תיעוד הקוד

AVLNode

אם לא ניתן ערך value, אז הצומת מאותחל להיות עלה וירטואלי, בו הערך והמצביעים לבנים ולאב הם None, גובה ‎-1‎ וגודל 0. אחרת, המצביעים לבנים יצביעו על עלים וירטואליים חדשים, הגובה 0 והגודל 1.

- *successor*() - מחזיר את האיבר העוקב לאיבר הנוכחי, כלומר האיבר שדרגתו בעץ גדולה ב-1.
  אם לאיבר יש בן ימני, נלך לבן הימני ומשם נרד שמאלה עד שנגיע לאיבר ללא בן שמאלי, נחזיר את האיבר הזה.
  אם לאיבר אין בן ימני, נעלה שמאלה עד לעלייה הראשונה ימינה, ונחזיר את הצומת הראשון שעלינו אליו ימינה.
  בשני המקרים נחפש מסלול בין צומת ובין אב או צאצא קדמון. אורך המסלול הגדול ביותר הוא בין השורש לעלה העמוק ביותר, כלומר גובה העץ. אם כן, זמן הריצה חסום אסימפתוטית על ידי גובה העץ ולכן $O(logn)$.

- *predecessor*() - מחזיר את האיבר הקודם לאיבר הנוכחי, כלומר האיבר שדרגתו בעץ קטנה ב-1.
  נבצע הליך סימטרי למציאת ה- successor אם לאיבר יש בן שמאלי, נלך לבן השמאלי ומשם נרד ימינה עד שנגיע לאיבר ללא בן ימני, נחזיר את האיבר הזה.
  אם לאיבר אין בן שמאלי, נעלה ימינה עד לעלייה הראשונה שמאלה, ונחזיר את הצומת הראשון שעלינו אליו שמאלה.
  ניתוח הסיבוכיות זהה ל-successor, ולכן זמן הריצה הוא $O(logn)$.

- *listToArrayRec(arr)*- הפונקציה מקבלת רשימה ומכניסה לתוכה את ערכי העץ לפי סדר הדרגות. נבצע סיור in-order בעץ בעזרת רקורסיה על שני הבנים של כל צומת, החל מהשורש. בכל צומת נכניס את ערך הצומת למערך arr, כלומר (1) עבודה. לכן בסך הכל סיבוכיות הפונקציה היא $O(n)$.

- *searchRec(val, i)*- נבצע סיור in-order בעץ בעזרת רקורסיה על שני הבנים של כל צומת, החל מהשורש. בכל צומת נבדוק האם val נמצא, כלומר (1) עבודה. לכן בסך הכל סיבוכיות הפונקציה היא $O(n)$.

- *join(xVal,lst)*- הפונקציה מקבלת שני תתי עצים וערך, כך שכל ערכי תת עץ אחד גדולים מהערך val שגדול מכל ערכי תת העץ השני. הפונקציה ממזגת את שני תתי העצים לעץ מאוזן מטיפוס AVLTreeList. הפונקציה מתבססת על הפעולה join במחלקה AVLTreeList. ניצור שני אובייקטים מסוג AVLTreeList ונגדיר את השורשים שלהם להיות שני ה-AVLNode שברצוננו למזג. כעת נקרא לפעולה AVLTreeList.join שפעולת בזמן $O(logn)$.

AVLTree

שדות נוספים:

root -נאתחל את השורש ברשימה ריקה להיות עלה וירטואלי.

Size – שומר את גודל תת העץ של הצומת (כולל הצומת)

firstItem - מצביע לאיבר הראשון ברשימה - הצומת בעל הדרגה המקסימלית בעץ.

lastItem- מצביע לאיבר האחרון ברשימה - הצומת בעל הדרגה המינימלית בעץ.

מתודות:

- *empty*() – בודק האם הרשימה ריקה. כלומר, בודק באמצעות *isRealNode* האם השורש הוא וירטואלי או לא. *isRealNode* פועלת בזמן קבוע, ולכן גם *empty()*. -$O(1)$.

- *Select(i)* – נחפש ומחזיר את הצומת **שדרגתו** i. נממש כפי שראינו בהרצאה ונעבור במסלול מהשורש ועד לצומת המבוקש, לפי גודל הבן השמאלי. הפונקציה עוברת על מסלול בין השורש לצומת כלשהו. אורך המסלול הגדול ביותר הוא בין השורש לעלה העמוק ביותר, כלומר גובה העץ. אם כן, זמן הריצה חסום אסימפתוטית על ידי גובה העץ ולכן $O(logn)$.

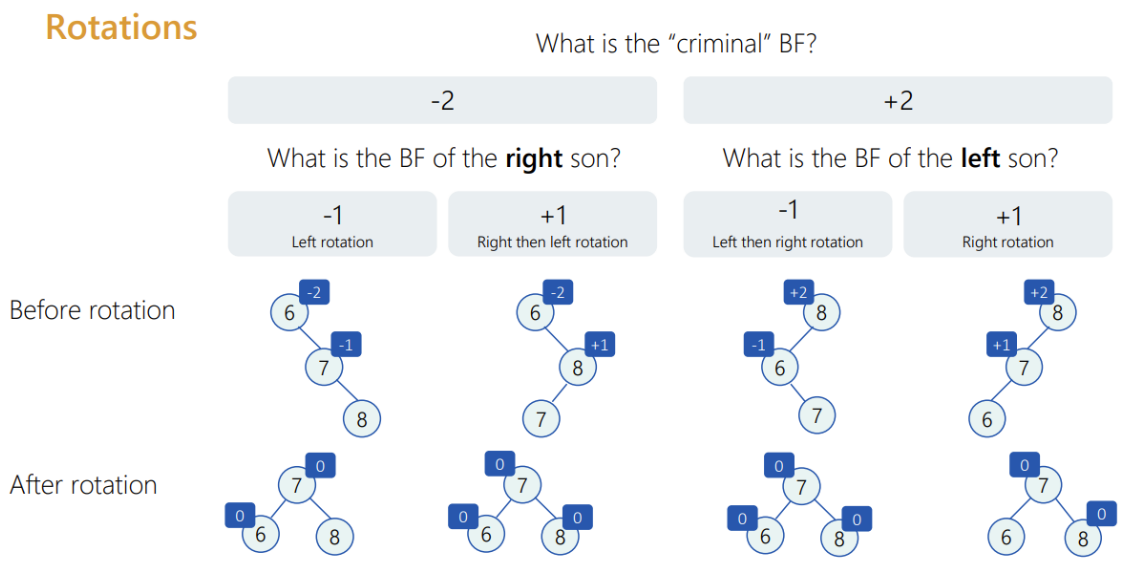
- *retrieveNode(i)* – האינדקסים של רשימה מתחילים ב-0, בעוד הדרגות של הצמתים בעץ מתחילות ב-1. לכן, כדי לקבל את הצומת שמייצג את **האיבר ברשימה במיקום i** נקרא ל- *Select(i+1)*. הסיבוכיות זהה ל- *Select* והיא $O(logn)$.

- *retrieve*(i) – אם i נמצא ברשימה נחזיר את ערך האיבר *retrieveNode(i)*. אחרת נחזיר None. הסיבוכיות זהה ל- *retrieveNode* והיא $O(logn)$.

- *updateFirstLast()* – הפונקציה מעדכנת את השדות *firstItem, lastItem*. אם הרשימה ריקה, נעדכן אותן ל-*None*. אחרת, נמצא את האיברים הראשון והאחרון ברשימה באמצעות *retrieveNode* ונצביע עליהם בשדות המתאימים. הסיבוכיות זהה ל- *retrieveNode* והיא $O(logn)$.
  כלומר, כל פעולה שנממש שפועלת בזמן $O(logn)$ או יותר, יכולה לתחזק את השדות הללו.

- *rotateRight(y)* – הפונקציה מבצעת גלגול אחד ימינה כפי שנלמד בהרצאה. בנוסף, הפונקציה מחזירה 1. נשתמש בערך שמוחזר כדי לספור את מספר פעולות האיזון מסוג גלגול. $O(1)$.
- *rotateLeft(x)* – פונקציה סימטרית ל- *rotateRight*.

- *insertRebalance(node)* – עוברת במסלול החל מצומת ועד לשורש. בכל צומת הפונקציה תבדוק את ה-balance factor של הצומת ותבצע על הצומת גלגולים בהתאמה, כך שתת העץ של הצומת מאוזן לפי השיטה:



בנוסף לאיזון לאחר הכנסה של איבר, נשתמש בפונקציה כדי לאזן עצים לאחר מיזוג. בשל כך, נבדוק גם מקרה בו הצומת הנוכחי לא מאוזן, אך שני הבנים שלו מאוזנים. אם לא התבצע גלגול, נעדכן את גובה הצומת, ונספור את עדכון הגובה בתור פעולת איזון. לאחר מכן נעדכן את גודל הצומת.

בעץ AVL רגיל ניתן היה לעצור את הפעולה כאשר מגיעים לעץ שגובהו לא השתנה, אולם כדי לשמר את שדה הגודל, נצטרך להמשיך עד השורש. המסלול האפשרי הארוך ביותר הוא מהעלה העמוק ביותר ועד לשורש, $(logn)$, ובכל צומת נבצע פעולות בזמן קבוע, לכן סיבוכיות הפעולה היא $(logn)$.

- $insert(i, val)$ – אם הרשימה ריקה, האיבר שנוסיף הוא השורש, לכן נאתחל מחדש את השורש בתור עלה עם ערך val. $O(1)$

אם נרצה להוסיף איבר בסוף הרשימה, נגש לאיבר האחרון בעזרת השדה lastItem, נכניס את האיבר החדש בתור בנו הימני ונאזן את העץ באמצעות *insertRebalance*. בנוסף נעדכן אותו בתור lastItem.

אם מוסיפים איבר במקום הראשון, נעדכן בהתאם את שדה firstItem.
אם נרצה להוסיף איבר במקום אחר ברשימה, תחילה נבדוק אם נמצא את האיבר שנמצא כעת במקום ה-i באמצעות *retrieveNode* בסיבוכיות לוגרימית. לאחר מכן אם לאותו איבר אין בן שמאלי, נכניס שם את האיבר החדש. אחרת נמצא את האיבר הקודם לו בסיבוכיות לוגריתמית. האיבר הקודם במצב זה הוא בהכרח עלה.נכניס את האיבר החדש בתור הבן הימני של האיבר הקודם.
לבסוף נעדכן את העץ באמצעות *insertRebalance*.

הפעולות *insertRebalance*, *retrieveNode*, *predecessor* פעולות כולן בסיביוכיות $O(logn)$ בעוד שאר הפעולות ב- $O(1)$, לכן התוכנית כולה רצה בסיבוכיות $O(logn)$.

- *deleteRebalance(node)* – הפעולה זהה ברובה ל-*insertRebalance* . ההבדל העיקרי בא לידי ביטוי בכך שכעת באיזון לאחר מחיקה, נספור גם שינויים בגובה של עצים שגולגלו וגם השתנה גובהם.
בנוסף, אין צורך במקרה המיוחד עבור *join*. הבדלים אלו הם מסיבוכיות *O(1)* ולכן במן הריצה של הפונקציה הוא כשל *insertRebalance()*, *O(logn)*.

- *delete*(i) – תחילה נבדוק אם i לא חורג מגבולות המערך, ונחזיר 1- אחרת.

כעת נאחזר את האיבר במקום ה-i באמצעות *retrieveNode*. אם יש לו שני בנים, נחליף אותו בעוקב שלו, ונמשיך עד שנגיע לצומת עם פחות משני בנים. אם לצומת יש שני בנים בפרט יש לו בן ימני. אם כך העוקב שלו נמצא בתת העת הימני. במקרה הגרוע ביותר, נצטרך לעעבור במסלול מהשורש עד לעלה. אורך מסלול זה הוא $O(logn)$ ולכן זהו הזמן למציאת העוקבים.

כאשר נגיע לצומת עם פחות מ-2 עלים נבדוק אם הוא עלה, אם כן נמחק אותו.
אחרת, נבדוק אם האיבר הוא בן יחיד. אם כן, נוכל ״לדלג״ על האיבר ולקשר בין בנו לאביו.

לבסוף נאזן את העץ באמצעות *deleteRebalance* ונעדכן את המצביעים לאיבר הראשון והאחרון באמצעות *updateFirstLast*. שתי פעולות אלה רצות גם הן ב- $O(logn)$. כל השלבים שביצענו פעלו בזמן לוגריתמי או קבוע, ולכן סיבוכיות הפונקציה היא $O(logn)$.

- *first()* – אם הרשימה ריקה, נחזיר *None*, אחרת נחזיר את הערך של *firstItem*. $O(1)$.
- *last()* – אם הרשימה ריקה, נחזיר *None*, אחרת נחזיר את הערך של *lasstItem*. $O(1)$.

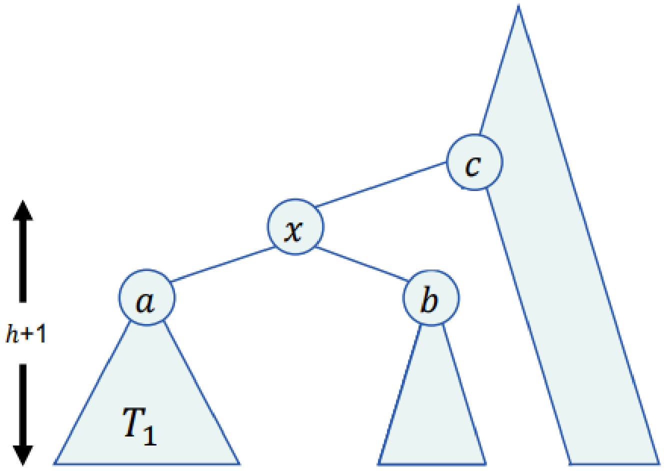
- *listToArray()* – הפונקציה יוצרת רשימה ומעבירה אותה לפונקציה – *listToArrayRec* במחלקה *AVLNode*, ולכן רצה באותו זמן $O(n)$.

- *length()* – גודל הרשימה הוא מספר האיברים בה, כלומר מספר הצמתים בעץ. לכן נחזיר את שדה *size* של השורש. $O(1)$.

- *getLeftSubTree(h)* – הפונקציה מתחילה מהשורש ועוברת על הבנים השמאליים עד שמגיעים לצומת שגובהו קטן או שווה ל-h. . אורך המסלול הגדול ביותר הוא בין השורש לעלה העמוק ביותר, כלומר גובה העץ. אם כן, זמן הריצה חסום אסימפתוטית על ידי גובה העץ ולכן $O(logn)$.
- *getRighttSubTree* – באופן דומה עבור בנים ימניים.

join(xVal,lst) – בה״כ lst היא הרשימה הגבוהה מבין השתיים. נחפש בעזרת *getLeftSubTree* את הצומת השמאלי הראשון בגובה של הרשימה השנייה. כעת נחבר את הבן השמאלי של הצומת ואת שורש הרשימה lst בתור בניו של x. נשים את x בתור בנו השמאלי של הצומת. נאזן מחדש את העץ הממוזג החל מהצומת. אורך המסלול מהצומת ועד לשורש הוא $O(logn)$ לכן האיזון לוקח $O(logn)$ זמן. גם *getLeftSubTree* עולה $O(logn)$. לכן סיבוכיות הפונקציה היא $O(logn)$. למעשה, אם נסמן את גבהי שתי הרשימות ב- *m,k* נשים לב ששתי הפעולות רצות רק על המסלול משורש העץ הגבוה מבין השניים ועד לצומת שגובהו כגובה העץ השני. כלומר אורך המסלול הוא בעצם $|m-k|$. אם כך, זמן הריצה הוא לוגריתמי שהפרש הגבהים של הרשימה.

-



- $concat(lst)$ - אם אחת הרשימות ריקות, נחזיר את השורש של הרשימה השנייה.
אחרת, "נוציא" מהרשימה הראשונה את האיבר האחרון שלה באמצעות *retrieve* ו-*delete* בסיבוכיות לוגריתמית. לאחר מכן נאחד את שתי הרשימות והערך שהופרד באמצעות *join* שבמחלקה *AVLNode*. *join* רצה גם כן בזמן לוגריתמי ולכן סיבוכיות הפונקציה היא $O(logn)$.

- $split(i)$ - הפונקציה מחפשת את האיבר ה-i בזמן לוגריתמי, ומפעילה את *splitLoop* עליו ועל שני ילדיו.

- $splitLoop(x, leftTree, rightTree)$ - בכל פעם נבדוק אם הצומת הנוכחי הוא בן ימני או שמאלי:

אם בן שמאלי, נמזג את *rightTree* עם האב של הצומת הנוכחי ועם תת העץ השמאלי שלו.

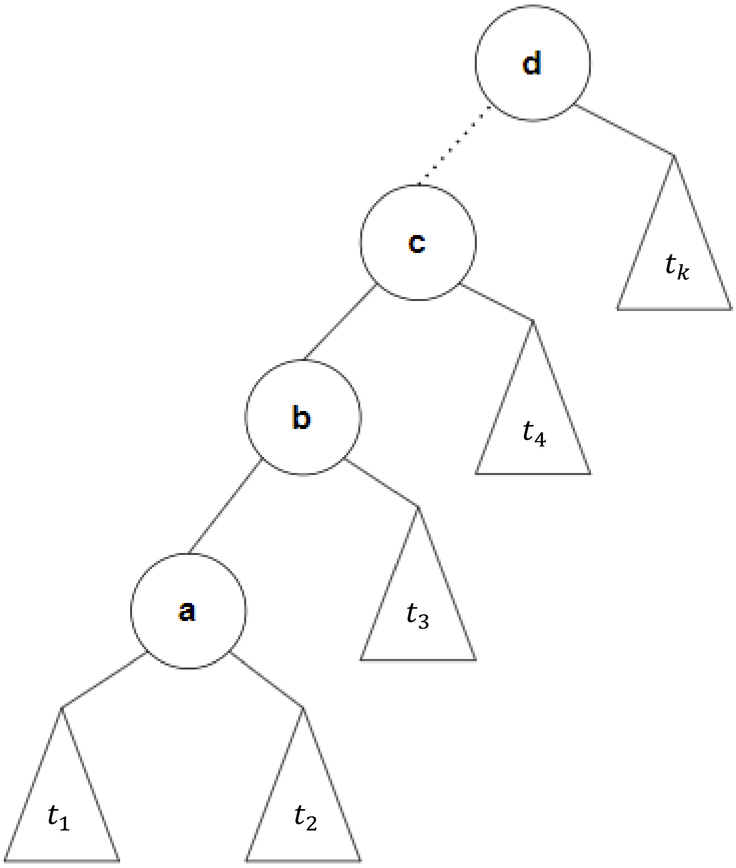
אם בן ימני, נמזג את תת העץ השמאלי של האב עם האב ו-*leftTree*. נמשיך כך עד השורש.

בניתוח נתייחס למקרה בו כל הצמתים במסלול מהאיבר ה-i עד השורש הם בנים ימניים, כלומר נבצע את כל $O(logn)$ רק על תתי העץ השמאליים של כל צומת במסלול.

במבט ראשוני נדמה שהפונקציה רצה בזמן של $O(log^2n)$, שכן נרוץ על מסלול באורך לוגריתמי, ובכל מסלול בו נבצע עבודה של $O(logn)$.

נזכר בכך שסיבוכיות מיזוג שני עצים שגביהם h_1, h_2 כאשר $h_1 \leq h_2$ היא $O(h_2 - h_1)$. בהרצאה ראינו את הלמה הבאה: קיים $c>0$ כך שלכל i מתקיים $height(join(t_1, ..., t_i)) \leq height(t_1)+c$.

כמו כן, בשל תכונת האיזון של עץ *AVL*, גובה כל בן של אב של צומת גדול או שווה לגובה הבן של הצומת. כלומר, בשרטוט הבא מתקיים $h_2 \leq h_3 \leq h_4 \leq ... \leq h_k$. זהו גם המקרה הגרוע ביותר, בו נבצע את כל $O(logn)$ המיזוגים על אותו צד.



כלומר, אם נבצע *O*(*logn)* מיזוגים, נקבל שהסיבוכיות היא טור טלסקופי שמצטמצם לסכום הגבהים של העץ הגבוה והנמוך ביותר שנמזג. שני הגבהים הללו הם *O(log n)*, לכן זוהי הסיבוכיות של כל הפונקציה.

$$O\left(\sum_{i=2}^{k} |height(t_i) - height(join(t_1, \dots, t_{i-1}))| = 1\right) = O(log\ n)$$

- *search(i)* – הפונקציה מחזירה 1- אם הרשימה ריקה, אחרת קוראת לפעולה *searchRec* במחלקה AVLNode שפועלת בזמן *O*(*n)*.

שאלה 1:

.1

מספר סידורי i	ניסוי 1- הכנסות	ניסוי 2 - מחיקות	ניסוי 3 – הכנסות ומחיקות לסירוגין
1	4237	2574	1746
2	8387	5096	3189
3	16713	10149	6507
4	33600	20900	13211
5	66844	41622	26314
6	133561	83310	52478
7	267575	166702	105978
8	535975	333128	210196
9	1071213	667164	421318
10	2141780	1331277	844818

.2 שלושת העמודות מתאימות לביטוי $O(n)$.

שאלה 2:

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	1.7893	6	1.7897	11
2	1.7455	10	1.7538	12
3	1.4891	5	1.6586	13
4	1.5266	10	1.754	15
5	1.6207	9	1.7551	16
6	1.5173	8	1.6794	17
7	1.6232	9	1.7443	18
8	1.6726	7	1.7225	20
9	1.7378	7	1.7176	21
10	1.5052	10	1.7367	22

ניתוח סיבוכיות ל-join ממוצע – מספר ה- join-ים המתבצעים בפעולת split הוא העומק של הצומת שעושים עליו split. הסיבוכיות של split היא $O(log\ n)$.
יהי צומת x, העומק שלו (d_x) שווה ל- $h - h_x$ כאשר h הוא הגובה של העץ ו- h_x הוא הגובה של x. ראינו שגובה ממוצע בעץ AVL הוא $O(1)$ לכן עומק ממוצע של צומת הוא $\Omega(h) = \Omega(log\ n)$.
אז הסיבוכיות של join ממוצע ב- split שנעשה על צומת x היא $O(\frac{log\ n}{d_x})$ לכן בתוחלת הסיבוכיות היא $O(1)$.

בפרט העומק של הצומת המקסימלי בתת העץ השמאלי הוא תמיד $\Omega(log\ n)$ בעץ AVL, לכן הסיבוכיות של join ממוצע ב- split שנעשה על הצומת המקסימלי בתת העץ השמאלי היא $O(1)$. אז הניתוח התיאורטי מסתדר עם תוצאות הניסוי.

ניתוח סיבוכיות ל-join מקסימלי – הצומת המקסימלי בתת העץ השמאלי הוא בן ימני וגם כל אב קדמון שלו חוץ מהבן של השורש הוא בן ימני. לכן כשמגיעים לאב הקדמון שלו שהוא הבן של השורש הערך rightTree נשאר על הערך המקורי שלו, שהוא הבן הימני של הצומת שהתחלנו ממנו, כלומר צומת וירטואלי. אז בפעולת ה-join האחרונה נאחד בין העץ המושרש בבן הימני של השורש לבין העץ המושרש בצומת וירטואלי. הסיבוכיות של פעולת ה-join הזו תהיה $\Theta(\log n)$.
כך גם יצאה הסיבוכיות של join מקסימלי לפי הניסוי.

שאלה 3:

מספר פעולות האיזון בממוצע / מספר סידורי i	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	1.975	498.5	0.494	0.494	1.912	1.842
2	1.9865	998.5	0.497	0.497	1.844	1.746
3	1.99	1498.5	0.4977	0.4977	1.87833	1.82133
4	1.99275	1998.5	0.4985	0.4985	1.869	1.84325
5	1.994	2498.5	0.499	0.499	1.8712	1.7886
6	1.99466	2998.5	0.49883	0.49883	1.8215	1.74
7	1.99542	3498.5	0.499	0.499	1.85914	1.7755
8	1.99612	3998.5	0.49925	0.49925	1.83312	1.7261
9	1.99644	4498.5	0.49944	0.49944	1.868	1.7878
10	1.9968	4998.5	0.4995	0.4995	1.8669	1.8549

עומק הצומת המוכנס בממוצע / מספר סידורי i	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	7.987	499.5	7.987	7.987	8.069	11.125
2	8.982	999.5	8.982	8.982	9.2095	11.3975
3	9.639	1499.5	9.639	9.639	9.73367	13.6947
4	9.97925	1999.5	9.97925	9.97925	10.1867	14.228
5	10.3644	2499.5	10.3644	10.3644	10.705	13.4948
6	10.637	2999.5	10.637	10.637	10.7928	13.6568
7	10.8317	3499.5	10.8317	10.8317	11.082	14.836
8	10.9778	3999.5	10.9778	10.9778	11.2501	16.2803
9	11.1812	4499.5	11.1812	11.1812	11.3588	14.7955
10	11.3631	4999.5	11.3631	11.3631	11.4976	15.2561

כמו שהיינו מצפים בסדרה המאוזנת עומק הצומת ומספר פעולות האיזון הם הקטנים ביותר ואין הבדל בין הכנסת סדרה מאוזנת לעץ AVL ל בין הכנסת סדרה מאוזנת לעץ רגיל (כי אין צורך בגלגולים). בנוסף אין הבדל גדול בין העומקים של הצמתים המוכנסים לעצי ה-AVL השונים והם תמיד $\Theta(\log n)$.

סדרת ההכנסות להתחלה בעץ רגיל יוצרת עץ שרוך ולכן עומק הצומת המוכנס ומספר פעולות האיזון הם $\Theta(n)$.

בסדרה האקראית היה יחסית מפתיע לראות שאין הבדל גדול בין העומק הממוצע ומספר פעולות האיזון של צומת המוכנס לעץ AVL ובין צומת המוכנס לעץ רגיל.