# תיעוד חיצוני

## צומת – AVLNode

פרמטרים:

* key (int) – מפתח לצומת, מספר שלם שייחודי לכל צומת.
* value (int) – ערך של צומת, מספר שלם.
* left (AVLNode) – בן שמאלי של צומת. מצביע על אובייקט צומת אחר. ערך ברירת מחדל: None.
* right (AVLNode) – בן ימני של צומת. מצביע על אובייקט צומת אחר. ערך ברירת מחדל: None.
* parent (AVLNode) – הורה של צומת. מצביע על אובייקט צומת אחר. ערך ברירת מחדל: None.
* height (int) – גובה של צומת. מספר טבעי עבור כל צומת אמיתי, או -1 עבור צומת וירטואלי

מתודות:

\_\_init\_\_(self, key, value)

מטרה: בנאי של המחלקה

דרך פעולה: אתחול של אובייקט לפי קלט של מפתח וערך. כל צומת מאותחל ללא בנים וללא הורה ובגובה -1. צומת וירטואלי מאותחל לפי הגדרה עם מפתח וערך None.

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

get\_left(self)

מטרה: להחזיר את הבן השמאלי

דרך פעולה: גישה לפרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

get\_right(self)

מטרה: להחזיר את הבן הימני

דרך פעולה: גישה לפרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

get\_parent(self)

מטרה: להחזיר הורה של צומת

דרך פעולה: גישה לפרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

get\_key(self)

מטרה: להחזיר מפתח של צומת

דרך פעולה: גישה לפרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

get\_value(self)

מטרה: להחזיר את הערך של צומת

דרך פעולה: גישה לפרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

get\_height(self)

מטרה: להחזיר גובה של צומת

דרך פעולה: גישה לפרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

set\_left(self, node)

מטרה: מגדיר בן שמאלי לצומת

דרך פעולה: דורס פרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

set\_right(self, node)

מטרה: מגדיר בן ימני לצומת

דרך פעולה: דורס פרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

set\_parent(self, node)

מטרה: מגדיר הורה לצומת

דרך פעולה: דורס פרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

set\_key(self, key)

מטרה: מגדיר מפתח של צומת

דרך פעולה: דורס פרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

set\_value(self, value)

מטרה: מגדיר ערך של צומת

דרך פעולה: דורס פרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

set\_height(self, height)

מטרה: מגדיר גובה לצומת

דרך פעולה: דורס פרמטר

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

update\_height(self)

מטרה: מעדכן גובה של צומת

דרך פעולה: לוקח את המקסימום בין הגבהים של בן ימני ושמאלי ומגדיר את המקסימום הזה כגובה של צומת

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

is\_real\_node(self)

מטרה: מחזיר True אם צומת הוא לא וירטואלי

דרך פעולה: ניגש לפרמטר key ומחזיר האם הוא None או לא

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

get\_balance\_factor(self)

מטרה: מחשב את ערך balance factor כפי שראינו בכיתה

דרך פעולה: מחזיר את ההפרש בין הגובה של הבן השמאלי לגובה של הבן הימני

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

is\_left\_child(self)

מטרה: מחזיר True אם הצומת הוא בן שמאלי של ההורה שלו

דרך פעולה: בודק אם קיים הורה ובודק אם הבן השמאלי של ההורה הוא הצומת שקרא לפונ'

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

is\_right\_child(self)

מטרה: מחזיר True אם הצומת הוא בן ימני של ההורה שלו

דרך פעולה: בודק אם קיים הורה ובודק אם הבן הימני של ההורה הוא הצומת שקרא לפונ'

סיבוכיות זמן:

ניתוח סיבוכיות במקרה הגרוע: -

## עץ – AVLTree

פרמטרים:

* root (AVLNode) – שורש של העץ, מצביע לצומת AVLNode. ערך ברירת מחדל: None
* \_size (int) – מספר הצמתים בעץ. מספר שלם אי-שלילי. ערך ברירת מחדל: 0

מתודות:

\_\_init\_\_(self)

מטרה: בנאי של המחלקה

דרך פעולה: מגדיר אובייקט עץ ריק

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: -

create\_tree(root)

מטרה: פונקציה סטטית שיוצרת עץ עם שורש קלט

דרך פעולה: מגדיר אובייקט עץ וקובע את הפרמטר root להיות הצומת קלט, קובע את הפרמטר \_size להיות 1, ומחזיר את העץ שהתקבל.

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: -

rebalance\_from\_node(self, node: AVLNode, should\_rebalance\_only\_once: bool)

מטרה: מבצע פעולות איזון (גלגולים או שינויי גובה) מהצומת קלט, למעלה עד לשורש העץ) ומחזיר את כמות פעולות האיזון שנעשו.

דרך פעולה: מאתחל משתנה rotate\_count=0. כל עוד לא הגענו לצומת ריק: מחשבים balance factor, אם : אם should\_rebalance\_only\_once==True וגם הגובה של הצומת גדול ב-1 מאחד הבנים שלו, אז קובעים את הצומת הבא להיות ההורה של הצומת, קוראים ל-rotate עם הצומת וה-bf שמחזיר את מספר הגלגולים שנעשו, וממשיכים לצומת הבא.

אחרת, מעדכנים את הגובה של הצומת, שזה מוסיף פעולת איזון אחת, ועולים להורה של הצומת.

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: בכל איטרציה של לולאת ה-while אנחנו מבצעים כמות קבועה של שינויי מצביעים, ומבצעים גלגולים. המקרה הגרוע הוא שאנחנו מתחילים מעלה ועולים כל הדרך עד לשורש העץ, אז הסיבוכיות היא לינארית בגובה העץ, כלומר אנחנו עושים עבודה.

search(self, key)

מטרה: חיפוש של מפתח במילון שאותו העץ מייצג

דרך פעולה: המתודה קוראת לפונקציה הרקורסיבית search\_rec עם השורש והמפתח שאותו מחפשים

סיבוכיות זמן: (זה הסיבוכיות של הפונ' הרקורסיבית)

ניתוח סיבוכיות זמן במקרה הגרוע: ניתוח רלוונטי בפונקציה הרקורסיבית

search\_rec(node, key)

מטרה: מציאת צומת שמיוצגת על ידי המפתח בתת העץ ששורשו הוא node

דרך פעולה: פונקציה רקורסיבית. מקרי בסיס: אם תת העץ שמיוצג על ידי הצומת ריק, נחזיר None (כלומר המפתח לא נמצא). אם תת העץ שמיוצג ע"י הצומת הוא בנוי רק מהצומת, מחזיר את הצומת.

אם המפתח שמחפשים קטן מהמפתח של הצומת הנוכחי אז באופן רקורסיבי מחפשים בתת העץ השמאלי. אם המפתח שמחפשים גדול מהמפתח של הצומת הנוכחי אז באופן רקורסיבי מחפשים בתת עץ הימני.

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: - המקרה הגרוע הוא להתחיל מהצומת ולסיים בעלה, ובכל איטרציה אנחנו עושים עבודה. מכך שזה עץ AVL אנחנו יודעים שגובה העץ הוא לכל היותר , אז סה"כ אנחנו עושים עבודה.

insert(self, key, val)

מטרה: הכנסה של צומת עם מפתח וערך שניתנים בקלט לעץ, ומחזיר את כמות פעולות האיזון שקרו בגלל ההכנסה

דרך פעולה: מעדכן גודל של העץ, מייצר צומת חדש עם המפתח וערך שהוכנסו. אם העץ ריק פשוט מגדירים את הצומת החדשה כשורש, אחרת קוראים לפונקציה BST\_insert שמכניסה צומת בצורה הסטנדרטית לעץ חיפוש בינארי. לאחר מכן מגדירים שני צמתים וירטואלים שיהיו בנים לצומת (כי הצומת בהכרח עלה), ולבסוף מחזירים את הפלט של rebalance\_from\_node שנותנים לו כקלט את ההורה של הצומת, ומסמנים את ה-flag של rebalance\_only\_once.

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: אנחנו מכניסים צומת לעץ בצורה הסטנדרטית לעץ חיפוש בינארי, שזה לוקח זמן. בנוסף אנחנו עושים פעולות של זמן, ומבצעים פעולת איזון אחת, אז עוד זמן. סה"כ .

BST\_insert(self, node, new\_node)

מטרה: הכנסה של צומת new\_node לעץ ששורשו node

דרך פעולה: פונקציה רקורסיבית, מכניסה צומת לעץ בדרך הסטנדרטית לעץ חיפוש בינארי

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: יהיה לינארי בגובה העץ, כלומר

rotate(self, node, bf)

מטרה: מבצע גלגולים כדי לשמור על ההגדרה של עץ AVL, ומחזירים את מספר הגלגולים שנעשו

דרך פעולה: אם ה-bf==2 אז מוצאים את ה-bf של הבן השמאלי, וכפי שנלמד בכיתה אם ה-bf של השמאלי הוא -1 אז מבצעים גלגול כפול (שמאלה ואז ימינה) על ידי קריאה לפונ' המתאימות. אחרת ה-bf של השמאלי הוא +1 ומבצעים גלגול יחיד ימינה.

אחרת מתקיים כי ה-bf של הצומת המקורי הוא -2, אז שואלים מה ה-bf של הימני ומבצעים באופן דומה גלגול כפול עבור bf==1 או גלגול יחיד אם bf==-1.

כאשר בכל המקרים האלה אנחנו מחזירים את מספר הגלגולים שנעשו

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: חישוב של bf של צומת ושל הילדים שלו לוקח זמן קבוע, וגלגולים לוקחים זמן קבוע

right\_rotate(self, node)

left\_rotate(self, node)

מטרה: מבצע גלגול ימינה/שמאלה

דרך פעולה: משנים מצביעים כפי שנלמד בכיתה. בודקים אם הצומת היה בן שמאלי או בן ימני, ומשנים מצביעים, כל זאת בהתאם לפסאודו-קוד שראינו בכיתה. לבסוף מעדכנים גובה של הצומת ושל ההורה שלו

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: שינוי מצביעים לוקח זמן קבוע, עדכון של גובה פעמיים גם לוקח זמן קבוע.

delete(self, node)

מטרה: מוחק צומת מהעץ ומחזיר את מספר פעולות האיזון שנעשו

דרך פעולה: ראשית מחסיר 1 מהגודל של העץ. קורא לפונ' BST\_delete כדי למחוק את הצומת מהעץ, ואז קורא לפונ' rebalance\_from\_node כדי לספור את פעולות האיזון, ומחזיר את זה.

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: הפונ' BST\_delete פועלת בזמן , והפונ' rebalance\_from\_node פועלת בזמן קבוע. אז סה"כ נקבל .

BST\_delete(self, node)

מטרה: מוחק צומת מעץ בדרך הסטנדרטית לעץ חיפוש בינארי, ומחזיר את ההורה של הצומת שנמחק

דרך פעולה: המתודה מפרידה בין 3 מקרים:

1. הצומת עלה
2. לצומת יש 2 ילדים
3. לצומת יש ילד אחד

אם הבן ימני והבן השמאלי לא צמתים אמיתיים אז אנחנו במקרה 1: אם אין הורה לצומת אז אנחנו במצב שהעץ בנוי רק משורש ופשוט מוחקים אותו. אחרת, יוצרים צומת וירטואלי וקובעים את ההורה שלו להיות ההורה של הצומת שעומד להימחק. אם הצומת הוא בן שמאלי אז מגדירים את הבן שמאלי של ההורה להיות הצומת הוירטואלי, אחרת מגדירים את הבן הימני של ההורה להיות הצומת הוירטואלי. ובכל מקרה קובעים את ההורה של הצומת שנמחק להיות None.

אם לצומת יש בן ימני ובן שמאלי אמיתיים אז אנחנו במקרה 2: מגדירים משתנה successor על ידי הפעלת הפונ' successor על הצומת, ומגדירים משתנה להורה של ה-successor. לאחר מכן אנחנו מורידים את ה-successor מהעץ, ומחליפים את הצומת ואת ה-successor. לאחר מכן אנחנו בודקים אם הצומת שנמחקה היא השורש, אז אנחנו קובעים את השורש של העץ להיות successor. אחרת אנחנו פשוט קובעים את ה-successor להיות הבן השמאלי או ימני של ההורה של הצומת שנמחק, תלוי איזה בן היה הצומת שנמחק. ולבסוף אנחנו מנתקים את הצומת מההורה שלו ומבניו.

אם אנחנו לא במקרה 1 או 2, אז אנחנו במקרה 3: בודקים אם הבן של הצומת הוא ימני או שמאלי, שומרים אל הבן מצביע, ומגדירים את set\_left/right של הצומת להיות None בהתאמה. אם ההורה של הצומת הוא None (כלומר הוא השורש) אז פשוט מגדירים את הבן להיות השורש החדש. אחרת, בודקים אם הצומת הוא בן ימני או שמאלי ומגדירים את הבן השמאלי/ימני של ההורה להיות הבן של הצומת שנמחק. ולבסוף מגדירים את ההורה של הבן שנמחק להיות ההורה של הצומת שנמחק ומתנקים את הצומת שנמחק מהעץ.

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: מקרים 1 ו-3 פועלים בזמן קבוע, זה רק שינוי של מצביעים. במקרה 2 הפעולה היחידה שלא לוקחת זמן קבוע זה successor, שפעולה זו לוקחת במקרה הגרוע . אז סה"כ נקבל זמן עבודה במקרה הגרוע של .

successor(node)

מטרה: מחזיר את הצומת שהוא ה-successor של צומת הקלט

דרך פעולה: מגדירים משתנה current להיות צומת הקלט. אם לצומת אין בן ימני, אז אנחנו עולים במעלה העץ עד שפונים ימינה, כלומר אומרים ש-current=current.get\_parent() כל עוד ההורה של currentהוא לא ריק ו-current הוא בן ימני. ברגע שעושים את הפנייה ימינה אנחנו מחזירים את ההורה של current.

אחרת לצומת יש בן ימני וצריך להחזיר את הצומת המינימלי בתת העץ הימני. כלומר מגדירים את current להיות הבן הימני, וכל עוד current הוא צומת אמיתי אנחנו מגדירים את current החדש להיות הבן השמאלי של current. לבסוף מחזירים את ההורה של current.

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: המקרה הגרוע הוא שאנחנו צריכים לעלות מעלה עד לשורש העץ, או לרדת משורש עד לעלה, ונבצע פעולות בזמן קבוע בכל איטרציה, כלומר סיבוכיות הזמן היא לינארית בגובה העץ. מכך שזהו עץ AVL הסיבוכיות תהיה .

avl\_to\_array(self)

מטרה: מחזיר מערך ממוין של האיברים במילון כזוגות סדורים של מפתח וערך

דרך פעולה: יוצר מערך (רשימה) ריק וקורא לפונ' הרקורסיבית avl\_to\_array\_rec עם המערך והשורש של העץ.

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: הפונ' הרקורסיבית לוקחת זמן לינארי בגודל העץ, אז זמן ריצה.

avl\_to\_array\_rec(node, array)

מטרה: פונקציה רקורסיבית שמבצעת in order traversal וממלאת את המערך בהתאם.

דרך פעולה: מקרה בסיס – אם הצומת ריקה, מפסיקים ריצה. אחרת מפעילים את הפונקציה באופן רקורסיבי על תת העץ השמאלי של הצומת, לאחר מכן שמים tuple של המפתח והערך של הצומת הנוכחית, ואז מפעילים את הפונקציה באופן רקורסיבי על תת העץ הימני

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: עוברים על כל הצמתים בעץ, ומבצעים פעולה בזמן קבוע בכל קריאה רקורסיבית, סה"כ נקבל זמן ריצה של .

size(self)

מטרה: מחזירה את כמות האיברים בעץ/מילון

דרך פעולה: גישה לשדה

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: -

split(self, node)

מטרה:

דרך פעולה:

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע:

join(self, tree2, key, val)

מטרה:

דרך פעולה:

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע:

join\_node(self, tree2, node)

מטרה:

דרך פעולה:

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע:

get\_root(self)

מטרה: מחזירה את השורש של העץ

דרך פעולה: גישה לשדה

סיבוכיות זמן:

ניתוח סיבוכיות זמן במקרה הגרוע: -