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

פרטי המגישים:

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

יובל רמות- מספר תז: 208115840, שם משתמש אוניברסיטאי: yuvalramot

***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.

***פונקציות:***

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.

private int to\_join(IAVLNode x, AVLTree t)- הפונקציה היא פונקציית עזר עבור join אשר מתבצעת אחרי שבjoin מטפלים בכך ששדות המינימום והמקסימום לא בהכרח נכונים בעץ המגיע לjoin (מצב שקורה למשל כאשר קוראים לה מתוך split). הפונקציה מחזירה את סיבוכיות הפעולה כפי שהראנו בהרצאה. אם שני העצים ריקים, שמים את x להיות שורש העץ ומעדכנים לו את הגובה וה size ואז מעדכנים את שני בניו להיות עלים וירטואלים, ואז מחזירים 0. אם לא נכנסנו לשם, שומרים במשתנים את הגובה הנמוך מבין הגובה של שני העצים ושומרים את שני העצים בשדות shorter ו higher בהתאם לגובה. שומרים משתנה res שמוגדר להיות גובה העץ הגבוה פחות קיי ועוד 1. כעת, אם שורש העץ הגבוה גדול מהמפתח של x, שומרים שני צמתים שמאותחלים להיות שורשי 2 העצים. אם res=1 או res=2, מאתחלים את x להיות אבא שלהם ואות כבנים שלו השורש הנמוך כבן שמאלי והגבוה כימני . מאתחלים את שורש העץ this להיות x, ומשנים את אבא של x לnull. מסדרים את הגובה והsize של x ואז מחזירים את המשתנה res ששמרנו. אחרת res אינו 1 או 2 ומאתחלים את this.root לשורש העץ הגבוה. שומרים משתנה parent שמתחיל מnull ורצים בלולאה כל עוד שורש העץ הגבוה אינו וירטואלי וגם גובה הצומת גדול ממש מk. בפנים הלולאה parent נהיה הצומת הנוכחי והצומת הנוכחי מקודם לבן השמאלי של עצמו. אחרי הלולאה שמים לx כבן ימני את הצומת שיצא מהלולאה (שהתחיל משורש העץ הגבוה וקודם שמאלה) וכבן שמאלי את שורש העץ הקטן. מעדכנים את ההורים שלהם בהתאמה לx ואז למשתנה parent שיצא מהלולאה שמים את x כבן שמאלי ואת אביו שלx להיות parent. אחרת, כלומר מפתח השורש של העץ הגבוה אינו גדול ממש מהמפתח של x. אם הוא קטן ממש ממנו, נבצע אותו דבר כמו בבלוק של אם השורש גדול ממפתחו של k עם השינויים: במקרה של res=1 או res=2 נחליף בצדדים את השורשים שנהיים בהתחלה הבנים של x. בהמשך, נתקדם בלולאה ימינה במקום שמאלה. אחרי הלולאה שוב מחליפים את הצדדים של שני הצמתים כבנים של x, וx נהיה הבן הימני של parent במקום שמאלי. בסוף הפונקציה מעדכנים את הגובה והsize של x לפי בניו וקוראים לrebalance כדי לאזן את העץ לאחר הjoin. מוחזר בסוף res מהפונקציה כפי שמבוקש בתיעוד של join. סיבוכיות הפעולה היא O(log(n)). אם שני העצים ריקים נסיים בזמן קבוע. אם נכנסנו למקרה res=1 או res=2 נבצע גם כן פעולות בזמן קבוע ונסיים. ואחרת נרוץ במורד העץ הנמוך ימינה או שמאלה עד שנגיע לצומת בדרגה נמוכה או שווה לk וזה עולה לכל היותר O(log(n)) לפי גובה העץ ולאחר מכן שוב נבצע פעולות בזמן קבוע. לבסוף נקראת rebalance שפועלת בסיבוכיות O(log(n)) גם כן ולכן סה"כ הסיבוכיות היא O(log(n)).

public int join(IAVLNode x, AVLTree t)- הפונקציה מטפלת בכך שהעצים שהיא פועלת עליהם עלולים להיות עם שדות מינימום ומקסימום לא מאותחלים ואז קוראת לפונקציה הקודמת כדי לבצע את האיחוד עצמו ומחזירה את מה שמוחזר מפונקציית העזר כפלט. הפונקציה מאפסת את x, כלומר מנתקת אותו מבניו ומההורה ומשנה את הגובה ל0 והsize ל1. אם שני העצים ריקים אז מאתחלים את המקסימום והמינימום של this ל x. אחרת, אם this ריק מאתחלים את המקסימום של x להיות הצומת עם המפתח המקסימלי בין המקסימום של t וx, ובאופן דומה מאתחלים את המינימום של this להיות הצומת עם המפתח המינימלי בין המינימום של t וx. אחרת, אם t ריק מבצעים בדיוק אותו דבר כמו בתנאי הקודם רק שבכל פעם משווים את המפתח של x למינימום ולמקסימום של this בהתאמה. אחרת, שני העצים לא ריקים ומאתחלים את המקסימום של this להיות המקסימום מבין 2 צמתי המקסימום של העצים ואת המינימום להיות המינימלי מבין צמתי המינימום. כעץ העץ מוכנים לביצוע to\_join ולכן קוראים לה עם x וt ומחזירים את מה שמוחזר ממנה. הסיבוכיות היא O(log(n)) מכיוון שלפני התנאים ובמהלך התנאים ביצענו שינויי שדות והצבעות בזמן קבוע ואחר כך קראנו לפונקציית העזר to\_join שלה סיבוכיות O(log(n)).

private int rebalance(IAVLNode node)- הפונקציה מאזנת את העץ לאחר פעולות insert ו join. האיזון מתבצע כפי שנלמד בהרצאה ולוקח בחשבון את המקרה המיוחד שעלול לקרות בjoin ולא בinsert. תחילה, שדה count מאותחל ל-0 והוא ייצג את כמות צעדי האיזון. רצים בלולאה כל עוד צומת הקלט אינו null. שומרים מצביעים לשני בניו של הצומת ושומרים במשתנים את 3 הגבהים של הצומת ושני בניו. אם גובה הצומת שווה לגובה של לפחות אחד מבניו: אם ההפרש בין גובה הצומת לגובה לפחות אחד הבנים הוא 1, מעדכנים את הגובה והsize של הצומת ומעלים ב1 את הcount. אחרת, ההפרש בגבהים בין הצומת לבין לפחות אחד הבנים הוא 2. אם גובה הצומת שווה לגובה הבן השמאלי מביטים בבנים של הבן השמאלי: אם ההפרש בגבהים בין הבן השמאלי של הצומת שלנו לבין שני הבנים שלו הוא 1, מבצעים סיבוב ימינה סביב הקשת של הצומת הנוכחי והבן השמאלי ומעלים ב2 את הcount. אחרת אם ההפרש בגובה בין הבן השמאלי לבנו השמאלי הוא 1 מסובבים ימינה סביב הקשת של הצומת ובנו השמאלי ומעלים ב2 את הcount. אחרת, מבצעים סיבוב שמאלה-ימינה סביב הצומת, בנו השמאלי, והבן הימני של הבן השמאלי ומעלים ב5 את הcount. אחרת, גובה הצומת לא שווה לגובה של הבן השמאלי ונבצע אותן פעולות באופן סימטרי בצד הפוך: מביטים בבנים של הבן הימני: אם ההפרש בגבהים בין הבן הימני של הצומת שלנו לבין שני הבנים שלו הוא 1, מבצעים סיבוב שמאלה סביב הקשת של הצומת הנוכחי והבן הימני ומעלים ב2 את הcount. אחרת אם ההפרש בגובה בין הבן הימני לבנו הימני הוא 1 מסובבים שמאלה סביב הקשת של הצומת ובנו הימני ומעלים ב2 את הcount. אחרת, מבצעים סיבוב ימינה-שמאלה סביב הצומת, בנו הימני, והבן השמאלי של הבן הימני ומעלים ב5 את הcount. לאחר כל ההתניות הללו, בכל מקרה (בכל איטרציה של הלולאה) מתקנים את הגובה והsize של הצומת הנוכחי ולבסוף מקדמים את הצומת הנוכחי למעלה להיות אבא שלו. זה סוף הלולאה. לאחר סוף הלולאה סיימנו את האיזון של העץ ונחזיר את count שהצטבר. סיבוכיות הפעולה היא O(log(n)) מכיוון שבכל איטרציה של הלולאה ביצענו שינויי שדות ומצביעים אשר כולם בזמן קבוע והתקדמנו בכל פעם למעלה בעץ ולכן בהתאם לגובה העץ יהיו O(log(n)) איטרציות כאלו. לכן נקבל שסיבוכיות האיזון היא O(log(n)).

private boolean right\_rotation(IAVLNode parent, IAVLNode child)- הפונקציה מבצעת סיבוב ימינה של העץ לפי הקשת בין 2 הצמתים בקלט. הפונקציה מוודאת תחילה שאכן הקלט תקין, כלומר ששני הצמתים אינם null ושchild הוא בן שמאלי של parent ואם אחד התנאים לא מתקיים מחזירים false ולא מבצעים דבר. מסמנים בtemp את הבן הימני של child ובprev\_parent את אביו של parent. מבצעים שינויי מצביעים: אבא של child נהיה prev\_parent , ההורה של parent נהיה child , בנו השמאלי של parent נהיה temp , בנו הימני של child נהיה parent, ההורה של temp נהיה parent. מעדכנים את הגובה והsize של parent,child. כעת, אם prev\_parent אינו null לוקחים דגל בוליאני להיות true אמ"מ parent הוא הבן הימני של prev\_parent. אם כך הדבר, שמים כבן ימני של prev\_parent את child ואחרת (אם הבוליאני false) שמים אותו כבן שמאלי. מעדכנים את הגובה והsize של parent. אחרת, prev\_parent הוא null ונהפוך את child להיות שורש העץ. אם השורש שווה לparent אז הופכים את השורש לchild. לבסוף מחזירים true כי ביצענו בהצלחה את הסיבוב. סיבוכיות הפעולה היא O(1) מכיוון שבמהלך כל זמן ריצת הפונקציה ביצענו שינויי שדות וחילופי הצבעות אשר כולם בזמן קבוע.

private boolean left\_rotation(IAVLNode parent, IAVLNode child)- הפונקציה מבצעת סיבוב שמאלה של העץ לפי הקשת בין 2 הצמתים בקלט. הפונקציה מוודאת תחילה שאכן הקלט תקין, כלומר ששני הצמתים אינם null ושchild הוא בן ימני של parent ואם אחד התנאים לא מתקיים מחזירים false ולא מבצעים דבר. מסמנים בtemp את הבן השמאלי של child ובprev\_parent את אביו של parent. מבצעים שינויי מצביעים: אבא של child נהיה prev\_parent , ההורה של parent נהיה child , בנו השמאלי של parent נהיה temp , בנו השמאלי של child נהיה parent, ההורה של temp נהיה parent. מעדכנים את הגובה והsize של parent,child. כעת, אם prev\_parent אינו null לוקחים דגל בוליאני להיות true אמ"מ parent הוא הבן הימני של prev\_parent. אם כך הדבר, שמים כבן ימני של prev\_parent את child ואחרת (אם הבוליאני false) שמים אותו כבן שמאלי. מעדכנים את הגובה והsize של parent. אחרת, prev\_parent הוא null ונהפוך את child להיות שורש העץ. אם השורש שווה לparent אז הופכים את השורש לchild. לבסוף מחזירים true כי ביצענו בהצלחה את הסיבוב. סיבוכיות הפעולה היא O(1) מכיוון שבמהלך כל זמן ריצת הפונקציה ביצענו שינויי שדות וחילופי הצבעות אשר כולם בזמן קבוע.

private boolean left\_right\_rotation(IAVLNode parent, IAVLNode child, IAVLNode grand\_child)- מסמנים בדגל בוליאני true אמ"מ הצלחנו לבצע סיבוב שמאלה לפי child ו grand\_child (ומבצעים את הסיבוב). אם הדגל false, מחזירים false . אחרת, הצלחנו לסובב שמאלה, ומעדכנים את הגובה והsize של child,grand\_child. כעת, מנסים לבצע סיבוב ימינה לפי parent,grand\_child. אם חזר false, מסובבים חזרה ימינה סביב grand\_child,child כדי לבטל את השינויים ומוחזר false (במצב כזה הפונקציה בעצם לא עשתה דבר). אחרת, הצלחנו לבצע את הסיבוב הראשוני ימינה, ונעדכן את הגובה והsize של parent,grand\_child ונחזיר true. סיבוכיות הפעולה היא O(1) מכיוון שהיא ביצעה שינויי שדות והצבעות והשתמשה בפונקציות סיבוב ימינה ושמאלה שהן בO(1) גם כן.

private boolean right\_left \_rotation(IAVLNode parent, IAVLNode child, IAVLNode grand\_child)- מסמנים בדגל בוליאני true אמ"מ הצלחנו לבצע סיבוב ימינה לפי child ו grand\_child (ומבצעים את הסיבוב). אם הדגל false, מחזירים false . אחרת, הצלחנו לסובב ימינה, ומעדכנים את הגובה והsize של child,grand\_child. כעת, מנסים לבצע סיבוב שמאלה לפי parent,grand\_child. אם חזר false, מסובבים חזרה שמאלה סביב grand\_child,child כדי לבטל את השינויים ומוחזר false (במצב כזה הפונקציה בעצם לא עשתה דבר). אחרת, הצלחנו לבצע את הסיבוב הראשוני שמאלה, ונעדכן את הגובה והsize של parent,grand\_child ונחזיר true. סיבוכיות הפעולה היא O(1) מכיוון שהיא ביצעה שינויי שדות והצבעות והשתמשה בפונקציות סיבוב ימינה ושמאלה שהן בO(1) גם כן.

public static void hs\_modifier(IAVLNode node)- מעדכנת את הגובה והsize של צומת הקלט לפי בניו הישירים. בכל פעם שכתבנו שאנו מעדכנים את הגובה והsize של צומת בעצם השתמשנו בפונקציה הזו, אשר פועלת בO(1) ולכן אף פעם לא פגעה בסיבוכיות של הפעולות שהשתמשו בה. אם הצומת הוא עלה וירטואלי לא נעשה דבר. אחרת, משנים את שדה הגובה להיות מקסימום הגבהים בין הבן השמאלי לבן הימני , ומוסיפים לו 1, ומשנים את שדה הsize להיות סכום שדות הsize של שני הבנים, ומוסיפים עוד 1. הפונקציה מבצעת גישה לשדות בO(1) ולכן זו הסיבוכיות שלה.