Software Documentation

WAVL-TREE

המחלקה WAVL Tree מממשת עץ חיפוש בינרי. גובה העץ לכל היותר 2log(n) כאשר n הוא מספר האיברים בעץ.

העץ מאוזן באופן הבא: כאשר עבור עלה הפרשי הדרגות הוא 1,1 ועבור שאר הצמתים הדרגות האפשריות הן: (2,1), (1,2), (2,2), (1,1).

עבור פעולות הכנסה ומחיקה סיבוכיות זמן הריצה amortized היא O(1).

שדות המחלקה

EXT- זהו שדה סטטי מסוג WAVL NODE עבור עלים חיצוניים לעץ, כאשר שדה זה מייצג את כל העלים החיצוניים בעץ. ערכיו קבועים כאשר דרגתו היא 1- , ילדיו וההורה שלו null והוא מכיל מפתח פיקטיבי אינסוף שלילי.

Root- שדה מסוג WAVL NODE גם כן, שמייצג את שורש העץ. ההורה שלו מוגדר להיות null.

Size- שדה מסוג int שמייצג את כמות האיברים בעץ, מעודכן בעת מחיקה והכנסה.

Min- שדה מסוג WAVL NODE אשר מייצג את הצומת שמכיל את המפתח המינימלי בעץ.

Max- שדה מסוג WAVL NODE אשר מייצג את הצומת שמכיל את המפתח המקסימלי בעץ.

WAVL NODE

מחלקה שמייצגת צומת בעץ. כל צומת מכיל את השדות הבאים:

Left- מצביע לילד שמאלי

Right- מצביע לילד הימני

Parent- מצביע להורה

Rank- הדרגה של הצומת

Key- המפתח של הצומת

Info- מידע שמכיל הצומת

מתודות

מתודות רוחביות

אלו מתודות עזר שמשמשות פונקציות רבות בתוכנית ונועדו בעיקר כדי לשפר את קריאות הקוד.

```
Private int rankDiff(WAVLNode x,WAVLNode y)
```

הפונקציה מחשבת הפרשי דרגות בין שני צמתים (דרגת X פחות דרגת Y).

<u>סיבוכיות-</u> מספר פעולות קבוע ולכן O(1).

```
Private Boolean isLeaf(WAVLNode x)
```

הפונקציה מקבלת צומת ובודקת אם הוא עלה ע"י בדיקה אם שני הבנים הם EXT.

<u>סיבוכיות-</u> מספר פעולות קבוע ולכן O(1).

```
Private Boolean isUnaryLeft(WAVLNode x)
```

הפונקציה מקבלת צומת ובודקת אם יש לו בן אחד מצד שמאל (מצד ימין EXT).

<u>סיבוכיות-</u> מספר פעולות קבוע ולכן O(1).

```
Private Boolean isUnaryRight(WAVLNode x)
```

הפונקציה מקבלת צומת ובודקת אם יש לו בן אחד מצד ימין (מצד שמאל EXT).

<u>סיבוכיות-</u> מספר פעולות קבוע ולכן O(1).

מתודות ליבה

```
Public boolean empty()
```

מחזירה ערך true אם העץ ריק ואחרת false.

המתודה בודקת האם השורש הוא null, כלומר אין איברים בעץ.

<u>סיבוכיות:</u>O(1) , עלות בדיקה בודדת.

```
Public int size()
```

הפונקציה מחזירה את כמות האיברים בעץ.

<u>סיבוכיות-</u> הפונקציה מבצעת פעולה קבועה אחת שהיא החזרת הגודל שכן ישנו שדה שמתחזק את כמות האיברים ומתעדכן בעת הכנסה ומחיקה. לכן הסיבוכיות O(1).

```
public String search(int k)
```

הפונקציה מקבלת מפתח k ומחפשת את הצומת המתאים בעץ, אם לא נמצא כזה מחזירה null, אם נמצא היא מחזירה את ה info שלו. הפונקציה נעזרת בפונקציית עזר treeSearch לצורך חיפוש האיבר.

סיבוכיות: מלבד הקריאה לפונקציית העזר הפונקציה מבצעת כמות פעולות קבועה ולכן הסיבוכיות היא O(log(n)).

```
Private WAVLNode treeSearch(WAVLNode x, int k)
```

הפונקציה מבצעת חיפוש בינארי על הצומת x ומחזירה את הצומת שמכיל את המפתח, אם אין כזה תחזיר EXT.

סיבוכיות: במקרה הגרוע החיפוש יסתיים בEXT ויתחיל משורש העץ ולכן נרד את גובה העץ כולו בסיבוכיות(O(log(n)).

```
Public int insert(int k, String i)
```

הפונקציה מקבלת מפתח k וערך i ומכניסה אותו לעץ, לאחר מכן הפונקציה תשמר את הדרגות בעץ ע"י ביצוע איזון. לבסוף תחזיר את מספר פעולות האיזון שנדרשו. אם האיבר קיים בעץ הפונקציה תחזיר -1.

ניצור צומת חדש המכיל את הפרטים שקיבלנו. אם העץ ריק הצומת שיצרנו יהיה השורש, אחרת בעזרת פונקציית העזר treePosition נקבל את מיקום ההכנסה של האיבר(ההורה של הצומת החדש), נכניס אותו מימין או משמאל בהתאם לגודלו.

נבצע עדכון למצביעים של המינימום והמקסימום במידת הצורך(O(1)). נחלק למקרים כאשר ההורה הוא עלה או אונארי ונבצע מספר פעולות קבוע עבור כל מקרה. לאחר מכן נשלח לפונקציית האיזון.

סיבוכיות:(O(log(n))

Worst Case (Insert)=log(n)+log(n)+C=O(log(n))

כאשר C הוא קבוע שמייצג כמות פעולות קבועות שביצענו

Log(n) סיבוכיות worst case של פונקציית העזר treePosition

Log(n) סיבוכיות worst case של פונקציית העזר rebalancingTreeInsert.

```
Private WAVLNode treePosition(WAVLNode x, int k)
```

הפונקציה מקבלת צומת ומפתח k. הפונקציה מחזירה את מיקום ההכנסה של צומת בעל מפתח k בתת העץ של x, באמצעות חיפוש בינארי. אם k כבר קיים בתת העץ של x, הפונקציה מחזירה את הצומת לו שייך מפתח k.

סיבוכיות: המקרה הגרוע הוא כש- x הוא השורש ונצטרך לרדת עד שכבת העלים. במקרה זה נרד כגובה העץ ולכן הסיבוכיות היא O(log(n)).

```
Private int rebalancingTreeInsert(WAVLNode y,WAVLNode z,int counter)
```

מקבלת שני צמתים y ו z כך ש y הוא ההורה של z, ומונה. הפונקציה מכילה לולאת While שמפסיקה לרוץ כאשר תהליך האיזון מסתיים(תיקון הבעיה או הגעה לשורש).

הפונקציה מחלקת לשני מקרים לפי המיקום של z ביחס לy (בן שמאלי או ימני) וזאת בכדי להבדיל בין מקרי האיזון השונים. בכל מקרה מתבצעות מספר פעולות קבועות שאינן תלויות בכמות האיברים בעץ ולאחר מכן הלולאה תמשיך לרוץ או שתסתיים עם החזרת המונה(סיבוב בודד\כפול בעץ או שהבעיה לא התגלגלה במעלה העץ).

פונקציות עזר- rightRotation , leftRotation, doubleRightRotation, doubleLeftRotation.

סיבוכיות- בכל איטרציה מתבצעות מספר קבוע של פעולות. נבדוק באיזה case אנחנו נמצאים, לאחר הזיהוי נבצע פעולות בהתאם (שינויי דרגות , מצביעים).נשים לב שפונקציות העזר רצות בסיבוכיות זמן O(1) ולכן אינן משפיעות על סיבוכיות זמן הריצה אסימפטוטית. לבסוף הלולאה תסתיים או תמשיך לרוץ בהתאם למקרים. אם כן במקרה הגרוע נעלה במעלה העץ עד להגעה לשורש ולכן סיבוכיות זמן הריצה היא O(log(n)).

O(`rebalancingTreeInsert`)= C·log(n)=O(log(n))

```
Private void leftRotation(WAVLNode x)
```

הפונקציה מקבלת צומת X בעץ ומבצעת סיבוב שמאלה עם בנו הימני ע"י עדכון מצביעים.

סיבוכיות-מדובר בכמות פעולות קבועה ולכן הסיבוכיות היא O(1).

```
Private void rightRotation(WAVLNode x)
```

הפונקציה מקבלת צומת X בעץ ומבצעת סיבוב ימינה עם בנו השמאלי ע"י עדכון מצביעים.

סיבוכיות- מדובר בכמות פעולות קבועה ולכן הסיבוכיות היא O(1).

```
Private void doubleLeftRotation(WAVLNode x)
```

הפונקציה מבצעת שני סיבובים. הראשון סיבוב ימינה של בנו הימני של x והשני סיבוב שמאלה של x.

פונקציות עזר- rightRotation, leftRotation.

סיבוכיות- שימוש בשני סיבובים שכאמור רצים בסיבוכיות של O(1) ולכן הסיבוכיות היא O(1).

```
Private void doubleRightRotation(WAVLNode x)
```

הפונקציה מבצעת שני סיבובים. הראשון סיבוב שמאלה של בנו השמאלי של x והשני סיבוב ימינה של x.

פונקציות עזר- rightRotation , leftRotation.

סיבוכיות- שימוש בשני סיבובים שכאמור רצים בסיבוכיות של O(1) ולכן הסיבוכיות היא O(1).

```
Public int delete(int k)
```

הפונקציה מוחקת את הצומת שמכיל את המפתח k, אם המפתח אינו קיים בעץ הפונקציה תחזיר 1-.

בנוסף, הפונקציה תבצע איזונים על מנת לשמר את עץ הWAVL ותחזיר את כמות פעולות האיזון.

מציאת המפתח בעץ- ראשית הפונקציה בודקת אם המפתח מינימלי\מקסימלי בעץ. במקרה שבו הצומת מינימלי נעדכן את הצומת min להיות העוקב שלו, נסמן את הצומת להיות itemToDelete, ולמעשה נסיים את פעולת החיפוש בסיבוכיות O(1). נשים לב שהצומת המינימלי בהכרח בן שמאלי (אחרת יש צומת קטן ממנו בסתירה למינימליות).במקרה בו דרגתו 0 העוקב הוא צומת האב ובמקרה

בו הדרגה 1 (אין דרגות נוספות אפשריות) העוקב הוא הבן הימני שלו(שאינו מכיל בנים). אם כן בכל מקרה מציאת העוקב היא במספר פעולות קבוע ולכן בסיבוכיות O(1).

המקרה בו הצומת מקסימלי סימטרי (כמובן שנעדכן את הצומת max).

במידה והצומת אינו מקסימלי\מינימלי נשתמש בפונקציית העזר treeSearch לקבלת מיקום הצומת.

אם הצומת מכיל שני בנים נחליף אותו בעוקב שלו ונסמנו כitemToDelete. אם הצומת שורש נשלח לפונקציה deletingRoot, אחרת נבדוק האם הצומת הוא עלה או אונארי ונטפל בהתאם למקרים. אם הצומת עלה נמחק אותו ונהיה באחד משלושת המקרים הבאים:

1. לא קיימת בעיה בעץ ולכן נחזיר את המונה ונסיים את פעולת הפונקציה.

2. אביו של הצומת הנמחק הפך להיות עלה (2,2), במקרה זה נוריד את דרגתו ונשלח לפונקציית האיזון את אביו של צומת זה.

3. הפרש הדרגות הוא 3 ולכן נשלח לפונקציית האיזון את אביו של הצומת המחוק.

אם הצומת אונארי מימין\שמאל נעדכן את המצביעים כך ש"ידלגו" על הצומת בעץ וכך הוא למעשה ימחק. אם הדילוג יצר הפרש דרגות 3 נשלח לפונקציית האיזון את אביו של הצומת המחוק.

<u>פונקציות עזר-</u> rebalancingTreeDeletion, deletingRoot, successor, predecessor , swap, treeSearch.

<u>סיבוכיות-</u> אנו קוראים לפונקציות העזר פעם אחת בלבד בתהליך המחיקה. סיבוכיות זמן הריצה הגרועה ביותר של הפונקציות היא O(log(n)) ולכן זוהי גם סיבוכיות פונקציית המחיקה.

Worst Case (delete)=log(n)+log(n)+ log(n) +C=O(log(n))

כאשר C הוא קבוע שמייצג כמות פעולות קבועות שביצענו + הפונקציות swap, predecessor (בפונקציה זו בלבד מהסיבות שצוינו) ו-deletingRoot.

Log(n) סיבוכיות worst case של פונקציית העזר treeSearch.

Log(n) סיבוכיות worst case של פונקציית העזר rebalancingTreeDeletion.

Log(n) סיבוכיות worst case של פונקציית העזר successor.

```
Private int rebalancingTreeDeletion(WAVLNode z,int count)
```

הפונקציה מקבלת צומת ומונה ודואגת לאיזון העץ והחזרת מספר הפעולות שנדרשו לשם כך.

הפונקציה מכילה לולאה אשר בכל איטרציה בודקת 9 מקרים (כאשר 1-4 סימטריים ל5-8). בכל מקרה מתבצע מספר פעולות קבוע בהתאם לשינויי הדרגות ועדכוני המצביעים הנדרשים (סיבובים).

הלולאה מסתיימת כאשר הבעיה נפתרת או שהגענו לשורש העץ.

<u>פונקציות עזר-</u> rightRotation, leftRotation, doubleRightRotation, doubleLeftRotation.

<u>סיבוכיות-</u> בכל איטרציה מתבצעות מספר קבוע של פעולות. נבדוק באיזה case אנחנו נמצאים, לאחר הזיהוי נבצע פעולות בהתאם (שינויי דרגות , מצביעים).נשים לב שפונקציות העזר רצות בסיבוכיות זמן O(1) ולכן אינן משפיעות על סיבוכיות זמן הריצה אסימפטוטית. לבסוף הלולאה תסתיים או תמשיך לרוץ בהתאם למקרים. אם כן במקרה הגרוע נעלה במעלה העץ עד להגעה לשורש ולכן סיבוכיות זמן הריצה היא O(log(n)).

O(`rebalancingTreeDeletion`) = C·log(n) = O(log(n))

```
Private int deletingRoot(int count)
```

נשים לב שאם הגענו לפונקציה זו הרי שאנחנו בשורש ואין לו שני ילדים (כי אם היו ביצענו החלפה עם אחד מהם והאיבר למחיקה אינו השורש יותר ולכן זה מקרה אחר).

הפונקציה מקבלת מונה. ישנם 3 מקרים אפשריים: השורש הוא עלה, צומת אונארי משמאל או צומת אונארי מימין. נעדכן מצביעים ואת גודל העץ בהתאם לכל מקרה.

<u>סיבוכיות-</u> אנו מבצעים פעולות קבועות ולכן הסיבוכיות היא O(1).

```
Private WAVLNode successor(WAVLNode x)
```

הפונקציה מקבלת צומת ומחזירה את הצומת העוקב לו בעץ לפי ערך המפתח.

אם לצומת קיים בן ימני, נחזיר את המינימום בתת העץ הימני שלו. אחרת, נחזיר את האב הקדמון הראשון כך ש-x נמצא בתת העץ השמאלי שלו.

<u>פונקציות עזר-</u> `min(WAVLNode x)` .

<u>סיבוכיות-</u> במקרה הגרוע נשלח את שורש העץ ונרד עד העלים, או לחילופין נקבל עלה והצומת העוקב לו הוא שורש העץ. לכן המקרה הגרוע הוא גובה העץ, כלומר O(log(n)).

```
Private WAVLNode min(WAVLNode x)
```

הפונקציה מקבלת צומת ומחזירה את האיבר המינימלי בתת העץ שלו (כולל x).

<u>סיבוכיות-</u> המקרה הגרוע הוא כש- x הוא שורש העץ. במקרה זה נצטרך לרדת עד העלה השמאלי ביותר, כלומר סדר גודל של גובה העץ. לכן סיבוכיות זמן הריצה היא O(log(n)).

```
Private WAVLNode predecessor(WAVLNode x)
```

הפונקציה מקבלת צומת ומחזירה את הצומת הקודם לו בעץ לפי ערך המפתח.

אם לצומת קיים בן שמאלי, נחזיר את המקסימום בתת העץ השמאלי שלו. אחרת, נחזיר את האב הקדמון הראשון כך ש-x נמצא בתת העץ הימני שלו.

<u>פונקציות עזר-</u> `max(WAVLNode x)` .

<u>סיבוכיות-</u> במקרה הגרוע נשלח את שורש העץ ונרד עד העלים, או לחילופין נקבל עלה והצומת הקודם לו הוא שורש העץ. לכן המקרה הגרוע הוא גובה העץ, כלומר O(log(n)).

```
Private void swap(WAVLNode x,WAVLNode y)
```

הפונקציה מקבלת שני משתנים מסוג WAVLNode (צמתים) ו"מחליפה" בין המיקומים שלהם ע"י החלפה בין הinfo והkey שלהם.

<u>סיבוכיות-</u> מדובר במספר קבוע של פעולות ולכן הסיבוכיות היא O(1).

```
private WAVLNode max(WAVLNode x)
```

הפונקציה מקבלת צומת ומחזירה את האיבר המקסימלי בתת העץ שלו (כולל x).

<u>סיבוכיות-</u> המקרה הגרוע הוא כש- x הוא שורש העץ. במקרה זה נצטרך לרדת עד העלה הימני ביותר, כלומר סדר גודל של גובה העץ. לכן סיבוכיות זמן הריצה היא O(log(n)).

```
public String min()
```

הפונקציה מחזירה את ה- info של המפתח הקטן ביותר בעץ או null אם העץ ריק.

<u>סיבוכיות-</u> הפונקציה מחזירה את ה-info של שדה המחלקה min או null אם העץ ריק (בדיקה שלוקחת זמן קבוע) ולכן סיבוכיות זמן הריצה היא קבועה, O(1).

```
public String max()
```

הפונקציה מחזירה את ה- info של המפתח הגדול ביותר בעץ או null אם העץ ריק.

<u>סיבוכיות-</u> הפונקציה מחזירה את ה-info של שדה המחלקה max או null אם העץ ריק (בדיקה שלוקחת זמן קבוע) ולכן סיבוכיות זמן הריצה היא קבועה, O(1).

```
Public int[] keysToArray()
```

הפונקציה מחזירה מערך המכיל ערכים מסוג int ובו כל המפתחות שבעץ באופן ממויין מהקטן לגדול. אם העץ ריק הפונקציה תחזיר מערך ריק.

<u>סיבוכיות-</u> הפונקציה עושה שימוש בפונקציית העזר sortedOrder ומלבדה מבצעת כמות פעולות קבועה (שאינה תלויה בכמות האיברים בעץ), לכן הסיבוכיות היא O(n).

```
Private int sortedOrder(WAVLNode x,int[] arr, int position)
```

זו פונקציה רקורסיבית שמקבלת צומת (שורש תת העץ שיש לקרוא), מערך שאיבריו מסוג int וערך מספרי שמייצג מיקום במערך. הפונקציה עוברת על כל תת עץ משמאל לימין וממלאת את המערך במפתחות של הצמתים המתאימים החל מהמיקום שקיבלה. בסיום הפונקציה מחזירה את המיקום החדש ממנו נמשיך למלא את המערך .

<u>סיבוכיות-</u> עוברים על כל איבר לכל היותר פעמיים בסריקה של העץ (בדרך מהשורש לעלים ובחזרה) ובסה"כ בסיבוכיות O(n).

```
public String[] infoToArray()
```

הפונקציה מחזירה מערך המכיל ערכים מסוג String ובו כל הinfo שבעץ באופן ממויין בסדר עולה לפי המפתחות המתאימים לערכים. אם העץ ריק הפונקציה תחזיר מערך ריק.

<u>סיבוכיות-</u> הפונקציה עושה שימוש בפונקציית העזר sortedOrderInfo ומלבדה מבצעת כמות פעולות קבועה(שאינה תלויה בכמות האיברים בעץ), לכן הסיבוכיות היא O(n).

```
Private int sortedOrderInfo(WAVLNode x,String[]arr, int position)
```

זו פונקציה רקורסיבית שמקבלת צומת (שורש תת העץ שיש לקרוא), מערך שאיבריו מסוג String וערך מספרי שמייצג מיקום במערך. הפונקציה עוברת על כל תת עץ משמאל לימין וממלאת את המערך בinfo של הצמתים המתאימים החל מהמיקום שקיבלה. בסיום הפונקציה מחזירה את המיקום החדש ממנו נמשיך למלא את המערך .

<u>סיבוכיות-</u> עוברים על כל איבר לכל היותר פעמיים בסריקה של העץ (בדרך מהשורש לעלים ובחזרה) ובסה"כ בסיבוכיות O(n).

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	10000	2.47	1.38	15	8
2	20000	2.49	1.39	16	8
3	30000	2.49	1.39	17	8
4	40000	2.48	1.39	17	8
5	50000	2.48	1.39	18	10
6	60000	2.48	1.39	18	10
7	70000	2.48	1.39	19	10
8	80000	2.48	1.39	19	10
9	90000	2.48	1.39	19	10
10	100000	2.49	1.39	19	10

ראינו בהרצאה את פונקצית הפוטנציאל הבאה:

$$Potential = \Phi = (\text{number of 1,1-nodes}) + 2 \times (\text{number of 3,2 and 2,2-nodes})$$

לפיה סיבוכיות זמן הריצה Amortized של פעולות ה-insert וה deleteקבוע בסדרה של הכנסות ומחיקות ולכן ציפינו שמספר פעולות האיזון הממוצע יהיה גם כן קבוע ולא ישתנה כאשר נוסיף איברים. אכן ניתן לראות בטבלה שהנתונים מתיישבים עם ההנחה שלנו.

לגבי מספר פעולות האיזון המקסימלי, ציפינו שמספר הפעולות יגדל עם הגדלת כמות האיברים ויהיה בסדר גודל log(n) (n מספר האיברים בעץ). זאת מכיוון שעבור הכנסה\מחיקה בודדת סיבוכיות זמן הריצה הגרועה ביותר היא בגובה העץ, כלומר log(n), כאשר האיזון מסתיים בשורש העץ. ניתן לראות שאכן מספר פעולות האיזון המקסימלי גדל בdelete וגם בinsert עם הגדלת כמות האיברים והוא בסדר גודל המתאים להנחה.