**תרגיל מעשי 1 – מבני נתונים**

**מגישים:**

tamirdennis – 208538702 – טמיר דניס

rotemtzaban - 31539406 – רתם צבן

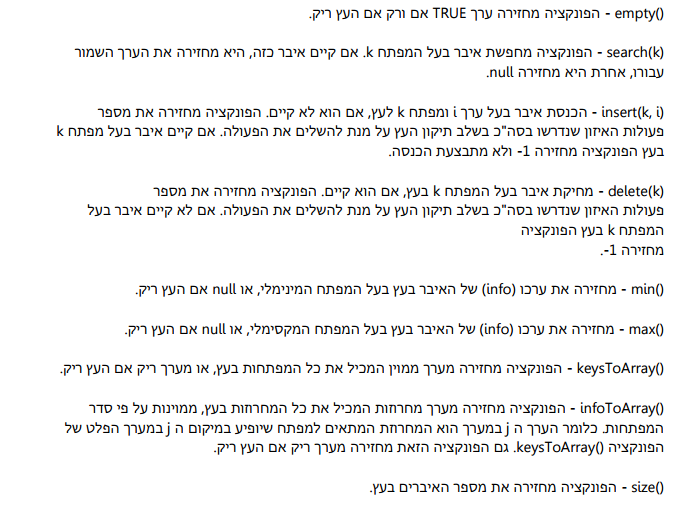
**תיאור מחלקת WAVLTree:**

המחלקה מממשת את מבני הנתונים עץ WAVL. עץ WAVL הינו עץ בינארי מאוזן. לכל איבר בעץ יש ערך (info) מסוג מחרוזת ומפתח (key) מסוג מספר שלם. כל המפתחות שונים זה מזה. מהגדרת עץ WAVL תכונותיו הן שלכל צומת המכיל ערך ומפתח יש הפרש דרגות (ranks) בין שתי ילדיו בעץ. עם זאת תמיד בסוף העץ יהיו רק צמתים שנקראים צמתים חיצוניים ללא מידע ובעלי דרגה שהיא -1. תמיד בעץ WAVL אם מסמנים את ההפרש של של צומת לא חיצוני בין ילדו הימני ב- x ובין ילדו השמאלי ב-y אז (x,y) בעץ כזה חייב להיות (1,1) או (1,2) או (2,1) או (2,2) חוץ מהעלים בעץ (עלים הם צמתים בעלי שני ילדים חיצוניים) שתמיד יהיו עם הפרשים (1,1) עם ילדיהם. עץ אשר מקיים את התנאים הללו הינו עץ מאוזן עם גובה שהוא O(log n) כאשר n מספר הצמתים בעץ.

למחלקה ישנה תת מחלקה WAVLNode פרטית עם השדות int key, String info, int rank, WAVLNode parent, WAVLNode left, WAVLNode right ועם מתודות עזר פשוטות שלוקחות O(1).

שדות המחלקה הם WAVLNode root,int size, WAVLNode min, WAVLNode max, WAVLNode external. כך ש- external הוא final ומייצג את כל הצמתים החיצוניים לכן הוא ילד של הרבה צמתים אבל אין לו parent והוא עם rank = -1.

במחלקה WAVLTree למשתמש ניתנת האפשרות להשתמש בפעולות הבאות:



הסברים נוספים מופיעים בתיעוד הקוד.

**סיבוכיות:**

empty(), min(), max(), size(): O(1)

search: O(log n)

הפונקציה findNearestNode, לוקחת O(log n) זמן, מכיוון שהיא מתחילה בראש העץ ויכולה לרדת עד הקצה, וגובה העץ הוא O(log n).

min, max: O(1)

KeysToArray: O(n)

המתודה הרקורסיבית שנקראת ממנה תעבור על כל אחד מהקודקודים בעץ בדיוק פעם אחת.

infoToArray: O(n)

delete, insert : O(log n)

הקריאה לfindNearestNode לוקחת O(log n) זמן, ולאחר מכן פעולות האיזון לוקחות O(log n) בworst case, וO(1) ב-amortized. פעולות האיזון נעשות במתודות הפרטיות שנקראות ב insert וdelete ששמן והסיבוכיות שלהן הינו:

rebalanceInsert : O(log n)

rebalanceDelete: O(log n)

כל אחת מהמתודות האלה פועלות מתחתית העץ – בצומת אונרי או עלה, ובמקרה הגרוע ממשיכות עד לשורש העץ בצורה רקורסיבית. לכן מספר הפעולות בהן הוא כגובה עץ WAVL כלומר O(log n).

כדאי לציין גם שהפונקציה rebalanceDelete קוראת לפונקצית עזר removeNodeFromTree לצומת מתאים על מנת להסיר מצביעים אל צומת מסויים בעץ וכך למחוק אותו מהעץ – פעולות אלו לוקחות O(1) זמן. בנוסף rebalanceDelete קוראת במידת הצורך ל- findSuccessor עם אחת הצמתים בעץ. פעולה זו יכולה להתחיל מהשורש ולמצוא successor רק בתחתית העץ ולכן לוקחת O(log n) במקרה הגרוע.

טבלת מדידות :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| מספר סידורי | מספר פעולות | מספר פעולות האיזון הממוצע לפעולת insert | מספר פעולות האיזון הממוצע לפעולת delete | מספר פעולות האיזון המקסימלי לפעולת delete | מספר פעולות האיזון המקסימלי לפעולת insert |
| 1 | 10000 | 3.41 | 3 | 12 | 15 |
| 2 | 20000 | 3.42 | 3 | 14 | 18 |
| 3 | 30000 | 3.40 | 3 | 14 | 17 |
| 4 | 40000 | 3.41 | 3 | 14 | 17 |
| 5 | 50000 | 3.39 | 3 | 13 | 18 |
| 6 | 60000 | 3.41 | 3 | 14 | 18 |
| 7 | 70000 | 3.41 | 3 | 14 | 18 |
| 8 | 80000 | 3.41 | 3 | 14 | 20 |
| 9 | 90000 | 3.41 | 3 | 15 | 19 |
| 10 | 100000 | 3.4 | 3 | 15 | 21 |

למדנו בכיתה שמספר הפעולות הamortized של פעולות rebalance בפעולות insert וdelete הוא O(1)

לכן ציפינו שמספר הפעולות הממוצע לאורך הריצות ישאר פחות או יותר קבוע.

מספר הפעולות המקסימלי האפשרי הוא O(log n), מכיוון שזהו גובה העץ, ולכל רמה בעץ מתבצע מספר סופי של פעולות איזון, לכן ציפינו שמספר הפעולות המקסימלי יהיה O(log n)

ניתן לראות כי אכן מספר הפעולות הממוצע נשאר קבוע, ומספר הפעולות המקסימלי עולה בקצב איטי, שנראה כמו O(log n)