אוניברסיטת תל אביב

סמסטר א' תשפ"ה

**מבני נתונים - פרויקט מספר 1 - עץ מאוזן**

מגיש 1: מגיש 2:

שם: **רון שמש** שם: **רועי שלאין**

**הקדמה**

בתרגיל זה שני חלקים:

1. חלק מעשי: מימוש של עץ AVL. עמודים 1-2 במסמך מתארים חלק זה.
2. חלק ניסויי-תאורטי: בהתבסס על המימוש מהחלק המעשי, נבצע מספר ניסויים עם ניתוח תאורטי נלווה. עמודים 3-4 מתארים חלק זה.

**שימו לב:** בסוף המסמך (עמוד 4) ישנן הוראות הגשה – הקפידו לפעול לפיהן. **תאריך הגשה: 5.1.2025**.

חלק מעשי

**דרישות**

בתרגיל זה יש לממש עץ AVL, לפי ההגדרות שניתנו בכיתה. לכל איבר בעץ יש ערך (value) שהוא מחרוזת, ומפתח (key) שהוא מספר שלם. כל המפתחות שונים זה מזה, והסדר על צמתי העץ מתייחס כרגיל אך ורק למפתחות. המימוש יהיה **בשפת python 3.13 וצריך להיות מבוסס על קובץ השלד המופיע באתר הקורס**.   
הפעולות שיש לממש הן:

|  |  |
| --- | --- |
| **פעולה** | **תיאור** |
| search(k) | הפונקציה מחפשת איבר בעל מפתח k. היא מחזירה זוג סדור , כאשר הוא מצביע לצומת המתאים (או None אם לא קיים), ו- הוא אורכו בקשתות של מסלול החיפוש+1. |
| finger\_search(k) | הפונקציה מחפשת איבר בעל מפתח k, החל מהצומת המקסימלי. היא מחזירה זוג סדור , כאשר הוא מצביע לצומת המתאים (או None אם לא קיים), ו- הוא אורכו בקשתות של מסלול החיפוש+1. |
| insert(k, v) | הכנסת איבר בעל ערך v ומפתח k לעץ, ניתן להניח שהמפתח לא קיים כבר בעץ. הפונקציה מחזירה שלשה , כאשר מצביע לצומת שנוצר, מספר הקשתות על מסלול ההכנסה, ו- מספר מקרי שינוי גובה (promote) שנדרשו במהלך האיזון. |
| finger\_insert(k, v) | הכנסת איבר בעל ערך v ומפתח k לעץ החל מהצומת המקסימלי, ניתן להניח שהמפתח לא קיים כבר בעץ. הפונקציה מחזירה שלשה , כאשר מצביע לצומת שנוצר, מספר הקשתות על מסלול ההכנסה, ו- מספר מקרי שינוי גובה (promote) שנדרשו במהלך האיזון. |
| delete(x) | מחיקת הצומת x מהעץ בהינתן מצביע. |
| join(t, k, v) | הפונקציה מקבלת עץ נוסף t שכל המפתחות שלו קטנים, או שכולם גדולים, מהמפתחות של העץ הנוכחי, כאשר המפתח k נמצא ביניהם. על הפונקציה לאחד לעץ הנוכחי את העץ הנוסף והאיבר החדש (k, v). לאחר הפעולה העץ t אינו שמיש, כלומר אסור למשתמש (טסטר) לקרוא לו יותר. |
| split(x) | הפונקציה מקבלת מצביע לצומת x בעץ. עליה לפצל את העץ לשניים ולהחזיר , כך ש- יכיל את המפתחות הקטנים מ-x ו- את הגדולים. לאחר הפעולה העץ הנוכחי וגם אינם שמישים. |
| avl\_to\_array() | הפונקציה מחזירה מערך ממוין (ע״פ המפתחות) של האיברים במילון כאשר כל איבר מיוצג ע״י זוג סדור של (key, value). |
| max\_node() | הפונקציה מחזירה מצביע לצומת בעל המפתח המקסימלי בעץ. |
| size() | הפונקציה מחזירה את מספר האיברים בעץ. |
| get\_root() | הפונקציה מחזירה מצביע לשורש העץ. |

**לצורך מימוש פעולות אלו, ניעזר במחלקה AVLNode המופיעה בקובץ**.

נדרוש שלכל עלה יהיו 2 בנים "וירטואליים", כלומר צמתים ללא מפתח. באופן זה, נוח יותר לממש גלגולים מכיוון שלכל צומת יהיו 2 בנים.

המחלקה AVLNode מכילה את השדות הבאים, שעליכם לתחזק:

key – המפתח של הצומת.

value – הערך של הצומת.

left – הבן השמאלי של הצומת.

right – הבן הימני של הצומת.

parent – ההורה של הצומת.

height – גובה הצומת.

בנוסף, המחלקה תומכת בפעולה הבאה:

is\_real\_node – מחזירה TRUE אם הצומת מייצג צומת אמיתי בעץ (קרי: צומת שאינו וירטואלי).

**מספר הבהרות והסברים:**

1. **בפעולות החיפוש אנו סופרים את אורכו בקשתות של המסלול בעץ בין הצומת שבו התחיל החיפוש והצומת שבו הסתיים החיפוש, ומוסיפים לתוצאה 1.**
2. **בפעולות ההכנסה סופרים את אורכו בקשתות של המסלול בעץ בין הצומת שבו התחיל החיפוש והצומת שהכנסנו מייד לאחר ההכנסה, לפני פעולות איזון. באופן שקול, זהה לאורך המסלול להורה שלו +1, ולכן אם מחפשים מפתח שלא קיים בעץ לפני שמכניסים אותו מקבלים תוצאה זהה.  
   לדוגמה, אם משתמשים בפעולה** finger\_search כדי להכניס צומת מקסימלי חדש, מקבלים .
3. **עבור פעולות ההכנסה, כאשר מפעילים את אלגוריתם האיזון כפי שנלמד בכיתה, סופרים אך ורק כמה פעמים היינו במקרה 1** (promote) **של שינוי שדה גובה, ומתעלמים מגלגולים.  
   לדוגמה, נתבונן בסדרת הכנסה של 3, ואז 1, ואז 2. בהכנסה הראשונה אין פעולות איזון. בהכנסה של 1 מעדכנים את הגובה של 3 (מקרה 1), פעולה אחת. בהכנסה של 2 מעדכנים את הגובה של 1 (מקרה 1) ולאחר מכן בשורש 3 אנחנו נמצאים במקרה 3 של גלגול כפול אותו לא סופרים, לכן שוב מחזירים 1.**
4. **חיפוש החל מהמקסימום: עולים מהמקסימום עד הצומת הראשון בעל מפתח קטן יותר (או השורש) וממנו יורדים כרגיל. (חיפוש זה יעיל יותר רק עבור מפתחות גדולים שקרובים למקסימום.)**

**הערות חשובות:**

1. **המימוש יבוצע על ידי מילוי קובץ השלד. מותר להחליף את תוכן הפונקציות הקיימות ולהוסיף פונקציות ושדות חדשים.** אסור לשנות את חתימות הפונקציות הקיימות ואת שמות השדות הקיימים כדי לא לפגוע בטסטר (כן מותר להוסיף פרמטרים עם ערך ברירת מחדל). על כל הפונקציות/מחלקות להופיע בקובץ יחיד.
2. **אין להשתמש באף מימוש ספרייה של מבנה נתונים.**
3. עליכם לממש את כל הפעולות בסיבוכיות המיטבית.

**סיבוכיות**

יש לציין בקוד ולהסביר בקצרה במסמך התיעוד את סיבוכיות זמן הריצה במקרה הגרוע (האסימפטוטית, במונחי O הדוקים) של כל פונקציה **שמכילה לולאות/רקורסיה**, כתלות במספר האיברים בעץ n.

**פלט**

אין צורך בפלט למשתמש.

**תיעוד**

בנוסף לבדיקות אוטומטיות של הקוד שיוגש, קובץ המקור ייבדק גם באופן ידני. חשוב להקפיד על תיעוד לכל פונקציה, וכמות סבירה של הערות. **הקוד צריך להיות קריא**, בפרט הקפידו על בחירת שמות משתנים ועל אורך השורות.

יש להגיש בנוסף לקוד גם מסמך תיעוד חיצוני. המסמך יכלול את תיאור המחלקה שמומשה, ואת תפקידו של כל חבר במחלקה. עבור כל פונקציה במחלקה יש לפרט מה היא עושה, כיצד היא פועלת ומה סיבוכיות זמן הריצה שלה. בפרט, אם פונקציה קוראת לפונקציית עזר, **יש** להתייחס גם לפונקציית העזר בניתוח. עבור פונקציות שעולות זמן קבוע יספיק תיאור קצר ולא לפרט את ניתוח הסיבוכיות.

**בדיקות**

התרגילים ייבדקו באמצעות תוכנת טסטר שקוראת לפונקציות המפורטות מעלה בתרחישים שונים, ומוודאת את נכונות התוצאות. קובץ הטסטר שלנו **לא יפורסם** לפני הבדיקות. **מומלץ מאוד לממש אוסף בדיקות עבור המימוש**, לא בשביל ההגשה, אלא כדי לבדוק שהקוד לא רק רץ, אלא גם נכון!

בקובץ שתגישו **לא תהיה פונקציית main ולא יהיו הרצות קוד/הדפסות**, דבר זה יפגע בטסטר שיבדוק לכם את התרגילים. אין צורך להגיש את הקוד הנוסף שכתבתם לחלק הניסויי.

חלק ניסויי/תאורטי

בשאלה זו נדון ב insertion-sort באמצעות AVL Finger Tree. המיון מתבצע באופן הבא: מכניסים את האיברים לפי הסדר (הלא ממוין) אל העץ, כאשר החיפוש בהכנסת כל איבר חדש מתחיל מהמקסימום הנוכחי, ובסיום מבצעים סריקת in-order לקבלת הסדר הממוין. עבור עלות בניית העץ, ננתח בנפרד את עלות החיפושים ואת מספר פעולות האיזון.

* לצורך הניתוח, נמיין מערכים בגדלים שונים. גודל המערך שנמיין יהיה כאשר , ואיבריו יהיו הטבעיים עד . למשל, עבור המערך בגודל , ועבור המערך בגודל .
* לכל גודל של מערך, נבצע 4 ניסויים נפרדים:
  + בניסוי הראשון נמיין מערך **ממוין**, מקטן לגדול.
  + בניסוי השני נמיין מערך **ממוין הפוך**, מגדול לקטן.
  + בניסוי השלישי סדר האיברים במערך יהיה **אקראי**.
  + בניסוי הרביעי ניקח מערך ממוין ועבור כל אינדקס פרט לאחרון , נבצע החלפה עם האיבר הבא בסיכוי חצי.

הערה: בסעיפים הבאים, עבור ניסויים אקראיים, יש לקחת את הממוצע על פני 20 ניסויים.

1. יש למלא בטבלה הבאה את סך עלויות האיזון **ללא גלגולים** עבור כל אחד מהניסויים. הסבירו מהו החסם העליון התאורטי על סך עלויות האיזון כולל גלגולים, והאם הערכים בטבלה מתאימים. לסיום, נמקו מדוע תוספת הגלגולים לספירה אינה משנה אסימפטוטית.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי i | עלות איזון במערך ממוין | עלות איזון במערך ממוין-הפוך | עלות איזון במערך מסודר אקראית | עלות איזון במערך עם היפוכים סמוכים אקראיים |
| 1 | 430 | 430 | 379 | 422 |
| 2 | 873 | 873 | 775 | 846 |
| 3 | 1760 | 1760 | 1567 | 1739 |
| 4 | 3535 | 3535 | 3182 | 3525 |
| 5 | 7086 | 7086 | 6305 | 7015 |
| 6 | 14189 | 14189 | 12753 | 14032 |
| 7 | 28396 | 28396 | 25490 | 28163 |
| 8 | 56811 | 56811 | 50705 | 56209 |
| 9 | 113642 | 113642 | 101365 | 112344 |
| 10 | 227305 | 227305 | 202869 | 224382 |

אנו מצפים כי סך עלויות האיזון יהיו בסיבוכיות O(nlog(n)) מכיוון ש insert מתבצע בסיבוכיות לוגריתמית ואנו מבצעים n הכנסות שונות בכל פעם. בנוסף, יש לשים לב כי עלויות האיזון עם או בלי רוטציה הן לוגריתמיות ולכן גם תוצאה שמחסרים ממנה את עלויות הרוטציות עדיין התוצאה תהיה זהה.

ניתן לראות כי עבור כל העמודות אנו רואים כי בין כל שורה לשורה הזמן הוא פחות או יותר כפול 2 כמצופה שכן אנו מגדילים את המערך פי 2, ככל ש n גדל ההשפעה הלוגריתמית קטנה וההשפעה הלינארית נשארת קבועה – 2. לכן, ניתן לראות כי השערתנו לגבי השפעת הרוטציות הייתה נכונה.

1. בהינתן מערך בגודל נגדיר היפוך בתור זוג אינדקסים כך שמתקיים , ונסמן את מספר ההיפוכים הכולל במערך ב-. ניתן לשים לב כי באופן כללי וככל שיש פחות היפוכים כך המערך קרוב יותר לממוין. יש למלא בטבלה הבאה את מספר ההיפוכים במערך הקלט עבור כל אחד מהניסויים (מספיק עד ).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי i | מספר היפוכים במערך ממוין | מספר היפוכים במערך ממוין-הפוך | מספר היפוכים במערך מסודר אקראית | מספר היפוכים במערך עם היפוכים סמוכים אקראיים |
| 1 | 0 | 24531 | 13833 | 101 |
| 2 | 0 | 98346 | 48465 | 228 |
| 3 | 0 | 393828 | 196857 | 467 |
| 4 | 0 | 1576200 | 795168 | 857 |
| 5 | 0 | 6306576 | 3123967 | 1787 |

1. יש למלא בטבלה הבאה את סך עלויות החיפוש עבור כל אחד מהניסויים.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי i | עלות חיפוש במערך ממוין | עלות חיפוש במערך ממוין-הפוך | עלות חיפוש במערך מסודר אקראית | עלות חיפוש במערך עם היפוכים סמוכים אקראיים |
| 1 | 221 | 2708 | 2043 | 313 |
| 2 | 443 | 6288 | 4709 | 640 |
| 3 | 887 | 14334 | 10874 | 1294 |
| 4 | 1775 | 32200 | 25726 | 2508 |
| 5 | 3551 | 71482 | 61217 | 5071 |
| 6 | 7103 | 157148 | 138166 | 10145 |
| 7 | 14207 | 342686 | 292742 | 20265 |
| 8 | 28415 | 742176 | 670918 | 40618 |
| 9 | 56831 | 1597986 | 1439727 | 81214 |
| 10 | 113663 | 3423268 | 3018953 | 162346 |

1. כדי לחסום מלמעלה את סך עלויות החיפוש באופן תאורטי, נסמן ב- את מספר האיברים לפני האיבר באינדקס שגדולים ממנו.
   1. הסבירו מדוע .

הסבר I

כל היפוך במערך הוא יחס בין איבר לכל אחד מהאיברים שלפניו בעלי ערך גדול מהאיבר הנוכחי, לכן כל היפוך לא יכול להיספר יותר מפעם 1 שכן אנו בודקים כל זוג סדור של איברי המערך פעם אחת

* 1. חסמו את עלות החיפוש בעת הכנסת האיבר ה- כפונקציה של , והסיקו כי סך עלויות החיפוש לאורך סדרת ההכנסות הוא .  
     הערה: כאשר הסיבוכיות היא עבור , ניתן להמיר לביטוי כדי לעבור לביטוי אחד פשוט וחוקי שחוסם את שניהם מלמעלה.

הסבר II

ננתח את עלות החיפוש של מספר. כאשר אנו מחפשים את המיקום להכנסת איבר החל מהאיבר המקסימלי, במקרה הטוב ביותר אנו עלות החיפוש תהיה 1 אם נכניס איבר מקסימלי, ובמקרה הגרוע ביותר נעלה עד השורש ונרד עד האיבר המינימלי בעלות של O(log(n)), לכן בסה"כ החיפוש יהיה מסיבוכיות , נשתמש בהכוונה ונבטא בעזרת . כעת, נחשב את סכום עלויות החיפושים, ובעזרת חוקי לוגריתמים נראה כי:

*כנדרש.*

* 1. השתמשו באי-שוויון הממוצעים כדי לחסום מלמעלה את סך עלויות החיפוש כפונקציה של . הסבירו מדוע זוהי העלות הדומיננטית מבחינה אסימפטוטית.

הסבר III

נסמן , נשתמש בחסם עלויות החיפוש שקיבלנו בסעיף קודם.

נמצא חסם עליון, ע״י חסימת המקרה הגרוע ביותר – כאשר מס׳ ההיפוכים הוא מקסימלי, במערך ממוין הפוך.

במקרה זה, מתקיים כי לכל איבר במערך, כל האיברים שנמצאים לפניו הם היפוכים חוקיים, ולכן .

מכאן, כי מהחסם שקיבלנו בסעיף קודם מתקיים:

**מעברים:**

מעבר שני : אי שוויון הממוצעים

מעבר שלישי: לפי חוקי לוגריתמים

מעבר רביעי: הוספת 2 ל i זניחה בניתוח big o.

מעבר אחרון: חוקי לוג' + הוספת 2 כדי להימנע מעלות 0.

* 1. השוו בין החסם שהתקבל ותוצאות הניסויים.

הסבר IV

ניתן לראות בתוצאות הניסוי כי עבור מערך ממוין הפוך בו ב עלות החיפוש שקיבלנו היא בסיבוכיות O(nlog(I)), עבור מערך ממוין שבו I=0, עלות החיפוש היא n בדיוק. בתוצאות הניסוי שקיבלנו ניתן גם לראות כי עבור מערך ממוין כמעט התוצאות היו בסדרי גודל דומים לשל מערך ממוין. לעומת זאת עבור מערך אקראי נראה כי התוצאות ינועו בין החסמים בהתאם לכמות ההיפוכים I שהתקבלו במערך האקראי.

הוראות הגשה

הגשת התרגיל תתבצע באופן אלקטרוני באתר הקורס במודל.

**הגשת התרגיל היא בזוגות בלבד!**

כל זוג יבחר **נציג/ה** ויעלה **רק** תחת שם המשתמש של הנציג/ה את קבצי התרגיל (תחת קובץ **zip**) למודל.

על ההגשה לכלול שלושה קבצים:

1. קובץ המקור (הרחבה של קובץ השלד שניתן) תחת השם **AVLTree.py**.
2. קובץ טקסט **info.txt** המכיל את פרטי הזוג **באנגלית:** מספר ת"ז, שמות, ושמות משתמש.
3. מסמך תיעוד חיצוני, המכיל גם את תוצאות המדידות. את המסמך יש להגיש בפורמט **pdf**.

שמות קובץ התיעוד וקובץ הzip צריכים לכלול את שמות המשתמש האוניברסיטאיים של **הזוג המגיש** לפי הפורמט **AVLTree\_username1\_username2**.pdf/zip, בתוכן הקבצים יש לציין את שמות המשתמש, תעודות הזהות ושמות המגישים (בכותרת המסמך ובשורת הערה בקובץ המקור).

הגשת שיעורי הבית באיחור - באישור מראש בלבד. הגשה באיחור ללא אישור תגרור הורדת נקודות מהציון.

**הגשת התרגיל היא חובה לשם קבלת ציון בקורס**.

**בהצלחה!**

תרגיל מעשי 1 – קורס מבנה נתונים

חלק א' קובץ תיעוד

1. search\_parent

תיאור הפונקציה: **הפונקציה מקבלת את העץ (**self**) ואת הערך לחיפוש. החל מהשורש לפי השוואת ערכי ה**KEY **הפונקציה מתקדמת בעץ עד למציאה או הגעה לסוף העץ. לבסוף תחזיר את האב של הצומת המתאימה ואת כמות הצעדים שבוצעו עד האב.**

סיבוכיות: **לכל היותר נבצע איטרציות כעומק העץ. עבור** n **איברים עומקו של עץ** AVL **יהיה** log(n) **ולכן הסיבוכיות במקרה הגרוע תהיה** O(logn).

1. Search

תיאור הפונקציה: **הפונקציה מקבלת את העץ self ואת הערך לחיפוש key. הפונקציה משתמשת ב search\_parent לטובת החיפוש ומחזירה בהתאם ליחס בין המפתח של האב המוחזר ובין המפתח שקיבלנו לחפש את הצומת הנכון. מכיוון שנעשה שימוש ב search\_parent מאותם מניעים גם הסיבוכיות של פונקציה זו תהיה O(log(n)).**

1. Finger\_Search

תיאור הפונקציה: **הפונקציה תקבל את העץ (self) והערך לחיפוש. הפונקציה תתחיל מהאיבר הימני התחתון ביותר בעץ על ידי שליפת הערך max, נפריד את החיפוש לשני שלבים: עלייה וירידה. כל עוד המפתח של הצומת הנוכחי קטן מהמפתח המבוקש נמשיך את החיפוש בהורה, אם המפתח שווה, נחזיר את הצומת ואם המפתח קטן או הגענו לשורש, נבצע search מהצומת הנוכחי.**

סיבוכיות: **נחשב עבור המקרה הגרוע ביותר, בו נרצה לחפש עלה בצד השמאלי של העץ. במקרה זה יש לנו O(log(n)) עליות עד השורש בתוף הפונקציה, ומשם עוד O(log(n)) ירידות** **מתוך הקריאה לפונקציה search. לכן בסך הכל: O(finger\_search)=O(log(n)+log(n)) = O(log(n))**

1. Perform\_insert

תיאור הפונקציה: **הפונקציה מקבלת את** העץ **(self)** **וכן את המפתח והערך שקיבלנו להכנסה צומת החדש ובנוסף תקבל מצביע לאבא של הצומת שנרצה להכניס. הצומת תבדוק באיזה מיקום יש להכניס את הצומת כילד של המצביע ולאחר מכן תבצע לולאה שמטרתה לבדוק שלאחר פעולת ההכנסה העץ מאוזן והגבהים מעודכנים. בתוך הלולאה, בהתאם למקרים שנלמדו בכיתה הפונקציה תבחר אם לעדכן גובה או לבצע רוטציה באמצעות הפונקציה rebalance.**

סיבוכיות: **ההשמה של הצומת תתבצע ב O(1) לאחר מכן העלייה עד השורש במקרה של n איברים תהיה מסדר גודל של O(log(n)). האיזונים יעלו O(1) בכל פעם ובסך הכל יתבצעו לכל היותר O(log(n)) פעמים ולכן הפונקציה מסדר גודל:**

**O(perform\_insert) = O(log(n)+1\*log(n)) = O(log(n))**

**תת פונקציות: rotate\_left/rotate\_right- מבצעות רוטציה לימין או רוטציה לשמאל בהתאם למקרים השונים כפי שנלמד בכיתה בסיבוכיות O(1)**.

1. Rebalance

תיאור הפונקציה: הפונקציה מקבלת את העץ **(self)** וכן את השורש של תת העץ עליו צריך לבצע רוטציה, הפונקציה תבדוק באיזו רוטציה מדובר בהתאם להפרשי הגבהים של צאצאיו, ותקרא לפונקציות **rotate\_left/rotate\_right** בהתאם בסיבוכיות **O(1).**

סיבוכיות: **הפונקציה מבצעת מספר פעולות שלא תלוי בקלט וכן קוראת פעם יחידה לפונקציה שמבצעת פעולות ללא תלות בקלט ולכן הפונקציה מסיבוכיות O(1).**

1. Insert

תיאור הפונקציה: **הפונקציה תקבל את העץ (self) יחד עם מפתח וערך להכניס באיבר חדש. ראשית הפונקציה תחפש את המקום המתאים באמצעות search\_parent בסיבוכיות O(logn). לאחר מכן היא תקרא לפונקציה perform\_insert. לאחר מכן נשתמש בפונקציית search\_parent** **על מנת למצוא את** e **בסיבוכיות O(logn**)**.**

סיבוכיות: **הסיבוכיות תהיה** O(logn)**. נסכום את כלל ההפעלות של פונקציות העזר לפונקצייה ונקבל:**

**O(insert) = O(logn) + O(logn) + O(logn)= O(logn).**

1. Finger\_insert

תיאור הפונקציה: **הפונקציה תקבל את העץ (**self**) יחד עם מפתח וערך להכניס באיבר חדש. בדומה ל**insert **תמצא את המקום המתאים בחיפוש החל מהאיבר המקסימלי בעזרת search\_parent בסיבוכיות log(n). תבצע הכנסה בעזרת perform insert באופן דומה ל**insert **תמצא את** e **המתאים הפעם בעזרת finger\_search.**

סיבוכיות: **הסיבוכיות תהיה O(logn). נסכום את כלל ההפעלות של פונקציות העזר לפונקצייה ונקבל:**

**O(Finger\_insert) = O(logn) + O(logn) + O(logn) = O(logn).**

1. Delete

תיאור הפונקציה: **הפונקציה תקבל את העץ (self) ואת האיבר הנדרש למחיקה. הפונקציה תפריד למקרים השונים, אם צריך תבצע החלפה עם הצומת בעלת המפתח הבא או הקודם בתור. לאיבר זה יהיה לכל היותר ילד אחד ולכן המחיקה קלה יותר. לאחר המחיקה הפונקציה תבצע עדכון גבהים ואיזון עד השורש בהתאם למקרים השונים.**

**הסבר פונקציות עזר:**

**to\_replace: פונקציה שבודקת אם צריך להחליף את הצומת עם האיבר הבא או הקודם**

**Successor/predecessor:** הפונקציות שמוצאות איבר קודם או הבא בסיבוכיות **O(log(n))**

סיבוכיות: **הסיבוכיות תהיה O(logn). במקרה הגרוע ביותר מציאת צומת מחליף תהיה בסיבוכיות O(log(n)), לאחר מכן ביצוע ההחלפה והמחיקה יהיה מסדר O(1). בתוך הלולאה שאחראית על העדכונים, בכל איטרציה נבצע מספר סופי של פעולות שלא תלוי בגודל העצם, כולל הרוטציות. לכן בסך הכל נקבל כי:**

**O(delete) = O(1)+O(log(n))+O(1)\*O(log(n)) = O(log(n))**

1. Join

תיאור הפונקציה: **הפונקציה ראשית בודקת האם אחד העצים ריק ומגיבה בהתאם. בהנחה ושני העצים אינם ריקים היא משווה את הגבהים ומטפלת במקרה "הקל" בו הפרש הגבהים מאפשר השמה פשוטה של הצומת. במקרה בו אין אפשרות כזו, הפונקציה תמצא בעץ הגדול יותר את הצומת המתאים להכנסה ותבצע את פעולת האיחוד. לאחר מכן הפונקציה תרוץ עד השורש ותבצע עדכוני גבהים או רוטציות אם צריך**

סיבוכיות: **הסיבוכיות תהיה O(log(n)). במקרה הגרוע התנאים הראשונים יהיו בסיבוכיות O(1), פעולת החיפוש תהיה בסיבוכיות O(log(n)), פעולת ההכנסה תהיה בסיבוכיות O(1) והריצה עד השורש תהיה בסיבוכיות O(log(n)) ובכל איטרציה תבצע מספר סופי של פעולות. לכן נקבל:**

**O(join) = O(1) + O(log(n)) + O(1)\*O(log(n)) = O(log(n))**

1. split

תיאור הפונקציה: הפונקציה מקבלת מצביע לאיבר בעץ ותפצל את העץ לעץ בעל האיברים הגדולים מהמצביע ועץ של הקטנים. הפונקציה תטפל במקרה של עץ ריק. אחרת, באמצעות לולאת while תבצע ריצה במעלה העץ כאשר עבור תתי עצים וצמתים גדולים מהמפתח הרלוונטי תשתמש בפעולת **join ובפעולת insert** על מנת ליצור עץ של הערכים הגדולים ובאופן דומה עבור עץ הערכים הקטנים (תתי עצים שמאליים). לבסוף תחזיר את שני העצים.

סיבוכיות: **הפונקציה תרוץ במעלה העץ עד השורש ותבצע join בכל שלב כעלות ההפרש בין הגבהים ו insert כעלות העומק של העץ החדש.**

**כפי שהוכח בכיתה העלות הכוללת של פעולה זו היא** **O(log(n)) על אף שלמראית עין ניכר כי**

1. Avl\_to\_array

תיאור הפונקציה: **הפונקציה מקבלת את העץ ומחזירה list שמכיל את כל איברי העץ מסודרים ב in-order**

סיבוכיות: **הפונקציה תעבור בכל האיברים בדיוק פעם אחת ופועלת** O(1) **בכל איבר ולכן הסיבוכיות תהיה** O(n)**.**

1. Max\_node

תיאור הפונקציה: **הפונקציה תחזיר בכל פעם את השדה של האיבר המקסימלי (מתוחזק בפעולות** insert, delete, split, join**).**

סיבוכיות: O(1)

1. size

תיאור הפונקציה: **הפונקציה מחזירה את השדה** size **של העץ. השדה מתוחזק בכלל הפעולות הרלוונטיות מלבד** split **(לפי הנחיה).**

סיבוכיות: **השדה מתוחזק בכל הפעולות בעץ ולכן** O(1)**.**

1. Get\_root

תיאור הפונקציה: **מקבלת את** self **ומחזירה את השורש שלו.**

סיבוכיות: O(1)