**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, ומספר איזונים. מעדכנת את שדות גובה וגודל לפי הגובה והגודל של הבנים שלו. בנוסף בודקת האם הגובה שלו השתנה ואם כן מעדכנת את HeightUpdate להיות אמת.  הפונקציה **מחזירה** את מספר האיזונים (ועוד אחד במקרה שהגובה שונה). | 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). | 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 |  |  |
| ImplementRotation |  |  |
| RightRotation |  |  |
| LeftRotation |  |  |
| delete |  |  |
| delete\_style\_balancing |  |  |
| first |  |  |
| last |  |  |
| listToArray  listtoArray\_rec - פונקציה פנימית של listToArray |  |  |
| length |  |  |
| split |  |  |
| 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 |  |  |
| getRoot |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

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 |  |  |
|  |  |  |

updateMeasurements

**לעדכן שזה סיבוכיות 0(1)**

**Fix size rec – לוג n – לעדכן שמתמשות בה גם בדליט וג באינסרט**

**מתודות של העצי -AVL ניתוח סיבוכיות**