תיעוד פרויקט תכנותי - עץ דרגות

מגישים :

נועה ארז

ת"ז: 322467044

שם משתמש: noaerez

גיא גולדרט

ת"ז: 318515996

שם משתמש: GuyGoldrat

תיאור המחלקה AVLNode:

כל אובייקט מטיפוס AVLNode מכיל את השדות הבאים:

1. ערך (value)
2. מצביע לבן השמאלי (left)
3. מצביע לבן הימני (right)
4. מצביע להורה (parent)
5. גובה הצומת (height)
6. גודל תת העץ של הצומת (size)

פעולות המחלקה AVLNode:

שם הפונקציה	תיאור	סיבוכיות זמן
getLeft()	הפונקציה מחזירה את הבן השמאלי של הצומת, או None אם אין כזה.	O(1)
getRight()	הפונקציה מחזירה את הבן הימני של הצומת, או None אם אין כזה.	O(1)
getParent()	הפונקציה מחזירה את ההורה של הצומת, או None אם אין כזה.	O(1)
getValue()	הפונקציה מחזירה את ה value של הצומת, או None אם הצומת הוא וירטואלי.	O(1)
getHeight()	הפונקציה מחזירה את גובה הצומת, או 1- אם הצומת הוא וירטואלי.	O(1)
getSize()	הפונקציה מחזירה את גודל תת העץ של הצומת. במידה והצומת היא עלה תחזיר 1.	O(1)
setLeft(node)	הפונקציה מעדכנת את הבן השמאלי של הצומת להיות node.	O(1)
setRight(node)	הפונקציה מעדכנת את הבן הימני של הצומת להיות node.	O(1)
setParent(node)	הפונקציה מעדכנת את ההורה של הצומת להיות node.	O(1)
setValue(val)	הפונקציה מעדכנת את השדה value להיות שווה לערך val	O(1)
setHeight(h)	הפונקציה מעדכנת את השדה height להיות שווה לערך h	O(1)
setSize(s)	הפונקציה מעדכנת את השדה size להיות שווה לערך s	O(1)
isRealNode()	הפונקציה מחזירה TRUE אם הצומת מייצג צומת אמיתי בעץ (צומת שאינו וירטואלי) .	O(1)

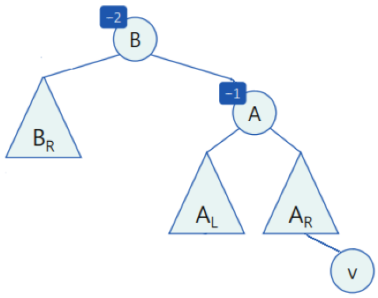
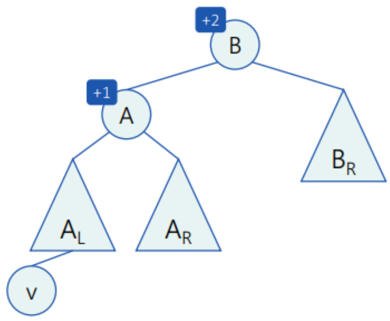
תיאור המחלקה AVLTreeList :

כל אובייקט מטיפוס AVLTreeList מכיל את השדות הבאים:

1. מצביע לראש הרשימה (root)
2. אורך הרשימה (len)
3. מצביע לאיבר הראשון ברשימה (firstItem)
4. מצביע לאיבר האחרון ברשימה (lastItem)
5. מצביע לצומת וירטואלי (virtualNode)

פעולות המחלקה AVLTreeList:

שם הפונקציה	תיאור	סיבוכיות זמן
getRoot()	הפונקציה מחזירה את שורש העץ המייצג את הרשימה.	O(1)
updateHeightNode(node)	בהינתן צומת node הפונקציה מעדכנת את שדה הגובה שלה בהתאם לגבהיי הבנים.	O(1)
getBalanceFactor(node)	בהינתן צומת node, הפונקציה מחזירה את ערך ה-BalanceFactor שלה, המחושב באמצעות שדות הגובה של הבנים.	O(1)
empty()	הפונקציה מחזירה True אם הרשימה ריקה, אחרת תחזיר False.	O(1)
retrieve(i)	הפונקציה מחזירה את ערכו של האיבר ה i ברשימה המתקבל באמצעות קריאה לפונקציה Tree_select עם האינדקס i+1 ושורש העץ.	O(log n) זמן ריצה קבוע מלבד הקריאה לפונקציה Tree_select שסיבוכיות הזמן שלה O(log n).
Tree_select(node , i)	הפונקציה מחזירה מצביע לאיבר במיקום ה i בתת העץ שהצומת node שורש שלו, $1 \leq i \leq$ length . מימוש הפעולה בהתאם למימוש שראינו בהרצאה, כלומר באמצעות שדה ה size נחשב את הדרגה של כל צומת (דרגת הצומת שווה למיקום האיבר ברשימה) ונמשיך בהתאם רקורסיבית לתת עץ הימני או השמאלי בעץ. בכל קריאה רקורסיבית עם תת העץ הימני נעדכן את ערכו של i בהתאם לתת העץ בו מתבצע החיפוש כעת.	O(log n) בכל צומת נפנה לתת עץ הימני או השמאלי בהתאם לi ושדות ה size. במקרה הגרוע עד למציאת האיבר ה i נעבור במסלול מהשורש ועד לעלה. כיוון שמדובר בעץ AVL אורכו המירבי הוא log n, לכן הסיבוכיות לוגריתמית בn, כאשר n הוא מספר הצמתים בעץ.
calculateSize(node)	בהינתן צומת node הפונקציה מחשבת את ערך השדה size המתאים לnode באמצעות שדות הבנים בזמן קבוע.	O(1)

leftRotate(node)	 הפונקציה מקבלת כקלט צומת node עם BalanceFactor = -2 (צומת B בתמונה), ומבצעת גלגול שמאלה באמצעות שינוי המצביעים הבאים (סימטרי ל rightRotate) : B.right $\leftarrow$ A.left B.right.parent $\leftarrow$ B A.left $\leftarrow$ B A.parent $\leftarrow$ B.parent A.parent.left/right $\leftarrow$ A B.parent $\leftarrow$ A כמו כן, עדכון שדות הsize וheight של B,A המעורבים בגלגול.	O(1) הפונקציה משנה ערכי שדות ומצביעים בזמן קבוע.
rightRotate(node)	 הפונקציה מקבלת כקלט צומת node עם BalanceFactor=2 (צומת B בתמונה), ומבצעת גלגול ימינה באמצעות שינוי המצביעים הבאים (סימטרי ל leftRotate) : B.left $\leftarrow$ A.right B.left.parent $\leftarrow$ B A.right $\leftarrow$ B A.parent $\leftarrow$ B.parent A.parent.left/right $\leftarrow$ A B.parent $\leftarrow$ A כמו כן, עדכון שדות הsize וheight של B,A המעורבים בגלגול.	O(1) הפונקציה משנה ערכי שדות ומצביעים בזמן קבוע.
left_rightRotate(node)	בהינתן מצביע לצומת node, הפונקציה קוראת לפונקציה leftRotate עם הבן השמאלי של node ולאחר מכן לפונקציה rightRotate עם הצומת node.	הפונקציות leftRotate, rightRotate מתבצעות בזמן קבוע, לכן סיבוכיות הפעולה left_rightRotate היא O(1)
right_leftRotate(node)	בהינתן מצביע לצומת node, הפונקציה קוראת לפונקציה rightRotate עם הבן הימני של node ולאחר מכן לפונקציה leftRotate עם הצומת node.	הפונקציות leftRotate, rightRotate מתבצעות בזמן קבוע, לכן סיבוכיות הפעולה right_leftRotate היא O(1)

insert(i , val)	הפונקציה מקבלת כקלט ערך val ומספר i ($0 \le i \le$ length) ומכניסה למיקום ה-i ברשימה איבר שערכו val. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו על מנת לשמר את תכונת האיזון. במידה והרשימה ריקה נגדיר את שורש העץ להיות הצומת החדשה. אחרת, נחפש את האיבר הi ברשימה באמצעות קריאה לפעולה Tree_select עם i+1 ונשמור מצביע אליו (item_i). במידה ואין לו בן שמאלי נוסיף את הצומת החדשה כבן השמאלי שלו. אחרת, נמצא את הצומת predecessor המתאימה לאיבר שקדם לו ברשימה ונוסיף את הצומת החדשה כבן הימני של predecessor. נעדכן במידת הצורך את המצביעים firstItem, lastItem לתחילת הרשימה ולסופה, נגדיל את אורך הרשימה ב1 ונאזן את העץ החל מהצומת החדשה שהכנסנו באמצעות קריאה לפונקציה balance. נחזיר את הערך המוחזר מהפעולה balance המציין את מספר פעולות האיזון אשר נדרשו במהלך איזון העץ.	O(log n) במקרה הגרוע, נשתמש בפעולה Tree_select שעלותה היא O(log n) על מנת למצוא את האיבר הi ברשימה (item_i). לאחר מכן, מציאת הצומת predecessor נעשית על ידי קריאה לפעולה max עם תת העץ השמאלי של item_i, לכן עלותה היא O(log m) כאשר m מייצג את מספר האיברים בתת העץ בו מתבצע חיפוש המקסימלי. עדכון השדות מתבצע בזמן קבוע ועלות הקריאה לפונקציה balance היא O(log n). משום שכל הפעולות מתבצעות בזו אחר זו נקבל שסיבוכיות הזמן במקרה הגרוע של הפעולה insert היא O(log n).
balance(node)	הפונקציה מקבלת כקלט מצביע לצומת node, ומתקנת את העץ על מנת לשמר את תכונת האיזון. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו. נאתחל משתנה balanceCnt לספירת פעולות האיזון. איזון העץ מתחיל מההורה של node, נסמנו node.parent . כלומר, מתבצע מעבר על הצמתים במסלול מnode.parent ועד לשורש העץ, כאשר בכל שלב נעדכן מצביע לצומת הקודמת בה ביקרנו במסלול. לכל צומת נעדכן את שדות size ו- height, ונחשב את ערך BalanceFactor שלה. אם BalanceFactor = 2/-2 , תת העץ מצומת זו אינו מאוזן לכן נקרא לפונקציה rotate עם הצומת הנוכחית והצומת הקודמת בה ביקרנו. אחרת, במידה וצומת שינתה את גובהה נספור שינוי זה כפעולת איזון ונמשיך כלפי מעלה. נסכום במשתנה balanceCnt את הפלט המתקבל מהפעולה rotate בכל שלב יחד עם שינויי הגובה ולבסוף נחזיר אותו כפלט.	O(log n) כיוון שמדובר בעץ AVL אורך המסלול מהצומת node ועד השורש הוא לכל היותר log n. עדכון השדות, בדיקת שינויי הגובה וחישוב BalanceFactor מתבצעים בזמן קבוע. הפונקציה rotate אשר אחראית על ביצוע הגלגול המתאים מתבצעת גם היא בזמן קבוע. לכן, הסיבוכיות הכוללת לוגריתמית בn כאשר n הוא מספר האיברים בעץ.
rotate(node , nodeChild)	הפונקציה מקבלת מצביע לצומת node ומצביע לאחד הבנים שלה nodeChild, מחשבת עבורם את ערכי BalanceFactor וקוראת בהתאם לפונקציית הגלגול המתאימה (leftRotate, rightRotate, left_rightRotate או right_leftRotate) עם הצומת node. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בשלב זה בהתאם לסוג הגלגול.	O(1) חישוב ערכי BalanceFactor עבור הצמתים שהתקבלו כקלט וביצוע גלגול מתאים נעשים בזמן קבוע.

delete(i)	בהינתן אינדקס i ($0 \le i < \text{length}$) הפונקציה מוחקת את האיבר הi מהרשימה. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו על מנת לשמר את תכונת האיזון. ראשית, נשמור מצביע node לאיבר במיקום ה- i ברשימה המתקבל באמצעות קריאה לפעולה Tree_select עם האינדקס i+1. נגדיר את המשתנים replaceNode ו updateFrom בהתאם לצומת node : **1.** node הוא **עלה**: replaceNode מצביע לצומת וירטואלי, updateFrom מצביע להורה של node. **2.** ל- node **בן יחיד**: replaceNode ו- updateFrom מצביעים לבן זה. **3.** ל- node **שני בנים**: נמחק את successor(node) באמצעות קריאה לdelete עם i+1, ונגדיר את updateFrom להצביע על הצומת successor(node). כעת, נקרא לפעולה switchNodes(node, replaceNode) , אשר מחליפה בין שתי הצמתים שהתקבלו כקלט באמצעות שינויי מצביעים. נעדכן את אורך הרשימה ואת שדות size ו- height של updateFrom. נעדכן בנוסף את השדות firstItem ו-lastItem במקרה של מחיקת האיבר הראשון או האחרון ברשימה. לבסוף, נקרא לפעולה balance עם הצומת replaceNode ממנה נתחיל את שלב איזון העץ כלפי מעלה. נחזיר את מספר פעולות האיזון המתקבלות כפלט מהפעולה balance. במידה ולצומת node היו שני בנים (מקרה 3) נוסיף לתוצאה את מספר פעולות האיזון שנדרשו במחיקת הsuccessor.	O(log n) נשתמש בפעולה Tree_select שעלותה היא O(log n) על מנת למצוא את האיבר הi ברשימה. מציאת ה - successor ומחיקתו דורשות O(log n) זמן במקרה הגרוע כל אחת. עלות הקריאה לפונקציה switchNodes ושינוי השדות המתאימים היא O(1). לבסוף, סיבוכיות הפונקציה balance היא O(log n), כאשר n הוא מספר הצמתים בעץ. לכן, הסיבוכיות הכוללת היא O(log n).
switchNodes (oldNode,newNode)	הפונקציה מקבלת כקלט מצביעים לצמתים oldNode ו- newNode ומשנה את המצביעים הרלוונטיים כך שהצומת החדשה newNode מחליפה במיקומה את oldNode.	O(1) הפונקציה מבצעת שינויי מצביעים בלבד בהתאם לצמתים שהתקבלו לכן עלותה קבועה.
first()	הפונקציה מחזירה את ערכו של האיבר הראשון ברשימה בזמן קבוע באמצעות תיחזוק מצביע לתחילת הרשימה.	O(1)
last()	הפונקציה מחזירה את ערכו של האיבר האחרון ברשימה בזמן קבוע באמצעות תיחזוק מצביע לסוף הרשימה.	O(1)
min(node)	הפונקציה מחזירה מצביע לאיבר הראשון ברשימה. אם הרשימה ריקה תחזיר מצביע לצומת וירטואלי.	O(log n) נתקדם מהשורש לבן השמאלי בכל פעם עד שנגיע לעלה. אורך המסלול הארוך ביותר מהשורש לעלה בעץ AVL הוא log n, כאשר n הוא מספר הצמתים.
max(node)	הפונקציה מחזירה מצביע לאיבר האחרון ברשימה. אם הרשימה ריקה תחזיר מצביע לצומת וירטואלי.	O(log n) נתקדם מהשורש לבן הימני בכל פעם עד שנגיע לעלה. אורך המסלול הארוך ביותר מהשורש לעלה בעץ AVL הוא log n, כאשר n הוא מספר הצמתים.

listToArray()	הפונקציה מחזירה מערך המכיל את איברי הרשימה לפי סדר האינדקסים, או מערך ריק במידה והרשימה ריקה. הפונקציה נעזרת בפעולה פנימית in_order(node) אשר מקבלת כקלט את שורש העץ node ומבצעת בו סיור in_order שבמהלכו נבנה מערך הפלט הדרוש.	O(n) ראינו בתרגול שבסיור in_order בעץ חיפוש בינארי על כל קשת עוברים לכל היותר פעמיים. מספר הקשתות בעץ הוא n-1, לכן העלות היא O(n).
length()	הפונקציה מחזירה את אורך הרשימה השמור בשדה len.	O(1)
successor(node)	בהינתן מצביע לצומת node המתאימה לאיבר x ברשימה, הפונקציה מחזירה מצביע לצומת המתאימה לאיבר העוקב של x ברשימה. במידה ול- node יש בן ימני, נחזיר את העלה שנמצא במסלול השמאלי ביותר משורש תת העץ הימני של node באמצעות הפעולה min. אחרת, נעלה כלפי מעלה עד שנגיע לצומת y מתוך הבן השמאלי שלה, ואתה נחזיר.	O(log n) בדומה למציאת successor בעץ AVL , מתבצע מעבר על מסלול שאורכו לכל היותר גובה העץ, לכן עלות הפעולה היא O(log n) .
split(i)	בהינתן אינדקס ברשימה i $(0 \le i \le \text{length})$, הפונקציה חוצה אותה לשתי רשימות, ומחזירה מערך באורך 3 באופן הבא: [left , val , right] כאשר left זו רשימת האיברים עד לאינדקס i-1, val הוא ערכו של האיבר במיקום ה i ברשימה, ו right זו רשימת האיברים מאינדקס i+1 ועד סוף הרשימה. ראשית, נשמור מצביע x לאיבר במיקום ה- i ברשימה המתקבל באמצעות קריאה לפעולה Tree_select עם האינדקס i+1. נאתחל שני עצים חדשים לרשימות הפלט T1, T2 כך ש T1 הוא תת העץ השמאלי של x, כלומר האיברים שקטנים ממנו, ואילו T2 הוא תת העץ הימני של x, כלומר האיברים שגדולים ממנו. נמשיך במעלה העץ לאורך המסלול מהצומת x ועד השורש. בכל שלב, במידה והצומת הנוכחית היא בן שמאלי להורה שלה, נקרא מתוך העץ T2 לפונקציה join עם מצביע לצומת ההורה בתור צומת מקשרת, ותת העץ הימני של ההורה כרשימה נוספת. אחרת, במצב בו הצומת הנוכחית היא בן ימני להורה שלה, נקרא מתוך העץ T1 לפונקציה join עם מצביע לצומת ההורה בתור צומת מקשרת, ותת העץ השמאלי של ההורה כרשימה נוספת. בסיום, נקבל עץ T1 המייצג את רשימת האיברים עד לאינדקס i-1, ועץ T2 המייצג את רשימת האיברים מאינדקס i+1 ועד לסוף הרשימה המקורית, כנדרש. נעדכן את השדות firstItem , lastItem עבור שתי הרשימות שהתקבלו.	O(log n) הפונקציה ממומשת בהתאם למימוש שראינו בהרצאה, באמצעות סדרת של פעולות join שעלות כל אחת מהן היא כהפרש הגבהים של העצים שאוחדו, לכן סיבוכיות הזמן הכוללת היא O(log n).

concat(T2)	הפונקציה מקבלת כקלט רשימה T2 , משרשרת אותה אל סוף הרשימה הנוכחית ומחזירה את הערך המוחלט של הפרש הגבהים של עצי הAVL שמוזגו. אם אחת הרשימות ריקה הפונקציה מעדכנת במידת הצורך את המצביע לשורש העץ הנוכחי. אם אחת הרשימות מכילה איבר בודד, נכניס אותו למיקום המתאים ברשימה השנייה ובמידת הצורך נשנה את המצביע לשורש העץ. אחרת, נשמור מצביע x לאיבר האחרון ברשימה הנוכחית, נמחק אותו ונקרא לפונקציה join(x , T2) . לבסוף, נדאג לתחזק את השדה lastItem של הרשימה הנוכחית.	O(height(T2) - height(T1)+1) ובמקרה הגרוע O(log n) הכנסה לאחד העצים נעשית בעלות של O(log n) כאשר n הוא מספר הצמתים בעץ. גישה לאיבר האחרון ברשימה הנוכחית מתבצע בזמן קבוע באמצעות תיחזוק מצביע לסוף הרשימה. סיבוכיות הפונקציה join היא O(height(T2) - height(T1)+1) ובמקרה הגרוע O(log n), כאשר n הוא מספר הצמתים הכולל בשני העצים יחד. לכן בסך הכל נקבל סיבוכיות של O(log n) במקרה הגרוע.
join(x , T2)	הפונקציה מקבלת כקלט מצביע לצומת x ורשימה T2 , הפונקציה מוסיפה את x לסוף הרשימה הנוכחית ומשרשרת לאחר מכן את T2 . בסיום הפעולה הרשימה הנוכחית מורכבת מ [self , x , T2] . במידה ואחת הרשימות ריקה או מכילה איבר בודד, נכניס את האיברים הרלוונטיים לרשימה השנייה ונעדכן את המצביע לשורש העץ במידת הצורך. אחרת, ניגש לעץ הגבוה מבין השניים, ונחפש לאורך המסלול השמאלי / הימני מהשורש (בT2 / בT1) את הצומת הראשונה b שגובהה קטן או שווה לגובה העץ הנמוך. כעת, נגדיר את b ואת העץ הנמוך מבין השניים להיות הבנים של הצומת x ונעדכן את שדות הsize ו - height של x בהתאם. נקרא לפעולה balance עם הצומת x על מנת לשמר את איזון העץ. במידה והעץ המייצג את T2 גבוה יותר, נעדכן את השורש לעץ הרשימה הנוכחית להיות שורש העץ של T2. נעדכן את אורך הרשימה הנוכחית להיות סכום אורכי הרשימות ועוד 1.	O(abs(height(T2) - height(T1))+1) ובמקרה הגרוע O(log n) מפני ששדות הגבהים של הצמתים בעץ מתוחזקים, נוכל להתקדם לבן השמאלי / ימני בכל שלב (בהתאם לעץ הגבוה) עד שנמצא צומת b המתאימה לדרישה. עלות חיפוש כזה היא כהפרש הגבהים, כלומר O(height(T2) - height(T1)+1). סיבוכיות פעולת האיזון balance במקרה הגרוע היא כהפרש הגבהים, משום שזה אורכו של המסלול מהצומת x ועד השורש, כאשר בכל צומת מתבצעת עבודה בזמן קבוע. לכן, נקבל שסיבוכיות הפונקציה היא הפרש הגבהים ובמקרה הגרוע O(log n).
search(value)	הפונקציה מחזירה את האינדקס הראשון ברשימה בו מופיע הערך value או 1- אם לא קיים כזה. הפונקציה ניגשת לאיבר הראשון ברשימה באמצעות תחזוק מצביע אליו, ומבצעת ממנו לכל היותר n קריאות לפעולה successor. נעצור ברגע שנמצא צומת עם ערך שווה לvalue.	O(n) גישה לאיבר הראשון נעשית באמצעות תיחזוק מצביע אליו לכן עלותה O(1). הוכחנו בתרגול כי זמן הריצה של ביצוע n-1 פעמים פעולת successor ברצף החל מהאיבר הראשון ברשימה הוא O(n). זאת מכיוון שישנן n-1 קשתות בעץ ונצטרך לעבור על כל קשת לכל היותר פעמיים. לכן, זמן הריצה במקרה הגרוע הוא O(n) .

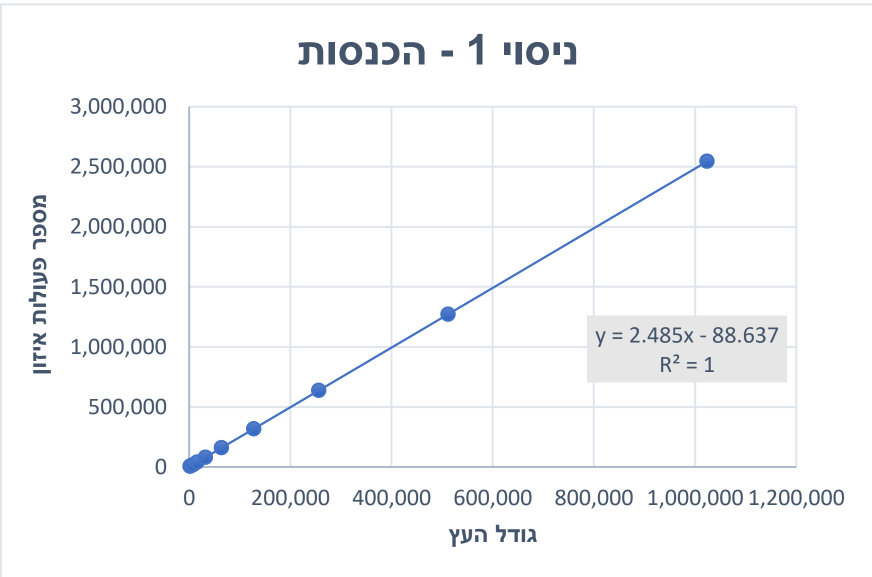
append(val)	הפונקציה מכניסה את val למקום האחרון ברשימה באאמצעות קריאה לפעולה insert. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו על מנת לשמר את תכונת האיזון.	O(log n) באופן זהה לסיבוכיות הפונקציה insert.
getTreeHeight()	הפעולה מחזירה את גובה העץ כולו, 1- עבור עץ ריק.	O(1)

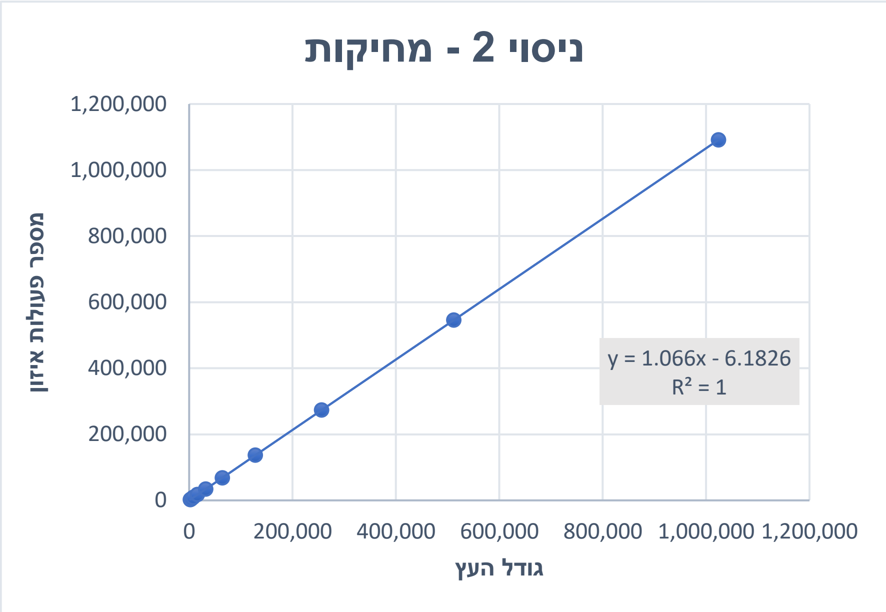
חלק תיאורטי

שאלה 1

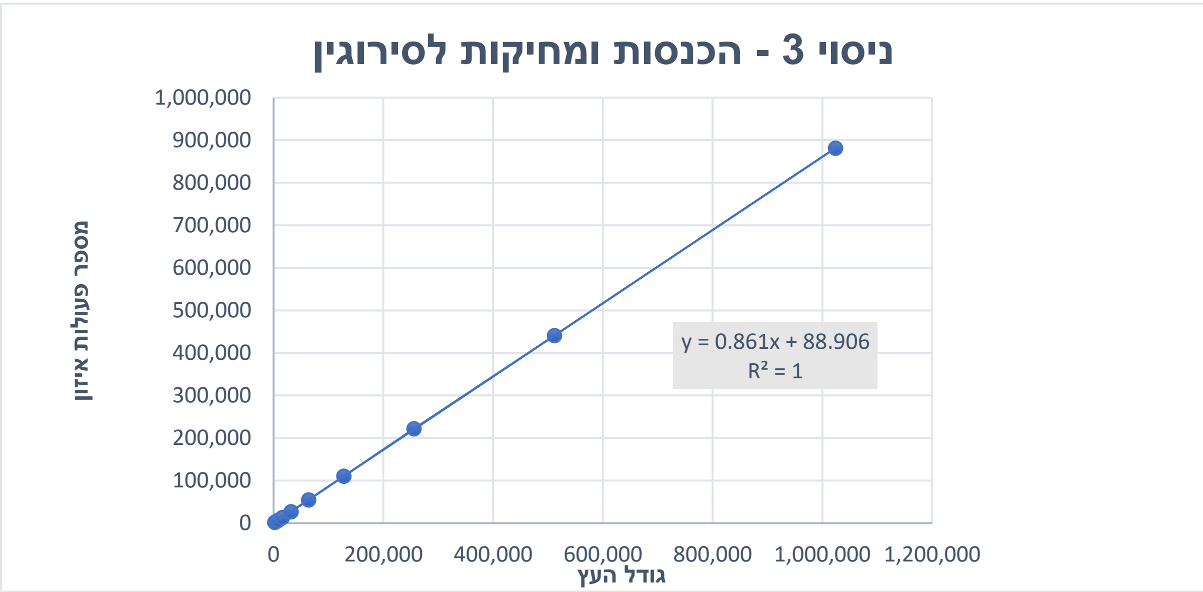
מספר סידורי i	ניסוי 1 - הכנסות	ניסוי 2 - מחיקות	ניסוי 3 - הכנסות ומחיקות לסירוגין
1	4,982	2,121	1,749
2	9,980	4,319	3,492
3	19,886	8,557	6,955
4	39,976	17,031	13,744
5	79,300	34,092	27,107
6	158,991	68,193	54,765
7	318,075	136,347	110,668
8	636,171	272,848	221,673
9	1,270,848	545,920	440,830
10	2,545,180	1,091,512	881,493

הביטוי האסימפטוטי התואם לכל עמודה הוא $O(n)$.





ניסוי 2 - מחיקות
y = 1.066x - 6.1826
R² = 1
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
מספר פעולות איזון
0
200,000
400,000
600,000
800,000
1,000,000
1,200,000
גודל העץ



ניסוי 3 - הכנסות ומחיקות לסירוגין
y = 0.861x + 88.906
R² = 1
1,000,000
900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
מספר פעולות איזון
0
200,000
400,000
600,000
800,000
1,000,000
1,200,000
גודל העץ

שאלה 2

.1

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	1.45	4	1.7	12
2	1.8	4	1.7	13
3	1.54	3	1.83	14
4	1.7717	9	1.57	15
5	1.75	4	1.92	17
6	1.52	5	1.84	18
7	1.27	5	1.52	19
8	1.53	7	1.764	20
9	1.54	8	1.77	21
10	1.5	5	1.625	22

.2

- עלות join ממוצע לשני התרחישים:

מהטבלה ניתן להסיק כי עלות join ממוצע עבור split אקראי הוא O(1). כלומר, רוב פעולות ה join שנבצע יהיו על תתי עצים בעלי הפרש גבהים נמוך, ואילו איחוד של עצים בעלי הפרש גבהים גבוה הוא נדיר יחסית. לכן, נקבל כי amortized time של פעולת join היא O(1).

התוצאות מתיישבות עם ניתוח הסיבוכיות התיאורטי, לפיו עלות פעולת split היא O(log n).

במקרה הגרוע, בעת ביצוע split לפי צומת שהיא עלה בעץ, מספר פעולות join שנצטרך לבצע הוא כגובה העץ, וכיוון שהעלות הממוצעת לפעולת join היא קבועה, נקבל חסם עליון של O(log n) לפעולת split בהתאם לניתוח הסיבוכיות התיאורטי.

- עלות join ממוצע עבור split של האיבר המקסימלי בתת העץ השמאלי :

לאיבר המקסימלי בתת העץ השמאלי של השורש אין בן ימני, לכן מתכונת האיזון של AVL נובע שהוא בעצמו עלה או שיש לו בן שמאלי שהוא עלה.

הוכחנו בתרגיל בית 2 שעומקו של כל עלה בעץ AVL מגובה h הוא לפחות $\frac{h}{2}$, לפיכך, עומקו של איבר זה בעץ הוא לכל הפחות $\frac{h}{2}-1$, כלומר $O(\log n)$. לכן, עבור split של האיבר מקסימלי בתת העץ השמאלי נצטרך לבצע O(log n) פעולות join.

עלות סדרת פעולות join היא:

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

לכן, $Average\ join = \frac{O(\log n)}{O(\log n)} = O(1)$. כלומר, קיבלנו שעלות join ממוצע עבור split של האיבר המקסימלי בתת העץ השמאלי היא O(1) בהתאמה לתוצאות הניסוי.

3.

עלות join מקסימלי עבור split של האיבר המקסימלי בתת העץ השמאלי :

בעת ביצוע split של האיבר המקסימלי בתת העץ השמאלי, נתקדם במעלה העץ החל מהאיבר אותו בחרנו ועד השורש. בכל פעם נעלה אל ההורה מתוך הבן הימני שלו, לכן נבצע join לעצים המכילים איברים שקודמים לאיבר ממנו התחלנו ברשימה. הפנייה הראשונה להורה מתוך בנו השמאלי תהיה כאשר נפנה לשורש מתוך תת העץ השמאלי שלו. במצב זה נבצע join לתת העץ הימני של השורש יחד עם תת העץ הימני של הצומת ממנה התחלנו את הפיצול. צומת זו היא המקסימלית בתת העץ, לכן תת העץ הימני שלה הוא ריק וגובהו 1-.

כעת, נקבל שעלות join מקסימלי עבור פעולת split כמתואר, המוגדרת להיות הפרש הגבהים המקסימלי של העצים שאוחדו שווה לכל היותר לגובה תת העץ הימני של השורש ועוד 1, כלומר שווה לגובה העץ כולו h או ל- h-1.

לכן, העלות המקסימלית של פעולת join היא O(log n).
התוצאות תואמות לניתוח הסיבוכיות התיאורטי, לפיו סיבוכיות הפעולה join היא כהפרש הגבהים, כלומר O(abs (height(T2) - height(T1))+1), ובמקרה הגרוע בו אחד העצים ריק O(log n).

שאלה 3

מספר פעולות איזון בממוצע	**עץ AVL -** הכנסות בתחילת הרשימה	**עץ לא מאוזן -** הכנסות בתחילת הרשימה	**עץ AVL -** הכנסות סדרה מאוזנת	**עץ לא מאוזן -** הכנסות סדרה מאוזנת	**עץ AVL -** סדרת הכנסות אקראית	**עץ לא מאוזן -** סדרת הכנסות אקראית
1	2.974	498	0.994	0.994	2.51	1.878
2	2.986	998	0.997	0.997	2.44	1.924
3	2.989	1498	0.997	0.997	2.46	1.88
4	2.992	1998	0.998	0.998	2.50	1.89
5	2.993	2498	0.999	0.999	2.44	1.88
6	2.994	2998	0.999	0.999	2.46	1.91
7	2.995	3498	0.999	0.999	2.47	1.88
8	2.996	3998	0.999	0.999	2.47	1.87
9	2.996	4498	0.999	0.999	2.49	1.83
10	2.996	4998	0.999	0.999	2.47	1.88

מספר פעולות איזון בממוצע:

הכנסות בתחילת הרשימה – מתוצאות הניסוי עולה כי מספר פעולות האיזון בממוצע בעץ AVL שואף לקבוע. בכל הכנסה לתחילת הרשימה נבצע עדכון גבהים ולבסוף גלגול ימינה במידת הצורך. חלק מההכנסות ידרשו פעולות איזון רבות יחסית ואילו חלק מההכנסות ידרשו רק פעולת איזון אחת. בסך הכל נקבל כי בכל הכנסה כזו נבצע 3 פעולות איזון בממוצע (גלגול ימינה ושינוי גובה של שתי צמתים).

לעומת זאת, בכל הכנסה בתחילת הרשימה המיוצגת על ידי עץ לא מאוזן נשנה את הגבהים של כל שאר האיברים בעץ ולכן תוצאות הניסויי מתאימות לציפייה.

הכנסות בסדרה מאוזנת – מכיוון שסדר ההכנסה לרשימה אחראי לכך ששני העצים יהיו תמיד מאוזנים, לא ידרשו פעולות גלגול של עץ AVL ולכן למעשה מספר פעולות האיזון הכולל והממוצע (הכוללות רק שינויי גובה של צמתים) יהיה זהה בין שני העצים.

הכנסות אקראיות – ציפינו לכך שבממוצע בהכנסות אקראיות העץ שייווצר יהיה יחסית מאוזן, כאשר בכל הכנסה המפרה את תכונת האיזון יתבצע גלגול מתאים והגדלת סכום פעולות האיזון. מתוצאות הניסוי עולה כי מספר פעולות האיזון בממוצע בסדרת הכנסות אקראית בעץ AVL שואף לקבוע.

כמו כן, מתוצאות הניסוי עולה כי מספר פעולות האיזון הממוצע של סדרת הכנסות אקראית בעץ AVL נמוך מזה של סדרת הכנסות בתחילת הרשימה בעץ AVL. עובדה זו תואמת לציפייה שלנו מפני שסדרת הכנסות לתחילת הרשימה דורשת פעולות איזון רבות יותר מאשר הכנסה אקראית שייתכן ולא דורשת פעולות איזון כלל.

לעומת זאת, בעץ לא מאוזן נספור כפעולת איזון רק את שינויי הגובה, ומכיוון שההכנסה נעשית באופן אקראי העץ יישאר מאוזן יחסית. מתוצאות הניסוי עולה כי מספר פעולות האיזון הממוצע של סדרת הכנסות אקראית בעץ לא מאוזן גבוה מזה של סדרת הכנסות מאוזנת לעץ לא מאוזן. עובדה זו תואמת לציפיות שלנו מפני שסדרת הכנסות מאוזנת מניבה עץ בעל הגובה המינימלי גם ללא מנגנון איזון וסביר שבה מספר פעולות האיזון הממוצע יהיה הנמוך ביותר מבין כל ההכנסות.

עומק הצומת המוכנס בממוצע	**עץ AVL -** הכנסות בתחילת הרשימה	**עץ לא מאוזן -** הכנסות בתחילת הרשימה	**עץ AVL -** הכנסות סדרה מאוזנת	**עץ לא מאוזן -** הכנסות סדרה מאוזנת	**עץ AVL -** סדרת הכנסות אקראית	**עץ לא מאוזן -** סדרת הכנסות אקראית
1	7.89	499.5	7.89	7.89	8.07	11.10
2	8.98	999.5	8.98	8.98	9.14	12.26
3	9.63	1499.5	9.63	9.63	9.71	15.39
4	9.97	1999.5	9.97	9.97	10.10	13.63
5	10.36	2499.5	10.36	10.36	10.46	13.35
6	10.63	2999.5	10.63	10.63	10.77	15.56
7	10.83	3499.5	10.83	10.83	10.95	14.52
8	10.97	3999.5	10.97	10.97	11.13	14.45
9	11.18	4499.5	11.18	11.18	11.29	16.26
10	11.36	4999.5	11.36	11.36	11.48	14.50

עומק הצומת המוכנס בממוצע:

הכנסות בתחילת הרשימה – גובה עץ AVL בעל n איברים הוא בגובה $h = \lfloor \log n \rfloor$ כאשר אנו מוסיפים כל פעם איבר בעומק של h. מכיון שברמה הi של העץ יש 2^i איברים, מרבית האיברים יכנסו ברמה התחתונה בעץ ולכן עומק הצומת המוכנס בממוצע יהיה קרוב ל log n, בהתאם לציפייה שלנו.

לעומת זאת, בעץ לא מאוזן ההכנסה של האיבר הi תהיה בעומק i , לכן בסך הכל עומק הצומת המוכנס בממוצע עבור עץ בגודל k שווה לסכום סדרה חשבונית חלקי k , כלומר: $\frac{k^2}{2\cdot k} = \frac{k}{2}$.

הכנסות סדרה מאוזנת – נשים לב כי אין הבדל בין התוצאות של עץ AVL לעץ לא מאוזן. גובה עץ מאוזן בעל n איברים הוא בגובה $h = \lfloor \log n \rfloor$ כאשר אנו מוסיפים כל פעם איבר בעומק של h. מכיוון שברמה הi של העץ יש 2^i איברים, מרבית האיברים יכנסו ברמה התחתונה בעץ ולכן עומק הצומת המוכנס בממוצע יהיה קרוב ל log n.

הכנסה אקראית – התוצאות בהכנסה אקראית לעץ AVL דומות לתוצאות בהכנסה מאוזנת. ציפינו לכך שבממוצע בהכנסות אקראיות העץ שייווצר יהיה יחסית מאוזן, לכן התוצאות תואמות לציפיות שלנו. בדומה לסדרת הכנסות מאוזנת, מרבית האיברים יכנסו ברמה התחתונה בעץ ולכן עומק הצומת המוכנס בממוצע יהיה קרוב ל log n.
בעץ לא מאוזן סביר שהעץ יהיה עמוק יותר ולכן ייתכנו הכנסות של איברים בעומק גדול יותר, דבר המשפיע על עומק הצומת הממוצע.