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))**

פונקצייה זו מקבלת את האיבר שנוסף לעץ ומשמשת כמעיין פונקציית מעטפה שעושה איזונים בסיסיים וקוראת לפונקציית עזר נוספת רקורסיבית כדי לוודא שהעץ מאוזן ולאזן אותו. היא מחזירה את מספר פעולות האיזון. 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 מלבד אלו הנעשות על ידי פונקציית העזר, ולכן סיבוכיות הריצה של פונקציה זו תהיה זהה לסיבוכיות הריצה של הפונקציה rebalanceInsertRec.

**rebalanceInsertRec (method) O(log(n))**

פונקצייה זו מקבלת מצביע לאיבר כלשהו בעץ החל ממנו יש לאזן ואת מספר הפעולות שנעשו עד כה בעץ. היא מאזנת באופן רקורסיבי את העץ (כשדרוש היא קוראת לעצמה עם האיבר הבא אותו יש לאזן), ומחזירה את כמות הפעולות החדשה שנעשו עד כה.תנאי העצירה לפוקציה רקורסיבית זו הוא אם האיבר שהוזן הוא שורש העץ. במקרה כזה פעולת האיזון הושלמה ויש להחזיר את מספר פעולות האיזון שנעשו.

תחילה נעזרת הפונקצייה בפונקציית העזר caseInsrt אשר מקבלת את האיבר הנדרש ובוחנת באיזה מקרה מדובר (כלומר מה הם הפרשי הדרגות בסביבה הקרובה של תת העץ). פונקציית העזר מחזירה int אשר מסמן את מספרו הסידורי של המקרה על מנת שנדע כיצד להתמודד איתו. עוד על הסימונים בתיאור פונקציית העזר.

אם הפונקצייה **החזירה** **0** אז אין בעיה בעץ וניתן לסיים את התהליך ולהחזיר את מספר פעולות האיזון הכולל.

**במקרה 1** הפונקציה תעלה את דרגת ההורה של האיבר הנבחן, תספור זאת כפעולת איזון, ותקרא מחדש לפונקצייה רקורסיבית זאת כאשר מזינים לה מצביע להורה של האיבר הנבחן ("גילגלנו" את הבעיה מעלה), ואת כמות פעולות האיזון שנעשו עד כה. המקרה הסימטרי זהה.

\*הבהרה: כאשר נתייחס למקרים סימטרים נעזר בסימן ' / ' כדי להסביר את הפעולה הנעשות בכל מקרה בנפרד, בהתאמה. למשל אם נתאר את מקרים 3/6 נכתוב שנעשית פעולה3/פעולה6. כלומר פעולה3 נעשית במקרה 3 ופעולה6 נעשית במקרה6.

**במקרים 2/5** הפונקציה תבצע סיבוב ימינה/שמאלה על האיבר בעזרת פונקציית העזר rotate, תוריד את דרגתו של האיבר שנמצא מימין/שמאל לאיבר הנבחן ותסיים על הפעולה על ידי החזרת כמות פעולות האיזון שנעשו עד כה.

**במקרים 3/6** הפונקציה תשמור מבעוד מועד מצביע לאיבר מימין/שמאל האיבר הנבחן (צ'אק/נוריס בקוד), תבצע סיבוב סביב צ'אק/נוריס שמאלה/ימינה ואז ימינה/שמאלה בעזרת פונקציית העזר rotate, ואז תשנה את הדרגות הדרושות: דרגת האיבר הנבחן והאיבר מימין/שמאל לצ'אק/נוריס תרד, ודרגתו של צ'אק/נוריס תעלה. לבסוף תסיים הפונקצייה את פעולתה ותחזיר את כמות פעולות האיזון שנעשו עד כה.

במקרה והפונקציה לא עשתה דבר היא תחזיר 0.

מדובר בכמות פעולות קבועה בכל קריאה לפונקציה, ולכל היותר הבעיה "גולגלה" מתחילת העץ ועד שורש העץ, כלומר בכל רמה של העץ נעשתה כמות פעולות גבוהות וגובה העץ חסום ע"י log(n) ולכן סיבוכיות הזמן של פונקציה זו הוא log(n) (בידיעה שפונקציות העזר אינן גורעות מזמן ריצה זה).

**caseInsrt (method) (O(1)**

פונקציה זו מקבלת מצביע לאיבר אותו יש לבחון, ומחזירה את מספרו הסידורי של המקרה המדובר לפי הפרשי הדרגות בסביבה המיידית שלו. תחילה היא שומרת מצביע נוסף להורה שלו. כעת היא משתמשת בפונקציות העזר rankLeft ו- rankRight כדי לבחון מה הפרש הדרגות בין האיבר שמוזן אליהן והאיבר שמשמאל/ימין בהתאמה. המספרים שהיא מחזירה מייצגים את מספרי המקרים כפי שהוצגו במצגת, כאשר למקרים הסימטריים הוספנו 3 למספר הסידורי (המקרה הסימטרי של 2 הוא 5 לדוגמה). אם הפרשי הדרגות של ההורה הם 1-0 מדובר במקרה 1. אחרת:

אם הפרש הדרגות של ההורה מימין ושמאל הוא 2 ו0 בהתאמה :

אם הפרש הדרגות של האיבר מימין הוא 2 מדובר במקרה 2, אחרת אם הפרש הדרגות של האיבר משמאל הוא 2 מדובר במקרה 3.

אחרת אם הפרש הדרגות של ההורה משמאל ומימין הוא 2 ו0 בהתאמה:

אם הפרש הדרגות של האיבר מימין הוא 2 מדובר במקרה 6, אחרת אם הפרש הדרגות של האיבר משמאל הוא 2 מדובר במקרה 5.

אם לא הגענו לאף אחד מהמקרים הבעייתיים נחזור 0.

סיבוכיות פונקצייה זו היא O(1) משום שהיא מבצעת כמות פעולות קבועה.

**Rotate**

פונקצייה זו מקבלת איבר וערך בוליאני ומבצעת על האיבר פעולת "סיבוב" שמאלה אם הוזן לה "אמת" וימינה אחרת. היא אינה מחזירה דבר. בהתאם לסיבוב שמאלה/ימינה היא שומרת מצביע לאיבר משמאל/ימין לאיבר הנבחן (b במצגת של אינסרט, ולכן נקרא לו בי מעתה), מצביע נוסף להורה של האיבר (z במצגת) ולהורה של z (מחוץ למצגת, נקרא לו סבא). כעת היא משימה את z בתור הילד השמאלי/ימני של האיבר, משימה את הסבא בתור ההורה של האיבר, ואם הסבא אינו null אז גם משימה את האיבר בתור הילד החדש של הסבא (לאחר שבחנה האם הוא צריך להיות ילד ימין או שמאל שלו). אם הסבא הוא null, משמע שz היה השורש וכעת נשים את האיבר בתור השורש ונשים את null בתור ההורה שלו. כעת נשים את האיבר להיות ההורה של z ונשים את b להיות הילד הימני/שמאלי של z. אם b אינו null אז נשים את z להיות ההורה של b.

**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**