מבני נתונים – תרגיל מעשי 1

עץ WAVL

מגישים: אופיר פפר – 203565833. שם משתמש במודל: ofirfeffer

אילור יפרח – 205828478. שם משתמש במודל: ilorifrach

מחלקת WAVLNode

המחלקה מייצגת צומת בודד בעץ בינארי.

שדות המחלקה:

int key – המפתח של הצומת. מספר שלם.

String info – המידע שמחזיק הצומת. מחרוזת.

int rank – דרגת הצומת. מספר שלם גדול או שווה ל0.

WAVLNode parent – מצביע להורה של הצומת בעץ. Null במקרה שהצומת הוא השורש.

WAVLNode rightChild – מצביע לבן הימני של הצומת בעץ. Null במקרה שאין בן ימני.

WAVLNode leftChild– מצביע לבן השמאלי של הצומת. Null במקרה שאין בן שמאלי.

מתודות:

1. Getter לכל אחד מן השדות:
   - **Integer getKey()** - מחזירה את המפתח (מספר).
   - **String getInfo()** - מחזירה את המידע (מחרוזת).
   - **Integer getRank()** - מחזירה את הדרגה (מספר).
   - **WAVLNode getParent()** - מחזירה את ההורה (צומת).
   - **WAVLNode getRightChild()** - מחזירה את הבן הימני (צומת).
   - **WAVLNode getLeftChild()** - מחזירה את הבן השמאלי (צומת).

   סיבוכיות כל ה Getters - O(1).

2. **void setRightChild(WAVLNode rightChild)** – הפונקציה מעדכנת את הבן הימני של הצומת ובנוסף מעדכנת את ההורה של אותו בן ימני להיות הצומת הנוכחי.
   סיבוכיות: O(1).
3. **void setLeftChild(WAVLNode leftChild)** – הפונקציה מעדכנת את הבן השמאלי של הצומת ובנוסף מעדכנת את ההורה של אותו בן שמאלי להיות הצומת הנוכחי.
   סיבוכיות: O(1).

Enum NodeType

מייצג סוג של צומת בעץ מתוך 4 סוגים אפשריים: עלה, צומת אונארי עם בן ימני, צומת אונארי עם בן שמאלי, צומת בינארי.

מתודות:

1. **NodeType of(WAVLNode node)** – המתודה מקבלת צומת בעץ ומחזירה טיפוס מסוג NodeType המייצג את סוג הצומת.
   סיבוכיות: O(1)

Enum RankDiff

מייצג את הפרש הדרגות של צומת בינו לבין הבן השמאלי שלו ובינו לבין הבן הימני שלו. שמו של כל constant בenum זה הוא מהצורה Dx_y כאשר x מבטא את הפרש הדרגות בין צומת לבנו השמאלי ו-y מבטא את הפרש הדרגות בין צומת לבנו הימני.

הפרשי הדרגות האפשריים (חוקיים ולא חוקיים):

D0_1, D0_2, D1_0, D1_1, D1_2, D1_3, D2_0, D2_1, D2_2, D2_3, D3_1, D3_2.

מתודות:

1. **RankDiff of(WAVLNode node)** – המתודה מקבלת צומת בעץ ומחזירה טיפוס מסוג RankDiff המייצג את הפרש הדרגות של הצומת.
   סיבוכיות: O(1)

מחלקת WAVLTree

המחלקה מתארת עץ חיפוש בינארי מאוזן מסוג WAVLTree עם מנגנון איזון במעמד ההכנסה וההוצאה של ערכים.

חוקיות העץ:

1. שורש העץ הוא null אם ורק אם גודל העץ הוא 0, כלומר העץ ריק.
2. צומת בעץ שהוא עלה דרגתו 0.
3. הפרש הדרגות של צומת בעץ מכל אחד מבניו הוא לכל הפחות 1 ולכל היותר 2.
4. לכל צומת בעץ מתקיים שהמפתח של בנו הימני, אם קיים כזה, גדול מן המפתח של הצומת.
5. לכל צומת בעץ מתקיים שהמפתח של בנו השמאלי, אם קיים כזה, קטן מן המפתח של הצומת.

שדות המחלקה:

WAVLNode root – שורש העץ.

int size – גודל העץ.

WAVLNode min – מצביע לצומת המכיל את המפתח המינימלי בעץ.

WAVLNode max - מצביע לצומת המכיל את המפתח המקסימלי בעץ.

int[] sortedKeys – מערך של המפתחות בעץ.

String[] sortedInfo – מערך של המידע בעץ.

מתודות:

1. **WAVLTree()** – בנאי העץ. מאתחל את כל השדות לnull ואת גודל העץ ל0.
   סיבוכיות: O(1)

2. **void setRoot(WAVLNode root)** – מתודה המקבלת כפרמטר צומת ומשנה את השורש להיות צומת זה. בנוסף, מאתחלת את ההורה של השורש להיות null.
   סיבוכיות: O(1)

3. **boolean empty()** – מתודה המחזירה true אם ורק אם העץ ריק. עושה זאת ע"י בדיקה אם גודל העץ הוא 0.
   סיבוכיות: O(1)

4. **String search(int k)** – מתודה המקבלת מספר שלם k, מחפשת צומת בעץ עם המפתח k ומחזירה את המידע שמחזיק הצומת (את ה info). המתודה מפעילה את פונקציית העזר search(int k, WAVLNode node) עם המספר k ושורש העץ וזו מבצעת את הפונקציונליות הנדרשת.
   סיבוכיות: O(log n)

5. **WAVLNode search(int k, WAVLNode node)** – מתודה המקבלת מספר שלם k וצומת בעץ שהוא שורש העץ או שורש של תת-עץ. המתודה תחזיר צומת בעץ המכיל את המפתח k, או null אם אין צומת כזה בתת העץ הנתון. אופן פעולה: ע"י לולאת while נטייל בעץ. אם המפתח k קטן מהמפתח של הצומת בו אנו נמצאים, נעבור לבן השמאלי של הצומת. אם המפתח k גדול מהמפתח של הצומת בו אנו נמצאים, נעבור לבין הימני של הצומת. אם המפתח k שווה למפתח של הצומת בו אנו נמצאים, נחזיר צומת זה. אם הגענו ל external node (null) המפתח לא קיים בעץ ונחזיר null.
סיבוכיות: O(log n)

6. **int insert(int k, String i)** - המתודה מקבלת מפתח ומחרוזת מידע ויוצרת צומת חדש עם ערכים אלו. המתודה מכניסה את הצומת החדש לעץ באם לא קיים בעץ מפתח זהה ומחזירה את מספר מהלכי האיזון שהתבצעו עקב ההכנסה.
משום שאנו מתחזקים מצביעים לאיבר המינימלי והאיבר המקסימלי, המתודה מעדכנת את המינימום ו/או המקסימום בהתאם ע"י קריאה למתודה updateMinMaxOnInsert(WAVLNode node).
אם העץ ריק, השורש מאותחל להיות הצומת החדש שנוצר. והמתודה תחזיר שמספר מהלכי האיזון הוא 0.
בנוסף, גודל העץ גדל ב1 כי נוצר צומת חדש, לכן המתודה מעדכנת את שדה גודל העץ בהתאם.
אם העץ אינו ריק, **המתודה תחפש את מקום ההכנסה לצומת החדש כך שלא תופר חוקיות העץ הבינארי. חיפוש זה נעשה בלולאת while המטיילת בצמתי בעץ:**
אם המפתח של הצומת החדש זהה למפתח הצומת הנוכחי, אזי המפתח קיים כבר בעץ. במקרה זה נחזיר 1-.
נמשיך בחיפוש מקום ההכנסה לפי תכונות העץ הבינארי - כל צומת קטן מהבן הימני שלו וגדול מהבן השמאלי שלו:
בכל פעם שאנו מסתכלים על צומת מסוים נבדוק האם המפתח החדש גדול מהמפתח של הצומת הנוכחי, אם כן, נרד ימינה בעץ (לבן הימני של הצומת הנוכחי). אם לא, נרד שמאלה בעץ (לבן השמאלי של הצומת הנוכחי).
אם הגענו לצומת שהמפתח שלו קטן מהמפתח של הצומת החדש, והבן הימני שלו null, נכניס את הצומת החדש כבן הימני של הצומת הנ"ל.
אם הגענו לצומת שהמפתח שלו גדול מהמפתח של הצומת החדש, והבן השמאלי שלו null, נכניס את הצומת החדש כבן השמאלי של הצומת הנ"ל.
לאחר הכנסת האיבר החדש במקום המתאים, יתכן שהופרה חוקיות עץ הWAVL ,לכן נריץ מתודת איזון עץ לאחר הכנסה - rebalanceAfterInsertion(WAVLNode node).
מתודת האיזון תחזיר את מספר מהלכי האיזון שעשתה, ואת מספר זה נחזיר לבסוף.
סיבוכיות: O(log n).

7. **int rebalanceAfterInsertion(WAVLNode node)** – מתודת האיזון מקבלת צומת שהוא ההורה של האיבר החדש שהכנסנו ומבצעת תהליכי איזון על מנת שעץ הWAVL יהיה תקין ויקיים את כל תכונות העץ. מתודה זו מחזירה את מספר פעולות האיזון שעשתה.

באם הצומת שקיבלה המתודה הוא null נחזיר שמספר פעולות האיזון הוא 0 (סיימנו את התהליך האיזון כי הגענו לשורש).

נתחזק משתנה (prev) אשר שומר את הפרש הדרגות של הצומת הקודם בו ביקרנו בתהליך האיזון. תחילה הצומת node הוא האבא של הצומת החדש שהוכנס. משום שהאיבר שהוכנס תמיד יהיה עלה, Prev מאותחל להיות צומת 1_1 (דרגת עלה היא 0 ושני בניו הם עלים חיצוניים שדרגתם היא 1-, לכן הפרשי הדרגות של עלה זה 1_1).

נתחזק משתנה counter אשר סופר את כמות מהלכי האיזון.

המתודה תטייל למעלה בעץ ותבצע פעולות איזון בהתאם למקרים השונים. האיזון יסתיים בכמה מקרים: הגעה לשורש העץ, ביצוע פעולה טרמינלית (rotation) או הגעה לצומת שאינו מפר את חוקיות העץ.

- **מקרה ראשון -** צומת 1_0. במקרה זה נעשה promotion – הגדלת דרגת הצומת ב 1. מהלך שאינו סופי (אינו מסיים תהליך איזון), לכן ייתכן שהבעיה עלתה למעלה. נשנה את prev להיות 2_1 שהוא הפרש הדרגות הנוכחי של הצומת לאחר הpromotion.
- **מקרה שני -** צומת 0_1. סימטרי למקרה הראשון. נשנה את prev להיות 1_2.
- **מקרה שלישי -** צומת 2_0.

אם בנו של צומת 2_0 שאנו שומרים במשתנה prev הוא צומת 2_1 נעשה סיבוב ימינה. פעולת סיבוב מהווה מהלך איזון 1 ולכן נגדיל את המשתנה הסופר ב1.

סיבוב הוא פעולה טרמינלית - כלומר לאחריה העץ תקין וניתן להפסיק את תהליך האיזון. נחזיר את כמות מהלכי האיזון שהתבצעו.

אחרת, בנו של צומת 2_0 שאנו שומרים במשתנה prev הוא צומת 1_2 ונעשה סיבוב כפול - סיבוב שמאלה של הבן השמאלי ואז עוד סיבוב ימינה של הצומת. פעולת סיבוב כפול מהווה 2 מהלכי איזון ולכן נגדיל את המשתנה הסופר ב2.

סיבוב כפול הוא פעולה טרמינלית - כלומר לאחריה העץ תקין וניתן להפסיק את תהליך האיזון. נחזיר את כמות מהלכי האיזון שהתבצעו.

- **מקרה רביעי-** צומת 2_0. סימטרי למקרה הקודם. במקרה זה בהכרח יתבצע סיבוב או סיבוב כפול שהן פעולות טרמינליות שלאחריהן יסתיים תהליך האיזון ונחזיר את מספר פעולות האיזון.
- **מקרה חמישי -** צומת 1_1.

מקרה תקין שאינו מפר את תכונות עץ WAVL. לכן נחזיר את המשתנה הסופר מהלכי איזון.

סיבוכיות worst case: O(log n)

סיבוכיות amortized: O(1)

8. **void updateMinMaxOnInsertion(WAVLNode node)** - מתודה המקבלת צומת, שהוא צומת חדש בתהליך הכנסה לעץ ומעדכנת את המינימום ו/או המקסימום של העץ בהתאם.

המתודה משווה בין המפתח של האיבר החדש לאיבר המינימלי ששמור כשדה במחלקה.
באופן דומה, היא משווה בין המפתח של האיבר החדש לאיבר המקסימלי ששמור כשדה במחלקה.

- אם המפתח של הצומת החדש (הצומת שקיבלה המתודה כפרמטר) קטן מהמפתח של המינימום ששמור כרגע, נשנה את המינימום להיות האיבר החדש.
- אם המפתח של הצומת החדש (הצומת שקיבלה המתודה כפרמטר) גדול מהמפתח של המקסימום ששמור כרגע, נשנה את המקסימום להיות האיבר החדש.
- אם המינימום/המקסימום ששמור לנו עד כה הוא null, העץ בהכרח ריק ולכן האיבר החדש הוא בהכרח גם המינימלי וגם המקסימלי. נשנה את המינימום והמקסימום להיות האיבר החדש.

אחרת, לא נשנה דבר.

סיבוכיות: O(1)

9. **int delete(int k)** – מתודה המקבלת מספר שלם k ומטרתה למחוק צומת בעץ שהמפתח שלו הוא k. לבסוף תחזיר המתודה את מספר פעולות האיזון שנדרשו לאחר המחיקה.

תחילה, המתודה תקרא לפונקציה search על מנת למצוא את הצומת בעץ בעל המפתח k.
אם לא נמצא צומת כזה, תחזיר המתודה 1-. אחרת, תקרא לפונקציית updateMinMaxOnDeletion שזו תעדכן את המינימום והמקסימום באם יש צורך בכך ולבסוף תקרא לפונקציית delete עם הצומת שנמצא (הצומת בעץ בעל מפתח k).

סיבוכיות: O(log n)

10. **int delete(WAVLNode node, boolean isLeftChild)** – המתודה מקבלת צומת בעץ ומשתנה בוליאני שערכו אמת אם הצומת הוא הבן השמאלי של ההורה שלו. המתודה מוחקת את הצומת מהעץ ומחזירה את מספר מהלכי האיזון שהתבצעו לאחר המחיקה בעץ.

המתודה פועלת שונה במחיקה של סוגי צמתים שונים ולכן מחולקת למקרים –

- **מקרה ראשון** - עלה.

במקרה זה נפעיל את המתודה - deleteLeafOrUnaryNode וזו תמחק את הצומת מהעץ.
לאחר מכן נפעיל מתודה שמבצעת איזון לאחר מחיקה - rebalanceAfterDeletion(node.parent). ניתן למתודה את ההורה של הצומת והיא תחזיר את מספר מהלכי האיזון שביצעה אותו נשמור במשתנה res שיוחזר לבסוף במתודה.

- **מקרה שני** - צומת אונארי בעל בן שמאלי.

במקרה זה נפעיל את המתודה deleteLeafOrUnaryNode עם בנו השמאלי של הצומת וזו תמחק את הצומת מהעץ.

לאחר מכן נפעיל באופן דומה את המתודה שמבצעת איזון לאחר מחיקה - הפעם ניתן למתודה את הבן השמאלי של הצומת (קיים כזה כי מדובר בצומת אונארי).

- **מקרה שלישי** - צומת אונארי בעל בן ימני.

המקרה הסימטרי למקרה השני.

- **מקרה רביעי** - צומת בינארי (בעל 2 ילדים).

במקרה זה נפעיל את המתודה deleteNodeWithTwoChildren אשר מוחקת צומת בינארי מהעץ. המתודה הנ"ל מחזירה את הצומת ממנו יש להתחיל את תהליך האיזון (ישנן 2 אפשרויות), נשמור צומת זה.

לאחר מכן נפעיל את מתודת האיזון לאחר מחיקה - הפעם ניתן למתודה את הצומת ששמרנו.

סיבוכיות: O(log n)

11. **void deleteLeafOrUnaryNode(WAVLNode parent, WAVLNode childOfChild, boolean isLeftChild)** – מטרת המתודה למחוק מן העץ צומת שהוא עלה או צומת עם בן אחד בלבד. המתודה מקבלת את ההורה של אותו צומת (null אם מדובר במחיקת השורש), את בנו של הצומת (null אם מדובר בעלה) ומשתנה בוליאני שערכו אמת אם הצומת אותו אנו מוחקים הוא בנו השמאלי של אביו. אופן הפעולה: אם ההורה הוא null, אזי מדובר במחיקת שורש העץ ולכן נגדיר את שורש העץ להיות הבן היחיד של הצומת (או null אם אין כזה). אחרת, אם הצומת שאנו מוחקים הוא בן שמאלי של אבא שלו, נגדיר את בנו השמאלי של ההורה להיות בנו היחיד של הצומת (או null אם אין כזה). אחרת, נגדיר את בנו הימני של ההורה להיות בנו היחיד של הצומת (או null אם אין כזה).

סיבוכיות: O(1)

12. **WAVLNode deleteNodeWithTwoChildren(WAVLNode node, boolean isLeftChild)** - המתודה מקבלת צומת ומשתנה בוליאני שערכו אמת אם הצומת הוא הבן השמאלי של ההורה שלו. מטרת המתודה היא למחוק את הצומת מן העץ ע"י החלפת הצומת עם העוקב שלו.

המתודה מוצאת את העוקב של הצומת ע"י הפעלת פונקציית min על תת העץ הימני של הצומת (העוקב של צומת הוא האיבר הכי שמאלי בתת העץ ששורשו הוא הבן הימני של הצומת, לכן זה שקול למציאת מינימום בתת עץ זה).

מרגע שנמצא העוקב ההחלפה מתבצעת בכמה שלבים:

א. תחילה, מנותק העוקב מההורה שלו. אם לעוקב היה בן ימני (לא יכול להיות לו בן שמאלי) אז הוא יוגדר כבן שמאלי של האב (בדומה למחיקת צומת אונארי או עלה).

ב. מגדירים את ההורה של הצומת הנמחק כהורה של העוקב. בכך למעשה נמחק הצומת.

ג. מעדכנים את בניו של העוקב להיות בניו של הצומת הנמחק.

לבסוף המתודה מחזירה את ההורה של הצומת העוקב לפני ההחלפה, או את העוקב עצמו במידה ואותו הורה הוא הצומת אותו אנו מוחקים.

סיבוכיות: O(1).

13. **int rebalanceAfterDeletion(WAVLNode node)** – מתודה שמטרתה לאזן את העץ לאחר פעולת מחיקה. המתודה מקבלת צומת ממנו נתחיל סריקה כלפי מעלה בעץ ונבצע פעולות איזון. המתודה תחזיר את מספר פעולות האיזון שביצעה. אופן פעולה:

א. אם הצומת שהתקבל הוא null הרי שהגענו לשורש ולכן העץ מאוזן. נחזיר 0.

ב. אם הצומת שהתקבל הוא עלה והפרשי הדרגות שלו הם 2_2 נבצע פעולת demotion ונמשיך את האיזון כלפי מעלה.

ג. נטייל בעץ כלפי מעלה ונבצע פעולות איזון שונות בהתאם למקרים השונים:

a. מקרה 1 – הפרשי הדרגות של הצומת הם 2_3 או 3_2. במקרה זה נעשה demotion לצומת – נוריד את דרגתו ב-1.

b. מקרה 2 – הפרשי הדרגות של הצומת הם 3_1:

i. אם הפרשי הדרגות של הבן הימני הם 2_2 נבצע double demotion, כלומר נוריד את דרגת הצומת ואת דרגת הבן הימני ב-1.

ii. אם הפרשי הדרגות של הבן הימני הם 1_1 או 2_1 נבצע פעולת סיבוב לשמאל של הצומת. פעולה זו היא טרמינלית, כלומר לאחריה העץ מאוזן ולכן נחזיר את מספר פעולות האיזון.

iii. אחרת, נבצע פעולת סיבוב כפול ימינה ואז שמאלה. פעולה זו היא טרמינלית, כלומר לאחריה העץ מאוזן ולכן נחזיר את מספר פעולות האיזון.

c. מקרה 3 – הפרשי הדרגות של הצומת הם 3_1: מקרה סימטרי למקרה הקודם.

ד. האיזון מסתיים באחת מהפעולות הטרמינליות או בהגעה לשורש.

סיבוכיות worst case: O(log n)

סיבוכיות amortized: O(1)

14. **void updateMinMaxOnDeletion(int k)** -

מתודה המקבלת מפתח k שהוא המפתח של הצומת אותו מחקנו ומעדכנת את המינימום ו/או המקסימום בהתאם.

יש לעדכן רק אם מחקנו את המינימום/המקסימום של העץ.

המתודה מחולקת למקרים בהתאם לסוג הצומת שנמחק:

- מקרה ראשון – הצומת שנמחק הוא המינימום והמינימום עלה.

במקרה זה משום שמדובר בעץ בינארי מאוזן, נעדכן את המינימום להיות ההורה של האיבר שנמחק.

- מקרה שני - הצומת שנמחק הוא המינימום והמינימום הוא צומת אונארי בעל בן ימני (לא יתכן שהוא צומת אונארי בעל בן שמאלי כי אז המינימום היה הבן השמאלי שלו).

במקרה זה נעדכן את המינימום להיות הבן הימני של האיבר שנמחק. הבן הימני הוא בהכרח המינימום משום שמדובר בעץ מאוזן. במחיקת מינימום ידוע שהמינימום החדש הוא העוקב שלו, כלומר הבן הכי שמאלי של בנו הימני. בגלל שהעץ מאוזן ניתן להניח שלבן הימני אין בן שמאלי, אחרת העץ לא היה מאוזן. לכן המינימום החדש הוא הבן הימני.

- מקרה שלישי - הצומת שנמחק הוא מקסימום והמקסימום הוא עלה.
  במקרה זה נעדכן את המקסימום להיות ההורה של האיבר שנמחק.
- מקרה רביעי - הצומת שנמחק הוא המקסימום והמקסימום הוא צומת אונארי בעל בן שמאלי (לא יתכן שהוא צומת אונארי בעל בן ימני כי אז המקסימום היה הבן הימני שלו). במקרה זה נעדכן את המקסימום להיות הבן השמאלי של האיבר שנמחק. זה נכון מאותה סיבה כמו במקרה השני.

סיבוכיות: O(1)

15. **String min()** – המתודה מחזירה את הinfo של הצומת המוחזק בשדה המינימום. אם שדה זה הוא null תחזיר המתודה null.
    סיבוכיות: O(1)

16. **WAVLNode min(WAVLNode node)** - מתודה המקבלת צומת שהוא שורש של תת-עץ ומחזירה את הצומת המינימלי בתת העץ. מתכונות העץ האיבר המינימלי יהיה האיבר השמאלי ביותר בעץ. המתודה מגיעה לאיבר זה ע"י טיול שמאלה בעץ עד אשר מגיעה לצומת שבנו השמאלי הוא null ומחזירה את צומת זה.
    סיבוכיות: O(log n)
    חשוב לציין כי השימוש בפונקציה זו הוא לצורכי מציאת העוקב של צומת בתהליך מחיקה. את המינימום והמקסימום של העץ כולו אנו מתחזקים במעמד ההכנסה וההוצאה בסיבוכיות O(1).

17. **String max()** - המתודה מחזירה את הinfo של הצומת המוחזק בשדה המקסימום. אם שדה זה הוא null תחזיר המתודה null.
    סיבוכיות: O(1)

18. **int[] keysToArray()** - מתודה המחזירה מערך המכיל את כל מפתחות העץ בסדר ממויין. אם המתודה יצרה כבר מערך כזה והעץ לא השתנה מאז, מערך זה מוחזק בשדה sortedKeys ולכן תחזיר את שדה זה. בכל מקרה אחר המתודה יוצרת מערך חדש המאותחל לפי גודל העץ (size) וקוראת למתודת עזר רקורסיבית אשר מבצעת את מילוי המערך keysToArray(this.root, arr, 0).
    סיבוכיות worst cast: O(n).
    סיבוכיות best case: O(1).

19. **int keysToArray(WAVLNode node, int[] arr, int i)** – מתודה המקבלת צומת בעץ, מערך ואינדקס i. זוהי פונקציה רקורסיבית שמטיילת בכל צמתי העץ inorder וממלאת את המערך במפתחות העץ.
סיבוכיות: O(n)

20. **String[] infoToArray()** - מתודה המחזירה מערך המכיל את כל המידע בעץ ממויין על פי מפתחותיו. אם המתודה יצרה כבר מערך כזה והעץ לא השתנה מאז, מערך זה מוחזק בשדה sortedInfo ולכן תחזיר את שדה זה. בכל מקרה אחר המתודה יוצרת מערך חדש המאותחל לפי גודל העץ (size) וקוראת למתודת עזר רקורסיבית אשר מבצעת את מילוי המערך infoToArray(this.root, arr, 0).
סיבוכיות worst cast: O(n).
סיבוכיות best case: O(1).

21. **int infoToArray(WAVLNode node, String[] arr, int i)** - מתודה המקבלת צומת בעץ, מערך ואינדקס i. זוהי פונקציה רקורסיבית שמטיילת בכל צמתי העץ inorder וממלאת את המערך במידע המוחזק בצמתי העץ.
סיבוכיות: O(n)

22. **int size()** - מתודה המחזירה את גודל העץ ששמור כשדה במחלקה.
סיבוכיות: O(1).

23. **int getRank(WAVLNode node)** – פונקציית עזר המקבלת צומת בעץ ומחזירה את דרגתו. אם הצומת הוא null אזי מדובר בexternal node ולכן תחזיר הפונקציה 1-.
סיבוכיות: O(1)

24. **void rotateRight(WAVLNode node, boolean afterDeletion)** -
המתודה מקבלת צומת בעץ ומשתנה בוליאני שערכו אמת אם הסיבוב נעשה לאחר פעולת מחיקה. מטרת המתודה לבצע סיבוב לימין של הצומת. הסיבוב מתבצע כך:
   א. נשמור במשתנה k את בנו השמאלי של צומת הפרמטר.
   ב. נגדיר את בנו השמאלי של צומת הפרמטר להיות בנו הימני של k.
   ג. נגדיר את בנו הימני של k להיות צומת הפרמטר.
לאחר הסיבוב המתודה תקרא לפונקציה finishRotation שזו תחבר את תת-העץ המסובב ששורשו הצומת k לשאר העץ וכמו כן תבצע שינויי דרגות בהתאם לערכו של המשתנה afterDeletion.
סיבוכיות: O(1)

25. **void rotateLeft(WAVLNode node, boolean afterDeletion)** –

המתודה מקבלת צומת בעץ ומשתנה בוליאני שערכו אמת אם הסיבוב נעשה לאחר פעולת מחיקה. מטרת המתודה לבצע סיבוב לשמאל של הצומת. הסיבוב מתבצע כך:

ד. נשמור במשתנה k את בנו הימני של צומת הפרמטר.

ה. נגדיר את בנו הימני של צומת הפרמטר להיות בנו השמאלי של k.

ו. נגדיר את בנו השמאלי של k להיות צומת הפרמטר.

לאחר הסיבוב המתודה תקרא לפונקציה finishRotation שזו תחבר את תת-העץ המסובב ששורשו הצומת k לשאר העץ וכמו כן תבצע שינויי דרגות בהתאם לערכו של המשתנה afterDeletion.

סיבוכיות: O(1)

26. **void finishRotation(WAVLNode node, WAVLNode k, WAVLNode oldParent, boolean isLeftChild, boolean afterDeletion)** - מתודה זו מסיימת תהליך של סיבוב ע"י חיבור תת-העץ המסובב לשאר העץ. כמו כן המתודה מבצעת שינויי דרגות בצמתים המעורבים בסיבוב.

הפרטמרים המתקבלים: node – הצומת שלפני הסיבוב היה שורש תת העץ.

k - הצומת שלפני הסיבוב היה בנו של node.

oldParent – הצומת שלפני הסיבוב היה ההורה של node.

isLeftChild – משתנה בוליאני שערכו אמת אם לפני הסיבוב היה הצומת node בנו השמאלי של oldParent.

afterDeletion – משתנה בוליאני שערכו אמת אם הסיבוב התבצע לאחר פעולת מחיקה.

אופן הפעולה: k הוא השורש של תת-העץ שעבר סיבוב, לכן בהתאם לערכו של isLeftChild נחבר אותו כבן שמאלי או ימני של oldParent. בכך בעצם חיברנו את תת-העץ לשאר העץ.

את דרגת הצומת node נוריד ב-1. לבסוף אם פעולת הסיבוב התבצעה לאחר פעולת מחיקה נעלה את דרגת k ב-1.

סיבוכיות: O(1)

27. **void doubleRotateLeftRight(WAVLNode node, boolean afterDeletion)** –

המתודה מקבלת צומת בעץ ומשתנה בוליאני שערכו אמת אם מדובר בסיבוב כפול לאחר פעולת מחיקה.

המתודה מבצעת סיבוב כפול שמאלה ואז ימינה, כלומר הפעלת מתודת סיבוב שמאלה על הבן השמאלי של הצומת באמצעות פונקציית rotateLeft ואז הפעלת מתודת סיבוב ימינה על הצומת באמצעות פונקציית rotateRight.

אם הסיבוב הכפול מתבצע לאחר פעולת מחיקה נוריד את דרגת הצומת ב-1.

אם הסיבוב הכפול מתבצע לאחר פעולת הכנסה נגדיל את דרגת ההורה של הצומת ב-1.

סיבוכיות: O(1)

28. **- void doubleRotateRightLeft(WAVLNode node, boolean afterDeletion)**

המתודה מקבלת צומת בעץ ומשתנה בוליאני שערכו אמת אם מדובר בסיבוב כפול לאחר פעולת מחיקה.

המתודה מבצעת סיבוב כפול ימינה ואז שמאלה, כלומר הפעלת מתודת סיבוב ימינה על הבן הימני של הצומת באמצעות פונקציית rotateRight ואז הפעלת מתודת סיבוב שמאלה על הצומת באמצעות פונקציית rotateLeft.

אם הסיבוב הכפול מתבצע לאחר פעולת מחיקה נוריד את דרגת הצומת ב-1.

אם הסיבוב הכפול מתבצע לאחר פעולת הכנסה נגדיל את דרגת ההורה של הצומת ב-1.

סיבוכיות: O(1)

מדידות

מספר סידורי	מספר פעולות	מספר פעולות האיזון הממוצע לפעולת insert	מספר פעולות האיזון הממוצע לפעולת delete	מספר פעולות האיזון המקסימלי לפעולת insert	מספר פעולות האיזון המקסימלי לפעולת delete
1	10,000	2.47	1.38	15	7
2	20,000	2.48	1.38	16	8
3	30,000	2.48	1.39	17	8
4	40,000	2.48	1.39	17	9
5	50,000	2.49	1.38	18	9
6	60,000	2.47	1.38	18	9
7	70,000	2.47	1.38	18	9
8	80,000	2.48	1.39	19	9
9	90,000	2.47	1.38	19	9
10	100,000	2.48	1.39	19	10

לפי הנלמד בכתה לגבי עצי WAVL, הסיבוכיות worst case לפעולות איזון הוא O(log n) אך הסיבוכיות הממוצעת (amortized) היא O(1). מכאן נגזרות הציפיות שלנו למדידות: ערך קבוע במדידת מספר פעולות האיזון הממוצע ללא תלות בגודל העץ, וגידול איטי במדידת מספר הפעולות המקסימלי כתלות בגודל העץ.

התוצאות שקיבלנו תואמות את הציפיות האלו: מספר פעולות האיזון הממוצע לפעולת insert היה קבוע 2.48 עבור כל גדלי העצים, כך גם עבור הממוצע לפעולת delete שעומד על 1.38. במספר פעולות האיזון המקסימלי אכן נראה גידול איטי: עבור פעולת insert גדל הערך מ15 עבור עץ בגודל 10,000 ל19 עבור עץ בגודל 100,00. כך גם לגבי הערך המקסימלי עבור פעולת delete שגדל מ-7 ל-10.

משמעות המדידות היא הוכחה נוספת בדבר העלות הממוצעת לפעולות איזון העץ לאחר insertion ו-deletion, שהיא קבועה ללא תלות בגודל העץ.