תיעוד

:CircularList

שמרנו במחלקה ארבעה שדות:

1. **len** - אורך הרשימה
2. **start** - האינדקס של האיבר הראשון ברשימה
3. **maxLen** - אורך מקסימלי אפשרי לרשימה
4. **arr** - מערך באורך maxLen, המכיל איברים מסוג Item

מתודות:

- **circularList(maxLen) :**
  - **תפקיד** - הבנאי של המחלקה.
  - **תיאור האלגוריתם** - את שדה ה-maxLen של הרשימה נגדיר להיות הערך שמקבלת המתודה. בנוסף, נאתחל את שדה ה-len ושדה ה-start לאפס. לבסוף, ניצור מערך בגודל maxLen.
  - **סיבוכיות זמן** - $O(1)$ - שלוש ההשמות והקצאת המערך מתבצעות ב-$O(1)$.
- **:retrieve(i)**
  - **תפקיד** - הפונקציה מחזירה את האיבר באינדקס ה-i. במידה והאינדקס לא חוקי, הפונקציה מחזירה null.
  - **תיאור האלגוריתם** - אם האינדקס שהועבר אינו חוקי, נחזיר null. אחרת, נחשב את האינדקס של התא של arr המאחסן את האיבר ה-i, ונחזיר את האיבר המבוקש.
  - **סיבוכיות זמן** - $O(1)$ - מספר פעולות של השוואה ושליפה ממערך ב-$O(1)$.
- **: insert(i, k, s)**
  - **תפקיד** - הכנסת איבר בעל מפתח k וערך s לאינדקס i ברשימה. במידה והאינדקס לא חוקי, או שהרשימה מלאה, הפונקציה תחזיר 1-. אם ניתן להכניס את האיבר, הפונקציה מכניסה אותו ומחזירה 0.
  - **תיאור האלגוריתם** - אם האינדקס שהועבר אינו חוקי, או שהרשימה מלאה, הפונקציה תחזיר 1-. אחרת, אכן ניתן להכניס את האיבר. ראשית, ניצור איבר מסוג Item, בעל מפתח k וערך s. בהינתן אינדקס i, הפונקציה תבדוק לאן הוא קרוב יותר, לתחילת הרשימה או לסופה. באופן זה, נדע לאיזה כיוון נרצה להסית את חלק מאיברי המערך, בעלות הנמוכה ביותר. וכך נסית תא אחד שמאלה את כל האיברים שלפני האינדקס i, או תא אחד ימינה את כל האיברים שאחרי האינדקס i. נכניס את האיבר שיצרנו במקום ה-i של הרשימה (נחשב את האינדקס של

התא של הרלוונטי ב-arr). נשנה את שדה ה-start במידת הצורך, נעדכן את שדה ה-len, ונחזיר 0.

- **סיבוכיות זמן** - $\boldsymbol{O(min\{i,n-i\})}$ - מתבצעת הסתה של $min\{i,n-i\}$ איברים. הסתת איבר יחיד מתבצעת ב$O(1)$ ולכן הסיבוכיות היא $O(min\{i,n-i\})$.

- **delete(i) :**
  - **תפקיד** - מחיקת איבר באינדקס i. אם האינדקס שהועבר אינו חוקי, הפונקציה תחזיר 1-. אחרת, הפונקציה תמחוק את האיבר, ותחזיר 0.

תיאור האלגוריתם- במידה והאינדקס שהועבר אינו חוקי, נחזיר 1-. בהינתן אינדקס i, הפונקציה תבדוק לאן הוא קרוב יותר, לתחילת הרשימה או לסופה. באופן זה, נדע לאיזה כיוון נרצה להסית את חלק מאיברי המערך, בעלות הנמוכה ביותר. וכך, נסית תא אחד ימינה את כל האיברים שלפני האינדקס i, או תא אחד שמאלה את כל האיברים שאחרי האינדקס i. נשנה את שדה ה-start במידת הצורך, נעדכן את שדה ה-len, ונחזיר 0.

- **סיבוכיות זמן** - $\mathbf{O(min\{i,n-i\})}$ - מתבצעת הסתה של $min\{i,n-i\}$ איברים. הסתת איבר יחיד מתבצעת ב$O(1)$ ולכן הסיבוכיות היא $O(min\{i,n-i\})$.

AVLNode

שמרנו במחלקה 7 שדות:

- **key** - מפתח הצומת
- **value** - הערך השמור בצומת
- **left** - צומת מסוג AVLNode שמהווה בן שמאלי של הצומת
- **right** - צומת מסוג AVLNode שמהווה בן ימני של הצומת
- **parent** - צומת מסוג AVLNode שמהווה הורה של הצומת
- **size** - גודל תת העץ שהצומת היא שורשו
- **height** - גובה הצומת

במחלקה מתודות get ו-set לכלל השדות, והיא מממשת את המנשק IAVLNode.

AVLTree

שמרנו במחלקה שלושה שדות:

- **root**- שורש העץ, או null אם העץ ריק
- **Minimum**- הצומת עם המפתח המינימלי, או null אם העץ ריק
- **Maximum** - הצומת עם המפתח המקסימלי, או null אם העץ ריק

צמתי העץ ממומשים בעזרת המחלקה AVLNode.

מתודות:

- **AVLTree() :**
  - **תפקיד** - בנאי המחלקה
  - **תיאור האלגוריתם** - בעת יצירת עץ חדש, שורשו וצמתי המינימום והמקסימום שלו מאותחלים להיות null.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$
- **empty() :**
  - **תפקיד** - בדיקה האם העץ ריק
  - **תיאור האלגוריתם** - המתודה מחזירה true אם העץ ריק (אם השורש הוא null), ו-false אחרת.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$
- **search(k) :**
  - **תפקיד** - חיפוש צומת בעל מפתח k, והחזרת הערך שבצומת במידה וקיים, או null אם לא קיים.
  - **תיאור האלגוריתם** - אם העץ ריק, המפתח k בוודאי לא נמצא בו, ולכן מוחזר null. אחרת, באמצעות מתודת עזר (find), מייצרים מצביע לצומת בעל מפתח K אם קיים, או לצומת אחר (שאמור להיות ההורה שלו, במידה ויוכנס לעץ במבנהו הנוכחי) אם לא קיים. אם הצומת שהוחזר הוא בעל מפתח k, מוחזר הערך שבצומת. אחרת, מוחזר null.
  - **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - תחילה נבדק האם העץ ריק - $O(1)$. אם לא, מתבצע באמצעות המתודה find חיפוש בעץ ב-$O(\log n)$.
- **insert(k, i) :**
  - **תפקיד** - הכנסת צומת בעל מפתח k וערך i לעץ, והחזרת מספר הגלגולים שנעשו, או 1- אם כבר קיים בעץ צומת עם מפתח k.

    תיאור האלגוריתם - תחילה יוצרים צומת חדש עם המפתח והערך שהתקבלו. לאחר מכן, באמצעות מתודת עזר (updateMinMaxBeforeInsert), מבצעים עדכון של שדות המינימום והמקסימום של העץ.

    אם העץ ריק, מעדכנים את שדה השורש של העץ להיות הצומת החדש שיצרנו. לא בוצעו גלגולים ולכן מוחזר 0.

    אם העץ לא ריק, באמצעות מתודת עזר (find), מייצרים מצביע לצומת בעל מפתח K אם קיים, או לצומת שאמור להיות ההורה שלו, במידה ויוכנס לעץ במבנהו הנוכחי.

    אם הוחזר צומת בעל מפתח k, זה אומר שלא ניתן להכניס את הצומת הרצוי לעץ ומוחזר 1-.

    אחרת, באמצעות מתודת עזר (insertAsSonOf), מכניסים את הצומת החדש שיצרנו כבן של הצומת שהוחזר.

    לאחר מכן, מבצעים באמצעות מתודת עזר (balance) איזון לעץ, ומחזירים מספר הגלגולים שהתבצעו בעת האיזון.
  - **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - תחילה מייצרים צומת חדש ב- $O(1)$. עדכון של שדות המינימום והמקסימום באמצעות המתודה updateMinMaxBeforeInsert מתבצע ב-$O(\log n)$. חיפוש צומת בעל מפתח k באמצעות המתודה find מתבצע ב-$O(\log n)$. הכנסת הצומת כבן של צומת אחר באמצעות המתודה insertAsSonOf מתבצע ב-$O(1)$. איזון העץ באמצעות המתודה balance מתבצע ב-$O(\log n)$. הפעולות מתבצעות אחת אחרי השניה ולכן סיבוכיות הקוד היא $O(\log n)$.
- **find(k) :**
  - **תפקיד** - מתודת עזר- חיפוש צומת בעל מפתח k, והחזרת הצומת במידה וקיים. אם לא, יוחזר הצומת שאמור להיות ההורה של הצומת הנתון, במידה והצומת יוכנס לעץ במבנהו הנוכחי.

- **תיאור האלגוריתם** - מתבצע חיפוש בעץ, מהשורש כלפי מטה. המשך המסלול מכל צומת נקבע על ידי k ומפתח הצומת הנוכחי (זהו עץ חיפוש בינארי). החיפוש נעצר כאשר מגיעים לצומת עם המפתח k או לעלה שמפתחו אינו k, ומוחזר הצומת בו סיימנו.
- **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - מתבצע חיפוש בעץ, אשר במקרה הגרוע מתחיל בשורש העץ וממשיך מטה עד שמגיעים לעלה כלשהו. גובה העץ הוא $O(\log n)$ ולכן החיפוש גם כן מתבצע ב-$O(\log n)$.

- **updateMinMaxBeforeInsert(node) :**
  - **תפקיד** - מתודת עזר - עדכון שדות המינימום והמקסימום של העץ, לפני הכנסת צומת נתון.
  - **תיאור האלגוריתם** - אם העץ ריק, המינימום והמקסימום החדשים של העץ יהיו הצומת שיוכנס. אחרת, נבדוק באמצעות השוואת מפתחות האם הצומת שהתקבל אמור להיות מקסימום או מינימום, ונשנה את השדות במידת הצורך.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - במתודה השוואות והשמות ב-$O(1)$.
- **insertAsSonOf(parent, son) :**
  - **תפקיד** - מתודת עזר - הכנסת הצומת son לעץ, כבן של parent.
  - **תיאור האלגוריתם** - מתבצעת בדיקה איזה בן של parent הצומת son צריך להיות, ומקשרים בין שני הצמתים באמצעות המצביעים הרלוונטים.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - במתודה השוואות ושינוי פוינטרים ב-$O(1)$.
- **heightCal(x) :**
  - **תפקיד** - מתודת עזר - חישוב הגובה העדכני של צומת והחזרתו.
  - **תיאור האלגוריתם** - מתבצע חישוב גובה הצומת לפי הנוסחה שנלמדה בכיתה.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - במתודה השוואות וחישובים ב-$O(1)$.
- **sizeCal(x):**
  - **תפקיד** - מתודת עזר - חישוב הגודל העדכני של צומת (מספר הצמתים בתת העץ שצומת זה הוא השורש שלה) והחזרתו.
  - **תיאור האלגוריתם** - מתבצע חישוב גודל הצומת לפי הנוסחה שנלמדה בכיתה.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - במתודה השוואות וחישובים ב-$O(1)$.
- **leftRotation(x) :**
  - **תפקיד** - מתודת עזר - הפונקציה מבצעת גלגול שמאלה על תת העץ ש-x השורש שלו, ומחזירה את השורש החדש של תת עץ זה.
  - **תיאור האלגוריתם** - מתבצע גלגול שמאלה, באמצעות שינוי פוינטרים, כפי שנלמד בכיתה. לאחר מכן, באמצעות מתודות עזר (sizeCal,heightCal), מעדכנים את הגבהים והגדלים של הצמתים המעורבים בגלגול.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - שינוי הפוינטרים בתחילת הקוד מתבצע ב-$O(1)$. לאחר מכן, עדכון הגבהים והגדלים באמצעות המתודות heightCal ו- sizeCal מתבצע גם כן ב-$O(1)$.
- **rightRotation(x) :**
  - **תפקיד** - מתודת עזר - הפונקציה מבצעת גלגול ימינה על תת העץ ש-x השורש שלו, ומחזירה את השורש החדש של תת עץ זה.
  - **תיאור האלגוריתם** - מתבצע גלגול ימינה, באמצעות שינוי פוינטרים, כפי שנלמד בכיתה. לאחר מכן,באמצעות מתודות עזר (sizeCal,heightCal), מעדכנים את הגבהים והגדלים של הצמתים המעורבים בגלגול.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - שינוי הפוינטרים בתחילת הקוד מתבצע ב-$O(1)$. לאחר מכן, עדכון הגבהים והגדלים באמצעות המתודות heightCal ו- sizeCal מתבצע גם כן ב-$O(1)$.

- **BFcal(x) :**
  - **תפקיד** - מתודת עזר - חישוב ה-BF של צומת והחזרתו.
  - **תיאור האלגוריתם** - מתבצע חישוב ה-BF של הצומת לפי הנוסחה שנלמדה בכיתה.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - במתודה השוואות וחישובים ב-$O(1)$.
- **balance (x) :**
  - **תפקיד** - מתודת עזר - איזון העץ והחזרת מספר הגלגולים שנעשו בעת האיזון.
  - **תיאור האלגוריתם** - מתבצע מעבר על העץ, מצומת נתונה עד לשורש העץ. עבור כל צומת בו עוברים, באמצעות מתודות עזר (sizeCal,heightCal), מעדכנים את גובה וגודל הצומת. אם הצומת הוא עבריין AVL, באמצעות מתודות עזר (leftRotation,rightRotation), מבצעים את הגלגולים המתאימים, ומעדכנים משתנה שסופר את מספר הגלגולים שנעשו. לבסוף מחזירים את ערכו של משתנה זה.
  - **סיבוכיות זמן** - $\boldsymbol{O(log\, n)}$ - במתודה לולאה שרצה במקרה הגרוע מעלה כלשהו עד לשורש העץ, ומשום שזהו עץ AVL, לולאה זו מבצעת $O(log\, n)$ איטרציות. בכל איטרציה, מתבצעות באמצעות המתודה BFcal חישובים ב-$O(1)$ ובעת הצורך מתבצעים באמצעות המתודות rightRotation ו-leftRotation גלגולים ב-$O(1)$. כמו כן, עדכון הגובה והגודל באמצעות המתודות heightCal ו-sizeCal מתבצע גם כן ב-$O(1)$.
- **delete(k) :**
  - **תפקיד** - מחיקת צומת בעל מפתח k מהעץ, והחזרת מספר הגלגולים שנעשו, או 1- אם לא קיים בעץ צומת עם מפתח k.
  - **תיאור האלגוריתם** - אם העץ ריק, המפתח k בוודאי לא נמצא בו, ולכן מוחזר 1-.

    אם הוא לא ריק, באמצעות מתודת עזר (updateMinMaxBeforeDelete), מבצעים עדכון של שדות המינימום והמקסימום של העץ.

    לאחר מכן, באמצעות מתודת עזר (find), מייצרים מצביע לצומת בעל מפתח K אם קיים, או לצומת שאמור להיות ההורה שלו, במידה ויוכנס לעץ במבנהו הנוכחי.

    אם הוחזר צומת בעל מפתח שאינו k, זה אומר שלא קיים בעץ צומת עם מפתח k
    ולכן מוחזר 1-.

    אחרת, באמצעות אחת ממתודות העזר (deleteLeaf, deleteWithTwoChildren, deleteWithOnlyRightChild, deleteWithOnlyLeftChild), נמחק את הצומת שהוחזר.

    נבצע באמצעות מתודת עזר (balance) איזון של העץ ונחזיר את מספר הגלגולים שנעשו.
  - **סיבוכיות זמן** - $\boldsymbol{O(log\, n)}$ - עדכון של שדות המינימום והמקסימום באמצעות המתודה updateMinMaxBeforeDelete מתבצע ב-$O(log\, n)$. חיפוש צומת בעל מפתח k באמצעות המתודה find מתבצע ב-$O(log\, n)$. מחיקת הצומת באמצעות אחת ממתודות העזר מתבצעת במקרה הגרוע ב-$O(log\, n)$. איזון העץ באמצעות המתודה balance מתבצע ב-$O(log\, n)$. הפעולות מתבצעות אחת אחרי השניה ולכן סיבוכיות הקוד היא $O(log\, n)$.
- **updateMinMaxBeforeDelete (k) :**
  - **תפקיד** - מתודת עזר - עדכון שדות המינימום והמקסימום של העץ, לפני מחיקת צומת בעל מפתח k.
  - **תיאור האלגוריתם** - אם k הוא המפתח של המינימום בעץ כרגע, נעדכן את שדה המינימום להיות העוקב שלו. אם זהו עלה, העוקב שלו הוא אביו, ואחרת, נשתמש בפונקציית עזר (successorWithRightChild) למציאת העוקב.

    אם k הוא המפתח של המקסימום בעץ כרגע, נעדכן את שדה המקסימום להיות הקודם שלו. אם זהו עלה, הקודם שלו הוא אביו, ואחרת, נשתמש בפונקציית עזר (predecessorWithLeftChild) למציאת הקודם.

- **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - במתודה מספר השוואות ב-$O(1)$. מציאת העוקב באמצעות המתודה successorWithRightChild מתבצעת ב-$O(\log n)$, ומציאת הקודם באמצעות המתודה predecessorWithLeftChild מתבצעת ב-$O(\log n)$.

- **deleteLeaf(node) :**
  - **תפקיד** - מתודת עזר - מחיקת עלה מהעץ והחזרת הצומת שבפעולת איזון העץ שתתבצע בהמשך, נטפס ממנו כלפי מעלה. אם לא יהיה צורך באיזון, נחזיר null.
  - **תיאור האלגוריתם** - אם הצומת הוא השורש, מעדכנים את שדה השורש של העץ להיות null, ומחזירים null. אחרת, מוצאים איזה בן הוא של הוריו ומוחקים אותו. המתודה תחזיר את ההורה של הצומת שנמחק.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - במתודה מספר השוואות והשמות ב-$O(1)$.
- **deleteWithOnlyRightChild (node) :**
  - **תפקיד** - מתודת עזר - מחיקת צומת עם בן ימני בלבד מהעץ והחזרת הצומת שבפעולת איזון העץ שתתבצע בהמשך, נטפס ממנו כלפי מעלה. אם לא יהיה צורך באיזון, נחזיר null.
  - **תיאור האלגוריתם** - אם הצומת הוא השורש, מעדכנים את שדה השורש של העץ להיות הבן הימני של הצומת, ומחזירים null. אחרת, מוצאים איזה בן הוא של הוריו, מוחקים אותו, ומעדכנים את הפוינטרים הרלוונטיים. המתודה תחזיר את ההורה של הצומת שנמחק.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - במתודה מספר השוואות והשמות ב-$O(1)$.
- **deleteWithOnlyLeftChild (node) :**
  - **תפקיד** - מתודת עזר - מחיקת צומת עם בן שמאלי בלבד מהעץ והחזרת הצומת שבפעולת איזון העץ שתתבצע בהמשך, נטפס ממנו כלפי מעלה. אם לא יהיה צורך באיזון, נחזיר null.
  - **תיאור האלגוריתם** - אם הצומת הוא השורש, מעדכנים את שדה השורש של העץ להיות הבן השמאלי של הצומת, ומחזירים null. אחרת, מוצאים איזה בן הוא של הוריו, מוחקים אותו, ומעדכנים את הפוינטרים הרלוונטיים. המתודה תחזיר את ההורה של הצומת שנמחק.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ - במתודה מספר השוואות והשמות ב-$O(1)$.
- **deleteWithTwoChildren (node) :**
  - **תפקיד** - מתודת עזר - מחיקת צומת עם שני בנים מהעץ והחזרת הצומת שבפעולת איזון העץ שתתבצע בהמשך, נטפס ממנו כלפי מעלה. אם לא יהיה צורך באיזון, נחזיר null.
  - **תיאור האלגוריתם** - תחילה, באמצעות מתודת עזר (successorWithRightChild), מוצאים את העוקב של הצומת, ומוחקים אותו באמצעות מתודת עזר (deleteLeaf, deleteWithOnlyRightChild). כעת מעדכנים את הפוינטרים הרלוונטים כך שבמקום הצומת שרוצים למחוק יהיה הצומת העוקב שלו, ולבסוף מנתקים את הצומת מהעץ. המתודה תחזיר את ההורה של העוקב של הצומת שנמחק.
  - **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - מציאת העוקב באמצעות המתודה successorWithRightChild מתבצע ב-$O(\log n)$. מחיקת העוקב באמצעות המתודות deleteLeaf ו-deleteWithOnlyRightChild מתבצע ב-$O(1)$. שינוי הפוינטרים לשם ניתוק הצומת מהעץ והחלפתו בעוקב שלו מתבצע גם כן ב-$O(1)$.
- **successorWithRightChild(node) :**
  - **תפקיד** - מתודת עזר - מציאת והחזרת ה-successor של צומת נתון בעל בן ימני.
  - **תיאור האלגוריתם** - הפונקציה מקבלת צמתים שלהם בן ימני ומחפשת את הצומת עם המפתח המינמלי בתת העץ הימני של הצומת. המתודה ניגשת לבן הימני של הצומת וממנו יורדת כל הדרך שמאלה עד שמגיעה לצומת המינמלי בתת עץ זה.

- **סיבוכיות זמן** - $O(\log n)$ - במתודה לולאה שרצה במקרה הגרוע מהבן הימני של הצומת הנתון, לעלה כלשהו. הגובה של תת העץ ששורשו הוא הבן הימני הנ"ל הוא לכל היותר $O(\log n)$.

- **predecessorWithLeftChild(node) :**
  - **תפקיד** - מתודת עזר - מציאת והחזרת ה- predecessor של צומת נתון בעל בן שמאלי.
  - **תיאור האלגוריתם**- הפונקציה מקבלת צמתים שלהם בן שמאלי ומחפשת את הצומת עם המפתח המקסימלי בתת העץ השמאלי של הצומת. המתודה ניגשת לבן השמאלי של הצומת וממנו יורדת כל הדרך ימינה עד שמגיעה לצומת המקסימלי בתת עץ זה.
  - **סיבוכיות זמן** - $O(\log n)$ - במתודה לולאה שרצה במקרה הגרוע מהבן השמאלי של הצומת הנתון, לעלה כלשהו. הגובה של תת העץ ששורשו הוא הבן השמאלי הנ"ל הוא לכל היותר $O(\log n)$.

- **min() :**
  - **תפקיד** - החזרת ערכו (value) של הצומת בעל המפתח המינימלי, או null אם העץ ריק.
  - **תיאור האלגוריתם** - אם העץ ריק, מוחזר null. אחרת, הפוקנציה מחזירה את ערכו של הצומת השמור בשדה המינימום של העץ.
  - **סיבוכיות זמן** - $O(1)$

- **max() :**
  - **תפקיד** - החזרת ערכו (value) של הצומת בעל המפתח המקסימלי, או null אם העץ ריק.
  - **תיאור האלגוריתם** - אם העץ ריק, מוחזר null. אחרת, הפוקנציה מחזירה את ערכו של הצומת השמור בשדה המקסימום של העץ.
  - **סיבוכיות זמן** - $O(1)$

- **keysToArray() :**
  - **תפקיד** - הפונקציה מחזירה מערך ממוין המכיל את כל המפתחות בעץ, או מערך ריק אם העץ ריק.
  - **תיאור האלגוריתם** - במידה והעץ ריק, הפונקציה מחזירה מערך ריק. אחרת, יוצרים מערך באורך כמות הצמתים בעץ. לאחר מכן, באמצעות מתודת עזר (keysToArrayRec), מכניסים למערך את מפתחות הצמתים בעץ, בסדר ממוין.
  - **סיבוכיות זמן** - $O(n)$ - יצירת המערך מתבצע ב-$O(1)$. מילוי המערך באמצעות המתודה keysToArrayRec מתבצע ב-$O(n)$.

- **keysToArrayRec(x, arr, index) :**
  - **תפקיד** - מתודת עזר - מתודה רקורסיבית להכנסת מפתחות של הצמתים בעץ למערך שקיבלה, לפי סדר הופעתם בעץ.
  - **תיאור האלגוריתם** - המתודה מקבלת צומת, מערך ואינדקס, ומעדכנת את המערך במקום. היא מבצעת סיור inorder בעץ, שבמהלכו היא מכניסה את מפתחות הצמתים למערך, לפי סדר הופעתם. בכל קריאה רקורסיבית, המתודה קוראת לעצמה עבור תת העץ השמאלי של הצומת, מכניסה למערך את המפתח של האיבר שמיוצג על ידי הצומת, ואז קוראת לעצמה עבור תת העץ הימני של הצומת. המתודה מחזירה לעצמה את האינדקס בו עליה להכניס למערך את האיבר הבא, ומקדמת אותו במהלך הרקורסיה.
  - **סיבוכיות זמן** - $O(n)$ - עלות לינארית למספר הצמתים בעץ, שכן המתודה מבקרת בכל צומת בעץ פעם אחת. בכל קריאה רקורסיבית מתבצעות פעולות השמה בעלות קבועה.

- **infoToArray() :**
  - **תפקיד** - הפונקציה מחזירה מערך מחרוזות המכיל את כל המחרוזות בעץ, ממוינות על פי סדר המפתחות, או מערך ריק אם העץ ריק.

- **תיאור האלגוריתם** - במידה והעץ ריק, הפונקציה מחזירה מערך ריק. אחרת, יוצרים מערך באורך כמות הצמתים בעץ. לאחר מכן, באמצעות מתודת עזר (infoToArrayRec), מכניסים למערך את ערכי הצמתים בעץ, לפי סדר המפתחות המתאים.
- **סיבוכיות זמן** - $\boldsymbol{O(n)}$ - יצירת המערך מתבצע ב-$O(1)$. מילוי המערך באמצעות המתודה infoToArrayRec מתבצע ב-$O(n)$.

- **infoToArrayRec(x, arr, index) :**
  - **תפקיד** - מתודת עזר - מתודה רקורסיבית להכנסת המידע של הצמתים בעץ למערך שקיבלה, מסודר לפי סדר המפתחות בעץ.
  - **תיאור האלגוריתם** - המתודה מקבלת צומת, מערך ואינדקס, ומעדכנת את המערך במקום. היא מבצעת סיור inorder בעץ, שבמהלכו היא מכניסה את מפתחות הצמתים למערך, לפי סדר הופעתם. בכל קריאה רקורסיבית, המתודה קוראת לעצמה עבור תת העץ השמאלי של הצומת, מכניסה למערך את הערך של האיבר שמיוצג על ידי הצומת, ואז קוראת לעצמה עבור תת העץ הימני של הצומת. המתודה מחזירה לעצמה את האינדקס בו עליה להכניס למערך את האיבר הבא, ומקדמת אותו במהלך הרקורסיה.
  - **סיבוכיות זמן** - $\boldsymbol{O(n)}$ - עלות לינארית למספר הצמתים בעץ, שכן המתודה מבקרת בכל צומת בעץ פעם אחת. בכל קריאה רקורסיבית מתבצעות פעולות השמה בעלות קבועה.
- **size() :**
  - **תפקיד** - המתודה מחזירה את מספר הצמתים בעץ.
  - **תיאור האלגוריתם** - אם העץ ריק, המתודה מחזירה 0. אחרת, מחזירה הגודל של צומת השמורה בשדה ה- root של העץ.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$
- **getRoot() :**
  - **תפקיד** - מחזירה את השורש של העץ
  - **תיאור האלגוריתם** - מחזירה הצומת השמורה בשדה ה- root של העץ.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$

TreeList

את הרשימה העצית מימשנו באמצעות המחלקה AVLTree, כאשר האיברים ברשימה מיוצגים על ידי צמתי העץ (מהמחלקה AVLNode). בעת החזרת איבר מהרשימה, נחזיר איבר מסוג item עם מפתח וערך תואם לצומת.

שמרנו במחלקה ארבעה שדות:

- **tree** - עץ AVL המכיל את הערכים והמפתחות של האיברים שברשימה
- **len** - אורך הרשימה
- **first** - צומת שמייצג את האיבר הראשון ברשימה, או null אם הרשימה ריקה
- **last** - צומת שמייצג את האיבר האחרון ברשימה, או null אם הרשימה ריקה

מתודות:

- **TreeList() :**

- **תפקיד** - בנאי המחלקה
- **תיאור האלגוריתם** - בעת יצירת רשימה חדשה, אורך הרשימה יאותחל לאפס, והאיבר הראשון והאחרון יהיו null (שכן הרשימה ריקה). כמו כן, ניצור עץ AVL שבהמשך יכיל את הערכים והמפתחות של האיברים שיהיו ברשימה.
- **סיבוכיות זמן** - $\boldsymbol{O(1)}$

• **:retrieve(int i)**

- **תפקיד** - הפונקציה מחזירה את האיבר באינדקס ה-i. במידה והאינדקס לא חוקי, הפונקציה מחזירה null.
- **תיאור האלגוריתם** - באמצעות מתודת עזר (retrieveNode), מוצאים את הצומת שמייצגת את האיבר הi'י ברשימה (אם האינדקס לא חוקי, מתודת העזר מחזירה null). במידה והצומת שהוחזרה אינו null, יוצרים איבר מסוג item, עם מפתח וערך הצומת, ומחזירים אותו. אחרת מוחזר null.
- **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - מציאת הצומת המבוקש באמצעות מתודת העזר retrieveNode בעלות $O(\log n)$, ואתחול item חדש ב-$O(1)$.

• **:retrieveNode(i)**

- **תפקיד** - מתודת עזר - הפונקציה מחזירה את הצומת שמייצג את האיבר באינדקס ה-i. במידה והאינדקס לא חוקי, הפונקציה מחזירה null.
- **תיאור האלגוריתם** - אם האינדקס אינו חוקי, מוחזר null. אם האינדקס הוא 0, או **len-1,** מוחזר first או last בהתאמה. אחרת, מתבצע חיפוש של הצומת המבוקש, בדומה לאופן מימוש פעולת הsearch שנלמדה בכיתה.
- **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - במתודה מספר בדיקות והשוואות ב-$O(1)$. לאחר מכן, מתבצע חיפוש של הצומת הרלוונטי, אשר במקרה הגרוע מתחיל בשורש העץ וממשיך מטה עד שמגיעים לעלה כלשהו. גובה העץ הוא $O(\log n)$ ולכן החיפוש מתבצע גם כן ב-$O(\log n)$.

• **: insert(i, k, s)**

- **תפקיד** - הכנסת איבר בעל מפתח k וערך s לאינדקס i ברשימה. במידה והאינדקס לא חוקי, הפונקציה תחזיר 1-. אם ניתן להכניס את האיבר, הפונקציה מכניסה אותו ומחזירה 0.
- **תיאור האלגוריתם** - אם האינדקס שהועבר אינו חוקי, הפונקציה תחזיר 1-. אחרת, יוצרים צומת חדש המכיל את המפתח והערך שהועברו למתודה. במידה והעץ ריק, בעזרת מתודה עזר (insertToEmptyList) מכניסים את הצומת לעץ ומעדכנים את השדות הרלוונטים. במידה והעץ אינו ריק, באמצעות מתודת עזר (insertToNonEmptyList), מכניסים את האיבר לעץ, מעדכנים את השדות הרלוונטים, ומאזנים ומעדכנים את העץ באמצעות מתודת עזר (balance). לבסוף מחזירים 0.
- **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - יצירת צומת חדש ב-$O(1)$. במקרה הגרוע, בו ההכנסה היא לעץ שאינו ריק, אנו מבצעים קריאה לשתי מתודות עזר insertToNonEmptyList ו-balance בעלות $O(\log n)$.

- **insertToEmptyList(newNode):**
  - **תפקיד** - מתודת עזר - הכנסת צומת לרשימה ריקה.
  - **תיאור האלגוריתם** - המתודה מכניסה את הצומת לרשימה באמצעות המתודה insert של AVLTree (כך מתעדכן שורש העץ להיות הצומת שהתקבל). לאחר מכן, מעדכנים את שדות ה-first וה-last גם כן לצומת זה, ומעלים את אורך הרשימה באחד.
  - **סיבוכיות זמן** - $\boldsymbol{O(1)}$ – הפעלת ה-insert של AVLTree מתבצעת תמיד על עץ ריק ולכן סיבוכיות פעולה זו היא $O(1)$. עדכון השדות לאחר מכן מתבצע גם הוא ב-$O(1)$.
- **insertToNonEmptyList(i,newNode):**
  - **תפקיד** - מתודת עזר - הכנסת צומת לרשימה שאינה ריקה באינדקס i. המתודה מחזירה את הצומת שבפעולת איזון עץ ה-AVL שבשדה ה-tree שתתבצע בהמשך, נטפס ממנו כלפי מעלה.
  - **תיאור האלגוריתם** - אם האינדקס הינו 0 או len, מכניסים את הצומת לעץ באמצעות שדה ה-first או ה-last ומעדכנים אותם בהתאם. אחרת, באמצעות מתודת עזר (retrieveNode), מוצאים את האיבר הi ברשימה כרגע. במידה ואין לו בן שמאלי, מכניסים את הצומת כבן שמאלי שלו. אחרת, באמצעות מתודת עזר (retrieveNode), מוצאים את האיבר באינדקס i-1, ומכניסים את הצומת כבן ימני שלו (בהכרח לצומת זה אין בן ימני, כי אם היה לו בן ימני, אז האיבר באינדקס הi היה האיבר באינדקס 0 בתת העץ הימני שלו, ולכן ל-i אין בן שמאלי, בסתירה). מעדכנים את אורך הרשימה ומחזירים את ההורה של האיבר שהכנסנו.
  - **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - במקרה הגרוע מתבצעות שתי קריאות למתודת עזר retrieveNode בעלות של $O(\log n)$. כמו כן, ישנם מספר השוואות, עדכוני שדות, ושינוי פוינטרים ב-$O(1)$.
- **delete(i) :**
  - **תפקיד** - מחיקת איבר באינדקס i. אם האינדקס שהועבר אינו חוקי, הפונקציה תחזיר 1-. אחרת, הפונקציה תמחוק את האיבר, ותחזיר 0.
  - **תיאור האלגוריתם** - במידה והאינדקס שהועבר אינו חוקי, נחזיר 1-. אחרת, באמצעות מתודת עזר (retrieveNode) נמצא את הצומת שמייצג את האיבר שנרצה למחוק. אם האיבר הוא הראשון ו/או האחרון, נעדכן את שדות ה-first וה-last בהתאם. אחרת, נמחק את הצומת הרצוי באמצעות אחת ממתודות העזר (deleteLeaf, deleteWithTwoChildren, deleteWithOnlyLeftChild, deleteWithOnlyRightChild). לאחר מכן, נבצע באמצעות מתודת עזר (balance) איזון של העץ, נעדכן את שדה ה-len, ונחזיר 0.
  - **סיבוכיות זמן** - $\boldsymbol{O(\log n)}$ - מציאת הצומת שמייצג את האיבר שנרצה למחוק באמצעות מתודת העזר retrieveNode בעלות של $O(\log n)$. עדכון השדות הרלוונטים של הרשימה ומחיקת הצומת באמצעות אחת ממתודות העזר מתבצעות במקרה הגרוע ב-$O(\log n)$. איזון העץ באמצעות המתודה balance מתבצע ב-$O(\log n)$. הפעולות מתבצעות אחת אחרי השניה ולכן סיבוכיות הקוד היא $O(\log n)$.

מדידות

1. יתרון לרשימה מעגלית - הכנסה בסוף:

המיקום שבחרנו להכנסת האיברים לשם הדגשת יתרון הרשימה המעגלית על פני הרשימה העצית הוא סוף הרשימה. זאת, משום שהכנסה לרשימה המעגלית בסופה (או תחילתה) מתבצעת ב-$O(1)$, שכן אין צורך בהזזת אף איבר ברשימה. לעומת זאת, הכנסה לרשימה עצית בסופה (או תחילתה) מתבצעת ב-$O(log\, n)$ כאשר n הוא מספר הצמתים ברשימה בעת ההכנסה.

מבחינת זמן הכנסה ממוצע, ציפינו לראות יתרון לרשימה המעגלית על פני הרשימה העצית, כפי שאכן קרה. בנוסף, משום שההכנסה לרשימה המעגלית היא ב-$O(1)$, ציפינו שזמן ההכנסה הממוצע לרשימה יהיה יחסית זהה עבור כל אורך רשימה. כמו כן, לא ציפינו לראות עלייה משמעותית בזמן ההכנסה הממוצע לרשימה עצית, זאת משום שקצב העלייה של log הוא יחסית איטי. יתר על כן, מכך שבחרנו להכניס בכל פעם בסוף הרשימה, ציפינו שנזדקק למספר רב של פעולות גלגול, כאשר מרבית הגלגולים יהיו גלגולים שמאלה. אכן, ניתן לראות כי בממוצע בכל הכנסה התבצע גלגול אחד שמאלה, ולמעשה כלל לא התבצעו גלגולים ימינה (דבר הנובע מהעובדה שעברייני AVL שייוצרו בעץ תמיד יהיו עם נטייה ימינה, כלומר לעבריין BF -2 ולבנו הימני BF -1).

מספר סידורי	מספר פעולות	זמן הכנסה ממוצע עבור רשימה מעגלית	זמן הכנסה ממוצע עבור רשימה עצית	כמות גלגולים ימינה ממוצעת עבור רשימה עצית	כמות גלגולים שמאלה ממוצעת עבור רשימה עצית
1	10000	$7.43 \cdot 10^{-8}$	$9.69 \cdot 10^{-8}$	0	0.9986
2	20000	$4.28 \cdot 10^{-8}$	$3.51 \cdot 10^{-7}$	0	0.9992
3	30000	$3.83 \cdot 10^{-7}$	$1.29 \cdot 10^{-7}$	0	0.9995
4	40000	$2.72 \cdot 10^{-8}$	$1.34 \cdot 10^{-7}$	0	0.9996
5	50000	$1.46 \cdot 10^{-8}$	$1.27 \cdot 10^{-7}$	0	0.9997
6	60000	$1.48 \cdot 10^{-8}$	$1.09 \cdot 10^{-7}$	0	0.9997
7	70000	$1.38 \cdot 10^{-8}$	$1.12 \cdot 10^{-7}$	0	0.9997
8	80000	$1.36 \cdot 10^{-8}$	$1.22 \cdot 10^{-7}$	0	0.9998
9	90000	$1.79 \cdot 10^{-8}$	$1.17 \cdot 10^{-7}$	0	0.9998
10	100000	$1.86 \cdot 10^{-8}$	$1.37 \cdot 10^{-7}$	0	0.9998

2. יתרון לרשימה עצית - הכנסה באמצע:

המיקום שבחרנו להכנסת האיברים לשם הדגשת יתרון הרשימה העצית על פני הרשימה המעגלית הוא אמצע הרשימה. כאמור, סיבוכיות הכנסת איבר לאינדקס ה-i ברשימה מעגלית היא $O(\min\{i, n-i\})$ כאשר n הוא אורך הרשימה בעת ההכנסה. מכך ניתן להסיק כי הכנסת איבר לאמצע הרשימה דורשת את העלות הגבוהה ביותר מבחינת זמן ריצה. נקבל כי עלות ההכנסה באמצע רשימה באורך n היא-

$O\left(min\left\{\left\lfloor\frac{n}{2}\right\rfloor, n-\left\lfloor\frac{n}{2}\right\rfloor\right\}\right) = O\left(\left\lfloor\frac{n}{2}\right\rfloor\right) = O(n)$. לעומת זאת, הכנסה לרשימה עצית מתבצעת תמיד ב-$O(log\ n)$ כאשר n הוא מספר הצמתים ברשימה בעת ההכנסה. נשים לב כי על אף שצורת הכנסה זו עלולה לדרוש לא מעט גלגולים, כל גלגול מתבצע ב-$O(1)$ ולכן אין זה פוגע ביתרון הרשימה העצית.

מבחינת זמן הכנסה ממוצע, ציפינו לראות יתרון לרשימה העצית על פני הרשימה המעגלית, כפי שאכן קרה. בנוסף, משום שההכנסה לרשימה המעגלית היא ב-$O(n)$, ציפינו שזמן ההכנסה הממוצע לרשימה יגדל לינארית ביחס לאורך הרשימה. כמו כן, לא ציפינו לראות עלייה משמעותית בזמן ההכנסה הממוצע לרשימה עצית, זאת משום שקצב העלייה של log הוא יחסית איטי. יתר על כן, מכך שבחרנו להכניס בכל פעם באמצע הרשימה, ציפינו שנזדקק ליחסית הרבה גלגולים, ושמספר הגלגולים ימינה ושמאלה יהיה יחסית זהה, כפי שאכן קרה.

מספר סידורי	מספר פעולות	זמן הכנסה ממוצע עבור רשימה מעגלית	זמן הכנסה ממוצע עבור רשימה עצית	כמות גלגולים ימינה ממוצעת עבור רשימה עצית	כמות גלגולים שמאלה ממוצעת עבור רשימה עצית
1	10000	$1.07 \cdot 10^{-5}$	$1.27 \cdot 10^{-6}$	0.8103	0.8118
2	20000	$3.68 \cdot 10^{-5}$	$9.85 \cdot 10^{-7}$	0.8115	0.8119
3	30000	$2.32 \cdot 10^{-5}$	$8.53 \cdot 10^{-7}$	0.8116	0.8122
4	40000	$2.95 \cdot 10^{-5}$	$5.50 \cdot 10^{-7}$	0.8117	0.8125
5	50000	$3.68 \cdot 10^{-5}$	$3.94 \cdot 10^{-7}$	0.8120	0.8123
6	60000	$4.38 \cdot 10^{-5}$	$4.33 \cdot 10^{-7}$	0.8120	0.8123
7	70000	$5.23 \cdot 10^{-5}$	$2.70 \cdot 10^{-7}$	0.8121	0.8124
8	80000	$5.97 \cdot 10^{-5}$	$2.91 \cdot 10^{-7}$	0.8121	0.8124
9	90000	$6.69 \cdot 10^{-5}$	$3.19 \cdot 10^{-7}$	0.8122	0.8124
10	100000	$7.39 \cdot 10^{-5}$	$2.63 \cdot 10^{-7}$	0.8122	0.8124

3. הכנסה רנדומלית:

מבחינת זמן הכנסה ממוצע, ציפינו לראות יתרון לרשימה העצית על פני הרשימה המעגלית, כפי שאכן קרה. שכן, מספר ההזזות (הזזת איברים לשם פינוי מקום לאיבר שיוכנס) בעת הכנסה לרשימה מעגלית באורך n נע בין 0 ל-$\frac{n}{2}$. משום שהכנסה רנדומלית "מפזרת" את האיברים באופן יחסית אחיד, ציפינו שכמות ההזזות הממוצעת בעת הכנסה תהיה סביב $\frac{n}{4}$. סיבוכיות ההכנסה הינה לינארית לכמות הזזות ולכן בממוצע סיבוכיות כל הכנסה היא $O(\frac{n}{4}) = O(n)$. לעומת זאת, הכנסה לרשימה עצית מתבצעת תמיד ב-$O(log\ n)$ כאשר n הוא מספר הצמתים ברשימה בעת ההכנסה. בנוסף, משום שמדובר בהכנסה רנדומלית, ציפינו שמספר הגלגולים שנזדקק להם יהיה נמוך מההכנסות הקודמות, משום שהכנסה חוזרת במיקום ספציפי מייצרת תמיד חוסר איזון בנקודה זו בעץ, ואילו הכנסה רנדומלית "מפזרת" באופן יחסית אחיד את האיברים.

מספר סידורי	מספר פעולות	זמן הכנסה ממוצע עבור רשימה מעגלית	זמן הכנסה ממוצע עבור רשימה עצית	כמות גלגולים ימינה ממוצעת עבור רשימה עצית	כמות גלגולים שמאלה ממוצעת עבור רשימה עצית
1	10000	$5.30 \cdot 10^{-6}$	$1.25 \cdot 10^{-6}$	0.3482	0.3488
2	20000	$8.45 \cdot 10^{-6}$	$1.01 \cdot 10^{-6}$	0.3459	0.3447
3	30000	$1.22 \cdot 10^{-5}$	$6.76 \cdot 10^{-7}$	0.3461	0.3417
4	40000	$1.55 \cdot 10^{-5}$	$7.10 \cdot 10^{-7}$	0.3470	0.3511
5	50000	$1.83 \cdot 10^{-5}$	$1.01 \cdot 10^{-6}$	0.3494	0.3477
6	60000	$2.21 \cdot 10^{-5}$	$7.58 \cdot 10^{-7}$	0.3508	0.3498
7	70000	$2.55 \cdot 10^{-5}$	$5.17 \cdot 10^{-7}$	0.3488	0.3483
8	80000	$2.93 \cdot 10^{-5}$	$4.01 \cdot 10^{-7}$	0.3521	0.3494
9	90000	$3.31 \cdot 10^{-5}$	$5.65 \cdot 10^{-7}$	0.3459	0.3494
10	100000	$3.68 \cdot 10^{-5}$	$6.26 \cdot 10^{-7}$	0.3491	0.3478