**חלק מעשי**

**Class: AVLTree**

מימוש של מילון על ידי עץ AVL. אובייקט של מילון הוא צומת בעץ (יש לו מפתח וערך).

שדות:

*root* – שורש של עץ.

מתודות:

*\_set\_root(node)* – מתודת עזר פנימית. מקבלת צומת ומגדירה אותו בתור שורש חדש של העץ. מוחקת את ההורה של הצומת*. O(1)*

*search(key)* – מקבלת ערך של מפתח ומחזירה את הצומת של המפתח אם המפתח הזה נמצא בעץ, אחרת מחזירה *None*. עובדת כמו חיפוש בעץ בינארי רגיל, לכן עוברת מקסימום בכמות הצמתים שווה לגובה העץ. כלומר *O(log(n))*

*insert(key. val)* – מקבלת מפתח וערך שאותם רוצים להכניס למילון, ויוצרת צומת כזה בעץ. במימוש שלנו יש ילדים וירטואליים, אז חלק הראשון של הכנסה הוא פשוט למצוא ילד וירטואלי מתאים למפתח ולהכניס בו אותם. חיפוש כזה כידוע לוקח *O(log(n))*. החלק השני הוא לבצע איזון אחרי ההכנסה. בעזרת הפונקציה הפנימית *\_rebalance\_from\_node\_to\_root* נעבור על כל הצמתים בדרך מהצומת שהכנסנו עד השורש, נבדוק את הbalance factor שלהם ונחשב מחדש את הגובה. חישוב גובה ואיזון יחיד לוקחים זמן קבוע. אז סה"כ המתודה תיקח *O(log(n))* ולא משנה שזה היה פעמיים.

*\_rebalance\_subtree(node, bf, inserting)* – פונקציית עזר פנימית. מאזנת את תת העץ הנתון שהשורש שלו הוא node. לפי הbf שהוא balance factor של ה*node* ולפי הbalance factor של הילדים של השורש נבצע אחד מ4 הגילגולים. פרמטר *inserting* נועד בשביל השיקול הזה – תנאים לגלגולים הם שונים עבור איזון אחרי ההכנסה ואחרי המחיקה או חיבור. לוקחת *O(1)*

*\_left\_rotation(node)* – מתודת עזר פנימית. מבצעת גלגול שמאלה לפי הצומת הנתון. מחזירה את הצומת הגבוה ביותר עכשיו. *O(1)*

*\_right\_rotation(node)* – מתודת עזר פנימית. מבצעת גלגול ימינה לפי הצומת הנתון. מחזירה את הצומת הגבוה ביותר עכשיו. *O(1)*

*\_recalc\_height(node)* – מתודת עזר פנימית שנוצרה לשם נוחות. מחשבת מחדש את גובה הצומת ומחדשת את הערך באובייקט של צומת. *O(1)*

*\_calc\_bf(node)* - מתודת עזר פנימית שנוצרה לשם נוחות. מחשבת את הbalance factor של ה*node*. *O(1)*

*\_add\_virtual\_children(node)* – מתודת עזר פנימית. מוסיפה ילדים וירטואליים לnode. *O(1)*

*delete(node)* – מתודת מחיקת האובייקט ממילון (צומת מעץ). חלק הראשון של המחיקה הוא מחיקה כרגיל מעץ בינארי שכידוע לוקחת *O(log(n))*. חלק השני הוא לעבור מצומת הנמחק פיזי עד השורש ולאזן הכל מחדש בעזרת *\_rebalance\_from\_node\_to\_root* שהיא לוקחת גם *O(log(n))* זמן. סה"כ *O(log(n))*.

*\_rebalance\_from\_node\_to\_root(starting\_node, inserting)* – מתודת עזרת פנימית. עוברת על כל צומת מ*starting\_node* עד השורש מפעילה *\_rebalance\_subtree* עליהם (*inserting* הועבר ישר ל*\_rebalance\_subtree*), כלומר מאזנת את תתי עץ שלהם. הפעלה לוקחת זמן קבוע, אבל העיקר של העלות הוא לעבור על כל הצמתים וזה לוקח *O(log(n)).*

*\_replace\_node(old\_node, new\_node)* – מתודת עזר פנימית. מחליפה את old\_node בעץ ב*new\_node*. בפרט, מעתיקה את על הקשרים של צומת השנה ומפעילה את  *\_connect\_in\_place\_of\_deleted* (כי המתודה משומשת רק בהקשר של מחיקה) ואת *\_disconnect\_node\_internally*. הכל לוקח *O(1)*

*\_connect\_in\_place\_of\_deleted(node, child)* – מתודת עזר פנימית. מחברת את ההורה של *node* לילד שלו – *child*. לוקחת *O(1)*

*\_disconnect\_node\_internally(node)* – מתודת עזר פנימית. מנתקת את כל הקשרים "היוצאים" מהnode. לוקחת *O(1)*

*avl\_to\_array()* – מחזירה את מערך עם כל הזוגות של מפתח\ערך שנמצאים במילון וסדר עולה לפי המפתח. פעולת רקורסיבית בעזרת הפונקציית העזר *\_avl\_to\_array\_util(arr, node)* שבהינתן המערך שבסוף מחזירים מפונקצייה הראשית וצומת node, באופן רקורסיבי עוברת על כל צמתי של תת העץ של צומת הנתון ומכניסה אותם למערך. סה"כ לקוחת *O(n)*

*size()* – מחזירה את כמות האובייקטים במילון. פועלת באופן רקורסיבי בעזרת הפונקציית עזר *\_size\_util* שמקבלת צומת node וסופרת כמות האיברים בתת העץ שלה. סה"כ לוקחת *O(n)*

*split(node)* – מפצלת את העץ לפי צומת הנתונה לשני עצים, כך שבאחד יהיו כל הצמתים שמפתחות שלהם הם גדולים מצומת הנתונה ובשני – קטנים ומחזירה את "מיני מערך" עם שניהם. מתודה עוברת מצומת הנתונה עד השורש של העץ המקורי ומפעילה מתודת join על הילד השני של הצומת אליו הגענו והעץ הרלוונטי ("גדול" או "קטן" מהצומת הנתונה).לוקחת .*O(log(n))*

*join(tree2, key, val)* – מחברת שני עצים – *tree2* ו*self* לפי הצומת עם מפתח וערך *key/val* (כאשר מפתחות של *tree2* ו*self* נמצאים משני הצדדים של אי שוויון עם מפתח הנתון). עובדת באופן הבא: אם אין הבדל בגבהים – מחברת ישר עם צומת החדשה בתור שורש (זמן קבוע). אם יש – עוברת בעץ היותר גבוה עד הצומת עם גובה של עץ הקטן (*O(log(n))*), מחברת את העץ היותר קטן ותת העץ של צומת שהגענו אליו משני הצדדים של צומת שיצרנו, ואז מחברת אותה להורה של הצומת שמצאנו (מחליפה אותו בעצם). כל החיבורים לוקחים זמן קבוע, אך אחרי החיבור האחרון נרצה לעבור מצומת החדשה על השורש החדש של העץ ולאזן (נפעיל *\_rebalance\_from\_node\_to\_root*). זה גם יקח *O(log(n))*. סה"כ *O(log(n))*.

**חלק ניסויי/תיאורטי**

1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 2.4 | 11 | 2.4 | 11 |
| 2 | 2.4 | 12 | 2.6 | 13 |
| 3 | 2.4 | 14 | 2.5 | 14 |
| 4 | 2.5 | 15 | 2.4 | 15 |
| 5 | 2.5 | 16 | 2.6 | 16 |
| 6 | 2.5 | 17 | 2.4 | 17 |
| 7 | 2.5 | 18 | 2.2 | 18 |
| 8 | 2.5 | 19 | 2.6 | 20 |
| 9 | 2.6 | 17 | 2.6 | 21 |
| 10 | 2.6 | 15 | 2.4 | 22 |

2. עלות join ממוצע עבור split אקראי- עבור צומת אקראי node בעץ AVL סיבוכיות הפיצול היא  *כלומר קיים כך שעלות split קטנה מ .*

*כמו כן בפונקציה split אנו מבצעים join כ פעמים ושאר הפעולות לוקחות זמן קבוע לכן העלות הממוצעת של join ב split היא*

עלות join ממוצע עבור split של האיבר מקסימלי בתת העץ השמאלי – ההסבר הקודם נכון לכל צומת בעץ ובפרט עבור צומת זה לכן *גם פה העלות קבועה .*

3. עלות join מקסימלי עבור split של איבר מקסימלי בתת העץ השמאלי – עלות פעולת join של 2 עצי AVLהיא הפרשי גבהי העצים + 1.בעץ AVL treeעם n צמתים, נבחין בכך שבפעולת ה split הזו אנו עושים בהתחלה tree.height – 1 פעולות join של תת עץ עם תת העץ האח השמאלי שלו (שיש להם אבא משותף) ומשום שמדובר בעץ AVL הפרשי הגבהים שלהם חסומים על ידי 1 ולכן עלות join חסומה על ידי 2. לבסוף יש join של עץ ריק עם כל תת העץ הימני של tree והצומת tree.root שמשום שגובה תת העץ הימני הוא הפרש הגבהים בין תת העץ הימני לעץ הריק הוא  *ולכן זה גם עלות ה join. קיבלנו ש*עלות join מקסימלי *במקרה זה היא .*

*עלות join מקסימלי עבור splitאקראי -* בעץ AVL treeעם n צמתים*, בפעולת split אנו עושים join על תתי עץ של tree .מספר הצמתים שלהם חסום מלמעלה על ידי n, ולכן הגובה שלהם הוא* ולכן גם עלות פעולת join *היא ובפרט פעולת ה join המקסימלית.*