מבני נתונים – תרגיל מעשי 1

מדידות

1. Insert/delete:

מספר סידורי	מספר פעולות	מספר פעולות האיזון הממוצע לפעולת insert	מספר פעולות האיזון הממוצע לפעולת delete	מספר פעולות האיזון המקסימלי לפעולת insert	מספר פעולות האיזון המקסימלי לפעולת delete
1	10,000	3.073	2.272	15	28
2	20,000	3.036	2.273	16	28
3	30,000	3.039	2.282	17	29
4	40,000	3.059	2.272	17	29
5	50,000	3.061	2.275	18	26
6	60,000	3.056	2.283	18	33
7	70,000	3.072	2.28	21	30
8	80,000	3.07	2.274	19	27
9	90,000	3.074	2.287	19	32
10	100,000	3.07	2.281	19	30

מספר פעולות האיזון המקסימלי:

נצפה שמספר פעולות האיזון המקסימלי בהכנסה או במחיקה יהיה בסדר גודל של $\log n$ (עד כדי כפל בקבוע). זאת משום שהפעולה היקרה ביותר תתבצע כאשר העץ מלא (כאשר יש בעץ קרוב ל-n איברים), ואחריה צריך לאזן לאורך כל גובה העץ (logn), מהעלה לשורש.
התוצאות תואמות את הציפיות, משום שככל שמספר האיברים בעץ גדל, עלות הפעולה המקסימלית גדלה בצורה לוגריתמית.

מספר פעולות האיזון הממוצע:

נצפה שממוצע פעולות האיזון יהיה שקול לחישוב סיבוכיות amortized של סדרת n הפעולות (או הכנסה או מחיקה). כלומר, מכיוון שבכל סדרת n פעולות יהיו גם פעולות זולות (כשהעץ קטן) וגם יקרות (כשהעץ גדול), נצפה שבממוצע הן "יתקזזו" והממוצע יישאר קבוע.
התוצאות תואמות את הציפיות, שכן נראה שהממוצע תמיד נשאר קבוע ולא תלוי ב-n.

2. Split/join:

מספר סידורי	עלות join ממוצע עבור split אקראי	עלות join מקסימלי עבור split אקראי	עלות join ממוצע עבור split של איבר מקס בתת העץ השמאלי	עלות join מקסימלי עבור split של איבר מקס בתת העץ השמאלי
1	2.428	5	2.909	15
2	2.333	7	2.333	17
3	2.846	6	2.923	18
4	2.357	5	2.923	18
5	2.785	4	2.642	19
6	2.583	5	2.733	19
7	2	5	2.8125	19
8	3	5	2.5625	19
9	2.714	9	2.294	19

10	2.3125	7	2.471	20

פיצול לפי הצומת המקסימלי x בתת העץ השמאלי:

נצפה שבפיצול כזה יתבצעו הרבה פעולות מיזוג זולות (מיזוג העצים שקטנים יותר מהצומת x), משום ש-x הוא תמיד בן ימני ולכן ההפרש בין ה-rank של שני העצים חסום ע"י קבוע. לאחר מכן, תתבצע פעולת מיזוג אחת יקרה – בין תת העץ הימני של x (זהו עץ ריק, כי הצומת הוא המקסימלי בתת העץ) לתת העץ הימני של השורש. הפרש ה-rank הוא כגובה העץ – logn.

לכן, נצפה שהעלות המקסימלית של מיזוג תהיה בערך logn, ושהיא תעלה ככל שגודל העץ עולה (כי יש יותר צמתים ולכן גם הגובה עולה).

התוצאות תואמות את הציפיות, ונראה שגובה העץ אכן עולה בצורה לוגריתמית.

נצפה שהעלות הממוצעת של פעולת מיזוג תהיה שקולה לסיבוכיות ה-amortized של מיזוג, כלומר – לא תהיה תלויה בגובה העץ ותישאר קבועה. זאת משום שפעולות מיזוג זולות "מתקזזות" עם הפעולה היקרה: אם נסמן את גובה העץ ב-h (זהו גם מספר פעולות המיזוג וגם עלות הפעולה היקרה), ואת עלות פעולת מיזוג זולה ב-c (הפרש ה-rank של העצים במתמזגים בפעולה זולה), נקבל סה"כ:

$$\frac{h + c(h-1)}{h} = c + 1 - \frac{c}{h} \leq 2c$$

התוצאות תואמות את הציפיות, שכן הממוצע לא משתנה עם גודל העץ ונשאר קבוע.

פיצול לפי צומת אקראי:

משום שהצומת נבחר באקראי, הסיכוי לפעולה בעלות של גובה העץ הוא קטן – לשם כך הוא צריך להיות הצומת המינימלי או המקסימלי בתת עץ עם rank מאוד גדול. לכן, נצפה שהעלות המקסימלית תהיה נמוכה מגובה העץ, ותשתנה בין עץ לעץ (כי הצומת אקראי).

התוצאות תואמות את הציפיות, שכן העלות המקסימלית של מיזוג נעה בטווח מסוים שקטן משמעותית מגובה העץ.

נצפה שהעלות הממוצעת תהיה דומה לעלות הממוצעת בסעיף הקודם: כלומר תהיה שקולה לסיבוכיות ה-amortized ולא תהיה תלויה בגודל העץ (תישאר קבועה). זאת לפי ההוכחה שראינו בכיתה על כך שפעולת הפיצול היא בסיבוכיות O(logn) ולכן עלות מיזוג אחד תהיה בממוצע O(1).

התוצאות תואמות את הציפיות, משום שהממוצע נשאר קבוע גם כשגודל העץ משתנה.

מהסיבות הללו, העלות הממוצעת של מיזוג בפיצול לפי איבר אקראי היא O(1). אסימפטוטית, הסיכוי לפעולת מיזוג יקרה מאוד (כגובה העץ) עולה ככל שהעץ גדל, לכן הסיבוכיות של העלות המקסימלית של מיזוג היא O(logn).

<u>מסמך תיעוד</u>

<u>המחלקה AVLTree:</u>

זו מחלקה המממשת עץ AVL – עץ חיפוש בינארי מאוזן.

שדות המחלקה:

- IAVLNode root – שורש העץ, מיוצג באמצעות מחלקת צומת AVL.

בנאי המחלקה:

- AVLTree() : יוצר עץ ריק (השורש הוא צומת וירטואלי).
- AVLTree(IAVLNode node) : יוצר עץ מצומת קיים ותת העץ שלו. אם הצומת נמצא בתוך תת עץ של עץ אחר, הבנאי מנתק אותו מהעץ המקורי שלו.

מתודות המחלקה:

- empty() : מתודה בוליאנית שמחזירה true אם העץ ריק (אם השורש שלו לא וירטואלי). המתודה מבצעת בדיקה פשוטה באמצעות המתודה IsRealNode בסיבוכיות O(1).
- search(int k) : מתודה שמקבלת מספר טבעי k ומחזירה את ה-value של הצומת ש-k הוא המפתח שלו אם הוא קיים בעץ, ואחרת מחזירה 1-. המתודה עושה שימוש במתודת העזר treePosition בסיבוכיות O(logn). ובנוסף מבצעת בדיקות פשוטות על השדות של הצומת. בסך הכול הסיבוכיות היא O(logn).
- insert(int k, String i) : מתודה שמקבלת מספר טבעי k ומחרוזת i ומכניסה לעץ צומת חדש כך שהמפתח שלו הוא k וה-value הוא i. המתודה מחזירה מספר טבעי המייצג את כמות פעולות האיזון שנעשו בהכנסת הצומת. המתודה עושה שימוש במתודת העזר treePosition ב-O(logn) למציאת המיקום להכנסת הצומת. לאחר מכן נעשה שימוש במתודת העזר updateSizeUp לעדכון הגדלים בעץ בסיבוכיות O(logn) ולבסוף נקראת הפעולה rebalance של המחלקה Rebalance לביצוע פעולות האיזון בסיבוכיות O(logn). בסך הכול הסיבוכיות O(logn).
- delete(int k) : המתודה מקבלת מספר טבעי k ומוחקת מהעץ את הצומת שיש לו מפתח k. המתודה עושה שימוש במתודת העזר treePosition בסיבוכיות O(logn) למציאת הצומת הדרוש. לאחר מכן, בהתאם למקרה, נקראות מתודות העזר deleteLeaf או deleteNodeWithOneChild המוחקות את הצומת מהעץ בסיבוכיות O(logn), או שבמקרה שלצומת יש 2 ילדים נקראת מתודת העזר successor בסיבוכיות O(logn) ולאחר מכן אחת ממתודות העזר של המחיקה. נקראת הפונקציות updateSizeUp שמעדכנת את הגדלים בעץ בסיבוכיות O(logn) ולבסוף נקראת המתודה rebalance לביצוע פעולות האיזון בסיבוכיות O(logn). בסך הכול הסיבוכיות היא O(logn).
- min() : המתודה מחזירה את ה-value של הצומת בעל המפתח הקטן ביותר בעץ. המתודה עושה שימוש במתודת העזר minNode למציאת הצומת המינימלי בסיבוכיות O(logn).
- max() : המתודה מחזירה את ה-value של הצומת בעל המפתח הגדול ביותר בעץ, ע"י טיול ימינה לאורך כל העץ בסיבוכיות O(logn).
- keysToArray() : המתודה מחזירה מערך המכיל את כל מפתחות הצמתים בעץ מסודרים בסדר עולה. היא קוראת למתודת העזר הרקורסיבית keysArray בסיבוכיות O(n).
- infoToArray() :המתודה מחזירה מערך המכיל את כל ערכי הצמתים בעץ, לפי סדר עולה של מפתחות הצמתים. היא קוראת למתודת העזר הרקורסיבית infoArray בסיבוכיות O(n).

- size() : מתודה שמחזירה את גודל העץ. זהו ערך השמור בשדה size של השורש, שהוא שדה של המחלקה AVLTree ולכן הסיבוכיות היא O(1).
- getRoot() : מתודה שמחזירה את שורש העץ, ששמור כשדה של המחלקה AVLTree, בסיבוכיות O(1).
- split(int k) : המתודה מקבלת מספר טבעי k ומחזירה מערך של שני עצי AVL : smaller, שמכיל עץ עם כל הצמתים הקטנים מ-k, ו-larger, שמכיל עץ AVL עם כל הצמתים הגדולים מ-k. לשם כך, המתודה מטיילת מהצומת שהמפתח שלו k אל השורש ומוסיפה את תתי העצים לעץ החדש המתאים – smaller או larger. כשהיא מטיילת ימינה, היא ממזגת במתודה join את העץ smaller עם צומת ההורה ותת העץ השמאלי שלו, וכשהיא מטיילת שמאלה היא ממזגת במתודה join את העץ larger עם צומת ההורה ותת העץ הימני שלו. הסיבוכיות היא O(logn), לפי ההוכחה שנלמדה בכיתה.
- join(IAVLNode x, AVLTree t) : המתודה מקבלת עץ וצומת כך שכל צמתי העץ גדולים או קטנים מהצומת x, ובהתאמה הצומת x גדול או קטן מכל צמתי העץ עליו מופעלת המתודה. המתודה ממזגת את שני העצים דרך הצומת x ומחזירה את הפרש ה-rank של שני העצים פלוס אחד (זוהי הסיבוכיות של איזון העץ במקרה הגרוע). היא קוראת למתודה moveRight או moveLeft (בהתאם לסימטריה) ב-O(logn), שמוצאת את הצומת שיהיה הבן הימני/השמאלי של x בעץ הגדול יותר. היא ממזגת בין העצים דרך x ע"י קריאה למתודות עזר ב-O(1) ואז קוראת למתודה rebalance כדי לאזן את העץ המאוחד ב-O(logn). הסיבוכיות סה"כ היא O(logn).

מתודות עזר :

- keysArray(int[] arr,IAVLNode dode, int left) : מתודת עזר רקורסיבית למתודה keysToArray שמקבלת מערך של int, צומת בעץ ומספר טבעי, ומעדכנת את המערך כך שיכיל את מפתחות תת העץ של הצומת node בסדר עולה. עבור כל צומת, המתודה מכניסה את המפתח לאינדקס המתאים ואז קוראת לעצמה ברקורסיה על הבן השמאלי והימני. באופן זה היא עוברת על כל צומת בעץ פעם אחת בלבד ולכן הסיבוכיות היא O(n).
- infoArray(String[] arr, IAVLNode node, int left) : מתודת עזר רקורסיבית למתודה infoToArray שמקבלת מערך של String, צומת בעץ ומספר טבעי, ומעדכנת את המערך כך שיכיל את ערכי הצמתים בתת העץ של node לפי סדר עולה של המפתחות. עבור כל צומת, המתודה מכניסה את הערך לאינדקס המתאים ואז קוראת לעצמה ברקורסיה על הבן השמאלי והימני. באופן זה היא עוברת על כל צומת בעץ פעם אחת בלבד ולכן הסיבוכיות היא O(n).
- treePosition(IAVLNode root, int k) : מתודה שמקבלת שורש ומספר טבעי k, ומחזירה את הצומת שנמצא בתת העץ של השורש והמפתח שלו הוא k. אם k לא קיים בעץ, המתודה מחזירה את הצומת האחרון שהיא ראתה בחיפוש (מי שצריך להיות אבא של k). המתודה משתמשת באלגוריתם של חיפוש בינארי בסיבוכיות O(logn).
- minNode(IAVLNode root) : מתודה שמקבלת שורש ומחזירה את הצומת המינימלי בתת העץ שלו. המתודה יורדת את כל גובה העץ, ובעץ AVL הנ"ל שקול לסיבוכיות O(logn).

- successor(IAVLNode node): המתודה מקבלת צומת בעץ ומחזירה את הצומת בעל המפתח העוקב לו. במקרה הגרוע מדובר על בטיול בגובה העץ, לכן הסיבוכיות היא O(logn).
- updateSizeUp(IAVLNode node): המתודה מקבלת צומת בעץ ועוברת החל מהצומת דרך כל האבות עד השורש, ובכל פעם קוראת למתודת העזר updateSize לעדכון הגודל של תת העץ של הצומת. העלייה עד השורש היא בגובה העץ, בסיבוכיות O(logn).
- deleteLeaf(IAVLNode node): המתודה מקבלת צומת עלה ומוחקת אותו מהעץ. המתודה עושה שימוש ב-updateSizeUp לעדכון הגדלים החדשים של אבות הצומת שנמחק, בסיבוכיות O(logn).
- deleteNodeWithOneChild(IAVLNode node): המתודה מקבלת צומת שיש לו בדיוק ילד אחד ומוחקת אותו מהעץ. המתודה עושה שימוש ב-updateSizeUp לעדכון הגדלים של אבות הצומת שנמחק, בסיבוכיות O(logn).
- height(): המתודה מחזירה את גובה העץ, שהוא שדה ששמור בשורש – סיבוכיות O(1).
- moveRight(int rank, int size): מתודת עזר למתודה join שמטיילת כלפי מטה על השלד הימני של העץ ומחזירה את הצומת הראשון שגובהו קטן או שווה ל-rank. לאורך הטיול היא מעדכנת את שדה ה-size של כל צומת דרכו היא עוברת כך שיכלול גם את העץ והצומת שנוספים ב-join. במקרה הגרוע הטיול הוא בסיבוכיות O(logn).
- moveLeft(int rank, int size): מתודת עזר למתודה join שמטיילת כלפי מטה על השלד השמאלי של העץ ומחזירה את הצומת הראשון שגובהו קטן או שווה ל-rank. לאורך הטיול היא מעדכנת את שדה ה-size של כל צומת דרכו היא עוברת כך שיכלול גם את העץ והצומת שנוספים ב-join. במקרה הגרוע הטיול הוא בסיבוכיות O(logn).
- setJoinNodeRight(IAVLNode node): מתודת עזר ל-join שמעדכנת את הבן הימני של הצומת דרכו מתבצע המיזוג ע"י שינויי מצביעים ב-O(1).
- setJoinNodeLeft(IAVLNode node): מתודת עזר ל-join שמעדכנת את הבן השמאלי של הצומת דרכו מתבצע המיזוג ע"י שינויי מצביעים ב-O(1).

<u>המחלקה Rebalance</u>:

מחלקת עזר סטטית המכילה את כל המתודות של פעולות האיזון הנדרשות לעץ וגם את הבדיקות הרלוונטיות.

מתודות המחלקה:

- updateHeight(IAVLNode node): המתודה מעדכנת את הגובה של הצומת שהתקבל לפי גובה הבנים שלו ומחזירה את הפרש הגבהים בין הגובה המקורי לגובה המעודכן. המתודה משתמשת בחישובים פשוטים ולכן הסיבוכיות היא O(1).
- updateSize(IAVLNode node): המתודה מקבלת צומת ומעדכנת את ה-size שלו לפי גודל הבנים שלו. המתודה משתמשת בחישובים פשוטים ולכן הסיבוכיות היא O(1).
- rotateRight(AVLTree tree, IAVLNode node): המתודה מקבלת עץ וצומת ומסובבת את הצומת ימינה, כך שההורה של הצומת יהפוך להיות הבן הימני שלו. היא מעדכנת את גובה וגודל הצומת ע"י קריאה למתודות updateHeight ו-updateSize ב-O(1) ומשנה מספר קבוע

של מצביעים, ולכן הסיבוכיות היא O(1). המתודה מחזירה את מספר פעולות האיזון שהתבצעו בגלגול.

- rotateLeft(AVLTree tree, IAVLNode node): המתודה מקבלת עץ וצומת ומסובבת את הצומת שמאלה, כך שההורה של הצומת יהפוך להיות הבן שמאלי שלו. היא מעדכנת את גובה וגודל הצומת ע"י קריאה למתודות updateHeight ו-updateSize ב-O(1) ומשנה מספר קבוע של מצביעים, ולכן הסיבוכיות היא O(1). המתודה מחזירה את מספר פעולות האיזון שהתבצעו בגלגול.
- rotateRL(AVLTree tree, IAVLNode node): המתודה מקבלת עץ וצומת ומסובבת אותו ימינה ואז שמאלה ע"י קריאה למתודות rotateRight ו-rotateLeft. אלו מתודות ב-O(1) ולכן הסיבוכיות היא O(1). המתודה מחזירה את מספר פעולות האיזון שהתבצעו בשני הגלגולים.
- rotateLR(AVLTree tree, IAVLNode node): המתודה מקבלת עץ וצומת ומסובבת אותו שמאלה ואז ימינה ע"י קריאה למתודות rotateLeft ו-rotateRight. אלו מתודות ב-O(1) ולכן הסיבוכיות היא O(1). המתודה מחזירה את מספר פעולות האיזון שהתבצעו בשני הגלגולים.
- balanced(IAVLNode node): מתודה שמקבלת צומת בעץ ובודקת אם הוא מאוזן, כלומר אם יחס הגבהים בין הצומת לילדיו חוקי בעץ AVL – סיבוכיות O(1).
- requiresPromote(int leftDiff, int rightDiff): המתודה מקבלת את הפרשי הצמתים בין צומת לילדיו ובודקת אם צריך לבצע על הצומת פעולת promote – סיבוכיות O(1).
- requiresDemote(int leftDiff, int rightDiff): המתודה מקבלת את הפרשי הצמתים בין צומת לילדיו ובודקת אם צריך לבצע על הצומת פעולת demote – סיבוכיות O(1).
- requiresRotateRight(int leftDiff, int rightDiff, IAVLNode leftChild): המתודה מקבלת את הפרשי הצמתים בין צומת לילדיו וגם את הילד השמאלי של הצומת ובודקת אם צריך לבצע על הצומת פעולת rotateRight – סיבוכיות O(1).
- requiresRotateLeft(int leftDiff, int rightDiff, IAVLNode rightChild): המתודה מקבלת את הפרשי הצמתים בין צומת לילדיו וגם את הילד הימני של הצומת ובודקת אם צריך לבצע על הצומת פעולת rotateLeft – סיבוכיות O(1).
- requiresLR(int leftDiff, int rightDiff, IAVLNode leftChild): המתודה מקבלת את הפרשי הצמתים בין צומת לילדיו וגם את הילד השמאלי של הצומת ובודקת אם צריך לבצע על הצומת פעולת rotateLR – סיבוכיות O(1).
- requiresRL(int leftDiff, int rightDiff, IAVLNode rightChild): המתודה מקבלת את הפרשי הצמתים בין צומת לילדיו וגם את הילד הימני של הצומת ובודקת אם צריך לבצע על הצומת פעולת rotateRL – סיבוכיות O(1).
- rebalance(AVLTree tree, IAVLNode node): המתודה משתמשת בכל מתודות הבדיקה הנ"ל וקוראת בהתאם לצורך לכל מתודות האיזון/סיבוב הנדרשות עד שהעץ מאוזן.
  הסיבוכיות היא O(logn) משום שבמקרה הגרוע האיזון יימשך עד שורש העץ.

<u>המחלקה AVLNode (מממשת את המנשק IAVLNode):</u>

מחלקה המייצגת צומת בעץ AVL וממשת את המנשק IAVLNode. כל המתודות בה נמצאות גם במנשק, וממומשות באמצעות בדיקות פשוטות בסיבוכיות O(1).

שדות המחלקה:

- int key – המפתח של הצומת.
- String value – המידע של הצומת.
- IAVLNode right – הבן הימני של הצומת.
- IAVLNode left – הבן השמאלי של הצומת.
- int height – הגובה/הדרגה של הצומת.
- int size – גודל תת העץ של הצומת.
- IAVLNode parent – ההורה של הצומת.

בנאי המחלקה:

- AVLNode() : יוצר צומת וירטואלי.
- AVLNode(int key, String value) : יוצר צומת עם מפתח key וערך value.

מתודות המחלקה:

- getKey() – מחזירה את המפתח של הצומת.
- getValue() – מחזירה את המידע של הצומת.
- setLeft(IAVLNode node) : מגדירה את הבן השמאלי של הצומת ל-node.
- getLeft() : מחזירה את הילד השמאלי של הצומת, או null אם לא קיים כזה.
- myGetLeft() : מחזירה את הילד השמאלי של הצומת (בין אם אמיתי או וירטואלי).
- setRight(IAVLNode node) : מגדירה את הבן הימני של הצומת ל-node.
- getRight() : מחזירה את הילד הימני של הצומת, או null אם לא קיים כזה.
- myGetRight() : מחזירה את הילד הימני של הצומת (בין אם אמיתי או וירטואלי).
- setParent(IAVLNode node) : מגדירה את ההורה של הצומת ל-node.
- getParent() : מחזירה את ההורה של הצומת.
- isRealNode() : מחזירה true אמ"ם הצומת הוא אמיתי (לא וירטואלי), false אחרת.
- setHeight(int newHeight) : מגדירה את הגובה/דרגה של הצומת ל-newHeight.
- getHeight() : מחזירה את הגובה/דרגה של הצומת.
- setSize(int newSize) : מגדירה את גודל תת העץ של הצומת ל-newSize.
- getSize() : מחזירה את גודל תת העץ של הצומת.
- isALeaf() : מחזירה true אמ"ם הצומת הוא עלה בעץ, false אחרת.
- isLeftChild() : מחזירה true אמ"ם הצומת הוא ילד שמאלי, false אחרת.
- isRightChild() : מחזירה true אמ"ם הצומת הוא ילד ימני, false אחרת.