**מבני נתונים – פרויקט מעשי 1 - קובץ תיעוד** תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**פרטים אודות מגישות-**

הגר פייתן

* שם משתמש - hagarleap
* תעודת זהות - 206825176

גל קריאל

* שם משתמש - galkariel
* תעודת זהות - 318459666

**בקובץ שלנו יש שתי מחלקות –**

**AVLNode** – מחלקה המייצגת צמתים.

**AVLTreeList** – מחלקה המייצגת עץ, המורכב מצמתים ממחלקת AVLNode. כאשר העץ מממש List.

נפרט עבור כל מתודה מה היא עושה, כיצד פועלת ומה הסיבוכיות זמן ריצה שלה.

עבור פונקציות שעולות זמן קבוע – רק נתן תיאור קצר כפי שנדרש בקובץ ההנחיות.

**AVLNode**

**השדות במחלקה :**

* Value – הערך שנמצא בצומת
* AVLNode left – בן ימני
* AVLNode right – בן שמאלי
* AVLNode parent - הורה
* Height – גובה
* Size - גודל
* HeightUpdate – האם הגובה עודכן או לא (ביטוי בוליאני)

**בנאים של המחלקה:**

בנאי המקבל ערך אחד – value ומעדכן את השדות בערכים הבאים –

* Value – ה- value שקיבל
* AVLNode left – None
* AVLNode right – None
* AVLNode parent - None
* Height – -1
* Size - 0
* HeightUpdate – False

**פירוט המתודות במחלקה:**

|  |  |  |
| --- | --- | --- |
| שם המתודה | הסבר ופירוט | ניתוח סיבוכיות זמן ריצה |
| getLeft | מחזירה מצביע לבן השמאלי של הצומת (שדה left) | O(1) - זמן קבוע |
| getRight | מחזירה מצביע לבן הימני של הצומת (שדה right) | O(1) - זמן קבוע |
| getParent | מחזירה מצביע להורה של הצומת (שדה parent) | O(1) - זמן קבוע |
| getValue | מחזירה את השדה value של הצומת | O(1) - זמן קבוע |
| getHeight | מחזירה את השדה height של הצומת | O(1) - זמן קבוע |
| getSize | מחזירה את השדה size של הצומת | O(1) - זמן קבוע |
| getHeightUpdate | מחזירה את השדה HeightUpdate של הצומת | O(1) - זמן קבוע |
| setLeft | מקבלת AVLNode, ומעדכנת את המצביע של הבן השמאלי של הצומת להיות AVLNode | O(1) - זמן קבוע |
| setRight | מקבלת AVLNode, ומעדכנת את המצביע של הבן הימני של הצומת להיות AVLNode | O(1) - זמן קבוע |
| setParent | מקבלת AVLNode, ומעדכנת את המצביע של ההורה של הצומת להיות AVLNode | O(1) - זמן קבוע |
| setValue | מקבלת ערך ומעדכנת את השדה value של הצומת להיות הערך שקיבלה | O(1) - זמן קבוע |
| setHeight | מקבלת ערך ומעדכנת את השדה Height של הצומת להיות הערך שקיבלה | O(1) - זמן קבוע |
| setHeightUpdate | מקבלת ערך ומעדכנת את השדה HeightUpdate של הצומת להיות הערך שקיבלה | O(1) - זמן קבוע |
| setSize | מקבלת ערך ומעדכנת את השדה Size של הצומת היות הערך שקיבלה | O(1) - זמן קבוע |
| isRealNode | מקבלת צומת ובודקת האם הצומת אמיתי (אם הגובה שלו -1 וגם הערך שלו None). ו**מחזירה** אמת/שקר בהתאם. | O(1) - זמן קבוע |
| updateMeasurements | מקבלת צומת. מעדכנת את שדות גובה וגודל הצומת לפי הגובה והגודל של הבנים שלו. בנוסף בודקת האם הגובה שלו השתנה ואם כן מעדכנת את HeightUpdate להיות אמת. | O(1) - זמן קבוע |
| fix\_size\_rec | מקבלת צומת. מבצעת לולאה אשר רצה מהצומת עד שמגיעה לשורש. בכל איטרציה מעדכנת את הגודל של הצומת בה נמצאת (לפי הגודל של הבנים שלה). | העדכון מתבצע מצומת עד הרוש. גובה עץ AVL הוא לוגריתמי במספר הצמתים. לכן יהיו logn איטרציות. בכל איטרציה מתבצע זמן קבוע לכן בסך הכל נקבל -  **O(log(n))** |
| getBF | מקבלת צומת ו**מחזירה** את ה- balance factor שלה (באמצעות הגבהים של הבנים של הצומת) אם זה צומת אמיתי. אם זה צומת וירטואלי מחזירה 0. | O(1) - זמן קבוע |
| getPredecessor | כמו שראינו בכיתה-  מקבלת צומת ומחזירה את הצומת ה"קודם" שלו –  אם לצומת יש בן שמאלי נלך פעם אחת שמאלה ואז כל הדרך ימינה וקיבלנו את "הקודם". אחרת נעלה מעלה (לכיוון השורש) עד הפנייה הראשונה למעלה שמאלה והיא תהיה הצומת "הקודם". | **אם לצומת יש בן שמאלי –** מבצעים קריאה לפונצקית MaxNode() שהיא בסיבוכיות **O(log(n))**  **אחרת** נכנסים ללולאה שעולה כל פעם להורה עד הפנייה הראשונה למעלה שמאלה- כלומר לכל היותר log(n) איטרציות. בכל איטרציה מתבצע זמן קבוע אז נקבל עלות של **O(log(n))** עבור מקרה זה.  ובסה"כ לכל מקרה נקבל **סיבוכיות כוללת O(log(n))** |
| getSuccessor | *כמו שראינו בכיתה- מקבלת צומת ומחזירה את הצומת ה"עוקב שלה-*  *אם לצומת אין בן ימני נעלה למעלה (לכיוון השורש) עד הפניה הראשונה ימינה וזה יהיה הצומת ה"עוקב". אחרת יש לצומת בם ימני אז נרד אליו ואז נלך כל הדרך שמאלה וקיבלנו את ה"עוקב".* | **אם לצומת יש בן ימני –** מבצעים קריאה לפונצקית MinNode() שהיא בסיבוכיות **O(log(n))**  **אחרת** נכנסים ללולאה שעולה כל פעם להורה עד הפנייה הראשונה למעלה ימינה- כלומר לכל היותר log(n) איטרציות. בכל איטרציה מתבצע זמן קבוע אז נקבל עלות של **O(log(n))** עבור מקרה זה.  ובסה"כ לכל מקרה נקבל **סיבוכיות כוללת O(log(n))** |
| AVLdelete | מקבלת צומת ומעדכנת את המצביע של הבן הימני , הבן השמאלי וההורה של הצומת להיות None. | O(1) - זמן קבוע |
| is\_left\_child | מקבלת צומת ובודקת האם הצומת הוא בן שמאלי. אם כן מחזירה אמת. אחרת מחזירה שקר. | O(1) - זמן קבוע |
| MinNode | מקבלת צומת והולכת כל הדרך שמאלה עד שמגיעה לצומת שהיא עלה- מחזירה אותה. | לולאה שיורדת כל פעם שמאלה עד שמגיעים לעלה- כלומר לכל היותר log(n) איטרציות (כגובה העץ). בכל איטרציה מתבצע זמן קבוע אז נקבל סיבוכיות כוללת **O(log(n))**. |
| MaxNode | מקבלת צומת והולכת כל הדרך ימינה עד שמגיעה לצומת שהיא עלה- מחזירה אותה. | לולאה שיורדת כל פעם ימינה עד שמגיעים לעלה- כלומר לכל היותר log(n) איטרציות (כגובה העץ). בכל איטרציה מתבצע זמן קבוע אז נקבל סיבוכיות כוללת **O(log(n))**. |
| Tree\_rank | מקבלת צומת ומחזירה את האינדקס של הצומת "כרשימה". | לולאה עולה בכל איטרציה להורה (עד שמגיעים לשורש)- כלומר לכל היותרההתבצעו log(n) איטרציות (כגובה העץ). בכל איטרציה מתבצע זמן קבוע אז נקבל סיבוכיות כוללת **O(log(n))**. |

**AVLTreeList**

**השדות במחלקה :**

* Root – הצומת שמייצגת את השורש
* Min – הצומת שמייצגת את האיבר הכי קטן בעץ (לפי אינדקס בעצם כי זה treeList)
* Max - הצומת שמייצגת את האיבר הכי גדול בעץ (לפי אינדקס בעצם כי זה treeList)

**בנאים של המחלקה:**

הבנאי לא מקבל ערכים כלל, אנו מעדכנים את השדות בערכים הבאים –

* Root – צומת עם ערך value - None
* Min – השורש
* Max - השורש

**פירוט המתודות במחלקה**

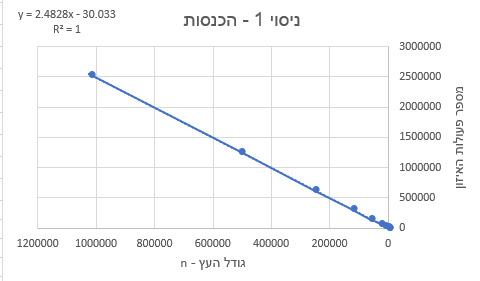
(פונקציות פנימיות יתועדו בתוך הפונקציה החיצונית להן)

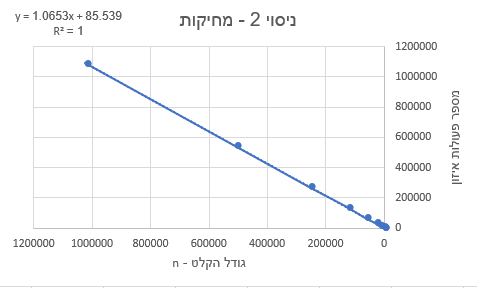
|  |  |  |
| --- | --- | --- |
| שם המתודה | הסבר ופירוט | ניתוח סיבוכיות זמן ריצה |
| Empty | מקבלת עץ ובודקת האם העץ ריק, אם כן מחזירה אמת. אחרת מחזירה שקר. | O(1) - זמן קבוע |
| TreeSelect -  TreeSelectRec פונקציה פנימית של TreeSelect | מקבלת אינדקס ומחזירה את הצומת במיקום האינדקס בעץ-רשימה. | TreeSelect היא פונקציית מעטפת אשר קוראת לפונקציה הרקורסיבית TreeSelectRecעם השורש של העץ. לכן הסיבוכיות תהיה על פי TreeSelectRec:  בכל קריאה רקורסיבית יש סיבוכיות זמן קבועה כי רק עושים מספר סופי של פעולות בעלות קבועה.  לכן הסיבוכיות תלויה בכמות הקריאות הרקורסיביות. כמות הקריאות תלויה בגובה העץ, כיוון שבכל קריאה נלך לבן הימני או השמאלי עד שנגיע לצומת במיקום של האינדקס.  במקרה הגרוע נגיע לעלה, כלומר יהיו כמות קריאות בגודל גובה העץ. כיוון שמדובר בעצי AVL גובה העץ יהיה logn. ולכן במקרה הגרוע יהיו logn קריאות רקורסיביות – כלומר הסיבוכיות הכוללת תהיה **O(log(n))**. |
| retrieve | מקבלת אינדקס ומחזירה את ערך האיבר(צומת) במקום ה-i אם קיים, אחרת היא מחזירה None | קריאה אחת לפונקציית TreeSelect ()- שהיא בעלת סיבוכיות logn והחזרת הערך של הצומת שהוחזרה מהפונקציה TreeSelect.  כלומר הסיבוכיות הכוללת תהיה **O(log(n))**. |
| insert | מבצעת הכנסת צומת לפי האלגוריתם מהשיעור. מחזירה את כמות פעולות האיזון. | אם העץ ריק, מתבצע פעולות שלוקחות זמן קבוע.  אחרת:  -אם האינדקס 0 או אורך הרשימה, מ,תבצע הכנסה בעזרת מינימום או מקסימום- זמן קבוע.  -אחרת, מוצאים את מקום ההכנסה בעזרת TreeSelect, סיבוכיות log(n)  - בסוף מתקיים לולאת while שיכולה לעבור על גובה העץ במקרה הגרוע. לפי הצורך יתבצע גלגול יחיד (סיבוכיות log(n) ללולאה)  אם נכנסים לאיטרציה של הלולאה, אתקיים update measurements על הצומת, שלוקח זמן קבוע. ייתכן כי יתבצע ImplementRotation (O(1)) אם הbalanceFactor גדול מ1 (O(1)).  סה"כ במקרה הכי גרוע, הסיבוכיות הינו **O(log(n))**. |
| ImplementRotation | קובע איזה סוג של רוטציה לבצע על בסיס הbalance factor של הצומת הנתונה ובניו. | getBF בעל סיבוכיות קבוע, וRightRotation וLeftRotation בעלי סיבוכיות קבוע. לכן סיבוכיות קבוע O(1) |
| RightRotation | משנה מצביעים ומעדכן גבהים וגדלים על מנת לבצע סיבוב ימין כי שלמדנו בכיתה | מתקיים רק שינוי של מצבעים ע"י פונקציות שפועלות בזמן קבוע. לכן סיבוכיות הפונקציה הינו קבוע O(1) |
| LeftRotation | בדומה ל RightRotation רק בכיוון שמאל (סימטרי – ימין הופך לשמאל). | O(1) - זמן קבוע |
| delete | בהינתן אינדקס i ועץ, מוחק את איבר ה-i. מחזיר כמות הפעולות איזון | קודם מחפש את הצומת שעלינו למחוק בעזרת TreeSelect, O(log(n)).  אז מוחק את הצומת לפי האלגוריתם שלמדנו:  -מקרה 1: מוחקים את הצומת בעזרת בפונקציות קבועות, O(1)  -מקרה 2: מתבצע מחיקה על בסיס מיקום הבן שלו. בכל מקרה מתקיים מחיקה בעזרת פונקציות קבועות, O(1)  -מקרה 3: מפעילים פעם אחת getSuccessor, בעלת סיבוכיות O(log(n)). אחר כך מוחקים עם פונקציות קבועות.  אחר כך מפעילים delete\_style\_balancing, סיבוכיות log(n)  סה"כ במקרה הגרוע: 3log(n)  = **O(log(n))**. |
| delete\_style\_balancing | מבצע גלגולים ועדכונים מהצומת הנתונה עד לשורש (עבור delete ו-join/split) | יש לולאת while שעוברת מהצומת עד השורש, במקרה הכי גרוע log(n) איטרציות. בכל איטרציה מפעילים updatemeasurements, ולפי הצורך מופעל ImplementRotation (שניהם O(1))  בסוף אם לא הגענו לשורש, אז מפעילים fix\_size\_rec על מנת לתקן שדה הsize- O(log(n)).  אז סה"כ **O(log(n))**. |
| first | מחזיר את שדה value של איבר הראשון | O(1) כי אנחנו מתחזקות שדה min ב-AVLTreeList |
| last | מחזיר את שדה value של איבר האחרון | O(1) כי אנחנו מתחזקות שדה max ב-AVLTreeList |
| listToArray  listToArray\_rec - פונקציה פנימית של listToArray | הופך את העץ לרשימה של פייתון באופן רקורסיבי | בונים רשימה ריקה ואז שולחים את השורש לפונקציה פנימית listToArray\_rec. שם עושים אלגוריתם דומה לtreewalk, רק כל פעם שולח את הvalue של הצומת לרשימה. עוברים על כל העץ לכן סיבוכיות O(n) |
| length | מחזירה את אורך הרשימה | מחזירים את שדה הגודל של השורש- זמן קבוע O(1) |
| split | מפצל רשימה לשני רשימות על במקום האינדקס שנתון, ומחזיר רשימה עם שני הרשימות החדשות ו את האיבר במקום האינדקס | קודם, נמצא את הצומת בעזרת TreeSelect, O(log(n))  אם i=0 או האיבר האחרון, אז פשוט נבצע מחיקה רגילה ונחזיר עץ ריק, ואת הרשימה המעודכנת. O(log(n))  אחרת, נאתחל את העצים large ו small עם התת עץ השמאלי וימני, וזה עניין של הפעלת מספר פונקציות קבועות.  אז בעזרת לולאת while נעבור מהצומת עד לשורש ונפעיל join על התת עץ שממנו לא "עלינו" והעץ הרלוונטי. יש לנו משתנה בולאני come\_from\_left ששומר אם עלינו אליו מצד שמאל או ימין, ובעזרתו הפונקציה מבחינה אם להפעיל join על העץ small או large. עלות המצטברת של הjoin זה log(n) כפי שלמדנו בשיעור.  לכן סה"כ במקרה הגרוע הסיבוכיות הינו **O(log(n))**. |
| concat | מקבלת רשימה ומשרשרת אותה אל סוף הרשימה הנוכחית. הפונקציה מחזירה את הערך המוחלט של הפרש הגבהים של עצי הAVL שמוזגו. | בתחילת הפונקציה מתבצעות פעולות בזמן קבוע. לאחר מכן  אם העץ לא ריק-  מתבצעות פעולות קבועות (קריאות לפונקציות שפועלות בזמן קבוע), קריאה אחת לפונקציית delete- O(log(n)), קריאה אחת לפונקציית AVL\_join - O(log(n).  אם העץ ריק- נבצע פעולה בזמן קבוע.  לכן בכל מקרה נקבל סיבוכיות כוללת O(log(n)). |
| AVL\_join  join פונקציה פנימית של AVL\_join  has\_left\_child - פונקציה פנימית של join  has\_right\_child- פונקציה פנימית של join | מקבלת שני עצי AVL וצומת מקשר X ומחזירה עץ מאוחד. | עושים מספר פעולות בזמן קבוע ואז קוראים לפונקציה פנימית join- לכן הסיבוכיות תהיה על פי join:  בתחילת הפונקציה יש לולאת while שיורדת מהשורש של העץ הגדול יותר למטה עד שמגיעה לגובה של העץ הקטן יותר. במקרה הכי גרוע העץ הקטן הוא עץ ריק לכן נקבל log(n) איטרציות. בכל איטרציה פעולות בזמן קבוע לכן נקבל שהלולאה פועלת בסיבוכיות **O(log(n).**  לאחר מכן מתבצעות פעולות בזמן קבוע ועבור מקרים מסוימים תהיה קריאה לאחת מהפונקציות has\_left\_child או  has\_right\_child-  בשתי הפונקציות האלה בודקים האם לצומת מסוים יש בן ימני/שמאלי – **זמן קבוע.**  לאחר מכן מתבצעות פעולות נוספות **בזמן קבוע.**  ובסוף יש קריאה אחת לפונקציית  delete\_style\_balancing עם העץ המאוחד-  **O(log(n).**  לכן בסה"כ נקבל סיבוכיות כוללת **O(log(n)).** |
| search  search\_rec - פונקציה פנימית של search | בהינתן ערך, מחפשים אם קיים צומת עם הערך הזה ומחזירים את האינדקס של המקרה הראשון שלו ברשימה. אם לא נמצא מחזירים -1 | עוברים על כל הצמתים בעץ בפונקציה רקורסיבית פנימית search\_rec ובכל קריאה רקורסיבית יש פעולות שמתקיימות בזמן קבוע. לכן יש n קריאות של O(1). לכן סיבוכיות הרקורסיה הינה O(n).  בסוף קוראים לtree\_rank אם נמצא הערך המבוקש. סיבוכיות הקריאה הינה O(log(n)). אז מטענה הסיבוכיות הכוללת זה **O(n).** |
| getRoot | מחזיר את השורש אם הרשימה לא ריקה, אחרת None | זמן קבוע O(1) |

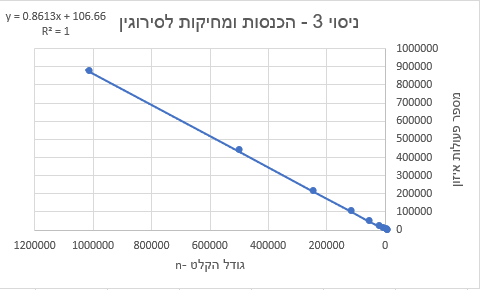
**חלק ניסויי/תיאורטי**

**שאלה 1 - סעיף 1**

|  |  |  |  |
| --- | --- | --- | --- |
| מספר סידורי i | ניסוי 1 - הכנסות | ניסוי 2 - מחיקות | ניסוי 3 - הכנסות ומחיקות לסירוגין |
| 1 | 4935 | 2130 | 1749 |
| 2 | 9816 | 4245 | 3473 |
| 3 | 19726 | 8522 | 6866 |
| 4 | 39586 | 17073 | 13598 |
| 5 | 79221 | 34146 | 27523 |
| 6 | 158789 | 68393 | 55030 |
| 7 | 318022 | 136532 | 110282 |
| 8 | 636806 | 272863 | 220760 |
| 9 | 1269953 | 545712 | 442745 |
| 10 | 2542596 | 1090789 | 881209 |







**שאלה 1 - סעיף 2**

ניתן לראות מהשרטוטים עבור כל אחד מהניסויים כי קיבלנו ביטוי אסימפטוטי **לינארי בגודל הקלט.**

זאת כיוון שערך ה-R בריבוע שווה לאחד – כלומר קירוב טוב של התוצאות לגרף הלינארי.

**שאלה 2- סעיף 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 | 1.454 | 5 | 1.6 | 12 |
| 2 | 1.75 | 4 | 1.333 | 13 |
| 3 | 2.0 | 5 | 1.75 | 14 |
| 4 | 1.538 | 8 | 1.5 | 16 |
| 5 | 1.333 | 5 | 2.083 | 17 |
| 6 | 1.928 | 5 | 1.6 | 18 |
| 7 | 1.562 | 7 | 1.625 | 19 |
| 8 | 1.666 | 6 | 1.388 | 20 |
| 9 | 1.777 | 6 | 2.0 | 21 |
| 10 | 1.833 | 6 | 1.4 | 23 |

**שאלה 2- סעיף 2**

תרחיש ראשון - ניתוח תאורטי של עלות ממוצעת של join עבור split **אקראי** –

יהי עץ AVL עם n איברים וגובה log(n). יהיה צומת אקראי עם גובה h כלשהי, עבורו נבצע split.

לפי האלגוריתם, אנחנו מאתחלים שני עצים: smaller עם התת עץ השמאלי של צומת האקראי ו-bigger עם התת עץ ימני של הצומת האקראי. שניהם בגובה בין **h-1 או h-2** (מהבנייה של עץ AVL).

אז, נעלה כל פעם להורה (עד שנגיע לשורש) ונסמן אותו node\_parent. כל פעם יהיה פעולת join על אחד מהעצים smaller או bigger עם התת עץ של node\_parent שממנו **לא** עלינו (בהתאם לאלגוריתם שלמדנו).

נשים לב- סה"כ **כמות פעולות הjoin שנבצע יהיה log(n)-h.**

לפי מה שלמדנו בהרצאה, עלות של פעולת join אחד הינו O(bigger\_tree.height() – smaller\_tree.height() +1) (כלומר הפרש הגבהים בין העץ הגבוה היותר לעץ הנמוך ביותר פלוס אחד).

ועלות כוללת של פעולות join תהיה ההפרש הגבהים בין העץ הגבוה ביותר לעץ הנמוך ביותר פלוס מספר פעולות ה-join שעשינו (ראינו בכיתה - בטור טלסקופי).

נסכום את פעולות ה-join עבור smaller וbigger בנפרד, ואז סכום שני הסכומים יהווה את הסכום הכולל של פעולות ה-join. נחלק סכום זה ב log(n)-h וקיבלנו את הממוצע. אכן:

* smaller: מתחילים עם עץ בגובה כפי שציינו **מעלה**.

אז העץ הנמוך ביותר הנו בגובה h-1 או h-2, והעץ הגבוה ביותר יהיה תת העץ השמאלי של השורש (אנו עולים עד השורש לפי האלגוריתם) ולכן יהיה בגובה log(n)-1 או log(n)-2. אז הפרשי הגבהים (בין העץ הגבוה ביותר לעץ הנמוך ביותר) עבור smaller בערך log(n)-h.

* Bigger: מתחילים עם עץ בגובה כפי שציינו **מעלה**.

אז העץ הנמוך ביותר הנו בגובה h-1 או h-2, והעץ הגבוה ביותר יהיה תת העץ הימני של השורש (אנו עולים עד השורש לפי האלגוריתם) ולכן יהיה בגובה log(n)-1 או log(n)-2. אז הפרשי הגבהים (בין העץ הגבוה ביותר לעץ הנמוך ביותר) עבור smaller בערך log(n)-h.

קיבלנו כי הסיבוכיות הכוללת של של פעולות ה-join עבור עבור smaller וbigger הנה O(log(n)-h)

בנוסף הראנו כי כמות פעולות הjoin שנבצע יהיה log(n)-h. לכן העלות הממוצעת של פעולת join בודדת תהיה-

O(log(n)-h)/ (log(n)-h) כלומר שווה ל-O(1).

נבחין כי התוצאות שקיבלנו הנם קבועים – בין 1.3 לבין 2 – ולכן התוצאות מתיישבות עם ניתוח הסיבוכיות התיאורטי לעלות ממוצעת עבור פעולת join בודדת (מספר קבוע ללא תלות ב-n).

תרחיש שני - ניתוח תאורטי של עלות ממוצעת של join עבור split של **האיבר** מקסימלי בתת העץ השמאלי-

ניתן להבחין כי תרחיש זה הנו מקרה פרטי של התרחיש הקודם(איבר מקסימלי בתת העץ השמאלי הוא מקרה פרטי של איבר אקראי בעץ) ולכן העלות הממוצעת של פעולת join בודדת תהיה שווה ל-O(1).

נבחין כי התוצאות שקיבלנו הנם קבועים – בין 1.3 לבין 2.08 – ולכן התוצאות מתיישבות עם ניתוח הסיבוכיות התיאורטי לעלות ממוצעת עבור פעולת join בודדת (מספר קבוע ללא תלות ב-n).

**שאלה 2- סעיף 3**

נעשה ניתוח תאורטי של עלות של join מקסימלי עבור split של **האיבר** מקסימלי בתת העץ השמאלי-

יהי עץ AVL עם n איברים וגובה log(n). יהיה צומת x הצומת **המקסימלית** בתת העץ השמאלי (הולכים מהשורש פעם אחת שמאלה ואז כל הדרך ימינה), ונבצע עליו split.

לפי האלגוריתם, אנחנו מאתחלים שני עצים:

smaller עם התת עץ השמאלי של x - והוא לכל היותר בעל צומת אחת (לכן גובהו לכל היותר שווה ל0 כיוון שמדובר בעץ AVL מאוזן) ו-bigger עם התת עץ ימני של x שהינו עץ ריק וגובהו מינוס 1 (אחרת x לא שווה ל**איבר** מקסימלי בתת העץ השמאלי).

נבחין כי נבצע פעולות join רק על עץ smaller עד שנגיע לשורש, מעצם זה שצומת x הוא הצומת **המקסימלית** בתת העץ השמאלי. כלומר נחבר את כל תתי העצים עד השורש לעץ smaller ונקבל בכל פעם הפרש גבהים נמוך בין שני העצים ולכן גם סיבוכיות join נמוכה.

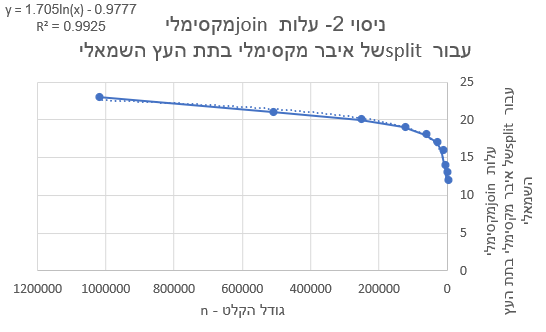
כשנגיע לשורש נבצע פעולה **יחידה** וסופית של join **על עץ Bigger** בינו לבין התת עץ הימני של השורש. עלות פעולת join זו תהיה בעלת העלות המקסימלית (או שווה) מכיוון ועבורה ההפרש בין הגבהים של העצים שעושים עליהם Join הוא הכי גדול (או שווה במקרה של עצים ספציפיים)-

התת עץ הימני של השורש בעל גובה log(n)-1 או log(n)-2 (היות ומדובר העץ AVL) והגובה של עץ Bigger הוא מינוס אחת (הסברנו מעלה מדוע).

לכן העלות המקסימלית של join ((log(n)-1)-(-1)+1 = log(n)+1 או (log(n)-2)-(-1)+1 = log(n)). כלומר O(log(n)).

כלומר בסך הכל קיבלנו כי העלות של join מקסימלי עבור split של **האיבר** מקסימלי בתת העץ השמאלי הנה O(log(n)) .

ניתן לראות כי התוצאות שקיבלנו מתיישבות עם ניתוח הסיבוכוית התיאורטי (O(log(n)) שכן הגרף שמייצג את התוצאות הנו גרף לוגריתמי בגודל הקלט. וניתן לסמוך על הגרף מכיוון שערך ה-R בריבוע קרוב לאחד – כלומר קירוב טוב של התוצאות לנוסחת הגרף.



**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 | 2.974 | 499.5 | 1.053 | 1.068 | 2.426 | 2.509 |
| 2 | 2.986 | 999.5 | 1.040 | 1.051 | 2.424 | 2.414 |
| 3 | 2.989 | 1499.5 | 1.378 | 1.387 | 2.475 | 2.383 |
| 4 | 2.992 | 1999.5 | 1.033 | 1.040 | 2.496 | 2.437 |
| 5 | 2.993 | 2499.5 | 1.647 | 1.653 | 2.462 | 2.472 |
| 6 | 2.994 | 2999.5 | 1.372 | 1.378 | 2.493 | 2.456 |
| 7 | 2.995 | 3499.5 | 1.176 | 1.181 | 2.486 | 2.446 |
| 8 | 2.996 | 3999.5 | 1.029 | 1.033 | 2.483 | 2.448 |
| 9 | 2.996 | 4499.5 | 1.8257 | 1.830 | 2.494 | 2.467 |
| 10 | 2.996 | 4999.5 | 1.643 | 1.647 | 2.492 | 2.460 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 | 7.987 | 499.5 | 8.318 | 2.054 | 9.288 | 18.623 |
| 2 | 8.982 | 999.5 | 9.292 | 2.051 | 10.3145 | 18.251 |
| 3 | 9.639 | 1499.5 | 13.713 | 2.733 | 11.147 | 20.088 |
| 4 | 9.979 | 1999.5 | 10.285 | 2.05 | 11.599 | 21.519 |
| 5 | 10.364 | 2499.5 | 18.064 | 3.278 | 11.827 | 23.36 |
| 6 | 10.637 | 2999.5 | 15.053 | 2.732 | 12.243 | 25.152 |
| 7 | 10.831 | 3499.5 | 12.903 | 2.3416 | 12.638 | 22.770 |
| 8 | 10.977 | 3999.5 | 11.290 | 2.049 | 12.725 | 22.055 |
| 9 | 11.181 | 4499.5 | 21.872 | 3.642 | 13.109 | 25.307 |
| 10 | 11.363 | 4999.5 | 19.684 | 3.277 | 12.823 | 25.947 |

**נענה בנפרד מה היינו מצפות שתהיינה התוצאות לעצים לא מאוזנים:**

עבור הכנסות כסדרה חשבונית-

נסמן ב-n את מספר ההכנסות.

* מספר פעולות האיזון בממוצע

כל הכנסה של איבר חדש תהיה משמאל (נקבל עץ שהוא "שרוך") לכן בכל הכנסה נצטרך לעדכן את הגובה (פעולת איזון) של כל הצמתים שמעליו.

קיבלנו כי בכל הכנסה כמות פעולות האיזון גדלה ב-1. כלומר הסכום הסופי של כל פעולות האיזון שקול לסכום סדרה חשבונית-

נעשה ממוצע ואז נצפה שמספר פעולות האיזון בממוצע יהיה-

אכן התוצאות האמיתיות מסתדרות עם ציפייה זו שכן ניתן לראות כי מספר פעולות האיזון הממוצע הוא בדיוק לפי הנוסחה שציינו מעלה (לכל רצף של הכנסות בכל כמות).

* עומק הצומת המוכנס בממוצע
* **להשלים**

עבור הכנסות כסדרה מאוזנת-

* מספר פעולות האיזון בממוצע

היינו מצפות שהתוצאות יהיו דומות לתוצאות AVL כיוון שסדרה ההכנסות שבנינו "גורמת" לעץ להיות ככל שאפשר מאוזן. בנוסף נצפה שיהיה מספר פעולות איזון קטן מאוד וקבוע לאור סדרת ההכנסות שבנינו.

התוצאות האמיתיות מסתדרות עם ציפייה זו – דומות מאוד לתוצאות של עץ AVL ומספר פעולות האיזון הממוצע נע בין 1 ל-2 (כלומר קבוע ולא תלוי בכמות ההכנסות).

* עומק הצומת המוכנס בממוצע

**להשלים**

עבור הכנסות כסדרה אקראית-

* מספר פעולות האיזון בממוצע

היינו מצפות שיהיו תוצאות לא אחידות – הממוצע יהיה עבור עצים מסוימים מאוד גדול ועבור עצים מסוימים מאוד קטן בגלל האקראיות של המיקום של הכנסת הצמתים (אפשר לקבל עצים מאוד מאוזנים מצד אחד ומהצד שני אפשר לקבל עצים שנראים כמו "שרוך").

אמנם בתוצאות האמיתיות מספר פעולות האיזון הממוצע **לכל רצף** של הכנסות **בכל כמות** נע בין 2 ל-3 כלומר מספר קבוע ומאוד דומה לתוצאות של עץ AVL -בשונה ממה שציפינו.

* עומק הצומת המוכנס בממוצע

**להשלים**

**כעת עבור עצים מאוזנים (AVL):**

* מספר פעולות האיזון בממוצע

ראינו בכיתה כי **לכל רצף של הכנסות** לעץ AVL הסיבוכיות המשוערכת (amortized) עבור איזונים היא קבועה O(1). לכן היינו מצפות שמספר פעולות האיזון הממוצע **לכל רצף** של הכנסות (סדרה חשבונית/מאוזנת/אקראית) **יהיה מספר קבוע ללא תלות בכמות ההכנסות**.

אכן התוצאות האמיתיות מסתדרות עם ציפייה זו שכן ניתן לראות כי מספר פעולות האיזון הממוצע **לכל רצף** של הכנסות **בכל כמות** נע בין 1 ל-3 כלומר מספר קבוע.

* עומק הצומת המוכנס בממוצע

בנוסף לכך, היינו מצפות שעומק הצומת המוכנס בממוצע יהיה בקירוב של log(n) כאשר n מייצג את כמות האיברים שהכנסנו לעץ. אכן התוצאות האמיתיות מסתדרות עם ציפייה זו (ניתן לראות בטבלה).