בס"ד

מגישים:

נדב גסנר, שם משתמש nadavgasner, ת.ז 204057566

רועי קורן, שם משתמש roikoren, ת.ז 305428369

**מחלקת WAVLTree**

**שדות:**

root - צומת מטיפוס IWAVLNode, שורש העץ.

EXT - צמתים חיצוניים.

MIN - מצביע לצומת בעל המפתח הקטן ביותר בעץ.

MAX - מצביע לצומת בעל המפתח הגדול ביותר בעץ.

**מתודות:**

(הערה – כל עדכוני הצמתים מתבצעים באמצעות המתודות של WAVLNode המתועדות בהמשך)

**public** WAVLTree()

בנאי, יוצר עץ שהשורש שלו הוא צומת חיצוני. **סיבוכיות:** O(1).

**public** WAVLTree(IWAVLNode root)

בנאי המקבל צומת ויוצר עץ בעל צומת יחיד (שהוא גם השורש), הצומת הנתון. (האבא של שורש העץ מוגדר להיות צומת חיצוני).

**סיבוכיות:** O(1).

**public** **boolean** empty()

מחזיר ערך בוליאני, true אם העץ ריק וfalse אחרת.

בודק את השדה root, אם הוא צומת חיצוני העץ ריק.

**סיבוכיות:** O(1).

**public** IWAVLNode recSearch(IWAVLNode node, **int** k)

מתודת עזר, המבצעת חיפוש בתת העץ של צומת נתון ומחזירה את הצומת בעל המפתח k. אם אין כזה, מחזירה צומת שבנו צריך להיות הצומת בעל המפתח k אם היינו רוצים להכניס צומת בעל מפתח כזה (או null אם העץ ריק).

המתודה פועלת באופן רקורסיבי, תחילה בודקת אם הצומת עצמו חיצוני, אם כן סימן שהעץ ריק (כי בדיקה זו נעשית באופן ישיר על הצומת רק כאשר הצומת שהוזן הוא עצמו חיצוני, וזה יכול לקרות רק אם הפעלנו על השורש של עץ ריק. במהלך הקריאות הרקורסיביות לא נגיע לצומת חיצוני כי הבדיקה בהמשך נעשית לפני הקריאה הרקורסיבית) . אחרת בודקת אם הצומת הנתון הוא בעל מפתח k, אם כן מחזירה אותו, אחרת הולכת לפי הגודל של k ביחס למפתח של הצומת הנתון. אם הליכה בכיוון הזה מביאה צומת חיצוני, סימן שהצומת לא נמצא וצריך להחזיר את הצומת הנתון כאבא פוטנציאלי. אחרת מפעילה את המתודה רקורסיבית על הבן המתאים.

**סיבוכיות:** המתודה עוברת במקרה הגרוע את כל גובה העץ בדרך לצומת אותו היא מחזירה, לכן O(logn)

**public** String search(**int** k)

הפונקציה מחפשת איבר בעל המפתח k. אם קיים איבר כזה, היא מחזירה את הערך השמור עבורו. אחרת היא מחזירה null.

הפונקציה קוראת ל- recSearch על שורש העץ, בודקת אם הצומת שהוחזר הוא אכן בעל מפתח k (כי recSearch יכולה להחזיר null או אבא פוטנציאלי להכנסה) ואם כן מחזירה את ערכו, ואם לא מחזירה null.

**סיבוכיות:** הקריאה ל recSearch היא O(logn), שאר הפעולות רצות בזמן קבוע לכן סה"כ O(logn).

**private** **void** rightRotate(IWAVLNode node)

מתודת עזר, מבצעת פעולת rotation לכיוון ימין סביב צומת נתון. הצומת הנתון יעלה מעלה לכיוון ימין והאבא הנוכחי שלו ירד ויהיה הבן השמאלי שלו, כשכל תתי העצים, הדרגות והגדלים של תתי העצים של כל צומת יעודכנו בהתאם. הפונקציה מעדכנת את כל השדות המתאימים של הצמתים הרלוונטיים.

**סיבוכיות:** .O(1)

**private** **void** leftRotate(IWAVLNode node)

מתודת עזר, סימטרי ל rightRotate.

**private** **int** insertRebalance(IWAVLNode toRebalance)

מתודת עזר, מקבלת צומת לאחר שבוצעה הכנסה שלה לעץ, ומבצעת ריבלנס לעץ. מחזירה את מס' הפעולות שנעשו.

ראשית מאותחל משתנה מטיפוס int הסופר את מספר הפעולות. תחילה בודקים אם לאבא יש שני בנים, כלומר הוא לא היה עלה לפני ההכנסה ולכן אנחנו בcase B במצגת השיעור, ולא צריך לעשות שום פעולה. מוחזר המס' אפס. אחרת מתחילה לולאה, הנעצרת כאשר הגענו לשורש או אם הפרש הדרגות איננו אפס. בכל איטרציה קוראים ל getType() על הצומת (תיעוד בהמשך) וכך נבדק מה הפרשי הדרגות של הצומת עם בניו. כך מוחלט באיזה מקרה אנחנו מתוך שלושת המקרים במצגת. בכל מקרה מבוצעים פעולות הפרומוט והדימוט והרוטייט כפי שהוצגו במצגת, והמשתנה הסופר את הפעולות מעודכן בהתאם. כאשר case 2 ו-3 מטופלים בנפרד בשתי המקרים הסימטריים. אלו גם המקרים בהם התהליך מסתיים והפונקציה מחזירה את הערך של הקאונטר. בcase 1 אולי הבעיה עלתה למעלה, (הצומת עליו אנו עובדים יעודכן להיות האבא של הצומת הנוכחי) ולכן הלולאה עשויה להמשיך.

**סיבוכיות:** במקרה הגרוע הבעיה תפעפע עד לשורש, כלומר נעבור את כל גובה העץ (שאר הפעולות הם בסיבוכיות זמן קבועה), לכן .O(logn)

**public** **int** insert(**int** k, String i)

הכנסת איבר בעל מפתח k וערך i אם לא קיים איבר כזה. אם קיים מוחזר -1. אם לא מחזירה את מספר הפעולות שנעשו.

תחילה יוצרים צומת חדש להכנסה. אחר כך קוראים ל recSearch כדי לקבל את המקום שבו צריך להכניס (כלומר את האבא שבנו צריך להיות הצומת החדש). מתבצעת בדיקה אם צריך לעדכן את השדה MIN או MAX (אם המפתח החדש גדול מהמקס' או קטן מהמינ'). הבנים של הצומת החדש מעודכנים לצומת חיצוני. אם recSearch החזיר null, כלומר העץ ריק והצומת החדש מעודכן להיות השורש. אם recSearch החזיר צומת עם מפתח k, זה אומר שכבר קיים צומת כזה בעץ ומוחזר -1 ולא מבצעת ההכנסה. אחרת הצומת החדש מעודכן להיות הבן (הימני או השמאלי בהתאם למפתחות) של האבא שהוחזר. רצה לולאה עד לשורש שמעדכנת את השדה size בכל הצמתים ש'מעל' הצומת שהוכנס. לאחר מכן מתבצעת קריאה ל insertRebalance, ומס' הפעולות שהוחזר נשמר במשתנה. לאחר מכן רצה לולאה על השורש של העץ (ומעלה אם יש לו כעת אבא לא חיצוני), לבדוק אם לא שינינו אותו בתהליך הריבלנס. מוחזר הערך שהתקבל מ insertRebalance.

**סיבוכיות:** כל הפעולות מתבצעות בזמן קבוע, הקריאה ל recSearchול insertRebalance יכולות לעלות O(logn), וכך גם הלולאות לעדכון הsize והשורש. ביחד .O(logn)

**private** **void** switchRight(IWAVLNode node1, IWAVLNode node2)

מתודת עזר, מבצעת החלפה מלאה של שני צמתים כאשר node1 הוא הבן הימני של node2. כל השדות הרלוונטיים בשני הצמתים מעודכנים, ואם node2 הוא השורש הוא מעודכן בהתאם.

**סיבוכיות:** .O(1)

**private** **void** switchLeft(IWAVLNode node1, IWAVLNode node2)

מתודת עזר, סימטרי ל switchRight.

**private** **void** switchNode(IWAVLNode node1, IWAVLNode node2)

מתודת עזר, אם מדובר במקרה המתאים ל switchLeftאו switchRight מתבצעת קריאה אליהם. אחרת מתבצעות ההחלפות הרלוונטיות (switchLeft ו- switchRight נכתבו לצורך קריאות טובה יותר של הקוד).

**סיבוכיות:** .O(1)

**private** **int** deleteRebalance(IWAVLNode toRebalance)

מתודת עזר, מקבלת צומת לאחר שבוצעה מחיקה של הבן שלו, ומבצעת ריבלנס לעץ. מחזירה את מס' הפעולות שנעשו.

ראשית מאותחל משתנה rebalances מטיפוס int הסופר את מספר הפעולות. מתבצעת קריאה ל getType() על פיה אנו יודעים באיזה מקרה אנחנו מהמקרים שהוצגו במצגת. אם הצומת היא מסוג (2,2), כיוון שמחקנו את אחד מבניו שהיה עלה אז הוא נהיה עלה עכשיו וצריך לבצע לו דימוט. לאחר מכן מתחילה לולאה שממשיכה כל עוד אחד מהפרשי הדרגות של הצומת הוא 3 (או שהגענו לשורש). בכל פעם בודקים באיזה מקרה אנחנו מהמקרים האפשריים ופועלים בהתאם באמצעות קריאה למתודות הרלוונטיות והקאונטר של הפעולות מעודכן בהתאם.

**סיבוכיות:** במקרה הגרוע הבעיה תפעפע עד לשורש, כלומר נעבור את כל גובה העץ (שאר הפעולות הם בסיבוכיות זמן קבועה), לכן .O(logn)

**public** **int** delete(**int** k)

מחיקת איבר בעל מפתח k, מחזירה את מספר פעולות האיזון. אם אין איבר כזה מחזירה -1.

באמצעות קריאה ל recSearch בודקים אם קיים האיבר. אם לא מוחזר -1. אם לא, סימן ש recSearch החזירה את האיבר המבוקש. תחילה ישנו טיפול ידני למקרה שהנמחק הוא השורש ואין לו שני בנים: אם הוא עלה, מעדכנים את העץ להיות ריק ומחזירים 0, ואם יש לו בן אחד, מבצעים החלפה באמצעות switchRight או switchLeft (במקרה זה יש גם טיפול ידני בעדכון MIN וMAX). משווים את האיבר הנמחק למינ' ולמקס' ומעדכנים את השדה אם יש צורך (באמצעות קריאה לpredecessor או לsuccessor על המקס'\מינ', תיעוד בהמשך). אם לצומת הנמחק יש שני בנים, מבצעים החלפה (באמצעות switchNode) עם הsuccessor שלו. מתבצעת המחיקה, ההורה של הנמחק מעודכן לnull, ואם יש לו בן ימני או שמאלי מעדכנים את השדות שלהם כך שיתחברו להורה של הנמחק. מעדכנים את השדות של הנמחק לnull, ואז מעדכנים באמצעות לולאה שרצה עד לשורש את השדה size בכל הצמתים שמעל הנמחק. לאחר מכן מתבצעת קריאה ל deleteRebalance, על ההורה של הנמחק, ומס' הפעולות שהוחזר נשמר במשתנה. לאחר מכן רצה לולאה על השורש של העץ (ומעלה אם יש לו כעת אבא לא חיצוני), לבדוק אם לא שינינו אותו בתהליך הריבלנס. מוחזר הערך שהתקבל מ deleteRebalance.

**סיבוכיות:** כל הפעולות מתבצעות בזמן קבוע, הקריאה ל recSearchול insertRebalance יכולות לעלות O(logn), וכך גם הלולאות לעדכון הsize והשורש. ביחד .O(logn)

**public** String min()

מחזירה את ערכו של האיבר בעץ בעל המפתח המינימלי, או null אם ריק.

המתודה בודקת אם העץ ריק, אם כן מחזירה null, אחרת מחזירה את הערך של הצומת השמור בשדה MIN.

**סיבוכיות:** O(1).

**public** String max()

מחזירה את ערכו של האיבר בעץ בעל המפתח המקסימלי, או null אם ריק.

המתודה בודקת אם העץ ריק, אם כן מחזירה null, אחרת מחזירה את הערך של הצומת השמור בשדה MAX.

**סיבוכיות:** O(1).

**private** **void** recIOWalkKeys(IWAVLNode node, IntStack keysStack)

מתודת עזר, המקבלת צומת ומחסנית מטיפוס IntStack (תיעוד בהמשך), ובאמצעות הליכה רקורסיבית inorder בתת העץ של הצומת ממלאת את המחסנית במפתחות של תת העץ.

תחילה בודקים אם הצומת איננו חיצוני, קוראים למתודה רקורסיבית על הבן השמאלי של העץ, מכניסים את המפתח של שורש תת העץ, ואז קוראים למתודה על הבן הימני.

**סיבוכיות:** O(S), כאשר S הוא מס' האיברים בתת העץ.

**public** **int**[] keysToArray()

מחזיר מערך ממוין של כל המפתחות בעץ, או מערך ריק אם העץ ריק.

מאתחלים מחסנית מטיפוס IntStack, בגודל של הsize של תת העץ של השורש, קוראים ל recIOWalkKeys על השורש ומחזירים את המערך שנמצא במחסנית כעת. (כיוון שההליכה היא inorder המערך ממוין). אם העץ ריק המערך שיוחזר יהיה בגודל 0 ולכן ריק.

**סיבוכיות:** O(n).

**private** **void** recIOWalkString(IWAVLNode node, StringStack infoStack)

מתודת עזר, כמו recIOWalkKeys, רק עם מחסנית מטיפוס StringStack (תיעוד בהמשך). באמצעות הליכה רקורסיבית inorder בתת העץ של הצומת ממלאת את המחסנית בערך של הצמתים בתת העץ.

תחילה בודקים אם הצומת איננו חיצוני, קוראים למתודה רקורסיבית על הבן השמאלי של העץ, מכניסים את הערך של שורש תת העץ, ואז קוראים למתודה על הבן הימני.

**סיבוכיות:** O(S) , כאשר S הוא מס' האיברים בתת העץ (כיוון שאנחנו עוברים בכל צומת בתת העץ).

**public** String[] infoToArray()

מחזיר מערך ממוין לפי המפתחות של כל הערכים בעץ, או מערך ריק אם העץ ריק.

מאתחלים מחסנית מטיפוס StringStack, בגודל של הsize של תת העץ של השורש, קוראים ל recIOWalkString על השורש ומחזירים את המערך שנמצא במחסנית כעת. (כיוון שההליכה היא inorder המערך ממוין לפי המפתחות). אם העץ ריק המערך שיוחזר יהיה בגודל 0 ולכן ריק.

**סיבוכיות:** O(n).

**public** **int** size()

מחזיר את מס' האיברים בעץ, באמצעות בדיקה של השדה size של השורש.

**סיבוכיות:** O(1).

**public** IWAVLNode selectNode(IWAVLNode x ,**int** i)

מתודת עזר, מחזיר את האיבר הi בגודלו של תת העץ של הצומת x. האלגוריתם פועל באופן רקורסיבי על פי מה שהוצג במצגת השיעור, באמצעות שימוש בשדה size של הצמתים. מאתחלים משתנה r להיות גודל תת העץ השמאלי של x ועוד 1, אם אם הוא שווה לr אז x הוא הi בגודלו, אחרת קוראים רקורסיבית למתודה על הבן השמאלי או על הבן הימני כאשר מחפשים את האיבר הi-r בגודלו.

**סיבוכיות:** O(logS) , כאשר S הוא מס' האיברים בתת העץ של x.

**public** String select(**int** i)

הפונקציה מחזירה את הערך של האיבר עם המפתח הi בגודלו בעץ, או null אם העץ ריק או שאין איבר כזה.

תחילה בודקים אם העץ ריק או שגודלו הוא פחות מi. אם כן מחזירים null. אחרת באמצעות לולאה שמתחילה מהאיבר שבשדה MIN עולים למעלה עד שנתקלים בצומת שגודל תת העץ שלה גדול מi וקוראים עליה ל selectNode . בודקים אם לא הוחזר null ואם לא מחזירים את הערך של הצומת שהוחזר.

**סיבוכיות:** O(logi). במהלך העלייה שלנו הגענו לצומת שתת העץ שלו מכיל פחות מi צמתים, ואז עלינו עוד אחד. כלומר אנחנו עובדים בעץ שגובהו הוא 1+O(logi). לכן העלייה והירידה חזרה תעלה סה"כ O(logi).

**public** IWAVLNode getRoot()

מחזיר את השורש של העץ באמצעות השדה root.

**סיבוכיות:** O(1).

**private** IWAVLNode minNode(IWAVLNode node)

מתודת עזר, מחזירה את הצומת בעל המפתח המינימלי מתוך תת העץ של הצומת המוזן כקלט.

המתודה מפעילה לולאה שהולכת שמאלה עד שמגיעה לעלה חיצוני ומחזירה את העלה האחרון.

**סיבוכיות:** O(logS), כאשר S הוא גודל תת העץ של הצומת הנתון.

**private** IWAVLNode successor(IWAVLNode x)

מתודת עזר, מחזירה את הצומת הsuccessor של x.

עובדת על פי האלגוריתם שהוצג בשיעור, אם לצומת יש בן ימני מחזירים את המינימלי של תת העץ הימני שלו באמצעות קריאה ל minNode. אחרת שומרים את ההורה של x במשתנה y ועולים באמצעות לולאה עד שהגענו לשורש או עד שביצענו עלייה ימינה ומחזירים את הצומת אליו הגענו.

**סיבוכיות:** O(logn)במקרה הגרוע, אם עלינו או ירדנו את כל גובה העץ.

**private** IWAVLNode maxNode(IWAVLNode node)

מתודת עזר, סימטרי ל minNode, מחזירה את הצומת בעל המפתח המינימלי מתוך תת העץ של הצומת המוזן כקלט.

**סיבוכיות:** O(logS), כאשר S הוא גודל תת העץ של הצומת הנתון.

**private** IWAVLNode predecessor(IWAVLNode x)

מתודת עזר, מחזירה את הצומת הpredecessor של x.

עובדת על פי האלגוריתם שהוצג בשיעור, אם לצומת יש בן שמאלי מחזירים את המינימלי של תת העץ הימני שלו באמצעות קריאה ל maxNode. אחרת שומרים את ההורה של x במשתנה y ועולים באמצעות לולאה עד שהגענו לשורש או עד שביצענו עלייה שמאלה ומחזירים את הצומת אליו הגענו.

**סיבוכיות:** O(logn)במקרה הגרוע, אם עלינו או ירדנו את כל גובה העץ.

**מחלקת WAVLNode**

**public** **class** WAVLNode **implements** IWAVLNode{

**שדות:**

**private** **int** key; מפתח הצומת -

**private** String value; ערך הצומת -

**private** IWAVLNode left; הבן השמאלי של הצומת -

**private** IWAVLNode right; הבן הימני של הצומת -

**private** IWAVLNode parent; ההורה של הצומת -

**private** **int** rank; הדרגה של הצומת -

**private** **int** size; הגודל של תת העץ של הצומת -

**מתודות:**

**public** WAVLNode()

בנאי, מייצר צומת חיצוני.

**סיבוכיות**: O(1).

**public** WAVLNode(**int** key, String value)

בנאי, מייצר צומת עם מפתח key וערך value.

**סיבוכיות**: O(1).

**public** **int** getKey()

מחזיר את הערך שבשדה key.

**סיבוכיות**: O(1).

**public** String getValue()

מחזיר את הערך שבשדה value.

**סיבוכיות**: O(1).

**public** IWAVLNode getLeft()

מחזיר את הערך שבשדה left.

**סיבוכיות**: O(1).

**public** **void** setLeft(IWAVLNode left)

מעדכן את הערך שבשדה left.

**סיבוכיות**: O(1).

**public** IWAVLNode getRight()

מחזיר את הערך שבשדה right.

**סיבוכיות**: O(1).

**public** **void** setRight(IWAVLNode right)

מעדכן את הערך שבשדה right.

**סיבוכיות**: O(1).

**public** IWAVLNode getParent()

מחזיר את הערך שבשדה parent.

**סיבוכיות**: O(1).

**public** **void** setParent(IWAVLNode parent)

מעדכן את הערך שבשדה parent.

**סיבוכיות**: O(1).

**public** **boolean** isRealNode()

בודק אם ההורה של הצומת הוא null, אם כן סימן שהוא עלה חיצוני ומוחזר false. אחרת מוחזר true.

**סיבוכיות**: O(1).

**public** **int** getSubtreeSize()

מחזיר את הערך שבשדה size.

**סיבוכיות**: O(1).

**public** **int** getRank()

מחזיר את הערך שבשדה rank.

**סיבוכיות**: O(1).

**public** **void** setRank(**int** rank)

מעדכן את הערך שבשדה rank.

**סיבוכיות**: O(1).

**public** **void** promote()

מגדיל באחד את הערך שבשדה rank.

**סיבוכיות**: O(1).

**public** **void** demote()

מקטין באחד את הערך שבשדה rank.

**סיבוכיות**: O(1).

**public** **void** setSize(**int** size)

מעדכן את הערך שבשדה size.

**סיבוכיות**: O(1).

**public** **int**[] getType()

מחשב את הפרשי הדרגות בין הצומת לבניו ומחזיר מערך שבו המס' הראשון הוא הפרש הדרגות עם הבן שמאלי והשני הוא הפרש הדרגות עם הבן הימני.

**מחלקת StringStack**

מחסנית, מכילה שדות בהם מערך של מחרוזות ומשתנה end מטיפוס int המייצג את המקום בו אנחנו נמצאים במחסנית. הבנאי מגדיר את גודל המערך, והמתודה push מכניסה את האיבר הנתון בקלט למקום בו אנו נמצאים ומעדכנת את end להיות מקום אחד קדימה (פועלת בזמן קבוע). המתודה getArray מחזירה את המערך.

**מחלקת IntStack**

מחסנית, מכילה שדות בהם מערך של integers ומשתנה end מטיפוס int המייצג את המקום בו אנחנו נמצאים במחסנית. הבנאי מגדיר את גודל המערך, והמתודה push מכניסה את האיבר הנתון בקלט למקום בו אנו נמצאים ומעדכנת את end להיות מקום אחד קדימה (פועלת בזמן קבוע). המתודה getArray מחזירה את המערך.

**מדידות:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | 10000 | 3.4244 | 3.0064 | 15 | 13 |
| 2 | 20000 | 3.3981 | 3.0107 | 18 | 13 |
| 3 | 30000 | 3.4196 | 3.0091 | 17 | 14 |
| 4 | 40000 | 3.41605 | 3.004225 | 17 | 13 |
| 5 | 50000 | 3.39052 | 3.01448 | 17 | 14 |
| 6 | 60000 | 3.39991 | 2.99925 | 18 | 14 |
| 7 | 70000 | 3.41981 | 3.00222 | 19 | 14 |
| 8 | 80000 | 3.413375 | 3.004475 | 18 | 14 |
| 9 | 90000 | 3.40066 | 3.0062111 | 19 | 14 |
| 10 | 100000 | 3.40496 | 3.01205 | 19 | 14 |

כיוון שראינו שamortized מס' פעולות האיזון לפעולת insert/delete הוא O(1), זה אומר שאם נעשה ממוצע של פעולות האיזון על פני n פעולות נקבל מס' קבוע של פעולות איזון פר הכנסה\מחיקה. זה מה שציפינו לקבל בטבלה ואכן ניתן לראות שבכל אחד מהניסויים מס' הפעולות הממוצע הוא בערך קבוע (בערך 3.4 בהכנסה ו3 במחיקה), זאת על אף שכפי שניתן לראות היו הכנסות\מחיקות שהצריכו 13 עד 19 פעולות איזון. משמעות המדידות היא שהניתוח בכיתה לסיבוכיות זמן הריצה amortized למס' פעולות איזון לסדרת מחיקות\הכנסות אכן עומד במבחן המציאות.