע של קריאות לפונקציות fixHeight, rotateLeft, rotateRight, diagnoseDelete – כולן פונקציות שלוקחות O(1) זמן ולכן הלולאה רצה בסה"כ ב((O(log(n זמן.

(סיבוכיות הזמן של fixSizeTree, fixHeight, rotateLeft, rotateRight. – מפורטת תחת Insert)

diagnoseDelete

הפונקציה מבצעת מספר קבוע של השוואות הפרשי גבהים – בין הצומת לילדיו, ובחלק מהמקרים גם רמה אחת מטה ומחזירה מחרוזת.

בסה"כ הפונקציה פועלת בO(1) זמן.

min, max, size, getRoot

מחזירות ערך השמור בשדה / בצומת אליו שמור מצביע בשדה min, max, size,getRoot, בהתאמה. דורש O(1) זמן.

keysToArrays, infoToArray

נשים לב שהפונקציות keysToArrays ,infoToArray רק מאתחלות מערך וקוראת לפונקציית עזר, ולכן ננתח את זמן הסיבוכיות של הפונקציות הרקורסיביות להן הן קוראות:

keysToArraysRec, infoToArrayRec

בכל ריצה יש 2 קריאות רקורסיביות – לתת העץ הימני והשמאלי של כל שורש. מכיוון שמדובר בעצי AVL, גודל העץ בכל קריאה רקורסיבית קטן בערך בחצי. בנוסף, בכל קריאה מתבצעות 2 פעולות נוספות - הוספת האיבר הנוכחי curr למערך, והגדלת loc ב-1. לכן, הפונקציה מתאימה לנוסחת הנסיגה $T(n) = 2T\left(\frac{n}{2}\right) + 1$.

פתרון נוסחת הנסיגה הוא θ(n) ולכן הפונקציה רצה ב O(n) זמן.

Join

נסמן diff בתור הפרשי הגבהים של שני העצים עליהם מבוצעת הפעולה.

1. ראשית, הפונקציה בודקת אם אחד מהעצים שהתקבלו הוא עץ ריק. אם כן, תבצע מספר קבוע של השמות וקריאה לפונקציית insert שפועלת בO של גובה העץ אליו מבוצעת הכנסה. במקרה זה, העץ אליו מבוצעת הכנסה הוא העץ הגבוה יותר, והפרשי הגבהים בין העצים שווה לגובהו ולכן הפונקציה פועלת בO(diff).
   סה"כ הפונקציה תפעל ב O(diff) זמן ותסיים.
2. לאחר מכן הפונקציה מוצאת את עץ המפתחות הגדולים ואת העץ הגבוה יותר. דורש O(1) זמן.
3. אם העצים באותו גובה – היא מבצעת קריאה לjoinEquals שפועלת בO(1) זמן ומסיימת.
4. אם העצים לא באותו גובה –

- מבצעת קריאה לsearchOnRight אם העץ הגבוה יותר הוא עץ המפתחות הקטנים או לsearchOnLeft אם העץ הגבוה יותר הוא עץ המפתחות הגדולים. הפונקציות הללו פועלות בO(diff).
- קוראת לjoinForReal שפועלת בO(1) זמן.
- מבצעת מספר קבוע של השמות.
- קוראת לfixSizeTree, כאשר הצומת שנשלח הוא נקודת האיחוד, ולכן דרגתו שווה לדרגת השורש של העץ הנמוך. לכן דורש O(diff).
- קוראת לrebalanceInsert עם העץ המאוחד וצומת ההורה של נקודת האיחוד. כפי שציינו, נקודת האיחוד היא צומת בגובה העץ הנמוך יותר, rebalanceInsert תעלה מנקודה זו עד לכל היותר שורש העץ המאוחד (שורש העץ הגבוה), ולכן פועלת בO(diff).

בסה"כ פונקציית join תפעל בO(diff).

נסמן בn1 את מספר הצמתים בעץ הגבוה, וב- n2 את מספר הצמתים בעץ הנמוך.

נסמן n = n1+n2. נשים לב כי מתקיים

$$O(diff) = O(logn1 - logn2) = O\left(\log\left(\frac{n1}{n2}\right)\right) = O(\log(n))$$

כאשר המעבר האחרון מתקיים כי $\frac{n1}{n2} \leq n$

כלומר, O(diff) חסום ע"י O(log(n)) כאשר n+1 הוא מספר הצמתים בעץ המאוחד.

לכן join פועלת בO(log(n)).

פונקציות להן קוראת join:

joinEquals

אם שני העצים שנרצה לאחד הם באותו גובה, נאחדם על ידי שינוי מספר קבוע של מצביעים.

בנוסף, נתקן את דרגת השורש החדש ע"י קריאה לפונקציה fixHeight שלוקחת O(1) זמן.

בסה"כ הפונקציה תפעל בO(1) זמן.

searchOnRight

אם בפעולת join העץ עם המפתחות הקטנים יותר הוא גם העץ הגבוה יותר, תתבצע קריאה לפונקציה זו. פונקציה זו מחפשת על הדופן הימנית של העץ הגבוה את נקודת האיחוד בין העצים.

נקודת האיחוד בין העצים תהיה הצומת בגובה שקטן או שווה לגובה של העץ הנמוך יותר.

לכן הפונקציה פועלת בO(diff).

searchOnLeft

אם בפעולת join העץ עם המפתחות הגדולים יותר הוא גם העץ הגבוה יותר, תתבצע קריאה לפונקציה זו. פונקציה זו מחפשת על הדופן השמאלית של העץ הגבוה את נקודת האיחוד בין העצים.

נקודת האיחוד בין העצים תהיה הצומת בגובה שקטן או שווה לגובה של העץ הנמוך יותר.

לכן הפונקציה פועלת בO(diff).

joinForReal

הפונקציה מאחדת את שני העצים בנקודה המתאימה בעץ הגבוה מבניהם. הפונקציה משנה מספר קבוע של מצביעים וקוראת לfixHeight שפועלת בO(1) זמן.

בסה"כ הפונקציה תפעל בO(1) זמן.

(הסיבוכיות של הפונקציות fizSizeTree, rebalanceInsert – מפורט תחת Insert).

Split

הפונקציה split מבצעת מספר פעולות :

- קריאה לפונקציה searchNode שמחזירה את הצומת לפיה יתבצע הפיצול. דורש O(log(n)) זמן.
- אם הצומת הוא לא עלה - קריאה לsetTreeWithRoot עם בן אחד או שני הבנים של הצומת. פועלת בO(1) זמן.
- לאחר מכן הפונקציה נכנסת ללולאה שמתקדמת במעלה בעץ, כלומר מתבצעות O(log(n)) איטרציות. בכל איטרציה מתבצעות הפעולות :
  - מספר קבוע של השוואות לוקח O(1) זמן.
  - בניית עץ חדש בעזרת הconstructer או קריאה לsetTreeWithRoot – דורש O(1) זמן.
  - אתחול של מספר קבוע של מצביעים ושדות בצומת אחד.
  - קריאה לjoin – הפונקציה join פועלת בO(diff).

כלומר נקבל שסיבוכיות הפונקציה split היא O(log(n)*diff).

כפי שלמדנו בכיתה, סיבוכיות זו חסומה ע"י O(log(n)).

setTreeWithRoot

פונקציה שמקבלת צומת כארגומנט – ומאתחלת את שורש העץ לצומת שקיבלה. דורש O(1) זמן.

לאחר מכן מתבצעת קריאה לfindMax, findMin שלוקחות O(log(n)) זמן.

סה"כ רץ בO(log(n)) זמן.

findMin,findMax

הפונקציות יורדות במורד הצלע הימנית / השמאלית של העץ עד לעלה המקסימלי /המינימלי ומחזירות אותו.

לוקחות O(log(n)) זמן.

(הפונקציות searchNode וjoin מפורטות לעיל)

מדידות

ניסוי 1:

מספר סידורי	מספר פעולות	מספר פעולות האיזון הממוצע לפעולת insert	מספר פעולות האיזון הממוצע לפעולת delete	מספר פעולות האיזון המקסימלי לפעולת insert	מספר פעולות האיזון המקסימלי לפעולת delete
1	10,000	3.446	2.395	15	26
2	20,000	3.38	2.391	17	33
3	30,000	3.322	2.356	17	33
4	40,000	3.316	2.351	17	33
5	50,000	3.304	2.332	18	33
6	60,000	3.281	2.316	19	34
7	70,000	3.263	2.296	20	38
8	80,000	3.234	2.275	20	38
9	90,000	3.225	2.264	20	39
10	100,000	3.198	2.250	20	38

<u>מספר פעולות איזון ממוצע:</u>

לפי ההסבר התיאורטי בכיתה מספר פעולות האיזון הממוצע עבור סדרה של הכנסות או מחיקות הינו מספר קבוע, כלומר amortized time של תהליך האיזון הינו O(1).

אכן, לפי הנתונים בטבלה, קיבלנו שמספר פעולות האיזון הממוצע, הן בinsert והן בdelete הוא מספר קבוע, ללא תלות בn.

כלומר, משמעות המדידות היא שאכן הamortized time של תהליך האיזון הוא קבוע.

<u>מספר פעולות איזון מקסימלי:</u>

לפי ההסבר התיאורטי בכיתה, במקרה הגרוע, מספר פעולות האיזון בפעולת הכנסה או מחיקה חסום ע"י O(log(n)).

אכן ניתן לראות במדידות, כי המספר המקסימלי של פעולות איזון בהכנסה הוא כ – log(n), ובמחיקה הוא כ – 2log(n). כלומר המספר המקסימלי של פעולות איזון הוא O(log(n)).

אמנם לא קיבלנו תוצאות מדידה השוות לlog(n) בדיוק, אך ניתן לראות כי עם הגידול בn, מספר הפעולות המקסימלי גדל ביחס log לקלט, ולא ביחס ליניארי.

כלומר, משמעות המדידות היא שבמקרה הגרוע מספר פעולות האיזון הוא אכן O(log(n)).

ניסוי 2:

מספר סידורי	עלות join ממוצע עבור split אקראי	עלות join מקסימלי עבור split אקראי	עלות join ממוצע עבור split של איבר מקס בתת העץ השמאלי	עלות join מקסימלי עבור split של איבר מקס בתת העץ השמאלי
1	2.833	6	2.384	15
2	2.769	5	2.714	16
3	2.666	4	2.692	17
4	2.466	7	2.437	17
5	2.5	4	2.4	18
6	2.25	6	2.187	18
7	2.437	7	2.687	18
8	2.714	4	2.375	18
9	2.533	6	2.4	18
10	2.875	5	2.6	19

(כולל הפירוט הנדרש בבונוס)

עלות join ממוצע:

לפי ההסבר התיאורטי בכיתה, עלות הפונקציה join שווה להפרש הגבהים של שני העצים אותם היא מחברת. מתוך הוכחה שראינו בכיתה, בפעולת split מתרחשת סדרה של פעולות join שעלות כל אחת מהן היא כאמור הפרש הגבהים בין 2 העצים שנשלחים לjoin. אם נסכם את העלות הכוללת של סדרת פעולות הjoin נקבל סכום טלסקופי ששווה להפרש בין גובה העץ פחות 1 (העץ האחרון עליו תתבצע פעולת join הוא בן של השורש) ובין גובה הצומת ממנו התחיל תהליך הsplit. במקרה הגרוע ביותר תהליך הsplit יתחיל מעלה, ולכן הפרש זה חסום ע"י גובה העץ. נקבל שעלות סדרת פעולות הjoin חסומה ע"י O(log(n)).

נשים לב, כי במהלך split מתבצעות O(log(n)) פעולות join, ולכן עלות פעולת join ממוצעת שווה לO(1) זמן.

נשים לב, כי המדידות עולות בקנה אחד עם ההסבר התיאורטי שראינו בכיתה, שכן קיבלנו עלות join ממוצע יחסית קבועה.

כלומר, אכן הamortized time של פעולת join במסגרת פעולת split הוא O(1) זמן.

עלות join מקסימלי:

לפי ההסבר התיאורטי בכיתה, עלות הפונקציה join שווה להפרש הגבהים של שני העצים אותם היא מחברת.

נשים לב כי במקרה שהצומת עליו מבצעים את הsplit הוא המקסימום בתת העץ השמאלי יתבצע התהליך הבא:

1. נטפס תמיד מימין לכיוון השורש, וניצור את עץ הקטנים, כאשר הפרש הדרגות בין העצים בכל join הוא לכל היותר 2.
2. כשנגיע לשורש, יתבצע הjoin הראשון עבור עץ ה"גדולים" שעד עתה היה ריק, ותת העץ הימני של השורש. לכן הפרש הדרגות בין העצים יהיה כגובה העץ פחות 1.

כלומר הjoin המקסימלי יהיה פעולת הjoin האחרונה ולכן פעולה זו תעלה O(log(n)).

אכן, ניתן לראות במדידות כי כשפעולת split מתבצעת על הצומת המקסימלית בתת העץ השמאלי של השורש, פעולת הjoin המקסימלית עולה O(log(n)).

נשים לב כי במקרה שהצומת עליו מבצעים את הsplit הוא צומת אקראי בעץ, הסיכוי שהצומת האקראי שיבחר יהיה הצומת המקסימלי בתת העץ השמאלי, כלומר שנקבל את המקרה הגרוע ביותר, קטן מאוד ($\frac{1}{n}$).

לכן עלות הjoin המקסימלי תהיה נמוכה יותר, כפי שניתן לראות במדידות בטבלה.