שמות המגישים: עידן יוסף, תמיר דוידי

תעודות זהות

עידן - 207522285

תמיר - 315852608

שמות משתמשים

עידן - idanyosef

תמיר - tamirdavidi1

ניתוח סיבוכיות – עץ דרגות

פעולות_AVLTree

public AVLTree()

O(1)

public boolean empty()

O(1)

public Boolean search(int k)

O(log n)

המתודה מחפשת צומת בעץ לפי מפתח. כל עוד לא הגענו לצומת וירטואלי אז אנחנו בודקים את הערך של הצומת שהגענו אליו ובהתאם למפתח של צומת זה או שאנחנו מחזירים את הערך השמור עבור צומת זה או שאנחנו ממשיכים לבן הימני או לבן השמאלי.אם הגענו לצומת וירטואלי אז מוחזר null.

הסיבוכיות O(log n) היא כי בכל איטרציה אנחנו ממשיכים לבן הימני או השמאלי ובמקרה הגרוע הגענו עד לעלה של העץ ומכיוון שגובה העץ חסום ע"י O(log n) אז הסיבוכיות היא O(log n).

public boolean is_avarian(AVLNode node)

O(1)

הפעולה בודקת אם הצומת הוא עבריין AVL

public int calc_BF(AVLNode node)

O(1)

הפעולה מחשבת את ערך ה-BF של צומת.

public int calc_height(AVLNode node)

O(1)

הפעולה מחשבת את הגובה של צומת

public int calc_size(AVLNode node)

O(1)

הפעולה מחשבת את ערך ה-size של צומת

public int calc_cnttrue(AVLNode node)

O(1)

הפעולה מחשבת את מספר ה-true בתת העץ השמאלי של הצומת כולל הצומת

public int calc_rightcnttrue(AVLNode node)

O(1)

הפעולה מחשבת את מספר ה-true בתת העץ הימני של הצומת כולל הצומת

public int calc_subcnttrue(AVLNode node)

O(1)

הפעולה מחשבת את מספר ה-true בתת העץ של הצומת כולל הצומת

public void size_bf_height(AVLNode node)

O(1)

הפעולה מעדכנת את השדות שמצריכים תחזוקה

public boolean changed_bf_from_zero(AVLNode node, int before_bf, int after_bf)

O(1)

הפעולה בודקת אם צומת שינה את ערך ה-BF שלו מ-0 לערך שונה מ-0

public int insert(int k, boolean i)

O(log n)

הכנסנו תחילה את הצומת החדש בתחתית העץ במקום המתאים בעלות לוגריתמית, לאחר מכן עלינו חזרה מהצומת אל השורש ותיקנו עבריין AVL אם קיים. תיקונים מתבצעים בזמן קבוע והעלייה והירידה מהשורש לעלה ובחזרה נעשית בזמן לוגריתמי סה"כ.

public int delete(int k)

O(log n)

בהתחלה אנחנו מחפשים את האיבר בעל המפתח k בעץ ואם המפתח לא נמצא אז הפונקציה מחזירה 1-. אם המפתח נמצא בעץ אז מוחקים את המפתח. פעולת החיפוש של הצומת לוקחת O(log n). לאחר מחיקת המפתח מהצומת עולים מהצומת אל השורש ומתקנים עבריין AVL אם קיים ובנוסף מעדכנים את המשתנה שסופר את פעולות האיזון שנדרשו בסה"כ בשלב תיקון העץ אם היה גלגול כלשהו בצומת הנוכחי או שהיה עדכון שדה גובה הצומת ולאחר שעלינו עד השורש וביצענו את פעולות האיזון הנדרשות אנחנו מחזירים את המשתנה שסופר את פעולות האיזון שנדרשו בסה"כ בשלב תיקון העץ. התיקונים מתבצעים בזמן קבוע והעליה מהצומת שמחקנו עד לשורש נעשית בזמן לוגריתמי ולכן הסיבוכיות היא O(log n).

public Boolean min()

O(1)

בהינתן שתחזקנו מצביע לצומת המינימלי ניתן לבצע זאת בזמן קבוע.

public Boolean max()

O(1)

בהינתן שתחזקנו מצביע לצומת המקסימלי ניתן לבצע זאת בזמן קבוע.

public int keys_in_order(int[] arr, AVLNode root, int i)

O(n)

סה"כ עוברים על כל צמתי העץ, כל צומת פעם יחידה, לכן סה"כ מתבצע בזמן לינארי. פעולת העזר של הפעולה keysToArray.

public int[] keysToArray()

O(n)

החלק הקובע את זמן הריצה הוא הקריאה לפעולת העזר keys_in_order שמתבצעת בזמן לינארי.

public boolean[] infoToArray()

O(n)

המתודה מחזירה מערך בוליאניים של הערכים בעץ והוא ממויין לפי סדר המפתחות בעץ. בהתחלה המתודה מאתחלת מערך ריק שגודלו כמספר הצמתים בעץ ולאחר מכן היא מבצעת קריאה ל- info_in_order ולאחר הקריאה היא מחזירה את המערך הממויין.

הסיבוכיות היא O(n) כי הסיבוכיות של info_in_order היא O(n).

public int info_in_order(boolean[] arr, AVLNode root, int i)

זוהי מתודה רקורסיבית שמכניסה את ערכי הצמתים בעץ למערך לפי סדר המפתחות בעץ. אם הצומת נמצא בעץ אז היא מבצעת קריאה רקורסיבית לבן השמאלי של הצומת ולאחר מכן היא מכניסה את ערכו של הצומת למערך ומקדמת את ערכו של האינדקס i ב-1 ואז היא מבצעת קריאה רקורסיבית לבן הימני של הצומת. אם הצומת לא נמצא בעץ אז היא מחזירה את האינדקס i.

הסיבוכיות היא O(n) כי המתודה עוברת על כל צומת במערך פעם אחת ומכניסה את ערכו למערך.

public int size()

O(1)

לאורך ההכנסות ומחיקות לעץ תחזקנו בכל צומת את השדה size שמייצג את מספר הצמתים בתת העץ שהצומת הוא השורש שלו, לכן נוכל פשוט להחזיר את ערך ה-size של שורש העץ בזמן קבוע.

public AVLNode getRoot()

O(1)

המתודה ניגשת לשדה ה-root ומחזירה את השורש של העץ.

הסיבוכיות היא O(1) כי זו פעולת גישה לשדה ה-root.

public boolean prefixXor(int k)

O(log n)

המתודה מחזירה את ה- Xor של הערכים הבוליאניים הנמצאים במבנה תחת מפתחות שקטנים או שווים ל-k. המתודה מחפשת צומת בעץ לפי מפתח. ומעדכנת את המשתנה שסופר את מספר ה-true שקטנים או שווים ל-k לפי השדה שסופר את מספר ה-true בתת העץ השמאלי של הצומת כולל הצומת. ולאחר מכן עולים עד השורש של העץ ובמקרה ש-k גדול יותר מהצומת שהגענו אליו אז נעדכן את המשתנה לפי השדה שסופר את מספר ה-true בתת העץ השמאלי של הצומת כולל הצומת ובמקרה ש-k קטן יותר אז ממשיכים להורה של הצומת. לאחר שהגענו לשורש העץ אז אנחנו בודקים אם המשתנה שסופר את מספר ה-true שקטנים או שווים ל-k , אם הוא מתחלק ב-2 עם שארית אז נחזיר true ואחרת נחזיר false.

הסיבוכיות היא O(log n) כי אנחנו מחפשים את הצומת לפי המפתח k ופעולה זאת לוקחת במקרה הגרוע O(log n) ומעדכנים את המשתנה שסופר את מספר ה-true שקטנים או שווים ל-k לפי השדה שסופר את מספר ה-true בתת העץ השמאלי של הצומת כולל הצומת. ולאחר מכן עולים עד השורש ומעדכנים את המשתנה שסופר את מספר ה-true שקטנים או שווים ל-k בהתאם לערך המפתח בצומת ופעולה זו לוקחת במקרה הגרוע O(log n) ולכן בסה"כ הסיבוכיות היא O(log n).

אם הגענו לצומת וירטואלי אז מוחזר null.

public AVLNode successor(AVLNode node)

O(log n)

במקרה הגרוע הפונקציה יכולה לרדת את כל עומק העץ (או לעלות את כל גובה העץ) במהלך חיפוש העוקב, גובה העץ חסום ב-O(log n), מלבד המעבר בין הצמתים אין פעולה משמעותית נוספת לכן הסיבוכיות הכוללת היא לוגריתמית במספר הצמתים בעץ.

public boolean succPrefixXor(int k)

O(k)

ראינו בהרצאה שהפעלת k פעמים את פעולת העוקב מתבצעת ב-O(k). הסיבה לכך היא שלא כל פעם שמבצעים את פעולת העוקב, הפונקציה צריכה לעבור על כל גובה העץ ולבצע O(log n) פעולות. ובסה"כ אנו עוברים על k איברים עוקבים לכן הפעולה הכוללת מתבצעת בזמן לינארי ב-k.

פעולות AVLNode

בנאי ראשון לצומת בעץ

public AVLNode(int key, AVLNode parent, AVLNode left, AVLNode right, int height, boolean info, boolean isReal) {

O(1)

בנאי נוסף לצומת בעץ

public AVLNode(int key, boolean info, boolean isReal)

O(1)

מחזירה את המפתח של הצומת

public int getKey()

O(1)

מחזירה את הערך של הצומת

public boolean getValue()

O(1)

מגדירה את הבן השמאלי של הצומת

public void setLeft(AVLNode node)

O(1)

מחזירה את הבן השמאלי של הצומת

public AVLNode getLeft()

O(1)

מגדירה את הבן הימני של הצומת

public void setRight(AVLNode node)

O(1)

מחזירה את הבן הימני של הצומת

public AVLNode getRight()

O(1)

מגדירה את שדה ה-size של צומת

public void setSize(int size)

O(1)

מעלה את שדה ה-size ב-1

public void size_up()

O(1)

מורידה את שדה ה-size ב-1

public void size_down()

O(1)

מחזירה את שדה ה-size

public int getSize()

O(1)

מגדירה את שדה ה-Balance factor size

public void setBF(int BF)

O(1)

מחזירה את שדה ה-Balance factor size

public int getBF()

O(1)

מחזירה את שדה ה-Balance factor size

public int getBF()

O(1)

מחזירה את מספר ה-true בתת העץ השמאלי של הצומת כולל הצומת.

public int getCnttrue()

O(1)

מגדירה את מספר ה-true בתת העץ השמאלי כולל הצומת

public void setCnttrue(int cnttrue)

O(1)

מחזירה את מספר ה-true בתת העץ הימני של הצומת כולל הצומת

public int rightCnttrue()

O(1)

מגדירה את מספר ה-true בתת העץ הימני של הצומת כולל הצומת

public void setrightCnttrue(int rightcnttrue)

O(1)

מחזירה את מספר ה-true בתת העץ של הצומת כולל הצומת

public int getsubCnttrue()

O(1)

מגדירה את מספר ה-true בתת העץ הצומת כולל הצומת

public void setsubCnttrue(int subcnttrue)

O(1)

מגדירה את ההורה של הצומת

public void setParent(AVLNode node)

O(1)

מחזירה את ההורה של הצומת

public AVLNode getParent()

O(1)

מחזירה אמת אם הצומת אמיתי ולא וירטואלי

public boolean isRealNode()

O(1)

מגדירה את גובה הצומת

public void setHeight(int height)

O(1)

מחזירה את גובה הצומת

public int getHeight()

O(1)

מחזירה אמת אם הצומת עלה

public boolean isLeaf()

O(1)

מסייעת בהדפסת העץ

public String getText()

O(1)

מדידות

מספר סידורי	עלות prefixXor ממוצעת (כל הקריאות) nanosec	עלות succPrefixXor ממוצעת (כל הקריאות) nanosec	prefixXor עלות ממוצעת 100 קריאות ראשונות nanosec	עלות succPrefixXor ממוצעת 100 קריאות ראשונות nanosec
1	2744	21276	3570	24630
2	5154	24170	3206	19274
3	1958	36074	1113	29327
4	1113	43836	884	26915
5	1845	27666	1131	26798

מסקנות זמן הריצה

ניתן להסיק בבירור מן התוצאות כי זמן הריצה של prefixXor מהיר יותר מאשר זה של succPrefixXor. על אף ששתי הפעולות פועלות כביכול באותה סיבוכיות זמן אסימפטוטית לוגריתמית במספר הצמתים בעץ, ניתן להסיק מהתוצאות שהקבועים שנחבאים בפעולת succPrefixXor גבוהים בהרבה מאלו של prefixXor.

מדידות - חלק 2- נמדד ב nanoseconds

עלות הכנסה ממוצעת עבור מספר סידורי i	עץ AVL - סדרה חשבונית	עץ ללא מנגנון איזון - סדרה חשבונית	עץ AVL - סדרה מאוזנת	עץ ללא מנגנון איזון - סדרה מאוזנת	עץ AVL סדרה אקראית	עץ ללא מנגנון איזון - סדרה אקראית
1	35175	28998	2069	3112	39399	4005
2	24467	34218	5152	2280	20431	4467
3	17451	34404	3537	2892	20138	3653
4	12712	35463	5515	2536	16826	3953
5	12216	43263	4569	2553	14361	4750

מסקנות זמן הריצה

ציפינו לכך שבעת הכנסת **סדרה חשבונית**, זמן הריצה של הכנסת מפתח לעץ לא מאוזן בממוצע יהיה ארוך מזמן הכנסת המפתח לעץ AVL, מאחר שבעץ AVL קיים מנגנון איזון שחוסם מלמעלה את זמן הריצה אסימפטוטית עבור כל הכנסה בניגוד לעץ ללא מנגנון שכזה. תוצאותה המדידות עולות בקנה אחד עם השערתינו, אכן זמן הריצה של הכנסת מפתח ארוך יותר בממוצע בעץ לא מאוזן מאשר בעץ AVL.
באשר להכנסת **סדרה מאוזנת** ציפינו לכך שזמן הריצה של הכנסת מפתח לעץ לא מאוזן יהיה דווקא מהיר יותר מאשר זמן ההכנסה לעץ AVL. התוצאות עולות בקנה אחד עם המדידות, זמן הכנסת מפתח בממוצע לעץ לא מאוזן אכן קצר יותר בכמעט רוב המדידות שביצענו עבור סדרה מאוזנת. נשים לב כי סדרת המפתחת שהכנסנו כבר יוצרת עץ מאוזן ולכן אין כל צורך בקטעי קוד שלמים שעוסקים בביצוע גלגולים ושינויי גובה למיניהם שמופיעים בעץ AVL ונחסכים מביצוע בעץ לא מאוזן.
היינו מצפים שבעת הכנסת **סדרת מפתחות אקראיים** נקבל שזמן הריצה של הכנסת מפתח בממוצע לעץ AVL יהיה נמוך מזה של עץ לא מאוזן בזכות מנגנון האיזון שחוסם מלמעלה את סיבוכיות הזמן של הכנסת מפתח בעץ מאוזן. נשים לב שבאופן מפתיע התוצאות שעולות מהן המדידות הפוכות, כלומר מהמדידות ניתן לזהות כי זמן ההכנסה הממוצע של מפתח לעץ לא מאוזן נמוך יותר מזה שנעשה בעץ AVL.