**Class: AVLTree**

מימוש של מילון על ידי עץ AVL. אובייקט של מילון הוא צומת בעץ (יש לא 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 מספר של המפתח ומחזירה אותו אם צומת עם מפתח הזה נמצא בעץ, אחרת מחזירה None. עובדת כמו חיפוש בעץ בינארי רגיל, לכן עוברת מקסימום בכמות הצמתים שווה לגובת 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)). חלק השני הוא לבצע איזון אחרי ההכנסה. בכזרת הפונקציה הפנימית \_rebalance\_from\_node\_to\_root נעבור על כל הצמתים בדרך מצומת שהכנסנו עד השורש, נבדוק את הbalance factor שלהם ונחשב מחדש את הגובה. חישוב גובה ואיזון יחיד לוקחים זמן קבוע. אז סה"כ מתודה תקח O(log(n)) ופחות משנה שזה היה פעמיים.

\_rebalance\_subtree(node, bf, inserting) – פונקציית עזר פנימית. מאזנת את תת העץ הנתון ששורש שלו הוא node. לפי הbf שהוא balance factor של הnode ולפי הbalance factor של הילדים של השורש נבצע אחד מ4 הגילגולים. פרמטר inserting נועד בשביל שיקול 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 גובת הצומת ומחדשת את הערך באובייקט של צומת. O(1)

\_calc\_bf(node) - מתודת עזר פנימית שנוצרה לשם נוחות. מחשבת את הbalance factor של הnode. 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 מstarting\_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 בnew\_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). join לוקחת O(log(n)) ולעבור על כל הורה גם לוקח O(log(n)). אז סה"כ O(log2 (n))

join(tree2, key, val) – מחברת שני עצים – tree2 וself לפי הצומת עם מפתח וערך key/val (כאשר מפתחות של tree2 וself 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 גבוע 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 בחיבורים לוקחים זמן קבוע, אך אחרי בחיבור 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)).