שם מגיש 1: Shalev Baruch
תעודת זהות 1: 213070477
שם משתמש 1: shalevbaruch
שם מגיש 2: Jonathan Nethanel
תעודת זהות 2: 214194839
שם משתמש 2: nethanel1

מסמך תיעוד לפרויקט במבני נתונים – List ADT implementation by AVL Tree

מחלקת AVLNode :
למחלקה יש את השדות size,height,left,right,parent,value , ובנוסף את השדה is_virtual שלמעשה מכיל את המידע שאומר אם ה node הנוכחי הוא virtual_node .

getLeft :
מחזירה את הבן השמאלי של ה node הנוכחי, none אם הוא וירטואלי
getRight :
מחזירה את הבן הימני של ה node הנוכחי, none אם הוא וירטואלי
getSize :
מחזירה את size של ה node הנוכחי, 0 אם הוא וירטואלי
getParent :
מחזירה את האבא של ה node הנוכחי, none אם הוא וירטואלי
getValue :
מחזירה את value של ה node הנוכחי, none אם הוא וירטואלי
getHeight :
מחזירה את הגובה של ה node הנוכחי, 1- אם הוא וירטואלי
get_BF :
מחזירה את ה- balance factor של ה node הנוכחי, 0 אם הוא וירטואלי
setLeft :
מגדירה את הבן השמאלי של ה- node הנוכחי
setRight :
מגדירה את הבן הימני של ה- node הנוכחי
setParent :
מגדירה את האבא של ה- node הנוכחי
setValue :
מגדירה את הערך של ה- node הנוכחי
setHeight :
מגדירה את הגובה של ה- node הנוכחי
isRealNode :
מחזירה אם הnode הנוכחי הוא וירטואלי או לא
get_node_index :
פונקציה זהה לחלוטין ל- select שראינו בכיתה, שכאמור פועלת ב- O(logn) ומחזירה את האיבר במקום ה – i ברשימה הממויינת (אצלנו ברשימה הרגילה כי ה – rankים הם באופן סמוי המפתחות). לרוב תקבל את השורש של העץ (לפני הקריאות הרקורסיביות).
Max :
מחזירה את האביר המקסימלי בתת העץ שמושרש ב- node הנוכחי (הולכת "ימינה עד הסוף"). פועלת ב-O(logn) .
Min :
מחזירה את האיבר המינימלי בתת העץ שמושרש ב- node הנוכחי (הולכת "שמאלה עד הסוף"). פועלת ב-O(logn) .
Successor/predecessor :
מחזירות את ה- node העוקב/קודם ל-node הנוכחי. זהות לפונקציות שראינו בכיתה וכאמור פועלות ב- O(logn) עבור קריאה בודדת. נזכיר שכמו שראינו בתרגול n קריאות successor מהמינימום (ואכן המימוש שלנו הוא fingerTree) הן O(n).
Recalculate_node_attributes :
פונקציית עזר שמעדכנת את השדות size ו- height באיטרציות השונות ב insert ו delete ב AVLTreeList.

במחלקה זאת בכל הפונקציות שלא צויין הסיבוכיות היא תמיד O(1) (חישוב קבוע/עדכון מצביעים..).

מחלקת AVLTreeList :

למחלקה 4 שדות, size שמאותחל ל-0 , root שמאותחל ל-AVLNode.virtual_node, lastItem שמאותחל ל-AVLNode.virtual_node ו - firstitem שמאותחל ל-AVLNode.virtual_node.
פונקציות המחלקה:
: Empty
החישוב נעשה ב-O(1) כמובן, זוהי פשוט בדיקה אם גודל הרשימה הוא 0.
: Retrieve
הפונקציה משתמשת אך ורק בפונקציה get_node_at של המחלקה AVLNode (פונקציה זהה לחלוטין ל select שראינו בכיתה) , שכאמור פועלת ב – O(logn) במקרה הגרוע. נשים לב כי אכן היה ניתן להציע מימוש שונה בו מתחילים מהאיבר המינימלי, עולים למעלה עד שקיבלנו תת עץ בגודל k ובו מבצעים select. באופן זה נקבל שהסיבוכיות היא O(logk). אך נשים לב ש-2 הפתרונות "משלימים" אחד את השני באופן מסויים – ל2 הפתרונות יש מקרה קצה בו הם פועלים ב- O(logn) ובמקרה בו הפתרון הראשון פועל ב- O(1) , המקרה השני פועל ב-O(logn) ולהפך. לדוגמה, בפתרון שאנחנו הצענו החזרת השורש תתבצע ב-O(1), אבל בפיתרון השני נצטרך לעלות מהמינימום עד לשורש, כלומר O(logn). באופן סימטרי עבור המקרה בו נחפש את האיבר המינימלי נקבל בדיוק את ההפך. לסיכום אין הכרעה ברורה בין הפתרונות בנוגע לשאלה איזה פתרון עדיף, וממילא במקרה הגרוע הסיבוכיות זהה.
: left_rotate/right_rotate
2 פונקציות עזר של המחלקה שמשמשות לתיקונים של הפרות איזון בעץ, 2 פעולות אלה מתבצעות ב- O(1) כפי שראינו בכיתה כי בעצם מתבצע שינוי של כמות קבועה של מצביעים ובדיקה של כמות קבועה של תנאים.
:Insert
ראשית מתבצעות כמה בדיקות של מקרי קצה (הכנסה ראשונה/הכנסה לסוף רשימה/הכנסה לתחילת רשימה), ובנוסף בודקים האם יש לעדכן את המצביע למקום הראשון/אחרון. לאחר מכן מפעילים את get_node_at באינדקס הדרוש כדי למצוא את המקום בו התבקשנו להכניס ערך חדש (זה כאמור קורה ב-O(logn). לאחר מכן מחשבים את ה - predecessor של הערך הנוכחי במקום הדרוש (גם כן ב – O(logn) כדי להפוך את הערך הנוכחי לבן הימני שלו על מנת לפנות מקום לערך החדש. לאחר מכן עולים במעלה העץ כדי לחפש הפרות איזון בעץ, נשים לב כי בניגוד למימוש בכיתה ממשיכים למעלה גם אחרי התיקון הראשון (אם קיים) כדי לעדכן את השדות size ו- height. יכולה להיות מקסימום הפרת איזון אחת ובה מבצעים תיקון בעזרת גלגולים ב-O(1), בשאר הריצות פשוט מעדכנים את הערכים גם כן ב-O(1) (בעזרת הבנים). בנוסף יש log n איטרציות (כי זהו עץ AVL), ולכן סה"כ נקבל O(logn) .
: Replace
פונקציית עזר שתשמש את delete, יש שם חישוב קבוע של שינוי מצביעים, כאמור O(1). הערה – כארגומנטים הפונקציה מקבלת x:AVLNode כשהמטרה בצורת כתיבה זאת היא שיהיה ברור על איזה אובייקט עושים פעולות בפונקציה, ושכבר ב"קומפילציה ראשונית" pycharm "יזהה" את האובייקט ולכן יידע איזה פונקציות/שדות קיימים לאובייקט.
:Delete
כמו ב- insert גם כאן נפתח בבדיקה של מקרה קצה, שוב נבדוק גם אם יש לשנות את המצביעים לאיבר הראשון/אחרון ואז נחשב את ה - node אותו נרצה למחוק ב-O(logn) . לאחר מכן נבצע מחיקה רגילה של BST שקורית ב- O(logn) במקרה בו צריך למשש את ה – successor . אחרי זה בדומה ל – insert נעלה במעלה העץ כדי לחפש הפרות איזון בעץ. גם כאן נמשיך עד לשורש, ובכל איטרציה יש O(1) עבודה גם במקרים בהם צריך לתקן עם גלגולים. סה"כ O(logn) . נשים לב כי כאן מימשנו פונקציית עזר שתבצע את הלולאה על מנת לפשט את הקוד.

: Fix_Up_Delete
מבצעת את הלולאה ב-delete , כאמור ב- O(logn) .
: First/Last
מחזירות את האיבר הראשון/אחרון בהתאמה, לשם כך נעזר במצביעים firstItem ו – lastItem שמחזיקים תמיד מצביע לאיבר הראשון והאחרון ברשימה בהתאמה, עדכון 2 מצביעים אלה נעשה ב-O(1) בזמן insert וdelete ולכן לא משפיע על זמני הריצה. כאמור 2 הפעולות לוקחות O(1).
: ListToArray
מאותחל מערך באורך הרשימה עם ערך None בכל האיברים ב- O(n), לאחר מכן נשמור במשתנה שקרוי minimum את המינימום (יש לנו שדה שמחזיק את המינימום) ב O(1), ואז עוברים בלולאה (n איטרציות), ובכל פעם מבצעים Minimum=minimum.successor() ומכניסים למקום המתאים במערך את minimum. לפי הנאמר בחלק של הניתוח של successor נקבל שהסיבוכיות של קריאה לsuccessor כ n פעמים החל מהמינימום עולה O(n) ומאחר שהכנסה של כל איבר למערך לוקחת זמן קבוע נקבל כי סך הכל סיבוכיות זמן הריצה של הפונקציה היא O(n) .

: Length
מחזירה את השדה size ב- O(1) .
: CloneBinaryTree
נשתמש בפונקציה זאת בתוך הפונקציות Permutation ו-Sort על מנת שנוכל פשוט לעבור inorder על העץ המשובט ולשנות את הערכים כרצוננו. פונקציה זאת גם למעשה מבצעת גרסה של סיור inorder ובונה מלמטה למעלה את העץ המשובט תוך כדי עדכון המצביעים והשדות המתאימים. הפונקציה מחזירה את ה-root של הרשימה המשובטת. כאמור הסיבוכיות היא O(n) .
: SetTree
בפונקציה זאת נשתמש כדי להגדיר עץ מתוך השורש שקיבלנו מהפונקציה הקודמת. מאותחל כאן עץ שהשורש שלו הוא התא שהתקבל מהפונקציה הקודמת, ובנוסף מעודכנים השדות המתאימים. נשים לב כי הסיבוכיות היא O(logn) כי יש לחשב את firstItem ו lastItem של השורש כי אין גישה בשלב זה לשדות של העץ המקורי.

: Sort
משתמשים ב- ListToArray ב – O(n) כדי לקבל את המערך המתאים לרשימה ולאחר מכן מבצעים עליו mergeSort בעזרת פונקציית העזר SORT ב- O(nlogn) כפי שלמדנו במבוא מורחב. אחרי זה נשבט את העץ בעזרת פונקציות העזר ב- O(n).
משם ההמשך זהה לגמרי למימוש של ListToArray רק שבמקום להכניס למערך את הערך של התא בכל שלב פשוט משנים את הערך של התא (בעץ המשובט הפעם כמובן) לערך המתאים במערך הממוין (אליו יש גישה ב- O(1)). סה"כ O(nlogn) .
: Permutation
זהה לחלוטין ל- sort רק שבמקום למיין את המערך שהתקבל מ-ListToArray מבצעים עליו את פונקצית העזר shuffler ב- O(n) ומשם ההמשך זהה. סה"כ O(n) .

: Search
שוב נקבל את המינימום (האיבר הראשון) ב- O(1) בעזרת המצביע שיש לנו . לאחר מכן שוב מבצעים קריאות ל- successor בלולאה בדומה לפונקציות הקודמות, והפעם בכל שלב בודקים האם מצאנו את הערך הדרוש ב-O(1) . במקרה הגרוע בו האיבר הדרוש נמצא במקום האחרון או לא נמצא בכלל הסיבוכיות היא O(n) .
: getRoot
מחזיר את השדה root ב- O(1) .
: SORT
פונקציית עזר שמממשת את mergeSort כפי שלמדנו במבוא מורחב שהסיבוכיות היא O(nlogn)
: Shuffler
פונקציית עזר שמבצעת פרמוטציה על רשימה. עוברת בלולאה שגודלה כגודל הרשימה (כלומר , n) ומחליפה בין האיברים בתוך הלולאה ב-O(1) . סה"כ פועלת ב- O(n) .

: Concat

עד הקריאות לפונקציה $joinSTB$ או $joinBTS$ יש בדיקות והשמות שלוקחות זמן קבוע, ובנוסף פעולת הכנסה אחת שהוכחנו שלוקחת $O(logn)$. לכן נותר להוכיח שהפונקציות $joinSTB$, $joinBTS$ עולות $O(logn)$ ונסיים.

הניתוח של 2 הפונקציות הללו דומה- כל פונקציה מתחילה בהשמות שלוקחות זמן קבוע, ולאחר מכן או שיורדים שמאלה בlst עד שמתקיים $temp.height \leq self.height$, או שיורדים ימינה ב$self$ עד שמתקיים $temp.height \leq lst.height$. מאחר שהגובה של שני העצים חסום על ידי $logn$, הירידה הזו בעצים זו עולה לכל היותר $O(logn)$. לאחר מכן בכל פונקציה יש בדיקות של תנאים וחיבורים של פוינטרים שלוקחות זמן קבוע, ולבסוף יש קריאה ל Fix Up Delete שהוכחנו שלוקחת $O(logn)$. לכן קיבלנו ש $joinSTB$, $joinBTS$ עולות $O(logn)$ ובסך הכל נקבל מכך ש $concat$ עולה $O(logn)$.

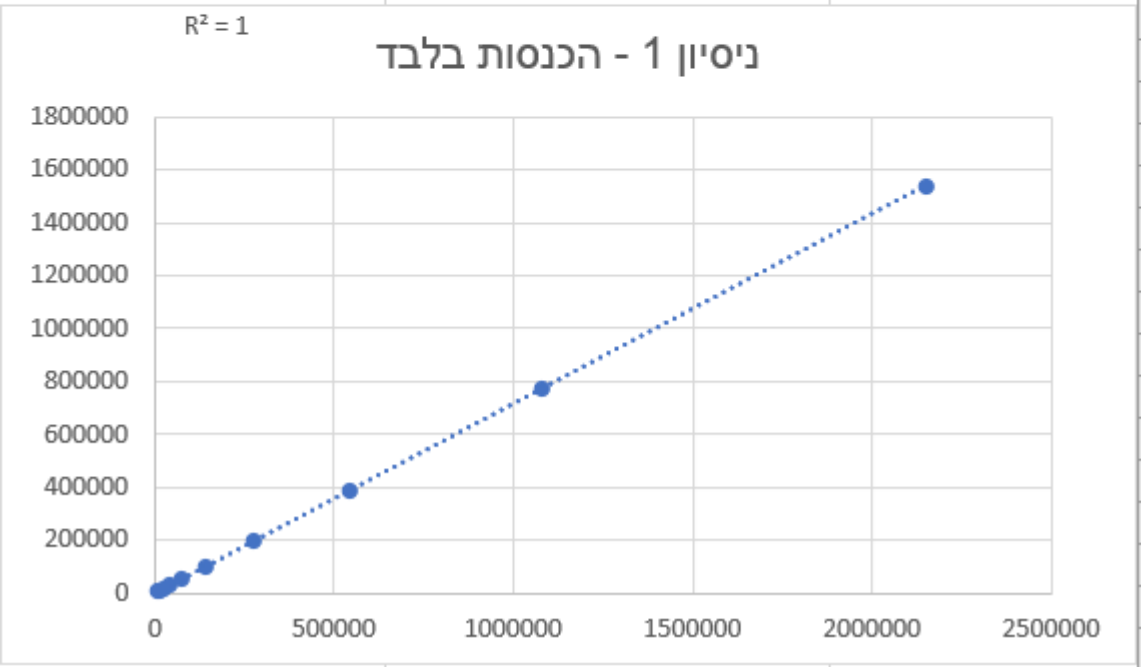
ניתוח חלק תיאורטי

שאלה 1

.1

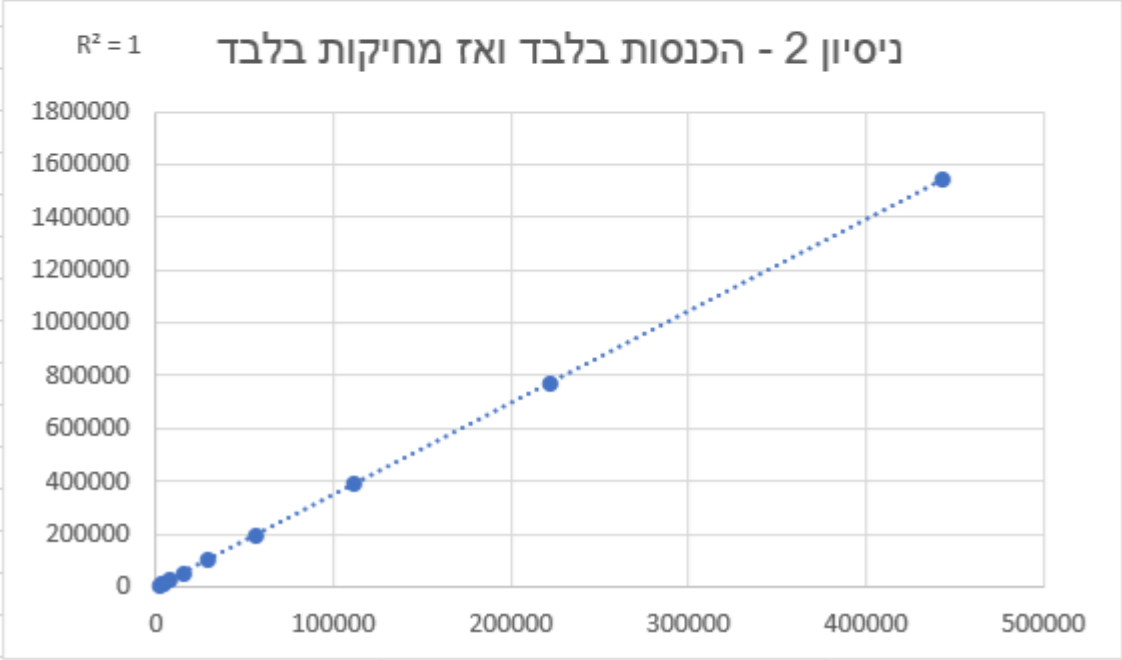
תוצאות ניסוי 1:

ניסיון 1 - הכנסות	גודל הקלט
2065	3000
6342	6000
14702	12000
31379	24000
65011	48000
132189	96000
265786	192000
533201	384000
1070583	768000
2142219	1536000



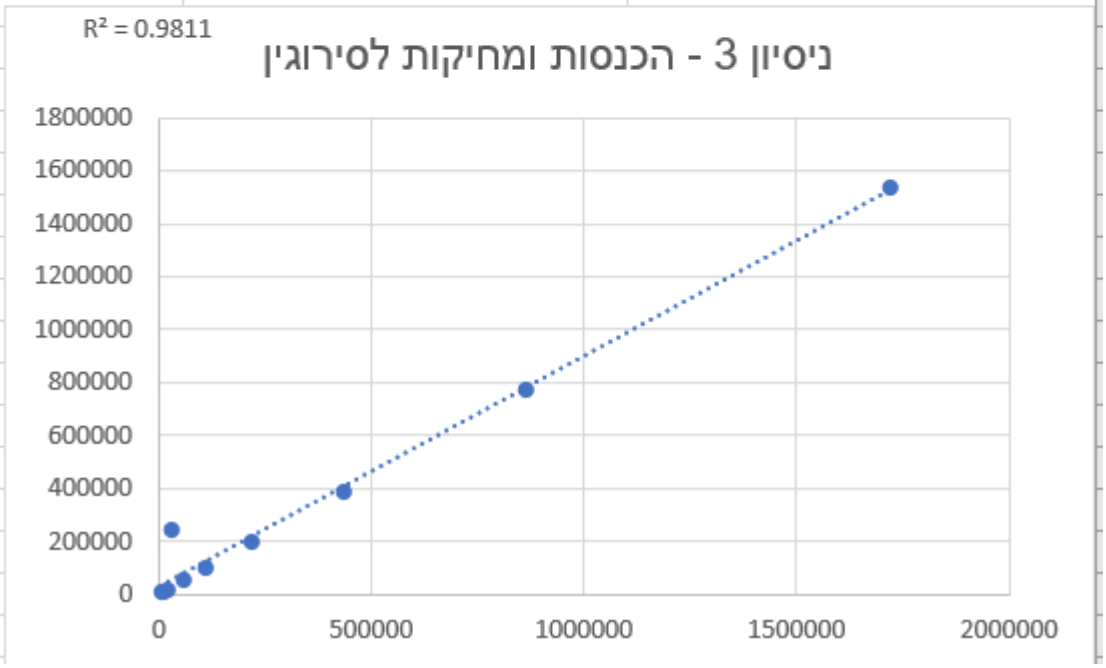
תוצאות ניסוי 2:

ניסיון 2 - מחיקות	גודל הקלט
455	3000
1400	6000
3104	12000
6607	24000
13488	48000
27314	96000
54794	192000
110292	384000
220544	768000
441174	1536000



תוצאות ניסוי 3:

ניסיון 3 - הכנסות ומחיקות לסירוגין	גודל הקלט
1695	3000
5055	6000
11576	12000
25333	240000
51860	48000
105036	96000
212249	192000
426627	384000
854903	768000
1712059	1536000



2. לפי התוצאות של הניסוי, יצא שכל אחד מהניסויים תואם לביטוי האסימפטוטי $O(n)$. הסקנו זאת מפני שבכל פעם שהקלט גדל פי 2, גם מספר פעולות האיזון שנדרשו כדי לתקן את העץ גדלו (בערך)פי 2, ובעיקר מכך שקו המגמה הינו ליניארי.

שאלה 2

טבלה ראשונה - כניסות בהתחלה:

מספר סידורי i	עץ Avl	מערך	רשימה מקושרת
1	0.1100173	0	0
2	0.216473579	0	0
3	0.476779222	0.015712023	0.015771389
4	1.023643494	0.078125954	0.01578827
5	2.163296223	0.297838449	0.024259329
6	4.616360426	1.182284355	0.089106083
7	9.978800774	4.672535896	0.189006805
8	21.59283876	20.8044436	0.445395231
9	48.96175718	88.74407148	0.928405523
10	101.3297954	430.2629547	1.819747925

טבלה שנייה - הכנסות אקראיות:

מספר סידורי i	עץ Avl	מערך	רשימה מקושרת
1	0.121999979	0	0.078318834
2	0.280258417	0.015620947	0.281957626
3	0.59967494	0.005762815	1.284241438
4	1.318270206	0.062711477	6.657070398
5	2.855836868	0.21989274	30.5931673
6	6.253283024	0.730882168	185.6476743
7	14.64121366	2.512094259	2171.476884
8	29.36092138	10.4843235	17371.81507
9	66.73848867	43.52143931	138974.5206
10	139.0449276	191.6644402	1111796.165

טבלה שלישית - הכנסות בסוף:

מספר סידורי i	עץ Avl	מערך	רשימה מקושרת
1	0.1100173	0	0
2	0.216473579	0	0
3	0.476779222	0.015712023	0.015771389
4	1.023643494	0.078125954	0.01578827
5	2.163296223	0.297838449	0.024259329
6	4.616360426	1.182284355	0.089106083
7	9.978800774	4.672535896	0.189006805
8	21.59283876	20.8044436	0.445395231
9	48.96175718	88.74407148	0.928405523
10	101.3297954	430.2629547	1.819747925

התוצאות תואמות את הציפיות. הסבר:

עבור רשימה מקושרת(לא דו כיוונית ויש מצביע נוסף לסוף הרשימה) :

נשים לב כי התוצאות של ההכנסות בסוף ובהתחלה הן שלוקח מעט זמן, וזה הגיוני כי על מנת לבצע את ההכנסה יש בעצם לבצע שינוי של פוינטרים בודדים. כמו כן, התוצאות של ההכנסות האקראיות הן שלוקח זמן רב יחסית, וזאת מפני שבממוצע המיקום שאליו נרצה להכניס יהיה קרוב למרכז הרשימה, ובכל פעם כזאת נצטרך לעבור איבר איבר החל מראש הרשימה כדי להגיע לצומת שאחריו מכניסים את האיבר המבוקש. מעבר זה לוקח זמן רב.

עבור מערך:

התוצאות של ההכנסות בהתחלה הן שלוקח זמן רב, וזה הגיוני כי בעצם בכל פעם יש צורך להזיז את כל האיברים שהיו כבר במערך קדימה. כמו כן, מאותה סיבה גם כשמכניסים בצורה אקראית הדבר לוקח זמן רב (בממוצע נבחר מיקום שקרוב למרכז הרשימה ואז יש להזיז בערך כמחצית מהאיברים לוקחת זמן קבוע ואין צורך להזיז אף איבר קדימה.

עבור עץ AVL:

ראשית, ניתן לראות כי התוצאות בכל המקרים דומות, וזה הגיוני כי בכל פעם שאנו מכניסים איבר, ללא תלות במיקום אנו נצטרך לעדכן את השדה $size$ (ולעיתים גם $height$), החל מהעלה שאנו נמצאים בו ועד לשורש. כמו כן,מאחר שהעץ הוא עץ AVL, הוא שומר על איזון מסויים, כך שלא ייתכן שירידה לעלה אחד היא הרבה יותר מהירה מירידה לעלה אחר.