**פרויקט 1**

**Biesan assiביסאן עאסי 208145268**

**yazanzoabiיזן זועבי 209105790**

**מבני נתונים**

**תיעוד חיצוני:**

**IAVLNode**

ממשק של צומת בעץ AVL.

**הפעולות:**

**public** **void** setKey(**int** key);

מעדכנת ערך המפתח בצומת להיות key**.**

**public** **int** getKey();

מחזירה המפתח של הצומת

**public** **void** setValue(String value);

קביעת ערך של המחרוזת בצומת להיות value.

**public** String getValue();

מחזירה הinfo של הצומת.

**public** **void** setChild(IAVLNode left, IAVLNode right);

קובעת הבנים לצומת כך שleft נקבע כבן שמלי ו right יהיה הבן הימיני לצומת.

**public** IAVLNode getChild();

החזרת הבן היחיד של הצומת אם יש לו בן יחיד, ואם לא אז החזרת אחד מהבנים הווירטואליים.

**public** **boolean** hasTwoChildren();

מחזירהtrue אמ"מ יש לצומת שני בנים אינם ווירטואלים.

**public** **void** setLeft(IAVLNode node);

מעדכן הבן השמאלי לצומת להיות node.

**public** IAVLNode getLeft();

החזרת הבן השמאלי של הצומת.

**public** **void** setRight(IAVLNode node);

מעדכן הבן הימיני לצומת להיות node.

**public** IAVLNode getRight();

החזרת הבן הימני של הצומת

**public** **void** setParent(IAVLNode node);

מעדכנת האב של הצומת להיות node.

**public** IAVLNode getParent();

החזרת האב של הצומת.

**public** **void** setHeight(**int** height);

קביעת ערכו של הגובה להיות height.

**public** **int** getHeight();

מחזירה הגובה של הצומת.

**public** **void** setSize(**int** size);

מעדכנת הsize בצומת "כלומר מספר הבנים שיש לץ הזה".

**public** **int** getSize();

מחזירה מספר הבנים לצומת.

**public** **boolean** isRealNode();

מחזירה true אמ"מ הצומת הוא צומת אמתי.

**public** **void** setKeysSum(**int** keysSum);

קביעת סכום המתפתחות בתת העץ המתחיל בצומת להיות keysSum

**public** **int** getKeysSum();

החזרת סכום המפתחות בתת העץ המתחיל בצומת

**public** **int** calculateBF();

חישוב ה- balance factor של הצומת

**public** **int** calculateHeight();

חישוב הגובה של הצומת

**public** **int** calculateKeysSum();

חישוב סכום המפתחות בתת העץ המתחיל בצומת

**public** **int** calculateSize();

חישוב מס' הצמתים בתת העץ המתחיל בצומת )כולל הצומת(

**public** **boolean** equals (IAVLNode x);

מחזירה true אמ"מ לצומת x יש אותו ערך של מפתח.

**public** **boolean** isLeaf();

מחזירה true אמ"מ הצומת הוא עלה.

**AVLNode**

מחלקה זו מממשת את הממשק . IAVLNode מחלקה המייצגת אובייקט של צומת אמיתי בעץ "אינו וירטואליים"

**השדות:**

**private int** key;

המפתח של הצומת

**private** String info

המחרוזת השמורה בצומת

**private** IAVLNode left;

הבן השמאלי של הצומת מוגדר כצומת "וירטואליים" עד העדכון הראשון

**private** IAVLNode right;

הבן הימני של הצומת מוגדר כצומת "וירטואליים" עד העדכון הראשון.

**private** IAVLNode parent;

ההורה של הצומת מוגדר כ NULL עד העדכון הראשון.

**private int** size;

מספר הצמתים בעץ שהצומת הוא השורש שלו כולל הצומת הנוכחית.

**private int** height;

הגובה של הצומת

O(1) - **הפעולות:** כל הפעולות במחלקה זו מתבצעות ב

הסביר על מתודות שאינם getter ו setter :

**public** AVLNode(String info, **int** key)

הבנאי למחלקת הצומת המפתח מאותחל להיות הערך של key שקיבלה המתודה

והערך info מאותחל גם להיות הערך הinfo שקיבלה המתודה והSize מאותחל להיות 1.

**public** **boolean** isRealNode()

מתודה זו בודקת אם אנחנו באותו צומת, מחזירה תמיד true

כי המחלקה זאת רק מיצירת צומת אמיתית.

**public int** calculateBF()

של הצומת לפי הנוסח:: balance factor - מתודה זו מחשבת את ה

*bf* = *left*.*height* − *r*.*height*

**public int** calculateHeight()

מתודה זו מחשבת את הגובה של הצומת בהתאם הגבהים של שני בניו לפי הנוסח:

*height* = max(*left*.*height*, *right*.*height*) +1

**public int** calculateSize()

חישוב מספר הצמתים בעץ שהצומת הוא שורשו וזאת לפי הנוסח:

*size* = *left*.*size* + *right*.*size* +1

**public** **boolean** equals(IAVLNode x)

מחזירה True אם לשתי צמתים יש אותו ערך של מפתח.

**public** IAVLNode getChild()

מתודה זו מניחה שלצומת יש בן אמתי אחד או שני בנים ווירטואליים.

אם יש בן אמתי אז הבן האמתי מוחזר.

אחרת,מוחזר הבן הווירטואלי.

**public boolean** hasTwoChildren()

מתודה זו בודקת אם לצומת יש שני צמתים אמתיים.

המתודה מחזירה TRUE אמ"מ הבנים לצומת אמתים.

**public boolean** isLeaf()

.false אחרת, מוחזר .true אם הצומת הוא עלה כלומר יש לו שני בנים ווירטואליים מוחזר

**notRealAVLNode**

מחלקה זו מממשת את הממשק . IAVLNode מחלקה המייצגת אובייקט של צומת "וירטואליים"

**השדות:**

**private** IAVLNode parent;

ההורה של הצומת.

.O(1) - **הפעולות:** כל הפעולות במחלקה זו מתבצעות ב

הסביר על מתודות שאינם setter במחלקה זאת "כי המתודות האלה לא מבצעות כלום.

**public** notRealAVLNode(IAVLNode parent)

הבנאי למחלקה מגדיר ההורה לצומת הווירטואיל.

**public** **int** getKey()

מחזיר תמיד 0.

**public** String getValue()

מחזירה מחרוזת של [NOTREAL].

**public** IAVLNode getLeft()

מחזירה NULL.

**public** IAVLNode getRight()

מחזירה NULL.

**public** IAVLNode getParent()

מחזירה צומת ההורה שהוגדרנו אותה בבנאי.

**public** **boolean** isRealNode()

בודקת אם הצומת אמיתית

מכוון שזו מחלקה עבור צומת ווירטואלית תמיד מחזירה false.

**public** **int** getHeight()

מחזירה -1 כי כך הגדרנו הגובה לצומת ווירטואלית

**public** **int** getSize()

מחזירה 0.

**public** **int** calculateBF()

מחזירה 0.

**public** **int** calculateHeight()

מחזירה -1.

**public** **int** calculateSize()

מחזירה 0.

**public** **boolean** equals(IAVLNode x)

מחזירה תמיד false.

**public** **boolean** isLeaf()

מחזירה תמיד false.

**public** **boolean** hasTwoChildren()

מחזירה תמיד false.

**public** IAVLNode getChild()

מחזירה NULL.

**AVLTree**

כפי שנלמד בקורס. AVL המחלקה מייצגת עץ

**השדות:**.

**private** IAVLNode root;

מצביע לשורש מוגד כצומת ווירטואלית עד העדכון הראשון.

**Private** IAVLNode min;

מצביע לצומת עם המפתח המינימלי.

**private** IAVLNode max;

מצביע לצומת עם המפתח המקסימלי.

**הפעולות:**

**public boolean** empty()

בודקת אם העץ ריק, מחזירה TRUE אם השורש הוא צומת אמתי.

O(1) : סיבוכיות זמן

**public** String search(**int** k)

המתודה מחפשת איבר בעל המפתח k. אם קיים איבר כזה, היא מחזירה את הערך השמור עבורו,

אחרת היא מחזירה null. מכוון שהעץ אצלנו עץ בינארי מאוזן אזי נקבל

O(log *n*) : סיבוכיות זמן

**private** IAVLNode FindeNodeOfKey(**int** x)

המתודה מחפשת צומת בעל המפתח k. אם קיים צומת כזה, היא מחזירה אותו

אחרת היא מחזירה null. מכוון שהעץ הוא עץ בינארי מאוזן אזי נקבל

O(log *n*) : סיבוכיות זמן

להבדיל מ search שמחזירה הערך של הצומת.

**private** IAVLNode FindNodeOfRank(AVLTree t, **int** k, OP op,**int** size)

**<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<**

**<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<**

**<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<**

**enum** OP

משתמשים באינם זה בשביל להשתמש במתודות מסוימות בשיטות אחרות.

INSERT ו DELETE משמשים אותנו במתודת Balance אשר מבדילים בה בין שני אלגוריתמים של איזון עץ אחרי הכנסה או הוצאה.

RIGHT ו LEFT משמשים אותנו ב FinedNodeOfRank בשל להבדיל בין צד ימין ושמאל בחיפוש.

SPLIT ו JOIN משמשים אותנו במתודת join בשביל לדעת אם צריך לעדכן min ו maxאחרי ביצוע JOIN.

**public int** insert(**int** k, String i)

מתודה זו מכניסה צומת חדש שהמפתח שלו הוא k והמחרוזת שלו היא i לעץ (אם המפתח k לא היה קיים בעץ). אם המפתח k היה קיים בעץ מוחזר -1 , אחרת מוחזר מספר הגלגולים שהתבצעו.

אם העץ ריק מוסיפים את הצומת החדש כשורש. אם לא אז מחפשים את המקום אליו הצומת החדש יוכנס (עוצרים כשמגיעים לצומת שהמפתח שלו הוא k או כשמגיעים לצומת ווירטואלי). אם החיפוש מסתיים בצומת שהמפתח שלו הוא k מחזירים -1 . אם החיפוש מסתיים בצומת ווירטואלי אז מוצאים את ההורה של הצומת הווירטואלי ומחברים את הצומת החדש כבן ימני או שמאלי להורה. אחר כך מבצעים איזון של העץ, מעדכנים את המינימום והמקסימום ומחזירים את מספר הגלגולים שהתבצעו במהלך פעולת איזון העץ.

O(log *n*) : סיבוכיות זמן

**private int** balance (IAVLNode node, OP operation)

מתודה זו מאזנת את העץ אחרי הכנסה/הסרה של צומת מהעץ ומחזירה את מספר הגלגולים שהתבצעו. אם operation הוא INSERT אז עוצרים אחרי הגלגול הראשון. אם operation הוא delete אז ממשיכים עד השורש. המתודה מממשת האלגוריתם שראינו בהרצאה לאיזון העץ אחרי הכנסה/הוצאה מעץAVL.

מתודה זו קוראת למתודה rotate שמבצעצ את הגלגולים.

O(log *n*) : סיבוכיות זמן

**private int** rotate(IAVLNode node)

מתודה זו קוראת לגלגול המתאים על הצומת node ומחזירים את מספר הגלגולים שהתבצעו:

1 .אם ה- balance factorשל הצומת הוא 2:

▪ אם ה- balance factor של הבן השמאלי הוא 1 או 0 מבצעים LL ומחזירים 1

▪ אם ה- balance factor של הבן השמאלי הוא 1 -מבצעים LR ומחזירים 2

2 .אם ה- balance factor של הצומת הוא 2:-

▪ אם ה- balance factor של הבן הימני הוא 1 -או 0 מבצעים RR ומחזירים 1

▪ אם ה- balance factor של הבן הימני הוא 1 מבצעים RL ומחזירים 2.

O(1) : סיבוכיות זמן

**private void** LL(IAVLNode node)

node מתודה זו מבצעת גלגול ימני על הצומת

O(1) : סיבוכיות זמן

**private void** RR(IAVLNode node)

node מתודה זו מבצעת גלגול שמאלי על הצומת

O(1) : סיבוכיות זמן

**private void** LR(IAVLNode node)

node ואז גלגול ימני על node מתודה זו מבצעת גלגול שמאלי על הבן השמאלי של

O(1( : סיבוכיות זמן

**private void** RL(IAVLNode node)

node ואז גלגול שמאלי על node מתודה זו מבצעת גלגול ימני על הבן הימני של

O(1) : סיבוכיות זמן

**private void** rightRotate(IAVLNode a)

a מתודה זו מבצעת גלגול ימני על הצומת

O(1) : סיבוכיות זמן

private void leftRotate(IAVLNode a)

a מתודה זו מבצעת גלגול שמאלי על הצומת

O(1) : סיבוכיות זמן

private boolean isRight(IAVLNode node)

הוא בן ימני של ההורה שלו node אמ"מ true מתודה זו מחזירה

O(1) : סיבוכיות זמן

private void updateHeightSize(IAVLNode node)

.node מתודה זו מעדכת את הגובה ואת מספר הצמתים בצומת

O(1) : סיבוכיות זמן

**private void** updateSizesButtomUp(IAVLNode node)

ועד השורש )בהתאם לערכים של השדות node - בכל הצמתים החל מ size מתודה זו מעדכנת את השדות

האלה אצל הבנים.

O (log n) : סיבוכיות זמן

public int delete(int k)

מתודה זו מוחקת את הצומת עם המפתח k מהעץ ומחזירה את מספר הגלגולים שהתבצעו במהלך המחיקה. אם הצומת לא נמצא בעץ מוחזר 1.- אם העץ ריק מחזירים 1 ,-אחרת מחפשים בעץ את הצומת עם המפתח k .אם החיפוש מסתיים בצומת ווירטואלי סימן שהמפתח לא נמצא בעץ ולכן נחזיר 1 .-אם לא הגענו לצומת ווירטואלי אז מצאנו את הצומת. אם הצומת הוא השורש והוא גם עלה אז מאפסים את העץ )קובעים את root להיות null .)אם לצומת יש שני בנים אמתיים אז נחליף את הצומת עם הצומת העוקב לו ונסמן את העוקב כצומד שעתיד להימחק מהעץ. בשלב זה אנחנו יודעים שלצומת שאנחנו רוצים למחוק יש ילד אחד או שהוא עלה. אם הצומת הוא השורש אז נקבע את השורש להיות הבן של הצומת ונחזיר 0 כי אין צורך בגלגול. נמחק את הצומת מהעץ )מחיקה של עלה או צומת שיש לו בן אחד( ונעדכן up-buttom את מספר הצמתים וסכום המפתחות בכל הצמתים עד השורש. בסוף נאזן את העץ, נעדכן את המקסימום ואת המינימום ונחזיר את מס' הגלגולים שהתבצעו.

O(log *n*) : סיבוכיות זמן

**private void updateMin()**

,null להיות min להיות הצומת בעל המפתח המינימלי. אם העץ ריק מעדכנים את min מתודה זו מעדכנת את השדה

אחרת הולכים שמאלים עד שמגיעים לצומת ווירטואלי ומחזירים את ההורה שלו.

O(log n) : סיבוכיות זמן

**private void updateMax()**

מתודה זו מעדכנת את השדה max להיות הצומת בעל המפתח המקסימלי. אם העץ ריק מעדכנים את max להיות null

אחרת הולכים ימינה עד שמגיעים לצומת ווירטואלי ומחזירים את ההורה שלו

O(log n) : סיבוכיות זמן

**private void swapNodes(IAVLNode y, IAVLNode x)**

.x - ו y בין הצמתים info - ו key מתודה זו מחליפה את השדות

O(1) : סיבוכיות זמן

**public String min()**

מחזירה את ערכו (info) של האיבר בעץ בעל המפתח המינימלי, או null אם העץ ריק.

O(1) : סיבוכיות זמן

**public String max()**

מחזירה את ערכו (info) של האיבר בעץ בעל המפתח המקסימלי, או null אם העץ ריק.

O(1) : סיבוכיות זמן

**private** IAVLNode Successor(IAVLNode x)

המתודה מחזירה את הצומת בעלת המפתח הבא למפתח של הצומת X בעץ.

אםx יש בן ימני אז מחפשים את הבן הכי שמאלי בתת עץ הימני של x

אם אין לו בן ימני אז עולים לאבא הראשון ש x הוא בתת עץ השמאלי שלו ואז מחפשים בתת עץ הימני שלו את הצומת הכי שמאלית. אם אין אבא כזה מחזירים null.

**public int[] keysToArray()**

המתודה מחזירה מערך ממוין המכיל את כל המפתחות בעץ, או מערך ריק אם העץ ריק.

המתודה קוראת למתודה successor n פעמים מהערך הכי קטן ולכן נקבל לפי מה שראינו בתרגיל שסיבכיות הזמן

O(n) : סיבוכיות זמן

**public String[] infoToArray()**

המתודה מחזירה מערך מחרוזות המכיל את כל המחרוזות בעץ, ממוינות על פי סדר המפתחות. כלומר הערך ה j במערך הוא המחרוזת המתאימה למפתח שיופיע במיקום ה j במערך הפלט של הפונקציה keysToArray(). גם הפונקציה הזאת מחזירה מערך ריק אם העץ ריק.

המתודה קוראת למתודה successor n פעמים מהערך הכי קטן ולכן נקבל לפי מה שראינו בתרגיל שסיבכיות הזמן

O(n) : סיבוכיות זמן

**public int size()**

המתודה מחזירה את מספר האיברים בעץ כלומר מבצעת getSize לroot ולכן נקבלש.

O(1) : סיבוכיות זמן

**public IAVLNode getRoot()**

מחזיר את השורש של העץ root.

O(1) : סיבוכיות זמן

**public** AVLTree[] split(**int** x)

המתודה מקבלת מפתח x שנמצא בעץ. המתודה מפרידה את העץ ל-2 עצי AVL ומחזירה מערך שמכיל שניהם כאשר המפתחות של האחד גדולים מ-x ושל השני קטנים מ x.

המתודה משתמשת במתודת join עם enum SPLIT בשביל לאחד את העצים הקטנים\גדולים מ x.

המתודה פועלת לפי האלגוריתם שהוסבר בשיעור.

**public** **int** join(IAVLNode x, AVLTree t)

קוראת למתודה join עם enum.JOIN בשביל להבדיל מקריאת מSPLIT.

**private** **int** join(IAVLNode x, AVLTree t,OP op)

המתודה מקבלת צומת x ועץ t שכל הkeys- שלהם קטנים (או גדולים) מהkeys- של העץ הנוכחי. המתודה מאחדת את x,t לעץ הנוכחי כפי שמומש בהרצאה. המתודה מחזירה את העלות של פעולת ה-join (הפרש גבהיי העצים+1).

**מדידות**

**חלק ראשון:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | 10,000 | 0.6924 | 0.3955 | 2 | 6 |
| 2 | 20,000 | 0.6958 | 0.3937 | 2 | 7 |
| 3 | 30,000 | 0.6992 | 0.3977 | 2 | 9 |
| 4 | 40,000 | 0.6958 | 0.4049 | 2 | 7 |
| 5 | 50,000 | 0.6999 | 0.4010 | 2 | 8 |
| 6 | 60,000 | 0.6971 | 0.3994 | 2 | 7 |
| 7 | 70,000 | 0.6931 | 0.4008 | 2 | 7 |
| 8 | 80,000 | 0.6936 | 0.4016 | 2 | 7 |
| 9 | 90,000 | 0.6967 | 0.4000 | 2 | 7 |
| 10 | 100,000 | 0.702 | 0.3977 | 2 | 7 |

למדנו כי פעולת האיזון לאחר ההוספה יכולה לכלול גלגול אחד לכל היותר .(LL LR RL RR) לעומת זאת במחיקה אנחנו לא מפסיקים את התהליך לאחר הגלגול הראשון אלא ממשיכים למעלה אולי עד השורש. כלומר, היינו מצפים שמספר פעולות האיזון הממוצע לפעולת delete יהיה גדול יותר ממספר פעולות האיזון הממוצע לפעולת insert .אנחנו רואים שזה לא נכון יש יתרון קל לממוצע במקרה של הכנסה אולם המספרים שהתקבלו הם די קרובים למספרים שקיבלנו עבור ההוספה.

מאותה סיבה הינו מצפים שמספר פעולות האיזון המקסימלי אחרי הוצאה יהיה יותר גדול מאשר אחרי הכנסה ובמקרה הזה זה דפקא נכון אבל גם פה זה שונה בכפל בקבוע וזה לא גדל לאחר הגדלת העץ.

הייתי מצפה גם שככל שגודל העץ גדל יגדל ממוצע פעולות האיזון אבל זה לא מה שקרה.

**חלק שני:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| מספר סידורי | מספר פעולות | 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 | 1.8571 | 5 | 2.25 | 15 |
| 2 | 20,000 | 2.5 | 5 | 2.1538 | 6 |
| 3 | 30,000 | 2.6363 | 6 | 2.2142 | 17 |
| 4 | 40,000 | 1.5625 | 4 | 2.1333 | 17 |
| 5 | 50,000 | 2.5 | 8 | 2.0625 | 17 |
| 6 | 60,000 | 2.1333 | 4 | 2.1875 | 18 |
| 7 | 70,000 | 2 | 5 | 2.1333 | 18 |
| 8 | 80,000 | 2.5333 | 4 | 2.125 | 18 |
| 9 | 90,000 | 2.3333 | 4 | 2.1875 | 19 |
| 10 | 100,000 | 2.6 | 7 | 2.2666 | 19 |