אוניברסיטת תל אביב

סמסטר ב' תשפ"ב

**מבני נתונים - פרויקט חקר תכנותי מספר 1 - עץ דרגות**

**הקדמה:**

בתרגיל זה שני חלקים:

1. חלק המעשי: מימוש של List באמצעות עץ AVL. עמודים 1-2 במסמך זה מתארים את החלק הזה.
2. חלק ניסויי-תיאורטי: בהתבסס על המימוש מהחלק המעשי, נבצע מספר "ניסויים" עם ניתוח תיאורטי נלווה ושאלות חקר. עמודים 3-5 מתארים את החלק הזה.

**שימו לב:** בסוף המסמך (עמוד 5) ישנן הוראות הגשה – הקפידו לפעול לפיהן. **תאריך הגשה: 24.4**. בנוסף, יש לעקוב אחר השרשור הנעוץ בפורום בו נפרסם הבהרות חשובות.

**חלק מעשי**

**דרישות**

בתרגיל זה נממש את ה ADT **רשימה** באמצעות עץ AVL. לכל איבר ברשימה יש ערך (info). המימוש יהיה **בשפת python 3.9 וצריך להיות מבוסס על קובץ השלד המופיע באתר הקורס**.   
הפעולות שיש לממש הן:

|  |  |
| --- | --- |
| **פעולה** | **תיאור** |
| empty() | הפונקציה מחזירה ערך TRUE אם ורק אם הרשימה ריקה. |
| retrieve(i) | הפונקציה מחזירה את ערך האיבר במקום ה-i אם קיים, אחרת היא מחזירה None. |
| insert(i, s) | הכנסת איבר בעל ערך s לרשימה במקום ה-i, במידה וקיימים לפחות i איברים ברשימה. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בסה"כ בשלב תיקון העץ על מנת לשמר את תכונת האיזון. |
| delete(i) | מחיקת האיבר במקום ה-i ברשימה, אם הוא קיים. הפונקציה מחזירה את מספר  פעולות האיזון שנדרשו בסך הכל בשלב תיקון העץ על מנת לשמר את תכונת האיזון. אם לא קיימים מספיק איברים ברשימה הפונקציה מחזירה . |
| first() | מחזירה את ערך האיבר הראשון ברשימה, או None ברשימה ריקה. |
| last() | מחזירה את ערך האיבר האחרון ברשימה, או None ברשימה ריקה. |
| listToArray() | הפונקציה מחזירה מערך המכיל את איברי הרשימה לפי סדר האינדקסים, או מערך ריק אם הרשימה ריקה. |
| length() | הפונקציה מחזירה את מספר האיברים ברשימה. |
| split(i) | הפונקציה מקבלת אינדקס שנמצא ברשימה. על הפונקציה לחצות אותה לשתי רשימות, לפני האינדקס i ואחרי האינקס i (פירוט בקובץ שלד). יש לממש את הפונקציה על פי המימוש שנלמד בהרצאה לפיצול בעץ AVL בסיבוכיות . |
| concat(lst) | הפונקציה מקבלת רשימה. על הפונקציה לשרשר אותה אל סוף הרשימה הנוכחית. על הפעולה לרוץ בזמן . יש להחזיר את הערך המוחלט של הפרש הגבהים של עצי הAVL שמוזגו. |
| search(val) | החזרת האינדקס הראשון ברשימה בו מופיע הערך val, או אם לא קיים כזה. |

**בנוסף למימוש הפונקציות האלו, יש לממש את מחלקת AVLNode כפי שמתואר בקובץ**. מטעמי נוחות, נדרוש שלכל עלה יהיו 2 בנים "וירטואליים", כלומר, צמתים שלא מייצגים איברים במבנה הנתונים. באופן זה, נוח יותר לממש גלגולים מכיוון שלכל צומת יהיו 2 בנים וזה חוסך טיפול במקרי קצה.

למחלקה AVLNode יש את המתודות הבאות (המפרט המלא נמצא בקובץ השלד):

getHeight – מחזיר את הגובה של הצומת, או אם הצומת הוא וירטואלי.

getValue – מחזיר את הinfo של הצומת או None אם הצומת הוא וירטואלי.

getLeft – מחזיר את הבן השמאלי של הצומת, או None אם אין כזה.

getRight – מחזיר את הבן הימני של הצומת, או None אם אין כזה.

getParent - מחזיר את ההורה של הצומת, או None אם אין כזה.

isRealNode – מחזיר TRUE אם הצומת מייצג צומת אמיתי בעץ (קרי: צומת שאינו וירטואלי).

**הערות חשובות:**

1. **המימוש יבוצע על ידי מילוי קובץ השלד. במידת הצורך, ניתן להרחיב את המימוש** (למשל להוסיף פונקציות עזר שאינן מופיעות בשלד), אך **אסור לשנות את הגדרות הפונקציות לעיל**. על כל הפונקציות/מחלקות להופיע בקובץ יחיד.
2. **אין להשתמש באף מימוש ספרייה של מבנה נתונים.**

**סיבוכיות**

יש לתעד בקוד ובמסמך נפרד (ביותר פירוט) את סיבוכיות זמן הריצה במקרה הגרוע (האסימפטוטית, במונחי O הדוקים) של כל פונקציה **שמכילה לולאות/רקורסיה**, כתלות במספר האיברים בעץ n. עליכם להשיג סיבוכיות זמן ריצה (במקרה הגרוע ביותר) נמוכה ככל הניתן עבור כל אחת מהפונקציות.

**פלט**

אין צורך בפלט למשתמש.

**תיעוד**

בנוסף לבדיקות אוטומטיות של הקוד שיוגש, קובץ המקור ייבדק גם באופן ידני. חשוב להקפיד על תיעוד לכל פונקציה, וכמות סבירה של הערות. **הקוד צריך להיות קריא**, בפרט הקפידו על בחירת שמות משתנים ועל אורך השורות.

יש להגיש בנוסף לקוד גם מסמך תיעוד חיצוני. המסמך יכלול את תיאור המחלקה שמומשה, ואת תפקידו של כל חבר במחלקה. עבור כל פונקציה במחלקה יש לפרט מה היא עושה, כיצד היא פועלת **ומה סיבוכיות זמן הריצה שלה**. בפרט, אם פונקציה קוראת לפונקציית עזר, **יש** להתייחס גם לפונקציית העזר בניתוח. עבור פונקציות שעולות זמן קבוע יספיק להביא רק תיאור קצר ולא לפרט את ניתוח הסיבוכיות.

**חלק ניסויי/תיאורטי**

**שאלה 1:**

1. .

|  |  |  |  |
| --- | --- | --- | --- |
| מספר סידורי i | ניסוי 1 - הכנסות | ניסוי 2 - מחיקות | ניסוי 3 - הכנסות ומחיקות לסירוגין |
| 1 | 3801 | 2068 | 2938 |
| 2 | 7409 | 4218 | 5821 |
| 3 | 15365 | 8548 | 11753 |
| 4 | 30197 | 14143 | 23309 |
| 5 | 60103 | 34134 | 47033 |
| 6 | 122286 | 68461 | 93640 |
| 7 | 242532 | 136365 | 186567 |
| 8 | 484901 | 273075 | 374073 |
| 9 | 969532 | 544948 | 747992 |
| 10 | 1937314 | 1093255 | 1518231 |
| O(n) | O(1.9n) = O(n) | O(n) | O(1.5n) = O(n) |

**שאלה 2:**

בשאלה זו נרצה לנתח את העלות של פעולות ה-join המתרחשות במהלך ביצוע split.  
נגדיר את העלות של פעולת join בודדה כערך המוחלט של הפרש הגבהים של העצים שאוחדו.

* לצורך הניתוח, נבנה עצי AVL בגדלים שונים. מספר האיברים שנכניס לעץ יהיה כאשר i=1,…,10. כלומר, עבור i=1 העץ בגודל 2000, ועבור i=10 העץ בגודל כמיליון. את האיברים יש להכניס **בסדר אקראי**.
* לכל גודל של עץ, נבצע 2 ניסויים נפרדים (ולכן נבנה שני עצים עם אותו סדר הכנסה, לכל גודל):
  + בניסוי אחד נבצע split על אינדקס אקראי ברשימה.
  + בניסוי שני נבצע split על האינדקס המקסימלי בתת העץ השמאלי של השורש.

1. תעדו בטבלה שלהלן את העלות הממוצעת של פעולות ה-join ואת העלות של פעולת ה-join היקרה ביותר.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |  |  |  |  |
| ... |  |  |  |  |
| 10 |  |  |  |  |

1. נתחו באופן תיאורטי את העלות של **join ממוצע לשני התרחישים** (split אקראי או על האיבר המסוים שבחרנו), והסבירו אם התוצאות מתיישבות עם ניתוח הסיבוכיות התאורטי.
2. נתחו באופן תיאורטי את העלות של **join מקסימלי בתרחיש אחד** של split על האיבר המסוים שבחרנו, והסבירו אם התוצאות מתיישבות עם ניתוח הסיבוכיות התאורטי.

**שאלה 3:**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| מספר פעולות האיזון בממוצע  מספר סידורי | עץ AVL  סדרה חשבונית | עץ ללא מנגנון איזון  סדרה חשבונית | עץ AVL  סדרה מאוזנת | עץ ללא מנגנון איזון  סדרה מאוזנת | עץ AVL  סדרה אקראית | עץ ללא מנגנון איזון  סדרה אקראית |
| 1 | 1.984 | 499.5 | 0.995 | 0.995 | 2.077 | 2.49 |
| 2 | 1.9915 | 999.5 | 0.9975 | 0.9975 | 2.0085 | 2.43 |
| 3 | 1.993 | 1499.5 | 0.998 | 0.998 | 2.019 | 2.481 |
| 4 | 1.9955 | 1999.5 | 0.99875 | 0.99875 | 2.017 | 2.45 |
| 5 | 1.996 | 2499.5 | 0.9989 | 0.9992 | 2.023 | 2.39 |
| 6 | 1.9966 | 2999.5 | 0.999 | 0.999 | 2.007 | 2.42 |
| 7 | 1.997 | 3499.5 | 0.9991 | 0.99914 | 2.01 | 2.4 |
| 8 | 1.9978 | 3999.5 | 0.9993 | 0.99937 | 2.02 | 2.46 |
| 9 | 1.998 | 4499.5 | 0.9995 | 0.9995 | 2.007 | 2.41 |
| 10 | 1.9981 | 4999.5 | 0.9996 | 0.9996 | 2.02 | 2.42 |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| עומק הצומת המוכנס בממוצע  מספר סידורי | עץ AVL  סדרה חשבונית | עץ ללא מנגנון איזון  סדרה חשבונית | עץ AVL  סדרה מאוזנת | עץ ללא מנגנון איזון  סדרה מאוזנת | עץ AVL  סדרה אקראית | עץ ללא מנגנון איזון  סדרה אקראית |
| 1 | 8.9 | 499.5 | 7.98 | 7.98 | 8.7 | 11.28 |
| 2 | 9.9 | 999.5 | 8.98 | 8.98 | 9.7 | 12.98 |
| 3 | 10.6 | 1499.5 | 9.63 | 9.63 | 10.3 | 13.83 |
| 4 | 10.9 | 1999.5 | 9.97 | 9.97 | 10.77 | 13.83 |
| 5 | 11.3 | 2499.5 | 10.36 | 10.36 | 11.09 | 14.968 |
| 6 | 11.6 | 2999.5 | 1063 | 1063 | 11.34 | 14.734 |
| 7 | 11.8 | 3499.5 | 10.81 | 10.81 | 11.60 | 15.89 |
| 8 | 11.9 | 3999.5 | 10.97 | 10.97 | 11.74 | 15.9 |
| 9 | 12.18 | 4499.5 | 11.18 | 11.18 | 11.969 | 16.07 |
| 10 | 12.36 | 4999.5 | 11.36 | 11.36 | 12.1 | 17.803 |

מה הייתם מצפים שתהיינה התוצאות, והאם התוצאות האמיתיות מסתדרות עם ציפייה זו? הסבירו.  
תשובה:

מספר פעולות איזון:

עץ AVL:

1. סדרה חשבונית:

כפי שננתח בהמשך הכנסה בסדרה מאוזנת נראה כי כל צומת מתעדכן לפחות פעם אחת.

כל פעם שנתחיל למלא שורה חדשה נצטרך בשלב מסוים לעדכן את כל הגבהים לפחות פעם אחת בעץ. בנוסף על מנת להגיע לבסוף לתצורה מאוזנת של העץ בכל הכנסה כמעט נרצה לבצע מספר פעולות גלגול על מנת ״להזיז את הצמתים ימינה״ לכן עבור כל הכנסה יהיו כ2 פעולות גלגול. דומה לתוצאת הניסוי

1. סדרה מאוזנת: נשער כי לכל פעולת הכנסה נצטרך לאזן (לעדכן גבהים) לכל היותר פעם אחת לכל הכנסה.

אם ננסה לחשב את כמות האיזונים כאשר מכניסים את האיברים באופן המיטבי נראה כי אין בכלל פעולת גלגול אלה רק פעולות עדכון גובה.

נסתכל על עץ מאוזן ומלא בעומק מסוים ונחשב את כמות האיזונים בהכנסה של שורה חדשה בעץ.

כל עלה בעץ המקורי יקבל שני בנים כך שגובה כל אחד מההורים(גם הקדומים) של אותו עלה יקבלו פעולת עדכון אחת, ז״א בכל הכנסה של שורה חדשה בעץ והגעה למצב של עץ מלא נבצע כ n/2 (ערך עליון) עדכונים, אם נסכום בעבור כל גובה בעץ n/2 עדכונים כאשר n הוא כמות הצמתים בעץ עד אותו הגובה נקבל סדרה הנדסית השואפת לאחד. תוצאה זו דומה לתוצאת הניסוי.

1. סדרה אקראית: היינו מצפים כי זה יהיה זהה לתוצאות הניסוי הראשון.

עץ ללא מנגנון איזון:

1. סדרה חשבונית: כיוון שאנו מכניסים בתחילת הסדרה כל פעם, נקבל שבכל הכנסה נצטרך לעדכן את כלל הצמתים מעל הצומת ולכן נקבל סכום של סדרה הנדסית מאפס ועד n כפי שקיבלנו.
2. סדרה מאוזנת: כיון שהסדרה מאוזנת נקבל סיטואציה זהה לניתוח בעץ AVL ולכן הציפייה זהה.
3. סדרה אקראית:

עומק העץ:

עץ AVL:

1. עץ מאוזן:

עבור עץ מאוזן נרצה לבנות סדרה לטובת חישוב העומק הממוצע:

בגובה הראשון יש עלה אחד בעומק אפס

לאחר מכן שני עלים בעומק אחד

לאחר מכן ארבע עלים בעומק שתים

לאחר מכן שמונה עלים בעומק שלוש

.

.

בעומק h נקבל 2^h עלים

ניתן לחשוב על כך גם שכמחצית מהאיברים בעומק המקסימלי, מחציתם בעומק אחד מתחת וכן הלאה ולכן אם נציב ונחשב נקבל תוצאות דומות למה שקיבלנו.

1. סדרה חשבונית: נצפה כי התוצאות יהיו זהות ואולי קצת יותר גבוהות מהתוצאה שבעץ המאוזן כיוון שהעומק הממוצע אמור קרוב לעץ המאוזן כיוון שבשאיפה העץ מאזן את עצמו כל הזמן עד כדי קבוע.
2. אקראי: נצפה כי זה יהיה בין המקרה הגרוע בו מכניסים כל פעם באיבר הראשון לבין המקרה מיטבי בו הסדרה מאוזנת ואין גלגולים.

|  |  |  |  |
| --- | --- | --- | --- |
| **:AVLNode()** | | | |
| **פונקציותget():** | | | |
| **סיבוכיות** | **סיבוכיות במילים** | **תיאור במילים** | **שם הפונקציה** |
| **O(1)** | **איטרציה יחידה להחזרת שדה** | **מחזירה את שדה הdepth של הnode** | **getDepth()** |
| **מחזירה את שדה הsize של הnode** | **getSize()** |
| **מחזירה מצביע לבן השמאלי** | **getLeft()** |
| **מחזירה מצביע לבן הימני** | **getRight()** |
| **מחזירה מצביע להורה** | **getParent()** |
| **מחזירה את שדה הbf של הnode** | **getBalanceFactor()** |
| **מחזירה את שדה הvalue של הnode** | **getValue()** |
| **מחזירה את שדה הheight של הnode** | **getHeight()** |
| **O(logn)** | **מטייל כלפי מעלה/מטה בעץ לכן חסום על ידי גובה העץ שהוא log n** | **מחזירה מצביע לעלה המקסימלי בתת העץ** | **getMax()** |
| **מחזירה מצביע לעלה המינמלי בתת העץ** | **getMin()** |
| **מחזירה מצביע לאיבר בעל האינדקס העוקב בעץ** | **getSucceor()** |
| **מחזירה מצביע לאיבר בעל האינדקס הקודם בעץ** | **getPredecessor()** |
| **פונקציות set():** | | | |
| **O(1)** | **איטרציה יחידה לעדכון שדה** | **מעדכנת שדה Depth** | **setDepth(x)** |
| **מעדכנת שדה Size** | **setSize(i)** |
| **מעדכנת שדה Left (מצביע לAVLNode())** | **setLeft(node)** |
| **מעדכנת שדה Right (מצביע לAVLNode())** | **setRight(node)** |
| **מעדכנת שדה Parent (מצביע לAVLNode())** | **setParent(node)** |
| **מעדכנת שדה Value** | **setValue(x)** |
| **מעדכנת שדה Height** | **setHeight(h)** |
| **פונקציות update():** | | | |
| **O(1)** | **עדכון שדה באמצעות משיכת נתונים מהילדים, ללא לולאות או חזרות** | **מעדכן את שדה size באמצעות סכום size הבנים + 1** | **updateSize()** |
| **מחשב את הBF באמצעות הפרש גבהי הילדים** | **updateBalanceFactor()** |
| **מעדכן את שדה height באמצעות מציאת מקסימום הגובה בין הבנים + 1** | **updateHeight()** |
| **קריאה ל3 פונקציות מסיבוכיות O(1)** | **קורא לפונקציות: updateSize, updateHeight, updateBalanceFactor** | **updateNodeInfo()** |
| **פונקציות is: מחזירות Boolean** | | | |
| **O(1)** | **גישה לאחד השדות של הצומת הנוכחית ובדיקה בהתאם** | **מחזיר האם הצומת מייצג צומת אמיתי באמצעות בדיקת הגובה מחזיר True אם כן** | **isRealNode()** |
| **מחזיר Trueאם לצומת קיים בן שמאלי בלבד** | **haveOnlyLeftSon()** |
| **מחזיר Trueאם לצומת קיים בן ימני בלבד** | **haveOnlyRightSon()** |
| **מחזיר Trueאם הצומת הוא בן שמאלי** | **isLeftSon()** |
| **מחזיר Trueאם הצומת הוא בן ימני** | **isRightSon()** |
| **מחזיר True אם לצומת קיים הורה** | **haveParent()** |
| **מחזיר Trueאם הצומת הוא עלה(ללא בנים)** | **isLeaf()** |
| **פונקציות נוספות:** | | | |
| **O(1)** | **עדכון שדה אחד בצומת** | **מגדיל את שדה size באחד** | **incraeseSizeByOne()** |
| **מקטין את שדה size באחד** | **decreaseSizeByOne()** |
| **O(1)** | **שינוי מצביעים של ההורה ושל הילד בלבד** | **הפונקציה נקראת כחלק מפעולת הdelete אחראית על מחיקת צומת בעל בן יחיד באמצעות שינוי המצביעים של הילד שלו ושל ההורה שלו לטובת ״התעלמותו מהעץ״** | **bypass()** |

|  |  |  |  |
| --- | --- | --- | --- |
| **מחקלת AVLTreeList():** | | | |
|  | | | |
| **O(logn)** | **קוראת לפונקציה treeSelect שהיא מסיבוכיות o(logn)** | **מחזירה את האיבר הi במבנה**  **קוראת לtreeSelect ומקבלת ממנו מצביע לאיבר הi ואז מחזירה את שדה הvalue** | **Retrive(i)** |
| **קוראת לפונקציה treeSelectRec שהיא מסיבוכיות o(logn)** | **פונקצית מעטפת לtreeSelectRec, מקבלת אינדקס וnode ומחזירה מצביע לצומת הi בתת עץ של node** | **treeSelect(root,i)** |
| **מטיילת בגבהי העץ שחסום על ידי o(logn)** | **באמצעות טיול בעץ ביחס לשדות הsize מחזירה מצביע לצומת הi** | **treeSelectRec(root,i)** |
| **קוראת לפונקציה insertRec שהיא מסיבוכיות o(logn)** | **פונקצית מעטפת לinsertRec** | **Insert(i, val)** |
| **מטיילת בגבהי העץ שחסום על ידי o(logn)** | **מייצרת צומת חדש עם ערך val ומכניסה אותו לאינדקס הi** | **insertRec(i, v)** |
| **מטיילת בגבהי העץ שחסום על ידי o(logn)** | **מוחקת את האיבר הi ממבנה הנתונים** | **Delete(i)** |
| **קוראת לפונקציה fixTreeשהיא מסיבוכיות o(logn)** | **אחראית על איזון העץ לאחר מחיקה, קוראת לfixTree** | **fixTreeAfterDeletion(node)** |
| **O(n)** | **קוראת לפונקציה listToArrayRec שהיא מסיבוכיות o(logn)** | **מחזירה רשימה לפי מבנה הנתונים, פונקצית מעטפת לlistToArrayRec** | **listToArray()** |
| **הליכת inorder על כל איברי העץ** | **מחזירה רשימה לפי מבנה הנתונים באמצעות inorder walk** | **listToArrayRec(node)** |
| **O(1)** | **משנה את שדות המצביע לעץ בלבד** | **מוחק את נתוני העץ לעץ נקי** | **deleteAllTree()** |
| **פונקציות איזון** | | | |
| **O(logn)** | **מטייל כלפי מעלה בעץ והאיבר שהוכנס/נמחק לכן חסום ובכל צומת מבצע גלגול במידת הצורך** | **פונקציה האחראית לאיזון העץ לאחר פעולות מחיקה והכנסה, מקבל bool המתאר האם זאת פעולה לאחר מחיקה או הכנסה** | **fixTree(node,bool)** |
| **O(1)** | **שינוי מצביעים לצמתים A,B ועדכון השדות שלהם** | **גלגול ימינה ואז שמאלה, מחזירה 2 כמספר האיזונים, קוראת לפונקציוח leftRotate, rightRotate בהתאם** | **rightThenLeft(B)** |
| **גלגול שמאלה ואז ימינה, מחזירה 2 כמספר האיזונים, קוראת לפונקציוח leftRotate, rightRotate בהתאם** | **leftThenRight(B)** |
| **גלגול שמאלה מחזירה 1** | **leftRotate(B)** |
| **גלגול ימינה מחזירה 1** | **rightRotate(B)** |
| **פונקציות get:** | | | |
| **O(1)** | **גישה/שינוי שדות העץ** | **אחראית על עדכון הצמתים שלקחו חלק בגלגול** | **updateNodesInfo(A,B)** |
| **מחזירה את שדה Value של האיבר באינדקס 0** | **First()** |
| **מחזיר את שדה הvalue של האיבר באינקדס האחרון** | **Last()** |
| **מחזירה את אורך הרשימה** | **Length()** |
| **מחזירה מצביע לאיבר הראשון בעץ** | **getFirstNode()** |
| **מחזירה מצביע לאיבר האחרון בעץ** | **getLastNode()** |
| **מחזירה true אם העץ ריק** | **Empty()** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**פירוט נוסף על פונקציות נבחרות:**

1. **treeSelectRec(root,i):**

**מקבלת: מצביע לצומת ואינדקס i.**

**מחזירה: מצביע לצומת הi ביחס לroot**.

**תיאור פעולת הפונקציה: באמצעות רקורסיה והשוואת שדות הsize נטייל בעץ. במידה וi גדול מגודל תת העץ השמאלי(+1) נדע שהאינדקס i נמצא בתת עץ הימני, במידה וi קטן משדה הsize זה אומר שיש פחות איברים בתת עץ השמאלי ולכן האיבר נמצא שם. כך בהתאם נלך לאחד מתתי העצים ונקרא לפונקציה רקורסיבית שוב עם שינוי הi בהתאם.**

**סיבוכיות: כיוון שבכל צומת אנו מבצעים השוואת לאינדקס ז״א עבודה קבועה אנו תלויים בגובה העץ שכידוע בעץ AVL הגובה חסום על ידי logn ולכן הפונקציה מסיבוכיות O(logn).**

1. **Insert(I,val):**

**מקבלת: אינדקס בו נכניס את הערך val**

**מחזירה: מספר האיזונים שבוצעו.**

**תיאור פעולת הפונקציה: הפונקציה ראשית מייצרת node חדש עם הערך val וראשית במידת הצורך מעדכנת את שדות הroot,last,first של העץ על מנת ששדות אלה יתוחזקו באופן קבוע לגישה מהירה אחרי זה**.

**לאחר מכן קוראת לפונקציה insertRec() ממנה נקבל את מספר האיזונים.**

**סיבוכיות: עדכון השדות הוא O(1) והפונקציה insertRec היא בסיבוכיות של O(logn) ולכן סה״כ נקבל סיבוכיות O(logn).**

1. **insertRec(i, root, nodeToInsert,depth):**

**מקבלת: מקבלת אינדקס, מצביע לצומת שאותו נרצה להכניס ומצביע לצומת שבו אנו נמצאים כרגע.**

**מחזירה: מספר האיזונים שבוצעו.**

**תיאור פעולת הפונקציה: באמצעות רקורסיה בדומה לtreeSelect נטייל בעץ עד שנגיע למקום בו נרצה להכניס את הצומת הנוסף. תנאי העצירה בפונקציה הנ״ל כאשר הגענו לתת עץ בגודל 0/1/2 והאינדקס הוא אחד מהם ובהתאמה נקשר את הצומת החדש לצומת המתאים.**

**בכל שלב בו נרד בעץ נעדכן את שדה size של הצומת ממנו אנו באים.**

**לבסוף נקרא לפונקציה fixtree על ההורה של הצומת אותו הכנסנו ומהפונקציה נקבל counter שהוא סופר את כמות האיזונים שביצענו על העץ, אותו נחזיר עם סיום פעולת הפונקציה.**

**סיבוכיות: בכל צומת אנו מבצעים בדיקות ומעדכנים את שדה size לכן כל איטרציה היא O(1) סה״כ גובה העץ חסום על ידי logn ולכן נקבל סיבוכיות o(logn), הפונצקיה fixTree נקראת פעם אחת בסוף תהליך ההכנסה והיא בסיבוכיות זהה.**

1. **Delete(i):**

**מקבלת: אינדקס i עבורו נמחק את האיבר הi ברשימה.**

**מחזירה:מספר האיזונים שבוצעו**

**תיאור פעולת הפונקציה: באמצעות retrive נקבל מצביע לצומת אותו נרצה למחוק.**

**אם לצומת יש שני בנים, נמצא את היורש שלו נבצע על היורש bypass ונחליף בין היורש לצומת שאנו רוצים למחוק(החלפת מצביעים ולא ערכי val).**

**במידה ולצומת יש בן אחד נבצע על הצומת bypass.**

**לאחר מכן נקרא לfixTreeAfterDeletion שמעדכנת שדות וקוראת לfixTree ממנה נקבל את מספר האיזונים שבוצעו.**

**סיבוכיות: retrive – O(logn) נקראת פעם אחת. bypass(1) – נקראת פעם אחת. Fixtree – O(logn) נקראת פעם אחת. סה״כ נקבל O(logn)**

1. **listToArrayRec:**

**מחזירה: רשימה לפי אינדקסים של איברי העץ.**

**תיאור פעולת הפונקציה: באמצעות הליכת in-order על העץ אוספת את ערכי הVal של כל הצמתים ומכניסה אותם לרשימה.**

**סיבוכיות:** **כפי שלמדנו בכיתה o(n)**

1. **fixTree(node,bool):**

**מקבלת: מצביע ממנו נרצה לאזן את העץ, true אם זאת פעולת איזון לאחר הכנסה.**

**מחזירה: מספר האיזונים שבוצעו בעץ.**

**תיאור פעולת הפונקציה: מהמצביע שקיבלנו נטפס כלפי שורש העץ ונעבור על הצמתים, ראשית נשמור את ערך גובה הצומת לפני עדכון גובהו ולאחר מכן נעדכן את שדות הHeight, Bf במידה וגובה העץ לא השתנה נסיים את הפעולה.**

**במידה והשתנה הגובה נבדוק האם הabs(bf) = 2 במידה וכן נבצע גלגול במידת הצורך ונשמור counter לספירת כמות הגלגולים שביצענו עד השורש.**

**סיבוכיות: כיוון שאנו עולים בעץ ובכל צומת מבצעים עדכון שדות בלבד נראה כי אנו החסם העליון של פעולת הפונקציה הוא גובה העץ שהוא חסום על ידי logn לכן סיבוכיות הפונקציה היא O(logn).**