מחלקת AVLNode

|  |  |  |
| --- | --- | --- |
| שדה (חבר) | ערך דיפולטי | תפקיד |
| Key | none | מפתח של הצומת |
| Value | none | ערך של הצומת |
| left | none | צומת שמאלי (none אם לא קיים כזה) |
| Right | none | צומת ימני (none אם לא קיים כזה) |
| Parent | none | צומת הורה (none אם שורש ולא קיים הורה) |
| height | -1 | גובה (מספר הקשתות מהעלה הכי קרוב). -1 אם עץ וירטואלי |
| size | 0 | גודל תת העץ של הצומת (0 אם עץ וירטואלי) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | תפקיד | ערך החזרה | תיאור הלוגיקה | סיבוכיות |
| Get\_key(self) | מחזיר את המפתח של הצומת self (none אם וירטואלי) | Self.key |  |  |
| Get\_value(self) | מחזיר את הערך של הצומת (none אם וירטואלי) | Self.value |  |  |
| Get\_left(self) | מחזיר את הצומת השמאלי (none אם לא קיים כזה) | Self.left |  |  |
| Get\_right(self) | מחזיר את הצומת הימני (none אם לא קיים כזה) | Self.right |  |  |
| Get\_parent(self) | מחזיר את ההורה (none אם לא קיים כזה) | Self.parent |  |  |
| Get\_height(self) | מחזיר את הגובה של תת העץ של הצומת (-1 אם וירטואלי) | Self.height |  |  |
| Get\_size(self) | מחזיר את הגובה של תת העץ של הצומת (0 אם וירטואלי) | Self.size |  |  |
| Set\_key(self, key) | מגדיר מפתח של צומת | self |  |  |
| Set\_value(self, value) | מגדיר ערך של צומת | Self |  |  |
| Set\_left(self, node) | מגדיר צומת כבן שמאלי | Self |  |  |
| Set\_right(self, node) | מגדיר צומת כבן ימני | Self |  |  |
| Set\_parent(self, node) | מגדיר צומת כהורה | Self |  |  |
| Set\_height(self, height) | מגדיר גובה לתת העץ של הצומת | Self |  |  |
| Set\_size(self, size) | מגדיר גודל לתת העץ של הצומת | Self |  |  |
| Is\_real\_node(self) | מחזיר האם הצומת אמיתי או וירטואלי | Self.key!=none |  |  |
|  |  |  |  |  |

מחלקת AVLTree

|  |  |  |
| --- | --- | --- |
| שם שדה (חבר) | ערך דיפולטי | תפקיד |
| root | none | שורש העץ |
|  |  |  |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם הפונקציה | תפקיד | ערך החזרה | לוגיקה | סיבוכיות |
| Search(self,key) | מחזיר מצביע לצומת המתאים למפתח | Node (none אם לא קיים כזה) | 1. אם העץ ריק נחזיר none 2. נבדוק האם שורש העץ הוא הערך הנדרש    1. אם כן נחזיר אותו    2. אם הערך הנדרש קטן יותר מהשורש, נחזור לסעיף 1 עם הבן השמאלי    3. אם הערך הנדרש גדול יותר נחזור לסעיף 1 עם הבן הימני | כאשר n גודל העץ. זו הסיבוכיות במקרה הגרוע בגלל שגובה העץ חסום על ידי  *כפי שראינו בהרצאה* |
| Insert\_node\_bst(self, key, val) | הכנסה לעץ חיפוש בינארי רגיל  כולל שינוי הsize, height של הצמתים המושפעים מהתהליך | מצביע לצומת החדש שנוצר | 1. אם העץ ריק – מכניס את הצומת כשורש והגדל את גודל הצומת הנוכחי ב1  2. אם העץ צומת יחיד- הכנס שמאלה אם קטן יותר או ימינה אם גדול יותר והגדל את גודל הצומת הנוכחי ב1  3. אחרת  3.1 אם הצומת בעל מפתח קטן יותר, חזור לסעיף 2 עם תת העץ השמאלי  3.2 אם הצומת בעל מפתח גדול יותר, חזור לסעיף 2 עם תת העץ הימני  4. אם הצומת נכנס כבן יחיד להורה שלו, הגדל ב1 את גובה הצמתים במסלול שלו לשורש שגובהם השתנה | כאשר n גודל העץ. זו הסיבוכיות במקרה הגרוע בגלל שגובה העץ חסום על ידי  *כפי שראינו בהרצאה* |
| Calculate\_BF(self, node | מחשב את הBF של הצומת | BF | גובה תת העץ השמאלי – גובה תת העץ הימני |  |
| Is\_height\_changed(self, node, son\_value) | נקרא לאחר הכנסה של צומת עבור צמתים במסלול מההכנסה עד השורש, ומחזיר האם הכנסת הצומת גררה שינוי גובה | bool | בדיקה מאיזה כיוון של הצומת הייתה ההכנסה (נניח בהכ ימין).  אם הגובה מימין גדול ממש מהגובה משמאל (אז ההכנסה מימין אכן גרמה לשינוי בגבוה) החזר אמת |  |
| rotate(self, criminal\_node, criminal\_node\_bf, criminal\_son\_bf) | מקבלת את הצומת ה"עבריין" ואת הBF שלו ושל הבן שלו (הרלוונטי) ומפעילה את הגלגול הימני | מספר פעולות הגלגול שבוצעו | קריאה לסוג הגלגול הרלוונטי כפי שהוגדר בהרצאה –  2, 1 – right  2, -1 – left then right  -2, -1 – left  -2, 1 – right then left |  |
| Right\_rotation(self,node) | מבצעת גלגול ימינה | None | 1.גלגול ימינה כפי שהוצג בהרצאה  2.תיקון הsize של הצומת שממנו התחיל הגלגול ושל הצומת שהחליף אותו (אלה היחידים שהשתנה להם הגודל)  3.תיקון הגבהים של הצומת שממנו התחיל הגלגול ושל הצמתים מהמסלול ממנו לשורש (אלה המקומות היחידים שהיה יכול להיות בהם שינוי) | סעיפים 1 ו2 מכילים מספר קבוע של פעולות.  סעיף 3 יכול במקרה הכי גרוע לבצע לולאה על כל המסלול לשורש, ובגלל שהמסלול חסום על ידי O(logn) זאת הסיבוכיות הכוללת |
| Left\_rotation(self,node) | מבצעת גלגול שמאלה | None | אנלוגי לגלגול ימינה (עד כדי השינויים בכיוונים) | זהה לגלגול ימינה |
| Insert(self, key, val) | הכנסה לעץ AVL (כולל הכנסה תוך שמירה על דרישות העץ בעזרת גלגולים) | מספר פעולות הגלגול שבוצעו | לפי הלוגיקה שהוצגה בהרצאה:  1.הכנסה לעץ חיפוש בינארי רגילה  2.כל עוד ההורה של הצומת המוכנס קיים  2.1 חשב את הBF של ההורה  2.2 אם הBF קטן ממש מ2 וגם ההורה לא שינה גבוה – סיים  2.3 אם הBF קטן ממש מ2 אבל ההורה כן שינה גובה – קרם את ההורה להיות ההורה שלו וחזור לסעיף 2  2.4 אם הBF (בערך מוחלט) שווה ל2 בצע גלגול וסיים | סעיף 1 לוקח O(logn) כפי שלמדנו בכתה  סעיף 2 יוצר לולאה שתרוץ לכל היותר O(logn) פעמים כי זה גובה העץ.  בתוך הלולאה, שלושת הסעיפים הראשונים הם בסיבוכיות קבועה ולכן לא משפיעים.  סעיף 2.3 בסיבוכיות O(logn) אבל אם הוא מתבצע זה פעם אחת בלבד.  לכן סה"כ  (הכנסה לעץ, ריצה על המסלול עד לשורש, לכל היותר גלגול אחד) |