פרוייקט מבני נתונים 1

AVL Tree

AVLNode Class:

משתנים:

- leftNode – תת הבן השמאלי של הצומת. אם הצומת הוא עלה אזי תת הצומת השמאלי יהיה צומת וירטואלי בדרגה -1.
- rightNode – תת הבן הימני של הצומת. אם הצומת הוא עלה אזי תת הצומת השמאלי יהיה צומת וירטואלי בדרגה -1.
- parentNode – ה"אבא" של הצומת. אם הצומת הוא שורש אז האבא הוא null.
- rank – דרגה של הצומת המייצגת גם את הגובה שלו. מאותחל באופן קבוע ל-1 אלא אם הוא עלה.
- key – המפתח של הצומת.
- Info – המידע של הצומת.
- IsLeaf – משתנה בוליאני המציין האם הצומת הוא עלה.

בנאים:

- AVLNode(int keyParam, String infoParam): מאתחל עלה.
- AVLNode() – מאתחל צומת וירטואלית.

מתודות:

- getKey() – מחזיר את המפתח.
- getValue() – מחזיר את המידע.
- setLeft(IAVLNode node) – קובע את הבן השמאלי של הצומת.
- getLeft() – מחזיר את המצביע לבן השמאלי.
- setRight(IAVLNode node) – קובע את הבן הימני של הצומת.
- getRight() – מחזיר את המצביע לבן הימיני.
- setParent(IAVLNode node) – קובע את האבא של הצומת.
- IAVLNode getParent() – מחזיר מצביע לאבא של הצומת.
- isRealNode() – פונקציית עזר הבודקת האם מדובר בצומת וירטואלי או לא.
- setHeight(int height) – קובע את הגובה של העץ, ובעץ AVL הגובה שווה לדרגה.
- getHeight() – מחזיר את הגובה, כלומר את הדרגה.

כל המתודות פועלות בסיבוכיות O(1), הן מחזירות ערך או מצביע.

– Delete(int k)

פונקציה שמוחקת מתוך העץ את הצומת עפ"י המפתח K ומחזירה את מספר התיקונים שהיינו צריכים לבצע. אם המפתח לא קיים אנו מחזירים -1. סיבוכיות המתודה היא O(logn).

- ראשית אנו בודקים האם העץ הוא בעל שורש בלבד. אם מפתח השורש שווה לk אז נמחק אותו.
- אם הוא לא שורש, נבצע חיפוש של הצומת עם המפתח הנ"ל בעץ בעזרת פונקציית עזר deleteHelper(AVLNode node, int k). אם לא נמצא אותו נחזיר -1 ואם נמצא אותו נייחס אותו במשתנה mNode.

  בתוך פונקציית העזר אנו מחפשים את הצומת עפ"י ערך המפתח של הצומת. אם הוא גדול מהצומת הנוכחית נלך ימינה, ולא נלך שמאלה. אם הגענו לצומת שמעניינת אותנו נשתמש בפונקצייה DeleteNode(AVLNode node, AVLNode parent). הפוקנציה הזאת מבצעת את המחיקה עצמה:

  א. אם הצומת הוא עלה, נמחק אותו.
  ב. אם הוא צומת אונארי, נעקוף אותו.
  ג. אם הוא צומת בינארי, נמצא את העוקב שלו ע"י findSuccessor(AVLNode node), נחליף ביניהם ונמחק את הצומת.

  כעת, אחרי שמחקנו את הצומת, נחזיר את המצביע לצומת שהיה מעליו בגלל שייתכן שנוצרה בעיה מבחינת הדרגות בזמן המחיקה.

- כעת נספור את מספר התיקונים שיש לבצע כדי שהעץ יהיה עץ תקין. נצטרך להפריד למקרים על פי מה שלמדנו בכיתה. נתקן את העץ ע"י הפונקציה fixTreeDeletion(AVLNode node:

  הפונקציה תעלה במעלה העץ עד שהוא לא יהיה אף אחד מארבעת המצבים הבעייתים בעת מחיקה שאותם הגדרנו לפני במחלקה NodeStatus. עבור כל צומת הוא בודק האם הוא מקיים את אחד המצבים. אם כן הוא יבצע את התיקון וייספור את מספר התיקונים שהוא עשה(סיבובים ושינוי דרגות). המצבים CaseOne,CaseTwo,CaseThree,CaseFour הם אותם שמות של המצבים המתוארים במצגת.

  א. אם זה המצב הראשון, נוריד את הדרגה של הצומת ונמשיך.
  ב. אם זה המצב השני, נבצע סיבוב ונוריד את הדרגה של הצומת.
  ג. אם זה המצב השלישי, נבצע סיבוב ונוריד את הדרגה של הצומת ב2.
  ד. אם זה המצב הרביעי נבצע סיבוב כפול, נעדכן את הדרגה של הצומת ושל האבא.

- הסיבוכיות של הפונקציה תהיה תלויה בכל המשתנים הבאים:

  א. deleteHelper – חיפוש בעץ בינארי כפי שהוסבר מקודם הוא O(logn).
  ב. DeleteNode – מחיקה של צומת O(const).
  ג. findSuccessor – חיפוש של צומת בעץ בינארי O(logn).
  ד. fixTreeDeletion – מעבר על העץ ותיקון שלו. במקרה הגרוע אנו עוברים על כל הצמתים ועושים תיקונים בסיבוכיות של O(const) ובסה"כ $O(const * logn) =$ $.O(logn)$

ה. סך הסיבוכיות יהיה $O(logn + const + logn + logn) = O(logn)$

– split(int x)

מתודה שמפצלת את העץ שלנו לשני תתי עצים עפ"י ערך מסויים – כל הערכים שקטנים ממנו וכל הערכים שגדולים ממנו. בסוף התהליך "ייתפרק" העץ שלנו ואנו נחזיר מערך עם שני תתי העצים הנ"ל.

ראשית, נבדוק אם הערך קיים בעץ. החיפוש הוא בסיבוכיות של log(n). אם הוא לא בעץ, נכניס אותו לעץ בסיבוכיות log(n). כעת, נבדוק האם הצומת שאנו מסתכלים עליו הוא שורש – אם הוא שורש פשוט נחזיר את שני תתי העצים שלו.

אם לא, ניצור שני תתי עצים, ונתחיל לולאה לכיוון השורש ונבדוק האם הערך של הצומת גדול או קטן מהערך של המפתח אותו אנו מחפשים, ונוסיף אותו לעץ המתאים בעזרת פונקציית join בסיבוכיות logn. ניצור מצביע לתת עץ של הצומת הרלוונטי בסיבוכיות O(1) ונכניס אותו לפונקצייה. במקרה הגרוע, עלינו מעלה כלפי מעלה, כך שעבור כל צומת אנו מבצעים logn פעולות ולכן בסה"כ קיבלנו סיבוכיות של $O((\log n)^2)$

מספר סידורי	מספר פעולות	מספר פעולות האיזון הממוצע לפעולתinsert	מספר פעולות האיזון הממוצע לפעולתdelete	מספר פעולות האיזון המקסימלי לפעולתinsert	מספר פעולות האיזון המקסימלי לפעולת delete
1	10,000		2.69		26882
2	20,000		2.68		53683
3	30,000		2.68		80439
4	40,000		2.68		107518
5	50,000		2.68		134384
6	60,000		2.68		161072
7	70,000		2.68		187974
8	80,000		2.68		215487
9	90,000		2.68		242038
10	100,000		2.68		268930