**מבני נתונים – פרויקט מעשי 1 - קובץ תיעוד**

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.

הוראות הגשה הגשת התרגיל תתבצע באופן אלקטרוני באתר הקורס במודל.

כל זוג יבחר נציג/ה ויעלה רק תחת שם המשתמש של הנציג/ה את קבצי התרגיל )תחת קובץ zip )למודל. על ההגשה לכלול שלושה קבצים

: 1 .קובץ המקור )הרחבה של קובץ השלד שניתן( תחת השם

py.AVLTreeList. 2קובץ טקסט txt.info המכיל את פרטי הזוג: מספר ת"ז, שמות, ושמות משתמש. 3

.מסמך תיעוד חיצוני, המכיל גם את תוצאות המדידות.

את המסמך יש להגיש באחד הפורמטים .pdf או doc, docx :הבאים שמות קובץ התיעוד וקובץ הzip צריכים לכלול את שמות המשתמש האוניברסיטאיים של הזוג המגיש לפי הפורמט zip/doc/pdf.username2\_username1\_AVLTreeList .…/

בתוכן הקבצים יש לציין את שמות המשתמש, תעודות הזהות ושמות המגישים )בכותרת המסמך ובשורת הערה בקובץ המקור(

**פרטים אודות מגישות-**

הגר פייתן

* שם משתמש
* תעודת זהות

גל קריאל

* שם משתמש
* תעודת זהות

**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 |  |  |
| getSuccessor |  |  |
| AVLdelete |  |  |
| is\_left\_child |  |  |
| MinNode |  |  |
| MaxNode |  |  |
| Tree\_rank |  |  |

**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 |  |  |
| TreeSelect -  TreeSelectRec פונקציה פנימית של TreeSelect |  |  |
| retrieve |  |  |
| insert | מבצע הכנסת צומת לפי האלגוריתם מהשיעור. מחזיר כמות הפעולות איזון | 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 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 | ממש כמו RightRotation רק בכיוון שמאל | 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 | מחזיר אורך הרשימה | מחזירים פשוט את שדה גודל 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 |  |  |
| AVL\_join  join פונקציה פנימית של AVL\_join  has\_left\_child - פונקציה פנימית של join  has\_right\_child- פונקציה פנימית של join |  |  |
| 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) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

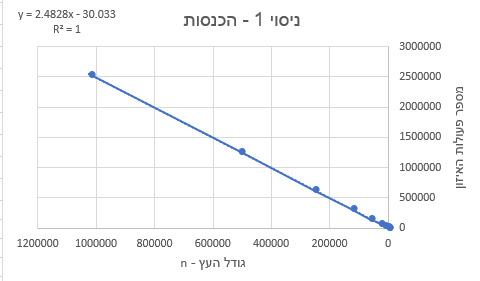
בכל אלה לא נגעתי עדיין – רק מילאתי את הטבלה הכוללת

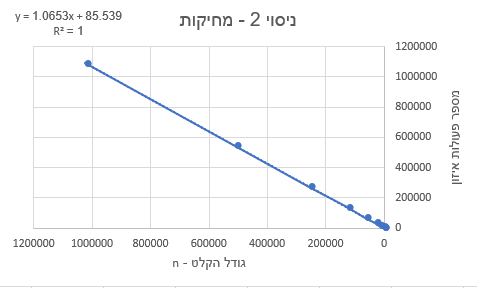
|  |  |  |
| --- | --- | --- |
| שם מתודה | סיבוכיות | הסבר |
| Empty() | O(1) | לא מכיל לולאה/רקורסיה לכן קבוע |
| Retrieve() | logn | קריאה לפונקציית TreeSelectRec() והסיבוכיות נובעת משם. |
| TreeSelectRec()  Tree select –  הפונקציית מעטפת | logn | מקבלת אינדקס ומחזירה את האיבר באינדקס המתאים.  בכל קריאה רקורסיבית יש סיבוכיות זמן קבועה כי רק עושים מספר סופי של פעולות מתמטיות.  לכן הסיבוכיות תלויה בכמות הקריאות הרקורסיביות והן תלויות בגובה העץ, כיוון שעבור כל קלט הולכים לבן הימני/השמאלי.  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 logn. |
| First() | (1) | לא מכיל לולאה/רקורסיה לכן קבוע |
| Last() | (1) | לא מכיל לולאה/רקורסיה לכן קבוע |
| getRoot() | O(1) |  |
| listToArray | O(n) | עברנו על כל צומת פעם אחת בלבד בעזרת פונקציה רקורסיבית דומה ל-tree\_walk ממבוא מורחב. בכל קריה רקורסיבית מתבצע פעולה של append, וזה מתקיים בזמן קבוע. אז סהכ יש n קריאות עם סיבוכיות O(1)- סהכ O(n) |
| Length() | O(1) | לא מכיל לולאה/רקורסיה לכן קבוע |
| Tree\_Rank() | O(log(n)) | מקבלת node ומחזירה את האידנקס של ה-node. הnode בהכרח נמצא בעץ.  בכל רמה בעץ אנחנו מבזבזים זמן קבוע לכן הסיבוכיות תלויה בגובה העץ- log(n) |
| Search() | O(n) | עוברים על כל הצמתים בעץ בפונקציה רקורסיבית פנימית 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). |
| MinNode | O(log(n)) | אותו קוד כמו first, רק מחזיר AVLnode וניתן להתחיל עם כל node שנרצה (שלא וירטואלי) |
| getSuccessor | O(log(n)) | מימשנו כמו בשיעור\*\*\*\* |
| getPredecesor | O(log(n)) | מימשנו כמו בשיעור\*\*\*\* |
| getBF | O(1)) | לא מכיל לולאה/רקורסיה לכן קבוע |
| MaxNode | O(log(n) | אותו קוד כמו last, רק מחזיר AVLnode וניתן להתחיל עם כל node שנרצה (שלא וירטואלי) |
| updatePathMeasurements | **סיבוכיות 0(1)** |  |
| ImplementRotation | O(1) | לא מכיל לולאה/רקורסיה. יש מספר קבוע של קריאות לפונקציות שמתבצעות גם בזמן קבוע ( InsertRotation, LeftRotation) לכן בסך הכל זמן הריצה הוא קבוע |
| LeftRotation | O(1) | לא מכיל לולאה/רקורסיה לכן קבוע . (וכל הפעולות תבצעות בזמן קבוע בפונקציה). |
| RightRotation | O(1) | לא מכיל לולאה/רקורסיה לכן קבוע . (וכל הפעולות תבצעות בזמן קבוע בפונקציה). |
| Insert | O(log(n)) | לא מכיל לולאה/רקורסיה לכן הסיבוכיות היא כסיבוכיות הפעולות/קריאות לפונקציות שאנו משתמשים בהם בפונקציה-  MaxNode()  TreeSelectRec()  getPredecesor()  updatePathMeasurements()  getBF()  ImplementRotation ()  כל הפעולות בפונקציה הן בסיבוכיות לכל היותר ,O(log(n)) ואנו משתמשים בהן מספר קבוע של פעמים לכן בסך הכל נקבל O(log(n)). |
| Is\_left\_child | O(1) | מקבלים צומת עם הורה.  הפונקציה בודקת האם צומת הוא בן שמאלי או ימני- אם שמאלי מחזיר אמת. אם ימני מחזיר שקר.  לא מכיל לולאה/רקורסיה והפעולות שיש הן בזמן קבוע לכן הסיבוכיות היא O(1). |
| delete | O(log(n)) | מחיקת האיבר במקום ה-i ברשימה, אם הוא קיים. הפונקציה מחזירה את מספר (i(delete פעולות האיזון שנדרשו בסך הכל בשלב תיקון העץ על מנת לשמר את תכונת האיזון. אם לא קיימים מספיק איברים ברשימה הפונקציה מחזירה 1.  בתחילת הפונקציה אין לולאה/רקורסיה לכן הסיבוכיות היא כסיבוכיות הפעולות/קריאות לפונקציות שאנו משתמשים בהם בפונקציה-  TreeSelectRec()  getSuccessor()  updatePathMeasurements()  Delete\_style\_balancing  כל הפעולות אלה הן בסיבוכיות לכל היותר .,O(log(n)  לכן בסך הכל נקבל O(log(n)). |
| Delete\_style\_balancing |  | מבצעים את איזון לעץ החל מצומת מסויימת. הולכים מהצומת עד השורש ומאזנים.  יש לולאה אשר עושה log n איטרציות (לכל היותר) בגלל שאנחנו עולים בכל איטרציה להורה מהצומת המסויימת, בכל איטרציה יש פעולות בזמן קבוע/קריאה לפונקציות שהן בזמן קבוע –  ImplementRotation (),getBF()  לכן בסך הכל נקבל O(log(n)). |
| AVLDelete | O(1) | אין רקורסיה/ לולאה. פעולות בזמן קבוע |
| AVLjoin | O(log(n)) | הפונקציה מאחדת שני עצי AVL בהינתן צומת מקשר X.ומחזירה עץ מאוחד.  עושים מספר פעולות בזמן קבוע ואז קוראים לפונקציה פנימית join-  בפונקציה זו יש לולאה שמחפשת את הצומת בה הגובה בין שתי העצים בערך שווים – logn  לאחר מכן עושים מהצומת המקשרת עד לשורש איזונים – logn.  טיפול מקרה קצה: אם רשימה אחת ריקה, אז במקום להפעיל פעולות join רגילות, הפונקציה מבצעת insert באינדקס 0 או list.length. insert לוקת גם log(n) זמן, אז סיבוכיות לא מושפע.  לכן בסה"כ נקבל סיבוכיות של logn |
| Concat | O(||) | הפונקציה מקבלת רשימה. על הפונקציה לשרשר אותה אל סוף הרשימה הנוכחית.  הפונקציה מחזירה את הערך המוחלט של הפרש הגבהים של עצי הAVL שמוזגו.  בתחילת הפונקציה אין לולאה/רקורסיה לכן הסיבוכיות היא כסיבוכיות הפעולות/קריאות לפונקציות שאנו משתמשים בהם בפונקציה-  AVLjoin  לכן בסך הכל נקבל O(log(n)). |
| Split |  |  |
|  |  |  |

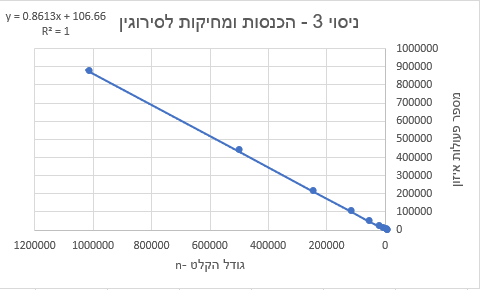
**חלק ניסויי/תיאורטי**

**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 |







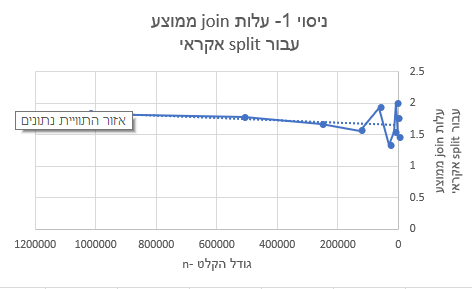
**שאלה 1 - סעיף 2**

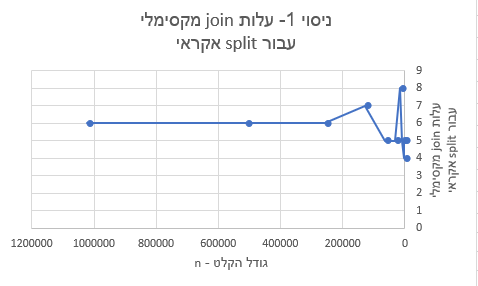
ניתן לראות מהשרטוטים עבור כל אחד מהניסויים כי קיבלנו ביטוי אסימפטוטי **לינארי בגודל הקלט.**

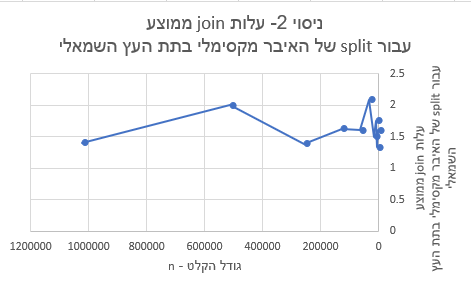
זאת כיוון שערך ה-R בריבוע שווה לאחד – כלומר קירוב טוב של התוצאות לגרף הלינארי.

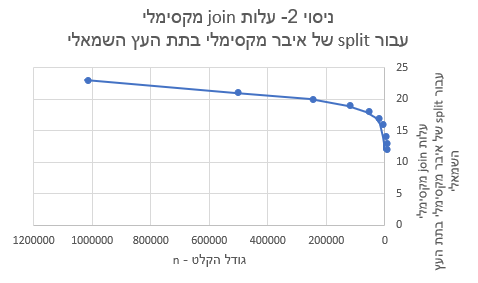
**שאלה 2- סעיף 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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**

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.

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 logn-1 או logn-2. אז הפרשי הגבהים (בין העץ הגבוה ביותר לעץ הנמוך ביותר) עבור smaller בערך logn-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 logn-1 או logn-2. אז הפרשי הגבהים (בין העץ הגבוה ביותר לעץ הנמוך ביותר) עבור smaller בערך logn-h.

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

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

O(logn-h)/ (logn-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**

נתחו באופן תיאורטי את העלות של **join מקסימלי בתרחיש אחד** של split על האיבר המסוים שבחרנו, והסבירו אם התוצאות מתיישבות עם ניתוח הסיבוכיות התאורטי.

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 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) 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, שם נבצע פעולה יחידה וסופית של join על עץ Bigger כאשר עלות פעולה זו תהיה בעלת העלות המקסימלית היות ועבורה ההפרשים בין העצים שעושים עליהם Join הם הכי גדולים-

בן ימני של השורש logn-1 ו-0

לכן העלות המקסימלית של join תהיה logn (logn-1 -1=0).

שניהם בגובה בין **h-1 או h-2** (מהבנייה של עץ AVL).

**שאלה 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| מספר פעולות האיזון בממוצע  מספר סידורי | עץ AVL  סדרה חשבונית (הכנסות בהתחלה) | עץ ללא מנגנון איזון  סדרה חשבונית  (הכנסות בהתחלה) | 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 | 1.0405 | 1.051 | 2.424 | 2.414 |
| 3 | 2.9896666666666665 | 1499.5 | 1.3783333333333334 | 1.387 | 2.4753333333333334 | 2.383 |
| 4 | 2.9925 | 1999.5 | 1.03375 | 1.04025 | 2.49675 | 2.437 |
| 5 | 2.9938 | 2499.5 | 1.647 | 1.6538 | 2.4628 | 2.4728 |
| 6 | 2.9945 | 2999.5 | 1.3725 | 1.3781666666666668 | 2.493333333333333 | 2.4568333333333334 |
| 7 | 2.995285714285714 | 3499.5 | 1.1764285714285714 | 1.1812857142857143 | 2.4867142857142857 | 2.446142857142857 |
| 8 | 2.996 | 3999.5 | 1.029375 | 1.033625 | 2.483875 | 2.448125 |
| 9 | 2.9963333333333333 | 4499.5 | 1.8257777777777777 | 1.8304444444444445 | 2.4945555555555554 | 2.4674444444444443 |
| 10 | 2.9967 | 4999.5 | 1.6432 | 1.6474 | 2.4925 | 2.4604 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| עומק הצומת המוכנס בממוצע  מספר סידורי | עץ AVL  סדרה חשבונית  (הכנסות בהתחלה) | עץ ללא מנגנון איזון  סדרה חשבונית  (הכנסות בהתחלה) | 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 | 9.2925 | 2.0515 | 10.3145 | 18.2515 |
| 3 | 9.639 | 1499.5 | 13.713666666666667 | 2.7333333333333334 | 11.147 | 20.088666666666665 |
| 4 | 9.97925 | 1999.5 | 10.28525 | 2.05 | 11.59975 | 21.51925 |
| 5 | 10.3644 | 2499.5 | 18.0644 | 3.2786 | 11.8276 | 23.36 |
| 6 | 10.637 | 2999.5 | 15.053666666666667 | 2.7321666666666666 | 12.2435 | 25.152333333333335 |
| 7 | 10.831714285714286 | 3499.5 | 12.903142857142857 | 2.3418571428571426 | 12.638428571428571 | 22.77042857142857 |
| 8 | 10.97775 | 3999.5 | 11.29025 | 2.049125 | 12.725625 | 22.05575 |
| 9 | 11.181222222222223 | 4499.5 | 21.872 | 3.642 | 13.109111111111112 | 25.307111111111112 |
| 10 | 11.3631 | 4999.5 | 19.6848 | 3.2778 | 12.8231 | 25.9474 |

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