המחלקה AVLTree :

כללי:

פונקציות:

Empty:

מחזירה true אמ"מ השורש (root) מצביע על null או שהינו צומת דיגיטלי זה קורה אמ"מ העץ ריק

סיבוכיות O(1).

Search :

מחפשת את האיבר בעל המפתח K, אופן פעולה קוראת לפונקציית העזר find על הערך K אם לא חזר null מחזירה את הערך של הצומת שחזר אחרת מחזירה null.

סיבוכיות: פונקציית העזר find רצה ב-log(n) (פירוט בהמשך) ושאר הפונקציה הינו פעולות קבועות כלומר O(1) סה"כ נקבל O(logn).

פונקציית העזר find: מקבלת מפתח K ובודקת אם יש צומת בעץ בעל המפתח אם כן מחזירה את הצומת אם לא מחזירה null. אופן פעולה: אופן באופן זהה לפונקציית חיפוש בראינו בהרצאה, נטייל במורד העץ ובכל צומת נבדוק אם הוא הצומת המתאים אם לא אם עלינו ללכת שמאלה או ימינה בהתאם להשוואה בין המפתחות אם הגענו לצומת וירטואלי נפסיק את החיפוש ונחזיר null. סיבוכיות במקרה הגורע logn עומק העץ.

rotate\_left או rotate\_right:

מבצעת סיבוב שמאלה על 2 צמתים קשר הקשת שלהם היא ציר הסיבוב. מבצעים החלפה בחיבורים של הקשתות המתאימות ובודקים מקרה קצה במידה ומדובר בעדכון של השורש. מעדכנים מצביעים לאמא של כל צומת. סיבוכיות כל הפעולות בקריאה לפונק' קבועות ולכן O(1).

Delete:

אם קיים צומת בעץ בעל מפתח K מוחקת אותו ומחזירה את סך פעולות האיזון באיזון העץ לאחר המחיקה.

אופן הפעולה:

קוראת לפונקציה find שבודקת אם קיים כזה צומת סיבוכיות O(logn)

אם לא קיים מחזירה -1, אם קיים מבצעים את רצץ הפעולות הבא:

בודקים אם מדובר במקרה קצה של עדכון מינימום או מקסימום, מעדכנים ע"י קריאה לפונק' successor או predecessor בהתאמה (סיבוכיות O(logn) ) לאחר מכן מקטינים את הגודל של העץ ב-1. ואז קוראים לפונקציה deleteRetrieve שמוחקת את העץ ע"י אפיון מקרה המחיקה ומחיקה בצורה המתאימה ומחזירה את הצומת שצריך לאזן. אם בוודאות אין כזה היא מחזירה null. אם הערך null מחזירים 0, אחרת קוראים לפונקציית האיזונים reBalanceDelete עם הצומת שצריך לעבור איזון, הפונקצייה עובדת באופן רקורסיבי ומאזנת את העץ ע"פ המקרים שנלמדו בהרצאה וסופרת את מס' השינויים שנעשים בעץ, סוכמת ומחזירה את סכום השינויים בעץ. במקרה הגרוע ביותר נעלה עד השורש כלומר O(logn) איטרציות שכל אחת מתקיימת בזמן קבוע (פירוט בהמשך). סה"כ הסיבוכיות ברצף הקריאות הינו O(logn)\*3 = O(logn).

reBalanceDelete:

פונקצייה רקורסיבית שמאזנת את העץ לאחר מחיקה בהינתן צומת שממנו צריך להתחיל\ להימשך תהליך האיזון. שיטת עבודה: בהינתן קלט של צומת מסוים בעץ ומס' השינויים שנעשו בתהליך האיזון הנוכחי עד כה, הפונקציה בודקת את הפרשי הדרגות (הגבהים) של הצומת הנתון ביחס לבניו (ע"י קריאה ל- getRankLeft, getRankRight O(1) ). ולפי זה מסווגת לאיזה מקרה איזון מחדש צריך לקרוא או האם הצומת מאוזן וישר לסיים את הריצה הרקורסיבית. בחלק ממקרה הסיווג יש צורך הבחנה לתתי מקרים באופן הבא:

(3,1) 🡪 (1,1) / (2,1) / (1,2), (1,3) 🡪 (1,1) / (2,1) / (1,2), (2,2) , מצב מאוזן חוקי

לאחר סיווג המקרה ותת המקרה בעת הצורך הפונקציה קוראת לאחת מ-4 פונקציות האיזון המתאימות כאשר 3 מבצעות סוג תיקון מסוים ואת התיקון הסימטרי המקביל הפונקציות הללו יודעות איזה מקרה מדובר ע"י משתנה בוליאני.

בהגעה לצומת שהינו מאוזן, נמשיך ונעלה במעלה העץ ונעדכן את שדה ה-size ע"י קריאה לפונק' הרקורסיבית updateTillRoot.

נשים לב שכל הבדיקות והקריאות במהלך ריצת הפונקציה בכל קריאה קבועות ולכן הפונקציה רצה ב- O(1), (נקראת לכל היותר O(logn) לאחר מחיקה בודדת).

updateTillRoot:

מקבלת צומת ומעדכנת את כל הצמתים מצומת זה ועד השורש ע"י קריאה לפונקציה – updateNode

שמעדכנת את הגובה וגודל של צומת לפי בניו. לכל היותר נעלה את כל גובה העץ ולכן הסיבוכיות הינה O(logn).

reBalanceCase22: פונקציית האיזון לאחר מחיקה של המקרה (2,2), מבצעת דימוט לצומת המבוקש ואז אם הגענו לשורש מחזירה 1, אחרת קוראת ל reBalanceDelete על צומת האבא ועדכון המונה שקיבלנו עד כה +1. כל הפעולות קבועות ולכן O(1) .

reBalanceCase3112

פונקציית האיזון לאחר מחיקה של המקרה (3,1) - (1,2)והמקרה הסימטרי (1,3) -(2,1). מבצעים גלגול ימני ואז גלגול שמאלה לפי הצמתים המתאימים או אם אנחנו עושים את המקרה הסימטרי באופן הפוך (יודעים באיזה מקרה מדובר ע"י משתנה בוליאני בקלט) לאחר מכן מעדכנים את הדרגות של כל הצמתים הרלוונטיים. ואז אם הגענו לשורש מחזירים 6 סך השינויים שעשינו, אחרת קוראת ל reBalanceDelete על צומת האבא ועדכון המונה שקיבלנו עד כה +6. כל הפעולות קבועות ולכן O(1).

reBalanceCase3121

פונקציית האיזון לאחר מחיקה של המקרה (1,3) - (1,2)והמקרה הסימטרי (3,1) -(2,1). מבצעים גלגול שמאלה לפי הצמתים המתאימים או אם אנחנו עושים את המקרה הסימטרי באופן הפוך (יודעים באיזה מקרה מדובר ע"י משתנה בוליאני בקלט) לאחר מכן מעדכנים את הדרגות של כל הצמתים הרלוונטיים. ואז אם הגענו לשורש מחזירה 3 סך השינויים שעשינו, אחרת קוראת ל reBalanceDelete על צומת האבא ועדכון המונה שקיבלנו עד כה +3. כל הפעולות קבועות ולכן O(1).

reBalanceCase3111

פונקציית האיזון לאחר מחיקה של המקרה (1,3) - (1,1)והמקרה הסימטרי (3,1) -(1,1). מבצעים גלגול שמאלה לפי הצמתים המתאימים או אם אנחנו עושים את המקרה הסימטרי באופן הפוך (יודעים באיזה מקרה מדובר ע"י משתנה בוליאני בקלט) לאחר מכן מעדכנים את הדרגות של כל הצמתים הרלוונטיים. זאת פעולת איזון סופית לכן מחזירים 3 סך עלות איזון של הפעולה. כל הפעולות קבועות ולכן O(1).

deleteRetrieve

הפונקציה מקבלת צומת ובודקת כיצד עלינו למחוק אותו ע"י סיווג המצב שלו – עלה \ צומת אונארי שמאלי או ימני \ או שיש לו משפחה (איזה חמודים 😊) לאחר הסיווג הפונקציה קוראת לפונקציית המחיקה המתאימה, שתמחק ותחזיר את הצומת שצריך להתחיל את תהליך איזון העץ ממנו שאותו נחזיר לקורא). כל הבדיקות והקריאות הינם קבועות ולכן O(1).

deleteRetrieveFamily

מוחקת צומת מהעץ במקרה שמדובר בצומת עם 2 בנות. המחיקה מתבצעת באופן שנראה בהרצאה – ניקח את ה- successor של הצומת הנמחק – נמחק אותו (נשמור במשתנה זמני את הצומת שצריך לעבור איזון בעקבות המחיקה) מהעץ ונחליף אותו אם הצומת המבוקש (קיומו של ה-successor מובטח כי יש לצומת המבוקש 2 בנים) ולבסוף נחזיר את הצומת שצריך איזון (אם מדובר במקרה קצה של שורש נעדכן את המצביע). כל הבדיקות מתרחשות בזמן קבוע ולכן O(1).

deleteRetrieveRight או deleteRetrieveLeft:

מבצעת מחיקה לצומת אונארי שמאלי או ימני כפי שנראה בהרצאה ע"י 'דילוג' על הצומת, במקרה קצה של שורש נעדכן את השורש, נחזיר את צומת האב שצריך לעבור איזון, כל הבדיקות מתרחשות בזמן קבוע ולכן O(1).

deleteRetrieveLeaf

מבצעת מחיקה לצומת עלה כפי שנראה בהרצאה ע"י 'דילוג' על הצומת וחיבור האב עם הצומת הוירטואלי, במקרה קצה של שורש נעדכן את השורש, נחזיר את צומת האב שצריך לעבור איזון, כל הבדיקות מתרחשות בזמן קבוע ולכן O(1).

successor

בהינתן צומת הפונקציה מחפשת את ה- successor באופן דומה כפי שנראה בהרצאה: אם יש לצומת בן ימני אז successor הוא המינימום בתת העץ הימני ולכן נמצא אותו ע"י קריאה לפונק' myMin שמחזירה את הצומת המינימלי בתת עץ. אם אין נעלה במעלה העץ עד שנגיע לצומת שהגענו אליו משמאל ואותו נחזיר, אם אין והגענו לnull נחזיר null סה"כ סיבוכיות במקרה הגרוע נעלה מהמקסימום את כל גובה העץ log(n) או נרד כמעט את כל גובה העץ ע"י myMin.

myMax / myMin

הפונקציה מקבלת צומת ומחזירה את המינימום \ מקסימום בתת העץ שלו ע"י ירידה שמאלה \ ימינה בתת העץ כל עוד אפשר, במקרה הגרוע log(n).

המחלקה AVLnode:

כל הפונקציות במחלקה פועלות בצורה קבועה ואינן תלויות בקלט כלומר כולן פעולות בסיבוכיות O(1).

isRealNode()

בודקת אם צומת אינו וירטואלי ע"י בדיקה אם המפתח שלו שונה מ -1.

getKey

מחזירה את המפתח של צומת

getValue

מחזירה את הערך של צומת

setLeft או setRight

מעדכנת את הבן השמאלי או הימני בהתאם לפונקציה בהתאם לקלט

getLeft או getRight

מחזירה את הבן שמאלי או הימני של צומת בהתאמה

setParent

מעדכנת את ההורה של צומת

getParent

מחזירה את ההורה של צומת

setHeight

מעדכנת את הגובה של צומת

getHeight

מחזירה את הגובה של צומת

getRankLeft או getRankRight

מחזירה את הפרש הגבהים של צומת ביחס לבן השמאלי \ ימני שלו

setHeightAlone

מעדכנת את הגובה של הצומת בהתאם לבנים שלו, ע"י בדיקת הגבהים שלהם, לקיחת המקסימום והוספת 1.

setSizeAlone

מעדכנת את גודל הצומת בהתאם לגודל בניו +1.

updateNode:

מעדכנת את הגובה והגודל של צומת ע"י קריאה לפונקציות setHeightAlone, setSizeAlone.