**חלק ניסויי/תיאורטי**

**שאלה 1:**

|  |  |  |  |
| --- | --- | --- | --- |
| מספר סידורי i | ניסוי 1 – הכנסות | ניסוי 2 - מחיקות | ניסוי 3 - הכנסות ומחיקות לסירוגין |
| 1 | 4915 | 2214 | 1757 |
| 2 | 9873 | 4472 | 3431 |
| 3 | 19836 | 8763 | 6972 |
| 4 | 40030 | 17611 | 14206 |
| 5 | 79396 | 35429 | 27695 |
| 6 | 158722 | 70467 | 56083 |
| 7 | 317445 | 141057 | 111753 |
| 8 | 635488 | 281067 | 223815 |
| 9 | 1271307 | 564312 | 447557 |
| 10 | 2544002 | 1128244 | 896808 |
| O(n) | O(2.48n) = O(n) | O(1.1n) = O(n) | O(0.9n) = O(n) |

**שאלה 2:**

1. תעדו בטבלה שלהלן את העלות הממוצעת של פעולות ה-join ואת העלות של פעולת ה-join היקרה ביותר.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 2.11111 | 8 | 3.875 | 11 |
| 2 | 2.91666 | 10 | 3.77777 | 12 |
| 3 | 3.0 | 12 | 3.5 | 13 |
| 4 | 4.09091 | 14 | 3.53846 | 15 |
| 5 | 4.28571 | 16 | 5.0 | 16 |
| 6 | 3.5625 | 15 | 4.0 | 17 |
| 7 | 4.06666 | 17 | 4.29411 | 18 |
| 8 | 4.52941 | 16 | 4.58823 | 19 |
| 9 | 4.05555 | 20 | 5.82352 | 21 |
| 10 | 3.61111 | 15 | 5.42105 | 22 |

**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 | 0.995 | 0.995 | 2.48 | 2.49 |
| 2 | 2.986 | 999.5 | 0.9975 | 0.9975 | 2.4865 | 2.43 |
| 3 | 2.989 | 1499.5 | 0.998 | 0.998 | 2.459 | 2.481 |
| 4 | 2.992 | 1999.5 | 0.99875 | 0.99875 | 2.487 | 2.45 |
| 5 | 2.993 | 2499.5 | 0.9989 | 0.9992 | 2.478 | 2.39 |
| 6 | 2.9945 | 2999.5 | 0.999 | 0.999 | 2.457 | 2.42 |
| 7 | 2.995 | 3499.5 | 0.9991 | 0.99914 | 2.486 | 2.4 |
| 8 | 2.996 | 3999.5 | 0.9993 | 0.99937 | 2.469 | 2.46 |
| 9 | 2.9963 | 4499.5 | 0.9995 | 0.9995 | 2.487 | 2.41 |
| 10 | 2.9967 | 4999.5 | 0.9996 | 0.9996 | 2.487 | 2.42 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 | 8.9 | 499.5 | 7.98 | 7.98 | 8.7 | 11.28 |
| 2 | 9.9 | 999.5 | 8.98 | 8.98 | 9.7 | 12.98 |
| 3 | 10.6 | 1499.5 | 9.63 | 9.63 | 10.3 | 13.83 |
| 4 | 10.9 | 1999.5 | 9.97 | 9.97 | 10.77 | 13.83 |
| 5 | 11.3 | 2499.5 | 10.36 | 10.36 | 11.09 | 14.968 |
| 6 | 11.6 | 2999.5 | 1063 | 1063 | 11.34 | 14.734 |
| 7 | 11.8 | 3499.5 | 10.81 | 10.81 | 11.60 | 15.89 |
| 8 | 11.9 | 3999.5 | 10.97 | 10.97 | 11.74 | 15.9 |
| 9 | 12.18 | 4499.5 | 11.18 | 11.18 | 11.969 | 16.07 |
| 10 | 12.36 | 4999.5 | 11.36 | 11.36 | 12.1 | 17.803 |

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

מספר פעולות איזון:

עץ AVL:

1. סדרה חשבונית:

כפי שננתח בהמשך הכנסה בסדרה מאוזנת נראה כי כל צומת מתעדכן לפחות פעם אחת.

כל פעם שנתחיל למלא שורה חדשה נצטרך בשלב מסוים לעדכן את כל הגבהים לפחות פעם אחת בעץ. בנוסף על מנת להגיע לבסוף לתצורה מאוזנת של העץ בכל הכנסה כמעט נרצה לבצע מספר פעולות גלגול על מנת ״להזיז את הצמתים ימינה״ לכן עבור כל הכנסה יהיו כ2 פעולות גלגול. דומה לתוצאת הניסוי

1. סדרה מאוזנת: נשער כי לכל פעולת הכנסה נצטרך לאזן (לעדכן גבהים) לכל היותר פעם אחת לכל הכנסה.

אם ננסה לחשב את כמות האיזונים כאשר מכניסים את האיברים באופן המיטבי נראה כי אין בכלל פעולת גלגול אלה רק פעולות עדכון גובה.

נסתכל על עץ מאוזן ומלא בעומק מסוים ונחשב את כמות האיזונים בהכנסה של שורה חדשה בעץ.

כל עלה בעץ המקורי יקבל שני בנים כך שגובה כל אחד מההורים(גם הקדומים) של אותו עלה יקבלו פעולת עדכון אחת, ז״א בכל הכנסה של שורה חדשה בעץ והגעה למצב של עץ מלא נבצע כ n/2 (ערך עליון) עדכונים, אם נסכום בעבור כל גובה בעץ n/2 עדכונים כאשר n הוא כמות הצמתים בעץ עד אותו הגובה נקבל סדרה הנדסית השואפת לאחד. תוצאה זו דומה לתוצאת הניסוי.

1. סדרה אקראית: היינו מצפים כי זה יהיה זהה לתוצאות הניסוי הראשון.

עץ ללא מנגנון איזון:

1. סדרה חשבונית: כיוון שאנו מכניסים בתחילת הסדרה כל פעם, נקבל שבכל הכנסה נצטרך לעדכן את כלל הצמתים מעל הצומת ולכן נקבל סכום של סדרה הנדסית מאפס ועד n כפי שקיבלנו.
2. סדרה מאוזנת: כיון שהסדרה מאוזנת נקבל סיטואציה זהה לניתוח בעץ AVL ולכן הציפייה זהה.
3. סדרה אקראית: התוצאה תהייה איפשהו בין חשבונית למאוזנת ויכולה להשתנות כל פעם תלוי באקראיות שבה הכנסנו

עומק העץ:

עץ AVL:

1. עץ מאוזן:

עבור עץ מאוזן נרצה לבנות סדרה לטובת חישוב העומק הממוצע:

בגובה הראשון יש עלה אחד בעומק אפס

לאחר מכן שני עלים בעומק אחד

לאחר מכן ארבע עלים בעומק שתים

לאחר מכן שמונה עלים בעומק שלוש

.

.

בעומק h נקבל 2^h עלים

ניתן לחשוב על כך גם שכמחצית מהאיברים בעומק המקסימלי, מחציתם בעומק אחד מתחת וכן הלאה ולכן אם נציב ונחשב נקבל תוצאות דומות למה שקיבלנו.

1. סדרה חשבונית: נצפה כי התוצאות יהיו זהות ואולי קצת יותר גבוהות מהתוצאה שבעץ המאוזן כיוון שהעומק הממוצע אמור קרוב לעץ המאוזן כיוון שבשאיפה העץ מאזן את עצמו כל הזמן עד כדי קבוע.
2. אקראי: נצפה כי זה יהיה בין המקרה הגרוע בו מכניסים כל פעם באיבר הראשון לבין המקרה מיטבי בו הסדרה מאוזנת ואין גלגולים.

|  |  |  |  |
| --- | --- | --- | --- |
| **:AVLNode()** | | | |
| **פונקציותget():** | | | |
| **סיבוכיות** | **סיבוכיות במילים** | **תיאור במילים** | **שם הפונקציה** |
| **O(1)** | **איטרציה יחידה להחזרת שדה** | **מחזירה את שדה הdepth של הnode** | **getDepth()** |
| **מחזירה את שדה הsize של הnode** | **getSize()** |
| **מחזירה מצביע לבן השמאלי** | **getLeft()** |
| **מחזירה מצביע לבן הימני** | **getRight()** |
| **מחזירה מצביע להורה** | **getParent()** |
| **מחזירה את שדה הbf של הnode** | **getBalanceFactor()** |
| **מחזירה את שדה הvalue של הnode** | **getValue()** |
| **מחזירה את שדה הheight של הnode** | **getHeight()** |
| **O(logn)** | **מטייל כלפי מעלה/מטה בעץ לכן חסום על ידי גובה העץ שהוא log n** | **מחזירה מצביע לעלה המקסימלי בתת העץ** | **getMax()** |
| **מחזירה מצביע לעלה המינמלי בתת העץ** | **getMin()** |
| **מחזירה מצביע לאיבר בעל האינדקס העוקב בעץ** | **getSucceor()** |
| **מחזירה מצביע לאיבר בעל האינדקס הקודם בעץ** | **getPredecessor()** |
| **פונקציות set():** | | | |
| **O(1)** | **איטרציה יחידה לעדכון שדה** | **מעדכנת שדה Depth** | **setDepth(x)** |
| **מעדכנת שדה Size** | **setSize(i)** |
| **מעדכנת שדה Left (מצביע לAVLNode())** | **setLeft(node)** |
| **מעדכנת שדה Right (מצביע לAVLNode())** | **setRight(node)** |
| **מעדכנת שדה Parent (מצביע לAVLNode())** | **setParent(node)** |
| **מעדכנת שדה Value** | **setValue(x)** |
| **מעדכנת שדה Height** | **setHeight(h)** |
| **פונקציות update():** | | | |
| **O(1)** | **עדכון שדה באמצעות משיכת נתונים מהילדים, ללא לולאות או חזרות** | **מעדכן את שדה size באמצעות סכום size הבנים + 1** | **updateSize()** |
| **מחשב את הBF באמצעות הפרש גבהי הילדים** | **updateBalanceFactor()** |
| **מעדכן את שדה height באמצעות מציאת מקסימום הגובה בין הבנים + 1** | **updateHeight()** |
| **קריאה ל3 פונקציות מסיבוכיות O(1)** | **קורא לפונקציות: updateSize, updateHeight, updateBalanceFactor** | **updateNodeInfo()** |
| **פונקציות is: מחזירות Boolean** | | | |
| **O(1)** | **גישה לאחד השדות של הצומת הנוכחית ובדיקה בהתאם** | **מחזיר האם הצומת מייצג צומת אמיתי באמצעות בדיקת הגובה מחזיר True אם כן** | **isRealNode()** |
| **מחזיר Trueאם לצומת קיים בן שמאלי בלבד** | **haveOnlyLeftSon()** |
| **מחזיר Trueאם לצומת קיים בן ימני בלבד** | **haveOnlyRightSon()** |
| **מחזיר Trueאם הצומת הוא בן שמאלי** | **isLeftSon()** |
| **מחזיר Trueאם הצומת הוא בן ימני** | **isRightSon()** |
| **מחזיר True אם לצומת קיים הורה** | **haveParent()** |
| **מחזיר Trueאם הצומת הוא עלה(ללא בנים)** | **isLeaf()** |
| **פונקציות נוספות:** | | | |
| **O(1)** | **עדכון שדה אחד בצומת** | **מגדיל את שדה size באחד** | **incraeseSizeByOne()** |
| **מקטין את שדה size באחד** | **decreaseSizeByOne()** |
| **O(1)** | **שינוי מצביעים של ההורה ושל הילד בלבד** | **הפונקציה נקראת כחלק מפעולת הdelete אחראית על מחיקת צומת בעל בן יחיד באמצעות שינוי המצביעים של הילד שלו ושל ההורה שלו לטובת ״התעלמותו מהעץ״** | **bypass()** |

|  |  |  |  |
| --- | --- | --- | --- |
| **מחקלת AVLTreeList():** | | | |
|  | | | |
| **O(logn)** | **קוראת לפונקציה treeSelect שהיא מסיבוכיות o(logn)** | **מחזירה את האיבר הi במבנה**  **קוראת לtreeSelect ומקבלת ממנו מצביע לאיבר הi ואז מחזירה את שדה הvalue** | **Retrive(i)** |
| **קוראת לפונקציה treeSelectRec שהיא מסיבוכיות o(logn)** | **פונקצית מעטפת לtreeSelectRec, מקבלת אינדקס וnode ומחזירה מצביע לצומת הi בתת עץ של node** | **treeSelect(root,i)** |
| **מטיילת בגבהי העץ שחסום על ידי o(logn)** | **באמצעות טיול בעץ ביחס לשדות הsize מחזירה מצביע לצומת הi** | **treeSelectRec(root,i)** |
| **קוראת לפונקציה insertRec שהיא מסיבוכיות o(logn)** | **פונקצית מעטפת לinsertRec** | **Insert(i, val)** |
| **מטיילת בגבהי העץ שחסום על ידי o(logn)** | **מייצרת צומת חדש עם ערך val ומכניסה אותו לאינדקס הi** | **insertRec(i, v)** |
| **מטיילת בגבהי העץ שחסום על ידי o(logn)** | **מוחקת את האיבר הi ממבנה הנתונים** | **Delete(i)** |
| **קוראת לפונקציה fixTreeשהיא מסיבוכיות o(logn)** | **אחראית על איזון העץ לאחר מחיקה, קוראת לfixTree** | **fixTreeAfterDeletion(node)** |
| **O(n)** | **קוראת לפונקציה listToArrayRec שהיא מסיבוכיות o(logn)** | **מחזירה רשימה לפי מבנה הנתונים, פונקצית מעטפת לlistToArrayRec** | **listToArray()** |
| **הליכת inorder על כל איברי העץ** | **מחזירה רשימה לפי מבנה הנתונים באמצעות inorder walk** | **listToArrayRec(node)** |
| **O(n)** | **קוראת לפונקציה searchRec שהיא מסיבוכיות O(n)** | **מחזירה את האינדקס הראשון ברשימה שמכיל את val, פונקציית מעטפת לsearchRec** | **search(val)** |
| **במקרה הגרוע, הליכת inorder על כל איברי העץ** | **מחזירה את האינדקס הראשון ברשימה שמכיל את val, באמצעות פעולת getSuccessor עד שמקבלים את האיבר הרצוי** | **searchRec(val, node, i)** |
| **פונקציות איזון** | | | |
| **O(logn)** | **מטייל כלפי מעלה בעץ החל מהאיבר שהוכנס/נמחק ולכן חסום על ידי גובה העץ, בכל צומת מבצע גלגול במידת הצורך** | **פונקציה האחראית לאיזון העץ לאחר פעולות מחיקה והכנסה, מקבל fixAfterInsert המתאר האם זאת פעולה לאחר מחיקה או הכנסה** | **fixTree(node, fixAfterInsert)** |
| **O(1)** | **שינוי מצביעים לצמתים A,B ועדכון השדות שלהם** | **גלגול ימינה ואז שמאלה, מחזירה 2 כמספר האיזונים, קוראת לפונקציוח leftRotate, rightRotate בהתאם** | **rightThenLeft(B)** |
| **גלגול שמאלה ואז ימינה, מחזירה 2 כמספר האיזונים, קוראת לפונקציוח leftRotate, rightRotate בהתאם** | **leftThenRight(B)** |
| **גלגול שמאלה מחזירה 1 כמספר האיזונים** | **leftRotate(B)** |
| **גלגול ימינה מחזירה 1 כמספר האיזונים** | **rightRotate(B)** |
| **אחראית על עדכון הצמתים שלקחו חלק בגלגול** | **updateNodesInfo(A,B)** |
| **פונקציות get:** | | | |
| **O(1)** | **גישה/שינוי שדות העץ** | **מחזירה את שדה Value של האיבר באינדקס 0** | **First()** |
| **מחזיר את שדה הvalue של האיבר באינקדס האחרון** | **Last()** |
| **מחזירה את אורך הרשימה** | **Length()** |
| **מחזירה מצביע לאיבר הראשון בעץ** | **getFirstNode()** |
| **מחזירה מצביע לאיבר האחרון בעץ** | **getLastNode()** |
| **מחזירה true אם העץ ריק** | **Empty()** |
| **מחזירה מצביע לשורש של העץ** | **getRoot()** |
| **פונקציות שינוי מצביעים** | | | |
| **O(1)** | **משנה את שדות המצביע לעץ בלבד** | **מוחק את נתוני העץ לעץ נקי** | **deleteAllTree()** |
| **שינוי מצביעים של ההורה ושל הילד בלבד** | **הפונקציה נקראת כחלק מפעולת הsplit אחראית על ניתוק תת העץ ששורשו הוא root מהעץ הראשי באמצעות שינוי המצבעים של הצומת root ושל ההורה שלו** | **detachSubtree(root)** |
| **שינוי מצביעים של ההורה של הילד ושל הבנים של הילד בלבד** | **הפונקציה נקראת כחלק מפעולת הsplit, אחראית על ניתוק הצומת node מהעץ הראשי על ידי שינוי המצבעים שלו של הבנים שלו ושל ההורה שלו** | **detachNode(node)** |
| **גישה/שינוי שדות העץ** | **מעדכנת את השדה root של העץ עם מצביע לnode, אם לnode יש מצביע להורה אז הפונקציה משנה את המצביע לNone** | **setRoot(node)** |
| **O(logn)** | **בה"כ מטייל על הענף השמאלי של העץ הגבוה עד לצומת הראשון בגובה של העץ הנמוך, ולכן חסום על ידי גובה העץ** | **מוסיפה לעץ את העץ other באמצעות הצומת x שנמצאת ביניהם, מקבלת toRight המתאר האם other מתווסף להתחלה של העץ או לסוף של העץ** | **join(x, other ,toRight)** |
| **O(logn)** | **קוראת לפונקציות delete ו-join שתיהן מסיבוכיות O(logn)** | **מוסיפה את lst לסוף הרשימה, ומחזירה את הפרש הגבהים ביו lst לעץ המקורי** | **concat(lst)** |
| **O(logn)** | **קוראת לפונקציה treeSelect שהיא מסיבוכיות O(logn) ואז קוראת לפקונקציה Join לכל צומת במסלול בין האיבר הi לשורש – זה טור טלסקופי החסום ע"י גובה העץ** | **מפצלת את הרשימה המקורית לשתי רשימות: האיברים שלפני האינדקס i והאיברים שאחרי האינדקס i, מחזירה מערך המכיל את שתי הרשימות ואת הערך באינדקס i** | **split(i)** |

**פירוט נוסף על פונקציות נבחרות:**

1. **treeSelectRec(root,i):**

**מקבלת: מצביע לצומת ואינדקס i.**

**מחזירה: מצביע לצומת הi ביחס לroot**.

**תיאור פעולת הפונקציה: באמצעות רקורסיה והשוואת שדות הsize נטייל בעץ. במידה וi גדול מגודל תת העץ השמאלי(+1) נדע שהאינדקס i נמצא בתת עץ הימני, במידה וi קטן משדה הsize זה אומר שיש פחות איברים בתת עץ השמאלי ולכן האיבר נמצא שם. כך בהתאם נלך לאחד מתתי העצים ונקרא לפונקציה רקורסיבית שוב עם שינוי הi בהתאם.**

**סיבוכיות: כיוון שבכל צומת אנו מבצעים השוואת לאינדקס ז״א עבודה קבועה אנו תלויים בגובה העץ שכידוע בעץ AVL הגובה חסום על ידי logn ולכן הפונקציה מסיבוכיות O(logn).**

1. **Insert(I,val):**

**מקבלת: אינדקס בו נכניס את הערך val**

**מחזירה: מספר האיזונים שבוצעו.**

**תיאור פעולת הפונקציה: הפונקציה ראשית מייצרת node חדש עם הערך val וראשית במידת הצורך מעדכנת את שדות הroot,last,first של העץ על מנת ששדות אלה יתוחזקו באופן קבוע לגישה מהירה אחרי זה**.

**לאחר מכן קוראת לפונקציה insertRec() ממנה נקבל את מספר האיזונים.**

**סיבוכיות: עדכון השדות הוא O(1) והפונקציה insertRec היא בסיבוכיות של O(logn) ולכן סה״כ נקבל סיבוכיות O(logn).**

1. **insertRec(i, root, nodeToInsert,depth):**

**מקבלת: מקבלת אינדקס, מצביע לצומת שאותו נרצה להכניס ומצביע לצומת שבו אנו נמצאים כרגע.**

**מחזירה: מספר האיזונים שבוצעו.**

**תיאור פעולת הפונקציה: באמצעות רקורסיה בדומה לtreeSelect נטייל בעץ עד שנגיע למקום בו נרצה להכניס את הצומת הנוסף. תנאי העצירה בפונקציה הנ״ל כאשר הגענו לתת עץ בגודל 0/1/2 והאינדקס הוא אחד מהם ובהתאמה נקשר את הצומת החדש לצומת המתאים.**

**בכל שלב בו נרד בעץ נעדכן את שדה size של הצומת ממנו אנו באים.**

**לבסוף נקרא לפונקציה fixtree על ההורה של הצומת אותו הכנסנו ומהפונקציה נקבל counter שהוא סופר את כמות האיזונים שביצענו על העץ, אותו נחזיר עם סיום פעולת הפונקציה.**

**סיבוכיות: בכל צומת אנו מבצעים בדיקות ומעדכנים את שדה size לכן כל איטרציה היא O(1) סה״כ גובה העץ חסום על ידי logn ולכן נקבל סיבוכיות o(logn), הפונצקיה fixTree נקראת פעם אחת בסוף תהליך ההכנסה והיא בסיבוכיות זהה.**

1. **Delete(i):**

**מקבלת: אינדקס i עבורו נמחק את האיבר הi ברשימה.**

**מחזירה:מספר האיזונים שבוצעו**

**תיאור פעולת הפונקציה: באמצעות retrive נקבל מצביע לצומת אותו נרצה למחוק.**

**אם לצומת יש שני בנים, נמצא את היורש שלו נבצע על היורש bypass ונחליף בין היורש לצומת שאנו רוצים למחוק(החלפת מצביעים ולא ערכי val).**

**במידה ולצומת יש בן אחד נבצע על הצומת bypass.**

**לאחר מכן נקרא לfixTreeAfterDeletion שמעדכנת שדות וקוראת לfixTree ממנה נקבל את מספר האיזונים שבוצעו.**

**סיבוכיות: retrive – O(logn) נקראת פעם אחת. bypass(1) – נקראת פעם אחת. Fixtree – O(logn) נקראת פעם אחת. סה״כ נקבל O(logn)**

1. **listToArrayRec:**

**מחזירה: רשימה לפי אינדקסים של איברי העץ.**

**תיאור פעולת הפונקציה: באמצעות הליכת in-order על העץ אוספת את ערכי הVal של כל הצמתים ומכניסה אותם לרשימה.**

**סיבוכיות:** **כפי שלמדנו בכיתה o(n)**

1. **fixTree(node, fixAfterInsert):**

**מקבלת: מצביע ממנו נרצה לאזן את העץ, true אם זאת פעולת איזון לאחר הכנסה.**

**מחזירה: מספר האיזונים שבוצעו בעץ.**

**תיאור פעולת הפונקציה: מהמצביע שקיבלנו נטפס כלפי שורש העץ ונעבור על הצמתים, ראשית נשמור את ערך גובה הצומת לפני עדכון גובהו ולאחר מכן נעדכן את שדות הHeight, Bf במידה וגובה העץ לא השתנה נסיים את הפעולה.**

**במידה והשתנה הגובה נבדוק האם הabs(bf) = 2 במידה וכן נבצע גלגול במידת הצורך ונשמור counter לספירת כמות הגלגולים שביצענו עד השורש.**

**סיבוכיות: כיוון שאנו עולים בעץ ובכל צומת מבצעים עדכון שדות בלבד נראה כי אנו החסם העליון של פעולת הפונקציה הוא גובה העץ שהוא חסום על ידי logn לכן סיבוכיות הפונקציה היא O(logn).**

1. **:searchRec(val, node, i)**

**מקבלת: val הערך שנרצה למצוא ברשימה, node מצביע לאיבר הנוכחי, ו-i האינדקס של node**

**מחזירה: את האינדקס של האיבר הראשון ברשימה שמכיל את הערך val**

**תיאור פעולת הפונקציה: בקריאה הראשונה מפונקציית המעטפת, בודקת את הערך של האיבר הראשון ברשימה שהאינדקס שלו הוא 0, אם הערך שלו הוא הערך הרצוי val נחזיר את האינדקס שלו i אחרת ניגש לעוקב שלו בעזרת הפונקציה getSuccessor ונקדם את i ב-1 אם הגענו לסוף הרשימה ולא מצאנו איבר מתאים, הפונקציה מחזירה -1**

**סיבוכיות: במקרה הגרוע, האיבר עם הערך הרצוי מופיע בסוף הרשימה (או לא מופיע בכלל) ולכן נצטרך לבצע n פעולות successor, והוכחנו בתרגול שהסיבוכיות היא O(n + h) = O(n)**

1. **:join(x, other ,toRight)**

**מקבלת: x הצומת שנמצאת בין שתי הרשימות, other הרשימה שנרצה לשרשר, ו-toRight ערך בוליאני המציין האם other מווספת לסוף הרשימה**

**מחזירה: הערך המוחלט של הפרש הגבהים בין שתי הרשימות שנרצה לחבר**

**תיאור פעולת הפונקציה: נניח בה"כ שאנחנו משרשרים לסוף הרשימה, המקרה ההפוך סימטרי. אם העץ הגדול גבוה יותר נטייל על הענף השמאלי ביותר עד שנגיע לצומת הראשונה שהגובה שלה קטן או שווה לגובה של העץ הקטן, נהפוך אותה ואת השורש של העץ הקטן לבנים של x ואת ההורה הקודם שלה להורה של x, אם העץ הקטן גבוה יותר נפעל באופן דומה אך נטייל על הענף הימני ביותר של העץ הקטן במקום, כעט נבצע פעולות איזון אם נדרש. אם העצים בעלי גובה זהה עד כדי 1, נהפוך את השורשים של שתי העצים לבנים של x.**

**סיבוכיות: כפי שלמדנו בכיתה הסיבוכיות שווה ל- O(self.height – other.height+1) שבמקרה הגרוע שווה לO(logn)**

1. **:concat(lst)**

**מקבלת: lst רשימה שנרצה לשרשר לסוף הרשימה שלנו**

**מחזירה: הפרש הגבהים בין העצים שהיא חיברה**

**תיאור פעולת הפונקציה: הפונקציה ניגשת לאיבר האחרון הרשימה ומוחקת אותו באמצעות הפונקציה delete ולאחר מכן קוראת לפונקציה join כאשר הצומת המקשר הוא הצומת שמחקנו, ונציין שמחברים את lst לסוף הרשימה**

**סיבוכיות: הפונקציה קוראת לשתי פונקציות ששתיהן מסיבוכיות O(logn) במקרה הגרוע ולכן הסיבוכיות היא O(logn)**

1. **:split(i)**

**מקבלת: i האינקדס של הצומת שבאמצעותה נפצל את העץ**

**מחזירה: מערך המכיל את - רשימת האיברים עד האינדקס הi , הערך של האינדקס הi , ורשימת האיברים החל מהאינקס הi+1**

**תיאור פעולת הפונקציה: הפונקציה מוצאת את הצומת שנמצא באינדקס הi בעזרת הפונקציה treeSelect ומוסיפה את תת העץ הימני שלו לרשימה הגדולה, ואת תת העץ השמאלי שלו לרשימה הקטנה כעת מתחילים לטפס במעלה המסלול מהצומת באינדקס הi אל השורש ובכל פעם שנפנה ימינה נוסיף את תת העץ הימני לרשימה הגדולה, אחרת נוסיף את תת העץ השמאלי לרשימה הקטנה באמצעות הפונקציה Join**

**סיבוכיות: הפונקציה קוראת לפונקציה treeSelect שהיא בסיבוכיות O(logn) במקרה הגרוע, וקוראת לפונקציה join על כל צומת במסלול אל השורש – ראינו בכיתה שסכום הסיבוכיות הוא טור טלסקופי שחסום על ידי גובה העץ ולכן במקרה הגרוע הסיבוכיות היא O(logn)**

**ולכן בסה"כ הסיבוכיות במקרה הגרוע היא O(logn)**