**מבנה נתונים – תרגיל מעשי 1**

**תיעוד הפונקציות:** (הסיבוכיות המוצגת היא worst-case)

**מחלקת WAVLTree:**

שדות:

* root – שורש העץ.
* externalLeaf – מייצג עלה חיצוני.
* min – הצומת בעל המפתח המינימלי בעץ.
* max – הצומת בעל המפתח המקסימלי בעץ.

מתודות:

* NodeDirection – מגדירה משתנה מסוג NodeDirection, המייצג כיוון, שמקבל את הערכים Left ו-Right.
* WAVLTree – בנאי, מייצר עץ ריק.
* empty – מחזיר true אמ"מ העץ ריק. סיבוכיות: O(1)
* search – מחפשת את המפתח k בעץ, מחזירה את הערך שלו אם הוא קיים בעץ, null אחרת. סיבוכיות:O(logn)
* insert – מכניסה את המפתח k לעץ, וקוראת לפונקציה insertionBalance, שמאזנת את העץ (במידת הצורך). מחזירה את מספר פעולות האיזון שבוצעו. סיבוכיות: O(logn)
* delete – מוחקת את המפתח k מהעץ, וקוראת לפונקציה deletionBalance, שמאזנת את העץ (במידת הצורך). מחזירה את מספר פעולות האיזון שבוצעו. סיבוכיות: O(logn)
* min/max – מחזיר את הערך של המפתח המינימלי/מקסימלי בעץ, null אם העץ ריק. סיבוכיות: O(1)
* getMinNode/getMaxNode – מחזיר את הצומת בעל המפתח המינימלי/מקסימלי. סיבוכיות: O(1)
* inOrderWalk – מחזירה מערך שמכיל את כל הצמתים בעץ, ממויינים לפי מפתחות. סיבוכיות: O(n)
* keysToArray/infoToArray – מחזיר מערך של מפתחות/ערכים של צמתי העץ, ממוין לפי סדר המפתחות. אם העץ ריק מוחזר null. הפונקציות נעזרות ב-inOrderWalk. סיבוכיות: O(n)
* size – מחזיר את מספר האיברים בעץ. נעזרת בפונקציה getSubtreeSize, שמחזירה את מספר האיברים בתת עץ ששורשו הוא צומת נתון. סיבוכיות: O(1)
* getRoot – מחזירה את שורש העץ. סיבוכיות: O(1)
* setRoot – מחליפה את שורש העץ. סיבוכיות: O(1)
* select – מחזירה את האיבר ה-i בגודלו בעץ, null אם לא קיים כזה. סיבוכיות: O(logn)

**מחלקת WAVLNode:**

שדות:

* key – המפתח של צומת.
* value – המידע שמאוחסן בצומת.
* left – הבן השמאלי.
* right – הבן הימני.
* parent – האבא של הצומת, null אם הצומת הוא שורש.
* size – גודל תת העץ ששורשו הוא הצומת.
* rank – הדרגה של הצומת.

מתודות:

* WAVLNode – בנאי, מייצר צומת חדש.
* isRoot – מחזיר את שורש העץ שהצומת הוא חלק ממנו. סיבוכיות: O(1)
* setParent – משנה את האבא של הצומת. סיבוכיות: O(1)
* getKey – מחזיר את המפתח של הצומת, 1- אם הוא צומת חיצוני. סיבוכיות: O(1)
* getRank – מחזיר את הדרגה של הצומת. סיבוכיות: O(1)
* promotion – מקדם את הדרגה של הצומת ב-1. סיבוכיות: O(1)
* demotion – מוריד את הדרגה של הצומת ב-1. סיבוכיות: O(1)
* getValue – מחזיר את הערך של הצומת, null אם הוא צומת חיצוני. סיבוכיות: O(1)
* getLeft/getRight – מחזיר את הבן השמאלי/ימני של הצומת. נעזרת במתודה getChild, שמקבלת כיוון ומחזירה את הילד המתאים. סיבוכיות: O(1)
* isInnerNode – מחזיר true אמ"מ הצומת הוא צומת פנימי. סיבוכיות: O(1)
* isExternalNode – מחזיר true אמ"מ הצומת הוא צומת חיצוני. סיבוכיות: O(1)
* getSubtreeSize – מחזיר את מספר הצמתים בתת העץ ששורשו הוא הצומת. סיבוכיות: O(1)
* setSubtreeSize – מעדכן את השדה size. סיבוכיות: O(1)
* incSubtreeSize – מגדיל את size ב-1. סיבוכיות: O(1)
* decSubtreeSize – מקטין את size ב-1. סיבוכיות: O(1)
* updateSubtreeSize – מעדכן את השדה size על סמך גודל תתי העצים של הבנים. סיבוכיות: O(1)
* successor/predecessor – מחזיר את הצומת העוקב/הקודם של הצומת הנוכחי (לפי מפתח). אם לא קיים, מחזיר null. נעזר בפונקציה dCessor. סיבוכיות: O(logn)
* dCessor – פונקציית עזר, שמחזירה את האיבר העוקב/קודם, בהתאם לכיוון שקיבלה (ימין/שמאל בהתאמה). סיבוכיות: O(logn)
* getParent – מחזיר את האבא של הצומת. סיבוכיות: O(1)
* setChild – מעדכן את הבן הימני/שמאלי של הצומת להיות child בהתאם לכיוון שקיבל. סיבוכיות: O(1)
* getChild – מחזיר את הבן הימני/שמאלי של הצומת בהתאם לכיוון שקיבל. סיבוכיות: O(1)
* isLeaf – מחזיר true אמ"מ הצומת הוא עלה. סיבוכיות: O(1)
* switchWith – מחליף בין המיקום של שני צמתים בעץ, כשלב ביניים לפני מחיקה. מיועד להפעלה עם צומת עוקב. סיבוכיות: O(1)
* getRankDiff – מחזיר את הפרש הדרגות בין הצומת לבין הבן הימני/שמאלי בהתאם לכיוון d. סיבוכיות: O(1)
* getOppositeDirection – מחזיר את הכיוון המנוגד לכיוון d. סיבוכיות: O(1)
* getParentDirection – מחזיר את הכיוון של הצומת ביחס לאבא שלו (שמאל אם הוא בן שמאלי). סיבוכיות: O(1)
* rotate – מבצע סיבוב שמאלה/ימינה לפי הכיוון d, מעדכן את size של הצמתים שתת העץ שלהם משתנה, ואת שורש העץ הרלוונטי אם יש צורך. סיבוכיות: O(1)
* insertionBalance/deletionBalance – מאזן את העץ ומחזיר את מספר פעולות האיזון שנדרשו. סיבוכיות: O(logn), אמוטייזד: O(1)

מתודות רקורסיביות שמממשות את הפונקציות המקבילות להן ב-WAVLTree:

* searchNode – מחזיר את הערך של הצומת בעל המפתח k, או null אם הוא לא קיים. סיבוכיות: O(logn)
* insert – מוצא את מיקום ההכנסה של node ומכניס אותו לעץ, אם כבר קיים צומת בעל מפתח דומה, מחזיר 1- ולא משנה כלום. מגדיל ב-1 את size של הצמתים שעוברים דרכן במהלך חיפוש מקום ההכנסה. סיבוכיות: O(logn)
* delete – מוחק את הצומת בעל המפתח key, אם הוא לא קיים מחזיר 1- ולא משנה כלום. מקטין ב-1 את size של הצמתים שמושפעים מהמחיקה. סיבוכיות: O(logn)
* selectNode – מחזיר את הערך של המפתח ה-i בעץ הרלוונטי.

**מדידות:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| מספר סידורי | מספר פעולות | מספר פעולות האיזון הממוצע insert לפעולת | מספר פעולות האיזון המקסימלי insert לפעולת | מספר פעולות האיזון הממוצע לפעולת delete | מספר פעולות האיזון המקסימלי delete לפעולת |
| 1 | 10000 | 3.3967 | 15 | 2.6697 | 11 |
| 2 | 20000 | 3.4247 | 16 | 2.65845 | 11 |
| 3 | 30000 | 3.3986 | 18 | 2.672233333 | 12 |
| 4 | 40000 | 3.406425 | 18 | 2.6684 | 12 |
| 5 | 50000 | 3.4352 | 20 | 2.66622 | 12 |
| 6 | 60000 | 3.420183333 | 19 | 2.666666667 | 11 |
| 7 | 70000 | 3.430557143 | 18 | 2.664428571 | 11 |
| 8 | 80000 | 3.4067875 | 18 | 2.665875 | 12 |
| 9 | 90000 | 3.418177778 | 19 | 2.666611111 | 13 |
| 10 | 100000 | 3.41021 | 20 | 2.66839 | 13 |

ציפינו שהמספר הממוצע של פעולות איזון (לכל פעולת insert/delete) יהיה קבוע בסדרה רצופה של הכנסות או מחיקות ללא תלות בגודל הקלט. מהטבלה ניתן להסיק שזה אכן מה שקרה.

בנוסף, ניתן לראות שהמספר המקסימלי של פעולות איזון לפעולת insert בודדת (worst case) תלוי (כצפוי) בגודל הקלט (לוגריתמי).