**AVL Tree Implementation**

**שדות של AVLTree :**

**final static int *ZERO\_REBALANCES* = 0;**

קבוע סטטי סופי מספרי, מאותחל ל 0

**final static int *ONE\_REBALANCE* = 1;**

קבוע סטטי סופי מספרי, מאותחל ל 1

**final static int *TWO\_REBALANCES* = 2;**

קבוע סטטי סופי מספרי, מאותחל ל 2

**final static int *THREE\_REBALANCES* = 3;**

קבוע סטטי סופי מספרי, מאותחל ל 3

**final static int *FIVE\_REBALANCES* = 5;**

קבוע סטטי סופי מספרי, מאותחל ל 5

**final static int *SIX\_REBALANCES* = 6;**

קבוע סטטי סופי מספרי, מאותחל ל 6

**AVLNode *EXTERNAL\_NODE***

צומת וירטואלי

**AVLNode Root**

שורש העץ

**AVLNode Max**

איבר מקסימלי בעץ

**AVLNode Min**

איבר מינמלי בעץ

**פונקציות של AVLTree :**

|  |  |  |
| --- | --- | --- |
| **סיבוכיות** | **מה עושה** | **פונקציה** |
| O(1) | בנאי המתאחל עץ ריק, כך שבהתחלה הוא מאתחל את EXTERNAL\_NODE לצומת מדומה על ידי קריאה לבנאי בלי הפרמטרים של AVLNode. בנוסף מאתחל הבנאי מאתחל את השורש, מקסימום ומינימום לצומת עלה חיצוני | **public AVLTree** |
|  | מייצג את סוגי הקשתות | **Enum Edge** |
| O(1) | בודקת אם העץ ריק ע"י בדיקת ערך ההחזרה מהפעלת הפונקציה size() | **public boolean empty()** |
| O(logn) | מבצעת חיפוש עבור צומת בעץ לפי מפתח נתון k ומחזירה את ערכו ע"י קריאה ל- **helpsearch** (AVLNode, int) עם שורש של העץ והמפתח k | **public String search(int k)** |
| O(logn) | מחזירה את הצומת האחרון בו נתקלנו בחיפוש אחרי צומת עם מפתח נתון בתת עץ של צומת נתון לפי התהליך שלמדנו בהרצאה מספר 3, התהליך כולל חיפוש בינארי עם שמירה על הצומת אותו אנו מחפשים וגם על הצומת האחרון בו נפגשנו | **private AVLNode helpsearch(AVLNode node,int k)** |
| O(logn) | מכניסה צומת חדש עם מפתח וערך נתונים:   1. קוראת ל- empty() לבדיקת ריקנות של העץ, אם העץ ריק - מאתחלים את השורש, מקסימום ומינימום לצומת חדש עם המפתח והערך הנתונים וממשיכים. 2. אחרת, אם העץ לא ריק, 3. קוראת ל- helpSearch(AVLNode, int) עם שורש העץ והמפתח הנתונים בכדי למצוא את מיקום ההכנסה. 4. אם מפתח נקודת ההכנסה תואם למפתח הניתן כקלט אז האיבר כבר שוכב בעץ ולכן מחזירים מינוס אחד 5. אחרת: 6. עושים השוואה בין מפתח נקודת ההכנסה לבין מפתח הצומת הנתונה בכדי לבדוק את מיקום הצומת החדש, לפי בן שמאלי או ימיני, ומכניסים   הפונקציה קוראת ל NewInsertSize(AVLNode) עם נקודת ההכנסה בכדי לעדכן את הגדלים של כל האבות הקדמונים  כעת, קוראת ל-NewInsertMaxMin(AVLNode) עם הצומת החדש בכדי לעדכן את האיבר המינמלי והמקסימלי בעץ | **public int insert(int k, String i)** |
| O(logn) | מעדכנת את גדלי הצמתים בעץ אחרי פעולת הכנסה של צומת נתון, ע"י עלייה בעץ מהצומת עד השורש ומקטינה את הגודל שלהם ב 1. | **private void NewInsertSize(AVLNode point)** |
| O(1) | מעדכנת את הצומת המינמלי והמקסימלי בעץ לאחר פעולת הכנסה של צומת נתון:   1. אם מפתח הצומת המקסימלי קטן ממפתח הצומת שהכנסנו אז מעדכנים את המקסימום להיות הצומת שהכנסנו 2. אם מפתח הצומת המינמלי גדול ממפתח הצומת שהכנסנו אז מעדכנים את המינימום להיות הצומת שהכנסנו | **private void NewInsertMaxMin(AVLNode node)** |
| O(logn) | מבצעת איזון של העץ אחרי הכנסה במידת הצורך לפי התהליך שלמדנו בהרצאה 4, ע״י חלוקה למקרים של הפרשי הדרגות של צומת נתון ומחזירה את מספר הפעולות של האיזון אשר התבצעו | **private int InsertReBalance(AVLNode point)** |
| O(1) | מאזנת את מצב העץ אשר לאחר הכנסה כללה צומת נתון מסוג “20”, לפי התהליך שלמדנו בהרצאה 4 | **private int InsertReBalanceType20(AVLNode point)** |
| O(1) | מאזנת את מצב העץ אשר לאחר הכנסה כללה צומת נתון מסוג “02”, לפי התהליך שלמדנו בהרצאה 4 | **private int InsertReBalanceType02(AVLNode point)** |
| O(1) | מבצעת סיבוב ימינה לעץ עבור נקודה נתונה התהליך שלמדנו בהרצאה 3 | **private void RightRotate(AVLNode DesiredRotatePoint)** |
| O(1) | מבצעת סיבוב שמאלה לעץ עבור נקודה נתונה, לפי התהליך שלמדנו בהרצאה 3 | **private void LeftRotate(AVLNode DesiredRotatePoint)** |
| O(logn) | מוחקת צומת עם מפתח k מהעץ:   1. קוראת לפונקציה helpsearch(AVLNode node,int k) עם השורש והמפתח הנתון כדי למצוא את הצומת הנמחק. 2. אם הצומת הוא עלה חיצוני נסיק כי המפתח לא נמצא בעץ והפונקציה מחזירה -1 . 3. אחרת :  * קוראת לפונקציה NewDeleteMaxMin(AVLNode node) עם הצומת הנמחק בכדי לעדכן את המקסימום והמינימום בעץ * קוראת ל-deleteThis(AVLNode) בכדי למחוק את הצומת ולקבל את נקודת המחיקה * קוראת לפונקציהNewDeleteSize(AVLNode point) עם נקודת המחיקה בכדי לעדכן את שדה ה size של כל האבות של הצומת הנמחק בעץ * קוראים לפונקציה rebalanceAfterDeletion(AVLNode) עם צומת המחיקה בכדי לאזן את העץ, ומחזירה את מספר פעולות האיזון שהתבצעו | **public int delete(int k)** |
| O(logn) | מעדכנת את הצומת המינמלי והמקסימלי בעץ לאחר פעולת מחיקה של צומת. | **private void NewDeleteMaxMin(AVLNode node)** |
| O(logn) | מוחקת צומת נתון מהעץ ומחזירה את נקודת המחיקה לפי הסוג של הצומת:  1. אם הצומת הוא עלה:  בודקת אם הצומת הנמחק הוא השורש של העץ   1. אם כן – מעדכנת את השורש, המינ' והמקס' להיות צומת EXTERNAL\_NODE. 2. אחרת בודקים את סוג הבן ע״י הפונקציה IsLeftSon() :  * אם הצומת הוא בן שמאלי לאבא שלו מחליפים אותו בעלה חיצוני ע״י הפונקציה setLeft(AVLNode node) * אחרת, הצומת הוא בן ימיני לאבא שלו ולכן מחליפים אותו בעלה חיצוני ע״י הפונקציה setRight(AVLNode node)  1. אם הצומת הוא צומת אונארי ימיני:  * קוראים לפונקציה getRight() ושומרים את הבן הימיני * הופכים את האבא של הבן הימני להיות האבא של הצומת שבריצונינו למחוק * קוראים לפונקציה UnaryDeletion(AVLNode node AVLNode exactSon) כאשר הקלט הינו הצומת שברצונינו למחוק והבן הימיני שלו  1. אם הצומת הוא צומת אונארי שמאלי:  * קוראים לפונקציה getLeft() ושומרים את הבן השמאלי * הופכים את האבא של הבן השמאלי להיות האבא של הצומת שבריצונינו למחוק * קוראים לפונקציה UnaryDeletion(AVLNode node AVLNode exactSon) כאשר הקלט הינו הצומת שברצונינו למחוק והבן השמאלי שלו  1. אם הצומת הוא צומת פנימי:  * מוחקים את הצומת הרצוי לפי האלגוריתם שנלמד בהרצאה לגבי מחיקת צומת פנימי, כולל שימוש בפונקציה findSuccessor(AVLNode pnode)  1. אחרת, מחזירים מופע חדש של שגיאת זמן ריצה | **private AVLNode deleteThis(AVLNode node)** |
| O(1) | מוחקת צומת אונארי נתון מהעץ ומחזירה את נקודת המחיקה:   1. בודקת האם הצומת הנמחק הוא שורש. 2. אם כן – מאתחלת את השורש להיות בנו, אשר ניתן כקלט לפונקציה(exactSon ). 3. אחרת – קוראים לפונקציה IsLeftSon(), ובודקים את הצד של הצומת הנמחק. 4. אם הצומת הוא בן שמאלי לאביו, מחברים את האבא של הבן הנמחק עם הבן של הצומת הנמחק ע״י קריאה ל-setLeft( AVLNode) . 5. אם הצומת הוא בן ימיני לאביו, מחברים את האבא של הבן הנמחק עם הבן של הצומת הנמחק ע״י קריאה ל-setRight( AVLNode) .   מחזירה את אביו של הצומת הנמחק | **private AVLNode UnaryDeletion(AVLNode node, AVLNode exactSon)** |
| O(logn) | מחזירה את הצומת הבא אחרי צומת נתון בעץ בעזרת האלגוריתם שנלמד בהרצאה 3 | **private AVLNode findSuccessor(AVLNode node) {** |
| O(logn) | מעדכנת את גדלי הצמתים בעץ אחרי פעולת מחיקה של צומת נתון, ע"י עלייה בעץ מהצומת עד השורש ומקטינה את הגודל שלהם ב 1. | **private void NewDeleteSize(AVLNode point)** |
| O(logn) | מקבלת כקלט את הצומת של האבא של הבן הנמחק ומאזנת את העץ לאחר מחיקה לפי חלוקה למקרים שלמדנו בשיעור 4:   1. הצומת הינו עלה חיצוני 2. קשתות הצומת מסוג 2-2 אז מבצעים רקורסיה בעלות O(logn) 3. קשתות הצומת מסוג 3-1, בתהליך זה קוראים לפונקציה rebalanceAfterDeletionType31(AVLNode node) 4. קשתות הצומת מסוג 1-3, בתהליך זה קוראים לפונקציה rebalanceAfterDeletionType13(AVLNode node) | **private int rebalanceAfterDeletion (AVLNode point)** |
| O(logn) | מאזנת מצב אחרי מחיקה שבו קשתות הצומת הן מסוג 3-1 לפי התהליך שלמדנו בהרצאה 4:   1. בודקת מה הפרשי הדרגות של בנו הימני של הצומת הנתון 2. מחלקת למקרים לפי מה שלמדנו בהרצאה 4   מחזירה את מספר פעולות האיזון שהתבצעו | **private int rebalanceAfterDeletionType31(AVLNode point)** |
| O(logn) | מאזנת מצב אחרי מחיקה שבו קשתות הצומת הנתון הן 1-3 לפי התהליך שלמדנו בהרצאה 4:   1. בודקת מה הפרשי הדרגות של בנו הימני של הצומת הנתון 2. מחלקת למקרים לפי מה שלמדנו בהרצאה 4   מחזירה את מספר פעולות האיזון שהתבצעו | **private int rebalanceAfterDeletionType13(AVLNode point)** |
| O(1) | מחזירה את הvalue של הצומת ששמורה בשדה min | **public String min()** |
| O(1) | מחזירה את הvalue של הצומת ששמורה בשדה max | **public String max()** |
| O(n) | 1. קוראת לפונקציית עזר helpkeysToArray(), 2. ממירה את הפלט של פונקציית העזר מסטרינג לרשימה של מספרים | **public int[] keysToArray()** |
| O(n) | פונקציה רקורסיבית שמחזירה סטרינג של כל בkeys בעץ ,עוברת על כל הצמתים בעץ לפי שיטה שלמדנו inorder ובכל צומת היא מכניסה לסטרינג את הkey של הצומת:   1. קוראת לעצמה על הבן השמאלי 2. מכניסה לסטרינג את הkey של הצומת הנוכחית   וקוראת לעצמה על הבן הימני | **private String helpKeysToArray(AVLNode node, String str)** |
| O(n) | 1. קוראת לפונקציית עזר helpinfoToArray(), 2. ממירה את הפלט של פונקציית העזר מסטרינג לרשימה של סטרינגים | **public String[] infoToArray()** |
| O(n) | פונקציה רקורסיבית שמחזירה סטרינג של כל בvalue בעץ ,עוברת על כל הצמתים בעץ לפי שיטה שלמדנו inorder ובכל צומת היא מכניסה לסטרינג את הvalue של הצומת:   1. קוראת לעצמה על הבן השמאלי 2. מכניסה לסטרינג את הvalue של הצומת הנוכחית   וקוראת לעצמה על הבן הימני | **private String[] helpInfoToArray()** |
| O(1) | מחזירת את הsize של הצומת ששמורה בשדה Root | **public int size()** |
| O(1) | מחזירה את השורש של העץ, ואם הוא ריק אז היא מחזירה null | **public IAVLNode getRoot()** |
| O(logn) | מחפשת את הרשומה הרצויה ומפצלת את העץ לשני עצים, עץ ימני שמכיל כל האיברים שגדולים מ x ועץ שמאלי שמכיל את כל האיברים שקטנים מ x ומעדכנים העץ בעזרת join | **public AVLTree[] split(int x)** |
| O(logn) | עוברת מהשורש בכיוון הנכון עד שתמצא את הצומת בעל המפתח k ותחזיר מצביע עליו או עד שתגיע לעלה ותחזיר null.  מחזירה מצביע של הצומת בעל המפתח k אם נמצא, אחרת מחזירה null. | **private AVLNode find(int k)** |
| O(logn) | הפונקציה תאחד את x,t לעץ הנוכחי ותחזיר את העלות של פעולת ה-join (הפרש גבהיי העצים+1).  נתחיל בשורש של t , נרד במסלול הכי שמאלי שלו עד שנגיע לצומת הראשון b שמקיים:rank(b)<=k כך ש-k הוא הדרגה של העץ הנוכחי ,נהפוך את x להיות הבן השמאלי של c,b יהיה הבן הימני של x,ו-a(השורש של העץ הנוכחי) יהיה הבן השמאלי שלו.  במקרה שבו ה rank של x וגם של c הוא k+1 תהיה לנו קשת אחת שהפרש ה-rank בה הוא 0.תהליך האיזון מ-x יהיה בדיוק אותו תהליך שהיה לנו אחרי הכנסה לעץ AVL | **public int join(IAVLNode x, AVLTree t** |
| O(logn) | עוברת על הצמתים, מ- node עד השורש ומוסיפה 1 לשדה rank לצמתים במסלול זה אם יש צורך | **private void newHeight( AVLNode x )** |
| O(logn) | עוברת על הצמתים, מהאב של node עד השורש ומוסיפה 1 לשדה size של הצמתים במסלול זה | **private void plus(AVLNode node)** |

**שדות של AVLNode :**

**integer key**

מפתח צומת

**string value**

ערך צומת

**int size**

גודל של תת העץ של צומת, כולל הצומת

**int rank**

דרגה של צומת

**AVLNode parent**

הורה של צומת

**AVLNode left**

בן שמאלי של צומת

**AVLNode right**

בן ימני של צומת

**פונקציות של AVLNode :**

|  |  |  |
| --- | --- | --- |
| סיבוכיות | מה עושה | פונקציה |
| O(1) | בנאי ראשון | **private AVLNode()** |
| O(1) | בנאי שני | **private AVLNode(AVLNode node)** |
| O(1) | בנאי שלישי | **private AVLNode(int key, String value, AVLNode parent)** |
| O(1) | בנאי רביעי | **private AVLNode(AVLNode right, AVLNode left,int key, String value)** |
| O(1) | מחזיר את הערך בשדה key של הצומת | **public int getKey()** |
| O(1) | מחזיר את הערך בשדה value של הצומת | **public String getValue()** |
| O(1) | מגדיר את הצומת בשדה left לצומת המבוקש | **public void setLeft(IAVLNode)** |
| O(1) | מחזיר את הצומת ששמור בשדה left | **public IAVLNode getLeft()** |
| O(1) | מגדיר את הצומת בשדה right לצומת המבוקש | **private void setRight(IAVLNode)** |
| O(1) | מחזיר את הצומת ששמור בשדה right | **public IAVLNode getRight()** |
| O(1) | מגדיר את הצומת בשדה parent לצומת המבוקש | **private void setParent(IAVLNode)** |
| O(1) | מחזיר את הצומת ששמור בשדה parent | **public IAVLNode getParent()** |
| O(1) | בודק האם הצומת שווה לEXTERNAL\_NODE ואם כן מחזיר false אחרת true | **public boolean isRealNode()** |
| O(1) | מגדיר את הערך שנמצא בשדה rank לערך הרצוי | **public void setHeight(int height)** |
| O(1) | מחזיר את הערך ששמור בשדה rank | **public int getHeight()** |
| O(1) | מחזירה את הייצוג של Edge של שתי קשתות הצומת, בכך שנחשב את הפרשי הדרגות של אותה צומת עם בניה | **private *Edge* TypesOfEdges()** |
| O(1) | בודק האם אני הבן השמאלי של אבא שלי אם כן מחזיר true אם לא מחזיר false | **private boolean IsLeftSon()** |
| O(1) | מגדיל את הrank ב 1 | **private void promote()** |
| O(1) | מקטין את הrank ב 1 | **private void demote()** |
| O(1) | בודקת את המיקום של צומת יחסית לאביו, ומעדכנת את הבן של האב | **private void changeParent(AVLNode node)** |

מדידות

**מדידה ראשונה:**

*טבלת תוצאות:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **מספר סידורי** | **גודל המערך** | **כמות החילופים במיון רגיל עבור מערך אקראי** | **כמות החילופים במיון רגיל עבור מערך ממוין הפוך** | **עלות החיפושים במיון AVL עבור מערך אקראי** | **עלות החיפושים במיון AVL עבור מערך ממוין הפוך** |
| **1** | 10,000 | 25,033,910 | 49,995,000 | 214,933 | 231,332 |
| **2** | 20,000 | 99,340,297 | 199,990,000 | 485,831 | 502,660 |
| **3** | 30,000 | 223,364,634 | 449,985,000 | 777,357 | 788,084 |
| **4** | 40,000 | 398,455,981 | 799,980,000 | 1,014,199 | 1,085,316 |
| **5** | 50,000 | 625,715,342 | 1,249,975,000 | 1,372,404 | 1,386,164 |
| **6** | 60,000 | 897,384,888 | 1,799,970,000 | 1,639,023 | 1,696,164 |
| **7** | 70,000 | 1,217,034,692 | 2,449,965,000 | 1,867,110 | 2,010,628 |
| **8** | 80,000 | 1,593,593,672 | 3,199,960,000 | 2,288,130 | 2,330,628 |
| **9** | 90,000 | 2,021,709,330 | 4,049,955,000 | 2,594,781 | 2,650,628 |
| **10** | 100,000 | 2,504,091,160 | 4,999,950,000 | 2,908,587 | 2,972,324 |

**מדידה שנייה:**

*טבלת תוצאות:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| מספר סידורי | גודל המערך | עלות join ממוצע עבור split אקראי | עלות join מקסימלי עבור split אקראי | עלות join ממוצע עבור split של איבר מקס בתת העץ השמאלי | עלות join מקסימלי עבור split של איבר מקס בתת העץ השמאלי |
| 1 | 10,000 | 2.08 | 5 | 2.23 | 15 |
| 2 | 20,000 | 1.79 | 3 | 2.19 | 23 |
| 3 | 30,000 | 2.31 | 4 | 2.2 | 15 |
| 4 | 40,000 | 2.1 | 6 | 2.27 | 17 |
| 5 | 50,000 | 2.57 | 5 | 2.15 | 18 |
| 6 | 60,000 | 2.15 | 5 | 2.28 | 18 |
| 7 | 70,000 | 2.39 | 6 | 2.34 | 19 |
| 8 | 80,000 | 2.58 | 8 | 2.21 | 19 |
| 9 | 90,000 | 2.48 | 6 | 2.09 | 18 |
| 10 | 100,000 | 2.18 | 4 | 2.23 | 19 |