מבני נתונים – תרגילים מעשי 1

מגישים: איתמר גולדברגר ויואב לוי (מספרי ת.ז: 010001, 20432618)

**תיעוד**

WAVLTree (class)

זהו עץ WAVL כפי שהוגדר לנו בכיתה. כל איבר בעץ הוא WAVLNode בעל String השומר מידע וint- השומר key. מיון האיברים נעשה לפי הkey.

כאשר מכניסים איבר ראשון לעץ אז השדות root,minode,maxnode מצביעים אליו. כאשר העץ "מתרוקן" (נמחק האיבר האחרון) אז שלושת שדות אלו מצביעים חזרה ל-null.

Root (field)

שדה זה שומר מצביע אל WAVLNode שנקרא שורש העץ. הוא מאותחל ל-null. אם מוחקים את השורש אז פונקציית Delete מחפשת לו מחליף (הsuccesor).

זהו למעשה "ראש" העץ, וניתן להגיע לכל איבר בעץ אם מתחילים ממנו ומחפשים את האיבר השמאלי/ימני כדרוש.

Minode (field)

שדה זה שומר מצביע ל- WAVLNode בעל המפתח המינימלי בעץ. גם הוא מאותחל לnull. בכל הכנסת איבר בודקים האם האיבר החדש קטן מהמינימלי, ואם כן אז זהו המינימלי החדש. כאשר נמחק האיבר המינימלי עובר המצביע לאיבר המינימלי הבא (predeseccor).

Maxnode (field)

שדה זה שומר מצביע ל- WAVLNode בעל המפתח המקסימלי בעץ. גם הוא מאותחל לnull. בכל הכנסת איבר בודקים האם האיבר החדש גדול מהמקסימלי, ואם כן אז זהו המקסימלי החדש. כאשר נמחק האיבר המקסימלי עובר המצביע לאיבר המקסימלי הבא (sucessor).

Size: (field)

שדה זה שומר int המייצג את מספר הWAVLNode בעץ. הוא מאותחל ל-0 ובכל הוספת איבר גדל באחד, ובכל הורדת איבר קטן באחד.

Empty (method) O(1)

פעולה זו בוחנת האמת העץ ריק. היא עושה זאת על ידי נסיון גישה לשורש העץ דרך השדה root, ואם הוא null משמע שהעץ ריק (אם הוא תוחזק כראוי). לכן היא תחזיר true. אחרת תחזיר false.

זוהי כמות פעולות קבועה ולכן סיבוכיות הפונקציה היא O(1).

Search (method) O(log(n))

פונקציה זו מקבל int ומחפשת WAVLNode בעץ בעל מפתח זה, ומחזירה את השדה info שלו. מאחר ומימשנו פונקציית חיפוש מאוחר יותר בשם searchNode אשר מחזירה מצביע לWAVLNode – הפונקציה משתמשת בפונקציית העזר שתתואר מאוחר יותר כדי למצוא מצביע לWAVLNode הרלוונטי, ואז ניגשת לשדה הinfo שלו כדי להחזיר ערך זה.

זוהי כמות פעולות קבועה בנוסף לכמות הפעולות של searchNode ולכן יש להן אותה סיבוכיות אסימפטוטית.

Insert (method) O(log(n))

פעולה זו יוצרת WAVLNode עם מפתח מסוג int ומידע מסוג String, ומוסיפה אותו במקומו הראוי לפי מבנה WAVLTree. לאחר מכן היא מאזנת את העץ כך שיישאר חוקי. בנוסף, היא מחזירה את כמות פעולות האיזון שעשתה בתור int. היא עושה זאת כך:

תחילה הפונקצייה שומרת מצביע לשורש העץ, ויוצרת את האיבר המבוקש. היא נעזרת במתודה empty כדי לבחון האם העץ ריק –

אם כן היא מאתחל את האיבר החדש להיות השורש, האיבר המינימלי והאיבר המקסימלי בעץ. לבסוף היא מחזירה 0 משום שלא עשתה פעולות איזון.

אם לא היא ממשיכה הלאה בקוד. כעת היא בוחנת האם מפתח האיבר החדש גדול ממפתח האיבר המינימלי או קטן ממפתח האיבר המינימלי, ואם כן היא משנה את המצביעים של שדות אלו בהתאם.

השלב הבא הוא חיפוש בינארי רגיל למקום בו הWAVLNode החדש אמור להימצא – כלומר אם מצאנו את המפתח אז יש להחזיר -1 משום שמפתח זה כבר נמצא בעץ ואין להרשות כפילויות, אם המפתח החדש קטן מהמפתח הנוכחי אז נחפש באגף שמאל של תת העץ ואחרת נחפש באגף ימין. כאשר המצביע הבא הוא null נבין שבמיקום זה אמור להימצא האיבר החדש, נזכור לשמור מצביע לאיבר האמיתי האחרון (שאינו null), ונקשר את האיבר החדש במקומו. לאחר מכן נגדיל את שדה size ונקרא לפונקציית העזר rebalanceInsert על האיבר שהוספנו כדי לאזן את העץ על מנת שיישאר תקין.

גובה עץ WAVL בעל n איברים הוא לכל היותר log(n), ומאחר ובכל מעבר על לולאת הwhile ירדנו רמה בעץ, לכל היותר עד קצה העץ, אז כמות הפעולות האסימפטוטית שביצענו היא log(n). לאחר מכן נקראת פונקציית העזר של פעולת האיזון – ואם פונקצייה זו בעלת סיבוכיות של לא יותר מlog(n) פעולות באופן אסימפטוטי אז סיבוכיות הריצה המצטבר של Insert הוא אכן log(n), ואת זאת נראה בהמשך.

rebalanceInsert (method) O(log(n))

תחילה מאותחל מספר פעולות האיזון ל-0.פונקצייה זו מקבלת מצביע אל האיבר שנוסף, ובוחנת באיזה מקרה בסיס מדובר.

אם הוספנו איבר להורה מדרגה 1 (מקרה B במצגת) אז העץ כבר מאוזן וניתן להחזיר את כמות פעולות האיזון (0) ללא המשך טיפול.

אחרת (מקרה A במצגת) יש להעלות את דרגת ההורה, לספור פעולת איזון, ולקרוא לפונקציית העזר הרקורסיבית rebalanceInsertRec המקבלת מצביע להורה של האיבר שהוכנס, ואת מספר פעולות האיזון שנעשו עד כה (בשלב זה 1).

מדובר בכמות פעולות קבועה, ולכן סיבוכיות הריצה של פונקציה זו תהיה זהה לסיבוכיות הריצה של הפונקציה rebalanceInsertRec.

rebalanceInsertRec (method) O(log(n))

caseInsrt

rotate

delete

rebalanceDelete

rebalanceDeleteRec

caseDelete

rankRight

rankLeft

isLeft

removeLeaf

replaceUnari

findSuc

findPre

searchNode

min

max

keysToArray

infoToArray

size

WAVLNode

Key

Info

Rank

Right

Left

parent

WAVLNode

infoToString

keyToString