**צומת**

פונקציות מחלקה:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| אינדקס | פונקציה | תיאור הפונקציה | פונקציות עזר בשימוש | סיבוכיות זמן ריצה | הסבר לסיבוכיות |
|  | \_\_init\_\_(self, key, value) |  |  | O(1) |  |
|  | get\_key(self) |  |  | O(1) |  |
|  | get\_value(self) |  |  | O(1) |  |
|  | get\_left(self) |  |  | O(1) |  |
|  | get\_right(self) |  |  | O(1) |  |
|  | get\_parent(self) |  |  | O(1) |  |
|  | get\_height(self) |  |  | O(1) |  |
|  | get\_size(self) |  |  | O(1) |  |
|  | set\_key(self, key) |  |  | O(1) |  |
|  | set\_value(self, value) |  |  | O(1) |  |
|  | set\_left(self, node) |  |  | O(1) |  |
|  | set\_right(self, node) |  |  | O(1) |  |
|  | set\_parent(self, node) |  |  | O(1) |  |
|  | set\_height(self, h) |  |  | O(1) |  |
|  | set\_size(self, s) |  |  | O(1) |  |

פונקציות עזר:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| אינדקס | פונקציה | תיאור הפונקציה | פונקציות עזר בשימוש | סיבוכיות זמן ריצה | הסבר לסיבוכיות |
| 1 | create\_leaf\_with\_virtual\_nodes(key, value) | יוצר עלה בעץ שבניו וירטואליים |  | O(1) |  |
| 2 | compute\_balance\_factor(self) | מחשב את הBF של הצומת |  | O(1) |  |
| 3 | is\_empty\_node(node) | בודק האם הצומת היא None או צומת וירטואלי |  | O(1) |  |
| 4 | is\_leaf(self) | בודק האם הצומת הוא עלה |  | O(1) |  |
| 5 | has\_one\_child(self) | בודק האם לצומת ילד אחד בלבד |  | O(1) |  |
| 6 | is\_left\_child\_of\_parent(node) | בודק האם הצומת היא הבן השמאלי האב שלו |  | O(1) |  |
| 7 | update\_parents\_child(self, old\_child, new\_child) | בודקת אם old\_child הוא הבן הימני או השמאלי של self, ושמה את new\_child במקומו. |  | O(1) |  |
| 8 | is\_real\_node(self) | בודק אם הצומת הוא "אמיתי" או וירטואלי |  | O(1) |  |

**עץ**

פונקציות מחלקה:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| אינדקס | פונקציה | תיאור הפונקציה | פונקציות עזר בשימוש | סיבוכיות זמן ריצה | הסבר לסיבוכיות |
|  | \_\_init\_\_(self) |  |  |  |  |

פונקציות עזר:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| אינדקס | פונקציה | תיאור הפונקציה | פונקציות עזר בשימוש | סיבוכיות זמן ריצה | הסבר לסיבוכיות |
| 9 | set\_root(self, node) |  |  | O(1) |  |
| 10 | rotate\_right(self, old\_root: AVLNode) |  | 7 | O(1) |  |
| 11 | rotate\_left(self, old\_root: AVLNode) |  | 7 | O(1) |  |
| 12 | rotate\_left\_then\_right(self, old\_root: AVLNode) |  | 10,11 |  |  |
| 13 | rotate\_right\_then\_left(self, old\_root: AVLNode) |  | 10,11 |  |  |
| 14 | find\_successor\_for\_node\_with\_two\_childs(self, node: AVLNode) |  | 3 | O(logn) כגובה העץ |  |
| 15 | find\_parent\_for\_insert(self, key) |  | 8 | O(logn) כגובה העץ |  |
| 16 | find\_parent\_with\_illegal\_balance\_factor(self, node: AVLNode) | מחפשת צומת במעלה העץ (לכיוון השורש) שיש לו balance factor לא תקין. | 2 | O(logn) כגובה העץ | ״עולים למעלה בעץ״ מהצומת הנתון ועד צומת עם balance factor לא תקין. בכל צומת מבצעים פעולה בסיבוכיות O(1). המקרה הכי גרוע הוא שהמסלול הוא בגובה העץ – O(logn) ולכן הסיבוכיות של הפונקציה היא O(logn). |
| 17 | fix\_tree\_of\_illegal\_root(self, illegal\_root) | מתקנת את תת העץ שהשורש שלו הוא צומת עם balance factor לא תקין, באמצעות גילגולים. | 2, 10, 11, 12, 13 | O(1) |  |
| 18 | fix\_tree(self, node\_with\_illegal\_balance\_factor) | מתקנת את העץ ל-AVL לאחר שבוצעו בו שינויים, מצומת שעלול להיות בעייתי ולמעלה עד לשורש העץ | 16, 17 | O(logn) כגובה העץ | ״עולים למעלה בעץ״ (ה״עלייה״ עצמה מתבצעת בפונקציה 16 שמחפשת צומת עם balance factor לא תקין), אך במהלך כל הפונקציה עוברים על כל צומת במסלול למעלה רק פעם אחת. עבור כל צומת מבצעים פעולה ב-O(1). המסלול הארוך ביותר הוא מסלול בגובה העץ – O(logn), וזוהי הסיבוכיות של הפונקציה. |
| 19 | physical\_insert(self, leaf\_for\_insert: AVLNode) |  | 15 | O(logn) כגובה העץ |  |
| 20 | physical\_delete(self, node: AVLNode) |  | 8, 6 | O(1) |  |
| 21 | replace\_node\_in\_tree(self, old\_node: AVLNode, new\_node: AVLNode) |  | 9, 7 | O(1) |  |
| 22 | avl\_to\_array\_rec(self, node: AVLNode, array) |  | 3, 22 | O(n) |  |
| 23 | create\_tree(self, root) | יוצרת עץ AVL מהתת עץ שהשורש שלו הוא הצומת הנתון. | 9 | O(1) |  |
| 24 | join\_trees\_with\_equal\_heights(self, t1, t2, x) | מאחדת שני עצים כששניהם באותו הגובה. במקרה זה אין צורך לבצע חיפוש, כל עץ הופך לבן של הצומת המקשר. |  | O(1) |  |
| 25 | join\_trees\_left\_tree\_is\_smaller(self, t1, t2, x) | איחוד עצים כשהעץ עם הערכים הקטנים הוא ״נמוך״ יותר מהעץ עם הערכים הגדולים. | 18 | O(logn) כגובה העץ | עוברים על הצמתים הכי שמאליים בעץ הגבוה, במקרה הגרוע עוברים על גובה עץ בגודל n שהוא O(logn).  בנוסף הקריאה ל-fix\_tree (18) היא בסיבוכיות O(logn). |
| 26 | join\_trees\_right\_tree\_is\_smaller(self, t1, t2, x) | איחוד עצים כשהעץ עם הערכים הגדולים הוא ״נמוך״ יותר מהעץ עם הערכים הקטנים. | 18 | O(logn) כגובה העץ | עוברים על הצמתים הכי ימניים בעץ הגבוה, במקרה הגרוע עוברים על גובה עץ בגודל n שהוא O(logn).  בנוסף הקריאה ל-fix\_tree (18) היא בסיבוכיות O(logn). |
| 27 | join\_tree\_with\_array\_of\_node\_tuples(self, array\_of\_node\_tuples) | מקבלים רשימה של זוגות – כל זוג מכיל צומת מקשר ועץ. הפונקציה מאחדת את self עם העצים ברשימה כשהצמתים המקשרים מקשרים ביניהם. | קורא פונקציה join, 23 | O(logn) כגובה העץ | יש קריאה ל-join O(logn) פעמים עם תתי עצים שיש לאחד לצורך מימוש split. מכיוון שתתי העצים אינם תתי עצים שכל אחד מהם הוא בגובה O(logn), כי הם נוצרו בתהליך split של עץ יחיד, נסיק מניתוח הסיבוכיות שעשינו בשיעור שהסיבוכיות של הפונקציה היא O(logn). |

פונקציות שלהם:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| אינדקס | פונקציה | תיאור הפונקציה | פונקציות עזר בשימוש | סיבוכיות זמן ריצה | הסבר לסיבוכיות |
| 1 | search(self, key) |  | 8 | O(logn) כגובה העץ |  |
| 2 | insert(self, key, val) |  | 1, 9, 18, 19 | O(logn) כגובה העץ |  |
| 3 | delete(self, node: AVLNode) |  | 4, 5, 20, 14, 21, 18 | O(logn) כגובה העץ |  |
| 4 | avl\_to\_array(self) |  | 22 | O(n) |  |
| 5 | size(self) |  |  | O(1) |  |
| 6 | join(self, tree, key, val) | מאחדת שני עצים עם צומת מקשר ביניהם, כאשר  - בעץ אחד כל המפתחות קטנים מהצומת המקשר  - ובעץ השני כל המפתחות גדולים מהצומת המקשר | 1, 8, 24, 25, 26 | O(logn) כגובה העץ | קוראים פעם אחת לאחת מבין הפונקציות 24, 25, 26, כאשר הסיבוכיות של 25 ו-26 היא O(logn) |
| 7 | split(self, node: AVLNode) | מפצלת עץ לשני עצים לפי הצומת הנתון:  - עץ אחד מכיל את הצמתים עם המפתחות הקטנים מהמפתח של הצומת שקיבלנו - העץ השני מכיל את הצמתים עם המפתחות הגדולים מהמפתח של הצומת שקיבלנו | 23, 6, 27 | O(logn) כגובה העץ | ״עולים למעלה״ בעץ מהצומת שקיבלנו עד השורש – לולאה זו במקרה הגרוע היא בסיבוכיות גובה העץ – O(logn).  קוראים לפונקציה 27 פעם אחת, והסיבוכיות שלה היא O(logn). |
| 8 | rank(self, node) | מחזירה את ה״מיקום״ של הצומת כשממיינים את המפתחות של העץ לפי הסדר, כלומר את מספר האיברים בעץ שקטנים מהצומת שקיבלנו + 1. |  | O(logn) כגובה העץ | מבצעים מעבר יחיד על המסלול בעץ שמוביל לצומת הנתון, לכן במקרה הגרוע נעבור על מסלול שהוא בגובה העץ – O(logn), ולכן זו הסיבוכיות של הפונקציה. |
| 9 | select(self, i) | מחזירה את הצומת שה-rank שלו הוא i. |  | O(logn) כגובה העץ | מבצעים מעבר יחיד על המסלול בעץ שמוביל לצומת עם ה-rank הנתון, לכן במקרה הגרוע נעבור על מסלול שהוא בגובה העץ – O(logn), ולכן זו הסיבוכיות של הפונקציה. |
| 10 | get\_root(self) | מחזירה את השורש של העץ |  | O(1) |  |