AVL מבני נתונים - פרויקט מעשי עץ

2021 במאי 6

תוכן העניינים

4	המחלקה AVLNode			1
4	מחלקה:			
4	מחלקה:	בנאי המחל		
4	public AVLNode() הבנאי הריק	1.2.1		
4	public AVLNode(int key, boolean info, AVLNode parent) הבנאי	1.2.2		
5	המחלקה:	פעולות	1.3	
5	public boolean isRealNode() הפעולה	1.3.1		
5	public int getKey() הפעולה	1.3.2		
5	public Boolean getValue() הפעולה	1.3.3		
5	public void setLeft(AVLNode node) הפעולה	1.3.4		
5	public AVLNode getLeft() הפעולה	1.3.5		
5	public void setRight(AVLNode node) הפעולה	1.3.6		
5	public AVLNode getRight() הפעולה	1.3.7		
6	public void setChildInDir(AVLNode node, Direction dir) הפעולה	1.3.8		
6	public AVLNode getChildInDir(Direction dir) הפעולה	1.3.9		
6	public void setParent(AVLNode node) הפעולה	1.3.10		
6	public AVLNode getParent() הפעולה	1.3.11		
6	public void setHeight(int height) הפעולה	1.3.12		
6	public int getHeight() הפעולה	1.3.13		
6	public void setSize(int size) הפעולה	1.3.14		
6	public int getSize() הפעולה	1.3.15		
6	public void setSubTreeXor(boolean xor) הפעולה	1.3.16		
7	public boolean getSubTreeXor() הפעולה	1.3.17		
7	public void setSuccessor(AVLNode successor) הפעולה	1.3.18		
7		1.3.19		
7	public void setPredecessor(AVLNode predecessor) הפעולה	1.3.20		
7		1.3.21		
7	public int getBalanceFactor() הפעולה	1.3.22		
7		1.3.23		
7		1.3.24		
8		1.3.25		

8	AVLT	ולקה ree	המח
8	Direction המנייה	טיפוס	2.1
8	ערכי טיפוס המנייה	2.1.1	
8	public Direction reverseDir() הפעולה	2.1.2	
8	זמחלקה:	שדות ר	2.2
8	מחלקה:	בנאי ה	2.3
8		2.3.1	
8		פעולות	2.4
8		2.4.1	
8	public AVLNode getRoot() הפעולה	2.4.2	
9	public void setRoot(AVLNode root) הפעולה	2.4.3	
9	public boolean empty() הפעולה	2.4.4	
9	public int size() הפעולה	2.4.5	
9	public AVLNode getMin() הפעולה	2.4.6	
9	public void setMin(AVLNode min) הפעולה	2.4.7	
9	public AVLNode getMax() הפעולה	2.4.8	
9	public void setMax(AVLNode max) הפעולה	2.4.9	
9	public AVLNode searchNode(int k) הפעולה	2.4.10	
10	public Boolean search(int k) הפעולה	2.4.11	
10	public int insert(int k, boolean i) הפעולה	2.4.12	
10	$private\ void\ update Relations For New Left Child\ (AVLNode\ parent,\ AVLNode\ new Node)$ הפעולה הפעולה	2.4.13	
10	private void updateRelationsForNewRightChild(AVLNode parent, AVLNode newNode) פעולה	2.4.1	4
10	private AVLNode balanceTreeOnce(AVLNode node) הפעולה	2.4.15	
11	private void balanceNode(AVLNode node) הפעולה	2.4.16	
11	private boolean isUnbalanced(AVLNode node) הפעולה	2.4.17	
11	private void rotateInDir(AVLNode node, Direction dir) הפעולה	2.4.18	
11	private void rotateRight(AVLNode node) הפעולה	2.4.19	
12	private void rotateLeft(AVLNode node) הפעולה	2.4.20	
12	private void rotateLeftThenRight(AVLNode node) הפעולה	2.4.21	
12	private void rotateRightThenLeft(AVLNode node) הפעולה	2.4.22	
12	public int delete(int k) הפעולה	2.4.23	
14	private void replaceChildren(AVLNode oldParent, AVLNode newParent) הפעולה	2.4.24	
14	private Direction getDirectionFromParent(AVLNode node) הפעולה	2.4.25	
15	private int balanceTree(AVLNode node) הפעולה	2.4.26	
15	public int insert(int k, boolean i) הפעולה	2.4.27	
15	$private\ void\ update Relations For New Left Child\ (AVLNode\ parent,\ AVLNode\ new Node)$ הפעולה	2.4.28	
15	private void updateRelationsForNewRightChild(AVLNode parent, AVLNode newNode) פעולה	2.4.2	9
15	private AVLNode balanceTreeOnce(AVLNode node) הפעולה	2.4.30	
16	private void balanceNode(AVLNode node) הפעולה	2.4.31	

16	private void rotateInDir(AVLNode node, Direction dir) הפעולה	2.4.33
16	private void rotateRight(AVLNode node) הפעולה	2.4.34
17	private void rotateLeft(AVLNode node) הפעולה	2.4.35
17	private void rotateLeftThenRight(AVLNode node) הפעולה	2.4.36
17	private void rotateRightThenLeft(AVLNode node) הפעולה	2.4.37
17	public int delete(int k) הפעולה	2.4.38
19	private void replaceChildren(AVLNode oldParent, AVLNode newParent) הפעולה	2.4.39
19	private Direction getDirectionFromParent(AVLNode node) הפעולה	2.4.40
20	private int balanceTree(AVLNode node) הפעולה	2.4.41
20	public Boolean min() הפעולה	2.4.42
20	public Boolean max() הפעולה	2.4.43
20	public void inOrder(AVLNode node, int offset, AVLNode[] arr) הפעולה	2.4.44
20	public AVLNode[] nodesToArray() הפעולה	2.4.45
21	public int[] keysToArray() הפעולה	2.4.46
21	public boolean[] infoToArray() הפעולה	2.4.47
21	public boolean prefixXor(int k) הפעולה	2.4.48
21	public AVLNode successor(AVLNode node) הפעולה	2.4.49
21	nublic boolean succeptefix Yor(int k) Thuan	2 4 50

AVLNode המחלקה

1.1 שדות המחלקה:

- m .AVL אשר מייצג את המפתח של צומת m key מסוג ווt הקבוע הפרטי m key .1
- .AVL אשר מייצג את המידע של Boolean מסוג info מסוג מסוג.
- . המשתנה הפרטי אשר אות אשר מייצג את מייצג את height מסוג ווה המשתנה הפרטי. 3
- AVL אשר מייצג את אביו הישיר אבע AVLNode מסוג מייצג את אביו הישיר אבע מייצג את אבע אומת parent 4.
- .AVL אשר מייצג את בנו השמאלי הישיר של אומת זה בעץ AVLNode המשתנה הפרטי. המשתנה teft
- .AVL אשר מייצג את בנו הימני הישיר של AVLNode מסוג right מסוג הפרטי
- . המשתנה הפרטי subTreeSize מסוג int אשר מייצג את גודל תת-העץ אשר צומת זה הוא שורשו.
- xor אשר מעולת את פעולת הפרטי perfixXor אשר משמש למימוש אשר boolean מסוג מסוג subTreeXor מסוג את הפרטי אל שדות info של שדות של כל הצמתים בתת-העץ אשר צומת זה הוא שורשו.
- חמפתחות המפתח העוקב (הצומת או AVLNode אשר מייצג את הצומת העוקב (הצומת או successor מסוג מסוג אשר מייצג את הצומת האוא האיבר עם המפתח המקסימלי בעץ, ערך שדה זה יהיה AVL של הצמתים בעץ) של צומת זה בעץ ערך שדה או האיבר עם המפתח המקסימלי בעץ.
- בקבוצת המפתח הפרטי predecessor מסוג AVLNode אשר מייצג את הצומת הקודם (הצומת עם המפתח הקודם בקבוצת בקבוצת אות 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:

public AVLNode() הבנאי הריק 1.2.1

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.

 $O\left(1
ight)$ איז מתבצע מספר קבוע של השמות, ולכן סיבוכיות זמן הריצה הכוללת היא

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

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

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

- . השדה key מאותחל לפרמטר key שניתן על ידי המשתמש •
- . השדה $in \, fo$ מאותחל לפרמטר $in \, fo$ שניתן על ידי המשתמש
- . השדה height מאותחל ל-0, שהוא גובה של תת-עץ בעל צומת אחד.
- . שניתן על ידי המשתמש parent שניתן לפרמטר אותחל parent מאותחל
- של אסר אסר מאותחל אסר אכן אבן אדי המשתמש, שכן אידי אניתן לפרמטר לפרמטר לפרמטר אותחל אותחל אותחל אות השל השליח אניתן אל אידי המשתמש אותחל לפרמטר לפרמטר אותחל אות
- השדה לצומת, הוירטואלי של העץ (AVLTree) בו הוא נוצר, שכן הייצוג של חוסר קיום ילד לצומת, הוא פולד לצומת הוירטואלי. בו הוא נוצר, שכן הייצוג של חוסר קיום ילד לצומת, הוא צומת וירטואלי.
 - בו הוא נוצר. (AVLTree) מאותחל לצומת הוירטואלי של העץ right השדה •

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 AVI.Tree

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

1.3 פעולות המחלקה:

public boolean isRealNode() הפעולה 1.3.1

מה היא עושה: מחזירה האם הצומת הוא אמיתי.

 $O\left(1
ight)$ שיבוכיות זמן הריצה: ישנה השוואה וגישה אחת לשדה לכן

public int getKey() הפעולה 1.3.2

מה היא עושה: מחזירה את המפתח של הצומת.

 $O\left(1\right)$ שיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public Boolean getValue() הפעולה 1.3.3

מה היא עושה: מחזירה את המידע של הצומת.

ארצה היא וכן הריצה, לכן סיבוכיות (טיבו isRealNode) ארצה אחת לשדה, לכן סיבוכיות אחת הריצה היא isRealNode) ארצה ב- $O\left(1\right)$.

public void setLeft(AVLNode node) הפעולה 1.3.4

. הנוכחי האומת של הצומת כבן השמאלי אמיתי, קובעת אמיתי, האומת הצומת הצומת מה היא עושה: אם הצומת המוכחי

ארצה היא וכן הריצה, לכן סיבוכיות (טיבו isRealNode) ארצה אחת לשדה, לכן סיבוכיות אחת הריצה היא isRealNode) ארצה ב- $O\left(1\right)$.

public AVLNode getLeft() הפעולה 1.3.5

מה היא עושה: מחזירה את הבן השמאלי של הצומת.

 $O\left(1
ight)$ אחת לשדה לכן הריצה: ישנה ישנה לשדה לכן

public void setRight(AVLNode node) הפעולה 1.3.6

. הנוכחי הצומת של הימני אותו כבן הימני אמיתי, הוא אמיתי, הוא החוא הצומת אם הצומת אם היא עושה: אם הצומת הנוכחי

ארצה היא וכן הריצה, לכן סיבוכיות (טיבו isRealNode) ארצה אחת לשדה, לכן סיבוכיות אחת הריצה היא isRealNode) ארצה ב- $O\left(1\right)$.

public AVLNode getRight() הפעולה 1.3.7

מה היא עושה: מחזירה את הבן הימני של הצומת.

 $O\left(1
ight)$ אחת לשדה לכן ישנה ישנה אחת לבדה לכן

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

.dira בהינתן לכיוון הנתון setLeft(AVLNode node) או setRight(AVLNode node) בהינתן לכיוון הנתון מה היא עושה:

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

public AVLNode getChildInDir(Direction dir) הפעולה 1.3.9

.dir בהינתן לכיוון הנתון getRight() מה היא עושה: קוראת ל

או (1), כן סיבוכיות מן הריצה ואר שנה היא getLeft() או (1) או וואה וקריאה לפעולה וקריאה לפעולה ישנה היא פעלה ($O\left(1\right)$.

public void setParent(AVLNode node) הפעולה 1.3.10

. הנוכחי הצומת אם אותו אותו אמיתי, קובעת הוא האומת node אם הצומת אותו מה היא עושה:

סיבוכיות אחת לשדה, לכן סיבוכיות ומן הריצה היא isRealNode() שרצה לפעולה (שנה קריאה לפעולה ומן הריצה היא $O\left(1\right)$ וכן גישה אחת לשדה, לכן סיבוכיות ומן הריצה היא $O\left(1\right)$.

public AVLNode getParent() הפעולה 1.3.11

מה היא עושה: מחזירה את אביו הישיר של הצומת.

 $O\left(1\right)$ אחת לשדה לכן פיבוכיות אמן הריצה: ישנה גישה אחת לשדה לכן

public void setHeight(int height) הפעולה 1.3.12

.height אומת להיות של הגובה את קובעת את קובעת מה היא עושה:

 $O\left(1\right)$ סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public int getHeight() הפעולה 1.3.13

מה היא עושה: מחזירה את הגובה של הצומת.

 $O\left(1
ight)$ אחת לשדה לכן הריצה: ישנה גישה אחת לשדה לכן

public void setSize(int size) הפעולה 1.3.14

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

 $O\left(1
ight)$ אחת לשדה לכן הריצה: ישנה ישנה לשדה לכן

public int getSize() הפעולה 1.3.15

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

 $O\left(1\right)$ אחת לשדה לכן הריצה: ישנה גישה אחת לשדה לכן

public void setSubTreeXor(boolean xor) הפעולה 1.3.16

x ביות הנוכחי היות הצומת הוא שורשו בתת-העץ של כלל הצמתים על המידע על את מעולת מנוכחי הנוכחי מה היא עושה: קובעת את תוצאת פעולת אחדע של המידע אל המידע המידע של המידע אונה היא אונה היא אונה היא אונה המידע של המידע המידע המידע של המידע של המידע המידע המידע של המידע המידע של המידע המידע המידע של המידע המיד

 $O\left(1\right)$ אחת לשדה לכן פיבוכיות אמן הריצה: ישנה גישה אחת לשדה לכן

public boolean getSubTreeXor() הפעולה 1.3.17

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

 $O\left(1
ight)$ סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public void setSuccessor(AVLNode successor) הפעולה 1.3.18

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

 $O\left(1\right)$ אחת לשדה לכן הריצה: ישנה גישה אחת לשדה לכן

public AVLNode getSuccessor() הפעולה 1.3.19

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

 $O\left(1\right)$ שיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public void setPredecessor(AVLNode predecessor) הפעולה 1.3.20

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

 $O\left(1\right)$ שיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public AVLNode getPredecessor() הפעולה 1.3.21

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

 $O\left(1
ight)$ סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public int getBalanceFactor() הפעולה 1.3.22

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.

אשר רצות getHeight(), isRealNode(), getLeft(), getRight() אשר הפעולות וינה קריאה למספר קבוע של הפעולות ישנה קריאה למספר קבוע של הפעולות $O\left(1\right)$.

public void updateNodeFields() הפעולה 1.3.23

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 על שדות xor מעודכן להיות תוצאת פעולה

AVLNode של המחלקה "set" ו-"get" ו-"get" של השמות, שימוש במספר קבוע של המחלקה מחלקה המחלקה הריצה: מתבצע מספר קבוע של פעולות אריתמטיות ופעולות או מתבצעים ב- $O\left(1\right)$ זמן, ולכן סיבוכיות זמן הריצה הכוללת של הפעולה היא $O\left(1\right)$.

public int getChildCount() הפעולה 1.3.24

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.

.O(1)ב ב-(ולם ב-isRealNode() קפtLeft(), getRight(), inwail בפעולות (שימוש בפעולות (isRealNode() כולם ב-(וע מספר קבוע של השמות, ושימוש בפעולות (O(1).

public boolean isLeftChild() הפעולה 1.3.25

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.

אשר רצים ב- $O\left(1\right)$. לכן סיבוכיות זמן הריצה הכוללת של getParent() ו- $O\left(1\right)$ אשר רצים ב-פעולות זמן הריצה הכוללת של הפעולה היא $O\left(1\right)$.

AVLTree המחלקה

Direction טיפוס המנייה 2.:

ערכי טיפוס המנייה 2.1.1

- .Right הכיוון.
- .Left הכיוון.2

public Direction reverseDir() הפעולה 2.1.2

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

 $O\left(1
ight)$ איבוכיות זמן הריצה: הפעולה זו רצה בסיבוכיות

2.2 שדות המחלקה:

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

2.3 בנאי המחלקה:

public AVLTree() הבנאי הריק 2.3.1

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

2.4 פעולות המחלקה:

public AVLNode getVirtualNode() הפעולה 2.4.1

מה היא עושה: מחזירה את הצומת הוירטואלי של העץ.

 $O\left(1\right)$ אחת לשדה לכן הריצה: ישנה גישה אחת לשדה לכן

public AVLNode getRoot() הפעולה 2.4.2

מה היא עושה: מחזירה את השורש של העץ.

 $O\left(1
ight)$ אחת לשדה לכן הריצה: ישנה גישה אחת לשדה לכן

public void setRoot(AVLNode root) הפעולה 2.4.3

מה היא עושה: קובעת את שורש העץ להיות root.

 $O\left(1\right)$ אחת לשדה לכן פיבוכיות אמן הריצה: ישנה גישה אחת לשדה לכן

public boolean empty() הפעולה 2.4.4

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

סיבוכיות אחת, לכן סיבוכיות ומן הריצה היא getRoot() שרצה ב- $O\left(1\right)$ וכן פעולת השוואה אחת, לכן סיבוכיות ומן הריצה היא $O\left(1\right)$.

public int size() הפעולה 2.4.5

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) שורש

O(1) שרצות אמן הריצה: ישנה קריאה לפעולות (getRoot() ו-getRoot() שרצות ב-O(1), לכן סיבוכיות אמן הריצה היא

public AVLNode getMin() הפעולה 2.4.6

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.

 $O\left(1
ight)$ סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public void setMin(AVLNode min) הפעולה 2.4.7

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

 $O\left(1\right)$ סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public AVLNode getMax() הפעולה 2.4.8

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.

 $O\left(1
ight)$ סיבוכיות זמן הריצה: ישנה גישה אחת לשדה לכן

public void setMax(AVLNode max) הפעולה 2.4.9

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

 $O\left(1
ight)$ אחת לשדה לכן ישנה ישנה ישנה לכן לכן סיבוכיות אמן הריצה:

public AVLNode searchNode(int k) הפעולה 2.4.10

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 אמן הריצה: חיפוש בעץ חיפוש בינארי עובר במסלול מהשורש אל עלה בעץ, מסלול שאורכו לכל היותר h+1 כאשר אותר $O(\log n)$ מתקיים אותר בעץ הוא הוא גובה העץ, וכן בעץ AVL מתקיים הואר הוא גובה העץ, וכן בעץ איטרציות מספר פעולות שכפי שראינו סיבוכיות אמן הריצה שלהן הוא O(1) (כמו גם הסיבוכיות של petRoot שמתבצעת לפני הלולאה). לכן סיבוכיות אמן הריצה הכוללת היא $O(\log n)$

public Boolean search(int k) הפעולה 2.4.11

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

.searchNode(int k) אחרת, משתמשת בצומת החזירה חולה מחזירה הפעולה מחזירה החולר אחרת, משתמשת בצומת החזר מפתח אילו ערך המפתח של node הוא node הוא node אילו ערך המפתח של node הוא node אומת עם מפתח חולר של מוחזר את המידע של הצומת. אחרת, המשמעות היא שלא נמצא בעץ צומת עם מפתח node מעוחד ווער

שיבוכיות אמן, כלל הפעולות מתבצעות אפarchNode(int k) שיבוכיות מחקריאה היחידה לפעולה מתבצעות מתבצעות מתבצעות מחקריאה היחידה לפעולה מתבצעות מתבצעות ב- $O\left(\log n\right)$ מון, כלל הפעולות מתבצעות ב- $O\left(1\log n\right)$ מון, כלל הפעולות של הפעולה היא

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

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:

- :חעץ ריק. 1
- יוצרת צומת חדש ומגדירה אותו בתור השורש, האיבר המינימלי והמקסימלי.
 - : העץ לא ריק.

משתמשת ב (searchNode(int k) כדי למצוא את הצומת שאמור להיות אביו של הצומת החדש, ומכניסה את הצומת החדש updateRela- או updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode) או updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode) ולאחר מכן נאזן את העץ בעזרת tionsForNewRightChild(AVLNode parent, AVLNode newNode) פעם אחת עך ידי קריאה יחידה (אן שתיים במידה ונדרשה פעולת איזון).

setMax(AVLNode ,setMin(AVLNode min) ,setRoot(AVLNode root) ,empty() סיבוכיות זמן הריצה: ב"מקרה הטוב" הראינו כי (O(1) ב"מקרה הגרוע" ישנן מספר קריאות למתודות אחרות בקוד - קריאה למקרה הגרוע" ישנן מספר קריאות לכן סה"כ (O(1) לכן סה"כ (O(1) ב"מקרה הגרוע" ישנן מספר קריאות למתודות אחרות בקוד - קריאה לאשר מתבצעת ב- $O(\log n)$ זמן וקריאה ל($O(\log n)$ זמן ושתי הפעולות האחרות מתבצעות ב-O(1). לכן מסצוכיות הזמן הכוללת של הפעולה היא $O(\log n)$

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

מה היא עושה: מגדירה את newNode בתור בנו השמאלי של node תוך התחשבות במקרי הקצה ועדכון המצביעים הרלוונטים. newNode בתור בנו השמאלי של updateSucces- ,setMin(AVLNode min) ,getMin() ,setLeft(AVLNode node) סיבוכיות זמן הריצה: הראינו כי הפעולות היא $O\left(1\right)$ מתבצעות בסיבוכיות זמן הריצה הכוללת היא $O\left(1\right)$ מתבצעות בסיבוכיות זמן הריצה הכוללת היא $O\left(1\right)$

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

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

updateSucces- ,setMax(AVLNode min) ,getMax() ,setRight(AVLNode node) **סיבוכיות זמן הריצה:** הראינו כי הפעולות sor(AVLNode node, AVLNode newNode) מתבצעות בסיבוכיות זמן הריצה הכוללת היא $O\left(1\right)$

private AVLNode balanceTreeOnce(AVLNode node) הפעולה 2.4.15

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 $O\left(1\right)$ לכןחלק זה הוא getMin() ,setLeft(AVLNode node), הראינו כי הפעולות בעוסף, הראינו מצריך איזון. בנוסף, הראינו כי הפעולות מהצומת הנתון אל הצומת שצריך איזון. בנוסף, הראינו כי הפעולות getMin() ,setLeft(AVLNode node) ,setMin(AVLNode min) מתבצעות בסיבוכיות זמן $O\left(1\right)$, לכן סיבוכיות זמן הריצה הכוללת היא $O\left(1\right)$.

private void balanceNode(AVLNode node) הפעולה 2.4.16

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

סיבוכיות זמן הריצה: הפונקציה קוראת לאחת מארבע פעולות האיזון

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

ומבצעת שאילתות

- getBalanceFactor() השאילתה
 - getLeft() השאילתה

O(1) לכן סך הפעולות הוא O(1)

private boolean isUnbalanced(AVLNode node) הפעולה 2.4.17

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

 $O\left(1
ight)$ ב- $O\left(1
ight)$ לכן סך הפעולות הוא getBalanceFactor() אילתת מבצעת שאילתת מבצעת שאילתת

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

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.

 $O\left(1
ight)$ שיבוכיות הרצות הריצה: ישנן קריאות לשאילותות ומתודות הרצות ישנן קריאות סיבוכיות

: השאילתות

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

הפעולות

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

private void rotateRight(AVLNode node) הפעולה 2.4.19

מה היא עושה: מבצעת סיבוב יחיד ימינה מהצומת הנתון.

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

private void rotateLeft(AVLNode node) הפעולה 2.4.20

מה היא עושה: מבצעת סיבוב יחיד שמאלה מהצומת הנתון.

סיבוכיות אמן הריצה: הראינו כי הפעולה rotateInDir(AVLNode node, Direction dir) מתבצעת בסיבוכיות מן הריצה: הראינו כי הפעולה $O\left(1\right)$. סיבוכיות אמן הריצה הכוללת היא $O\left(1\right)$.

private void rotateLeftThenRight(AVLNode node) הפעולה 2.4.21

מה היא עושה: מבצעת סיבוב שמאלה וימינה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות

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

O(1) איא הכוללת היצה ומן הריצה לכן סיבוכיות אמן אמן O(1), לכן סיבוכיות מתבצעות בסיבוכיות היא

private void rotateRightThenLeft(AVLNode node) הפעולה 2.4.22

מה היא עושה: מבצעת סיבוב ימינה ושמאלה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות

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

 $O\left(1\right)$ מתבצעות בסיבוכיות זמן $O\left(1\right)$, לכן סיבוכיות זמן הריצה מתבצעות און מון

public int delete(int k) הפעולה 2.4.23

מה היא עושה: מוחקת את הצומת בעל המפתח הנתון מהעץ אם הוא קיים ומחזירה את כמות פעולות האיזון שנדרשו לאיזון העץ, או - 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 ביצד היא פועלת: updateSuccessor(AVLNode node, AVLNode newNode) ולאחר מכן נוקטת באחד מששת המקרים המתאימים:

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

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

: האיפוס יתבצע באמצעות הפעולות

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

 $O\left(1\right)$ אשר כל אחת פועלת ב- $O\left(1\right)$ בסיבוכיות כוללת

.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

- getSuccessor() השאילתה
 - getRight() השאילתה •

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

:חפעולות והפעולות בסיבוכיות אשר כולן בסיבוכיות

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

 $O\left(1
ight)$ אשר גם הן בסיבוכיות $O\left(1
ight)$ ולכן הסיבוכיות אשר גם הן בסיבוכיות

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.

ואת באמצעות השאילתות:

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

אשר כולן בסיבוכיות $O\left(1\right)$ והפעולות:

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

 $O\left(1\right)$ אשר גם הן בסיבוכיות ולכן חלכן ולכן $O\left(1\right)$ היא

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

(א) לצומת בן יחיד:

נבצע מעקף עם בן זה.

נשתמש בשאילתות

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

והפעולות:

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

(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

- getPredecessor() השאילתה
 - getParent() השאילתה

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

והפעולות:

- 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) הפעולה

 $O\left(1
ight)$ ראינו כי כלל הפעולות והשאילתות מתבצעות בסיבוכיות

(ג) הצומת הוא עלה בעץ:

נבצע מעקף עם הבן שלו שהוא צומת וירטואלי.

: זאת באמצעות השאילתות

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

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

.balanceNode(AVLNode node) ובסיום רצה הפעולה

 $O\left(\log n\right)$ הרצה בסיבוכיות balanceNode(AVLNode node) למעט למעט בסיבוכיות מתבצעות בסיבוכיות השאילתות מתבצעות בסיבוכיות $O\left(\log n\right)$ לכן הפעולה רצה בסיבוכיות ל $O\left(\log n\right)$.

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

.newParent להיות ילדי oldParent מה מעבירה את ילדי

oldParent על מנת להגדיר את ילדי setLeft(AVLNode node) ,setRight(AVLNode node) על מנת להגדיר את ילדי ופיצד היא פועלת: קוראת לפעולות (setParent על מנת להגיד את newParent ואז משתמשת ב(aude) היות ילדי newParent ואז משתמשת ב(oldParent).

סיבוכיות זמן הריצה: הראינו כי הפעולות

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

 $O\left(1
ight)$ מתבצעות בסיבוכיות זמן $O\left(1
ight)$, לכן סיבוכיות זמן הריצה הכוללת היא

private Direction getDirectionFromParent(AVLNode node) הפעולה 2.4.25

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

. ביצד היא פועלת: קוראת ל(isLeftChild() ולפי ערך ההחזרה שלו מחזירה את הכיוון המתאים

סיבוכיות אמן הריצה ווא הריצה (1) מתבצעת מתבצעת ווא isLeftChild() מתבצעת הריצה: הראינו כי הפעולה ($O\left(1\right)$ מתבצעת בסיבוכיות הריצה: הראינו כי הפעולה ($O\left(1\right)$

private int balanceTree(AVLNode node) הפעולה 2.4.26

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 אורך המסלול בין הצומת הצומת של 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 הנתון אל הצומת המחזר. אנו קוראים בלולאה שוב ושוב לפעולה עם הערך המסלול מהצמת הנתון אל השורש שהיא חסומה בגובה העץ הלא הוא $O(\log n)$, לכן סיבוכיות זמן הריצה הכוללת היא $O(\log n)$.

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

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:

- :חעץ ריק. 1
- יוצרת צומת חדש ומגדירה אותו בתור השורש, האיבר המינימלי והמקסימלי.
 - : העץ לא ריק. 2

משתמשת ב (searchNode(int k) כדי למצוא את הצומת שאמור להיות אביו של הצומת החדש, ומכניסה את הצומת החדש updateRela- או updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode) או updateRelationsForNewLeftChild(AVLNode parent, AVLNode newNode) ולאחר מכן נאזן את העץ בעזרת tionsForNewRightChild(AVLNode parent, AVLNode newNode) פעם אחת עך ידי קריאה יחידה (אן שתיים במידה ונדרשה פעולת איזון).

setMax(AVLNode ,setMin(AVLNode min) ,setRoot(AVLNode root) ,empty() סיבוכיות זמן הריצה: ב"מקרה הטוב" הראינו כי (O(1) ב"מקרה הגרוע" ישנן מספר קריאות למתודות אחרות בקוד - קריאה למקרה הגרוע" ישנן מספר קריאות לכן O(1) לכן סה"כ (O(1) ב"מקרה הגרוע" ישנן מספר קריאות למתודות אחרות בקוד - קריאה לאשר מעם או פעמיים כך שסה"כ עוברים על העץ אשר מתבצעת ב- $O(\log n)$ זמן וקריאה ל($O(\log n)$ זמן ושתי הפעולות האחרות מתבצעות ב-O(1). לכן מסיבוכיות הזמן הכוללת של הפעולה היא $O(\log n)$.

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

מה היא עושה: מגדירה את newNode בתור בנו השמאלי של node תוך התחשבות במקרי הקצה ועדכון המצביעים הרלוונטים. newNode בתור בנו השמאלי של updateSucces- ,setMin(AVLNode min) ,getMin() ,setLeft(AVLNode node) סיבוכיות זמן הריצה: הראינו כי הפעולות היא $O\left(1\right)$ מתבצעות בסיבוכיות זמן הריצה הכוללת היא $O\left(1\right)$ מתבצעות בסיבוכיות זמן הריצה הכוללת היא $O\left(1\right)$

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

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

updateSucces- ,setMax(AVLNode min) ,getMax() ,setRight(AVLNode node) **סיבוכיות זמן הריצה:** הראינו כי הפעולות sor(AVLNode node, AVLNode newNode) מתבצעות בסיבוכיות זמן הריצה הכוללת היא $O\left(1\right)$

private AVLNode balanceTreeOnce(AVLNode node) הפעולה 2.4.30

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 $O\left(1\right)$ לכןחלק זה הוא getMin() ,setLeft(AVLNode node), הראינו כי הפעולות בעוסף, הראינו מצריך איזון. בנוסף, הראינו כי הפעולות מהצומת הנתון אל הצומת שצריך איזון. בנוסף, הראינו כי הפעולות getMin() ,setLeft(AVLNode node) ,setMin(AVLNode min) מתבצעות בסיבוכיות זמן $O\left(1\right)$, לכן סיבוכיות זמן הריצה הכוללת היא $O\left(1\right)$.

private void balanceNode(AVLNode node) הפעולה 2.4.31

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

סיבוכיות זמן הריצה: הפונקציה קוראת לאחת מארבע פעולות האיזון

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

ומבצעת שאילתות

- getBalanceFactor() השאילתה
 - getLeft() השאילתה •

 $O\left(1
ight)$ לכן סך הפעולות הוא $O\left(1
ight)$

private boolean isUnbalanced(AVLNode node) הפעולה 2.4.32

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

 $O\left(1
ight)$ ב- $O\left(1
ight)$ לכן סך הפעולות הוא getBalanceFactor() אילתת מבצעת שאילתת מבצעת שאילתת

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

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.

 $O\left(1
ight)$ שיבוכיות הרצות הריצה: ישנן קריאות לשאילותות ומתודות הרצות ישנן קריאות סיבוכיות

: השאילתות

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

הפעולות

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

private void rotateRight(AVLNode node) הפעולה 2.4.34

מה היא עושה: מבצעת סיבוב יחיד ימינה מהצומת הנתון.

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

private void rotateLeft(AVLNode node) הפעולה 2.4.35

מה היא עושה: מבצעת סיבוב יחיד שמאלה מהצומת הנתון.

סיבוכיות אמן הריצה: הראינו כי הפעולה rotateInDir(AVLNode node, Direction dir) מתבצעת בסיבוכיות מן הריצה: הראינו כי הפעולה $O\left(1\right)$. סיבוכיות אמן הריצה הכוללת היא $O\left(1\right)$.

private void rotateLeftThenRight(AVLNode node) הפעולה 2.4.36

מה היא עושה: מבצעת סיבוב שמאלה וימינה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות

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

O(1) איא הכוללת היצה ומן הריצה לכן סיבוכיות אמן אמן O(1), לכן סיבוכיות מתבצעות בסיבוכיות היא

private void rotateRightThenLeft(AVLNode node) הפעולה 2.4.37

מה היא עושה: מבצעת סיבוב ימינה ושמאלה מהצומת הנתון.

סיבוכיות זמן הריצה: הראינו כי הפעולות

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

 $O\left(1\right)$ מתבצעות בסיבוכיות זמן $O\left(1\right)$, לכן סיבוכיות זמן הריצה מתבצעות און מון

public int delete(int k) הפעולה 2.4.38

מה היא עושה: מוחקת את הצומת בעל המפתח הנתון מהעץ אם הוא קיים ומחזירה את כמות פעולות האיזון שנדרשו לאיזון העץ, או - 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 **כיצד היא פועלת:** updateSuccessor(AVLNode node, AVLNode newNode) ולאחר מכן נוקטת באחד מששת המקרים המתאימים:

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

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

: האיפוס יתבצע באמצעות הפעולות

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

 $O\left(1\right)$ אשר כל אחת פועלת ב- $O\left(1\right)$ בסיבוכיות כוללת

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

- getSuccessor() השאילתה
 - getRight() השאילתה •

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

אשר כולן בסיבוכיות $O\left(1\right)$ והפעולות:

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

 $O\left(1
ight)$ אשר גם הן בסיבוכיות $O\left(1
ight)$ ולכן הסיבוכיות אשר גם הן בסיבוכיות

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.

ואת באמצעות השאילתות:

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

\cdot אשר כולן בסיבוכיות $O\left(1 ight)$ והפעולות

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

 $O\left(1
ight)$ אשר גם הן בסיבוכיות $O\left(1
ight)$ ולכן הסיבוכיות הכוללת היא

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

(א) לצומת בן יחיד:

נבצע מעקף עם בן זה.

נשתמש בשאילתות

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

והפעולות:

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

(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

- getPredecessor() השאילתה
 - getParent() השאילתה

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

והפעולות:

- 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) הפעולה

 $O\left(1
ight)$ ראינו כי כלל הפעולות והשאילתות מתבצעות בסיבוכיות

(ג) הצומת הוא עלה בעץ:

נבצע מעקף עם הבן שלו שהוא צומת וירטואלי.

: זאת באמצעות השאילתות

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

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

.balanceNode(AVLNode node) ובסיום רצה הפעולה

 $O\left(\log n\right)$ הרצה בסיבוכיות balanceNode(AVLNode node) למעט למעט בסיבוכיות מתבצעות בסיבוכיות הפעולות והשאילתות מתבצעות בסיבוכיות $O\left(\log n\right)$ לכן הפעולה רצה בסיבוכיות לסות הפעולה רצה בסיבוכיות לסות הפעולה רצה בסיבוכיות וחסיבוכיות לסות הפעולה רצה בסיבוכיות וחסיבו

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

.newParent להיות ילדי oldParent מה מעבירה את ילדי

oldParent על מנת להגדיר את ילדי setLeft(AVLNode node) ,setRight(AVLNode node) על מנת להגדיר את ילדי ופיצד היא פועלת: קוראת לפעולות (setParent על מנת להגיד את newParent ואז משתמשת ב(aude) היות ילדי newParent ואז משתמשת ב(oldParent).

סיבוכיות זמן הריצה: הראינו כי הפעולות

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

 $O\left(1
ight)$ מתבצעות בסיבוכיות זמן $O\left(1
ight)$, לכן סיבוכיות זמן הריצה הכוללת היא

private Direction getDirectionFromParent(AVLNode node) הפעולה 2.4.40

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

מיצד היא פועלת: קוראת ל(isLeftChild() ולפי ערך ההחזרה שלו מחזירה את הכיוון המתאים.

היא הכוללת זמן הריצה ומן (1) מתבצעת מתבצעת (1) מתבצעת הריצה isLeftChild() מתבצעת כי הפעולה הריצה: הראינו כי הפעולה $O\left(1\right)$

private int balanceTree(AVLNode node) הפעולה 2.4.41

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 אורך המסלול בין הצומת שלוביות המותכבר balanceTreeOnce(AVLNode node) מתבצעת בסיבוכיות אורך המסלול בין 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 $O(\log n)$.

public Boolean min() הפעולה 2.4.42

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

סיבוכיות (1) מתבצעות פיבוכיות (empty(), getMin() פיבוכיות (1), לכן סיבוכיות (1), לכן סיבוכיות (1) איבוכיות (1), לכן סיבוכיות (1), $O\left(1\right)$ הריצה הכוללת היא (1).

public Boolean max() הפעולה 2.4.43

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

סיבוכיות אמן (1) מתבצעות בסיבוכיות (empty(), getMax() פיבוכיות אמן (1), לכן סיבוכיות אמן $O\left(1\right)$, לכן סיבוכיות הריצה הריצה הכוללת היא (1).

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

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

בעצ 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 $of\,fset$ וכמות הצמתים בתת-העץ שבנו השמאלי של $of\,fset$ הוא שורשו, על מנת שבמקומות שבין $of\,fset$ לאינדקס זה יוכנסו למערך כל הצמתים בתת-העץ שבנו השמאלי של node הוא שורשו). לאחר ההכנסה למערך, מתבצעת של node הוא שורשו). לאחר ההכנסה למערך, מתבצעת קריאה רקורסיבית לבן הימני של node עם $of\,fset$ שהוא האינדקס העוקב של האינדקס אליו node הוא מייצג "סיום מסלול" מקרה הבסיס של הפעולה הוא כאשר node הוא צומת שאינו אמיתי, כלומר צומת וירטואלי (שכן הוא מייצג "סיום מסלול").

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

public AVLNode[] nodesToArray() הפעולה 2.4.45

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

inOrder(AVLNode node, פיצד היא פועלת: ועם שורש איתו איתו לא ריק קוראת איץ, ואם הוא לא ריק קוראת מערך על מווער מערך יוצרת מערך (ואר הצמתים של העץ ממוינים על פי arr אשר כפי שהראינו מכניסה למערך, int offset, AVLNode[] arr) המפתחות.

inOrder(AVLNode node, int offset, AVLN- שיבוכיות אמן הריצה: הפעולות (empty() פיבוכיות מון הריצה: הפעולות (empty() וביות מון הריצה: הפעולות סיבוכיות מון הריצה: הפעולות (arr) מתבצעת ב-O(n) זמן. לכן (arr) מתבצעות ב-O(n) זמן, ואילו (O(n) זמן. לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא O(n)

public int[] keysToArray() הפעולה 2.4.46

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

ת שלהם, מערך של הצמתים בעץ ממוינים על פי המפתחות שלהם, nodesToArray() ליצד היא פועלת: משתמשת בפעולה מכניסה את המפתחות בסדר הממוין למערך arr.

סיבועות מתבצעות הריצה: הפעולות () ומן, ואילו (O(1) מתבצעות ב-size() מתבצעות הריצה: הפעולות () מון הריצה הפעולות מון הריצה הכוללת של הפעולה היא O(n) מון. לכן סיבוכיות זמן הריצה הכוללת של הפעולה היא

public boolean[] infoToArray() הפעולה 2.4.47

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

פלצד 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.

סיבוניות אמן הריצה: הפעולות () פחבצעות ב- $O\left(1\right)$ אמן, ואילו () פחבצעות הריצה: הפעולות () פחבצעות הריצה מתבצעות ב- $O\left(n\right)$ און, ואילו () היא $O\left(n\right)$ און, לכן סיבוכיות אמן הריצה הכוללת של הפעולה היא $O\left(n\right)$

public boolean prefixXor(int k) הפעולה 2.4.48

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

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

קפtSubTreeXor: פרגמונית (פtRoot(), getKey(), getLeft(), getRight(), getValue() פיבוכיות אמן הריצה: הראינו כי הפעולות (פעולות פיבוביות מפעולות המתבצעות ב-O(1) אמן. לפי ההסבר של הפעולה, הלולאה מבצעת לכל היותר h+1 כאשר h+1 הוא גובה העץ (שכן אור פעולות המתבצעות ב- $O(\log n)$ אמן. לפי ההסבר של ביותר בין שורש לצומת בעץ), וכן בעץ AVL מתקיים $O(\log n)$ לכן סיבוכיות אמן הריצה הכוללת היא $O(\log n)$.

public AVLNode successor(AVLNode node) הפעולה 2.4.49

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

סיבוכיות אמן הריצה ומן, לכן סיבוכיות מעולה ($O\left(1\right)$ היא פעולה המתבצעת פetSuccessor) מעולה הפעולה הריצה הראינו כי הפעולה ($O\left(1\right)$ היא פעולה המתבצעת ב- $O\left(1\right)$

public boolean succPrefixXor(int k) הפעולה 2.4.50

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

מת successor(AVLNode node) ביצד היא פועלת: מתחילה מהצומת המינימלי של העץ, ומבצעת עליו פעולות אומר k מתחילה מהצומת שהתקבל בפעולות ה-(successor(AVLNode node) (וכן הצומת המינימלי עצמו), הפעולה מחשבת עם ערכו הבוליאני, ולבסוף מחזירה את התוצאה.

סיבוכיות אמן הריצה: הראינו כי הפעולות (getMin(), getKey() הן פעולות המתבצעות ב-O(1) אמן. הפעולה מספר איטרציה מבצעת פעולות מספר הצמתים בעץ עם מפתחות שקטנים או שווים למפתח A, ובכל איטרציה מבצעת פעולת איטרציה מפצעת פעולת בעץ במספר הצמתים בעץ מפעולת בעץ איטרציות (כמספר סך כל הצמתים בעץ), לכן מפעולת (O(1)- מון. כלומר מתבצעות לכל היותר A איטרציות (כמספר סך כל הצמתים בעץ), לכן סיבוכיות אמן הריצה הכוללת היא