***מסמך תיעוד- ליהוא צור ויובל רמות***

פרטי המגישים:

ליהוא צור- מספר תז: 322216151, שם משתמש אוניברסיטאי: lihuzur

יובל רמות-

***המחלקה AVLNode:***

***שדות:***

private Integer key- המפתח של הצומת

private String info- המידע השמור בצומת

private IAVLNode left- מצביע לבן השמאלי

private IAVLNode right- מצביע לבן הימני

private IAVLNode parent- מצביע להורה הישיר של הצומת

private int height- גובה הצומת

private int size- גודל תת העץ ששורשו בצומת

***בנאים:***

public AVLNode()- בנאי שבונה צומת וירטואלי- צומת ללא מפתח, מידע או בנים. הבנאי מאתחל את שדות האבא, הבנים, המפתח והמידע לnull, את שדה הגובה למינוס אחד ואת שדה הגודל ל0. הסיבוכיות היא O(1) כי כל הפעולות שבוצעו הן בזמן קבוע.

public AVLNode(String info, int key)- בנאי שבונה צומת חדש מנותק. הבנאי מאתחל את שדות האבא והבנים לnull, את שדה המפתח לkey, את שדה המידע לinfo, את שדה הגובה ל0 ואת שדה הגודל ל1. הסיבוכיות היא O(1) כי כל הפעולות שבוצעו הן בזמן קבוע.

***פונקציות:***

public int getKey()- הפונקציה מחזירה את שדה הkey השמור בצומת אם הצומת אינו וירטואלי, ומינוס 1 עבור צומת וירטואלי. getKey נעזרת בפונקציה isRealNode כדי לבדוק את התנאי כאמור. הסיבוכיות היא O(1) כי כל הפעולות שבוצעו הן בזמן קבוע.

public String getValue()- הפונקציה מחזירה את שדה הinfo השמור בצומת אם הצומת אינו וירטואלי, ו null עבור צומת וירטואלי. getValue נעזרת בפונקציה isRealNode כדי לבדוק את התנאי כאמור. הסיבוכיות היא O(1) כי כל הפעולות שבוצעו הן בזמן קבוע.

public void setLeft(IAVLNode node)- הפונקציה משנה את השדה left של הצומת להיות צומת הקלט. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public IAVLNode getLeft()- הפונקציה מחזירה את השדה left של הצומת. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public void setRight(IAVLNode node)- הפונקציה משנה את השדה right של הצומת להיות צומת הקלט. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public IAVLNode getRight()- הפונקציה מחזירה את השדה right של הצומת. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public void setParent(IAVLNode node)- הפונקציה משנה את השדה parent של הצומת להיות צומת הקלט. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public IAVLNode getParent()- הפונקציה מחזירה את השדה parent של הצומת. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public boolean isRealNode()- הפונקציה מחזירה true אם הצומת איננו וירטואלי, ואחרת, false. הבדיקה מתבצעת על ידי בדיקת המפתח- אם הוא null, הצומת וירטואלי, ואחרת, הצומת אמיתי. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public int getHeight()- הפונקציה מחזירה את שדה הגובה של הצומת, ואם הצומת הוא וירטואלי, מוחזר מינוס 1. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public void setHeight(int height)- הפונקציה מעדכנת את שדה הגובה של הצומת להיות height. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public int getSize()- הפונקציה מחזיקה את שדה הגודל של הצומת, ואם הצומת וירטואלי, מוחזר 0. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public void setSize(int size)- הפונקציה מעדכנת את שדה הגודל של הצומת להיות size. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

***המחלקה AVLTree-***

***שדות-***

private static final IAVLNode virtual\_leaf = new AVLNode()- הצומת הוירטואלי שישמש כדי למלא את הבנים החסרים של צמתים אונאריים ושל עלים.

private IAVLNode root- השורש של העץ

private IAVLNode min- הצומת בעל המפתח המינימלי בעץ

private IAVLNode max- הצומת בעל המפתח המקסימלי בעץ

***בנאים-***

public AVLTree()- הבנאי מייצר עץ חדש ומאתחל את השורש להיות עלה וירטואלי, ואת שדות המינימום והמקסימום להיות null. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public boolean empty()- הפונקציה מחזירה true אם העץ ריק ו false אחרת. הבדיקה המתבצעת נעזרת בפונקציה isRealNode ובודקת האם השורש הוא עלה וירטואלי או לא, ומחזירה true או false בהתאמה, כאמור מעלה. הסיבוכיות היא O(1) שכן הפעולה קורית בזמן קבוע.

public String search(int k)- הפונקציה מחזירה את שדה הinfo של הצומת בעץ בעל המפתח k, ואם לא קיים כזה צומת, מוחזר null. הפונקציה מפעילה את goTo עם הערך k ומחזירה את שדה הinfo של הצומת שהוחזר מ goTo(k) . הסיבוכיות היא O(log(n)) מכיוון שהסיבוכיות של goTo היא O(log(n)), כפי שיוסבר בהמשך.

public IAVLNode goTo(int k)- הפונקציה מקבלת מספר k ומחפשת את הצומת בעץ בעל המפתח k. אם הצומת נמצא בעץ, הוא מוחזר כפלט, ואחרת, יוחזר צומת וירטואלי. הפונקציה מתחילה בחיפוש החל משורש העץ, ורצה בלולאה כל עוד הצומת הנוכחי אינו וירטואלי: אם הצומת הנוכחי בעל מפתח k, מצאנו ולכן נחזיר את הצומת. אחרת, אם המפתח של הצומת הנוכחי קטן מk, נמשיך לבדוק באיטרציה הבאה את הבן הימני של הצומת. אחרת, בהכרח המפתח של הצומת הנוכחי גדול מk, ונמשיך לבדוק באיטרציה הבאה את הבן השמאלי של הצומת. לאחר סיום הלולאה, כלומר לא הוחזר אף צומת כפלט, אז הצומת לא נמצא בעץ ויוחזר עלה וירטואלי. הסיבוכיות היא O(log(n)) כפי שהראינו בכיתה, בכל איטרציה אנו מוצאים את הצומת המבוקש, או לחלופין, יורדים ב1 בגובה, ולכן לאחר לכל היותר גובה העץ איטרציות, כלומר log(n), הפונקציה תסיים את פעולתה. בכל איטרציה מבוצעת עבודה קבועה ולכן סה"כ הסיבוכיות היא O(log(n)).

public int insert(int k, String i)-

public int delete(int k)-

public IAVLNode predecessor(IAVLNode node)- הפונקציה מחזירה את הצומת בעץ עם המפתח המקסימלי הקטן מהמפתח של node, אם קיים כזה, ואחרת מוחזר עלה וירטואלי. ראשית, אם לצומת node יש בן שמאלי, יורדים אליו, והחל ממנו רצים ימינה בעץ עד לצומת האחרון שאינו עלה וירטואלי, ואת הצומת הזה מחזירים כפלט. אחרת, לצומת אין בן שמאלי. אם הצומת אינו השורש, כלומר יש לו parent שאינו null, מתחילים לרוץ במעלה העץ החל מהאבא של הצומת node, וכל עוד הצומת הנוכחי הוא הבן השמאלי של אביו, ובתוך הלולאה מתבצע קידום של הבן והאב כל אחד לאביו, ומתבצעת בדיקה (לאחר הקידום)- אם הגענו לאב שהוא null, אז אין predecessor ומוחזר צומת וירטואלי. אם הקוד הגיע לסוף הלולאה, מוחזר הצומת הנוכחי בשדה האב (העליון מבין השניים שמתקדמים בלולאה) כפלט. אם הפונקציה לא הגיעה לסיום פעולתה באף אחד מהמצבים הקודמים, נסיק שהצומת שלנו הוא השורש ושהוא המינימלי בעץ ועל כן נחזיר צומת וירטואלי. סיבוכיות הפעולה היא O(log(n)) כפי שראינו בהרצאה. ב-2 הלולאות המתוארות בפונקציה, רצים במעלה או במורד העץ כך שבכל איטרציה גובה הצומת גדל או קטן ב-1 בהתאמה (בכל לולאה תמיד גדל או תמיד קטן..) ולכן יבוצעו לכל היותר גובה העץ איטרציות בכל לולאה. שאר הפעולות הן בזמן קבוע. לכן נקבל O(log(n)).

public IAVLNode successor(IAVLNode node)- הפונקציה מחזיקה את הצומת בעץ עם המפתח המינימלי הגדול יותר מהמפתח של node. הפונקציה פועלת בדומה לsuccessor , והופכת את הצדדים ב-2 הלולאות. ראשית, אם לצומת node יש בן ימני, יורדים אליו ולאחר מכן רצים שמאלה בעץ עד לצומת האחרון שאינו עלה וירטואלי, ואותו מחזירים כפלט. אחרת, אין לצומת בן ימני. אם הצומץ אינו השורש, כלומר יש לו parent שאינו null, מתחילים לרוץ במעלה העץ החל מהאבא של הצומת node, וכל עוד הצומת הנוכחי הוא הבן הימני של אביו, ובתוך הלולאה מתבצע קידום שך הבן והאב כל אחד לאביו, ומתבצעת בדיקה (לאחר הקידום)- אם הגענו לאב שהוא null, אז אין successor ומוחזר צומת וירטואלי. אם הלולאה הסתיימה, מוחזר הצומת הנוכחי בשדה האב (העליון מבין השניים) כפלט. אם הפונקציה לא הגיעה לסיום פעולתה באף שלב מהקודמים, אז הצומת node הוא בעצם השורש, ובנוסף הוא הצומת המקסימלי בעץ ולכן מוחזר צומת וירטואלי. סיבוכיות הפעולה היא בדומה לpredecessor, O(log(n)) מכיוון שמאותם שיקולים, ב-2 הלולאות מתבצעות פעולות בעלות קבועה בכל איטרציה וכל לולאה מתבצעת לכל היותר גובה העץ פעמים, וכל שאר הפעולות בפונקציה גם הן בזמן קבוע. לכן נקבל O(log(n)) באופן דומה.

public String min()- הפונקציה מחזירה את שדה ה info של הצומת בעץ בעל המפתח המינימלי, אשר שמור בשדה min. אם העץ ריק, min הוא null והפונקציה מחזירה null. הפונקציה נעזרת בפונקציה empty כדי לבדוק אם העץ ריק, ואם הוא לא ריק, מחזירה את שדה הinfo של הצומת שבשדה הmin. סיבוכיות הפעולה היא O(log(n)) מכיוון שבוצעו רק פעולות בזמן קבוע.

public String max()-הפונקציה מחזירה את שדה ה info של הצומת בעץ בעל המפתח המקסימלי, אשר שמור בשדה max. אם העץ ריק, max הוא null והפונקציה מחזירה null. הפונקציה נעזרת בפונקציה empty כדי לבדוק אם העץ ריק, ואם הוא לא ריק, מחזירה את שדה הinfo של הצומת שבשדה הmax. סיבוכיות הפעולה היא O(log(n)) מכיוון שבוצעו רק פעולות בזמן קבוע.

public int[] keysToArray()- הפונקציה מחזירה מערך ממוין של כל הצמתים בעץ. היא עוטפת פונקציית עזר keysToArrayHelp ומפעילה אותה עם גודל העץ (הsize של השורש), עם צומת השורש, ועם האינדקס ההתחלתי 0. הסיבוכיות היא O(n), שכן זו הסיבוכיות של פונקציית העזר ושאר הפעולות שביצענו הן בזמן קבוע.

private int keysToArrayHelp(int[] arr, IAVLNode node, int index)- הפונקציה פועלת רקורסיבית ולבסוף המערך arr מלא בכל איברי העץ ממויינים מהקטן לגדול. הפונקציה בכל פעם בודקת אם הצומת הנוכחי הוא עלה וירטואלי, ואם כן, מחזירה את אותו האינדקס שכן לא הוספנו איברים למערך כלומר לא התקדמנו במילוי. אחרת, הצומת אמיתי, ונפעיל את הפונקציה רקורסיבית עם הבן השמאלי של הצומת ועם אותו האינדקס ונשמור את התוצאה במשתנה index. לאחר מכן נשים במקום index במערך את המפתח של הצומת הנוכחי, ולבסוף נפעיל את הפונקציה שוב רקורסיבית, והפעם עם הבן הימני של הצומת הנוכחי ועם index+1. סיבוכיות הפעולה היא O(n), שכן יש n צמתים בעץ ואנו עוברים על כל צומת בדיוק פעם אחת, בעת הכנסתו לעץ, ושאר הפעולות הן בזמן קבוע.

public String infoToArray()- בדיוק באותו אופן כמו keysToArray, הפונקציה קוראת לפונקציית עזר משלה infoToArrayHelp עם שורש העץ, האינדקס 0 ועם מערך ריק בגודל מספר הצמתים בעץ. הסיבוכיות היא O(n) באותו אופן.

private int infoToArrayHelp(String[] arr, IAVLNode node, int index)- בדיוק באותו אופן כמו keysToArrayHelp, מכיוון שהמיון הוא עדיין על ידי המפתחות לפי דרישות השאלה, וההבדל היחיד הוא שבשלב ההכנסה של המפתח של הצומת הנוכחי, נכניס במקום את הinfo שלו. הסיבוכיות היא O(n) באותו האופן.

public int size()- מחזירה את גודל העץ, על ידי החזרת השדה size של השורש. הסיבוכיות היא O(1).

public IAVLNode getRoot()- מחזירה את שורש העץ, על ידי גישה לשדה root. הסיבוכיות היא O(1).

public AVLTree[] split(int x)- הפונקציה פועלת כמו בהרצאה ומפצלת את העץ לשני עצים חדשים לפי x כאשר עץ אחד מכיל את כל הצמתים עם המפתחות שקטנים מx והשני את כל הצמתים עם המפתחות הגדולים מx. הפונקציה מאתחלת מערך ריק של 2 עצים ואת 2 העצים smaller,bigger כעצים ריקים. שומרים במשתנה גם את הsuccessor והpredecessor של x. מפעילים את goTo על x כדי למצוא את הצומת עם המפתח x. נשמור מצביע node, ההורה של הצומת של x ובוליאני right שמתחיל מfalse שיקבע בכל פעם אם באנו מימין או משמאל. בודקים אם הצומת node הוא null ואם כן השדה right נהיה true אם הצומת של איקס היא הבן הימני של node ואחרת false. בודקים את הבנים של הצומת של x (בנפרד) ואם הם קיימים מנתקים אותם מההורה שלהם ומכניסים אותם לעצים smaller ו bigger בהתאמה (בן שמאלי יהיה שורש של smaller ולהפך). כעת מפעילים את complete\_disconnect על הצומת שמכיל את x כלומר מנתקים אותו מההורה ומהבנים שלו (ומנתקים אותם ממנו). כעת בלולאה כל עוד node אינו null שומרים בtemp את האבא של node, ואם הבוליאני right הוא true, מפעילים את Update\_side על node כדי שright יהיה מעודכן עבור האיטרציה הבאה. כעת מנתקים את הבן השמאלי של node מאביו בעזרת disconnect\_from\_parent. מאתחלים עץ חדש ריק ושמים את הבן השמאלי להיות השורש. כעת, מפעילים complete\_disconnect על node, ומאחדים את smaller עם העץ החדש בעזרת הצומת node. אם right היה false, מבצעים בדיוק אותן פעולות, רק שמה שעשינו לבן השמאלי עושים בדיוק לבן הימני במקום, ואת העץ החדש מאחדים עם bigger (עדיין בעזרת node). בכל מקרה, בלי קשר לערך של right, לפני סוף האיטרציה של הלולאה node מקבל את הערך של temp כלומר מתקדם להיות האבא של עצמו. לאחר הלולאה נטפל במינימום ובמקסימום של העץ. אם smaller אינו ריק, המינימום שלו נהיה שדה המינימום של העץ המקורי והמקסימום שלו נהיה הpredecessor של הצומת המקורי שהכיל את x (שמרנו את ה predecessor בהתחלה). באותו אופן, אם bigger לא ריק אז שדה המקסימום שלו נהיה שדה המקסימום של העץ המקורי, ושדה המינימום שלו נהיה הsuccessor של הצומת שמכיל את x. לבסוף, נשים במקום הראשון במערך את smaller, במקום השני את bigger ונחזיר את המערך. סיבוכיות הפעולה היא O(log(n)) כפי שהסברנו בהרצאה, מכיוון שבתחילה מצאנו את הצומת בעלות O(log(n)), ואחר כך טיפלנו בבנים שלו ואז עלינו בעץ עד למעלה כאשר בדרך ביצענו פעולות בזמן קבוע, ולכן עלינו לכל היותר O(log(n)) פעמים כגובה העץ. נקבל סה"כ סיבוכיות O(log(n)).

private Boolean update\_side(IAVLNode node)- הפונקציה מחזירה true אם הצומת node הוא הבן הימני של אביו וfalse אם היא הבן השמאלי של אביו או אם אין לצומת אבא. הסיבוכיות היא O(1) שהרי כל הבדיקות הללו הן בזמן קבוע.

private void disconnect\_from\_parent(IAVLNode node)- הפונקציה מנתקת את node מאביו ואת אביו ממנו. במקרה וnode הוא הבן השמאלי של אביו, שדה הleft של האבא נהיה עלה וירטואלי ולהפך אם node הבן הימני. לבסוף בכל מקרה האבא של node מאותחל לnull. הסיבוכיות היא O(1) כי ביצענו פעולות בזמן קבוע.

private void complete\_disconnect()- הפונקציה מנתקת לחלוטין את הצומת מאביו ומבניו ואותם ממנו. ראשית מפעילים את הפונקציה הקודמת כדי לנתק מהאבא, ואחר כך משנים את שדות left,right של node לnull ומשנים את גובה הצומת ל0 ואת הsize ל1.