תיעוד - WAVL Tree

WAVLTree:

לאיבר במחלקה 2 שדות פרטיים:
- WAVLNode root – שורש העץ
- WAVLNode EXT – צומת וירטואלי לייצוג עלה חיצוני

- **WAVLTree():**
  - בנאי המחלקה. מבצע השמה של שדה השורש ל-null, של שדה דרגת EXT ל-1- ושל שדה גודל של EXT ל-0.
  - סיבוכיות זמן ריצה O(1)

- **empty()**:
  - בודקת האם העץ ריק לפי השורש. אם הוא null אז העץ ריק ותחזיר true, אחרת תחזיר false.
  - סיבוכיות זמן ריצה O(1)

- **search(int k)**:
  - מחפשת צומת עם מפתח k בעץ. אם קיים, מחזירה את הערך ששמור בו, אחרת מחזירה null.
  - סיבוכיות זמן ריצה – O(logn)

- **insert(int k, String i)**:
  - יוצרת צומת עם מפתח k וערך i ומנסה להכניס אותו לעץ. אם כבר קיים צומת עם מפתח זה, מחזירה -1, אחרת מחזירה את מספר פעולות האיזון שהתבצעו לאחר הכנסת הצומת.
  - פונקציות עזר:
    - treePosition על מנת למצוא את מיקום ההכנסה.
    - updateSizes לעדכון גדלי הצמתים.
    - rebalanceInsert לאיזון העץ לאחר ההכנסה.
  - סיבוכיות זמן ריצה – O(logn)

- **treePosition(int k):**
  - הפונקציה עובדת בצורה זהה לחיפוש. אם האיבר כבר קיים, מחזירה את מיקום האיבר בעץ. אחרת, תחזיר את מיקום ההורה שאליו צומת עם מפתח k יחובר.
  - סיבוכיות זמן ריצה - O(logn)

- **connectNode(WAVLNode parent, WAVLNode node):**
  - מחברת את node להיות הבן המתאים של parent כתלות במפתחות שלהם. אם parent הוא null, הופכת את שורש העץ ל-node.
  - סיבוכיות זמן ריצה – O(1)

- **connectNode(WAVLNode parent, WAVLNode node, WAVLNode childToReplace):**
  - מחברת בין node ל parent במקום הבן הנוכחי ChildToReplace
  - סיבוכיות זמן ריצה – O(1)

- **rebalanceInsert(WAVLNode parent):**
  - מאזנת את העץ במיקום של הצומת parent לאחר הכנסת איבר. בודקת את המצב הנוכחי שהעץ נמצא בו ופועלת בהתאם לפי המצבים שלמדנו בכיתה. רצה בלולאה עד ביצוע פעולה סופית או עד הגעה לשורש. בכל איטרציה מקדמת את parent במעלה העץ.
  - פונקציות עזר:
    - singleRotation
    - doubleRotation
  - סיבוכיות זמן ריצה – O(logn)

- **singleRotation(WAVLNode parent, WAVLNode child, String direction):**
  - מבצעת פעולת סיבוב (כפי שנלמדה בהרצאה 4) בין parent ו-child. הכיוון מוגדר לפי המחרוזת direction.
  - סיבוכיות זמן ריצה – O(1)

- **doubleRotation(WAVLNode parent, WAVLNode child, String direction):**
  - מבצעת פעולת סיבוב כפולה (כפי שנלמדה בהרצאה 4) בעזרת 2 פעולות singleRotation בין parent ו-child. הכיוון מוגדר לפי המחרוזת direction.
  - פונקציות עזר:
    - singleRotation
  - סיבוכיות זמן ריצה – O(1)

- **updateNodeSize(WAVLNode node):**
  - מגדירה את הגודל של node לפי הגדלים של הבן הימני והשמאלי שלו.
  - סיבוכיות זמן ריצה - O(1)

- **delete(int k)**:
  - מחפשת את מיקום האיבר עם מפתח k בעץ בעזרת treePosition(k). במידה והוא לא בעץ הפונקציה לא עושה כלום, אחרת:
    אם יש לו 2 ילדים היא מחליפה בינו לבין האיבר העוקב בעץ בעזרת replaceWithSuccessor(node). לאחר מכן הפונקציה מוחקת את האיבר ומעדכנת את הגדלים בעזרת updateSizes(node, 1). לבסוף נפעיל את rebalanceDelete(node.parent) כדי לאזן את העץ.
  - פונקציות עזר:
    - treePosition
    - replaceWithSuccessor
    - updateSizes
    - rebalanceDelete
  - סיבוכיות זמן ריצה – O(logn)

- **rebalanceDelete(WAVLNode parent):**
  - מאזנת את העץ במיקום הצומת parent לאחר מחיקת איבר. בודקת את המצב הנוכחי שהעץ נמצא בו ופועלת בהתאם לפי המצבים שלמדנו בכיתה. רצה בלולאה עד ביצוע פעולה סופית או עד הגעה לשורש. בכל איטרציה מקדמת את parent במעלה העץ. מוודאת שלא נוצר עלה מדרגה 1.
  - פונקציות עזר:
    - singleRotation
    - doubleRotation
  - סיבוכיות זמן ריצה – O(logn)

- **replaceWithSuccessor(WAVLNode node):**
  - מחליפה מקומות בעץ בין node לאיבר העוקב שלו. משמשת את הפונקציה delete כאשר מנסים למחוק צומת עם 2 בנים. במקרה זה העוקב תמיד יהיה צומת אונארי או עלה.
  - פונקציות עזר:
    - successor
  - סיבוכיות זמן ריצה - O(logn)

- **onlyChild(WAVLNode succ):**
  - מקבלת עלה או צומת אונארי ומחזירה את הילד היחיד שלו (במקרה של עלה היא תחזיר את EXT).
  - סיבוכיות זמן ריצה - O(1)

- **min():**
  - מחזירה את הערך של הצומת עם המפתח הכי קטן בעץ. כדי למצוא את הצומת הזה משתמשת ב-minNode ומחזירה את ערכו.
  - סיבוכיות זמן ריצה - O(logn)

- **minNode(WAVLNode node):**
  - מחזירה את הצומת בעל המפתח הכי קטן בעץ. המימוש כמו שראינו בהרצאה 3.
  - סיבוכיות זמן ריצה - O(logn)

- **max():**
  - מחזירה את הערך של הצומת עם המפתח הכי גדול בעץ. כדי למצוא את הצומת הזה נשתמש ב-maxNode ונחזיר את ערכו.
  - סיבוכיות זמן ריצה - O(logn)

- **maxNode(WAVLNode node):**
  - מחזירה את הצומת בעל המפתח הכי גדול בעץ. המימוש כמו שראינו בהרצאה 3.
  - סיבוכיות זמן ריצה - O(logn)

- **keysToArray**():
  - מחזירה מערך ממוין של כל מפתחות העץ. הפונקציה מוצאת את האיבר המינימלי בעזרת minNode ומוסיפה את המפתח שלו למערך. לאחר מכן מבצעת לולאה עם מספר איטרציות

כגודל העץ, שבה משתמשת ב-successor כדי למצוא את האיבר הבא ומוסיפה את המפתח שלו למערך.
  - פונקציות עזר:
    - minNode
    - successor
  - סיבוכיות זמן ריצה - פעולת הפונקציה שקולה לריצה in-order על העץ, לכן כמו שראינו בתרגול 4 הסיבוכיות היא O(n).

- **successor(WAVLNode node):**
  - מחזירה את האיבר עם המפתח העוקב למפתח של node. אם הוא לא קיים מחזירה null. המימוש כמו שראינו בהרצאה 3.
  - סיבוכיות זמן ריצה - O(logn)

- **infoToArray()**:
  - מחזירה מערך ממוין של כל ערכי הצמתים לפי מפתחות העץ. הפונקציה מוצאת את האיבר המינימלי בעזרת minNode ומוסיפה את ערכו למערך. לאחר מכן מבצעת לולאה עם מספר איטרציות כגודל העץ, שבה משתמשת ב-successor כדי למצוא את האיבר הבא ומוסיפה את הערך שלו למערך.
  - פונקציות עזר:
    - minNode
    - successor
  - סיבוכיות זמן ריצה - פעולת הפונקציה שקולה לריצה in-order על העץ, לכן כמו שראינו בתרגול 4 הסיבוכיות היא O(n).

- **size()**:
  - מחזירה את גודל העץ בעזרת getSubtreeSize על השורש. אם העץ ריק מחזירה 0.
  - סיבוכיות זמן ריצה - O(1)

- **updateSizes(WAVLNode node, int a):**
  - מעדכנת את הגודל של כל הצמתים מ-node ועד לשורש בהוספה או החסרת 1, בהתאם ל-a.
  - סיבוכיות זמן ריצה - O(logn)

- **getRoot**:
  - מחזירה את שורש העץ - השדה root.
  - סיבוכיות זמן ריצה – O(1)

- **select(IWAVLNode x, int i)**:
  - אם העץ ריק מחזירה 1-, אחרת קוראת ל-subSelect על השורש והמספר i.
  - פונקציות עזר:
    - subSelect
  - סיבוכיות זמן ריצה – O(logn)

- **subSelect(IWAVLNode x, int i):**
  - מחזירה את ערכו של האיבר שהמפתח שלו הוא ה-i+1 בגודלו בעץ, כלומר יש i איברים עם מפתחות קטנים יותר משלו. המימוש כמו שראינו בהרצאה 4.
  - סיבוכיות זמן ריצה – O(logn)

WAVLNode:

לאיבר במחלקה 7 שדות פרטיים:

- String info – ערך הצומת
- Int key – מפתח
- Int rank – דרגה
- WAVLNode parent – מצביע לצומת האבא (ברירת מחדל – null)
- WAVLNode left – מצביע לבן השמאלי (ברירת מחדל – EXT)
- WAVLNode right – מצביע לבן הימני (ברירת מחדל – EXT)
- Int size – גודל תת העץ שהצומת הוא השורש שלו

- **WAVLNode(int key, String info):**
  - בנאי המחלקה. מבצע השמה של שדה המפתח ל-key, של שדה ערך ל-info, של שדה הדרגה ל-0 ושל שדה הגודל ל-1.
  - סיבוכיות זמן ריצה – O(1)

- **getKey()**:
  - מחזירה את המפתח מהשדה key.
  - סיבוכיות זמן ריצה – O(1)

- **getValue()**:
  - מחזירה את הערך מהשדה info.
  - סיבוכיות זמן ריצה – O(1)

- **getLeft()**:
  - מחזירה את הבן השמאלי מהשדה left.
  - סיבוכיות זמן ריצה – O(1)

- **getRight()**:
  - מחזירה את הבן הימני מהשדה right.
  - סיבוכיות זמן ריצה – O(1)

- **isRealNode()**:
  - מחזירה false אם הצומת לא EXT, אחרת true. הבדיקה מתבצעת ע"י השוואה לאובייקט EXT ובדיקה האם הדרגה היא 1-.

- סיבוכיות זמן ריצה – O(1)

- **getSubtreeSize():**
  - מחזירה את גודל תת העץ שהצומת הוא השורש שלו מהשדה size.
  - סיבוכיות זמן ריצה – 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	10000	2.48	1.38	16	7
2	20000	2.5	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	17	9
6	60000	2.48	1.39	18	9
7	70000	2.48	1.39	18	10
8	80000	2.48	1.39	19	10
9	90000	2.48	1.39	19	10
10	100000	2.48	1.39	19	10

על סמך מה שראינו בהרצאה, ציפינו לראות שמספר פעולות האיזון לפעולת insert ו-delete יהיה O(logn) עבור עץ עם n צמתים, ו-amortized O(1) עבור כל סדרה של פעולות הוספה ומחיקה. אכן ניתן לראות שעבור כל גודל עץ, במקרה הכי גרוע פעולת insert/delete מבצעת O(logn) פעולות איזון.
בנוסף, עבור סדרה של n פעולות insert/delete, כל פעולה מבצעת בממוצע 1.39 או 2.48 פעולות איזון בהתאמה, בלי תלות בגודל העץ או במספר הפעולות בסדרה, וזה סדר גודל קבוע.