תיעוד פונקציות

|  |  |  |  |
| --- | --- | --- | --- |
| שם המחלקה | שם הפונקציה | סיבוכיות | דרך פעולה |
| AVLTree | AVLTree | O(1) | זהו הבנאי של המחלקה. למחלקה זו ארבע שדות – שורש העץ, שהוא צומת, גודל העץ שהוא מספר שלם אי-שלילי ששומר את מספר הצמתים בעץ, min ו - max ששומרים מצביעים לצמתי המינימום או המקסימום אם קיימים ו-null אחרת. |
| AVLTree | setRoot | O(1) | הפונקציה מקבלת צומת, ומעדכנת את העץ כך שצומת הקלט תהפוך להיות שורש העץ. |
| AVLTree | empty | O(1) | הפונקציה בודקת האם העץ ריק, ומחזירה true אם אכן העץ ריק |
| AVLTree | Search | O(log(n)) | הפונקציה מקבלת מספר, ובודקת האם ישנה צומת בעץ שהמפתח שלה הוא בגודלו של מספר הקלט. הפונקציה קוראת לפונקציית עזר, search\_node שבאופן רקורסיבי מחפשת את הצומת בעץ החיפוש לפי גודל המפתח. הסיבוכיות היא O(log(n)) מכיוון שלכל היותר אנו נצטרך לבדוק צמתים כגובה העץ אשר חסום ע"י O(log(n)). אם הצומת קיים בעץ, הפונקציה תחזיר את המידע השמור בו, ואם הצומת לא קיים בעץ הפונקציה תחזיר null. |
| AVLTree | Search\_node | O(log(n)) | הפונקציה מקבלת את שורש העץ ואת המפתח שאנו מחפשים. הפונקציה פועלת באופן רקורסיבי ובכל שלב משווה בין גודל המפתח למפתח של הצומת. אם מפתח הקלט שווה למפתח הצומת, נחזיר את הצומת. אם הוא קטן יותר, נתקדם אל הבן השמאלי, ואילו אם הוא גדול יותר נתקדם אל הבן הימני. אחרי כל התקדמות נקרא שוב לפונקציה. אם הגענו לצומת שערכה null, ולא מצאנו את הצומת אז הצומת לא קיימת בעץ ונחזיר null. הסיבוכיות היא O(log(n)) שכן אנו נצטרך לבדוק לכל היותר מספר צמתים כגובה העץ אשר חסום ע"י O(log(n)). |
| AVLTree | Min | O(1) | מחזירה את הטקטסט השמור בצומת בעלת המפתח המינימאלי (שמור מצביע) |
| AVLTree | Max | O(1) | מחזירה את הטקטסט השמור בצומת בעלת המפתח המקסימאלי (שמור מצביע) |
| AVLTree | rightRotation | O(1) | הפונקציה מבצעת גלגול ימינה בכדי לשמור על העץ מאוזן. הפונקציה מקבלת צומת, והגלגול מתבצע ע"י החלפת המצביעים הרלוונטיים. |
| AVLTree | leftRotation | O(1) | הפונקציה מבצעת גלגול שמאלה בכדי לשמור על העץ מאוזן. הפונקציה מקבלת צומת, והגלגול מתבצע ע"י החלפת המצביעים הרלוונטיים. |
| AVLTree | calcHeight | O(1) | הפונקציה מחשבת את הגובה של צומת. היא מקבלת כקלט צומת, והגובה שלה הוא המקסימום בין הגבהים של הבנים שלה ועוד 1. |
| AVLTree | rightThenLeftRotation | O(1) | הפונקציה קוראת לפונקציות הגלגול שכבר מימשנו ומבצעת גלגול ימינה ולאחר מכן גלגול שמאלה. |
| AVLTree | leftThenRightRotation | O(1) | הפונקציה קוראת לפונקציות הגלגול שכבר מימשנו ומבצעת גלגול שמאלה ולאחר מכן גלגול ימינה. |
| AVLTree | getBalanceFactor | O(1) | הפונקציה מחשבת את מקדם האיזון של הצומת. אנו מגדירים את מקדם האיזון להיות ההפרש בין גובהו של תת העץ השמאלי לגובהו של תת העץ הימני. לפי גודלו של מקדם האיזון אנו יכולים לדעת האם תת העץ של הצומת מאוזן, ואם הוא לא מאוזן, אילו גלגולים אנו צריכים לבצע בכדי לאזן אותו. הפונקציה מקבלת צומת ומחזירה את מקדם האיזון שלו, שהוא מספר. |
| AVLTree | Insert | O(log(n)) | הפונקציה מקבלת מספר ומחרוזת, מייצר צומת שהמפתח שלו הוא המספר והמידע שלו היא המחרוזת, ומכניסה את הצומת לעץ. הפונקציה נעזרת בפונקציה insert\_rec בשביל ההכנסה. לאחר ההכנסה היא נעזרת בפונקציה calcHeightLeaf, בשביל לעדכן את הגבהים, ולאחר מכן בפונקציה keepBalanced בשביל לאזן את העץ בעזרת גלגולים. הסיבוכיות של הפונקציה היא סכום הסיבוכיות של פונקציות העזר. הסיבוכיות של שלוש פונקציות העזר היא O(log(n)), והסיבוכיות של הפונקציה היא  3 O(log(n)) = O(log(n)) |
| AVLTree | insert\_rec | O(log(n)) | הפונקציה מקבלת צומת ואת שורש העץ ומכניסה אותה לעץ החיפוש. הפונקציה פועלת באופן רקורסיבי. אם ההצומת שאנו נמצאים בה בעץ היא וירטואלית אז הצומת נכנסת במקומה, זהו תנאי העצירה. עבור כל צומת נבצע השוואה, אם המפתח של צומת הקלט גדול מהצומת הנוכחית נתקדם ימינה, אחרת נתקדם שמאלה. לאחר כל התקדמות נקרא שוב לפונקציה עם הצומת הנוכחית. הסיבוכיות היא O(log(n)) שכן אנו נצטרך לבדוק לכל היותר מספר צמתים כגובה העץ אשר חסום ע"י O(log(n)). |
| AVLTree | updateMinMaxInsert | O(1) | מעדכנת את המצביעים min ו-max בהתאם לאחר כל פעולת הכנסה. |
| AVLTree | nodeKind | O(1) | הפונקציה מקבלת צומת ומסווגת אותו לאחד משלושה סוגים. סוג 1 – אם הצומת הוא עלה, סוג 2 – אם לצומת יש בן 1, סוג 3 – אם לצומת יש שני בנים. הסוג יהיה לנו חשוב כשנרצה למחוק צומת מהעץ. הסיבוכיות היא O(1) שכן אנו רק בודקים את השדות של הצומת. ומבצעים מספר קבוע של בדיקות. |
| AVLTree | calcHeightLeaf | O(log(n)) | הפונקציה מקבלת צומת ומעדכנת את הגבהים של כל הצמתים שהיו יכולים להיות מושפעים מהגובה של צומת זו. הפונקציה עולה מהצומת עד השורש ומעדכנת את הגובה של כל הצמתים במסלול זה בעזרת calcHeight. נצטרך לחשב את הגובה (מספר קבוע של פעולות) לכל היותר O(log(n)) פעמים (כגובה העץ) ולכן הסיבוכיות היא O(log(n)). |
| AVLTree | keepBalanced | O(log(n)) | הפונקציה מקבלת צומת ומחשבת את מקדם האיזון שלה. אם מקדם האיזון תקין, הלולאה ממשיכה ומתקדמת להורה של הצומת. אם מקדם האיזון לא תקין, הפונקציה מבצעת את הגלגול הרלוונטי, לפי הערך של מקדם האיזון, כדי שמקדם האיזון יהיה תקין מה שיצביע על כך שהעץ מאוזן. keepBalanced יפעל מלמטה למעלה, על כל צומת מצומת הקלט ועד לשורש. אלו הם כל הצמתים שמקדם האיזון שלהם היה יכול להשתנות. הפונקציה keepBalanced סופר את מספר הפעמים שהשתמשנו בגלגול כלשהו ומחזירה את מספר הגלגולים שביצענו.  keepBalanced קורא לפונקציות בעלות סיבוכיות O(1) והוא קורא להן לכל היותר O(log(n)) פעמים (כגובה העץ), והסיבוכיות הכללית היא O(log(n)). |
| AVLTree | delete | O(log(n)) | הפונקציה מקבלת מספר כקלט ומחזירה מספר כפלט. הפונקציה מחפשת קודם כל האם הצומת נמצאת בעץ ע"י הפונקציה search. אם הצומת לא נמצאת בעץ נחזיר 1-. אחרת, נבדוק האם הצומת היא השורש, ואם כן נקרא לפונקציית העזר deleteRoot, אשר מטפלת במקרה זה. אחרת, נקרא לnodeKind כדי להבין מהו המקרה הרלוונטי. אם הפונקציה היא עלה, נמחוק אותו. אם לפונקציה יש בין אחד, נעקוף את הפונקציה בעזרת המצביעים, ונקשור בין האב שלו לבן שלו. אם לפונקציה יש 2 בנים, אז נמצא את הצומת העוקבת בעזרת הפונקציה successor, נמקם את הצומת העוקב במקום הצומת שברצוננו למחוק, ונחבר באמצעות המצביעים את הילד של העוקב עם האב שלו. לבסוף לכל מקרה נחשב את הגבהים שהשתנו ע"י calcHeightLeaf, ואז נקרא לפונקציה keepBalanced בכדי לשמור על העץ מאוזן. הפונקציה תחזיר את ערך ההחזרה של keepBalanced בשביל הפונקציה קראנו לארבע פונקציות בעלות סיבוכיות של O(log(n)), ולכן הסיבוכיות היא:  4 O(log(n)) = O(log(n)). |
| AVLTree | deleteRoot | O(log(n)) | במקרה בו יש למחוק את השורש נשתמש בפונקציה זו. הפונקציה תעבוד באופן מאוד דומה לdelete. בהתאם לסוג ההצומת, נדע אילו מצביעים לשנות. אם השורש הוא גם עלה אז נחזיר עץ ריק ואת הערך 0. אם לשורש רק בן אחד, אז נעדכן את הבן להיות שורש העץ. אם יש לעץ שני בנים אז נמצא את העוקב בעזרת הפונקציה successor, נהפוך את הצומת העוקב להיות שורש העץ, ונסדר את המצביעים של ההורה והבן הקודמים של העוקב. לבסוף נבצע calcHeightLeaf ולאחר מכן keepBalanced על הצומת הרלוונטי בכדי לעדכן את הגבהים ולשמור את העץ מאוזן. במקרה זה קראנו לשלוש פונקציות שהסיבוכיות שלהן היא O(log(n)), וסך כל הסיבוכיות היא  3O(log(n)) = O(log(n) |
| AVLTree | updateMinMaxDelete | O(log(n)) | מעדכנת לאחר פעולת מחיקה את המצביעים לאיבר המינימאלי והמקסימאלי באמצעות הפעולות successor למינימום אם מחקנו אותו ו-Predeseccor למקסימום אם מחקנו אותו. |
| AVLTree | successor | O(log(n)) | פונקציה הזאת מקבלת צומת, ומחזירה את הצומת העוקב שלו (הבא בסדר עולה). הפונקציה פועלת באופן הבא – אם קיים עוקב לצומת, הוא בהכרח יהיה בתת העץ הימני של הצומת שכן המפתח שלו גדול משל הצומת. בנוסף העוקב יהיה הצומת המינימלי בתת עץ זה. לכן נתקדם ימינה בצעד הראשון ואז נתקדם שמאלה בתת העץ עד שלא נוכל להתקדם שמאלה יותר. הצומת הראשון שאין לו בן שמאלי הוא העוקב ונחזיר אותו. אם אין בן ימני, נעלה במעלה העץ עד שנעלה לצומת שהגענו אליה מצד שמאל. מכיוון שאנו חסומים מלמעלה ע"י גובה העץ, זוהי גם הסיבוכיות של הפעולה – O(log(n)). |
| AVLTree | predecessor |  | פונקציה הזאת מקבלת צומת, ומחזירה את הצומת הקודם שלו (הבא בסדר עולה). הקוד עובד בצורה דומה לקוד של successor עם הבדלים שנובעים מסימטריה. |
| AVLTree | treeToNodeArray | O(n) | יוצרת מערך של צמתים ע"י מעבר inorder על העץ |
| AVLTree | infoToArray | O(n) | יוצרת את המערך של הצמתים באמצעות treeToNodeArray O(n) , ולאחר מכן עוברת על המערך ושומרת במערך חדש את כל ה-infos (O(n  סך הכל – הסיבוכיות – O(n) |
| AVLTree | keysToArray | O(n) | יוצרת את המערך של הצמתים באמצעות treeToNodeArray O(n) , ולאחר מכן עוברת על המערך ושומרת במערך חדש את כל ה- keys(O(n  סך הכל – הסיבוכיות – O(n) |
| AVLTree | Size | O(1) | מחזירה את גודל העץ השמור במערך |
| AVLTree | getRoot | O(1) | מחזירה את המצביע לשורש |
| AVLTree | Split | O(log(n)) | מקבלת צומת שנמצאת בעץ ומחזירה מערך שבו שני עצי AVL תקינים כך שכל הערכים באיבר הראשון של המערך קטנים מהערך שהתקבל, וכל האיברים באיבר השני גדולים מהערך שהתקבל.  הפונקציה פועלת כפי שנלמד בהרצאה הסיבוכיות O(log(n)) |
| AVLTree | minNode | O(log(n)) | מקבלת צומת שנמצאת בתוך העץ ומחזירה את הצומת המינימאלית שנמצאת בתת העץ ששורשו הצומת שהתקבלה |
| AVLTree | maxNode | O(log(n)) | מקבלת צומת שנמצאת בתוך העץ ומחזירה את הצומת המקסימאלית שנמצאת בתת העץ ששורשו הצומת שהתקבלה |
| AVLTree | createSubTree | O(log(n)) | מקבלת צומת ומחזירה עצם מסוג AVLTree שהשורש שלו מוגדר להיות הצומת הנתונה. הפונקציה מאתחלת את המצביעים min ו-max על ידי חיפושם בתת העץ. |
| AVLTree | updateMinMaxSplit | O(1) | מקבלת את המצביעים לעץ הקטן והגדול שיווצרו, וכן את הצומת שלפיה מחלקים את העץ, וכמו כן גם את העוקב והקודם של הצומת ומעדכנת בהתאם לפרמטרים את המצביעים למינימום ולמקסימום |
| AVLTree | Join | O(log(n)) | מקבלת עץ t2 וצומת x כך ש keys(t1)<x<keys(t2) ומשנה את העץ שעליו הופעלה הפעולה כך שיהיה איחוד של שני העצים. האלגוריתם פועל כפי שנלמד בהרצאה. הפונקציה מחזירה את סיבוכיות הפעולה. |
| AVLTree | calcComplexity | O(1) | מחשבת את הערך שצריך להחזיר בפונקציה join. |
| AVLTree | simpleMerge | O(1) | משנה את המצביעים בהתאם עבור המקרה שבו העץ שעליו הופעלה הפעולה, והעץ השני שהתקבל הם באותו גובה (עד כדי הבדל אחד) ומאחדת ביניהם תוך כדי שינוי הצומת. |
| AVLTree | setPointersForJoinLeft | O(1) | משנה את המצביעים בהתאם עבור המקרה בו יורדים דרך הענף השמאלי של העץ הגבוה יותר. |
| AVLTree | setPointersForJoinRight | O(1) | משנה את המצביעים בהתאם עבור המקרה בו יורדים דרך הענף הימני של העץ הגבוה יותר. |
| AVLTree | setHeightUpTree | O(log(n)) | הפונקציה מעדכנת את הגבהים של כל הצמתים בעץ לאחר פעולת ה-join. זו פעולה מקדימה לפעולות התיקון שעשויות להתבצע בהמשך. כלומר, אחרי פעולה זו ייתכן כי העץ שהתקבל איננו עץ AVL תקין. |
| AVLTree | getNodeWithHeight | O(log(n)) | הפונקציה מקבלת שני עצים, אחד גבוה ואחד נמוך, וביטוי בוליאני שמייצג האם העץ הגבוה מכיל ערכים גבוהים יותר מערך הפיצול. הפונקציה מחזירה בהתאם לתנאי (אם על הענף השמאלי ביותר או הימני ביותר) את הצומת בעלת גובה המתאים למיזוג. |
| AVLTree | updateMinMaxJoin | O(1) | מקבלת שני עצים גבוה ונמוך וביטוי בוליאני האם המפתחות בעץ הגבוה גדולים מערך המיזוג. הפונקציה מעדכנת את המצביעים למינימום ולמקסימום בהתאם לאחר פעולת המיזוג. |
| AVLNode | AVLNode | O(1) | זהו הבנאי של המחלקה. הבנאי מקבל מפתח שהוא מספר שלם ומידע שהוא מחרוזת. הבנאי יוצר צומת חדש ללא הורה וללא בנים, מגובה 0, שהמפתח שלו שווה למפתח הקלט והמידע השמור בו הוא המחרוזת שהוכנסה. |
| AVLNode | setInfo | O(1) | הפונקציה מקבלת מחרוזת, ומבצעת השמה כך שהשדה info של הצומת הופך להיות המחרוזת. |
| AVLNode | getValue | O(1) | הפונקציה מחזירה את המידע השמור בתוך צומת. |
| AVLNode | setKey | O(1) | הפונקציה מקבלת מספר שלם, והפכת את המפתח של הצומת להיות מספר זה. |
| AVLNode | getKey | O(1) | הפונקציה מחזירה את המפתח של הצומת. |
| AVLNode | setLeft | O(1) | הפונקציה מקבלת צומת נוספת, ומבצעת השמה כך שהצומת הנוסף הופך להיות הבן השמאלי של הצומת עליו הופעלה הפונקציה. |
| AVLNode | getLeft | O(1) | הפונקציה מחזירה את הבן השמאלי של הצומת. |
| AVLNode | setRight | O(1) | הפונקציה מקבלת צומת נוספת, ומבצעת השמה כך שהצומת הנוסף הופך להיות הבן הימני של הצומת עליו הופעלה הפונקציה. |
| AVLNode | getRight | O(1) | הפונקציה מחזירה את הבן הימני של הצומת. |
| AVLNode | setParent | O(1) | הפונקציה מקבלת צומת נוספת, ומבצעת השמה כך שהצומת הנוסף הופך להיות ההורה של הצומת עליו הופעלה הפונקציה. |
| AVLNode | getParent | O(1) | הפונקציה מחזירה את ההורה של הצומת. |
| AVLNode | getHeight | O(1) | הפונקציה מחזיקה את הגובה של הצומת. |
| AVLNode | setHeight | O(1) | הפונקציה מקבלת מספר שלם, ומבצעת השמה לשדה הheight של הצומת כך שהheight הופך להיות המספר השלם שהתקבל בקלט. |
| AVLNode | isRealNode | O(1) | הפונקציה בודקת האם הצומת היא אמיתית או צומת וירטואלית. הפונקציה היא בוליאנית ומחזירה true אם הצומת אמיתית וfalse אם הצומת וירטואלית. |
|  |  |  |  |