סיבוכיות: פשוט וקל

אני אתחיל בלציין שמעבר לרשימה זו, קיים נספח בסוף הקובץ (מתחיל בעמוד 5 וממשיך עד לסוף הקובץ, אז כן – זה כנראה מעמיק יותר מהרצוי) שמכיל ניתוח מעמיק יותר של כלל הפונקציות והסיבוכיות שלהן:

מחלקת $AVLNode$:

שדות שהתווספו:

1) min: מצביע על הצאצא עם המפתח המינימלי מבין כל הצאצאים של $self$.
2) max: מצביע על הצאצא עם המפתח המקסימלי מבין כל הצאצאים של $self$.
3) $size$: מספר שלם אי-שלילי המציין כמה צמתים אמיתיים (שאינם $None$ ואינם צמתים וירטואליים) מקושרים בתור צאצאים של $self$.

ניתוח מתודות (נתחיל במתודה $is\_real\_node(self)$, אחריה יבואו מתודות שלא היו מראש בקובץ השלד, ולבסוף שאר המתודות שהיו בקובץ השלד):

1. $is\_real\_node(self)$: אין דרישות קדם; סיבוכיות של $O(1)$.

מתודות שהתווספו לקובץ השלד ולא היו קיימות בו מראש:

2. $update\_fields(self)$: אין דרישות קדם; סיבוכיות של $O(1)$.
3. $is\_node(self, node)$: **קיימת דרישת קדם** לפיה $self$ ו- $node$ חייבים להיות מטיפוס $AVLNode$, חייבים לא להיות $None$ ולא להיות צומת וירטואלי; סיבוכיות של $O(1)$.
4. $sever\_ties(self)$: אין דרישות קדם; סיבוכיות של $O(1)$.
5. $set\_virtual(self)$: אין דרישות קדם; סיבוכיות של $O(1)$.
6. $child\_parent\_pointers(self, parent\_node, direction)$: אין דרישות קדם; סיבוכיות של $O(1)$.
7. $get\_\min(self)$: אין דרישות קדם; סיבוכיות של $O(1)$.
8. $set\_\min(self, node)$: אין דרישות קדם; סיבוכיות של $O(1)$.
9. $get\_\max(self)$: אין דרישות קדם; סיבוכיות של $O(1)$.
10. $set\_\max(self, node)$: אין דרישות קדם; סיבוכיות של $O(1)$.
11. $get\_size(self)$: אין דרישות קדם; סיבוכיות של $O(1)$.
12. $set\_size(self, h)$: אין דרישות קדם; סיבוכיות של $O(1)$.
13. $change\_size(self, delta)$: אין דרישות קדם; סיבוכיות של $O(1)$.
14. $balance\_factor(self)$: אין דרישות קדם; סיבוכיות של $O(1)$.
15. $get\_successor(self)$: **קיימת דרישת קדם** לפיה $self$ איננו $None$ ואיננו צומת וירטואלי; סיבוכיות במקרה הטוב של $O(1)$, ועבור d = עומק הצומת $self$ בעץ AVL שכולל את כל אבות אבותיו של $self$, את כל צאצאיו ואת $self$ עצמו, סיבוכיות במקרה הגרוע של $O(d+1)$. באופן שקול, עבור n = מספר הצמתים בעץ AVL שכולל את כל אבות אבותיו של $self$, את כל צאצאיו ואת $self$ עצמו, סיבוכיות במקרה הגרוע של $O(\log_2(n))$.
16. $rotation\_action(self, new\_child, boolean)$: **קיימת דרישת קדם** לפיה new_child חייב להיות צומת אמיתי (לא $None$ ולא צומת וירטואלי); סיבוכיות של $O(1)$.
17. $rotation\_node(self, direction, boolean)$: **קיימת דרישת קדם** לפיה $direction$ הוא המחרוזת "L", "l", "R" או "r"; סיבוכיות של $O(1)$.

18. $in\_oreder(self, direction)$: **קיימת דרישת קדם** לפיה $direction$ הוא המחרוזת "L", "l", "R" או "r" ; עבור n = מספר הצמתים הצאצאים של $self$, סיבוכיות של $O(n)$.

<u>מתודות שהיו מראש בקובץ השלד (מלבד $is\_real\_node(self)$):</u>

19. $get\_left(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
20. $get\_right(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
21. $get\_parent(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
22. $get\_key(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
23. $get\_value(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
24. $get\_height(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
25. $set\_left(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
26. $set\_right(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
27. $set\_parent(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
28. $set\_key(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
29. $set\_value(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
30. $set\_height(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.

<u>**מחלקת $AVLTree$:**</u>

למחלקה זו לא התווספו שדות.

ניתוח מתודות (נתחיל במתודות שלא היו מראש בקובץ השלד, ונמשיך בשאר המתודות שהיו בקובץ השלד):

<u>מתודות שהתווספו לקובץ השלד ולא היו קיימות בו מראש:</u>

1. $set\_self(self, tree)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
2. $rotation(self, direction, node)$: **קיימת דרישת קדם** לפיה $direction$ הוא המחרוזת "L", "l", "R" או "r" ; סיבוכיות של $O(1)$.
3. $re\_balance(self, parent\_node, old\_height, post\_insertion)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
4. $join\_with\_node(self, tree2, node)$: **קיימת דרישת קדם** לפיה $self$ ו - $tree2$ הם אובייקטים מסוג $AVLTree$ שאינם ריקים (כלומר השדה $root$ שלהם אינו מצביע ל - $None$ או לצומת וירטואלי). בנוסף, **קיימת דרישת קדם נוספת** לפיה ערך שדה ה - $height$ של הצומת אליו מצביע השדה $root$ של $self$ גדול מזה של $tree2$. בנוסף, **קיימת דרישת קדם נוספת** לפיה $node$ הינו צומת אמיתי (לא $None$ ולא צומת וירטואלי) שלא נמצא ב - $self$ ולא נמצא ב - $tree2$; סיבוכיות של:
$$O(self.root.get\_height(\,) - tree2.root.get\_height(\,) + 1)$$

<u>מתודות שהיו מראש בקובץ השלד:</u>

5. $search(self, key)$: אין דרישות קדם ; סיבוכיות במקרה הטוב של $O(1)$, ועבור n = מספר הצמתים ב - $self$, סיבוכיות במקרה הגרוע של $O(\log_2(n))$.
6. $insert(self, key, val)$: **קיימת דרישת קדם** לפיה key הוא איננו ערך של השדה key של אף אחד מהצמתים הקיימים ב - $self$ טרום פעולת המתודה ; סיבוכיות במקרה הטוב של $O(1)$, ועבור n = מספר הצמתים ב - $self$, סיבוכיות במקרה הגרוע של $O(\log_2(n))$.

7. $delete(self, node)$: **קיימת דרישת קדם** לפיה $node$ הוא צומת אמיתי (לא $None$ ולא צומת וירטואלי) שנמצא ב – $self$; סיבוכיות במקרה הטוב של $O(1)$, ועבור n = מספר הצמתים ב - $self$, סיבוכיות במקרה הגרוע של $O(\log_2(n))$.
8. $avl\_to\_array(self)$: אין דרישות קדם ; עבור n = מספר הצמתים ב - $self$, סיבוכיות של $O(n)$.
9. $size(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.
10. $split(self, node)$: **קיימת דרישת קדם** לפיה $node$ הוא צומת אמיתי (לא $None$ ולא צומת וירטואלי) שנמצא ב – $self$; סיבוכיות, בניתוח נאיבי, של $O(\log_2^2(n))$, ובניתוח מעמיק, של $O(\log_2(n))$.
11. $join(self, tree2, key, val)$: **קיימת דרישת קדם** לפיה כל המפתחות של הצמתים שב - $self$ קטנים מ - key, שקטן מכל המפתחות של הצמתים ב - $tree2$; סיבוכיות של $O(|self.root.get\_height(\,) - tree2.root.get\_height(\,)\,| \; + 1)$.
12. $get\_root(self)$: אין דרישות קדם ; סיבוכיות של $O(1)$.

חלק ניסויי / תיאורטי

1. להלן טבלה (ללא עיגולי התוצאות, קיימת אחת עם עיגולי התוצאות מתחתיה) המתארת את תוצאות עלות פעולת המתודה $join$ עם עץ בגודל $1000 \cdot 2^i$ עבור $i \in [1, 10] \cap \mathbb{N}$, בהתאם לצומת בו השתמשנו לפיצות העץ באמצעות המתודה $split$:

מספר סידורי i	עלות $join$ ממוצע עבור $split$ אקראי	עלות $join$ מקסימלי עבור $split$ אקראי	עלות $join$ ממוצע עבור $split$ של האיבר המקסימלי בתת העץ השמאלי	עלות $join$ מקסימלי עבור $split$ של האיבר המקסימלי בתת העץ השמאלי
$i = 1$	2.217325396825397	9	2.5	11
$i = 2$	2.237664807414808	9	2.45454545454546	12
$i = 3$	2.2622282717282722	10	2.5	13
$i = 4$	2.2507234456719756	9	2.4615384615384586	14
$i = 5$	2.2794170200714334	11	2.5	15
$i = 6$	2.2484871966269027	12	2.466666666666668	16
$i = 7$	2.2717889877580295	10	2.5	17
$i = 8$	2.26878695629276 2	15	2.4705882352941164	18
$i = 9$	2.27798605368921	11	2.5	19
$i = 10$	2.3094230512779284	13	2.473684210526322	20

לכל גודל עץ חזרתי על הניסוי 100 פעמים. בטבלה, כשמדובר על עלות ממוצעת, רשמתי את ממוצע העלויות הממוצעות של 100 החזרות על הניסוי, וכשמדובר על עלות מקסימלית, רשמתי את העלות המקסימלית של 100 החזרות על הניסוי. כשמדובר על עלות הפעולה $join$ – התייחסתי לעלות כאל הבדל הגבהים בין שני העצים שמחברים זה לזה (פלוס 1).
להלן הטבלה עם עיגולי התוצאות :

מספר סידורי i	עלות $join$ ממוצע עבור $split$ אקראי	עלות $join$ מקסימלי עבור $split$ אקראי	עלות $join$ ממוצע עבור $split$ של האיבר המקסימלי בתת העץ השמאלי	עלות $join$ מקסימלי עבור $split$ של האיבר המקסימלי בתת העץ השמאלי
$i = 1$	2.22	9	2.5	11
$i = 2$	2.24	9	2.45	12
$i = 3$	2.26	10	2.5	13
$i = 4$	2.25	9	2.46	14
$i = 5$	2.28	11	2.5	15
$i = 6$	2.25	12	2.47	16
$i = 7$	2.27	10	2.5	17
$i = 8$	2.29	15	2.47	18
$i = 9$	2.28	11	2.5	19
$i = 10$	2.31	13	2.47	20

2. סיבוכיות פיצול עץ AVL בצומת $node$ כלשהי היא $O(d_{node})$, כאשר d_{node} מתאר את עומת הצומת $node$.
כשאנחנו עושים $join$ לשני עצים על ידי צומת אב כלשהו (שלא נמצא באף אחד משני העצים) קיימים 3 מקרים:
   a. שני העצים ריקים. במקרה זה העלות של פעולת ה - $join$ תהיה 1.
   b. עץ אחד ריק, והשני לא ריק. נסמן את גובה העץ הלא ריק ב - h. במקרה זה העלות של פעולת ה - $join$ תהיה $(h-(-1))+1=h+2$. אנו יודעים ש - h הוא גובהו של עץ לא ריק, ולכן $h \geq 0$, ומכאן העלות של פעולת ה - $join$ במקרה זה היא לפחות $0+2=2$. כמו כן, אם נחזור להסתכל על השאלה שלנו ונסמן ב - h_{self} את גובה העץ המקורי שהיה לפני הפיצול, יכול להיווצר מצב שבו אנו מאחדים עץ ריק עם עץ בגובה $h_{self}-1$ או $h_{self}-2$ (דוגמה לכך היא המקרה בו מפצלים את העץ בצומת המקסימלי של תת העץ השמאלי של השורש שלו). לכן העלות של פעולת ה - $join$ במקרה זה היא לכל היותר $O\left(h_{self}\right)$. כלומר העלות של פעולת ה - $join$ במקרה זה היא בין 2 לבין $O\left(h_{self}\right)$.
   c. שני העצים לא ריקים. נסמן את גובה העץ השמאלי ב - h_{left}, ואת גובה העץ הימני ב - h_{right}. במקרה זה העלות של פעולת ה - $join$ תהיה $\left|h_{left}-h_{right}\right|+1$. אם נחזור להסתכל על השאלה שלנו, נשים לב כי $\left|h_{left}-h_{right}\right| \geq 1$, ולכן העלות של פעולת ה - $join$ במקרה זה היא לפחות $1+1=2$. כמו כן, יכול להיווצר מצב שבו אנו מאחדים עץ בן צומת בודד וילד של שורש העץ המקורי שהיה לפני הפיצול, כאן יתקיים $\left|h_{left}-h_{right}\right|=O\left(h_{self}\right)$. לכן העלות של פעולת ה - $join$ במקרה זה היא לכל היותר $O\left(h_{self}\right)$. כלומר העלות של פעולת ה - $join$ במקרה זה היא בין 2 לבין $O\left(h_{self}\right)$.

כשנתבונן בפיצול העץ $self$ בצומת אקראי $node$ כלשהו, נשים לב שרוב הפעמים נקבל הפעלה של $join$ במקרה b או במקרה c, ומעבר לכך – שברוב המקרים נפעיל את $join$ במקרה c כאשר שני העצים שיאוחדו יהיו בגדלים דומים, כך שהעלות של הפעולה תהיה קרובה ל – 2. לכן נקבל באמת שבממוצע העלות של פעולת ה - $join$ בפיצול העץ $self$ עם צומת $node$ אקראי יהיה קרוב ל – 2, אך גדול ממנו (בדיוק כפי שקיבלנו).

כשנתבונן בפיצול העץ $self$ בצומת המקסימלי של תת העץ השמאלי של השורש של $self$, אנחנו יודעים שיש לנו פעולת $join$ אחת שהעלות שלה היא $O(\mathrm{h_{self}})$ לפי מקרה b. שאר פעולות ה - $join$ יהיו או פעולה אחת לפי מקרה b שהעלות שלה תהיה 2 (הפעם הראשונה שאנחנו עושים join כדי ליצור את העץ השמאלי לאחר הפיצול), ובשאר הפעמים נפעל לפי מקרה c כאשר $|h_{left} - h_{right}| \in \{0, 1\}$, כלומר העלות של פעולת ה - join בשאר הפעמים תהיה 2 או 3. כשנעשה ממוצע לכל הפעולות האלו אכן צפוי להתקבל ערך בסביבות 2, שגדול ממנו (בדיוק כפי שקיבלנו).

3. קודם כל נתחיל בלציין שאנחנו משתמשים בפיצול למקרים של עלות הפעולה $join$ שהוצג בשאלה הקודמת.
כשנתבונן בפיצול העץ $self$ בצומת המקסימלי של תת העץ השמאלי של השורש של $self$, אנו יודעים כבר (מהשאלה הקודמת) שיש לנו פעולת $join$ אחת שהעלות שלה היא $O(\mathrm{h_{self}})$ לפי מקרה b, ומעבר לכך – שזוהי העלות המקסימלית של פעולת $join$ בתהליך פיצול זה. מתקיים $\log_2(1000) \sim 10$, ולכן לכל $i \in [1, 10] \cap \mathbb{N}$, מתקיים $h_{self} \sim 10 + i$. כלומר קיבלנו בדיוק מה שציפינו.
המקרה של פיצול העץ $self$ בצומת אקראי $node$ כלשהו לא שונה בהרבה: פה אנחנו לא יודעים לבטח שיש לנו הפעלה כזו קיצונית, והסבירות לבחור ב - $node$ שיגרום להפעלה במקרה קצה שכזה היא נמוכה, אך לאחר 100 חזרות על כל ניסוי אכן התקרבנו לזה. גם פה העלות המקסימלית תהיה אסימפטוטית כגובה העץ, והגענו למקרים שכאלה ברוב ה – iים שהכנסנו. בוודאות אכן הגענו לעלות של $O\left(\log_2\left(1000 \cdot 2^i\right)\right)$ כצפוי וכרצוי.

<u>נספח – סיבוכיות: ניתוח מעמיק</u>

אני אתחיל בלומר שלכל פונקציה קיים פירוט בקובץ ה - $.py$ שלי. פירוט זה הינו מעמיק (אך לא מדי, לדעתי – בול במידה, אולי קצת מעבר לחלק מהאנשים), ונוצר במחשבה שאדם אחר יוכל לתחזק את הקוד שלי במידת הצורך (לא שאני חושב שזה יקרה פה – אני מנסה ליצור הרגלים עבור עבודה עתידית בתור מתכנת, בתקווה שאמצא כמה שיותר מהר). פירוט זה הינו לא רק בצורה של docstring מפורט שמסביר על כל פרמטר, על ערכי החזרה וכו', אלא גם בצורה של $comments$ בתור הקוד עצמו, כל אחת ממוקמת בשורה (או בחלק) הרלוונטי לה. אם משהו ממה שאני רושם בקובץ זה אינו ברור – ניתוח הסיבוכיות מופיע בקובץ ה - $.py$ שלי בצורה של סיכום סיבוכיות כל

מתודה כבר ב - $docstring$ שלה, ו - $comments$ על השורות שגוררות את הסיבוכיות בהן רשום מה הסיבוכיות שהן גוררות.

נעיר מראש שסיבוכיות קריאה לערכו של שדה של אובייקט ממחלקה X בתוך מתודות של מחלקה זו היא $O(1)$, וכך גם סיבוכיות יישום ערך חדש לשדה שכזה. כך, למשל, בתוך המחלקה $AVLNode$, בתוך כל מתודה של $AVLNode$ ועבור כל אובייקט $node$ מסוג $AVLNode$, הקריאה $node.height$ הינה בסיבוכיות של $O(1)$. באופן דומה גם על אובייקטים מסוג $AVLTree$ בקריאות לשדות המחלקה.

ננתח את סיבוכיות כלל הפונקציות שבקובץ, מלבד אלו שרק מבצעות קריאה לשדות מחלקה – שכן, כאמור, הן בסיבוכיות של $O(1)$.

נתחיל עם מחלקת <u>***AVLNode***</u>. עבור מחלקה זו הוספנו את השדות min, max, ו - $size$. עבור צומת $node$ כלשהו מסוג $AVLNode$, השדות min ו - max הינם מצביעים לצומת עם המפתח המינימלי והמקסימלי (בהתאמה) מבין כל הצאצאים של $node$ (min מצביע לצאצא השמאלי ביותר ו - max מצביע לצאצא הימני ביותר). כמו כן, השדה $size$ מציין את כמות הצמתים שמקושרים ל - node בתור צאצאים שלו (לא רק ילדים, אל גם נכדים, נינים, וכן הלאה), כולל $node$ עצמו. באופן טבעי - אם node הוא $None$ או צומת וירטואלי, מתקיים $\mathrm{node.size} = 0$, ולכך גם שדה זה מאותחל. השדות min ו - max מאותחלים להיות $None$.

נתחיל בלנתח את המתודות שהיו קיימות בקובץ השלד מראש, ונתחיל דווקא מהמתודה $is\_real\_node(self)$, מאחר ויש שימוש במתודה זו גם בתוך מתודות אחרות במחלקה. נעיר רק שבמחלקה זו כל פעם שמופיע $self$, מדובר באובייקט בשם $self$ מהמחלקה AVLNode :

- עבור המתודה $is\_real\_node(self)$, בדיקה אם $self$ הינה $None$, אם המפתח שלה הינו $None$ או אם הגובה שלה הינו -1, נעשית בסיבוכיות של $O(1)$. החזרות $True$ או $False$ גם הן נעשות בסיבוכיות של $O(1)$, לכן סיבוכיות המתודה היא $O(1)$. **מעכשיו לא אמשיך לפרט על פעולות שלוקחות $O(1)$, אפרט רק על פעולות שמשליכות סיבוכיות לא קבועה ולפעמים (או אולי אפילו לרוב כי אני חופר) אציין כי קיימות פעולות שלוקחות $O(1)$!**
- עבור המתודות $get\_parent(self)$, $get\_right(self)$, $get\_left(self)$, $get\_key(self)$, $get\_value(self)$, $get\_height(self)$, set_left(self), set_right(self), $set\_parent(self)$, set_key(self), set_value(self), ו - set_height(self), סיבוכיות של $O(1)$.

עבור מתודות שהתווספו למחלקה (כאן גם אסביר מה תפקידן) :

- המתודה $update\_fields(self)$ בודקת כמה ילדים יש ל - $self$, ולפי כמות הילדים מעדכנת את השדות של $self$. סיבוכיות של $O(1)$.
- המתודה $is\_node(self, node)$ בודקת האם שתי הצמתים self ו - node זהים על ידי בדיקה שכל ערכי השדות שלהם זהים. למתודה זו **קיימת דרישת קדם** לפיה $self$ ו - node צריך להיות אובייקטים מסוג AVLNode שאינם None ואינם צמתים וירטואליים. סיבוכיות של $O(1)$.
- המתודה $sever\_ties(self)$ מיישמת את הערך $None$ לשדות $left$, right, $parent$, min, ו - max של $self$ ובכך מנתקת אותו מלהצביע על כל צומת אחר. סיבוכיות של $O(1)$.
- המתודה $set\_virtual(self)$ מיישמת את הערך $None$ לשדה key של $self$, את הערך -1 לשדה $height$, את הערך 0 לשדה $size$ ומבצעת את פעולת $sever\_ties$ עבור $self$. סיבוכיות של $O(1)$.

- המתודה $child\_parent\_pointers(self, parent\_node, direction)$ מבצעת קישור בין $self$ לבין הצומת $parent\_node$ במידה ו - $parent\_node$ אכן קיים ואינו וירטואלי. $direction$ מייצג מאיזה כיוון נצטרך לפנות מהצומת $parent\_node$ בכדי להגיע אל $self$. מתודה זו בסך הכל מיישמת ערכים בשדות של $self$ ושל $parent\_node$, ולכן היא בסיבוכיות של $O(1)$.
- המתודה $get\_\min(self)$ מחזירה את הצומת בעל המפתח המינימלי מבין כל צאצאיו של $self$ (במידה והוא קיים ואינו וירטואלי), מדובר בקריאה לערך בשדה ולכן סיבוכיות של $O(1)$.
- המתודה $set\_\min(self, node)$ מיישמת למצביע min של $self$ הצבעה אל הצומת $node$. סיבוכיות של $O(1)$.
- המתודה $get\_\max(self)$ מחזירה את הצומת בעל המפתח המקסימלי מבין כל צאצאיו של $self$ (במידה והוא קיים ואינו וירטואלי), מדובר בקריאה לערך בשדה ולכן סיבוכיות של $O(1)$.
- המתודה $set\_\max(self, node)$ מיישמת למצביע max של $self$ הצבעה אל הצומת $node$. סיבוכיות של $O(1)$.
- המתודה $get\_size(self)$ מחזירה את כמות הצאצאים שמקושרים אל $self$ בתור צאצאים (כולל $self$) עצמו. מדובר בקריאה לערך של השדה $size$ של $self$. סיבוכיות של $O(1)$.
- המתודה $set\_size(self, s)$ מיישמת את הערך s לשדה $size$ של $self$. סיבוכיות של $O(1)$.
- המתודה $change\_size(self, delta)$ מוסיפה את $delta$ לערך של השדה $size$ של $self$, ומיישמת את תוצאת החיבור לתוך ערך השדה. סיבוכיות של $O(1)$.
- המתודה $balance\_factor(self)$ מחזירה בכמה גדול ערך השדה $height$ של הילד הימני של $self$ מאשר ערך שדה זה של הילד השמאלי של $self$. סיבוכיות של $O(1)$.
- המתודה $get\_successor(self)$ מחזירה את הצומת אליו מצביע השדה min של הילד הימני של $self$ במידה וילד זה אכן קיים (חלק זה הינו בסיבוכיות של $O(1)$). אחרת, מתודה זו מסתכלת על כל האבות הקדמונים של $self$ ובודקת האם אחד מהם הוא בן שמאלי של ההורה שלו על ידי שימוש במתודה is_node. למתודה זו **קיימת דרישת קדם** לפיה $self$ אינו $None$ ואינו צומת וירטואלי. במקרה הגרוע ביותר, כאשר $self$ הוא צומת בעומק d, אליו מצביע השדה min של שורש עץ AVL, נקבל שסיבוכיות המתודה היא $O(d+1)$. באופן שקול, אם בעץ יש n צמתים, אזי $d = O(\log_2(n))$, ולכן סיבוכיות המתודה במקרה הגרוע ביותר תהיה $O(\log_2(n))$.
- המתודה $rotation\_action(self, new\_child, boolean)$ מבצעת גלגול לעץ ה - AVL ששורשו $self$ ומעדכנת את העץ (כלומר את שדות הצמתים שמקושרים כצאצאים של $self$). למתודה זו **קיימת דרישת קדם** לפיה new_child חייב להיות צומת אמיתי (לא $None$ ולא צומת וירטואלי). המתודה מחזירה את new_child לאחר כל העדכונים הללו במידה ו - $boolean$ הינו $True$, אחרת היא מחזירה $None$. מאחר ומדובר בעדכון שדות ובבדיקת תנאים בסיבוכיות קבועה בלבד, הסיבוכיות של המתודה הינה $O(1)$.
- המתודה $rotation\_node(self, direction, boolean)$ מקבלת את כיוון הגלגול הרצוי על ידי $direction$, ובהתאם אליו יוצרת מצביע לצומת בשם new_child (שהוא הבן הימני של $self$ אם $direction$ מציין סיבוב לשמאל, ואחרת הוא הבן השמאלי שלו), ומעדכנת את השדות של new_child ושל $self$ בהתאם למתרחש בגלגול הרצוי. למתודה זו **קיימת דרישת קדם** לפיה $direction$ הוא המחרוזת "L", "l", "R" או "r". לבסוף היא מחזירה את תוצאת הקריאה למתודה $rotation\_action$ על ידי הצומת $self$ עם

הפרמטרים new_child שנוצר ו - $boolean$ שניתן לה בתחילה. מדובר בעדכון שדות ובקריאה למתודה שהסיבוכיות שלה היא $O(1)$, ולכן סיבוכיות המתודה היא $O(1)$.

- המתודה $in\_order(self, direction)$ מחזירה רשימה של הצאצאים בכיוון הנתון על ידי $direction$ של $self$, הממוינת לפי ערכי השדה key של כל אחד מהצאצאים, מהקטן לגדול. למתודה זו **קיימת דרישת קדם** לפיה $direction$ הוא המחרוזת "L", "l", "R" או "r". המתודה עושה זאת על ידי קריאות רקורסיביות לעצמה עם ילדי הצומת $self$ בכל פעם, ולכן, בהינתן שלצומת $self$ יש n צאצאים, בכל אחד מהכיוונים ישנם $= O\left(\frac{n}{2}\right)$ $O(n)$ צאצאים. כל קריאה הינה בסיבוכיות של $O(1)$, ולכן סיבוכיות המתודה היא $O(n)$.

כעת נעבור למחלקה ***AVLTree***, אליה לא הוספתי שדות. נתחיל במתודות שלא היו מראש בקובץ השלד, ואסביר מה תפקידן. נעיר רק שבמחלקה זו כל פעם שמופיע $self$, מדובר באובייקט בשם $self$ מהמחלקה AVLTree :

- המתודה $set\_self(self\ , tree)$ מיישמת את הערך $None$ לשדה $root$ של $self$ אם $tree$ הינו $None$, ואחרת היא מיישמת את המצביע $tree.root$ לשדה זה. סיבוכיות של $O(1)$.
- המתודה $rotation(self, direction, node)$ מחזירה $None$ לאחר ביצוע הגלגול הרצוי בעץ. למתודה זו **קיימת דרישת קדם** לפיה $direction$ הוא המחרוזת "L", "l", "R" או "r". במקרה והמצביע $self.root$ מצביע על הפרמטר $node$ שהתקבל, המתודה מיישמת לשדה זה מצביע לתוצאה של הקריאה למתודה $rotation\_node$ של מחלקת $AVLNode$ על ידי הצומת $node$ ועם הפרמטרים $direction$ ו - $True$, אחרת היא מבצעת קריאה למתודה זו על ידי הצומת $node$ ועם הפרמטרים $direction$ ו - $False$ (ולא מיישמת את תוצאת קריאה זו, מזכיר – התוצאה תהא $None$). מאחר ואנו קוראים למתודה עם סיבוכיות של $O(1)$, נסיק כי סיבוכיות המתודה היא $O(1)$.
- המתודה $re\_balance(self, parent\_node, old\_height, post\_insertion)$ מבצעת איזון מחדש לעץ לאחר הוספת צומת חדש / מחיקת צומת קיים / פיצול העץ / איחוד העץ עם עץ אחר. ניזכר במקרים מההרצאה בהם אנו מבצעים איזון מחדש :

1. אם התוצאה של המתודה $balance\_factor$ שהופעלה עם צומת הורה לצומת שעבר שינוי (צומת שהתווסף / צומת שנמחק / הורה של החלק בעץ שעבר איחוד) היא קטנה מ -2 בערך מוחלט, והגובה של צומת הורה זה לא השתנה, אין צורך באיזון מחדש – רק בעדכון שדות הצמתים בדרך מצומת הורה זה ועד לשורש (כולל).
2. אם התוצאה של המתודה $balance\_factor$ שהופעלה עם צומת הורה לצומת שעבר שינוי (צומת שהתווסף / צומת שנמחק / הורה של החלק בעץ שעבר איחוד) היא קטנה מ -2 בערך מוחלט, והגובה של צומת הורה זה השתנה, נעדכן את שדות הצמתים בדרך מצומת הורה זה ועד לשורש (כולל), ונחשיב זאת כפעולת איזון מחדש אחת.
3. אם התוצאה של המתודה $balance\_factor$ שהופעלה עם צומת הורה לצומת שעבר שינוי (צומת שהתווסף / צומת שנמחק / הורה של החלק בעץ שעבר איחוד) היא 2 או -2, נבצע גלגול (כאשר גלגול בודד, בין אם לימין או לשמאל, נחשב כפעולת איזון מחדש אחת, ושני גלגולים ברצף, בין אם גלגול לימין ואז גלגול לשמאל ובין אם גלגול לשמאל ואז גלגול לימין, נחשבים כשתי פעולות איזון מחדש) על ידי קריאה למתודה $rotation$ על ידי $self$ עם הפרמטרים המתאימים. נעיר שכאן נכנס הפרמטר $post\_insertion$ לפעולה, ואם הוא $False$, התנאים לפיהם מתבצע גלגול מתאימים לתנאים שהוצגו בכיתה לאיזון מחדש לאחר מחיקת צומת.

המתודה מחזירה את מספר פעולות האיזון מחדש שנדרשו אחרי פעולת שינוי העץ, בדרך מצומת ההורה של הצומת שהשתנה אל שורש העץ, על ידי קריאה רקורסיבית לעצמה.

נשים לב שכל קריאה שכזו הינה בסיבוכיות של $O(1)$, שכן היא קוראת למתודות בסיבוכיות של $O(1)$ ומעדכנת שדות בלבד, וכי קיימות:

$self.root.get\_height() - parent\_node.get\_height() + 1$ חזרות כאלה. נסמן $h_{self} = self.root.get\_height()$ וגם:

$h_{parent\_node} = parent\_node.get\_height()$, אזי סיבוכיות המתודה היא $O\left(h_{self} - h_{parent\_node} + 1\right)$.

- המתודה $join\_with\_node(self, tree2, node)$: מאחדת בין העץ $self$ לבין העץ $tree2$, שגם הוא מסוג $AVLTree$, כך ש - $node$ יהיה השורש החדש של העץ שייווצר (עץ זה לאו דווקא יהיה מסוג $AVLTree$, ותיתכן אפשרות לצורך בסידור מחדש לאחר יצירתו). למתודה זו **קיימת דרישת קדם** לפיה $self$ ו - $tree2$ אינם עצים ריקים (כלומר קיים בהם לפחות שורש שאינו $None$ ואינו צומת וירטואלי). **קיימת דרישת קדם נוספת** לפיה הערך של השדה $height$ של הצומת אליו מפנה השדה $root$ של $self$ יהיה גדול מזה של $tree2$. ובנוסף, **קיימת דרישת קדם נוספת** לפיה $node$ הינו צומת אמיתי (לא $None$ ולא צומת וירטואלי) שאינו קיים ב - $self$ ואינו קיים ב - $tree2$. ראשית, על ידי לולאת $while$, המתודה מוצאת את הצומת הראשונה (בין אם מימין ובין אם משמאל, תלוי האם ערכי השדות key של הצמתים ב- $self$ קטנים או גדולים מאלה של הצמתים ב - $tree2$, בהתאמה) ב - $self$ שערך השדה $height$ שלה קטן או שווה לערך השדה $heigh$ של הצומת אליו מצביע השדה $root$ של $tree2$, וקוראת לצומת זה $child$. מציאת $child$ לוקחת $self.root.get\_height() - tree2.root.get\_height()$ חזרות על הלולאה, כל אחת בסיבוכיות של $O(1)$, לכן אם נסמן:
$h_{self} = self.root.get\_heigh()$ וגם $h_{tree2} = tree2.root.get\_height()$, אזי מציאת $child$ הינה בסיבוכיות של $O\left(h_{self} - h_{tree2}\right)$. להורה של צומת זה היא קוראת $parent\_node$. לאחר מכן היא מעדכנת את השדות של הצמתים $child$ ו - $parent\_node$ ושל שורשי העצים $self$ ו - $tree2$, קוראת למתודה $re\_balance$ על ידי $self$ עם הפרמטרים $parent\_node$, $parent\_node.get\_hegiht()$ ו – $False$. לבסוף היא מחזירה $None$. נשים לב שהמתודה $re\_balance$ תבצע:
$h_{self} - h_{tree2} + 1$ קריאות לעצמה, כל קריאה כזו בסיבוכיות של $O(1)$, ולכן סיבוכיות המתודה היא $O\left(h_{self} - h_{tree2} + 1\right)$.

עבור מתודות שהיו קיימות בקובץ השלד:

- המתודה $search(self, key)$ מחפשת צומת ב - $self$ שערך השדה key שלה יהיה הפרמטר key שניתן לה. ראשית היא בודקת אם $self$ אינו עץ ריק ואם הפרמטר key אכן יכול להיות בתוך העץ (לא קטן מהמפתח המינימלי ולא גדול מהמקסימלי). בהנחה ששני התנאים האלו התקיימו, היא בודקת האם הפרמטר key הינו המפתח המינימלי או המקסימלי שיש בעץ, ואם כן היא מחזירה את הצומת המתאים למפתח זה. במקרה הגרוע ביותר כל הבדיקות האלו נכשלו והמתודה צריכה לבדוק האם הפרמטר key יכול להיות בתת העץ הימני או בתת העץ השמאלי של השורש (בדיקה קלילה ב - $O(1)$), כשמצאה באיזה תת עץ הפרמטר יכול להיות היא מבצעת לעצמה קריאה רקורסיבית על ידי תת העץ המדובר ועם הפרמטר key ללא שינוי. במקרה הגרוע ביותר, הפרמטר key יהא זהה לצומת בעומק של $O(\log_2(n))$ (כאשר n הוא מספר הצמתים בעץ המקורי) שאינו הצומת בעל המפתח המינימלי או המקסימלי של העץ, ולכן יהיו $O(\log_2(n))$ קריאות רקורסיביות לפונקציה, כל אחת בסיבוכיות של $O(1)$. לכן סיבוכיות המתודה במקרה הגרוע ביותר היא $O(\log_2(n))$.

- המתודה $insert(self, key, val)$ מכניסה צומת חדש לעץ שערך השדה key שלו הוא הפרמטר key, וערך השדה $value$ שלו הפרמטר val, ומחזירה את כמות פעולות האיזון מחדש של העץ לאחר ההכנסה. למתודה זו **קיימת דרישת קדם** שהפרמטר key לא יהיה מפתח של צומת שקיים כבר ב - $self$. המתודה ראשית יוצרת צומת חדש בשם new_node שערכי השדות שלו תואמים לערכי השדות הרצויים בצומת שצריך להכניס. אם העץ היה ריק, השדה $root$ של $self$ יצביע על new_node והמתודה תחזיר 0. אחרת המתודה תמצא את הצומת בעץ שישמש כהורה לצומת החדש ותקרא לו $parent\_node$, ואז תקרא למתודה $child\_parent\_pointers$ בעזרת new_node עם הפרמטרים $parent\_node$ ו - $direction$ המתאים לכיוון אליו תוכנס new_node (משמאל או מימין ל - $parent\_node$). לאחר מכן המתודה תבצע עדכון לשדה $size$ של $parent\_node$ על ידי קריאה למתודה $change\_size$ על ידי $parent\_node$ עם הפרמטר $+1$, ולבסוף היא תחזיר את תוצאת הקריאה למתודה $re\_balance$ על ידי $self$ עם הפרמטרים $parent\_node$, $parent\_node.get\_height()$ ו - $True$. הסיבוכיות נובעת מהקריאה למתודה $re\_balance$. אם נסמן ב – n את כמות הצמתים ב - $self$, אזי המקרה הגרוע ביותר יהיה כשנכניס את הצומת החדש בתור ילד של צומת שנמצא בעומק המקסימלי. במקרה זה נקבל כי (מהניתוח של המתודה $re\_balance$): $h_{self} - h_{parent\_node} = O(\log_2(n))$, ולכן סיבוכיות המתודה במקרה הגרוע ביותר היא $O(\log_2(n))$.
- המתודה $delete(self, node)$ מוחקת את הצומת $node$ מ - $self$, ומחזירה את מספר פעולות האיזון מחדש שנדרשו כדי לאזן את העץ לאחר המחיקה. למתודה זו **קיימת דרישת קדם** לפיה $node$ הוא צומת אמיתי (לא $None$ ולא צומת וירטואלי) שקיים ב - $self$. ראשית המתודה קוראת להורה של $node$ בשם $node\_parent$, לילד השמאלי שלו בשם $node\_left$, ולילד הימני שלו בשם $node\_right$. קיימים 3 מקרים למחיקת הצומת $node$:

1. לצומת $node$ היו 2 ילדים אמיתיים (לא $None$ ולא צמתים וירטואליים): במקרה זה המתודה קוראת למתודה $get\_successor$ על ידי $node$, ולצומת המוחזר היא קוראת $successor$. נשים לב של - $successor$ אין ילד שמאלי (אחרת הוא היה היורש של $node$), וכי הוא בן ימני של צומת כלשהי (שכן ל - $node$ יש בן ימני, ולכן המתודה $successor$ רק תחזיר את הצומת אליו מצביע השדה min של $node$). לכן מציאת הצומת $successor$ הינה בסיבוכיות של $O(1)$. כמו כן, לילד של $successor$ המתודה קוראת $successor\_child$. כעת קיימים לנו שני מקרים:

A. המקרה בו $successor$ הוא לא $node\_right$: במקרה זה ל - $successor$ בוודאות יש הורה שלא ימחק. להורה זה המתודה קוראת בשם $successor\_parent$. לאחר עדכוני שדות מתאימים וקריאות למתודה $child\_parent\_pointers$ על מנת להסדיר את הקשר בין $node\_left$ ו - $node\_right$ לבין $successor$, וטיפול בשדה ה - $size$ של $successor\_parent$, מתבצעת בדיקה האם $successor\_child$ אכן צומת אמיתי (לא $None$ ולא צומת וירטואלי). לאחר בדיקה זו מסדירים את המצביעים של הבן השמאלי של $successor\_parent$, מצביע ההורה של $successor\_child$ (במידה והוא אכן צומת אמיתי), ושדה ה - min של $successor\_parent$. כעת המתודה בודקת האם $node$ הוא השורש של העץ. אם לא, המתודה מסדרת את השדה שהצביע על $node$ ב - $node\_parent$ להצביע על $successor$, ומעדכנת את השדות min ו - max של $node\_parent$ לפי הצורך.

B. המקרה בו $successor$ הוא כן $node\_right$: במקרה זה ל - $successor$ בוודאות יש הורה שכן ימחק, והוא $node$ עצמו. המתודה משתמשת במתודה $child\_parent\_pointers$ על מנת לקשר את $node\_left$ להיות בנו השמאלי של $successor$, ומעדכנת את שדה ה - min של $successor$. במידה ו - $node$ לא היה השורש של העץ, אזי $node\_parent$ איננו צומת וירטואלי ואיננו $None$, לכן המתודה מעדכנת את שדה ה - $size$ של $node\_parent$, ומסדרת את השדה שהצביע על $node$ ב - $node\_parent$ להצביע על $successor$, ומעדכנת את השדות min ו - max של $node\_parent$ לפי הצורך.

כך או כך, המתודה כעת קוראת למתודה $sever\_ties$ ולאחר מכן למתודה $set\_virtual$ על ידי $node$, ולבסוף מחזירה את תוצאת הקריאה למתודה $re\_balance$ על ידי $self$ עם הפרמטרים $successor$, $successor.get\_height()$, ו - $False$. נשים לב שעד שורת ההחזרה, סיבוכיות מקרה זה הייתה $O(1)$, ולכן סיבוכיות המתודה במקרה זה היא:

$O(self.root.get\_height() - successor.get\_height() + 1)$. במקרה הגרוע ביותר, $node$ הוא הורה (או ההורה של ההורה) של 2 עלים בעומק המקסימלי, שהוא $O(\log_2(n))$, כאשר n הוא מספר העלים בעץ.. לכן נקבל כי במקרה הגרוע ביותר מתקיים:

$self.root.get\_height() - successor.get\_height() = O(\log_2(n))$,
ולכן סיבוכיות המתודה במקרה הגרוע ביותר של מקרה זה היא $O(\log_2(n))$.

2. לצומת $node$ היה רק ילד אמיתי אחד (לא $None$ ולא צומת וירטואלי), גם פה קיימים לנו שני מקרים:
אם $node$ לא היה השורש של העץ, המתודה קוראת לילד האמיתי היחיד שלו בשם $child\_node$. היא מעדכנת את השדות min ו - max של $node\_parent$ בהתאם לכיוון המפנה אל $node$ מ - $node\_parent$ ומבצעת קישור בין $child\_node$ לבין $node\_parent$ על ידי קריאה למתודה $child\_parent\_poiters$.
אם $node$ היה השורש של העץ, המתודה מיישמת את המצביע $root$ של $self$ להצביע על הילד של $node$, קוראת למתודה $sever\_ties$ על מצביע זה, מעדכנת את שדות ה min – וה - max של מצביע זה, ואז קוראת למתודות $sever\_ties$ ו - $set\_virtual$ בזו אחר זו על ידי $node$. לבסוף היא מחזירה את הערך 1.
כמו שניתן לראות, סיבוכיות מקרה זה היא $O(1)$.

3. לצומת $node$ לא היו אף ילדים אמיתיים (לא $None$ ולא צומת וירטואלי, כלומר שניהם היו או $None$ או צומת וירטואלי), כלומר $node$ היה עלה. גם פה קיימים לנו 2 מקרים:
אם $node$ לא היה השורש של העץ, המתודה מעדכנת את השדות הרלוונטיים של $node\_parent$ (המצביע על $node$, min, ו - max).
אם $node$ היה השורש של העץ, המתודה מעדכנת את השדה $root$ של $self$ להיות $None$, ואז קוראת למתודות $sever\_ties$ ו - $set\_virtual$ בזו אחד זו בעזרת $node$. לבסוף היא מחזירה את הערך 0.
כמו שניתן לראות, סיבוכיות מקרה זה היא $O(1)$.

לאחר כל הפיצול הזה למקרים, במידה והמתודה לא הגיע למקרה שבו היא מחזירה ערך, המתודה קוראת למתודות $sever\_ties$ ו - $set\_virtual$ בזו אחר זו בעזרת $node$, ולבסוף מחזירה את תוצאת הקריאה למתודה $re\_balance$ על ידי $self$ עם הפרמטרים $node\_parent$, $node\_parent.get\_height()$, ו - $False$.

לכן סיבוכיות המתודה במקרה הגרוע ביותר היא כסיבוכיות המתודה $re\_balance$, שהיא $O(self.root.get\_height()-node.get\_height()+1)$. המקרה הגרוע ביותר הוא כאשר $node$ הוא צומת בעומק של $O(\log_2(n))$, כאשר n הוא מספר הצמתים בעץ. במקרה זה יתקיים:

$self.root.get\_height()-node.get\_height()=O(\log_2(n))$, ולכן סיבוכיות המתודה במקרה הגרוע ביותר היא $O(\log_2(n))$.

- המתודה $avl\_to\_array(self)$ מחזירה רשימה של כל הצמתים בעץ, מסודרים לפי גודל ערכי השדות key שלהם, מהקטן לגדול. היא עושה זאת על ידי קריאה למתודה in_order על ידי $self.root$, פעם אחת עם "L" (ואחריה רשימה בת איבר אחד שהוא $self.root$), ופעם אחת עם "R". סיבוכיות של $O(1)$.
- המתודה $size(self)$ מחזירה 0 אם העץ ריק, אחרת היא מחזירה את ערך השדה $size$ של הצומת אליו מצביע השדה $root$ של $self$. סיבוכיות של $O(1)$.
- המתודה $split(self,node)$ מחזירה רשימה בת 2 איברים, הראשון הוא תת העץ השמאלי של $node$ לפי הצמתים שיש ב - $self$, והשני הוא תת העץ הימני שלו. למתודה זו **קיימת דרישת קדם** לפיה $node$ הוא צומת ב - $self$. המתודה מייצרת שני עצים ריקים בשם $left\_subtree$ ו - $right\_subtree$. המתודה חוזרת על התהליך שתואר בשיעור על פיצול עץ, ובכך מעדכנת את שני העצים שיצרה בכל פעם. לכן, כפי שהוסבר בשיעור, אם נסמן ב – n את מספר הצמתים בעץ, נקבל שסיבוכיות המתודה במקרה הגרוע ביתר (בניתוח נאיבי) היא$O(\log_2^2(n))$. כאשר ננתח זאת יותר לעומק, ונסמן את גבהי העצים שיאוחדו להיות $left\_subtree$ בתור הסדרה $\{Tl_i\}_{i\in\mathbb{N}}$ ואת גבהי העצים שיאוחדו להיות $right\_subtree$ בתור הסדרה $\{Tr_i\}_{i\in\mathbb{N}}$, נקבל קודם כל שעל מנת ש - $self$ אכן יהיה עץ AVL, שתי הסדרות יהיו חייבות להיות מונוטוניות עולות. אם נסמן:
$S_{Tl}=\sum_{i\in\mathbb{N}\setminus(-\infty,1]}\left(\left|Tl_i-height\left(join(Tl_1,\dots,Tl_{i-1})\right)\right|\right)$ וגם:
$S_{Tr}=\sum_{i\in\mathbb{N}\setminus(-\infty,1]}\left(\left|Tr_i-height\left(join(Tr_1,\dots,Tr_{i-1})\right)\right|\right)$, אזי נקבל כי סיבוכיות המתודה תהיה $O(S_{Tl}+S_{Tr})$. כמו כן נקבל כי אם נסמן ב – n את מספר הצמתים ב - $self$ אזי $S_{Tl}=O(\log_2(n))$ וגם $S_{Tr}=O(\log_2(n))$ ולכן סיבוכיות המתודה במקרה הגרוע ביותר היא $O(\log_2(n))$.
- המתודה $join(self,tree2,key,val)$ מאחדת את העצים $self$ ו - $tree2$ בעזרת צומת שערך שדה ה - key שלו הוא ערך הפרמטר key, ושערך שדה ה - $value$ שלו הוא הפרמטר val. למתודה זו **קיימת דרישת קדם** לפיה המפתחות של כל הצמתים ב - $self$ קטנים מהפרמטר key, שקטן מכל המפתחות של כל הצמתים ב - $tree2$. ראשית המתודה בודקת האם אחד העצים ריק, במקרה כזה היא פשוט קוראת למתודה $insert$ על ידי העץ הלא ריק, עם הפרמטרים key ו - val הנתונים, ומחזירה את גובה העץ הלא ריק ועוד 2. אם נסמן $h_{self}=self.root.get\_height()$ וגם:
$h_{tree2}=tree2.root.get\_height()$, נקבל כי סיבוכיות מקרה זה היא $O\left(\max\left(h_{self},h_{tree2}\right)\right)$. נשים לב כי במקרה זה מתקיים $\min\left(h_{self},h_{tree2}\right)=-1$, ולכן ניתן לומר כי סיבוכיות מקרה זה היא בעצם $O\left(\left|h_{self}-h_{tree2}\right|\right)$.

אם שני העצים לא ריקים, המתודה בודקת האם $self$ הוא אכן העץ הגבוה יותר, אם לא היא הופכת את $self$ ואת $tree2$ על ידי שימוש במתודה set_self ובמשתנה זמני בשם tmp. המתודה מייצרת משתנה בשם $result$ ששווה להפרשי הגבהים פלוס 1, ואז יוצרת צומת חדש בשם $node$ שמתאים לפרמטרים שניתנו, וקוראת למתודה $join\_with\_node$ על ידי $self$ עם הפרמטרים $tree2$ ו – $node$, שזו סיבוכיות של:
$O\left(\left|h_{self}-h_{tree2}\right|+1\right)$. בכל אחד משני המקרים המתודה מוודאה להפוך את $self$ לעץ הגדול יותר (אם $self$ היה ריק היא משנה את המצביע $root$ שלו להצביע על

$tree2.root$), ואז משתמשת במתודה set_self בשביל להפוך את $tree2$ להיות $None$ אחרי תהליך האיחוד. לבסוף המתודה מחזירה את $result$. לכן סיבוכיות המתודה הינה $O(|h_{self} - h_{tree2}| + 1)$.

- המתודה $get\_root(self)$ מחזירה את הצומת אליו מצביע השדה $root$ של $self$ כל עוד הוא איננו $None$, שדה ה - $size$ שלו איננו 0, והוא איננו צומת וירטואלי. אחרת היא מחזירה $None$. סיבוכיות של $O(1)$.