מבני נתונים - פרויקט מעשי עץ AVL

20 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 צומת וירטואלי.
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 צומת וירטואלי.

1.2 בנאיי 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)

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.
- השדה $successor$ מאותחל לצומת הוירטואלי של העץ (AVLTree) בו הוא נוצר.

- השדה $predecessor$ 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 (אם הצומת וירטואלי מוחזר -1).

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, או null אם הצומת וירטואלי.

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 הנוכחי הוא אמיתי, קובעת את $node$ כבן השמאלי שלו.

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 הנוכחי הוא אמיתי, קובעת את $node$ כבן הימני שלו.

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: משימה את $node$ כבן השמאלי או הימני של הצומת הנוכחי, כתלות בכיוון 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: מחזירה את הבן השמאלי או הימני של הצומת, כתלות בכיוון 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 הנוכחי הוא אמיתי, קובעת את $node$ כאביו.

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, או null אם לצומת אין אבא.

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: אם הצומת הנוכחי הוא אמיתי, קובעת את $successor$ כעוקב שלו.

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

1.3.19 הפעולה public AVLNode getSuccessor()

מה היא עושה: אם הצומת הנוכחי הוא אמיתי, מחזירה את הצומת העוקב של הצומת הנוכחי, אחרת מחזירה צומת וירטואלי.

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

1.3.20 הפעולה public void setPredecessor(AVLNode predecessor)

מה היא עושה: אם הצומת הנוכחי הוא אמיתי, קובעת את $predecessor$ כקודם שלו.

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

1.3.21 הפעולה public AVLNode getPredecessor()

מה היא עושה: אם הצומת הנוכחי הוא אמיתי, מחזירה את הצומת הקודם של הצומת הנוכחי, אחרת מחזירה צומת וירטואלי.

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, אחרת מחזירה 0.

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() וגישה לשדה 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: פעולה 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. אם העץ הוא עץ ריק, $root$ מצביע לערך null.
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()

מה הוא עושה: יוצר אובייקט מטיפוס AVLTree אשר ניתן להתחיל להכניס אליו מפתחות ומידע בעזרת הפעולה insert(int k, boolean i), ומאתחל את הצומת הוירטואלי של העץ בעזרת קריאה לבנאי AVLNode().

סיבוכיות זמן הריצה: ראינו כי AVLNode() מתבצע ב-$O(1)$ זמן, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

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 (אם מכניסים צומת עם מפתח k לעץ, ולפני ביצוע פעולות איזון).

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 (לפני ביצוע פעולות האיזון), 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 המידע של צומת 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 protected void updateSuccessor(AVLNode node, AVLNode newNode)

מה היא עושה: מגדירה את העוקב של node להיות newNode ואת הקודם של newNode להיות node.

סיבוכיות זמן הריצה: הראינו כי הפעולות setPredecessor(AVLNode predecessor) ו-setSuccessor(AVLNode successor) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

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.

כיצד היא פועלת: משימה את newNode כבן השמאלי של parent, מעדכנת אותו להיות המינימום של העץ במידת הצורך וכן מעדכנת את שדות העוקב והקודם בצמתים הרלוונטים.

סיבוכיות זמן הריצה: הראינו כי הפעולות setLeft(AVLNode node), getMin(), setMin(AVLNode min) ו-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.

כיצד היא פועלת: משימה את newNode כבן הימני של parent, מעדכנת אותו להיות המקסימום של העץ במידת הצורך וכן מעדכנת את שדות העוקב והקודם בצמתים הרלוונטים.

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

2.4.15 הפעולה private Direction getDirectionFromParent(AVLNode node)

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

כיצד היא פועלת: קוראת לפעולה isLeftChild() 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 $O(1)$.

2.4.16 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: מבצעת גלגול באופן שראינו בהרצאה. הפעולה מעדכנת יחסים בין אבות לילדים, "מזיזה" תתי עצים למקומות חדשים מתאימים, מתקנת מצביעים, ומעדכנת שדות של צמתים על מנת לשמר את הנכונות. נציין כי לאחר הגלגול עצמו, הפעולה מעדכנת את המצביע לשורש העץ במידת הצורך.

סיבוכיות זמן הריצה: ישנן גישה לשדה ומספר קבוע של קריאות לפעולות שהראינו שרצות בסיבוכיות $O(1)$:

- הפעולה getDirectionFromParent(AVLNode node)

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

לכן סך הכל סיבוכיות זמן הריצה של הפונקציה היא $O(1)$.

2.4.17 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 $O(1)$.

2.4.18 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 $O(1)$.

2.4.19 הפעולה private void rotateLeftThenRight(AVLNode node)

מה היא עושה: מבצעת סיבוב שמאלה של הילד השמאלי של הצומת הנתון ואז מסובבת ימינה את הצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות rotateLeft(AVLNode node) ו-rotateRight(AVLNode node) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.20 הפעולה private void rotateRightThenLeft(AVLNode node)

מה היא עושה: מבצעת סיבוב ימינה של הילד הימני של הצומת הנתון ואז מסובבת שמאלה את הצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות rotateLeft(AVLNode node) ו-rotateRight(AVLNode node) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.21 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

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

ומבצעת מספר קבוע של שאילתות getRight(), getLeft(), getBalanceFactor(), כולן ב-$O(1)$ זמן (גם הגלגולים). לכן סך כל זמן הריצה הוא $O(1)$.

2.4.22 הפעולה private boolean isUnbalanced(AVLNode node)

מה היא עושה: מחזירה האם הצומת הוא "עבריין AVL" (כלומר האם גורם האיזון אינו בטווח הרצוי).

סיבוכיות זמן הריצה: הפעולה מבצעת לכל היותר שתי קריאות ל-getBalanceFactor() ב-$O(1)$ זמן, לכן לכן סיבוכיות זמן הריצה היא $O(1)$.

2.4.23 הפעולה protected int updateNode(AVLNode node)

מה היא עושה: מעדכנת את השדות של הצומת לאחר הכנסה/מחיקה. אם הצומת הוא "עבריין AVL", מאזנת אותו. אם התבצע שינוי גובה/פעולת גלגול, הפעולה מחזירה 1, אחרת מחזירה 0.

כיצד היא פועלת: שומרת את הגובה של הצומת לפני פעולות האיזון, ולאחר שמעדכנת אותו ואת שאר השדות באמצעות הפעולה updateNodeFields(), בודקת האם הגובה השתנה או שהצומת הוא "עבריין AVL". אם התשובה לאחת מהשאלות האלו היא כן, מפעילה על הצומת את הפעולה balanceNode(AVLNode node) (אשר מבצעת גלגול אם יש צורך), ומחזירה 1 שכן התבצעה פעולת איזון. אחרת, מחזירה 0.

סיבוכיות זמן הריצה: מבצעת את הפעולות getHeight(), updateNodeFields(), isUnbalanced(AVLNode node) ו-balanceNode(AVLNode node), כולן ב-$O(1)$ זמן. לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

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 ומשתמשת בפעולה setParent(AVLNode node) על מנת להגדיר את newParent בתור ההורה של הצמתים שהיו הילדים של oldParent.

סיבוכיות זמן הריצה: הראינו כי הפעולות getRight(), setRight(AVLNode node), setParent(AVLNode node), getLeft(), setLeft(AVLNode node) מתבצעות בסיבוכיות זמן $O(1)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(1)$.

2.4.25 הפעולה protected int balanceTree(AVLNode node)

מה היא עושה: מאזנת את העץ מהצומת הנתון ועד לשורש, מעדכנת את השדות של הצמתים הרלוונטים ובסיום מחזירה את כמות פעולות האיזון שהתבצעו.

כיצד היא פועלת: מתחילה מהצומת הנתון וממשיכה לעלות בעץ בלולאה עד שהיא מגיעה לאבא של השורש (הצומת הוירטואלי), כאשר בכל איטרציה קוראת לפעולה updateNode(AVLNode node) על מנת לעדכן את השדות המתאימים של כל צומת ולבצע עליהם גלגולים אם יש צורך. כמו כן, בכל איטרציה מוסיפה למונה של פעולות האיזון 0 או 1 בהתאם לערך ההחזרה של updateNode(AVLNode node).

סיבוכיות זמן הריצה: הראינו כי הפעולה updateNode(AVLNode node) מתבצעת ב-$O(1)$ זמן. אנו קוראים בלולאה שוב ושוב לפעולה עם כל צומת בנתיב עד שמגיעים לאביו הוירטואלי של השורש, כלומר המסלול שאנו מבצעים הוא מסלול מהצומת הנתון אל השורש. מסלול זה חסום בגובה העץ, שהוא $O(\log n)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(\log n)$.

2.4.26 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:

נחלק למקרים:

1. אם העץ ריק,

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

2. אם העץ לא ריק,

   משתמשת בפעולה searchNode(int k) כדי למצוא את הצומת שאמור להיות אביו של הצומת החדש (לפני פעולות האיזון), מכניסה את הצומת החדש בתור בנו בעזרת updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode) או updateRelationsForNewRightChild(AVLNode parent, AVLNode newNode) ולאחר מכן מאזנת את העץ ומעדכנת את השדות המתאימים בעזרת הפעולה balanceTree(AVLNode node).

סיבוכיות זמן הריצה: אם העץ ריק, הראינו כי הפעולות setRoot(AVLNode root), setMin(AVLNode min), setMax(AVLNode max) ו-empty() רצות ב-$O(1)$ זמן לכן סה"כ $O(1)$.
אם העץ אינו ריק, קריאה לפעולה searchNode(int k) מתבצעת ב-$O(\log n)$ זמן, קריאות לפעולות updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode) או updateRelationsForNewRightChild(AVLNode parent, AVLNode newNode) מתבצעות ב-$O(1)$ זמן וקריאה לפעולה balanceTree(AVLNode node) מתבצעת ב-$O(\log n)$ זמן.
לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O(\log n)$.

2.4.27 הפעולה public int delete(int k)

מה היא עושה: אם קיים בעץ צומת בעל המפתח k, מוחקת אותו מהעץ ומחזירה את כמות פעולות האיזון שנדרשו לאיזון העץ. אם לא קיים כזה צומת בעץ, הפעולה מחזירה -1.

כיצד היא פועלת: תחילה הפעולה מחפשת בעץ צומת עם המפתח הנתון. אם הוא לא קיים, מחזירה -1. אחרת, אם הוא קיים, מעדכנת מצביעים של העוקב והקודם שלו באמצעות הפעולות updateSuccessor(AVLNode node, AVLNode newNode), getSuccessor(), getPredecessor() ולאחר מכן נוקטת באחד משש המקרים המתאימים (בין היתר באמצעות בדיקה של כמות הילדים בעזרת getChildCount()):

1. אם הצומת הוא השורש וללא ילדים, משמע הוא הצומת היחידי בעץ, נגדיר את העץ להיות עץ ריק ונחזיר 0 כנגד המחיקה היחידה שבוצעה.
   איפוס העץ יתבצע באמצעות הפעולות:
   - הפעולה getRoot()
   - הפעולה setRoot(AVLNode root)
   - הפעולה setMin(AVLNode min)
   - הפעולה setMax(AVLNode max)
2. אם 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 ימני אז המעקף מתבצע עם צומת וירטואלי, שמדמה מחיקת עלה מעץ ושומר על נכונות המבנה. נציין כי אם הצומת הנמחק היה השורש, אין לו אב ולכן מגדירים את הבן הימני שלו כשורש.
   זאת באמצעות הפעולות:
   - הפעולה getMin()
   - הפעולה setMin(AVLNode min)
   - הפעולה getSuccessor()
   - הפעולה getRight()
   - הפעולה setParent(AVLNode node)
   - הפעולה getRoot()
   - הפעולה setLeft(AVLNode node)
   - הפעולה setRoot(AVLNode root)
   - הפעולה getVirtualNode()
3. אם 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 שמאלי אז המעקף מתבצע עם צומת וירטואלי, שמדמה מחיקת עלה מעץ ושומר על נכונות המבנה. נציין כי אם הצומת הנמחק היה השורש, אין לו אב ולכן מגדירים את הבן השמאלי שלו כשורש.
   זאת באמצעות הפעולות:
   - הפעולה getMax()
   - הפעולה setMax(AVLNode min)
   - הפעולה getPredecessor()
   - הפעולה getLeft()
   - הפעולה setParent(AVLNode node)
   - הפעולה getRoot()
   - הפעולה setRight(AVLNode node)
   - הפעולה setRoot(AVLNode root)

- הפעולה getVirtualNode()

4. אם הצומת הוא לא השורש והוא אינו המינימום או המקסימום בעץ, נחלק למקרים הבאים:

(א) אם לצומת בן יחיד, נבצע מעקף עם בן זה, בשימוש בפעולות

- הפעולה getLeft()
- הפעולה isRealNode()
- הפעולה getChildInDir(Direction dir)
- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה getDirectionFromParent(AVLNode node)
- הפעולה setParent(AVLNode node)

(ב) אם לצומת שני בנים, כפי שראינו בהרצאה הקודם לצומת חייב להיות בתת העץ של אותו הצומת, בפרט בתת העץ השמאלי. כמו כן, לא ייתכן כי לקודם יש ילד ימני, אחרת אחד הצמתים בתת העץ של הילד הימני היה הקודם. מכך נובע כי ניתן לבצע מעקף לצומת הקודם, ולהחליף אותו פיזית בצומת אותו אנו מוחקים מהעץ. נציין כי אם הצומת הנמחק היה השורש, נעדכן את השורש להיות הקודם שלו.
נבצע זאת באמצעות הפעולות:

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

(ג) אם לצומת אין בנים כלל, כלומר הוא עלה בעץ, נמחק אותו ונגדיר את הבן המתאים של אביו להיות צומת וירטואלי, זאת באמצעות הפעולות:

- הפעולה getDirectionFromParent(AVLNode node)
- הפעולה setChildInDir(AVLNode node, Direction dir)
- הפעולה getVirtualNode()

בכל מקרה, בסיום, רצה הפעולה balanceTree(AVLNode node) אשר מאזנת את העץ ומעדכנת את השדות המתאימים.

סיבוכיות זמן הריצה: תחילה מתבצע שימוש בפעולה empty() ב-$O(1)$ זמן, ואם העץ לא ריק גם searchNode(int k) ב-$O(\log n)$ זמן.
אם הצומת עם המפתח הנתון לא קיים בעץ, נפסיק את ריצת הפעולה.
אחרת, נשים לב כי כלל הפעולות בהן היה שימוש (וכל אחת מספר קבוע של פעמים) בכל אחד מהמקרים מתבצעות ב-$O(1)$ זמן, כמו גם גישות לשדה.
כמו כן, הפעולה balanceTree(AVLNode node) רצה בסיבוכיות זמן $O(\log n)$.
אזי סך הכל סיבוכיות זמן הריצה של הפעולה היא $O(\log n)$.

2.4.28 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 $O(1)$.

2.4.29 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 $O(1)$.

2.4.30 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 $O(n)$.

2.4.31 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 $O(n)$.

2.4.32 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 $O(n)$.

2.4.33 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 $O(n)$.

2.4.34 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 אלו 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 $h+1$ איטרציות כאשר h הוא גובה העץ (שכן זהו אורך המסלול הארוך ביותר בין השורש לצומת בעץ), וכן בעץ AVL מתקיים $h = O(\log n)$. לכן סיבוכיות זמן הריצה הכוללת היא $O(\log n)$.

2.4.35 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 הפעולות getSuccessor() ו-isRealNode() הן פעולות המתבצעות ב-$O(1)$ זמן, לכן סיבוכיות זמן הריצה היא $O(1)$.

2.4.36 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 מהמפתח 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 $O(n)$.

3 מדידות

הערה. תוצאות המדידות בטבלאות נכתבו במילי-שניות (ms).

3.1

מספר סידורי	עלות prefixXor ממוצעת (כל הקריאות)	עלות succPrefixXor ממוצעת (כל הקריאות)	עלות prefixXor ממוצעת (100 קריאות ראשונות)	עלות succPrefixXor ממוצעת (100 קריאות ראשונות)
1	0.054200	0.276200	0.011000	0.017900
2	0.124900	5.192400	0.011600	0.017100
3	0.204000	8.186000	0.012700	0.019400
4	0.334500	14.793000	0.012200	0.019200
5	0.377400	68.398800	0.014000	0.018700

ממוצע זמן הקריאות של prefixXor יותר קטן מממוצע זמן הקריאות של succPrefixXor, גם בממוצע על כל הקריאות וגם בממוצע על 100 הקריאות הראשונות. כלומר, התוצאות מתיישבות עם הניתוח התיאורטי של סיבוכיות הזמן, ואף נשים לב כי הגדילה בעלות הממוצעת של כל הקריאות ל-succPrefixXor הינה משמעותית יותר מאשר הגדילה הממוצעת של כל הקריאות ל-prefixXor, דבר שמסתדר עם העובדה שממוצע סיבוכיות זמן הריצה של succPrefixXor על כל הקריאות הוא לינארי בגודל הקלט, ואילו ממוצע סיבוכיות זמן הריצה של prefixXor הוא לוגריתמי בכמות הצמתים בעץ.

3.2

מספר סידורי/עלות הכנסה ממוצעת	עץ AVL - סדרה חשבונית	עץ ללא מנגנון איזון - סדרה חשבונית	עץ AVL - סדרה מאוזנת	עץ ללא מנגנון איזון - סדרה מאוזנת	עץ AVL - סדרה אקראית	עץ ללא מנגנון איזון - סדרה אקראית
1	0.183200	2.890700	0.179900	0.135800	0.233300	0.139600
2	0.435700	7.705900	0.369900	0.251000	0.573000	0.360200
3	0.687800	30.716700	0.566100	0.424600	0.790800	1.090900
4	1.222900	41.084200	0.585200	0.560800	0.852500	0.489700
5	1.267000	59.937100	0.706600	0.805100	1.101400	0.621500

עבור סדרה חשבונית, היינו מצפים שזמן הריצה בעץ AVL יהיה משמעותית נמוך יותר מזמן הריצה בעץ ללא מנגנון איזון, שכן סדרת ההכנסה לא מאוזנת כלל, ובהכנסה לעץ ללא מנגנון איזון נקבל עץ בצורת "שרוך", שהוא בעל גובה לינארי בגודל הקלט, וזה משתקף במדידות.

עבור סדרה מאוזנת, היינו מצפים שזמן הריצה של עץ ללא מנגנון איזון יהיה מהיר יותר מזה של עץ AVL שכן בשני המקרים נקבל עץ מאוזן, ואילו בעץ AVL אנו מבצעים "עבודה עודפת" של עלייה מהצומת המוכנס אל השורש על מנת לעדכן את שדות הגובה (ולבצע גלגולים במקרה שהעץ לא מאוזן). בפועל, תוצאות המדידה אכן מסתדרות עם ציפייה זו.

עבור סדרה אקראית, ראינו כי בתוחלת גובה של עץ ללא מנגנון איזון יהיה לוגריתמי בגודל הקלט, כלומר בתוחלת אמורים לקבל תוצאות דומות לאלו של הכנסת סדרה מאוזנת, וזה אכן מה שמקבלים (עם סטיות מועטות).