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 נמוך). |
| size | 0 | גודל תת העץ של הצומת (כולל עצמו) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |  |  |
| Convert\_node\_to\_real(self) | נקראת לאחר יצירת צומת חדש כדי להפוך אותו לצומת תקין שאינו וירטואלי |  | מגדירה ילדים וירטואלים (ומקשרת אותם להורה שלהם)  מגדירה גובה 0  מגדירה גודל 1 |  |

מחלקת AVLTree

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

פונקציות נדרשות -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | מצביע לצומת החדש שנוצר | 1. אם העץ ריק – הכנס את הצומת כשורש  2. אם העץ צומת יחיד- הכנס שמאלה אם קטן יותר או ימינה אם גדול יותר  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 | 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) |

פונקציות עזר –

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם | תפקיד | ערך החזרה | תיאור לוגיקה | סיבוכיות |
| Insert\_node\_bst(self, key, val) | הכנסה לעץ חיפוש בינארי רגיל | מצביע לצומת החדש שנוצר | 1. אם העץ ריק – הכנס את הצומת כשורש  2. אם העץ צומת יחיד- הכנס שמאלה אם קטן יותר או ימינה אם גדול יותר  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 | כאשר n גודל העץ. זו הסיבוכיות במקרה הגרוע בגלל שגובה העץ חסום על ידי  *כפי שראינו בהרצאה* |
| Delete\_leaf(self, node) | מחיקה של עלה | none | מגדירים להורה של node ילדים וירטואלים |  |
| Delete\_one\_child(self, node) | מחיקה של צומת שיש לו בן אחד וירטואלי ובן שני עם תוכן | none | מחברים את ההורה של הצומת לבן היחיד של הצומת |  |
| Get\_successor(self,node) | מחזירה את הצומת בעל הערך הגדול הכי קרוב לצומת. אם הצומת הוא המקסימלי בעץ – מחזירים none | AVLNode | אם יש לצומת בן ימני – הולכים ימינה ועד הסוף שמאלה  אחרת – עולים למעלה עד הפנייה הראשונה ימינה.  (זהה למימוש שהוצג בהרצאה | זו הסיבוכיות במקרה הגרוע בגלל שגובה העץ חסום על ידי  *כפי שראינו בהרצאה* |
| Delete\_node\_bst(self, node) | מחיקה מעץ חיפוש בינארי | none | אם הצומת עלה – מוחקת בעזרת הפונקציה delete\_leaf  אם הצומת בעל בן יחיד – מוחקים בעזרת delete\_one\_child  אחרת – מוצאים את הsuccessor של הצומת, מוחקים אותו (הוא בהכרח יהיה עלה או צומת בעל בן יחיד) ומציבים במקומו את התוכן של הצומת | זו הסיבוכיות במקרה הגרוע כיוון שלכל היותר נבצע פעולת successor אחת שזו העלות שלה. |
| Calculate\_bf(self, node) | חישוב BF של הצומת | BF | הפרש הגבהים בין הבן השמאלי לבן הימני | כי גובה הוא שדה של הצומת |
| Update(self, node) | מעדכנת שדות גובה וגודל של הצומת הנתון (לפי ערכי ילדיו) | none | מגדירים גודל להיות 1+ סכום גדלי הילדים  מגדירים גובה להיות 1+ מקסימום גבהי הילדים | משתמשת בשליפת שדות בלבד |
| Rotate(self, criminal\_node, criminal\_node\_bf) | מבצעת גלגול/ים מתאים מהצומת העבריין | מספר פעולות הגלגול שבוצעו | מפנה לגלגול שמאלה/ימינה/גם וגם לפי הערכים שהוצגו בהרצאה.  למשל – עבור bf=2 של הצומת וbf=1 של הבן השמאלי מבצעים גלגול ימינה | מבצעת לכל היותר שתי פעולות גלגול שגם הן בסיבוכיות הזאת. |
| Right\_rotation(self, node) | מבצעת גלגול אחד ימינה | none | שינוי מצביעים כך שהבן השמאלי של הצומת "יעלה" למעלה, הצומת יהפוך להיות בנו הימני (והילדים שלהם זזים בהתאם).  קריאה ל update עבור הצומת ובנו השמאלי (כי הם היחידים שעשויים לשנות גובה) | הזזת מצביעים (מספר פעולות קבוע) וקריאה לupdate שהוא גם |
| Left\_rotation(self, node) | מבצעת גלגול אחד שמאלה | none | סימטרי לגלגול ימינה | מאותו שיקול של גלגול ימינה |
| Inorder\_rec(self, node, avl\_list) | ממלאת את הרשימה באיברי העץ בסריקת inorder | None  אין צורך להחזיר כי הפונקציה עורכת את הרשימה | קוראים לפונקציה עם הבן השמאלי של הצומת  מכניסים לרשימה את הצומת  קוראים לפונקציה עם הבן הימני של הצומת | עוברים על כל צומת בעץ פעם אחת ומבצעים בו עבודה של |
| Select\_rec(self, node, i) | מחפשת את הצומת במיקום ה- i בתת העץ של node | AVLNode | אם node הוא הצומת הנדרש, מחזירים אותו.  אם i קטן מהמיקום של node - קוראים רקורסיבית לפונקציה שוב עם תת העץ השמאלי שלו.  אחרת – קוראים לפונקציה עם תת העץ הימני (ומחפשים ערך i מתאים) | נבצע לכל היותר ירידה אחת מלאה במורד העץ, וגובהו חסום על ידי logn |
|  |  |  |  |  |