**עץ WAVL**

קארין בל פדר

תמי לביא

תיאור המחלקה

למחלקת WAVLTree ישנם 3 משתני מחלקה, שלושתם מטיפוס WAVLNode (יורחב בהמשך):

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.

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:

# 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)

הניסוי:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 | 1.8414 | 36 |
| **2** | 20000 | 2.48275 | 16 | 1.8559 | 35 |
| **3** | 30000 | 2.4797 | 17 | 1.8565 | 39 |
| **4** | 40000 | 2.4802 | 18 | 1.865825 | 39 |
| **5** | 50000 | 2.48838 | 18 | 1.8549 | 38 |
| **6** | 60000 | 2.4810833333333333 | 18 | 1.8550666666666666 | 53 |
| **7** | 70000 | 2.491 | 18 | 1.8619571428571429 | 48 |
| **8** | 80000 | 2.4818375 | 19 | 1.8560125 | 49 |
| **9** | 90000 | 2.4804666666666666 | 19 | 1.8628555555555555 | 51 |
| **10** | 100000 | 2.48315 | 19 | 1.86004 | 46 |