קובץ תיעוד – יותם מנה 204717862, דניאל פאר 206283442

**מחלקת WAVLTree:**

**שדות:**

**private WAVLNode root**: שדה מופע מטיפוס WAVLNode המוגדר להיות שורש העץ.

**Private static WAVLNode sentinel**: שדה סטטי מטיפוס WAVLNodeהמשמש כל צומת שאינו בינארי בעץ בתור בן וירטואלי. השדה סטטי מטעמי חיסכון בזיכרון – כל צומת בעץ שאינו בינארי יצביע עליו כבן וירטואלי.

**private WAVLNode min:** שדה מופע מטיפוס WAVLNode השומר את הצומת עם המפתח הקטן ביותר.

**private WAVLNode max:** שדה מופע מטיפוס WAVLNode השומר את הצומת עם המפתח הגדול ביותר.

**מתודות:**

**public WAVLTree()**(בנאי): בנאי המחלקה לא מקבל ארגומנטים ויוצר עץ ריק ששורשו סנטינל. סבוכיות 0(1).

**Empy()**: מחזירה TRUE אם שורש העץ הוא סנטינל. סבוכיות 0(1).

**Private WAVLNode treePosition(int k)**: מתודת עזר המקבלת מפתח k ומחזירה עצם WAVLNode. משמשת את המתודות insert,delete,search. המתודה פועלת באופן דומה לחיפוש בעץ בינארי אך בניגוד למתודות שאותן היא משמשת היא מחזירה את הצומת עצמו ולא מפתח או ערך. סיבוכיות ריצה כתלות בגובה העץ: O(log n).

**Public String search(int k)**: המתודה מחזירה את הערך של הצומת שהמפתח שלו הוא K או את המחרוזת "-1" אם לא קיים כזה. המתודה נעזרת ב treePosition(int k) ובנוסף מבצעת O(1) פעולות נוספות. סה"כ סיבוכיות O(log n).

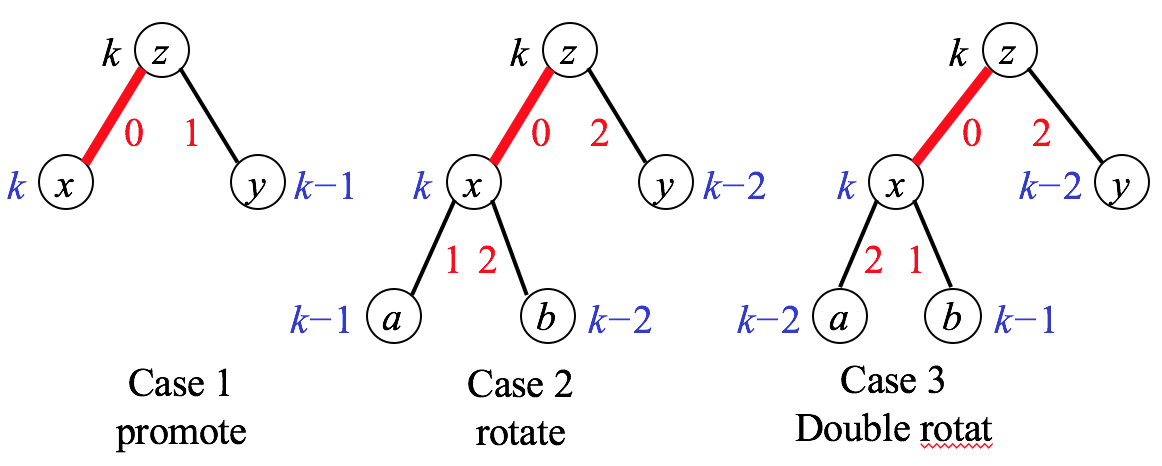
**Private int promote(WAVLnode node)**: המתודה מקבלת צומת בעץ, מעלה את דרגתו ב- 1 ומחזירה 1 (פעולת איזון יחידה). משמשת כפונקציית עזר של insertRebalance(),deleteRebalance(). סיבוכיות זמן O(1).

**Private int demote(WAVLnode node)**: סימטרי ל promote.

**Private int leftRotate/rightRotate(WAVLNode node, boolean isInsert)**: המתודה מקבלת צומת בעץ שמהווה "ציר" סיבוב. כאשר הסיבוב מתבצע כך שלבסוף הצומת node הופכת לאבא של ההורה הקודם שלה. אופן ביצוע הסיבוב זהה לזה המפורט במצגת. במקרה בו הסיבוב מתבצע לאחר מחיקה (isInsert==false) תתבצע פעולת promote ל- node בסוף בסיבוב. המתודה מחזירה 1 (פעולת איזון יחידה). המתודה מהווה מתודת עזר למתודות deleteRebalance,InsertRebalance. במהלך הסיבוב מתבצע מספר קבוע של החלפות פוינטרים ושינוי דרגות ולכן סיבוכיות זמן היא O(1).

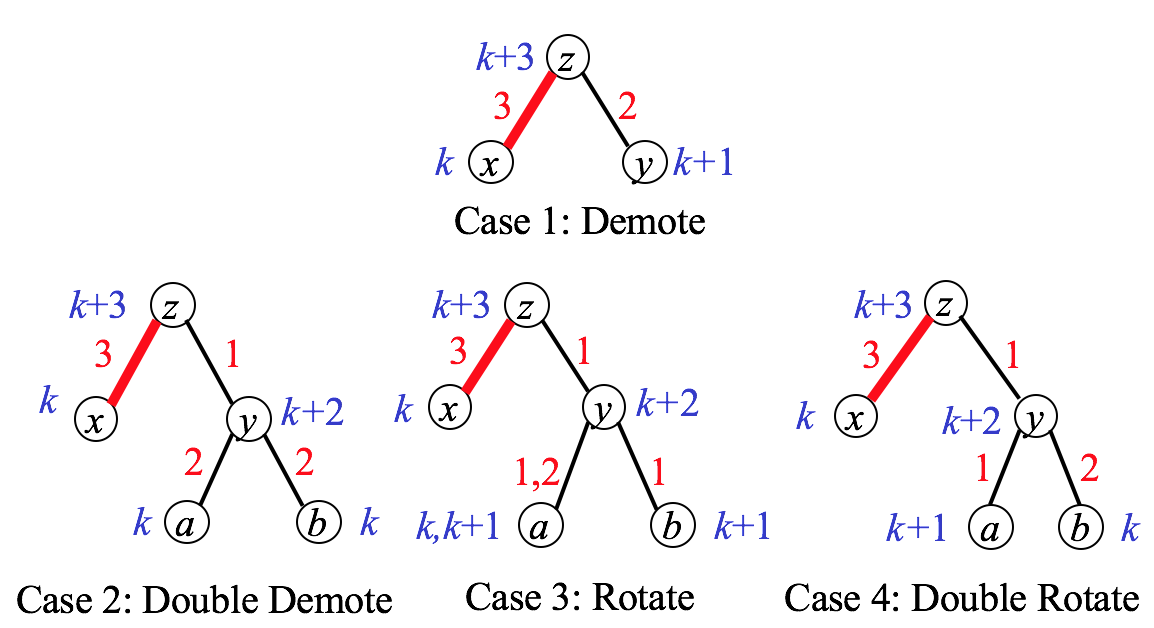
**Private int leftDoubleRotate/rightDoubleRotate(WAVLNode node, boolean isInsert)**: המתודה מקבלת צומת בעץ ומבצעת שני סיבובים יחידים בכיוון הפוך אחת מהשניה, כתלות בכיוון הסיבוב הכפול. המתודה מחזירה 2 (שתי פעולות סיבוב). המתודה מהווה מתודת עזר למתודות deleteRebalance,InsertRebalance. במהלך שני הסיבובים מתבצע מספר קבוע של החלפות פוינטרים ושינוי דרגות ולכן סיבוכיות זמן היא O(1).

**Private int insertRebalance(WAVLNode node)**: המתודה מקבלת צומת בעץ שהחל ממנו ולכל היותר עד לשורש מתבצעות פעולות האיזון על העץ לפי המקרים (ובאופן סימטרי לכיוון ההפוך):



המתודה מתחזקת מונה שסופר את מספר פעולות האיזון (promote=rotate=1, double rotate=2) בצורה רקורסיבית כאשר מקרה הבסיס הוא כאשר אין צורך בפעולות איזון ואז המתודה תחזיר 0. כלל הנסיגה הוא כל אחד מהמקרים הנ"ל. לצורך ביצוע האיזונים המתודה קוראת ל **demote,rotate,doubleRotate** הפועלות כולן בסיבוכיות O(1). במקרה הגרוע נדרשות מספר פעולות איזון כגובה העץ ולכן סיבוכיות המתודה היא O(log n).

**Private int deleteRebalance(WAVLNode node)**: המתודה מקבלת צומת בעץ שהחל ממנו ולכל היותר עד לשורש מתבצעות פעולות האיזון על העץ לפי המקרים (ובאופן סימטרי לכיוון ההפוך):



המתודה מתחזקת מונה שסופר את מספר פעולות האיזון (promote=demote=rotate=1, doubleRotate=2) בצורה רקורסיבית כאשר מקרה הבסיס הוא כאשר אין צורך בפעולות איזון ואז המתודה תחזיר 0. כלל הנסיגה הוא כל אחד מהמקרים הנ"ל. לצורך ביצוע האיזונים המתודה קוראת ל demote,rotate,doubleRotate הפועלות כולן בסיבוכיות O(1). במקרה הגרוע נדרשות מספר פעולות איזון כגובה העץ ולכן סיבוכיות המתודה היא O(log n).

**Private void increaseTreeSize(WAVLNode node):** המתודה מקבלת צומת בעץ ומגדילה את השדה subTreeSize שלו ושל כל האבות שלו ב-1. במקרה הגרוע נצטרך לבצע את ההגדלה במסלול שמתחיל בעלה ונגמר בשורש ולכן הסיבוכיות היא O(log n).

**Public int insert(int k, String i):** המתודה מקבלת מפתח וערך שהמשתמש מבקש להכניס לעץ וקוראת למתודת העזר treePosition שמחזירה פוינטר לצומת שיהיה ההורה של הצומת המוכנס. במקרה בו המפתח k כבר קיים בעץ, המתודה מחליפה את הערך i של הצומת ומחזירה 1-. בסוף ההכנסה מתבצעת קריאה למתודה insertRebalance(node), כאשר node הוא ההורה של הצומת שהכנסנו, ומוחזר מספר פעולות האיזון שבוצעו. המתודה מבצעת מספר קבוע של פעולות אך קוראת גם ל treePosition , increaseTreeSize ו- insertRebalance שסיבוכיות כל אחת מהן היא O(log n) לכן סה"כ הסיבוכיות היא:

O(log n) + O(1) + O(log n) + O(log n) = O(log n)

**Private WAVLNode findMinNode(WAVLNode node):** המתודה מקבלת צומת בעץ והחל ממנו עוקבת אחר כל הבנים השמאליים של node עד הגעה לצומת עם הערך המינימילי ומחזירה אותו. במקרה הגרוע node הוא שורש העץ ונבצע פעולות getLeft עד הגעה לעלה הכי שמאלי, כלומר נעבור על כל גובה העץ ולכן הסיבוכיות היא O(log n).

**Private WAVLNode findMaxNode(WAVLNode node):** סימטרי ל findMinNode.

**Private void decreaseTreeSize(WAVLNode node):** המתודה מקבלת צומת בעץ ומקטינה את השדה subTreeSize שלו ושל כל האבות שלו ב-1. במקרה הגרוע נצטרך לבצע את ההקטנה במסלול שמתחיל בעלה ונגמר בשורש ולכן הסיבוכיות היא O(log n).

**Public int delete(int k):** המתודה מקבלת מפתח שאותו נרצה להסיר מהעץ, מוצאת את מיקומו ע"י קריאה ל treePosition ופועלת באופן רקורסיבי למחיקתו מהעץ:

מקרה הבסיס הוא כאשר הצומת הוא לא צומת בינארי, ואז ניתן למחוק באמצעות "מעקף".

כלל הרקורסיה הוא החלפת node בעוקב שלו ע"י שימוש ב successor ואז מחיקת הצומת העוקב.

לאחר סיום המחיקה המתודה קוראת ל deleteRebalance המאזנת מחדש את העץ במקרה שהאיזון הופר.

המתודה נעזרת במתודות treePosition, successor, deleteRebalance שסיבוכיותן היא O(log n) ובנוסף מבצעת מספר קבוע של פעולות ולכן הסיבוכיות הכוללת של המתודה היא O(log n).

**Public String min():** המתודה מחזירה את הערך של הצומת השמור בשדה min. סיבוכיות O(1).

**:Public String max()** המתודה מחזירה את הערך של הצומת השמור בשדה max. סיבוכיות O(1).

public int keysToArray(): המתודה יוצרת מערך ריק בגודל העץ, ומאתלת צומת WAVLNode node המצביע על הצומת המינימלי בעץ (דרך שדה min). המתודה מכניסה את המפתח של node למערך ו"מקדמת" את node ע"י קריאה למתודה successor(). מכיוון שהמתודה עוברת על כל הצמתים בעץ סיבוכיות הזמן היא O(n).

**public String infoToArray():** זהה לkeysToArray, אך מכניס למערך את הערכים במקום את המפתחות. סיבוכיות O(n).

**public int size():** מחזירה את ערך השדה subTreeSize של שורש העץ עליו הופעלה המתודה. סיבוכיות O(1).

**Public WAVLNode getRoot():** מחזירה את ערך השדה root של העץ עליו הופעלה המתודה. סיבוכיות O(1).

**Public String select(int i):** בונה מערך ערכים ע"י קריאה למתודה infoToArray ומחזירה את הערך שבמקום ה- i-1 במערך – זהו הערך של האיבר ה-i בגודלו. סיבוכיות המתודה היא O(n) כתוצאה מהשימוש ב infoToArray.

**Public void setRoot(WAVLNode root):** מקבלת אובייקט מטיפוס צומת וקובעת אותו להיות שורש העץ עליו מופעלת המתודה. סיבוכיות O(1).

**מחלקת WAVLNode:**

**שדות:**

**private int key**: מפתח הצומת

**private String value**: ערך הצומת

**private WAVLNode left**: הבן השמאלי של הצומת

**private WAVLNode right**: הבן הימני של הצומת

**private WAVLNode parent**: ההורה של הצומת

**private int rank**: דרגת הצומת

**private int subTreeSize**: גודל תת העץ ששורשו הוא הצומת, כולל הצומת

**מתודות:**

**בנאים:**

* Public WAVLNode(): בנאי ריק, יוצר צומת וירטואלי שדרגתו 1-.
* Public WAVLNode(int key, String value): יוצר צומת המכיל מפתח וערך.
* Public WAVLNode(int key, String value, WAVLNode parent): יותר צומת המכיל מפתח וערך וקובע בנוסף את אבא של הצומת.

**Getters , Setters:** ממומשים כמתואר במנשק, כולל מתודות שאנחנו הוספנו שאינן במנשק לשדות rank, parent.

**Public WAVLNode successor(WAVLTree tree):** המתודה פועלת על מופע צומת בעץ ומקבלת את העץ כארגומנט בנוסף ומחזירה את העוקב של מופע הצומת. אם ל node אין בן ימני, המתודה תחזיר את הצומת המוחזר ע"י findMinNode(node). אחרת, המתודה "תעלה" בעץ עד שתמצא צומת שהיא בן שמאלי של ההורה שלה ותחזיר את אותו הורה. במקרה הגרוע המתודה תצטרך לעבור את כל גובה העץ ולכן הסיבוכיות היא O(log n).

**Public WAVLNode predeccessor(WAVLTree tree):** סימטרי ל successor.

מדידות:

לפי החומר התיאורטי, נצפה לתוצאות המצביעות על זמן אמורטייזד של פעולות איזון O(1), נתבונן בטבלת המדידות:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| מספר סידורי | מספר פעולות איזון לאחר הכנסה | מספר פעולות איזון לאחר מחיקה | 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 | 29967 | 9976 | 2.9967 | 0.9976 | 13 | 6 |
| 2 | 59965 | 19974 | 5.9965 | 1.9974 | 14 | 7 |
| 3 | 89963 | 29972 | 8.9963 | 2.9972 | 15 | 7 |
| 4 | 119963 | 39972 | 11.9963 | 3.9972 | 15 | 7 |
| 5 | 149962 | 49970 | 14.9962 | 4.997 | 16 | 7 |
| 6 | 179961 | 59970 | 17.9961 | 5.997 | 16 | 7 |
| 7 | 212202 | 69970 | 21.2202 | 6.997 | 16 | 8 |
| 8 | 243087 | 79970 | 24.3087 | 7.997 | 16 | 8 |
| 9 | 254640 | 89970 | 25.464 | 8.997 | 16 | 8 |
| 10 | 283480 | 99968 | 28.348 | 9.9968 | 17 | 8 |

* מספר פעולות האיזון לאחר מחיקה קרוב מאוד ל O(1) לפעולה
* מספר פעולות האיזון לאחר הכנסה גדולה בערך פי 3 ממספר הפעולות לאחר מחיקה, מכין שמדובר בהכפלה בקבוע התוצאות עדיין מתאימות למצופה – O(1) לפעולה.
* מספר פעולות האיזון המקסימלי באמת מתאים לעומק העץ O(log n).