מבנה נתונים- תיעוד תרגיל מעשי 1

# חלק א- תיעוד פונקציות

## מחלקת AVLTree

### בנאי AVLTree

**תיאור כללי**- בנאי ליצירת אובייקט חדש מסוג AVLNode.

**עקרון מימוש**:

שדות האובייקט- אתחול בהתאם לערכי ברירת המחדל הבאים:

* root- מצביע לשורש העץ, מאותחל מצביע ל- virtualLeaf
* min – מצביע לצומת שבה הערך המינימלי, אם העץ ריק אז השדה הוא null
* max – מצביע לצומת שבה הערך המקסימלי, אם העץ ריק אז השדה הוא null
* virtualLeaf – צומת ווירטואלית אליה יצביעו עלים וצמתים אונריים.

**סיבוכיות**-

### empty()

**תיאור כללי**- הפונקציה מחזירה ערך TRUE אם ורק אם העץ ריק.

**עקרון מימוש**- הפונקציה מסתכלת על שדה השורש root. אם הוא virtualLeaf מחזירה true, אחרת מחזירה false.

**סיבוכיות**-

### treePosition(int k, IAVLNode root)

**תיאור כללי**- הפונקציה מחפשת איבר בעל המפתח k. אם קיים איבר כזה, היא מחזירה את v הצומת, אחרת היא מחזירה את ההורה של הצומת.

**עקרון מימוש**- חיפוש בעץ בינארי, יוצרת מצביע ומתחילה מלהצביע לשורש העץ. בודקת אם k קטן, גדול או שווה למפתח של המצביע. אם המפתח שווה מחזירה את הצומת. אם קטן, מעדכנת את המצביע לבן השמאלי שלו, ואם גדול מעדכנת לבן הימני שלו. האיטרציה ממשיכה כל עוד לא הגענו לצומת שהמפתח שווה ל-k או עד שהגענו לעלה (הבנים שלו הם צמתים וירטואליים). אם נמצאה צומת ש-k שווה לערך בה מחזירים את הצומת. אחרת מחזירים את הצומת שהגענו אליה (העלה).

**סיבוכיות**-

### search(int k)

**תיאור כללי**- הפונקציה מחפשת איבר בעל המפתח k. אם קיים איבר כזה, היא מחזירה את הערך השמור עבורו, אחרת היא מחזירה null.

**עקרון מימוש**- קוראת ל-treePosition. משווה את המפתח של הצומת שהוחזרה ל-k, אם הם זהים, מחזירה את שדה ה-info של הצומת. אחרת, מחזירה null.

**סיבוכיות**-

### insert(int k, String s)

**תיאור כללי**- הכנסת איבר בעל ערך s ומפתח k לעץ, אם המפתח לא קיים. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בסה"כ בשלב תיקון העץ על מנת להשלים את הפעולה (גלגולי LR ו- RL נחשבים ל- 2 פעולות איזון). אם קיים איבר בעל מפתח k בעץ הפונקציה מחזירה -1 ולא מתבצעת הכנסה.

**עקרון מימוש**- הפונקציה מבצעת את הפעולות הבאות:

* יוצרת משתנה בשם rebalanceActions מסוג int עם הערך 0.

אם העץ ריק (הפונקציה empty מחזירה true):

* יוצרת אובייקט AVLNode חדש עם המפתח k והערך info
* מעדכנת את שדה ה-root של העץ להצביע לצומת זו
* מעדכנת את שדות ה-min וה-max של העץ להצביע לצומת החדשה
* מעדכנת את שדות הצומת החדשה:
  + setLeft – שיצביע לעלה הווירטואלי
  + setRight – שיצביע לעלה הווירטואלי
  + setParent – שיצביע לעלה הווירטואלי
  + setHight – שיהיה שווה ל-0
  + setSize – שיהיה שווה ל-1

אחרת:

* קוראת ל-treePosition(k) ושומרת במצביע בשם parent.
  + אם המפתח של הצומת שהוחזרה שווה ל-k (הצומת כבר בעץ) - מחזירה (-1).
  + אחרת-
    - יוצרת אובייקט AVLNode חדש עם המפתח k והערך info ושומרת במצביע בשם child.
    - משווה בין המפתח של ה-parent ל-k. אם k גדול מהמפתח מגדירה את הבן הימני של parent להצביע ל-child. אם k קטן מהמפתח הזה מגדירה שהבן השמאלי של parent יצביע ל-child (עם setRight או setLeft).
    - מעדכנת את שדות הצומת החדשה:
      * getLeft – שיצביע לעלה הווירטואלי
      * setRight – שיצביע לעלה הווירטואלי
      * setParent – שיצביע לצומת ההורה
      * setHight – להיות שווה ל-0
      * setSize – שיהיה שווה ל-1
    - **עדכון המינימום-** משווה בין המפתח של הצומת החדשה למפתח ששמור בשדה min. אם הוא קטן יותר- מעדכנת את min להצביע לצומת החדשה.
    - **עדכון המקסימום-** משווה בין המפתח של הצומת החדשה למפתח ששמור בשדה max. אם הוא גדול יותר- מעדכנת את max להצביע לצומת החדשה.
    - **מאזנת את העץ-** אם ל-parent כעת רק בן אחד (הוא היה עלה לפני ההוספה)- קוראת ל- rebalance(parent)ושומרת את הערך שהיא מחזירה ב- rebalanceActions.
    - **עדכון size-** קוראת ל- updateSize(child) לעדכן את שדות ה-size לאחר הוספת הצומת לעץ.
* מחזירה את rebalanceActions

**סיבוכיות**- בכל מקרה מכיוון ש-updateSize לוקחת

### delete(int k)

**תיאור כללי**- מחיקת איבר בעל המפתח k בעץ, אם הוא קיים. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו בסך-הכל בשלב תיקון העץ על מנת להשלים את הפעולה.אם לא קיים איבר בעל המפתח k בעץ הפונקציה מחזירה .

**עקרון מימוש**- הפונקציה מבצעת את הפעולות הבאות:

* יוצרת משתנה בשם rebalanceActions מסוג int עם הערך 0.
* קוראת ל-treePosition(k) ושומרת במצביע בשם x.
  + אם המפתח של הצומת שהוחזרה שונה מ-k (הצומת לא בעץ) - מחזירה (-1).
  + אחרת-
    - **עדכון המינימום-** קוראת ל- findSuccessor(x)- אם מוחזרת צומת מעדכנים את min להצביע לצומת זו. אם מוחזר null מעדכנים את min להיות מינוס אינסוף (זה אומר שמחקנו את השורש).
    - **עדכון המקסימום-** קוראת ל- findPredecessor(x)- אם מוחזרת צומת מעדכנים את max להצביע לצומת זו. אם מוחזר null מעדכנים את max להיות פלוס אינסוף (זה אומר שמחקנו את השורש).

בודקת:

* + - אם x זה עלה (שני הילדים הם עלים ווירטואליים) – קוראת ל- deleteLeaf(x) ושומרת את הצומת המוחזרת במשתנה y.
    - אם x זו צומת אונרית – קוראת ל- deleteUnar(x) ושומרת את הצומת המוחזרת במשתנה y.
    - אחרת – קוראת ל- deleteDouble(x) ושומרת את הצומת המוחזרת במשתנה y.
  + **מאזנת את העץ-** קוראת ל- rebalance(y) ושומרת את הערך שהיא מחזירה ב- rebalanceActions.
  + **עדכון size-** קוראת ל- updateSize(child) לעדכן את שדות ה-size לאחר מחיקת הצומת מהעץ.
* מחזירה את rebalanceActions

**סיבוכיות**- בכל מקרה מכיוון ש-updateSize לוקחת

### deleteLeaf(IAVLNode x)

**תיאור כללי**- הפונקציה מקבלת עלה ומוחקת אותו מהעץ.

**עקרון מימוש**- הפונקציה:

* שומרת את ההורה של x במשתנה y = getParent(x).
* בודקת אם x הוא השורש. אם כן אז משנה את המצביע של root להיות virtualLeaf
* אחרת- בודקת אם x הוא בן ימני או בן שמאלי של y ובהתאם מעדכנת את הבן הימני או השמאלי להיות צומת וירטואלית.
* מחזירה את y

**סיבוכיות**-

### deleteUnar(IAVLNode x)

**תיאור כללי**- הפונקציה מקבלת צומת אונרית ומוחקת אותה מהעץ

**עקרון מימוש**- הפונקציה:

* שומרת את ההורה של x במשתנה y = getParent(x).
* בודקת אם x הוא השורש. אם כן אז משנה את המצביע של root להיות הבן היחיד של x ומעדכנת את ההורה של הבן הזה להיות עלה וירטואלי.
* קוראת להורה בודקת אם x הוא בן ימני או בן שמאלי של y ובהתאם מעדכנת את הבן הימני או השמאלי להיות הבן היחיד של x. מעדכנת את ההורה של הבן היחיד להיות y.
* מחזירה את y

**סיבוכיות**-

### deleteDouble(IAVLNode x)

**תיאור כללי**- הפונקציה מקבלת צומת עם שני בנים ומוחקת אותה מהעץ. היא מבצעת זאת באמצעות מציאת ה-successor ומחיקתו והחלפת המקומות ביניהם כפי שראינו בהרצאה.

**עקרון מימוש**- הפונקציה:

* קוראת לפונקציית findSuccessor(x) ושומרת את הערך המוחזר ב-y (נשים לב ש-y לא יכול להיות null כי זה אומר שזו צומת ה-max והיא בהכרח עלה או אונרית וטיפלנו במקרה הזה. מצד שני ראינו כי y זו צומת אונרית או עלה בעצמה).
* שומרת את ההורה של y במשתנה z.
* בודקת:
  + אם y עלה- קוראת ל- deleteLeaf(y)
  + אם y אונרית- קוראת ל- deleteUnar(y)
* מעדכנת את השדות של y להיות אלו של x:
  + setRight(y) = getRight(x)
  + setLeft(y) = getLeft(x)
  + setParent(getRight(x)) = y
  + setParent(getLeft(x)) = y
  + מעדכנת את ההורה של y
    - אם x הוא שורש העץ:
      * מעדכנת את שדה root להצביע ל-y
      * מעדכנת עםsetParent(y) שיצביע לעלה ווירטואלי
    - אחרת- בהתאם לאם x הוא בן ימני או בן שמאלי מעדכנת את המצביע המתאים של ההורה של x להצביע ל-y ומעדכנת את setParent(y) = getParent(x)
* מחזירה את z

**סיבוכיות**-

### rebalance(IAVLNode x, int countActions)

**תיאור כללי**- מבצעת את פעולת האיזון של העץ. נקראת על ידי Insert ו-delete. סופרת את כמות פעולות האיזון שהיא עושה.

**עקרון מימוש**- פונקציה רקורסיבית. מקבלת שורש של תת עץ. שומרת ספירה של כמות פעולות האיזון שביצענו במשתנה countActions.

* תנאי עצירה:
  + אם הפרש ה-rank בין הצומת לבנים שלה הוא מהצורה (1,1), (1,2) או (2,1) מחזירה 0
  + אם getParent(x)== virtualLeaf זה אומר שהגענו לשורש העץ ותפסיק הקריאה הרקורסיבית. במקרה הזה סיימנו לאזן את העץ ונחזיר גם 0
* בודקת האם הפרש ה-rank בין השורש לבנים מהצורה (0,1) או (1,0), אם כן אז קוראת ל-promote. לאחר מכן קוראת רקורסיבית ל-rebalance על ההורה של x ומחברת את הערך המוחזר ל- countActions.
* בודקת האם הפרש ה-rank בין השורש לבנים מהצורה (2,2), אם כן אז קוראת ל-demote. לאחר מכן קוראת רקורסיבית ל-rebalance על ההורה של x ומחברת את הערך המוחזר ל- countActions.
* אחרת- בודקת אם הפרש ה-rank בין השורש לבנים מהצורה (0,2)
  + אם הבן y עם ההפרש 0 בין ה-rank הוא צומת מסוג (1,2) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

rotateRight(x, y) + demote(x) + countActions

* + אם הבן y עם ההפרש 0 בין ה-rank הוא צומת מסוג (2,1) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

doubleRotateRight(x, y) + demote(x) + demot(y) + promote(getRight(y)) + countActions

* אחרת- בודקת אם הפרש ה-rank בין השורש לבנים מהצורה (2,0)
  + אם הבן y עם ההפרש 0 בין ה-rank הוא צומת מסוג (2,1) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

rotateLeft(x, y) + demote(x) + countActions

* + אם הבן y עם ההפרש 0 בין ה-rank הוא צומת מסוג (1,2) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

doubleRotateLeft(x, y) + demot(x) + demote(y) + promote(getLeft(y)) + countActions

* אחרת- בודקת אם הפרש ה-rank בין השורש לבנים מהצורה (3,1)
  + אם הבן y עם ההפרש 1 בין ה-rank הוא צומת מסוג (1,1) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

rotateLeft(x, y) + demote(x) + promote(y)+ countActions

* + אם הבן y עם ההפרש 1 בין ה-rank הוא צומת מסוג (2,1) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

rotateLeft(x, y) + demote(x) + demote(x)+ countActions.

לאחר מכן קוראת רקורסיבית ל-rebalance על ההורה של x ומחברת את הערך המוחזר ל- countActions

* + אם הבן y עם ההפרש 1 בין ה-rank הוא צומת מסוג (1,2) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

+ demote(y) + promote(getLeft(y)) doubleRotateLeft(x, y) + demote(x) + demote(x)+ countActions

* אחרת- בודקת אם הפרש ה-rank בין השורש לבנים מהצורה (1,3)
  + אם הבן y עם ההפרש 1 בין ה-rank הוא צומת מסוג (1,1) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

rotateRight(x, y) + demote(x) + promote(y)+ countActions

* + אם הבן y עם ההפרש 1 בין ה-rank הוא צומת מסוג (2,1) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

rotateRight(x, y) + demote(x) + demote(x)+ countActions.

לאחר מכן קוראת רקורסיבית ל-rebalance על ההורה של x ומחברת את הערך המוחזר ל- countActions

* + אם הבן y עם ההפרש 1 בין ה-rank הוא צומת מסוג (1,2) – קוראת לפעולות האיזון ומחברת את הערך שמחזירות עם countActions:

+ demote(y) + promote(getRight(y)) doubleRotateRight(x, y) + demote(x) + demote(x)+ countActions

* מחזירה את countActions

**סיבוכיות**-

\*\*כנראה שאפשר לעשות את זה עם פחות מקרים, לראות תוך כדי כתיבה\*\*

### promote(IAVLNode x)

**תיאור כללי**- הפונקציה מבצעת את פעולת ה- promote ומעלה את ה-rank של הצומת שקיבלה ב-1

**עקרון מימוש**- מקבלת צומת ומגדילה את ה-rank שלה ב-1. מחזירה 1.

**סיבוכיות**-

### demote(IAVLNode x)

**תיאור כללי**- הפונקציה מבצעת את פעולת ה- demote ומורידה את ה של הצומת שקיבלה ב-1

**עקרון מימוש**- מקבלת צומת ומקטינה את ה-rank שלה ב-1. מחזירה 1.

**סיבוכיות**-

### rotateRight(IAVLNode x, IAVLNode y)

**תיאור כללי**- ביצוע גלגול יחיד ימינה.

**עקרון מימוש**- x מצביע לשורש, y מצביע לבן שהפרש ה-rank בין השורש אליו הוא 0.

* אם x מצביע לאותה צומת כמו ה-root (שורש העץ)- נעדכן את השורש של העץ להצביע ל-y
* מעדכנים את setLeft(x) = getRight(y)
* מעדכנים את setParent(getLeft(x)) = x
* מעדכנים את setRight(y) = x
* מעדכנים את setParent(x) = y
* מחזירה 1

**סיבוכיות**-

### rotateLeft(IAVLNode x, IAVLNode y)

**תיאור כללי**- ביצוע גלגול יחיד שמאלה.

**עקרון מימוש**- x מצביע לשורש, y מצביע לבן שהפרש ה-rank בין השורש אליו הוא 0.

* אם x מצביע לאותה צומת כמו ה-root (שורש העץ)- נעדכן את השורש של העץ להצביע ל-y
* מעדכנים את setRight(x) = getLeft(y)
* מעדכנים את setParent(getRight(x)) = x
* מעדכנים את setLeft(y) = x
* מעדכנים את setParent(x) = y
* מחזירה 1

**סיבוכיות**-

### doubleRotateRight(IAVLNode x, IAVLNode y)

**תיאור כללי**- ביצוע גלגול כפול ימינה.

**עקרון מימוש**- x מצביע לשורש, y מצביע לבן שהפרש ה-rank בין השורש אליו הוא 0. מוסיפים מצביע לבן הימני של y בשם z.

* אם x מצביע לאותה צומת כמו ה-root (שורש העץ)- נעדכן את השורש של העץ להצביע ל-z
* קוראים rotateLeft(y, z)
* קוראים ל- rotateRight(x, z)
* מחזירה 2

**סיבוכיות**-

### doubleRotateLeft(IAVLNode x, IAVLNode y)

**תיאור כללי**- ביצוע גלגול כפול שמאלה.

**עקרון מימוש**- xמצביע לשורש, y מצביע לבן שהפרש ה-rank בין השורש אליו הוא 0. מוסיפים מצביע לבן השמאלי של y בשם z.

* אם x מצביע לאותה צומת כמו ה-root (שורש העץ)- נעדכן את השורש של העץ להצביע ל-z
* קוראים rotateRight(y, z)
* קוראים ל- rotateLeft(x, z)
* מחזירה 2

**סיבוכיות**-

### findSuccessor(IAVLNode x)

**תיאור כללי**- הפונקציה מחזירה את הצומת עם המפתח העוקב שאחרי x.

**עקרון מימוש**- הפונקציה מקבלת את x.

* אם x היא צומת ה-max – מחזירה null (אין לה עוקב אחרי)
* יוצרת מצביע curr שמצביע תחילה על x.
* אם העלה הימני קיים אז:
  + עושה צעד אחד ימינה בעץ
  + עושה לולאה של צעדים שמאלה בעץ כל עוד לא הגענו לעלה.
  + כשהגיעה לעלה השמאלי ביותר- מחזירה אותו
* אחרת:
  + יוצרת מצביע נוסף parent שמצביע על ההורה של הצומת
  + מתחילה לולאה- כל עוד הצומת היא בן ימני של ההורה, ולא הגענו לשורש, מקדמים את שני המצביעים צעד למעלה. כאשר הצומת היא בן שמאלי- מחזירים את ההורה.

**סיבוכיות**-

### finsPredecessor(IAVLNode x)

**תיאור כללי**- הפונקציה מחזירה את הצומת עם המפתח העוקב שלפני x.

**עקרון מימוש**- הפונקציה מקבלת את x. יוצרת מצביע p.

* אם x היא צומת ה-min – מחזירה null (אין לה עוקב אחרי)
* אם העלה השמאלי קיים אז:
  + עושה צעד אחד שמאלה בעץ
  + עושה לולאה של צעדים ימינה בעץ כל עוד לא הגענו לעלה.
  + כשהגיעה לעלה הימני ביותר- מחזירה אותו
* אחרת:
  + יוצרת מצביע נוסף parent שמצביע על ההורה של הצומת
  + מתחילה לולאה- כל עוד הצומת היא בן שמאלי של ההורה, ולא הגענו לשורש, מקדמים את שני המצביעים צעד למעלה. כאשר הצומת היא בן ימני- מחזירים את ההורה.

**סיבוכיות**-

### min()

**תיאור כללי**- מחזירה את ערך (info) האיבר בעץ בעל המפתח המינימלי, או null בעץ ריק.

**עקרון מימוש**- מומש בעץ שדה min שמתעדכן בכל הכנסה ומחיקה של צומת מהעץ. הטיפול בעדכון הערך בצומת מתבצע כחלק מפעולת insert או delete. המימוש מחזיר את הערך השמור בשדה.

**סיבוכיות**-

### max()

**תיאור כללי**- מחזירה את ערך (info) האיבר בעץ בעל המפתח המקסימלי, או null בעץ ריק.

**עקרון מימוש**- מומש בעץ שדה max שמתעדכן בכל הכנסה ומחיקה של צומת מהעץ. הטיפול בעדכון הערך בצומת מתבצע כחלק מפעולת insert או delete. המימוש מחזיר את הערך השמור בשדה.

**סיבוכיות**-

### keysToArray()

**תיאור כללי**- הפונקציה מחזירה מערך ממוין המכיל את כל המפתחות בעץ, או מערך ריק אם העץ ריק.

**עקרון מימוש**- המימוש יוצר מערך ריק בגודל ששווה ל-size של השורש שהוא הגודל של העץ כולו. לאחר מכן עובר על העץ ב-inOrder על ידי שימוש במחסנית עזר- קוראים לבן השמאלי ומכניסים אותו למחסנית כל עוד יש בן שמאלי. ברגע שמגיעים ל-null עושים pop(), מכניסים את ערך ה-key של ה-pop() למערך, ולאחר מכן קוראים לבן הימני ובאופן דומה מכניסים את ערך ה-key למערך. ממשיכים עד שהמערך מלא.

**סיבוכיות**- עוברים על כל הצמתים בעץ ולכן

### infoToArray()

**תיאור כללי**- הפונקציה מחזירה מערך מחרוזות המכיל את כל המחרוזות בעץ, ממוינות על פי סדר המפתחות. כלומר הערך ה j במערך הוא המחרוזת המתאימה למפתח שיופיע במיקום ה j במערך הפלט של הפונקציה keysToArray() . גם הפונקציה הזאת מחזירה מערך ריק אם העץ ריק.

**עקרון מימוש**- המימוש יוצר מערך ריק בגודל ששווה ל-size של השורש שהוא הגודל של העץ כולו. לאחר מכן עובר על העץ ב-inOrder על ידי שימוש במחסנית עזר- קוראים לבן השמאלי ומכניסים אותו למחסנית כל עוד יש בן שמאלי. ברגע שמגיעים ל-null עושים pop(), מכניסים את ערך ה-info של ה-pop() למערך, ולאחר מכן קוראים לבן הימני ובאופן דומה מכניסים את ערך ה-info למערך. ממשיכים עד שהמערך מלא.

**סיבוכיות**- עוברים על כל הצמתים בעץ ולכן

### updateSize(IAVLNode x)

**תיאור כללי**- הפונקציה מקבלת צומת ומעדכנת את שדה ה-size מצומת זו ועד שורש העץ.

**עקרון מימוש**- הפונקציה איטרטיבית. משתמשת במצביע p. מתחילה מלהצביע על x. מעדכנת את שדה ה-size על ידי קריאה לפונקצייה p.setSize() של טיפוס מסוג TreeNode. מעדכנת את המצביע להצביע לאב של p ומבצעת שוב את אותה פעולה וכן הלאה. עוצרת כאשר הגיעה לשורש העץ.

**סיבוכיות**-

### Size()

**תיאור כללי**- הפונקציה מחזירה את מספר האיברים בעץ.

**עקרון מימוש**- המימוש מחזיר את getSize(root) של השורש שמהווה את גודל העץ כולו. שדה זה מתעדכן על ידי פעולות insert ו-delete.

**סיבוכיות**-

### split(int x)

**תיאור כללי**- הפונקציה מקבלת מפתח x שנמצא בעץ. על הפונקציה להפריד את העץ ל- 2 עצי AVL כאשר המפתחות של האחד גדולים מ- x ושל השני קטנים מ- x יש לממש את הפונקציה על פי המימוש שנלמד בהרצאה בסיבוכיות𝑂(log𝑛)

**עקרון מימוש**-

**סיבוכיות**-

### join(IAVLNode x, AVLtree t)

**תיאור כללי**- הפונקציה מקבלת צומת x ועץ t שכל ה keys- שלהם קטנים, או שכולם גדולים, מה keys- של העץ הנוכחי שביחס אליו קראנו ל- join . על הפונקציה לאחד את x,tלעץ הנוכחי כפי שמומש בהרצאה. על הפעולה לרוץ בזמן 𝑂(log𝑛) . על הפעולה להחזיר את העלות של פעולת ה- join )הפרש גבהי העצים+ 1 .)

**עקרון מימוש**-

**סיבוכיות**-

### getRoot()

**תיאור כללי**- הפונקציה מחזירה את השורש של העץ (אובייקט AVLNode )

**עקרון מימוש**- הרשימה תמיד מחזיקה מצביע לשורש, הפונקציה מחזירה את השורש בהתאם למצביע זה. פונקציית insert ו-delete מעדכנת את שדה זה במידת הצורך.

**סיבוכיות**-

## מחלקת AVLNode

### בנאי AVLNode

**תיאור כללי**- בנאי ליצירת אובייקט חדש מסוג AVLNode.

**עקרון מימוש**:

שדות האובייקט- אתחול בהתאם לערכי ברירת המחדל הבאים:

* left = null
* right = null
* parent = null
* rank = -1
* key = -1
* info = null
* size = 0

מימוש שני בנאים:

1. לצומת ווירטואלית- לא מקבל ערכים בקריאה. משאירים את ערכי ברירת המחדל של השדות.
2. לצומת אמיתית- מקבל את הערכים key ו-info ומעדכן בהתאם את שדות אלו. בנוסף, מעדכן שדה size להיות 1.

**סיבוכיות**-

### getKey()

**תיאור כללי**- מחזיר את המפתח של הצומת, או 1- אם הצומת הוא וירטואלי.

**עקרון מימוש**- מחזירה את השדה key (אם הצומת וירטואלית ה-key שלה שווה ל-(-1))

**סיבוכיות**-

### getValue()

**תיאור כללי**- מחזיר את ה info של הצומת או null אם הצומת הוא וירטואלי.

**עקרון מימוש**- מחזירה את השדה info (אם הצומת וירטואלית ה-info שלה הוא null).

**סיבוכיות**-

### setLeft(IAVLNode node)

***תיאור כללי****- מגדיר מצביע לבן השמאלי של הצומת.*

***עקרון מימוש****- מקבל את הצומת ומגדיר את שדה left להצביע לצומת זו.*

***סיבוכיות****-*

### getLeft(IAVLNode node)

**תיאור כללי**- מחזיר את הבן השמאלי של הצומת, או null אם אין כזה.

**עקרון מימוש**- בודקים את ערך ה-key של הבן השמאלי באמצעות שדה left שמצביע אליו. אם ה-key שונה מ-(-1) מחזירים את המצביע. אחרת, מחזירים null.

**סיבוכיות**-

### setRight(IAVLNode node)

***תיאור כללי****- מגדיר מצביע לבן הימני של הצומת.*

***עקרון מימוש****- מקבל את הצומת ומגדיר את שדה right להצביע לצומת זו.*

***סיבוכיות****-*

### getRight(IAVLNode node)

**תיאור כללי**- מחזיר את הבן הימני של הצומת, או null אם אין כזה.

**עקרון מימוש**- בודקים את ערך ה-key של הבן הימני באמצעות שדה right שמצביע אליו. אם ה-key שונה מ-(-1) מחזירים את המצביע. אחרת, מחזירים null.

**סיבוכיות**-

### setParent(IAVLNode node)

***תיאור כללי****- מגדיר את ההורה של הצומת.*

***עקרון מימוש****- מקבל את הצומת ומגדיר את שדה parent להצביע לצומת זו.*

***סיבוכיות****-*

### getParent()

***תיאור כללי****- מחזיר את ההורה של הצומת או null אם אין כזה (במקרה שהצומת היא השורש של העץ)*

***עקרון מימוש****-* בודקים את ערך ה-key של ההורה באמצעות שדה parent שמצביע אליו. אם ה-key שונה מ-(-1) מחזירים את המצביע. אחרת, מחזירים null.

***סיבוכיות****-*

### isRealNode()

**תיאור כללי**- מחזיר TRUE אם הצומת מייצג צומת אמיתי בעץ (קרי: צומת שאינו וירטואלי).

**עקרון מימוש**- בודקים את שדה ה-key של הצומת, אם הערך שונה מ-(-1) מחזירים TRUE, אחרת FALSE.

**סיבוכיות**-

### setHeight(int height)

***תיאור כללי****- מעדכן את גובה הצומת*

***עקרון מימוש****- משווה לשדה ה-rank של הצומת.*

***סיבוכיות****-*

### getHeight()

**תיאור כללי**- מחזיר את גובה הצומת, -1 עבור צומת וירטואלי. יש לממש בסיבוכיות O(1) .

**עקרון מימוש**- מחזיר את שדה height של הצומת.

**סיבוכיות**-

### getRank()

**תיאור כללי**- מחזיר את rank הצומת.

**עקרון מימוש**- מחזיר את שדה rank של הצומת.

**סיבוכיות**-

### setTempRank(int delta)

**תיאור כללי**- מאפשר להגדיר ערך ביניים ל-rank לטובת פעולות האיזון.

**עקרון מימוש**- מעדכן את שדה rank של הצומת לערך הנוכחי שלה + שינוי רצוי.

**סיבוכיות**-

### resetRank()

***תיאור כללי****- מעדכן את ה-rank של הצומת*

***עקרון מימוש****- מחשב*

***סיבוכיות****-*

### getSize()

**תיאור כללי**- מחזיר את גודל תת העץ שהצומת היא השורש שלו.

**עקרון מימוש**- מחזיר את שדה size של הצומת.

**סיבוכיות**-

### setSize()

**תיאור כללי**- מעדכן את גודל תת העץ שהצומת היא השורש שלו.

**עקרון מימוש**- מחבר 1 + getSize(getLeft()) + getSize(getRight())

**סיבוכיות**-