אמיתי וולף – Amitaiwolf - 325541878
אייבי כץ – eivikatz - 214208704

תיעוד סיבוכיות זמן הריצה

Class AVLNode

__init__(self, val)

* הפעולה מקבלת ערך val ומייצרת צומת עם אותו ערך.

* הפונקציה מאתחלת לכל צומת שני צמתים וירטואליים באופן רקורסיבי, אבל מפני שהם וירטואליים הרקורסיה מפסיקה מיד. עבור צמתים לא וירטואליים הפעולה מאתחלת את השדות – left, right, value, parent, height, size. במידה ו-val הוא None, אנחנו מתייחסים אליו כאל צומת וירטואלי ומאתחלים רק את השדות value ו-parent.

* הפעולה היא רקורסיבית אבל מתבצעות סך הכול שתי קריאות רקורסיביות שכל אחת מהן היא O(1), ולכן סה"כ סיבוכיות זמן הריצה היא O(1).

getLeft(self) – הפעולה מחזירה את הבן השמאלי אם אינו וירטואלי, אם וירטואלי תחזיר None. על מנת לגשת לבן וירטואלי יש לגשת לשדה left באופן ישיר, כי הפעולה getLeft תחזיר None במקום את הבן.

getRight(self) – בדומה ל-getLeft(self).

update(self) – הפעולה מעדכנת את השדות height, size של צומת ומחזירה 0 אם הגובה לא השתנה ו-1 אחרת. הפעולה עובדת ב-O(1).

join(self, left, right)

* הפעולה במחלקת AVLNode מקבלת שני עצים – left, right, ומאחדת אותם לעץ אחד בעזרת הצומת שאיתו קוראים לפעולה. הפעולה דואגת לשמור על איזון העץ ומחזירה את השורש של העץ החדש.

* הפעולה מתפצלת לשלושה מקרים לפי הפרש הגבהים בין העצים, ומבצעת את האיחוד בעזרת הפעולות joinEq, joinRL, joinLR. לאחר פעולת האיחוד הפעולה דואגת לעלות מ-self עד לשורש של העץ החדש ולעדכן את הצמתים שבמסלול בעזרת update.

בכל אחד משלושת המקרים, הפעולות רצות בסיבוכיות O(h) כאשר h הוא הפרש הגבהים בין העצים.

(נציין כי joinEq היא O(1) מכיוון שהפרש הגבהים בין העצים הוא אכן לכל היותר 1).

* לאחר האיחוד, הפעולה עולה עד לשורש כאשר כל איטרציה לוקחת O(1), וכמות האיטרציות היא המסלול בין self לשורש העץ החדש. אורך מסלול זה הוא h גם כן, ולכן במקרה הגרוע ביותר הלולאה תרוץ ב-O(h). מכאן, סה"כ הפעולה רצה ב-O(h) כאשר h הוא הפרש הגבהים בין העצים.

joinEq(self, left, right) – בהנחה שהפרש הגבהים של left ו-right הוא לכל היותר 1, הפעולה מאחדת בין העצים בצורה נאיבית. כאשר left הוא הבן השמאלי, right הוא הבן הימני, ו-self הוא השורש. העץ החדש הוא כמובן מאוזן. הפעולה עובדת ב-O(1).

joinLR(self, left, right) – בהנחה שגובה העץ השמאלי קטן מגובה העץ הימני, הפעולה מצרפת את העץ השמאלי ל-self ולאחר מכן רצה על הענף השמאלי של העץ הימני עד שהיא מגיעה לצומת בגובה קטן או שווה לגובה העץ השמאלי. באותו צומת הפעולה מאחדת בהתאם לאלגוריתם שראינו בהרצאה.

מתבצעת ריצה על הענף השמאלי של העץ הימני, ולכן סיבוכיות זמן הריצה במקרה הגרוע היא O(h) כאשר h הוא הפרש הגבהים בין left ו-right.

joinRL(self, left, right) – בדומה ל-joinLR רק שבמקרה זה גובה העץ הימני קטן מגובה העץ השמאלי.

toTreeList(self)

* הפעולה מחזירה את תת העץ שמושרש בצומת self בתור אובייקט AVLTreeList.

* הפעולה מאתחלת AVLTreeList ריק, ולאחר מכן מגדירה את השורש של אותה רשימה להיות self. הפעולה מחפשת את head ו-tail (first ו-last) בעזרת איטרציות על הענפים השמאלי והימני בהתאמה.

* כל לולאה רצה לכל היותר בגובה העץ ולכן סיבוכיות זמן הריצה של הפעולה היא O(h) עבור h גובה העץ.

<u>Class AVLTreeList</u>

balance(self, node, roll)

* הפעולה מקבלת צומת node ומבצעת את המסלול החל ממנו ועד לשורש העץ תוך עדכון הצמתים וגלגול צמתים לא מאוזנים. הפעולה מקבלת ארגומנט roll שמוגדר דיפולטית להיות True. הפעולה מגלגלת צמתים אם ורק אם הפרמטר roll הוא True. הפעולה מחזירה את מספר פעולות האיזון שנדרשו בדרך.

* בכל איטרציה הפעולה מבצעת עדכון וסוכמת את הפרשי הגבהים, מפני שבאיזון עץ השינוי בגובה יכול להיות לכל היותר 1 בערך מוחלט, הסכימה מונה את מספר הצמתים שהגובה שלהם השתנה. לאחר מכן אם צומת מסוים הוא לא מאוזן, מתבצעת קריאה ל-rotate.

* סיבוכיות זמן הריצה של הפעולה היא ככמות האיטרציות שלה כי כל איטרציה היא O(1) במקרה הגרוע, ולכן הסיבוכיות של הפעולה היא O(d) עבור d עומק הצומת node.

rotate(self, node) – בהינתן צומת לא מאוזן, הפעולה מבצעת פעולות גלגול על מנת לאזן אותו. הפעולה מחזירה את מספר פעולות הגלגול שהתבצעו. הפעולה מסווגת את הצומת הבעייתי לאחד מסוגי הגלגול שלמדנו (גלגול ימינה, גלגול שמאלה, גלגול שמאלה ואז ימינה, וגלגול ימינה ואז שמאלה) בהתאם לערך ה-balance factor שלו ושל בנו הישיר.

כל אחד מארבעת הגלגולים רץ ב-O(1), ולכן גם rotate רצה ב-O(1).

rotateR(self, node) – הפעולה מקבלת צומת עם balance factor 2 כאשר בנו השמאלי הוא עם balance factor 1 או 0, ומבצעת גלגול ימינה כפי שנלמד בהרצאה. הפעולה משנה מצביעים ולכן סיבוכיות זמן הריצה שלה היא O(1).

rotateL(self, node) – בדומה ל-rotateR, רק גלגול שמאלה.

rotateLR(self, node) – הפעולה מקבלת צומת עם balance factor 2 ובן שמאלי עם balance factor -1, ומבצעת גלגול כפי שנלמד בהרצאה. הפעולה משנה מצביעים ולכן סיבוכיות זמן הריצה שלה היא O(1).

rotateRL(self, node) – בדומה ל-rotateLR.

retrieve(self, i) – הפעולה מקבלת אינדקס i ומחזירה את הערך של הצומת במקום ה-i+1. הפעולה משתמשת ב-select ולכן יש להן את אותה סיבוכיות. Select רצה ב-O(logn) ולכן גם retrieve תרוץ ב-O(logn).

insert(self, i, val)

* הפעולה מכניסה צומת עם ערך val במקום ה-i בעץ ומחזירה את מספר פעולות האיזון שהתבצעו.

* הפעולה מאתחלת צומת עם הערך val. אם ההכנסה מתבצעת לסוף הרשימה, הפעולה מגדירה את הצומת החדש להיות הבן הימני של הצומת האחרון. אחרת, הפעולה משתמשת ב-select כדי למצוא את הצומת באינדקס i. אם לצומת זה אין בן שמאלי, הפעולה תכניס את הצומת החדש להיות הבן השמאלי שלו. אחרת, נלך ל-predecessor שלו ונגדיר את הצומת החדש להיות הבן הימני שלו. לבסוף, הפעולה קוראת ל-balance כדי לאזן ולסכום את מספר פעולות האיזון שנדרשו.

* הפעולה משתמשת ב-select, predecessor, balance כאשר כל אחת מהן רצות ב-O(logn), ולכן הסיבוכיות של insert היא O(logn). בפעולה לא השתמשנו בלולאות או ברקורסיה באופן מפורש, הסיבוכיות של הפעולה מגיעה מהקריאה לפעולות העזר.

delete(self, i)

* הפעולה מקבלת אינדקס i ומוחקת את האיבר ה-i בעץ ומחזירה את מספר פעולות האיזון שהיא ביצעה.

* הפעולה מוצאת בעזרת select מצביע לאיבר במקום ה-i ומסירה אותו מהעץ בעזרת remove ומחזירה את מספר פעולות האיזון ש-remove עשתה.

* הפעולה קוראת ל-select, remove, כאשר כל אחת מהן רצה ב-O(logn), ולכן גם delete תרוץ ב-O(logn).

remove(self, node, roll)

* הפעולה מקבלת צומת ואת הארגומנט רול. הפעולה מסירה את node ומחזירה את מספר פעולות האיזון שהיא מבצעת.

* הפעולות מחלקת למקרים. אם לצומת אין בן שמאלי הפעולה עוקפת צומת זה וקובעת שהבן של ההורה שלו יהיה הבן הימני של הצומת. אם לצומת אין בן ימני, הפעולה תעבוד באופן זהה בצורה סימטרית. אחרת, לצומת יש שני בנים, הפעולה מוצאת את ה-successor של הצומת, מסירה אותו בעזרת remove ושמה אותו במקום של node תוך שמירה על מצביעים תקינים. (נשים לב כי במקרה זה ל-successor אין שני בנים ובפרט אין לו בן שמאלי ולכן מובטח שהוא יכנס למקרה של הבן השמאלי והרקורסיה תעצור). בסוף כל אחד מהמקרים מתבצעת קריאה ל-balance ששומרת על איזון העץ. פרט למקרה האחרון שבו האיזון נעשה בעת הסרת ה-successor.

במידה והצומת נמצא בקצוות הרשימה, אנחנו מחליפים אותו ב-head או ב-tail המתאימים בעזרת קריאה ל-successor או predecessor ולכן הפעולה תרוץ בסיבוכיות של אותן פעולות.

במקרה אחר, מתבצעת קריאה ל-balance, ולכן סיבוכיות זמן הריצה של הפעולה היא O(logn).

במקרה האחרון אנחנו מבצעים קריאה ל-successor, בסיבוכיות O(logn). לאחר מכן מבצעים קריאה רקורסיבית ל-remove אשר מביאה אותנו למקרה הראשון, שהסיבוכיות שלו היא O(logn).

removeTail(self) – הפעולה מסירה את הצומת האחרון ברשימה (tail).

במקרה אחד הצומת הוא השורש, והכול רץ ב-O(1). במקרה השני מתבצעת קריאה ל-predecessor בסיבוכיות O(logn) ולסיום מתבצעת קריאה ל-balane עם ההורה של הצומת שמחקנו. הסיבוכיות במקרה זה היא O(logn).

first(self) – מחזיר את הערך של הצומת הראשון. אנחנו שומרים מצביעים לצומת זה, ולכן הפעולה רצה ב-O(1).

last(self) – מחזיר את הערך של הצומת האחרון. אנחנו שומרים מצביע לצומת זה, ולכן הפעולה רצה ב-O(1).

listToArray(self) – הפעולה מחזירה את רשימת הערכים ששמורים בעץ.

לשם כך, אנחנו פונים לאיבר הראשון, ולאחר מכן מבצעים n פעולות successor. מטענה מהתרגול, ביצוע n פעולות successor לוקח O(n).

split(self, i)

* הפעולה מפצלת את העץ לפי הצומת במקום ה-i לשני עצים מאוזנים כאשר אחד העצים מכיל את כל האיברים לפני האיבר ה-i, והעץ השני מכיל את כל האיברים אחרי האיבר ה-i.

* הפעולה קוראת ל-select על מנת לקבל מצביע לאיבר ה-i. לאחר מכן הפעולה רצה מהצומת עד השורש ובכל איטרציה מתבצע איחוד כפי שנלמד בהרצאה בעזרת join. הפעולה לבסוף משתמשת ב-toTreeList אשר מייצרת מהצמתים המקוריים את העצים.

* הקריאה ל-select לוקחת O(logn), בכל איטרציה של הלולאה אנחנו מבצעים קריאה ל-join אשר מומשה כפי שנלמד בהרצאה, ולכן הסיבוכיות שלה היא O(logn).

concat(self, lst)

* הפעולה מקבלת עץ נוסף lst ומשרשרת לסוף העץ הנוכחי את העץ lst. הפעולה מחזירה את הערך המוחלט של הפרש הגבהים בין העצים.

* הפעולה ניגשת לאיבר האחרון בעץ ומסירה אותו בעזרת removeTail ומבצעת פעולת join בין השורש של self לבין השורש של lst בעזרת ה-tail שנמחק. לבסוף הפעולה מחזירה את הפרש הגבהים בין העצים.

* סיבוכיות זמן הריצה של הפעולה במקרה הגרוע היא O(logn) כי עבור join הסיבוכיות היא הפרש הגבהים, ובמקרה הגרוע הפרש הגבהים הוא logn.

search(self, val)

* הפעולה מקבלת ערך val ומחזירה את המיקום של האיבר הראשון עם הערך val.

* בדומה ל-listToArray, הפעולה מתחילה מהאיבר הראשון בעץ ומבצעת קריאות successor עד שהיא מגיעה לצומת הראשון עם הערך val. אם לא נמצא יוחזר 1-.

* במקרה הגרוע יתבצעו n פעולות successor עד שנגיע לסוף הרשימה, ומטענה מהתרגול זה יעלה לנו O(n).

select(self, i)

* הפעולה מקבלת דרגה של צומת בעץ ומחזירה את הצומת עם הדרגה הזו.

* הפעולה עובדת בהתאם לאלגוריתם שנלמד בהרצאה. החיפוש מתחיל מהשורש - אם הדרגה שאנחנו מחפשים קטנה יותר מדרגת השורש, נפנה לבן השמאלי שלו. אם הדרגה גדולה מדרגת השורש, נפנה לבן הימני ונחסר מהדרגה שאנחנו מחפשים את דרגת השורש. כך באופן איטרטיבי עד שנגיע לצומת המבוקש.
* בפעולה יש לולאה שרצה על גובה העץ ולכן סיבוכיות זמן הריצה של הפעולה היא O(logn) כי עבור עץ AVL גובה העץ הוא O(logn).

successor(self, node)

* הפעולה מקבלת צומת node ומחזירה את המצביע לצומת הבא ברשימה, את ה-successor שלו.

* כפי שראינו בהרצאה, אם קיים ל-node בן ימני, נרד לבן הימני ולאחר מכן נרד עד הסוף לבן השמאלי. אחרת, נעלה להורה באופן איטרטיבי עד שמתבצעת עלייה ימינה.

* במקרה הגרוע נעבור על גובה העץ כולו ולכן הסיבוכיות תהיה O(logn) מפני שזה עץ AVL.

predecessor(self, node) – באופן דומה עבור ה-predecessor.

חלק ניסויי / תיאורטי

שאלה 1:

.1

מספר סידורי i	ניסוי 1 - הכנסות	ניסוי 2 - מחיקות	ניסוי 3 – הכנסות ומחיקות לסירוגין
1	5,973	1,317	1,781
2	12,420	2,606	3,742
3	24,696	5,341	7,447
4	49,588	10,618	14,755
5	98,980	21,194	29,301
6	197,657	42,546	58,489
7	395,885	84,933	118,075
8	793,156	170,243	235,095
9	1,584,024	341,196	470,731
10	3,167,925	680,546	940,897

2. קיבלנו כי כל אחד מהניסויים מקבל את הביטוי האסימפטוטי $f(n) = n$ עד כדי קבוע.

מהרצאה 8 למדנו כי כל פעולה בודדת של הכנסה/מחיקה לוקחת בממוצע O(1) ולכן נקבל כי הפעולות רצות ב-O(n). יחד עם זאת, למדנו כי ההכנסות והמחיקות לסירוגין לוקחות $\Omega(log\, n)$ לפעולה, ולכן צריך להתקבל O(nlogn), בשונה ממה שיצא לנו בניסוי 3.

שאלה 2:

.1

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
1	1.5	3	1.55	12
2	1.3	6	1.42	13
3	1.25	6	1.82	15
4	1.57	5	2.17	16
5	1.38	5	1.86	17
6	1.44	11	1.5	18
7	1.29	5	1.86	19
8	1.61	6	2.07	20
9	1.78	7	1.74	22
10	1.81	4	1.67	23

2. עבור הצומת המקסימלי בתת העץ הימני, העלות הממוצעת עבור עלייה היא סכום העלויות חלקי מספר העליות. נסמן: h – מספר העליות שמאלה. ניתן לראות כי העלות הממוצעת עבור עליה שמאלה הינו קבוע שאינו תלוי בגודל העץ. זאת מכיוון שמדובר בעץ AVL, והפרשי הגבהים בין הצמתים שביניהם אנחנו מאחדים הם לכל הפחות 0 ולכל היותר 3. נשים לב כי לאור עובדה זו, קיים מספר מצומצם של אפשרויות, ומכיוון שמדובר ב-split אקראי, מספר ההופעות של כל אחד מהפרשי הגבהים לא תלוי בגודל העץ. נשים לב כי האיחוד האחרון הינו איחוד של בן ריק עם כל תת העץ הימני, ולפי ההסבר מסעיף 3, העלות שלו היא בקירוב h עד כדי קבוע, וזאת כי העומק המינימלי של הצומת שהתקבל הינו לוגריתמי ב-n, וגם h לוגריתמי ב-n, לכן מתקבלת המסקנה. בנוסף, מספר העליות שנבצע שמאלה הינה h גם כן. מכאן נקבל:

$$\frac{\text{סכום עלות העליות}}{\text{מספר העליות}} = \frac{C \cdot h}{h+1} = O(1)$$

מכאן, קיבלנו כי אין תלות בגודל העץ ולכן זה מתיישב עם התוצאות שלנו.

בעת נטפל במקרה האקראי. יהי v צומת כלשהו בעץ. ניתחנו בביתה כי העלות של split עבורו הינה logn. מכאן, המרחק שלו מהשורש הוא logn עד כדי קבוע, לכן הוא מבצע logn עליות בסך הכול. ומכאן נקבל:

$$\frac{\text{סכום עלות העליות}}{\text{מספר העליות}} = \frac{clogn}{dlogn} = O(1)$$

מסקנה זו מתיישבת עם התוצאות שלנו, שכן עבור התוצאות שקיבלנו אין כל תלות במספר הצמתים בעץ.

3. הפעולה מתחילה מהצומת המקסימלי בתת העץ השמאלי ועולה עד לשורש. לאורך המסלול הפעולה מבצעת join בכל פעם שהיא עולה שמאלה. נשים לב כי הפעולה "תעלה" ימינה אך ורק באיטרציה האחרונה, והעלייה הזו תהיה לשורש. ה-join האחרון הוא איחוד ימני בין הבן הימני של השורש לבין הבן הימני של הצומת המקסימלי שקיבלנו (למעשה צומת ריק). הפרש הגבהים בפעולה זו הוא גובה הבן הימני + 1. נשים לב כי הפרש הגבהים בכל איחוד שמאלי שביצענו הוא לכל היותר 3. ולכן, האיחוד האחרון הוא אכן האיחוד הכי "יקר" והעלות שלו היא גובה הבן הימני של השורש + 1, וגובה הבן הימני הוא בקירוב גובה השורש (כי מדובר בעץ AVL), וגובה השורש הוא logn. לכן, נצפה שהפרש הגבהים המקסימלי של האיחוד האחרון יהיה בקירוב logn. התוצאות שקיבלנו אכן מתיישבות עם מסקנה זו.

שאלה 3:

מספר פעולות האיזון בממוצע **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	3.96	499.5	1.493	1.493	3.1	2.895
2	3.98	999.5	1.496	1.496	3.076	2.8
3	3.98	1499.5	1.497	1.497	3.083	2.81
4	3.99	1999.5	1.499	1.499	3.07	2.81
5	3.99	2499.5	1.499	1.499	3.087	2.825
6	3.99	2999.5	1.5	1.5	3.09	2.79
7	3.99	3499.5	1.5	1.5	3.07	2.8
8	3.99	3999.5	1.5	1.5	3.082	2.83
9	3.99	4499.5	1.5	1.5	3.09	2.75
10	3.99	4999.5	1.5	1.5	3.06	2.73

עומק הצומת המוכנס בממוצע **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	8.977	499.5	8	8	8.74	11.696
2	9.98	999.5	9	9	9.77	12
3	10.635	1499.5	9.64	9.64	10.3	12.3
4	10.97	1999.5	10	10	10.78	14.15
5	11.36	2499.5	10.36	10.36	11.08	14.33
6	11.63	2999.5	10.64	10.64	11.39	16.08
7	11.83	3499.5	10.83	10.83	11.55	16.49
8	11.98	3999.5	10.98	10.98	11.75	14.32
9	12.18	4499.5	11.18	11.18	11.95	14.45
10	12.36	4999.5	11.36	11.36	12.09	15.78

- לגבי ההכנסה של הסדרה המאוזנת, מפני שבהכנסה עבור עץ AVL לא התבצעו כל פעולות איזון, נצפה שהסרת פעולות האיזון לא ישפיעו על מהלך פעולת ההכנסה, ולכן מספר פעולות האיזון והעומק בממוצע זהה בין שני סוגי העצים.

- נתייחס לעומק הצומת המוכנס בממוצע עבור סדרה מאוזנת. בסדרת הכנסות זו אנחנו מכניסים איברים כפי שאנחנו מכניסים לערימה בינארית (רק כמובן בלי פעולות heapify) ובצורה הזו אנחנו יוצרים עץ כמעט מושלם. נשים לב כי ניתן לחסום מלעיל את ממוצע העומקים בעץ AVL. חסם זה יתקבל עבור עץ מושלם – כאשר הרמה האחרונה מלאה. נניח כי העץ הוא בגובה h, ו-n הוא מספר הצמתים בעץ = $2^h - 1$. נסכום את העומקים בכל רמה ונקבל את הביטוי:

$$\frac{1}{n}\sum_{i=0}^{h} i\cdot 2^i \sim \frac{1}{2^h}\sum_{i=0}^{h} i\cdot 2^i = \sum_{i=0}^{h} i\cdot 2^{i-h} = \sum_{k=0}^{h}\frac{h-k}{2^k}$$

$$= h\sum_{k=0}^{h}\frac{1}{2^k} - \sum_{k=0}^{h}\frac{k}{2^k} \le 2h - O(1) = O(h) = O(logn)$$

ומצאנו כי ב-W.C העומק הממוצע הוא O(logn), וזה אכן מתיישב עם הממצאים שהגענו אליהם.

- נתייחס להכנסות בהתחלה של עץ לא מאוזן. נשים לב כי העומק הממוצע ומספר פעולות האיזון זהים. זאת מכיוון שבעת הכנסת צומת, מספר פעולות האיזון הוא ככמות הצמתים מעליו שהגובה שלהם השתנה, ובמקרה של הכנסה בהתחלה אנחנו מייצרים "שרוך" ולכן מספר פעולות האיזון הוא כמספר הצמתים שיש בעץ, וזהו בדיוק העומק של אותו צומת. נחשב את העומק הממוצע בעץ:

$$\frac{\text{סכום העומקים}}{\text{מספר הצמתים}} = \frac{0+1+2+\cdots+(1000i-1)}{1000i} = \frac{(1000i-1)1000i}{2*1000i} = \frac{1000i-1}{2}$$

וזה אכן מתיישר עם התוצאות שהגענו אליהן.