מבני נתונים - פרויקט מעשי עץ AVL

6 במאי 2021

תוכן העניינים

1 המחלקה AVLNode

1.1 שדות המחלקה:

1. הקבוע הפרטי key מסוג int אשר מייצג את המפתח של צומת זה בעץ AVL.
2. הקבוע הפרטי $info$ מסוג boolean אשר מייצג את המידע של צומת זה בעץ AVL.
3. המשתנה הפרטי $height$ מסוג int אשר מייצג את גובה תת-העץ אשר צומת זה הוא שורשו.
4. המשתנה הפרטי $parent$ מסוג AVLNode אשר מייצג את אביו הישיר של צומת זה בעץ AVL.
5. המשתנה הפרטי $left$ מסוג AVLNode אשר מייצג את בנו השמאלי הישיר של צומת זה בעץ AVL.
6. המשתנה הפרטי $right$ מסוג AVLNode אשר מייצג את בנו הימני הישיר של צומת זה בעץ AVL.
7. המשתנה הפרטי $subTreeSize$ מסוג int אשר מייצג את גודל תת-העץ אשר צומת זה הוא שורשו.
8. המשתנה הפרטי $subTreeXor$ מסוג boolean אשר משמש למימוש יעיל של perfixXor ומייצג את תוצאת פעולת xor על שדות $info$ של כל הצמתים בתת-העץ אשר צומת זה הוא שורשו.
9. המשתנה הפרטי $successor$ מסוג AVLNode אשר מייצג את הצומת העוקב (הצומת עם המפתח העוקב בקבוצת המפתחות של הצמתים בעץ) של צומת זה בעץ AVL. אם הצומת הוא האיבר עם המפתח המקסימלי בעץ, ערך שדה זה יהיה null.
10. המשתנה הפרטי $predecessor$ מסוג AVLNode אשר מייצג את הצומת הקודם (הצומת עם המפתח הקודם בקבוצת המפתחות של הצמתים בעץ) של צומת זה בעץ AVL. אם הצומת הוא האיבר עם המפתח המינימלי בעץ, ערך שדה זה יהיה null.

1.2 בנאי המחלקה:

1.2.1 הבנאי הריק public AVLNode()

מה הוא עושה: פעולה בונה של המחלקה, מאתחלת ערכים לשדות מסוימים עבור צומת וירטואלי בעץ. כלומר, פעולה זו בונה צומת וירטואלי.

כיצד הוא פועל: מאתחל לשדות ערכים לפי ערכי הצומת הוירטואלי הרצויים.

סיבוכיות זמן הריצה: מתבצע מספר קבוע של השמות, ולכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

1.2.2 הבנאי public AVLNode(int key, boolean info, AVLNode parent)

הפרמטרים: int key - ערך המפתח של הצומת. boolean info - ערך המידע ששמור בצומת. AVLNode parent - אביו של הצומת בעץ AVL.

מה הוא עושה: פעולה בונה של המחלקה, מאתחלת את שדות המחלקה עבור עלה בעץ.

- השדה key מאותחל לפרמטר key שניתן על ידי המשתמש.
- השדה $info$ מאותחל לפרמטר $info$ שניתן על ידי המשתמש.
- השדה $height$ מאותחל ל-0, שהוא גובה של תת-עץ בעל צומת אחד.
- השדה $parent$ מאותחל לפרמטר $parent$ שניתן על ידי המשתמש.
- השדה $subTreeSize$ מאותחל ל-1, שהוא כמות הצמתים בתת-עץ בעל צומת אחד.
- השדה $subTreeXor$ מאותחל לפרמטר $info$ שניתן על ידי המשתמש, שכן זהו תוצאת פעולת xor על כלל המידע של הצמתים שנמצאים בתת-העץ ששורשו הוא צומת זה, כלומר תת-עץ עם צומת יחיד.
- השדה $left$ מאותחל לצומת הוירטואלי של העץ (AVLTree) בו הוא נוצר, שכן הייצוג של חוסר קיום ילד לצומת, הוא צומת וירטואלי.
- השדה $right$ מאותחל לצומת הוירטואלי של העץ (AVLTree) בו הוא נוצר.

כיצד הוא פועל: מאתחל חלק משדות המחלקה בהתאם לכללים והגדרות אחריהם אנו עוקבים בקורס, כאשר השמה חוקית לשדות successor ו-predecessor תתבצע במהלך הפעולה public int insert(int k, boolean i), פעולת הכנסת צומת של המחלקה AVLTree.

סיבוכיות זמן הריצה: מתבצע מספר קבוע של השמות, וכן קריאה למתודה getVirtualNode() אשר כפי שניתן יהיה לראות בהמשך, פועלת ב-$O(1)$ סיבוכיות זמן ריצה. לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

1.3 פעולות המחלקה:

1.3.1 הפעולה public boolean isRealNode()

מה היא עושה: מחזירה האם הצומת הוא אמיתי.

סיבוכיות זמן הריצה: ישנה השוואה וגישה אחת לשדה לכן $O(1)$.

1.3.2 הפעולה public int getKey()

מה היא עושה: מחזירה את המפתח של הצומת.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.3 הפעולה public Boolean getValue()

מה היא עושה: מחזירה את המידע של הצומת.

סיבוכיות זמן הריצה: ישנה קריאה לפעולה isRealNode() שרצה ב-$O(1)$ וכן גישה אחת לשדה, לכן סיבוכיות זמן הריצה היא $O(1)$.

1.3.4 הפעולה public void setLeft(AVLNode node)

מה היא עושה: אם הצומת $node$ הוא אמיתי, קובעת אותו כבן השמאלי של הצומת הנוכחי.

סיבוכיות זמן הריצה: ישנה קריאה לפעולה isRealNode() שרצה ב-$O(1)$ וכן גישה אחת לשדה, לכן סיבוכיות זמן הריצה היא $O(1)$.

1.3.5 הפעולה public AVLNode getLeft()

מה היא עושה: מחזירה את הבן השמאלי של הצומת.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.6 הפעולה public void setRight(AVLNode node)

מה היא עושה: אם הצומת $node$ הוא אמיתי, קובעת אותו כבן הימני של הצומת הנוכחי.

סיבוכיות זמן הריצה: ישנה קריאה לפעולה isRealNode() שרצה ב-$O(1)$ וכן גישה אחת לשדה, לכן סיבוכיות זמן הריצה היא $O(1)$.

1.3.7 הפעולה public AVLNode getRight()

מה היא עושה: מחזירה את הבן הימני של הצומת.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.8 הפעולה public void setChildInDir(AVLNode node, Direction dir)

מה היא עושה: קוראת ל**setRight(AVLNode node)** או **setLeft(AVLNode node)** בהינתן לכיוון הנתון ב**dir**.

סיבוכיות זמן הריצה: ישנה השוואה וקריאה לפעולה **setRight(AVLNode node)** או **setLeft(AVLNode node)** שרצות ב-$O(1)$, לכן סיבוכיות זמן הריצה היא $O(1)$.

1.3.9 הפעולה public AVLNode getChildInDir(Direction dir)

מה היא עושה: קוראת ל**getRight()** או **getLeft()** בהינתן לכיוון הנתון ב**dir**.

סיבוכיות זמן הריצה: ישנה השוואה וקריאה לפעולה **getRight()** או **getLeft()** שרצות ב-$O(1)$, לכן סיבוכיות זמן הריצה היא $O(1)$.

1.3.10 הפעולה public void setParent(AVLNode node)

מה היא עושה: אם הצומת $node$ הוא אמיתי, קובעת אותו כאבא של הצומת הנוכחי.

סיבוכיות זמן הריצה: ישנה קריאה לפעולה isRealNode() שרצה ב-$O(1)$ וכן גישה אחת לשדה, לכן סיבוכיות זמן הריצה היא $O(1)$.

1.3.11 הפעולה public AVLNode getParent()

מה היא עושה: מחזירה את אביו הישיר של הצומת.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.12 הפעולה public void setHeight(int height)

מה היא עושה: קובעת את הגובה של הצומת להיות $height$.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.13 הפעולה public int getHeight()

מה היא עושה: מחזירה את הגובה של הצומת.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.14 הפעולה public void setSize(int size)

מה היא עושה: קובעת את כמות הצמתים בתת-העץ ששורשו הוא הצומת הנוכחי להיות $size$.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.15 הפעולה public int getSize()

מה היא עושה: מחזירה את כמות הצמתים בתת-העץ ששורשו הוא הצומת הנוכחי.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.16 הפעולה public void setSubTreeXor(boolean xor)

מה היא עושה: קובעת את תוצאת פעולת xor על המידע של כלל הצמתים בתת-העץ ששורשו הוא הצומת הנוכחי להיות xor.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.17 הפעולה public boolean getSubTreeXor()

מה היא עושה: מחזירה את תוצאת פעולת xor על המידע של כלל הצמתים בתת-העץ ששורשו הוא הצומת הנוכחי.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.18 הפעולה public void setSuccessor(AVLNode successor)

מה היא עושה: קובעת את הצומת העוקב של הצומת הנוכחי להיות $successor$.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.19 הפעולה public AVLNode getSuccessor()

מה היא עושה: מחזירה את הצומת העוקב של הצומת הנוכחי.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.20 הפעולה public void setPredecessor(AVLNode predecessor)

מה היא עושה: קובעת את הצומת הקודם של הצומת הנוכחי להיות $predecessor$.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.21 הפעולה public AVLNode getPredecessor()

מה היא עושה: מחזירה את הצומת הקודם של הצומת הנוכחי.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

1.3.22 הפעולה public int getBalanceFactor()

מה היא עושה: אם הצומת הוא אמיתי, מחזירה את גורם האיזון (כפי שהוגדר בקורס - ההפרש בין גובה תת-העץ השמאלי של הצומת לבין גובה תת-העץ הימני של הצומת) של הצומת הנוכחי.

סיבוכיות זמן הריצה: ישנה קריאה למספר קבוע של הפעולות isRealNode(), getLeft(), getRight() ו-getHeight() אשר רצות ב-$O(1)$ לכן סיבוכיות זמן הריצה היא $O(1)$.

1.3.23 הפעולה public void updateNodeFields()

מה היא עושה: מעדכנת את השדות height, subTreeSize, subTreeXor של הצומת בהתאם לאלו של ילדיה בעץ.

כיצד היא פועלת: מעדכנת את הערכים לפי הנוסחאות הרקורסיביות שראינו בקורס בהנחת נכונות השדות של ילדיה.
שדה הגובה מעודכן להיות המקסימום בין גבהי ילדיו ועוד אחד (עבור הצומת עצמו).
שדה הגודל מעודכן להיות סכום גדלי תתי-העצים שילדי הצומת הם שורשיהם, ועוד אחד (עבור הצומת עצמו).
שדה ה-xor מעודכן להיות תוצאת פעולה xor על שדות xor תתי-העצים של ילדיו וכן על המידע של הצומת עצמו.

סיבוכיות זמן הריצה: מתבצע מספר קבוע של השמות, שימוש במספר קבוע של פעולות "get" ו-"set" של המחלקה AVLNode וכן מספר קבוע של פעולות אריתמטיות ופעולות xor. כל אלו מתבצעים ב-$O(1)$ זמן, ולכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O(1)$.

1.3.24 הפעולה public int getChildCount()

מה היא עושה: מחזירה את כמות הילדים הישירים של הצומת בעץ.

כיצד היא פועלת: עבור כל אחד משני ילדיה, בודקת האם הוא צומת וריטואלי או צומת אמיתי. אם הוא צומת אמיתי, מוסיפה אחד למניין הילדים.

סיבוכיות זמן הריצה: מתבצע מספר קבוע של השמות, ושימוש בפעולות getLeft(), getRight() ו-isRealNode(), כולם ב-$O(1)$.
לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O(1)$.

1.3.25 הפעולה public boolean isLeftChild()

מה היא עושה: מחזירה האם הצומת הוא בן שמאלי של אביו הישיר.

כיצד היא פועלת: בודקת האם המפתח של הצומת קטן מהמפתח של צומת האב. אם כן, לפי הגדרת עץ חיפוש בינארי, הצומת הוא בן שמאלי, ולכן מחזירה true. אחרת, סימן שהצומת הוא בן ימני, ולכן מחזירה false.

סיבוכיות זמן הריצה: ישנו שימוש בפעולות getKey() ו-getParent() אשר רצים ב-$O(1)$. לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O(1)$.

2 המחלקה AVLTree

2.1 טיפוס המנייה Direction

2.1.1 ערכי טיפוס המנייה

1. הכיוון $Right$.
2. הכיוון $Left$.

2.1.2 הפעולה public Direction reverseDir()

מה היא עושה: פעולה זו "מנגדת" את הכיוון הנתון. כלומר, עבור ערך $Right$ היא תחזיר $Left$ ולהיפך.

סיבוכיות זמן הריצה: הפעולה זו רצה בסיבוכיות $O(1)$.

2.2 שדות המחלקה:

1. הקבוע הפרטי $virtualNode$ מסוג AVLNode אשר מצביע לצומת הוירטואלי של העץ.
2. המשתנה הפרטי $root$ מסוג AVLNode אשר מצביע לשורש של העץ.
3. המשתנה הפרטי $minNode$ מסוג AVLNode אשר מצביע לצומת בעל המפתח המינימלי מבין המפתחות של כלל הצמתים (לא וירטואליים) של העץ. אם העץ הוא עץ ריק, $minNode$ מצביע לערך null.
4. המשתנה הפרטי $maxNode$ מסוג AVLNode אשר מצביע לצומת בעל המפתח המקסימלי מבין המפתחות של כלל הצמתים (לא וירטואליים) של העץ. אם העץ הוא עץ ריק, $maxNode$ מצביע לערך null.

2.3 בנאי המחלקה:

2.3.1 הבנאי הריק public AVLTree()

ללא מימוש, יוצר אובייקט מטיפוס AVLTree אשר ניתן להתחיל להכניס אליו מפתחות ומידע בעזרת הפעולה insert(int k, boolean i).

2.4 פעולות המחלקה:

2.4.1 הפעולה public AVLNode getVirtualNode()

מה היא עושה: מחזירה את הצומת הוירטואלי של העץ.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

2.4.2 הפעולה public AVLNode getRoot()

מה היא עושה: מחזירה את השורש של העץ.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

2.4.3 הפעולה public void setRoot(AVLNode root)

מה היא עושה: קובעת את שורש העץ להיות $root$.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

2.4.4 הפעולה public boolean empty()

מה היא עושה: מחזירה האם העץ הוא ריק (או באופן שקול, האם שורשו הוא null).

סיבוכיות זמן הריצה: ישנה קריאה לפעולה getRoot() שרצה ב-$O(1)$ וכן פעולת השוואה אחת, לכן סיבוכיות זמן הריצה היא $O(1)$.

2.4.5 הפעולה public int size()

מה היא עושה: מחזירה את כמות הצמתים בעץ (או באופן שקול את גודל תת-העץ ששורשו הוא שורש העץ, או 0 אם לא קיים שורש).

סיבוכיות זמן הריצה: ישנה קריאה לפעולות getRoot() ו-getSize() שרצות ב-$O(1)$, לכן סיבוכיות זמן הריצה היא $O(1)$.

2.4.6 הפעולה public AVLNode getMin()

מה היא עושה: מחזירה את הצומת בעל המפתח המינימלי מבין המפתחות של כלל הצמתים (לא וירטואליים) של העץ. אם העץ הוא עץ ריק, מחזירה null.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

2.4.7 הפעולה public void setMin(AVLNode min)

מה היא עושה: קובעת את הצומת בעל המפתח המינימלי מבין המפתחות של כלל הצמתים (לא וירטואליים) של העץ להיות min.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

2.4.8 הפעולה public AVLNode getMax()

מה היא עושה: מחזירה את הצומת בעל המפתח המקסימלי מבין המפתחות של כלל הצמתים (לא וירטואליים) של העץ. אם העץ הוא עץ ריק, מחזירה null.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

2.4.9 הפעולה public void setMax(AVLNode max)

מה היא עושה: קובעת את הצומת בעל המפתח המקסימלי מבין המפתחות של כלל הצמתים (לא וירטואליים) של העץ להיות max.

סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן $O(1)$.

2.4.10 הפעולה public AVLNode searchNode(int k)

מה היא עושה: מחזירה את הצומת בעץ (כאשר נתון שהעץ לא ריק) עם המפתח k, אם קיים צומת כזה. אם לא קיים בעץ צומת עם מפתח k, הפעולה מחזירה את הצומת שאמור להיות האבא הישיר של עלה עם מפתח k.

כיצד היא פועלת: מבצעת חיפוש של k בעץ חיפוש בינארי ("מתקדמת" שמאלה או ימינה בעץ בהתאם להשוואה בין מפתח של צומת נוכחי לבין k). אם נמצא צומת עם המפתח k, מחזירה אותו. אילו הצומת הבא אליו אמורים "להתקדם" בלולאה הוא צומת וירטואלי, סימן שלא קיים כזה צומת עם מפתח k בעץ, ומוחזר הצומת האחרון בו פגשנו בלולאה (אילו היה צומת עם מפתח k בעץ בתור עלה, צומת זה היה אביו, שכן המסלול שהלולאה עברה הוא המסלול בו הייתה עוברת במקרה זה).

סיבוכיות זמן הריצה: חיפוש בעץ חיפוש בינארי עובר במסלול מהשורש אל עלה בעץ, מסלול שאורכו לכל היותר $h+1$ כאשר h הוא גובה העץ, וכן בעץ AVL מתקיים $h = O(\log n)$. לכן, הפעולה searchNode(int k) מבצעת לכל היותר $O(\log n)$ איטרציות, ובכל איטרציה מתבצעה מספר פעולות שכפי שראינו סיבוכיות זמן הריצה שלהן הוא $O(1)$ (כמו גם הסיבוכיות של הפעולה getRoot שמתבצעת לפני הלולאה). לכן סיבוכיות זמן הריצה הכוללת היא $O(\log n)$.

2.4.11 הפעולה public Boolean search(int k)

מה היא עושה: מחזירה את הצומת בעץ עם המפתח k, אם קיים צומת כזה. אם לא קיים בעץ צומת עם מפתח k, הפעולה מחזירה null.

כיצד היא פועלת: אם העץ ריק, הפעולה מחזירה null. אחרת, משתמשת בצומת $node$ שהוחזר מהפעולה searchNode(int k). אילו ערך המפתח של $node$ הוא k, מוחזר את המידע של הצומת. אחרת, המשמעות היא שלא נמצא בעץ צומת עם מפתח k, ומוחזר null.

סיבוכיות זמן הריצה: חוץ מהקריאה היחידה לפעולה searchNode(int k) אשר מתבצעת ב-$O(\log n)$ זמן, כלל הפעולות מתבצעות ב-$O(1)$. לכן סיבוכיות הזמן הכוללת של הפעולה היא $O(\log n)$.

2.4.12 הפעולה public int insert(int k, boolean i)

מה היא עושה: מכניסה צומת חדש לעץ עם מפתח k ומידע i, אם לא קיים צומת כזה ומחזירה את כמות פעולות האיזון שבוצעו כולל פעולת ההכנסה. אם קיים בעץ צומת עם מפתח k, הפעולה מחזירה -1.

כיצד היא פועלת:

1. העץ ריק:

   יוצרת צומת חדש ומגדירה אותו בתור השורש, האיבר המינימלי והמקסימלי.

2. העץ לא ריק:

   משתמשת ב **searchNode(int k)** כדי למצוא את הצומת שאמור להיות אביו של הצומת החדש, ומכניסה את הצומת החדש בתור בנו בעזרת updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode) או updateRelationsForNewRightChild(AVLNode parent, AVLNode newNode) ולאחר מכן נאזן את העץ בעזרת balanceTreeOnce(AVLNode node) פעם אחת עד ידי קריאה יחידה (או שתיים במידה ונדרשה פעולת איזון).

סיבוכיות זמן הריצה: ב"מקרה הטוב" הראינו כי empty(), setRoot(AVLNode root), setMin(AVLNode min), setMax(AVLNode max) רצות ב$O(1)$ לכן סה"כ $O(1)$. ב"מקרה הגרוע" ישנן מספר קריאות למתודות אחרות בקוד - קריאה לsearchNode(int k) אשר מתבצעת ב-$O(\log n)$ זמן וקריאה לbalanceTreeOnce(AVLNode node) פעם או פעמיים כך שסה"כ עוברים על העץ מהצומת הנוכחי עד שורש שזה חסום בגובה העץ לכן גם כן $O(\log n)$ זמן ושתי הפעולות האחרות מתבצעות ב-$O(1)$. לכן סיבוכיות הזמן הכוללת של הפעולה היא $O(\log n)$.

2.4.13 הפעולה private void updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode)

מה היא עושה: מגדירה את newNode בתור בנו השמאלי של node תוך התחשבות במקרי הקצה ועדכון המצביעים הרלוונטים.

סיבוכיות זמן הריצה: הראינו כי הפעולות setLeft(AVLNode node), getMin(), setMin(AVLNode min), updateSuccessor(AVLNode node, AVLNode newNode) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.14 הפעולה private void updateRelationsForNewRightChild(AVLNode parent, AVLNode newNode)

מה היא עושה: מגדירה את newNode בתור בנו הימני של node תוך התחשבות במקרי הקצה ועדכון המצביעים הרלוונטים.

סיבוכיות זמן הריצה: הראינו כי הפעולות setRight(AVLNode node), getMax(), setMax(AVLNode min), updateSuccessor(AVLNode node, AVLNode newNode) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.15 הפעולה private AVLNode balanceTreeOnce(AVLNode node)

מה היא עושה: עולה מהצומת המועבר אל השורש עד שהיא נתקלת בצומת לא מאוזן, מאזנת אותו ומחזירה את אביו. אם העץ מאוזן הפעולה תחזיר את הצומת הוירטואלי.

סיבוכיות זמן הריצה: הפעולה עולה במסלול ישיר אל הצומת הלא מאוזן (או לחילופין השורש) עם פעולות $O(1)$ לכלחלק זה הוא חסום באורך המסלול מהצומת הנתון אל הצומת שצריך איזון. בנוסף, הראינו כי הפעולות setLeft(AVLNode node), getMin(), setMin(AVLNode min), updateSuccessor(AVLNode node, AVLNode newNode) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.16 הפעולה private void balanceNode(AVLNode node)

מה היא עושה: מפעילה את פעולת האיזון המתאימה על הצומת לפי החוקיות שראינו בכיתה.

סיבוכיות זמן הריצה: הפונקציה קוראת לאחת מארבע פעולות האיזון

- המתודה rotateRight(AVLNode node)
- המתודה rotateLeft(AVLNode node)
- המתודה rotateLeftThenRight(AVLNode node)
- המתודה rotateRightThenLeft(AVLNode node)

ומבצעת שאילתות

- השאילתה getBalanceFactor()
- השאילתה getLeft()

ב-$O(1)$ לכן סך הפעולות הוא $O(1)$.

2.4.17 הפעולה private boolean isUnbalanced(AVLNode node)

מה היא עושה: מחזירה האם גורם האיזון של הצומת בטווח המתאים.

סיבוכיות זמן הריצה: מבצעת שאילתת getBalanceFactor() ב-$O(1)$ לכן סך הפעולות הוא $O(1)$.

2.4.18 הפעולה private void rotateInDir(AVLNode node, Direction dir)

מה היא עושה: מבצעת פעולת איזון מסוג סיבוב יחיד ימינה או שמאלה בהינתן הכיוון dir.

כיצד היא פועלת: פועלת לפי האלגוריתם שראינו בכיתה אך בכלליות לשני הכיוונים: תחילה מעבירה את הילד המתאים, מתקנת מצביעים, מתקנת נכונות שדות ולבסוף מוודאת כי המצביע לשורש הוא נכון.

סיבוכיות זמן הריצה: ישנן קריאות לשאילותות ומתודות הרצות בסיבוכיות $O(1)$

השאילתות:

- השאילתה getDirectionFromParent(AVLNode node)
- השאילתה reverseDir()
- השאילתה getChildInDir(Direction dir)
- השאילתה getRoot()

הפעולות

- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה setParent(AVLNode node)
- הפעולה updateNodeFields()
- הפעולה setRoot(AVLNode root)

2.4.19 הפעולה private void rotateRight(AVLNode node)

מה היא עושה: מבצעת סיבוב יחיד ימינה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולה rotateInDir(AVLNode node, Direction dir) מתבצעת בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.20 הפעולה private void rotateLeft(AVLNode node)

מה היא עושה: מבצעת סיבוב יחיד שמאלה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולה rotateInDir(AVLNode node, Direction dir) מתבצעת בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.21 הפעולה private void rotateLeftThenRight(AVLNode node)

מה היא עושה: מבצעת סיבוב שמאלה וימינה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות

- הפעולה rotateRight(AVLNode node)
- הפעולה rotateLeft(AVLNode node)

מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.22 הפעולה private void rotateRightThenLeft(AVLNode node)

מה היא עושה: מבצעת סיבוב ימינה ושמאלה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות

- הפעולה rotateRight(AVLNode node)
- הפעולה rotateLeft(AVLNode node)

מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.23 הפעולה public int delete(int k)

מה היא עושה: מוחקת את הצומת בעל המפתח הנתון מהעץ אם הוא קיים ומחזירה את כמות פעולות האיזון שנדרשו לאיזון העץ, או -1 אם הצומת המתאים לא נמצא בעץ.

כיצד היא פועלת: תחילה הפעולה מוצאת את הצומת בעל המתפח הנתון, אם הוא קיים, מעדכנת את המצביעים של העוקב והקודם שלו באמצעות updateSuccessor(AVLNode node, AVLNode newNode) ולאחר מכן נוקטת באחד משמשת המקרים המתאימים:

1. הצומת הוא השורש וללא הילדים - משמע הוא הצומת היחידי בעץ

   נגדיר את העץ למצב של עץ "חדש" ללא צמתים ונחזיר 1 כנגד המחיקה היחדה שבוצעה.

   האיפוס יתבצע באמצעות הפעולות:

   - הפעולה setRoot(AVLNode root)
   - הפעולה setMin(AVLNode min)
   - הפעולה setMax(AVLNode max)

   אשר כל אחת פועלת ב-$O(1)$ בסיבוכיות כוללת של $O(1)$.

2. הצומת הוא המינימום בעץ:

   נעדכן את המינימום בעץ להיות העוקב של הצומת ונבצע מעקף עם בנו הימני - הבן היחידי שיכול להיות לו - אם אין לו בן אז הוא הלכה למעשה מוחלף בצומת וירטואלי פשוט כמו מחיקת עלה מעץ.

   זאת באמצעות השאילתות:

   - השאילתה getSuccessor()
   - השאילתה getRight()

- השאילתה getParent()
- השאילתה getRoot()
- השאילתה getVirtualNode()

אשר כולן בסיבוכיות $O(1)$ והפעולות:

- הפעולה setMin(AVLNode min)
- הפעולה setParent(AVLNode node)
- הפעולה setLeft(AVLNode node)
- הפעולה setParent(AVLNode node)

אשר גם הן בסיבוכיות $O(1)$ ולכן הסיבוכיות הכוללת היא $O(1)$.

3. הצומת הוא המקסימום בעץ:

נעדכן את המקסימום בעץ להיות הקודם של הצומת ונבצע מעקף עם בנו השמאלי - הבן היחידי שיכול להיות לו - אם אין לו בן אז הוא הלכה למעשה מוחלף בצומת וירטואלי פשוט כמו מחיקת עלה מעץ.

זאת באמצעות השאילתות:

- השאילתה getPredecessor()
- השאילתה getLeftt()
- השאילתה getParent()
- השאילתה getRoot()
- השאילתה getVirtualNode()

אשר כולן בסיבוכיות $O(1)$ והפעולות:

- הפעולה setMax(AVLNode min)
- הפעולה setParent(AVLNode node)
- הפעולה setRight(AVLNode node)
- הפעולה setParent(AVLNode node)

אשר גם הן בסיבוכיות $O(1)$ ולכן הסיבוכיות הכוללת היא $O(1)$.

4. הצומת לא השורש והוא אינו המינימום או המקסימום בעץ:

(א) לצומת בן יחיד:

נבצע מעקף עם בן זה.

נשתמש בשאילתות

- השאילתה getLeft()
- השאילתה isRealNode()
- השאילתה getChildInDir(Direction dir)
- השאילתה getParent()
- השאילתה getDirectionFromParent(AVLNode node)

והפעולות:

- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה setParent(AVLNode node)

(ב) לצומת שני בנים:

מאחר והקודם לצומת חייב להיות בתת העץ של אותו הצומת מאחר ויש לצומת שני ילדים נובע כי הקודם נמצא בתת העץ השמאלי ויתר על כן לא ייתכן כי יש לו ילד ימני אחרת הוא או ילד ימני שלו היה הקודם לכן נוכל לבצע מעקף לצומת הקודם ולאחר ש"שחררנו" אותו נחליף איתו את הצומת אותו אנו רוצים למחוק. לבסוף נבצע מעקף מהאב של הקודם או במקרה וזה בנו השמאלי של הצומת אז מהאיבר הקודם.

נבצע זאת באמצעות השאילתות:

- השאילתה getPredecessor()
- השאילתה getParent()

- השאילתה getLeft()
- השאילתה getDirectionFromParent(AVLNode node)
- השאילתה getRoot()

והפעולות:

- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה setParent(AVLNode node)
- הפעולה replaceChildren(AVLNode oldParent, AVLNode newParent)
- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה setRoot(AVLNode root)

ראינו כי כלל הפעולות והשאילתות מתבצעות בסיבוכיות $O(1)$.

(ג) הצומת הוא עלה בעץ:

נבצע מעקף עם הבן שלו שהוא צומת וירטואלי.

זאת באמצעות השאילתות:

- השאילתה getParent()
- השאילתה getDirectionFromParent(AVLNode node)
- השאילתה getVirtualNode()

והפעולה setChildInDir(AVLNode node, Direction dir) אשר כולן רצות בסיבוכיות $O(1)$.

ובסיום רצה הפעולה balanceNode(AVLNode node).

ראינו כי כלל הפעולות והשאילתות מתבצעות בסיבוכיות $O(1)$ למעט balanceNode(AVLNode node) הרצה בסיבוכיות $O(\log n)$ לכן הפעולה רצה בסיבוכיות $O(\log n)$.

2.4.24 הפעולה private void replaceChildren(AVLNode oldParent, AVLNode newParent)

מה היא עושה: מעבירה את ילדי oldParent להיות ילדי newParent.

כיצד היא פועלת: קוראת לפעולות setRight(AVLNode node), setLeft(AVLNode node) על מנת להגדיר את ילדי oldParent להיות ילדי newParent ואז משתמשת בsetParent(AVLNode node) על מנת להגיד את newParent בתור ההורה של ילדי oldParent.

סיבוכיות זמן הריצה: הראינו כי הפעולות

- השאילתה getRight()
- השאילתה getLeft()
- הפעולה setParent(AVLNode node)
- הפעולה setRight(AVLNode node)
- הפעולה setLeft(AVLNode node)

מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.25 הפעולה private Direction getDirectionFromParent(AVLNode node)

מה היא עושה: מחזירה את הכיוון באופן יחסי לאבא של הצומת, אם הצומת הנתון הוא השורש הכיוון שיוחזר הוא ימינה.

כיצד היא פועלת: קוראת לisLeftChild() ולפי ערך ההחזרה שלו מחזירה את הכיוון המתאים.

סיבוכיות זמן הריצה: הראינו כי הפעולה isLeftChild() מתבצעת בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.26 הפעולה private int balanceTree(AVLNode node)

מה היא עושה: מאזנת את העץ מהצומת הנתון ועד לשורש, בסיום מחזירה את כמות פעולות האיזון (ועוד אחת) שהתבצעו.

כיצד היא פועלת: מתחילה מהצומת הנתון וממשיכה בלולאה עד שהיא מגיעה לאבא של השורש (הצומת הוירטואלי) בקריאה ל balanceTreeOnce(AVLNode node) ומגדילה ב1 את המונה.

סיבוכיות זמן הריצה: הראינו כי הפעולה balanceTreeOnce(AVLNode node) מתבצעת בסיבוכיות אורך המסלול בין הצומת הנתון אל הצומת המוחזר. אנו קוראים בלולאה שוב ושוב לפעולה עם הערך המוחזר עד שאנו מגיעים לשורש כלומר בסיבוכיות של אורך המסלול מהצמת הנתון אל השורש שהיא חסומה בגובה העץ הלא הוא $O(\log n)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(\log n)$.

2.4.27 הפעולה public int insert(int k, boolean i)

מה היא עושה: מכניסה צומת חדש לעץ עם מפתח k ומידע i, אם לא קיים צומת כזה ומחזירה את כמות פעולות האיזון שבוצעו כולל פעולת ההכנסה. אם קיים בעץ צומת עם מפתח k, הפעולה מחזירה -1.

כיצד היא פועלת:

1. העץ ריק:

   יוצרת צומת חדש ומגדירה אותו בתור השורש, האיבר המינימלי והמקסימלי.

2. העץ לא ריק:

   משתמשת ב **searchNode(int k)** כדי למצוא את הצומת שאמור להיות אביו של הצומת החדש, ומכניסה את הצומת החדש בתור בנו בעזרת updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode) או updateRelationsForNewRightChild(AVLNode parent, AVLNode newNode) ולאחר מכן נאזן את העץ בעזרת balanceTreeOnce(AVLNode node) פעם אחת עד ידי קריאה יחידה (או שתיים במידה ונדרשה פעולת איזון).

סיבוכיות זמן הריצה: ב"מקרה הטוב" הראינו כי empty(), setRoot(AVLNode root), setMin(AVLNode min), setMax(AVLNode max) רצות ב$O(1)$ לכן סה"כ $O(1)$. ב"מקרה הגרוע" ישנן מספר קריאות למתודות אחרות בקוד - קריאה לsearchNode(int k) אשר מתבצעת ב-$O(\log n)$ זמן וקריאה לbalanceTreeOnce(AVLNode node) פעם או פעמיים כך שסה"כ עוברים על העץ מהצומת הנוכחי עד שורש שזה חסום בגובה העץ לכן גם כן $O(\log n)$ זמן ושתי הפעולות האחרות מתבצעות ב-$O(1)$. לכן סיבוכיות הזמן הכוללת של הפעולה היא $O(\log n)$.

2.4.28 הפעולה private void updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode)

מה היא עושה: מגדירה את newNode בתור בנו השמאלי של node תוך התחשבות במקרי הקצה ועדכון המצביעים הרלוונטים.

סיבוכיות זמן הריצה: הראינו כי הפעולות setLeft(AVLNode node), getMin(), setMin(AVLNode min), updateSuccessor(AVLNode node, AVLNode newNode) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.29 הפעולה private void updateRelationsForNewRightChild(AVLNode parent, AVLNode newNode)

מה היא עושה: מגדירה את newNode בתור בנו הימני של node תוך התחשבות במקרי הקצה ועדכון המצביעים הרלוונטים.

סיבוכיות זמן הריצה: הראינו כי הפעולות setRight(AVLNode node), getMax(), setMax(AVLNode min), updateSuccessor(AVLNode node, AVLNode newNode) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.30 הפעולה private AVLNode balanceTreeOnce(AVLNode node)

מה היא עושה: עולה מהצומת המועבר אל השורש עד שהיא נתקלת בצומת לא מאוזן, מאזנת אותו ומחזירה את אביו. אם העץ מאוזן הפעולה תחזיר את הצומת הוירטואלי.

סיבוכיות זמן הריצה: הפעולה עולה במסלול ישיר אל הצומת הלא מאוזן (או לחילופין השורש) עם פעולות $O(1)$ לכוחלק זה הוא חסום באורך המסלול מהצומת הנתון אל הצומת שצריך איזון. בנוסף, הראינו כי הפעולות setLeft(AVLNode node), getMin(), setMin(AVLNode min), updateSuccessor(AVLNode node, AVLNode newNode) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.31 הפעולה private void balanceNode(AVLNode node)

מה היא עושה: מפעילה את פעולת האיזון המתאימה על הצומת לפי החוקיות שראינו בכיתה.

סיבוכיות זמן הריצה: הפונקציה קוראת לאחת מארבע פעולות האיזון

- המתודה rotateRight(AVLNode node)
- המתודה rotateLeft(AVLNode node)
- המתודה rotateLeftThenRight(AVLNode node)
- המתודה rotateRightThenLeft(AVLNode node)

ומבצעת שאילתות

- השאילתה getBalanceFactor()
- השאילתה getLeft()

ב-$O(1)$ לכן סך הפעולות הוא $O(1)$.

2.4.32 הפעולה private boolean isUnbalanced(AVLNode node)

מה היא עושה: מחזירה האם גורם האיזון של הצומת בטווח המתאים.

סיבוכיות זמן הריצה: מבצעת שאילתת getBalanceFactor() ב-$O(1)$ לכן סך הפעולות הוא $O(1)$.

2.4.33 הפעולה private void rotateInDir(AVLNode node, Direction dir)

מה היא עושה: מבצעת פעולת איזון מסוג סיבוב יחיד ימינה או שמאלה בהינתן הכיוון dir.

כיצד היא פועלת: פועלת לפי האלגוריתם שראינו בכיתה אך בכלליות לשני הכיוונים: תחילה מעבירה את הילד המתאים, מתקנת מצביעים, מתקנת נכונות שדות ולבסוף מוודאת כי המצביע לשורש הוא נכון.

סיבוכיות זמן הריצה: ישנן קריאות לשאילותות ומתודות הרצות בסיבוכיות $O(1)$

השאילתות:

- השאילתה getDirectionFromParent(AVLNode node)
- השאילתה reverseDir()
- השאילתה getChildInDir(Direction dir)
- השאילתה getRoot()

הפעולות

- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה setParent(AVLNode node)
- הפעולה updateNodeFields()
- הפעולה setRoot(AVLNode root)

2.4.34 הפעולה private void rotateRight(AVLNode node)

מה היא עושה: מבצעת סיבוב יחיד ימינה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולה rotateInDir(AVLNode node, Direction dir) מתבצעת בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.35 הפעולה private void rotateLeft(AVLNode node)

מה היא עושה: מבצעת סיבוב יחיד שמאלה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולה rotateInDir(AVLNode node, Direction dir) מתבצעת בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.36 הפעולה private void rotateLeftThenRight(AVLNode node)

מה היא עושה: מבצעת סיבוב שמאלה וימינה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות

- הפעולה rotateRight(AVLNode node)
- הפעולה rotateLeft(AVLNode node)

מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.37 הפעולה private void rotateRightThenLeft(AVLNode node)

מה היא עושה: מבצעת סיבוב ימינה ושמאלה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות

- הפעולה rotateRight(AVLNode node)
- הפעולה rotateLeft(AVLNode node)

מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.38 הפעולה public int delete(int k)

מה היא עושה: מוחקת את הצומת בעל המפתח הנתון מהעץ אם הוא קיים ומחזירה את כמות פעולות האיזון שנדרשו לאיזון העץ, או -1 אם הצומת המתאים לא נמצא בעץ.

כיצד היא פועלת: תחילה הפעולה מוצאת את הצומת בעל המתפח הנתון, אם הוא קיים, מעדכנת את המצביעים של העוקב והקודם שלו באמצעות updateSuccessor(AVLNode node, AVLNode newNode) ולאחר מכן נוקטת באחד משמשת המקרים המתאימים:

1. הצומת הוא השורש וללא הילדים - משמע הוא הצומת היחידי בעץ

   נגדיר את העץ למצב של עץ "חדש" ללא צמתים ונחזיר 1 כנגד המחיקה היחדה שבוצעה.

   האיפוס יתבצע באמצעות הפעולות:

   - הפעולה setRoot(AVLNode root)
   - הפעולה setMin(AVLNode min)
   - הפעולה setMax(AVLNode max)

   אשר כל אחת פועלת ב-$O(1)$ בסיבוכיות כוללת של $O(1)$.

2. הצומת הוא המינימום בעץ:

   נעדכן את המינימום בעץ להיות העוקב של הצומת ונבצע מעקף עם בנו הימני - הבן היחידי שיכול להיות לו - אם אין לו בן אז הוא הלכה למעשה מוחלף בצומת וירטואלי פשוט כמו מחיקת עלה מעץ.

   זאת באמצעות השאילתות:

   - השאילתה getSuccessor()
   - השאילתה getRight()

- השאילתה getParent()
- השאילתה getRoot()
- השאילתה getVirtualNode()

אשר כולן בסיבוכיות $O(1)$ והפעולות:

- הפעולה setMin(AVLNode min)
- הפעולה setParent(AVLNode node)
- הפעולה setLeft(AVLNode node)
- הפעולה setParent(AVLNode node)

אשר גם הן בסיבוכיות $O(1)$ ולכן הסיבוכיות הכוללת היא $O(1)$.

3. הצומת הוא המקסימום בעץ:

נעדכן את המקסימום בעץ להיות הקודם של הצומת ונבצע מעקף עם בנו השמאלי - הבן היחידי שיכול להיות לו - אם אין לו בן אז הוא הלכה למעשה מוחלף בצומת וירטואלי פשוט כמו מחיקת עלה מעץ.

זאת באמצעות השאילתות:

- השאילתה getPredecessor()
- השאילתה getLeftt()
- השאילתה getParent()
- השאילתה getRoot()
- השאילתה getVirtualNode()

אשר כולן בסיבוכיות $O(1)$ והפעולות:

- הפעולה setMax(AVLNode min)
- הפעולה setParent(AVLNode node)
- הפעולה setRight(AVLNode node)
- הפעולה setParent(AVLNode node)

אשר גם הן בסיבוכיות $O(1)$ ולכן הסיבוכיות הכוללת היא $O(1)$.

4. הצומת לא השורש והוא אינו המינימום או המקסימום בעץ:

(א) לצומת בן יחיד:

נבצע מעקף עם בן זה.

נשתמש בשאילתות

- השאילתה getLeft()
- השאילתה isRealNode()
- השאילתה getChildInDir(Direction dir)
- השאילתה getParent()
- השאילתה getDirectionFromParent(AVLNode node)

והפעולות:

- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה setParent(AVLNode node)

(ב) לצומת שני בנים:

מאחר והקודם לצומת חייב להיות בתת העץ של אותו הצומת מאחר ויש לצומת שני ילדים נובע כי הקודם נמצא בתת העץ השמאלי ויתר על כן לא ייתכן כי יש לו ילד ימני אחרת הוא או ילד ימני שלו היה הקודם לכן נוכל לבצע מעקף לצומת הקודם ולאחר ש"שחררנו" אותו נחליף איתו את הצומת אותו אנו רוצים למחוק. לבסוף נבצע מעקף מהאב של הקודם או במקרה וזה בנו השמאלי של הצומת אז מהאיבר הקודם.

נבצע זאת באמצעות השאילתות:

- השאילתה getPredecessor()
- השאילתה getParent()

- השאילתה getLeft()
- השאילתה getDirectionFromParent(AVLNode node)
- השאילתה getRoot()

והפעולות:

- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה setParent(AVLNode node)
- הפעולה replaceChildren(AVLNode oldParent, AVLNode newParent)
- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה setRoot(AVLNode root)

ראינו כי כלל הפעולות והשאילתות מתבצעות בסיבוכיות $O(1)$.

(ג) הצומת הוא עלה בעץ:

נבצע מעקף עם הבן שלו שהוא צומת וירטואלי.

זאת באמצעות השאילתות:

- השאילתה getParent()
- השאילתה getDirectionFromParent(AVLNode node)
- השאילתה getVirtualNode()

והפעולה setChildInDir(AVLNode node, Direction dir) אשר כולן רצות בסיבוכיות $O(1)$.

ובסיום רצה הפעולה balanceNode(AVLNode node).

ראינו כי כלל הפעולות והשאילתות מתבצעות בסיבוכיות $O(1)$ למעט balanceNode(AVLNode node) הרצה בסיבוכיות $O(\log n)$ לכן הפעולה רצה בסיבוכיות $O(\log n)$.

2.4.39 הפעולה private void replaceChildren(AVLNode oldParent, AVLNode newParent)

מה היא עושה: מעבירה את ילדי oldParent להיות ילדי newParent.

כיצד היא פועלת: קוראת לפעולות setRight(AVLNode node), setLeft(AVLNode node) על מנת להגדיר את ילדי oldParent להיות ילדי newParent ואז משתמשת בsetParent(AVLNode node) על מנת להגיד את newParent בתור ההורה של ילדי oldParent.

סיבוכיות זמן הריצה: הראינו כי הפעולות

- השאילתה getRight()
- השאילתה getLeft()
- הפעולה setParent(AVLNode node)
- הפעולה setRight(AVLNode node)
- הפעולה setLeft(AVLNode node)

מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.40 הפעולה private Direction getDirectionFromParent(AVLNode node)

מה היא עושה: מחזירה את הכיוון באופן יחסי לאבא של הצומת, אם הצומת הנתון הוא השורש הכיוון שיוחזר הוא ימינה.

כיצד היא פועלת: קוראת לisLeftChild() ולפי ערך ההחזרה שלו מחזירה את הכיוון המתאים.

סיבוכיות זמן הריצה: הראינו כי הפעולה isLeftChild() מתבצעת בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.41 הפעולה private int balanceTree(AVLNode node)

מה היא עושה: מאזנת את העץ מהצומת הנתון ועד לשורש, בסיום מחזירה את כמות פעולות האיזון (ועוד אחת) שהתבצעו.

כיצד היא פועלת: מתחילה מהצומת הנתון וממשיכה בלולאה עד שהיא מגיעה לאבא של השורש (הצומת הוירטואלי) בקריאה ל balanceTreeOnce(AVLNode node) ומגדילה ב1 את המונה.

סיבוכיות זמן הריצה: הראינו כי הפעולה balanceTreeOnce(AVLNode node) מתבצעת בסיבוכיות אורך המסלול בין הצומת הנתון אל הצומת המוחזר. אנו קוראים בלולאה שוב ושוב לפעולה עם הערך המוחזר עד שאנו מגיעים לשורש כלומר בסיבוכיות של אורך המסלול מהצמת הנתון אל השורש שהיא חסומה בגובה העץ הלא הוא $O(\log n)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(\log n)$.

2.4.42 הפעולה public Boolean min()

מה היא עושה: מחזירה את ערכו של האיבר בעץ בעל המפתח המינימלי, או null אם העץ ריק.

סיבוכיות זמן הריצה: הראינו כי הפעולות empty(), getMin() ו-getValue() מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.43 הפעולה public Boolean max()

מה היא עושה: מחזירה את ערכו של האיבר בעץ בעל המפתח המקסימלי, או null אם העץ ריק.

סיבוכיות זמן הריצה: הראינו כי הפעולות empty(), getMax() ו-getValue() מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.44 הפעולה public void inOrder(AVLNode node, int offset, AVLNode[] arr)

מה היא עושה: פעולה רקורסיבית אשר מכניסה למערך arr החל מאינדקס $offset$ את הצמתים של תת-העץ ש-$node$ הוא שורשו ממוינים על פי המפתחות.

כיצד היא פועלת: פועלת בדומה להילוך in-order בעץ כפי שלמדנו בקורס, כאשר לאחר הקריאה הרקורסיבית לבן השמאלי, מכניסה כל צומת למערך arr באינדקס המתאים (אינדקס שהוא תוצאת החיבור של $offset$ וכמות הצמתים בתת-העץ שבנו השמאלי של $node$ הוא שורשו, על מנת שבמקומות שבין $offset$ לאינדקס זה יוכנסו למערך כל הצמתים בתת-העץ שבנו השמאלי של $node$ הוא שורשו, שאלו כל הצמתים שקטנים מ-$node$ בתת-העץ ש-$node$ הוא שורשו). לאחר ההכנסה למערך, מתבצעת קריאה רקורסיבית לבן הימני של $node$ עם $offset$ שהוא האינדקס העוקב של האינדקס אליו $node$ הוכנס במערך.
מקרה הבסיס של הפעולה הוא כאשר $node$ הוא צומת שאינו אמיתי, כלומר צומת וירטואלי (שכן הוא מייצג "סיום מסלול" מהשורש לצומת בעץ).

סיבוכיות זמן הריצה: בכל קריאה רקורסיבית של inOrder(AVLNode node, int offset, AVLNode[] arr) מתבצע מספר קבוע של קריאות לפעולות getLeft(), getRight() ו-getSize() אשר הראינו קודם כי מתבצעות ב-$O(1)$ זמן. הילוך מהצורה in-order עובר על כל הצמתים בעץ לכל היותר מספר קבוע של פעמים, לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O(n)$.

2.4.45 הפעולה public AVLNode[] nodesToArray()

מה היא עושה: יוצרת ומחזירה מערך אשר מכיל את כלל הצמתים בעץ ממוינים על פי המפתחות שלהם, או מערך ריק אם העץ ריק.

כיצד היא פועלת: יוצרת מערך כגודל העץ, ואם הוא לא ריק קוראת איתו ועם שורש העץ לפונקציה inOrder(AVLNode node, int offset, AVLNode[] arr), אשר כפי שהראינו מכניסה למערך arr החל מאינדקס 0 את הצמתים של העץ ממוינים על פי המפתחות.

סיבוכיות זמן הריצה: הפעולות empty() ו-size(), כמו גם שאר הפעולות חוץ מ-inOrder(AVLNode node, int offset, AVLNode[] arr), מתבצעות ב-$O(1)$ זמן, ואילו inOrder(AVLNode node, int offset, AVLNode[] arr) מתבצעת ב-$O(n)$ זמן. לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O(n)$.

2.4.46 הפעולה public int[] keysToArray()

מה היא עושה: מחזירה מערך ממוין המכיל את כל המפתחות בעץ, או מערך ריק אם העץ ריק.

כיצד היא פועלת: משתמשת בפעולה nodesToArray() על מנת לקבל מערך של הצמתים בעץ ממוינים על פי המפתחות שלהם, ולאחר מכן בעזרת לולאה מכניסה את המפתחות בסדר הממוין למערך arr.

סיבוכיות זמן הריצה: הפעולות empty() ו-size() מתבצעות ב-$O(1)$ זמן, ואילו nodesToArray() כמו גם הלולאה מתבצעות ב-$O(n)$ זמן. לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O(n)$.

2.4.47 הפעולה public boolean[] infoToArray()

מה היא עושה: מחזירה מערך בוליאנים המכיל את כל הערכים בעץ, ממויינים על פי סדר המפתחות, או מערך ריק אם העץ ריק.

כיצד היא פועלת: משתמשת בפעולה nodesToArray() על מנת לקבל מערך של הצמתים בעץ ממוינים על פי המפתחות שלהם, ולאחר מכן בעזרת לולאה מכניסה את הערכים שלהם בסדר הממוין למערך arr.

סיבוכיות זמן הריצה: הפעולות empty() ו-size() מתבצעות ב-$O(1)$ זמן, ואילו nodesToArray() כמו גם הלולאה מתבצעות ב-$O(n)$ זמן. לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O(n)$.

2.4.48 הפעולה public boolean prefixXor(int k)

מה היא עושה: מקבלת מפתח k כאשר נתון ש-k נמצא במבנה ומחזירה את תוצאת פעולת xor על הערכים הבוליאניים הנמצאים במבנה תחת מפתחות שקטנים או שווים ל-k.

כיצד היא פועלת: מבצעת הילוך בעץ במסלול שבין השורש לבין הצומת עם המפתח k. אם הצומת הנוכחי הוא עם מפתח גדול יותר מ-k, ממשיכים במסלול (ולא מחשבים אותו כחלק מ-xor). אם הצומת הנוכחי הוא עם מפתח קטן או שווה ל-k, נחשב את הערכים הבוליאניים שלו ושל הצמתים בתת-העץ השמאלי שלו ב-xor (בעזרת הפעולה getSubTreeXor()), שכן כל אלו הם צמתים עם מפתחות קטנים מ-k. כל הצמתים עם המפתחות שקטנים מ-k הם כל שנמצאים בעץ "משמאל" למסלול בין השורש לבין הצומת עם המפתח k (כולל משמאל לצומת עם המפתח k עצמו).

סיבוכיות זמן הריצה: הראינו כי הפעולות getRoot(), getKey(), getLeft(), getRight(), getValue() ו-getSubTreeXor הן פעולות המתבצעות ב-$O(1)$ זמן. לפי ההסבר של הפעולה, הלולאה מבצעת לכל היותר $h+1$ כאשר h הוא גובה העץ (שכן זהו אורך המסלול הארוך ביותר בין שורש לצומת בעץ), וכן בעץ AVL מתקיים $h = O(\log n)$. לכן סיבוכיות זמן הריצה הכוללת היא $O(\log n)$.

2.4.49 הפעולה public AVLNode successor(AVLNode node)

מה היא עושה: מקבלת צומת בעץ כקלט ומחזירה את העוקב שלו. אם לא קיים עוקב, מחזירה null.

סיבוכיות זמן הריצה: הראינו כי הפעולה getSuccessor() היא פעולה המתבצעת ב-$O(1)$ זמן, לכן סיבוכיות זמן הריצה היא $O(1)$.

2.4.50 הפעולה public boolean succPrefixXor(int k)

מה היא עושה: מקבלת מפתח k כאשר נתון ש-k נמצא במבנה ומחזירה את תוצאת פעולת xor על הערכים הבוליאניים הנמצאים במבנה תחת מפתחות שקטנים או שווים ל-k.

כיצד היא פועלת: מתחילה מהצומת המינימלי של העץ, ומבצעת עליו פעולות successor(AVLNode node) עד שמגיעה לצומת בעל המפתח k. עבור כל צומת שהתקבל בפעולות ה-successor(AVLNode node) (וכן הצומת המינימלי עצמו), הפעולה מחשבת xor עם ערכו הבוליאני, ולבסוף מחזירה את התוצאה.

סיבוכיות זמן הריצה: הראינו כי הפעולות getMin(), getKey() ו-getValue() הן פעולות המתבצעות ב-$O(1)$ זמן. הפעולה מבצעת מספר איטרציות כמספר הצמתים בעץ עם מפתחות שקטנים או שווים למפתח k, ובכל איטרציה מבצעת פעולת xor ופעולת successor(AVLNode node) ב-$O(1)$ זמן. כלומר מתבצעות לכל היותר n איטרציות (כמספר סך כל הצמתים בעץ), לכן סיבוכיות זמן הריצה הכוללת היא $O(n)$.