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
* min – מצביע לצומת שבה הערך המינימלי
* max – מצביע לצומת שבה הערך המקסימלי
* virtualNode – צומת ווירטואלית אליה יצביעו עלים וצמתים אונריים.

**סיבוכיות**-

### empty()

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

**עקרון מימוש**- הפונקציה מקבלת AVLNode, אם הוא null מחזירה true, אחרת מחזירה false.

**סיבוכיות**-

### treePosition(int k)

**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, מקבלת תחילה את השורש ובודקת אם k קטן, גדול או שווה לערך בשורש. אם המפתח שווה מחזירה את הערך בצומת. אם קטן, מתקדמת שמאלה ואם גדול מתקדמת ימינה וקוראת לבן המתאים ברקורסיה. אם מצאנו צומת ש-k שווה לערך בה מחזירים את הצומת. אם הגיעה ל-null מחזירה את ההורה.

**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 (ה-root == null):

* 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 להיות שווים ל-k
* מעדכנת את שדות הצומת החדשה:
  + getLeft – 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 השמאלי של parent להצביע ל-child. אם k קטן מהמפתח הזה מגדירה שהבן הימני 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 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 השמאלי של parent להצביע ל-child. אם k קטן מהמפתח הזה מגדירה שהבן הימני 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

### 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 x.parent = null זה אומר שהגענו לשורש העץ ותפסיק הקריאה הרקורסיבית. במקרה הזה סיימנו לאזן את העץ ונחזיר גם 0
* בודקת האם הפרש ה-rank בין השורש הבנים מהצורה (0,1) או (1,0), אם כן אז קוראת ל-promote. לאחר מכן קוראת רקורסיבית ל-rebalance על x.parent.
* 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 ל- rotateRight(x, y) ומחזירה את הערך שהוא מחזיר + countActions
  + אם הבן y עם ההפרש 0 בין ה-rank הוא צומת מסוג (2,1) – קוראת ל- doubleRotateRight(x, y) ומחזירה את הערך שהוא מחזיר + 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 ל- rotateLeft(x, y) ומחזירה את הערך שהוא מחזיר + countActions
  + אם הבן y עם ההפרש 0 בין ה-rank הוא צומת מסוג (1,2) – קוראת ל- doubleRotateLeft(x, y) ומחזירה את הערך שהוא מחזיר + countActions
* מחזירה את countActions

**סיבוכיות**-

### promote(IAVLNode x)

**תיאור כללי**- הפונקציה מבצעת את פעולה 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**-

### 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.

* מעדכנים את x.left = y.right
* מעדכנים את x.left.parent = x
* מעדכנים את y.right = x
* מעדכנים את x.parent = y
* מעדכנים את ה-rank של x להיות rank-1
* אם x מצביע לאותה צומת כמו ה-root (שורש העץ)- נעדכן את השורש של העץ להצביע ל-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.

* מעדכנים את x.right = y.left
* מעדכנים את x.right.parent = x
* מעדכנים את y.left = x
* מעדכנים את x.parent = y
* מעדכנים את ה-rank של x להיות rank-1
* אם x מצביע לאותה צומת כמו ה-root (שורש העץ)- נעדכן את השורש של העץ להצביע ל-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.

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

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

**סיבוכיות**-

### successor(IAVLNode x)

**תיאור כללי**-

**עקרון מימוש**-

**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. מתחילה מ-x ומעדכנת את שדה ה-size על ידי חיבור שלsize(x.left) + size(x.right) +1. מעדכנת את המצביע להצביע לאב של x 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 שדה size 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 עציAVL 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**- אם הצומת ווירטואלית מחזירה (-1), אחרת מחזירה את השדה key.

**סיבוכיות**-

### getValue()

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

**עקרון מימוש**- אם הצומת null מחזירה null, אחרת מחזירה את השדה info.

**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()

**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()

**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****- מחשב*

***סיבוכיות****-*

getHeight()

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

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

**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 + left.size + right.size

**סיבוכיות**-