**עץ WAVL**

קארין בל פדר – carinebel - 203336425

תמי לביא – tamilavi - 203010574

תיאור המחלקה

למחלקת WAVLTree ישנם4 משתני מחלקה, שלושת מטיפוס WAVLNode (יורחב בהמשך) ואחד int:

1. ***root*** המשמש כמצביע לשורש העץ הנוכחי.
2. ***max\_node*** המצביע לאיבר המקסימלי בעץ.
3. ***min\_node*** המצביע לאיבר המינימלי בעץ.
4. ***Size***int המחזיק את גודל העץ הנוכחי.

למחלקה שלוש מחלקות פנימיות:

1. ***AbsWAVLNode:*** מחלקה אבסטרקטית המייצגת קודקוד כלשהו בעץ. לכל קודקוד דרגה (rank) ואבא (parent) מסוג WAVLNode (מפורט בנקודה הבאה). זאת כיוון שקודקוד חיצוני לא יכול להיות אבא של קודקוד בעל מידע.
2. ***WAVLNode:*** מייצגת קודקוד בעל מידע בעץ, ועל כן יש לה מפתח (key) וערך (value). בנוסף מחזיק בן שמאלי וימני מסוג AbsWAVLNode. בנים אלו לעולם לא יהיו null - או שיהיו בעלי מידע – WAVLNode אחרים, או שיהיו מסוג WAVLExternalNode.

למחלקה מספר מתודות עזר לא טריוויאליות:

* *getRankDiff():* מחזיר את ההפרש בין הדרגה של הבן השמאלי לשל הבן הימני. משמש להחלטה לאיזה כיוון לאזן את העץ.
* *updateRank():* מעדכן את הדרגה בהתאם לילדיו. מחזיר Boolean האם הדרגה השתנתה.
* *isValidRankDiff():* האם הפרשי הדרגות של הקודקוד מול ילדיו חוקי, ובפרט אם הוא עלה פנימי האם דרגתו 0.
* *isRankDiffBigEnough():* מחזיר האם הפרשי הדרגות בין צמתים גדולים מספיק על מנת להיות תקינים.
* *isRankDiffSmallEnough():*מחזיר האם הפרשי הדרגות בין צמתים גדולים מספיק על מנת להיות תקינים.

1. ***WAVLExternalNode:*** מחלקה המייצגת עלים חיצוניים. מכאן שדרגתם תמיד 1-, ואין להם בנים.

מימוש הפונקציות:

# פונקציות עזר כלליות:

## ***Private void rotateRightLeftToRightRight(WAVLNode source, WAVLNode child):***

מתודת עזר שמטרתה לבצע פעולת רוטציה ראשונה מ-Right-Left ל-Right-Right במידה ונדרש double rotation. O(1).

## ***private int rebalanceRightSide(WAVLNode source):***

מתודת עזר שמטרתה לאזן את צידו הימני של NODE, כולל להחליף את מקומו. במידה ונדרשת רוטציה כפולה תקרא ל-rotateRightLeftToRightRight. תחזיר 1 במידה ובוצעה רוטציה יחידה, 2 אם בוצעה רוטציה כפולה, כלומר נקראה המתודה הקודמת. סיבוכיות זמן ריצה .

## ***private void rotateLeftRightToLeftLeft(WAVLNode source, WAVLNode child):***

אחראית על ביצוע רוטציה ראשונה במידה ונדרש double rotration בצד השמאלי של העץ. תחליף מצב Left-Right למצב Left-Left. .

## ***private int rebalanceLeftSide(WAVLNode node):***

מתודת עזר לאיזון צידו הימני של ה-node, כולל החלפת מקומו. במידה ונדרשת רוטציה כפולה תקרא לפונקציה הקודמת. תחזיר 1 אם בוצעה רוטציה יחידה, 2 אם כפולה. סיבוכיות זמן ריצה .

## ***private rebalance(WAVLNode node, int count):***

מתודת עזר רקורסיבית לאיזון העץ. תפסיק או כאשר הגענו ל-root ואיזנו אותו, או אם הקודקוד הנוכחי "ספג" את השינוי בעץ ולא נדרש עדכון ב-RANK שלו.

המתודה בודקת האם לאחר שינוי ה-RANK הקודקוד מאוזן. במידה ולא – נבדוק לאיזה כיוון חוסר האיזון נוטה, ובהתאם נקרא לאחת המתודות rebalanceLeftSide, rebalanceRightSide. במידה והקודקוד מאוזן (ואכן עודכן – אחרת היינו בתנאי עצירה) נאזן את אביו.

המתודה מחזירה מספר פעולות האיזון שבוצעו לפי הספירה הבאה:

* רק עדכון RANK (כלומר promotion\demotion) – 1
* רוטציה יחידה (כולל החלפת קודקודים ועדכון דרגה) – 1
* רוטציה כפולה (כולל 2 החלפות ועדכוני דרגה) – 2

סיבוכיות זמן ריצה: . במידה ונדרש לעדכן מעלה ועד השורש נגיע לסיבוכיות זמן ריצה זו (נלמד בשיעור).

## ***private WAVLNodefindClosestNode(int k):***

מתודת עזר שמטרתה למצוא את הקודקוד הקיים "הקרוב ביותר" למפתח k. המתודה פועלת מ-min כלפי מעלה, ומחפשת את האב הקדמון המשותף לmin ו-k. לאחר מכן תרד כלפי העלים.

הקודקוד שנחזיר יהיה אחד משלוש: קודקוד בעל מפתח k – כלומר מצאנו את הקודקוד עצמו. קודקוד בעל מפתח גדול מ-K שהבן השמאלי שלו הוא WAVLExternalNode (כלומר המקום החוקי להכניס קודקוד עם מפתח k משמאלו). קודקוד בעל מפתח קטן מ-K שהבן הימני שלו הוא WAVLExternalNode (כלומר המקום החוקי להכניס קודקוד עם מפתח k מימינו).

סיבוכיות זמן ריצה: .

## ***private int keysToArray(AbsWAVLNode source, int[] arr, int place):***

מתודת עזר רקורסיבית למילוי מערך במפתחות של העץ לפי הסדר. הפונקציה מקבלת מיקום, המערך למלא, והקודקוד שאנחנו רצים עליו כרגע. במידה והקודקוד הוא WAVLExternalNode, כלומר אין צורך להוסיף לתא – נחזיר את המיקום שקיבלנו. אחרת place יקבל את הערך של קריאה רקורסיבית לבן השמאלי של הפונקציה. במיקום שקיבלנו חזרה נשים את הערך של ה-NODE הנוכחי – כבר מילאנו את כל התת-עץ השמאלי שלו. כעת עלינו לקדם את place ב-1 ולמלא את התת עץ הימני שלו ע"י קריאה רקורסיבית לפונקציה. הערך שקיבלנו חזרה יהיה הערך שנחזיר לאבא שקרא.

סיבוכיות זמן ריצה: – צריך לעבור על כל קודקוד פעם אחת ולהכניסו.

## ***private int infoToArray(AbsWAVLNode source, String[] arr, int place):***

מתודה עזר רקורסיבית למילוי מערך בערכים של העץ לפי סדר המפתחות. הפונקציה פועלת כמו הפונקציה הקודמת – keysToArray רק ממלאת ערכים במקום מפתחות.

זמן ריצה:

## ***private int recursiveInsert(int k, String i):***

מתודת עזר להוספת איברים לעץ. המתודה משתמשת בשיטת upu (מלמטה) כפי שהוצג בתרגיל המעשי השלישי, על מנת לשפר את זמן הפעולה הממוצע.

## ***public int recursiveDelete(WAVLNode current, int k, WAVLNode parent, boolean toRebalance)***

מתודת עזר רקורסיבית שמחפשת את האיבר למחיקה וממיינת את האיברים למקרים שונים – במקרה שהאיבר למחיקה הוא עלה, צומת עם עלה אחד או צומת עם שניים. בכל אחד מהמקרים היא נעזרת במתודות עזר מתאימות למחיקת כל אחד מהסוגים, למציאת איבר להחליף עמו מקום (במקרה הצורך) ועל מנת לאזן את העץ.

זמן ריצה: .

## ***public int deleteLeaf(WAVLNode current, WAVLNode parent, boolean isLeft, boolean toRebalance)***

מתודת עזר שמוחקת עלה בעץ באמצעות הקצאה של העלה איבר ריק (WAVLExternalNode) עבור ההורה. מעדכנת איבר מינימלי/מקסימלי אם נחוץ, ומאזנת רק אם המחיקה של העלה אינה במסגרת מחיקה של צומת (מזהה ע"י הפרמטר toRebalance).

זמן ריצה: (במקרה ונדרש איזון) אחרת

## ***public int deleteUnaryNode(WAVLNode current, WAVLNode parent, boolean hasLeftLeaf, boolean toRebalance)***

מתודת עזר שמוחקת צומת בעלת עלה אחד ע"י עדכון ההורה של הצומת להצביע על העלה של צומת. מעדכנת איבר מינימלי/מקסימלי אם נחוץ, ומאזנת רק אם המחיקה של העלה אינה במסגרת מחיקה של צומת (מזהה ע"י הפרמטר toRebalance.

זמן ריצה: (במקרה ונדרש איזון) אחרת

## ***public boolean isLeaf(WAVLNode current)***

מתודת עזר שבודקת אם מדובר בצומת ללא עלים.

## ***public boolean isLeftUnary(WAVLNode current)***

מתודת עזר שבודקת אם מדובר בצומת אם עלה שמאלי בלבד.

## ***public boolean isRightUnary(WAVLNode current)***

מתודת עזר שבודקת אם מדובר בצומת אם עלה ימני בלבד.

## ***public WAVLNode findSuccessor(WAVLNode current)***

מתודת עזר הנעזרת במתודה הרקורסיבית recursiveSuccessor על מנת למצוא את תת-הצומת הקרובה ביותר.

## ***public WAVLNode recursiveSuccessor(WAVLNode current, int k)***

מתודת עזר שבודקת עבור כל מקרה מה הצומת הקרוב ביותר ל-k. עבור עלה היא מחזירה את העלה עצמו, עבור צומת היא מחזירה את הקרוב מבין השלושה – הצומת הקרוב ביותר מימין, הקרוב ביותר משמאל או הצומת עצמו.

# פונקציות מטרה:

## ***public int insert(int k, String i):***

מתודה שאחראית על הכנסת איבר חדש k במידה ואינו קיים בעץ. מחזירה את כמות פעולות האיזון השונות שנעשו בשביל לשמור על העץ.

המתודה קוראת ל-findClosestNode בכדי לדעת לאן להכניס. במידה וה-closeset בעל המפתח שנרצה להכניס- נסיים כאן ונחזיר -1. אחרת, נכניס את הקודקוד החדש במקומו ביחס ל-closest, ונשלח את ה-closest לאיזון מחדש (לא צריך לאזן את הקודקוד החדש – היצירה שלו היא כשהוא מאוזן).

ניתוח זמן ריצה: findClosestNode – , rebalance- לכן סה"כ

## ***Min:***

מחזיר את הערך של משתנה המחלקה min. במידה והוא ריק יחזיר NULL. זמן ריצה .

## ***MAX:***

מחזיר את הערך של משתנה המחלקה max. במידה והוא ריק יחזיר NULL. זמן ריצה .

## ***keysToArray:***

אם העץ ריק (נעזר ב-empty()) – נחזיר null. אחרת נקרא לפונקציה הרקורסיבית keysToArray(root,new int[size],0) ונחזיר את המערך.

ניתו זמן ריצה: , זמן הריצה של המתודה הפרטית.

## ***infoToArray:***

אם העץ ריק (בעזרת empty()) – מחזיר null. אחרת נקרא לפונקציה הרקורסיבית infoToArray ונחזיר את המערך שעליו בוצע השינוי.

ניתוח זמן ריצה: , זמן הריצה של המתודה הפרטית.

## ***Size:***

מחזירה את משתנה המחלקה size. O(1)

## ***DELETE:***

מתודה שאחראית על מחיקת איבר בעץ. המתודה מחזירה את מספר פעולות האיזון שהתבצעו במהלך המחיקה. המתודה מוודאת שהאיבר אכן נמצא בעץ. במידה וכן, המתודה נעזרת במתודה recursiveDelete על מנת למחוק את האיבר הרלוונטי.

במידה ומדובר בעלה, המתודה מעדכנת את הבן הרלוונטי של ההורה להצביע על איבר ריק באמצעות deleteLeaf. במידה ומדובר בצומת בעל עלה אחד, המתודה מעדכנת את הבן הרלוונטי של ההורה להצביע על העלה היחיד באמצעות deleteUnaryNode. במקרה ומדובר בצומת בעלת שני עלים, המתודה מוצאת את תת-האיבר הקרוב ביותר, מוחקת אותו ומעדכנת את האיבר לערכי תת-האיבר שנמחק.

לאחר המחיקה, המתודה מאזנת את העץ על מנת שישאר תקין.

ניתוח זמן ריצה: search – findSuccessor, – , rebalance- לכן סה"כ

הניסוי:

## ***תוצאות הניסוי:***

תוצאות צפויות: המספר המקסימלי הצפוי של פעולת insert הוא *. זאת כיוון שבביצוע פעולות insert בלבד העץ שקול לעץ AVL, ולכן היינו מצפים כי החסם יהיה , החסם על גובה העץ, שזה אכן מה שקיבלנו.*

במחיקה העץ יכול לראשונה לקבל דרגות של WAVL. המקרה הגרוע ביותר הוא אכן לעלות עד שורש העץ באיזון מחדש. כיוון שהתחלנו מעץ AVL, הגובה ההתחלתי חסום ב-, ולכל היותר ירד במחיקה. לכן נצפה שגם המספר המקסימלי של פעולות עדכון עבור delete יהיה חסום במספר זה.

מבחינת ממוצע: למדנו בכיתה כי זמן amortized של רצף הכנסות הוא , ואכן קיבלנו באופן עקבי כי המספר הממוצע לפעולות, כלומר ה-amortizedTime הוא .

כך גם עבור רצף מחיקות – למדנו בכיתה כי הוא , וקיבלנו בניסויים – זמן ריצה amortized שהוא .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **מספר סידורי** | **מספר פעולות** | **מספר ממוצע ל-insert** | **מספר מקסימלי ל-insert** | **מספר ממוצע ל-delete** | **מספר מקסימלי ל-delete** |
| **1** | 10000 | 2.4689 | 15 | 1.0081 | 14 |
| **2** | 20000 | 2.48275 | 16 | 1.00375 | 13 |
| **3** | 30000 | 2.4797 | 17 | 1.0039 | 15 |
| **4** | 40000 | 2.4802 | 18 | 1.00845 | 14 |
| **5** | 50000 | 2.48838 | 18 | 1.00238 | 15 |
| **6** | 60000 | 2.4810833333333333 | 18 | 1.0031 | 14 |
| **7** | 70000 | 2.491 | 18 | 1.00202857114285715 | 16 |
| **8** | 80000 | 2.4818375 | 18 | 1.006375 | 16 |
| **9** | 90000 | 2.4804666666666666 | 19 | 1.0069333333333332 | 15 |
| **10** | 100000 | 2.48315 | 19 | 1.00651 | 17 |