**תיעוד וסיבוכיות המתודות – תרגיל מעשי 1**

**מגישים:**

|  |  |  |
| --- | --- | --- |
|  | **מגיש א'** | **מגיש ב'** |
| **שם פרטי** | אדם | אסף |
| **שם משפחה** | טובי | מיכאלוביץ' |
| **תעודת זהות** | 215334822 | 200637270 |
| **שם משתמש אוניברסיטאי** | adamtuby | michaelovits |

**סיבוכיות ותיעוד:**

empty()

**סיבוכיות:** O(1)

פונקציה זו בודקת האם העץ ריק, היא עושה זאת באמצעות הבדיקה אם השורש של העץ הוא צומת וירטואלי או לא. בדיקה זו לוקחת אכן O(1) והיא מתבצעת על ידי מתודת העזר   
**public** **boolean** isRealNode() השייכת למחלקה AVLNode. פירוט של מתודה זו יהיה בהמשך המסמך. ובכן, שיטה זו נכונה מכך שכאשר אנו יוצרים מופע חדש של המחלקה AVLTree (כלומר יוצרים עץ), הוא מייצר שורש שהינו עץ וירטואלי. וכאשר אנו נכנסים איברים לעץ, צמתים אמיתיים יתפסו את מקומם של הצמתים הוירטואלים, ולכן השורש של העץ אמור להיות צומת אמיתי לאחר פעולת ההכנסה הראשונה לעץ. מכך שאם השורש הינו עץ וירטואלי אז העץ ריק.

search(**int** k)

**סיבוכיות:** O(log n)

נשים לב שמתודה זו לא מבצעת כמעט כלום אלא רק משתמשת בערך המוחזר שמחזירה מתודת העזר plain\_search(**int** k) שנתעד ממש עוד רגע. מתודת עזר זו מחזירה את הצומת בעץ שהמפתח שלו הוא k, ובמידה ולא קיים צומת כזה בעץ, היא תחזיר null. אז במידה ומתודת עזר זו מחזירה אכן צומת, נחזיר את ה info של אותו צומת. במידה והצומת null, נחזיר null. סיבוכיות מתודה זו היא כסיבוכיות מתודת העזר. סיבוכיות זאת היא אכן הסיבוכיות היעילה ביותר עבור חיפוש בעץ AVL. ראינו זאת בהרצאה.

plain\_search(**int** k)

**סיבוכיות:** O(log n)

מתודה זו היא מתודת עזר בשביל Search, הנועדה למנוע שכפולי קוד של המתודה Search, שכן מתודת Search מחזירה את הערך בוליאני שהצומת עם המפתח k מחזיק, וכן plain\_search מחזירה את המצביע לצומת המחזיק את המפתח k.   
מתודה זו מבצעת חיפוש נורמטיבי בעץ בינארי כמו שראינו בכיתה. ובכן, כיוון שמדובר בעץ AVL, אז גובה העץ הוא O(log n), ולכן נבצע לכל היותר O(log n) איטראציות בשיטת החיפוש שלמדנו בכיתה. כל איטראציה בחיפוש זה מכילה מספר קבוע של השוואות (לכל היותר 2). במידה ומצאנו צומת המקיימת את הנדרש בעץ, נחזיר את הצומת. במידה ולא נמצא הצומת, יוחזר null. על כן סיבוכיות מתודה זו היא כנ"ל.

insert(**int** k, **boolean** i)

**סיבוכיות:** O(log n)

מתודה זו תכניס צומת חדש עם המפתח k ועם ה info – i. היא בנוסף תחזיר את מספר הגילגולים ושינויי הגובה שנעשו בתהליך תיקון עברייני ה AVL לאחר ההכנסה. נשים לב כי מתודה זו כמו search אינה המתודה העיקרית, אלא רק מעטפת למתודת העזר הרקורסיבית insert\_rec שנתעד ממש עוד רגע. במידה וקיים כבר צומת בעץ עם מפתח k, היא מחזירה -1 ולא עושה כלום (ניתן לבדוק אם קיים כבר הצומת בעץ באמצעות plain\_search). במידה וזה לא נמצא בעץ, היא תיצור מערך changes[] שמחזיק ערך integer אחד בלבד שיכיל את מספר הגלגולים ועדכוני הגבהים במהלך תיקון עברייני ה AVL.  
כעת נעדכן את שורש העץ להיות הצומת החדש שתחזיר קריאת המתודה:  
insert\_rec(**this**.getRoot(), **new** AVLNode(k,i,0), changes)

ונחזיר כעת את changes[0].   
סיבוכיות המתודה היא כמובן כסיבוכיות הקריאה של מתודת העזר עם הפרמטרים הנ"ל. סיבוכיות זאת היא אכן היעילה ביותר עבור הכנסה בעץ AVL. ראינו זאת בהרצאה.

insert\_rec(AVLNode node, AVLNode new\_node, **int**[] change\_info)

**סיבוכיות:** O(log n)

מתודה זו היא מתודת העזר והמתודה הרקורסיבית שמשומשת על ידי המתודה insert.   
נתחיל בלהסביר מה הם הפרמטרים/ארגומנטים;  
**node** – זו יהיה הצומת הנוכחי שעליו אנו מתמקדים. כלומר, במידה וקראנו למתודה זו כאשר node הוא צומת כלשהו בצומת לאחר ההכנסה, אז קריאה זו תדאג לבדוק אם הוא עבריין AVL, ובמידה וכן, תבצע גלגולים כנדרש באמצעות מתודות העזר rightRotation(AVLNode N) ובאמצעות   
leftRotation(AVLNode N) שנתעד ממש עוד רגע. אם לא היה עבריין AVL, תבדוק אם גובהו השתנה. במידה וכן, תעלה את change\_info[0] באחד. במידה וכן היה עבריין AVL, תעלה את change\_info[0] באחד. בנוסף לזאת המתודה תעדכן את פרטיו, כגון גובהו, גודל העץ ששורשו נמצא בצומת המדובר, ושאר השדות שצומת AVLNode מחזיר בהם (כגון XOR). תהליך העדכון ייתבצע באמצעות מתודת עזר של המחלקה AVLNode, הנקראת update\_info() שתתועד בהמשך.

**new\_node** – זה יהיה מצביע לצומת החדש שאנו רוצים להכניס לעץ. צומת זה נוצר בקריאה הראשונה למתודת עזר זו מתוך המתודה insert, על ידי **new** AVLNode(k,i,0).

**change\_info** – ארגומנט זה יהיה מצביע למערך שמכיל בתוכו את מספר הצמתים שדרשו גילגול או עדכון גובה. נדרש לשלוח ארגומנט זה בכל קריאת רקורסיה, שכן אני לא מספיק בהכרה עם משתנים גלובלים בjava.

מבוצעות O(log n) קריאות רקורסיה למתודה זו, שכן תהליך ההכנסה מבקר ב O(log n) צמתים בלבד, וכן אנחנו קוראים לקריאת רקורסיה חדשה אך ורק כאשר אנחנו מבקרים בצומת חדש בתהליך ההכנסה (ביקור משמע המעבר בצומת בתהליך החיפוש של המיקום הטריוויאלי עבור הצומת החדש).  
מתודה זו תבדוק אם node הוא עבריין AVL באמצעות מתודת העזר של המחלקה הפנימית AVLNode, הנקראת **public** **int** BalanceFactor() מתודה זו מחזירה את ה Balance Factor של הצומת, ומבצעת גלגולים לפי הערך שלו ולפי דרישות שונות ועמוקות יותר של גלגולים.

delete(**int** k)

**סיבוכיות:** O(log n)

delete\_rec(AVLNode node, **int** key, **int**[] change\_info)

**סיבוכיות:** O(log n)

replace(AVLNode node, AVLNode tmp)

**סיבוכיות:** O(1)

rightRotation(AVLNode N)

**סיבוכיות:** O(1)

leftRotation(AVLNode N)

**סיבוכיות:** O(1)

min()

**סיבוכיות:** O(log n)

max()

**סיבוכיות:** O(log n)

keysToArray()

**סיבוכיות:** O(n)

keysToArray\_rec(AVLNode N, **int**[] arr, **int** pos)

**סיבוכיות:** O(n)

infoToArray()

**סיבוכיות:** O(n)

infoToArray\_rec(AVLNode N, **boolean**[] arr, **int** pos)

**סיבוכיות:** O(n)

size()

**סיבוכיות:** O(1)

getRoot()

**סיבוכיות:** O(1)

successor(AVLNode node)

**סיבוכיות:** O(log n)

prefixXor(**int** k)

**סיבוכיות:** O(log n)

prefixXor\_rec(AVLNode node, **boolean**[] out)

**סיבוכיות:** O(log n)

succPrefixXor(**int** k)

**סיבוכיות:** O(log n + n)

**public** **class** AVLNode

**מדידות**