סיבוכיות

|  |  |  |
| --- | --- | --- |
| פונקציה | סיבוכיות | הסבר |
| Empty | O(1) | בדיקה של קיום שורש – באם לא קיים (השורש מוגדר כשדה של מחלקת הAVLTree) |
| Search  recursiveSearch | O(logn) | חיפוש בינארי סטנדרטי – בכל צומת לבחור האם לרדת ימינה או שמאלה |
| Insert | O(logn) | התהליך מורכב ארבעה שלבים:  1)חיפוש סטנדרטי  2) הכנסת הצומת פיזית O(1)  3) לולאה שעולה במעלה הדרך עד לשורש. אורך הדרך לכל היותר O(logn) כגובה העץ.  4) בכל צומת בחינה האם יש צורך בסיבובים. |
| Delete | O(logn) | התהליך מורכב מחמישה שלבים:  1)חיפוש סטנדרטי  2) מחיקת הצומת פיזית ובדיקה על כמות הילדיםO(1)  3)במידת הצורך, הגעה לעוקב וביצוע מחיקה עליו ברקורסיה.  4) לולאה שעולה במעלה הדרך עד לשורש. אורך הדרך לכל היותר O(logn) כגובה העץ.  5) בכל צומת בחינה האם יש צורך בסיבובים על מנת לסדר מחדש את הBF. |
| Min | O(logn) | הפונקציה יורדת בלולאה עד לעלה השמאלי ביותר. מאחר שהעץ מאוזן, במקרה הגרוע ביותר הלולאה תיאלץ לפעול לאורך כמות צמתים השווה לגובה העץ – logn |
| Max | O(logn) | הפונקציה יורדת בלולאה עד לעלה הימני ביותר. מאחר שהעץ מאוזן, במקרה הגרוע ביותר הלולאה תיאלץ לפעול לאורך כמות צמתים השווה לגובה העץ – logn |
| keysToArray | O(n) | מעבר inorder על כל צמתי העץ. מבקרים בכל אחת מהצמתים פעם אחת בלבד. |
| infoToArray | O(n) | בדומה לפונקציה הקודמת, גם כאן מתבצע מעבר inorder על כל צמתי העץ ולאחר מכן העתקה שלו למערך המכיל את הערכים. מבקרים בכל אחת מהצמתים פעם אחת בלבד. |
| Size | O(1) | שדה של העץ שמתעדכן בכל מחיקה והכנסה של צומת חדש. |
| getRoot | O(1) | פעולה אחת של שליפת המידע מתא בזיכרון. |
| successor | O(logn) | הפונקציה בודקת קודם כל האם לצומת קיים בן ימני. במידה וכן ניקח את האיבר המינימלי מתת עץ שהבן המיני הוא השורש שלו – פעולה שלכל היותר חסומה על ידי logn שהוא גובה העץ הראשי. במידה ואין לו בן ימני, הפונקציה תחפש במעלה הדרך את הצומת הראשון שאין לו בן ימני (ולכל היותר תעלה logn רמות) כך שבכל רמה תתבצע פעולת השוואה אחת. |
| perfixXor | O(logn) |  |
| succPerfixXor | O(nlogn) | בתחילת הפונקציה יוצרים את מערך המפחות – פעולה שלוקחת O(n). לאחר מכן מבצעים חיפוש בינארי על כל אחד מאיברים הקטנים מk שהוזן. במקרה הגרוע זה יהיה nlogn פעולות. |

מדידות

* 1. עבור הפונקציה (היעילה) prefixXor (התוצאות מוצגות בננו-שניות):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי | עלות prefixXor ממוצעת )כל הקריאות( | עלות succPrefixXor ממוצעת )כל הקריאות( | עלות prefixXor ממוצעת)100 קריאות ראשונות( | עלות succPrefixXor ממוצעת)100 קריאות ראשונות( |
| 1 | 738 | 14269 | 125 | 4482 |
| 2 | 4105 | 47920 | 123 | 10042 |
| 3 | 159 | 46399 | 125 | 10248 |
| 4 | 122 | 73113 | 122 | 14357 |
| 5 | 123 | 90440 | 117 | 17468 |

מסקנות:

ציפינו שהמאה קריאות הראשונות יהיו די זהות בפונקציה הטובה משום שכמות הפעולות היא לא גבוהה במיוחד באף אחד מהמקרים, הבדלים של קבועים. אכן זאת הסיטואציה. באופן דומה משום שהעץ מאוזן ציפינו שגם הזמנים של הפונצקיה הלא יעילה יהיו במגמת עלייה משום שגודל העץ גדל וגם הפונקציה גדלה ביותר מאופן לינארי.

באותו אופן, ציפינו שעלות הקריאה הממוצעת בפונקציה היעילה תהיה די זהה משום שבתוחלת אנחנו קוראים לצומת שהיא לא עלה אבל היא בהחלט נמצאת אי שם ברמות התחתונות של העץ. כך שהממוצע המשוקלל בהחלט אמור לצאת זהה (יש מעט צמתים מאוד מאוד יעילים ומעט צמתים עם המון חישובים). בהסתכלות מאקרו על ממוצע של כלל הקריאות עם הפונקציה הלא יעילה, ניתן שוב להבחין בעלייה די משמעותית בממוצע משום שבכל קריאה נוסף עוד צומת לבדוק (סיבוכיות הפונקציה יותר גבוהה מלינארית) ולכן העלייה החדה במשך הזמן עבור פעולה אחת כזו.

1. השוואת עץ AVL עם עץ בינארי רגיל (התוצאות מוצגות בננו-שניות):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 | 16900 | 2106773 | 14042 | 16092 | 38213 | 32286 |
| 2 | 21503 | 8744797 | 19274 | 21721 | 75673 | 49623 |
| 3 | 45304 | 23001737 | 56514 | 29658 | 148753 | 93042 |
| 4 | 51346 | 39840796 | 29832 | 33337 | 78348 | 70400 |
| 5 | 79345 | StackOverflowError | 53490 | 38506 | 119519 | 58546 |

מסקנות: התוצאות שיצאו מסתדרות טוב מאוד עם מה שחשבנו שיקרה. בנוגע לסדרה חשבונית עולה של מספרים – זה אך טבעי שעבור עץ לא מאוזן ייקחו זמנים ארוכים מאוד להכנסה משום שהוא בכל הכנסה חייב לעבור n מספרים כדי להכניס את המספר הבא, וכאשר מגיעים למספרים גבוהים זה לוקח הרבה מאוד זמן. לעומת זאת, עץ מאוזן שומר על זמנים ארוכים אבל די סבירים בהתחשב במספרים הגדולים שהכנסנו, משום שמנגנון האיזון דואג שלא נצטרך לעבור יותר מlogn צמתים בכל n שאנחנו מכניסים.

בסדרה המאוזנת הזמנים יצאו יחסית קרובים – מה שנותן הגיון. ציפינו שהעץ המאוזן אולי יקח מעט יותר זמן בעבור הבדיקות שהוא עושה בכל הכנסה לגבי הBF כלומר הוא מוסיף עוד עלייה במעלה הדרך אל השורש מהצומת שהוכנס על מנת לוודא את הBF וזה ככל הנראה מקור הפער בין השניים.

בסדרת המספרים האקראית ציפינו כי עץ הAVL ללא ספק יהיה מהיר יותר אבל זו לא התוצאה בשטח. ככל הנראה הסבירות שבכל פעם יוכנס האיבר ה"גרוע" ביותר , כמו שקרה בסדרה החשבונית הוא אכן נמוך מאוד ובתוחלת רוב הפעמים נכניס צומת אקראית שהיא לא קטנה בצורה קיצונית משאר הצמתים או גדולה בצורה קיצונית משאר הצמתים וכך בעצם נוצרה סדרה סמי-מאוזנת שהוכנסה בכל פעם לעצים.