מבנה נתונים- תיעוד תרגיל מעשי 1

# חלק א- תיעוד פונקציות

## מחלקת AVLTree

### שדות AVLTree

אתחול בהתאם לערכי ברירת המחדל הבאים:

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

### empty()

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

**עקרון מימוש**- הפונקציה:

* קוראת ל-isRealNode() (פועלת ב-) על הצומת ששמורה בשורש העץ.
* אם היא 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**- הפונקציה מבצעת חיפוש בינארי בעץ:

* יוצרת מצביע p שתחילה מצביע לשורש העץ.
* בודקת אם המצביע לא בעלה ווירטואלי באמצעות הפונקציה isRealNode() ()
* בודקת אם k קטן, גדול או שווה למפתח של המצביע.
  + אם המפתח שווה מחזירה את הצומת.
  + אם קטן, מעדכנת את המצביע לבן השמאלי שלו.
  + אם גדול מעדכנת לבן הימני שלו.

האיטרציה ממשיכה כל עוד לא הגענו לצומת שהמפתח שווה ל-k או עד שהגענו לצומת שלא יכולנו לעשות צעד ימינה או שמאלה בהתאם לאם k גדול או קטן (הבן המתאים הוא virtualLeaf, בדקנו עם isRealNode() ב-).

אם נמצאה צומת ש-k שווה לערך בה מחזירים את הצומת. אחרת מחזירים את הצומת שהגענו אליה (ההורה של המקום המתאים ל-k).

**סיבוכיות**- (חיפוש בינארי)

### 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**- הפונקציה:

* קוראת ל-treePosition(k) (( למציאת המקום של k בחיפוש בינארי.
* 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:

* יוצרת משתנה בשם rebalanceActions מסוג int עם הערך 0.

אם העץ ריק (הפונקציה empty() מחזירה true ב-):

* 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 (כל פונקציה פועלת ב-):
  + setRight – שיצביע לעלה הווירטואלי
  + setLeft – שיצביע לעלה הווירטואלי
  + setParent – שיצביע לעלה הווירטואלי
  + setHight – שיהיה שווה ל-0
  + setSize – שיהיה שווה ל-1

אחרת:

* קוראת ל-treePosition(k) (שפועלת ב-) ושומרת במצביע בשם parent.
  + בודקת אם המפתח של הצומת שהוחזרה שווה ל-k (getKey פועלת ב-). אם כן אז הצומת כבר בעץ ולכן מחזירה (-1).
  + אחרת-
    - יוצרת אובייקט AVLNode חדש עם המפתח k והערך info ושומרת במצביע בשם child.
    - מעדכנת את שדות הצומת החדשה (כל פונקציה פועלת ב-):
      * getLeft – שיצביע לעלה הווירטואלי
      * setRight – שיצביע לעלה הווירטואלי
      * setParent – שיצביע לצומת ההורה
      * setHight – להיות שווה ל-0
      * setSize – שיהיה שווה ל-1
    - משווה בין המפתח של ה-parent ל-k (getKey פועלת ב-). 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 שפועלות ב-).
    - **עדכון המינימום-** משווה בין k למפתח ששמור בשדה min (getKey פועלת ב-). אם הוא קטן יותר- מעדכנת את min להצביע לצומת החדשה.
    - **עדכון המקסימום-** משווה בין המפתח של הצומת החדשה למפתח ששמור בשדה max (getKey פועלת ב-). אם הוא גדול יותר- מעדכנת את max להצביע לצומת החדשה.
    - **מאזנת את העץ-** קוראת ל- rebalance(parent)שמבצעת את פעולות האיזון של העץ וסופרת אותן, ומעדכנת את שדה ה-size בכל צומת רלוונטית (פועלת ב-). שומרת את הערך שהיא מחזירה ב- rebalanceActions.
* מחזירה את rebalanceActions

**סיבוכיות**- שנובע מפעולת ה-rebalance.

### 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) (שפועלת ב-) ושומרת במצביע בשם toDelete.
* בודקת אם המפתח של הצומת שהוחזרה שונה מ-k (getKey פועלת ב-). אם כן אז אין בעץ צומת עם המפתח k ולכן מחזירה (-1).
* בודקת אם בשדות min ו-max של העץ שמורה אותה הצומת. אם כן אז בעץ יש רק צומת בודדת ולכן נעדכן את שדות min ו-max להצביע ל-virtualLeaf.
* **עדכון המינימום-** קוראת ל- findSuccessor(min) שמחפשת את ה-successor ב-. מעדכנת את min להצביע לצומת המוחזרת.
* **עדכון המקסימום-** קוראת ל- findPredecessor(max) שמחפשת את ה-predecessor ב- . מעדכנת את max להצביע לצומת המוחזרת.
* יוצרת מצביע לצומת בשם startRebalance.
* בודקת אם toDelete עלה, צומת אונרית או צומת עם שני בנים. היא מפעילה את פונקציות getRight.isRealNode ו-getLeft.isRealNode שרצות ב- כל אחת.
  + אם toDelete זה עלה (שני הילדים הם עלים ווירטואליים) – קוראת ל- deleteLeaf(toDelete) שרצה ב- ושומרת את הצומת המוחזרת במשתנה startRebalance.
  + אם toDelete זו צומת אונרית – קוראת ל- deleteUnar(toDelete) שרצה ב- ושומרת את הצומת המוחזרת במשתנה startRebalance.
  + אחרת – קוראת ל- deleteDouble(toDelete) שרצה ב- ושומרת את הצומת המוחזרת במשתנה startRebalance.
* **מאזנת את העץ-** קוראת ל- rebalance(startRebalance) שמבצעת את פעולות האיזון של העץ וסופרת אותן, ומעדכנת את שדה ה-size בכל צומת רלוונטית (פועלת ב-) ושומרת את הערך שהיא מחזירה ב- rebalanceActions.
* מחזירה את rebalanceActions

**סיבוכיות**- שנובע מפעולת ה-rebalance.

### deleteLeaf(IAVLNode child)

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

**עקרון מימוש**- הפונקציה:

* שומרת את ההורה של child במשתנה parent עם קריאה ל- getParent(child) שרצה ב-.
* בודקת אם child מצביע לשורש העץ. אם כן אז משנה את המצביע של root להיות virtualLeaf (אם הצומת היא גם עלה וגם השורש- במחיקתה העץ הופך לריק).
* אחרת- בודקת אם child הוא בן ימני או בן שמאלי של parent (באמצעות getRight ו-getLeft שרצות ב-) ובהתאם מעדכנת את הבן הימני או השמאלי להיות צומת וירטואלית (באמצעות setRight או setLeft שרצות ב-).
* מחזירה את parent

**סיבוכיות**-

### deleteUnar(IAVLNode child)

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

**עקרון מימוש**- הפונקציה:

* שומרת את ההורה של child במשתנה parent עם קריאה ל- getParent(child) שרצה ב-.
* בודקת אם child מצביע לשורש העץ. אם כן אז משנה את המצביע של root להיות הבן היחיד של child ומעדכנת את ההורה של הבן הזה להיות עלה וירטואלי (באמצעות getRight, getLeft, isRealNode, setParent שרצות כל אחת ב-).
* אחרת
  + בודקת אם child הוא בן ימני או בן שמאלי של parent ובהתאם מעדכנת את הבן הימני או השמאלי של parent להיות הבן היחיד של child (באמצעות getRight, getLeft, isRealNode, setRight, setLeft, setParent שרצות כל אחת ב-)
* מחזירה את parent

**סיבוכיות**-

### deleteDouble(IAVLNode child)

**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(child) ושומרת את הערך המוחזר ב-successor (נשים לב ש- successor לא יכול להיות null כי זה אומר שזו צומת ה-max והיא בהכרח עלה או אונרית וטיפלנו במקרה הזה. מצד שני ראינו כי successor זו צומת אונרית או עלה בעצמה).
* שומרת את ההורה של successor במשתנה successorParent.
* בודקת:
  + אם successor עלה- קוראת ל- deleteLeaf(successor) שפועלת ב-
  + אם successor אונרית- קוראת ל- deleteUnar(successor) שפועלת ב-
* מעדכנת את השדות של successor להיות אלו של child באמצעות קריאה לפונקציות getRight, getLeft, getParent, setRight, setLeft, setParent, setHeight, getHeight שרצות כל אחת ב-.
* אם child הוא שורש העץ:
  + מעדכנת את שדה root להצביע ל- successor
  + אחרת- בהתאם לאם child הוא בן ימני או בן שמאלי מעדכנת את המצביע המתאים של ההורה של child להצביע ל- successor, באמצעות קריאה לפונקציות getRight, getLeft, setRight, setLeft שרצות כל אחת ב-.
* מחזירה את successorParent

**סיבוכיות**-

### rebalance(IAVLNode node, 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 שמבצעת פעולת איזון לוקלית בהתאם לאלגוריתם שנלמד בכיתה. שומרת ספירה של כמות פעולות האיזון שביצענו במשתנה countActions. מעדכנת תוך כדי את שדה size של הצמתים מהצומת שהוכנסה ועד השורש.

* תנאי עצירה:
  + אם ההורה עלה ווירטואלי זה אומר שהגענו לשורש העץ ותפסיק הקריאה הרקורסיבית. במקרה הזה סיימנו לאזן את העץ ולעדכן את שדות size של הצמתים הרלוונטיים ונחזיר 0.
* מחשבת את הפרש ה-rank בין הצומת לכל אחד מהבנים שלה באמצעות קריאה לפונקציות getHeight, getLeft, getRight שרצות כל אחת ב-.
* אם הפרש ה-rank בין הצומת לבנים שלה הוא מהצורה (1,1), (1,2) או (2,1)
  + מעדכנת את שדה ה-size של node באמצעות setSize שרצה ב-.
  + קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
* אם הפרש ה-rank בין הצומת לבנים שלה הוא מהצורה (0,1) או (1,0):
  + קוראת ל-promote על node שרצה ב- ומוסיפה את הערך המוחזר ל- countActions.
  + מעדכנת את שדה ה-size של node באמצעות setSize שרצה ב-.
  + קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
* אם הפרש ה-rank בין הצומת לבנים שלה הוא מהצורה (2,2)
  + קוראת ל- demoteעל node שרצה ב- ומוסיפה את הערך המוחזר ל- countActions.
  + מעדכנת את שדה ה-size של node באמצעות setSize שרצה ב-.
  + קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
* אם הפרש ה-rank בין הצומת לבנים שלה הוא מהצורה (0,2)
  + אם הפרש ה-rank בין הבן השמאלי לבנים שלו הוא מהצורה (1,1) (רלוונטי ל-join)
    - מבצעת סיבוב ימינה ואז promote לבן השמאלי שרצות כל אחת ב-, ומוסיפה את הערך שמחזירות ל- countActions.
    - מעדכנת את שדה ה-size של node ושל הבן השמאלי באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
  + אם הפרש ה-rank בין הבן השמאלי לבנים שלו הוא מהצורה (1,2)
    - מבצעת סיבוב ימינה ואז demote ל-node שרצות כל אחת ב-, ומוסיפה את הערך שמחזירות ל- countActions.
    - מעדכנת את שדה ה-size של node ושל הבן השמאלי באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
  + אם הפרש ה-rank בין הבן השמאלי לבנים שלו הוא מהצורה (2,1)
    - מבצעת סיבוב כפול ימינה, demote ל-node ולבן השמאלי, ו-promote להורה החדש של הבן השמאלי (במקור היה הבן הימני של הבן השמאלי). כל פעולה רצה ב-. מוסיפה את הערך שמחזירות ל- countActions
    - מעדכנת את שדה ה-size של node, של הבן השמאלי ושל ההורה החדש של הבן השמאלי (במקור היה הבן הימני של הבן השמאלי) באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
* אם הפרש ה-rank בין הצומת לבנים שלה הוא מהצורה (2,0)
  + אם הפרש ה-rank בין הבן הימני לבנים שלו הוא מהצורה (1,1) (רלוונטי ל-join)
    - מבצעת סיבוב שמאלה ואז promote לבן הימני שרצות כל אחת ב-, ומוסיפה את הערך שמחזירות ל-countActions.
    - מעדכנת את שדה ה-size של node ושל הבן הימני באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
  + אם הפרש ה-rank בין הבן הימני לבנים שלו הוא מהצורה (2,1)
    - מבצעת סיבוב שמאלה ואז demote ל-node שרצות כל אחת ב-, ומוסיפה את הערך שמחזירות ל-countActions.
    - מעדכנת את שדה ה-size של node ושל הבן הימני באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
  + אם הפרש ה-rank בין הבן הימני לבנים שלו הוא מהצורה (1,2)
    - מבצעת סיבוב כפול שמאלה, demote ל-node ולבן הימני, ו-promote להורה החדש של הבן הימני (במקור היה הבן השמאלי של הבן הימני). כל פעולה רצה ב-. מוסיפה את הערך שמחזירות ל- countActions
    - מעדכנת את שדה ה-size של node, של הבן הימני ושל ההורה החדש של הבן הימני (במקור היה הבן השמאלי של הבן הימני) באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
* אם הפרש ה-rank בין הצומת לבנים שלה הוא מהצורה (3,1)
  + אם הפרש ה-rank בין הבן הימני לבנים שלו הוא מהצורה (1,1)
    - מבצעת סיבוב שמאלה, demote ל-node ו-promote לבן הימני. כל פעולה רצה ב-. מוסיפה את הערך שמחזירות ל-countActions.
    - מעדכנת את שדה ה-size של node ושל הבן הימני באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
  + אם הפרש ה-rank בין הבן הימני לבנים שלו הוא מהצורה (2,1)
    - מבצעת סיבוב שמאלה, ו-demote ל-node פעמיים. כל פעולה רצה ב-. מוסיפה את הערך שמחזירות ל-countActions.
    - מעדכנת את שדה ה-size של node ושל הבן הימני באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
  + אם הפרש ה-rank בין הבן הימני לבנים שלו הוא מהצורה (1,2)
    - מבצעת סיבוב כפול שמאלה, demote ל-node פעמיים, demote לבן הימני, ו-promote להורה החדש של הבן הימני (במקור היה הבן השמאלי של הבן הימני). כל פעולה רצה ב-. מוסיפה את הערך שמחזירות ל- countActions
    - מעדכנת את שדה ה-size של node, של הבן הימני ושל ההורה החדש של הבן הימני (במקור היה הבן השמאלי של הבן הימני) באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
* אם הפרש ה-rank בין הצומת לבנים שלה הוא מהצורה (1,3)
  + אם הפרש ה-rank בין הבן השמאלי לבנים שלו הוא מהצורה (1,1)
    - מבצעת סיבוב ימינה, demote ל-node ו-promote לבן השמאלי. כל פעולה רצה ב-. מוסיפה את הערך שמחזירות ל-countActions.
    - מעדכנת את שדה ה-size של node ושל הבן השמאלי באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
  + אם הפרש ה-rank בין הבן השמאלי לבנים שלו הוא מהצורה (1,2)
    - מבצעת סיבוב ימינה, ו-demote ל-node פעמיים. כל פעולה רצה ב-. מוסיפה את הערך שמחזירות ל-countActions.
    - מעדכנת את שדה ה-size של node ושל הבן השמאלי באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
  + אם הפרש ה-rank בין הבן השמאלי לבנים שלו הוא מהצורה (2,1)
    - מבצעת סיבוב כפול ימינה, demote ל-node פעמיים, demote לבן השמאלי, ו-promote להורה החדש של הבן השמאלי (במקור היה הבן הימני של הבן השמאלי). כל פעולה רצה ב-. מוסיפה את הערך שמחזירות ל- countActions
    - מעדכנת את שדה ה-size של node, של הבן השמאלי ושל ההורה החדש של הבן השמאלי (במקור היה הבן הימני של הבן השמאלי) באמצעות setSize שרצה ב-.
    - קוראת רקורסיבית ל-rebalance על ההורה של node (getParent רצה ב-( ומוסיפה את הערך המוחזר ל- countActions.
* מאפסת את המצביעים לבנים של virtualLeaf שיצביעו ל-null (באמצעות setRight, setLeft שרצות ב-(
* מחזירה את countActions

**סיבוכיות**-

### promote(IAVLNode node)

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

**עקרון מימוש**- מקבלת צומת ומגדילה את ה-height שלה ב-1 באמצעות הפונקציות getHeight ו-setHeight שרצות כל אחת ב-. מחזירה 1 שזה כמות פעולות האיזון שביצעה.

**סיבוכיות**-

### demote(IAVLNode node)

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

**עקרון מימוש**- מקבלת צומת ומקטינה את ה-height שלה ב-1 באמצעות הפונקציות getHeight ו-setHeight שרצות כל אחת ב-. מחזירה 1 שזה כמות פעולות האיזון שביצעה.

**סיבוכיות**-

### rotateRight(IAVLNode node, IAVLNode left)

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

**עקרון מימוש**- node 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

### 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**-