**מסמך תיעוד AVLTree**

**סטודנט 1:** אליאור סגל, 326128006, eliorsegal.

**סטודנט 2:** רועי קישון, 205836810, roeekishon.

המחלקה AVLTree מייצגת מופעים של עצי AVL. יש בה את תת המחלקה AVLNode היורשת הממשת את המנשק IAVLNode. כל מופע של AVLNode מהווה node (או צומת) בעץ AVLTree כלשהו.

תיעוד הפונקציות במימוש:

Empty – בודקת אם העץ ריק, O(1).

SearchNode – מוצאת node מסויים בעץ לפי ה-key שלו. מחזירה null אם העץ ריק. מחזירה את ה-node שמחופש, ואם הוא לא קיים את ה-node האמיתי האחרון על ידי שימוש בתכונותיו של עץ חיפוש בינארי. O(log(n).

Successor – מחזירה יורש של node. Null אם ה-node לא אמיתי\קיים. מחזירה את ה-node היורש inorder. O(log(n)).

updateHeightToRoot – מעדכנת את גובה העץ בצורה מאוזנת על ידי קידומים, הורדות וסיבובים, מ-node x הניתן לה כארגומנט. עושה שימוש ב-balance factor (ההפרש בין גובה תתי העצים השמאלי והימיני), והסיבוכיות המקסימלית שלה תלויה בהפרשה הגובה של ה-node ממנו מאזנים לבין ה-root של העץ.

Rotate – מבצעת סיבוב של node בעץ לצורך איזונו. כל סיבוב לוקח זמן קבוע, O(1).

getBF – מחזירה את ערך ה-balance factor בין 2 הבנים של node מסויים, O(1).

Search – עושה שימוש ב-searchNode לעיל. O(log(n).

Insert – מוסיפה node לעץ עם מפתח וערך שניתנים כארגומנט. במידה והמפתח קיים מחזירה -1. סיבוכיות O(log(n) מכיוון שהמפתח המוסף הוא עלה וצריך לאזן את העץ לאחר ההוספה. מחזירה את מספר הסיבובים שנדרשו לצורך איזון העץ לאחר ההוספה.

Delete – מוחקת node עם מפתח נתון. מחזירה -1 אם המפתח לא קיים. אחרת מוחקת את ה-node ושמה במיקומו את ה-node המתאים במידת הצורך. לבסוף מחזירה את מספר הסיבובים שנדרשו לצורך איזון העץ לאחר המחיקה. O(log(n).

Min – מחזירה את הערך מה-node הקטן ביותר.O(log(n).

Max – מחזירה את הערך מה-node הגדול ביותר. O(log(n).

keysToArray – מחזירה מערך ממויין עם כל מפתחות העץ (ע"י הוספתם inorder למערך).O(n).

infoToArray – מחזירה מערך עם כל הערכים מכלל ה-node בעץ (ללא המפתחות, אך ממויין על פי המפתחות). O(n).

Size – מחזירה את גודל העץ (מספר ה-nodes שבו), סופרת inorder.O(n).

getRoot – מחזירה את node השורש (root). O(1) מכיוון שערך זה נשמר עבור כל עץ.

Split – מקבלת מפתח x ומחזירה מערך בו יש 2 עצים. עץ אחד מכיל את כל ה-nodes עם ה-keys שקטנים מ-X, ועץ שני מכיל את הגדולים מ-X. לכל היותר O(log(n) במקרה ו-x הוא עלה. על ידי שימוש יעיל ב join.

Join – מאחדת 2 עצים בעזרת תנאי קדם של node x מקשר. סיבוכיות תלויה בהפרש הגבהים בין העצים + 1.

\*כלל הפונקציות הקטנות במחלקה AVLNode שיורשת מ-IAVLNode לוקחות O(1), והן פועלות על פי התיאור הנדרש במנשק.

**תשובות לשאלות**

שאלה 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 38883 | 979807 | 35187 |
| 2 | 7998000 | 85763 | 3986495 | 76813 |
| 3 | 31996000 | 187523 | 16048587 | 182050 |
| 4 | 127992000 | 407043 | 64229413 | 394634 |
| 5 | 511984000 | 878083 | 256308881 | 831820 |

1. מספר החילופים במערך a הוא מספר הזוגות i<j כך שa[i]>a[j]. במערך ממוין הפוך כל 2 זוגות של אינדקסים מקיימים את "חילוף" ולכן מספר החילופים הוא כמספר הזוגות כלומר

עלות החיפושים במיון AVL עבור מערך ממוין-הפוך הוא סכום עלויות פעולות ה-search לאורך ההכנסות. עלות search בודד הוא אורך המסלול מהאיבר המקסימלי אל מיקום ההכנסה. המערך ממוין הפוך כל הכנסה היא של איבר מינימלי ולכן המסלול מהאיבר המקסימלי הוא אל השורש ואז אל האיבר המינימלי. בעץ AVL עם x מפתחות מסלול זה הוא בערך log(x) (הוא *).*

כלומר, לאחר שהכנסנו חצי מהמערך כל הכנסה נוספת היא *) ומכיוון שיש עוד פעמים אותה העלות של הכנסת חצי זה הוא ולכן העלות של המיון כולו הוא .*

אבל הכנסה בחצי הראשון של המערך היא לכל ה*יותר ) מכיוון שיש בעץ פחות מ מפתחות כלומר היא .*

בסך הכול קיבלנו כי *העלות של החיפושים היא .*

1. הערכים מסעיף א' מתאימים ל*ניתוח מסעיף ב' שכן מספר החילופים זהה במדויק וכן עלות החיפושים מקיימת קשר בקירוב טוב מאוד ללינארי עם :*
2. מספר החילופים h שווה ל כאשר הוא ה"אחריות" של כל איבר i, עלות החיפוש הנוסף בגלל הוא . בנוסף מתקיים כי:

בנוסף יש n הכנסות ולכן בסך הכול עלות החיפוש היא

שאלה 2



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי i | עלות join ממוצע עבור split אקראי | עלות join מקסימלי עבור split אקראי | עלות join ממוצע עבור split של האיבר מקסימלי בתת העץ השמאלי | עלות join מקסימלי עבור split של איבר מקסימלי בתת העץ השמאלי |
| 1 | 2.75 | 5 | 2.7777777 | 11 |
| 2 | 2.3636363 | 4 | 2.4545455 | 13 |
| 3 | 2.5 | 8 | 2.3076923 | 15 |
| 4 | 2.5833333 | 8 | 2.6666667 | 16 |
| 5 | 2.8461537 | 6 | 2.6923077 | 17 |
| 6 | 2.357143 | 7 | 2.4666667 | 18 |
| 7 | 2.7692308 | 9 | 2.6666667 | 19 |
| 8 | 2.9375 | 7 | 2.4444444 | 20 |
| 9 | 2.5263157 | 8 | 2.7777777 | 22 |
| 10 | 2.9444444 | 8 | 2.35 | 23 |

ראינו בשיעור כי אם עלינו לחבר עצים כאשר

*וכן*

*אז עלות ה*split *היא לכל היותר*

*אך מספר ה*Join *שאנחנו עושים הוא (בשני המקרים) (בגלל שזה תלוי בגובה העץ פחות גובה השורש).*

*כלומר העלות של* Join *ממוצע הוא*

*התוצאות אכן מתיישבות עם ניתוח הסיבוכיות התאורטי.*

1. *כאשר אנו עושים* split *על האיבר המקסימלי בתת העץ השמאלי, עץ המפתחות הגדולים ממנו (*split()[1]*) נשאר ריק עד "להגעתו" של האלגוריתם לשורש ואז הוא עושה* join *של עץ ריק עם תת העץ הימני ולכן עלות* join *זה היא המקסימלית והיא* O *של גודל העץ הימני כלומר .*

*התוצאות מתיישבות עם ניתוח הסיבוכיות התאורטי (שכן אנו מכפילים את* n *אך העלות המקסימלית עולה במספר קבוע.*