תרגיל מעשי 1 - קובץ תיעוד ובדיקות

מגישים:

גיא לוי, 318645694

ליעד אילוז, 208427435

תיעוד המחלקה והפונקציות

להלן חתימות כל הפונקציות שממומשות תחת המחלקה AVLTree, בחלוקה לפי סוג הפונקציה – אם נדרשנו לממש אותה במסגרת התרגיל או אם זו פונקציית עזר שהוספנו בעצמנו.

לא יופיע תיעוד על אופן הפעולה של פונקציות getter או setter שכל שהן עושות הוא לשלוף שדה או לעדכן שדה של אובייקט בערך ידוע מראש.

מופיע גם תיעוד קצר עבור השדות של שתי תתי המחלקות.

שדות של AVLTree

מצביע לשורש העץ	`private IAVLNode root;`
מצביע לצומת עם המפתח המינימלי בעץ	`private IAVLNode minimum;`
מצביע לצומת עם המפתח המקסימלי בעץ	`private IAVLNode maximum;`
מספר הצמתים העץ	`private int size;`

שדות של AVLNode

מחרוזת המייצגת את תוכן הצומת	`private String info;`
מפתח של צומת	`private int key;`
מצביע לצומת הבן הימני של הצומת	`private IAVLNode rightSon;`
מצביע לצומת הבן השמאלי של הצומת	`private IAVLNode leftSon;`
מצביע לצומת ההורה של הצומת	`private IAVLNode parent;`
גובה הצומת בעץ	`private int height;`
ערך בוליאני אם הצומת אמיתית	`private boolean isReal;`
מספר הצמתים בתת העץ של הצומת	`private int sizeNode;`

פונקציות שנדרשנו לממש בשלד התרגיל:

שם תת המחלקה	חתימת הפונקציה	סיבוכיות הפונקציה
AVLTree	public boolean empty()	$O(1)$
AVLTree	public String search(int k)	$O(\log n)$
AVLTree	public int insert(int k, String s)	$O(\log n)$
AVLTree	public int delete(int k)	$O(\log n)$
AVLTree	public String min()	$O(1)$
AVLTree	public String max()	$O(1)$
AVLTree	public int[] keysToArray()	$O(n)$
AVLTree	public String[] infoToArray()	$O(n)$
AVLTree	public int size()	$O(1)$
AVLTree	public IAVLNode getRoot()	$O(1)$
AVLTree	public AVLTree[] split(int x)	$O(\log n)$
AVLTree	public int join(IAVLNode x, AVLTree t)	$O(\|t1.rank - t2.rank\| + 1)$

פונקציות getter ו setter נוספות שנתבקשנו לממש עבור הממשק IAVLNode

שם תת המחלקה	חתימת הפונקציה	סיבוכיות הפונקציה
AVLNode	public int getSizeNode()	$O(1)$
AVLNode	public void setSizeNode(int s)	$O(1)$
AVLNode	public int getKey()	$O(1)$
AVLNode	public String getValue()	$O(1)$
AVLNode	public void setLeft(IAVLNode node)	$O(1)$
AVLNode	public IAVLNode getLeft()	$O(1)$
AVLNode	public void setRight(IAVLNode node)	$O(1)$
AVLNode	public IAVLNode getRight()	$O(1)$
AVLNode	public void setParent(IAVLNode node)	$O(1)$
AVLNode	public IAVLNode getParent()	$O(1)$
AVLNode	public boolean isRealNode()	$O(1)$
AVLNode	public void setHeight(int height)	$O(1)$
AVLNode	public int getHeight()	$O(1)$

פונקציות עזר שהוספנו בעצמנו:

סיבוכיות הפונקציה	חתימת הפונקציה	שם תת המחלקה
$O(\log n)$	private static void updateRoot(AVLTree tree)	AVLTree
$O(1)$	public int getSize()	AVLTree
$O(1)$	public IAVLNode getMaximum()	AVLTree
$O(1)$	public IAVLNode getMinimum()	AVLTree
$O(\log n)$	public IAVLNode treePosition(int key, IAVLNode curr)	AVLTree
$O(\log n)$	private IAVLNode getSuccessor(IAVLNode node)	AVLTree
$O(\log n)$	private IAVLNode getPredecessor(IAVLNode node)	AVLTree
$O(\log n)$	private IAVLNode deleteNode(IAVLNode node, String typeNode, String dirNode)	AVLTree
$O(root.height - node.height + 1)$	private static void updateSizeNodesFromNode(IAVLNode node)	AVLTree
$O(1)$	private static void updateSizeNode(IAVLNode node)	AVLTree
$O(\log n)$	private void updateMax()	AVLTree
$O(\log n)$	private void updateMin()	AVLTree
$O(\log n)$	public IAVLNode findMinNode(IAVLNode node)	AVLTree
$O(\log n)$	public IAVLNode findMaxNode(IAVLNode node)	AVLTree
$O(1)$	private String getKindOfNode(IAVLNode node)	AVLTree
$O(1)$	private boolean isUnary(IAVLNode node)	AVLTree
$O(1)$	private boolean isLeaf(IAVLNode node)	AVLTree
$O(1)$	private void rotateRight(IAVLNode currSon)	AVLTree
$O(1)$	private void rotateLeft(IAVLNode currSon)	AVLTree
$O(1)$	private void demote(IAVLNode node)	AVLTree
$O(1)$	private void promote(IAVLNode node)	AVLTree
$O(\log n)$	private int balanceAfterInsertion(IAVLNode newParent)	AVLTree
$O(\log n)$	private int balanceAfterDeletion(IAVLNode parentOfDeleted)	AVLTree
$O(1)$	private boolean isLegalAVLNode(IAVLNode node)	AVLTree

תיעוד פונקציות השלד תחת AVLTree

<u>הפונקציה ()empty:</u>

תכלית:

הפונקציה מחזירה true אם העץ ריק ואחרת false.

אופן פעולה:

הפונקציה מחזירה true אם שדה ה size של העץ שווה לאפס ואחרת מחזירה false.

סיבוכיות: $O(1)$.

<u>הפונקציה search(int k) :</u>

תכלית:

הפונקציה מחזירה את שדה ה-info של הצומת בעל מפתח k, או null אם אין כזה צומת בעץ.

אופן פעולה:

הפונקציה קוראת לפונקציית העזר treePosition על הערך k והשורש של העץ (סיבוכיות $O(\log n)$).

אם k קיים בעץ, treePosition תחזיר את הצומת המתאים והפונקציה תחזיר את המפתח שלו. אחרת, הפונקציה תחזיר null.

סיבוכיות: $O(\log n)$

פרט לקריאה לפונקציה treePosition (סיבוכיות $O(\log n)$) הפונקציה מבצעת מספר סופי של פעולות שלוקחות זמן קבוע.

הפונקציה insert(int k, String s) :

תכלית:

הפונקציה מכניסה צומת עם מפתח k ומידע s לעץ ודואגת להחזיר את העץ לאיזון. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו. במידה וצומת בעל מפתח k כבר קיים בעץ, הפונקציה תחזיר 1-.

אופן פעולה:

הפונקציה קוראת לפונקציה treePosition יחד עם הערך k והשורש של העץ.

אם קיבלנו null, משמעות הדבר שהעץ ריק ולכן נייצר צומת מתאים חדש ונחזיר 0 שכן לא בוצעו פעולות איזון. אחרת, אם מצאנו צומת עם מפתח k נחזיר 1-. אחרת, קיבלנו את ההורה של הצומת שאותו אנו מבקשים להכניס. אם כך, ניצור צומת חדש לפי המשתנים שהפונקציה קיבלה ונכניס אותו למקום המתאים מתחת ההורה שלו. לאחר מכן, נקרא לפונקציה balanceAfterInsertion (סיבוכיות $O(\log n)$) על ההורה של הצומת החדש ונחזיר את מספר התיקונים שהיא ביצעה. לאחר מכן נעדכן את שדות ה-size של כל הצמתים במסלול שבין הצומת החדש שהוספנו אל השורש בעזרת קריאה לפונקציה updateSizeNodeFromNode(par) (סיבוכיות $O(\log n)$).

סיבוכיות: $O(\log n)$.

הפונקציה מבצעת מספר קבוע של פעולות שלוקחות זמן קבוע פרט לפעולות הבאות:

קריאה לפונקציה treePosition – סיבוכיות $O(\log n)$.

קריאה לפונקציה balanceAfterInsertion – סיבוכיות $O(\log n)$.

קריאה לפונקציה updateSizeNodeFromNode(par) (סיבוכיות $O(\log n)$).

לכן בסה"כ סיבוכיות הפונקציה היא $O(\log n)$.

<u>הפונקציה public int delete(int k):</u>

תכלית: הפוקנציה מוחקת את המפתח *k* אם קיים בעץ ומחזירה את מספר פעולות האיזון שנדרשו לאחר מחיקת הצומת, אם לא קיים תחזיר 1-.

אופן הפעולה:

תחילה הפונקציה מחפשת את הצומת עם המפתח *k*, ומשזה לא קיים היא תחזיר 1-.

כאשר הצומת המבוקש *node* בעל מפתח *k* קיים, הפנקציה מסווגת את סוג הצומת ע"י קריאה ל-getKindOfNode(node).

הפונקציה תמחק את הצומת המבוקש מהעץ ע"י פעולה deleteNode ותקבל את ההורה של הצומת שנמחקה במשתנה parentOfDeleted.

הפונקציה תיקרא כעת לפונקציה balanceAfterDeletion על ההורה parentOfDeleted שתאזן את העץ לפי המצבים שלאחר מחיקת איבר ותחזיר את מספר פעולות האיזון שנדרשו.

כעת הפונקציה תעדכן את השדות sizeNode מהבן הימני של parentOfDeleted כלפי מעלה בעץ וכן מהבן השמאלי של parentOfDeleted כלפי מעלה בעץ ע"י הפונקציה updateSizeNodesFromNode.

כמו כן הפונקציה תעדכן את המינימום והמקסימום של העץ ע"י קריאה לפונקציות updateMin ו-updateMin. לבסוף הפונקציה תוריד את מספר הצמתים בעץ ב-1. הפונקציה תחזיר את מספר פעולות האיזון כפי שהפועלה balanceAfterDeletion החזירה.

סיבוכיות: $O(\log n)$

הפונקציה מבצעת במהלך הריצה קריאות לפונקציות:

deleteNode, balanceAfterDeletion, updateSizeNodesFromNode, updateMin, updateMax

כל הפונקציות הללו בסיבוכיות של $O(\log n)$ במקרה הגרוע. שאר הפונקציות והפעולות הן קבועות.

לכן הסיבוכיות הכוללת של הפונקציה הנ"ל היא $O(\log n)$.

הפונקציה ()min:

תכלית: הפונקציה מחזירה את המידע של הצומת בעל המפתח המינימלי בעץ. אם העץ ריק, היא מחזירה null. סיבוכיות: $O(1)$.

הפונקציה ()max:

תכלית: הפונקציה מחזירה את המידע של הצומת בעל המפתח המקסימלי בעץ. אם העץ ריק, היא מחזירה null. סיבוכיות: $O(1)$.

הפונקציה ()keysToArray :

תכלית:

הפונקציה מחזירה מערך ממוין שמכיל את כל המפתחות בעץ או מערך ריק אם העץ ריק.

אופן פעולה:

הפונקציה מתחילה מהמינימום של העץ (שאליו יש לנו מצביע) ומבצעת קריאות successor כמספר הצמתים בעץ. לכל צומת היא מוסיפה את המפתח שלו למערך.

סיבוכיות: $O(n)$.

בכיתה ראינו שמעבר in-order על עץ בגודל n בעזרת n קריאות ל- successor מתבצע ב-$O(n)$.

הפונקציה ()infoToArray :

תכלית:

הפונקציה היא פונקצית מופע על אובייקט מסוג AVLTree המייצג עץ AVL. הפונקצייה מחזירה מערך מסוג String המכיל את כל הערכים של הצמתים בעץ בסדר עולה לפי המפתחות של הצמתים. אם העץ ריק הפונקציה תחזיר מערך ריק.

אופן פעולה:

הפונקציה ניגשת תחילה לאיבר המינימלי בעץ ישירות (בעל המפתח המינימלי) ע"י המצביע השמור לכך. הפונקציה מבצעת n פעמים קריאה לפעולה getSuccessor(node) שבכל קריאה היא מעבירה את הצומת הבאה החל מהמינימום ועד למקסימום בעץ. בכל פעם, מוכנס הערך (info) של הצומת למערך באינדקס הבא להכנסה.

סיבוכיות: $O(n)$

מלבד חישובים קבועים, העולה מבצעת n קריאות לפונקציה getSuccessor(node) על צומת. נעיר כי פעולת getSuccessor(node) לוקחת במקרה הגרוע $O(\log n)$, אולם כאשר ביצענו סדרה של n קריאות לפונקציה מהצומת המינימלי ובכל פעם דילגנו מאיבר לאיבר הבא ע"י הפועלה, נקבל למעשה שכל פעולה כזו תעלה בממוצע $O(1)$. ניתן לראות זאת ע"י כך שבמהלך כל סדרת n הפעולות, נבקר בכל קשת בגרף לכל היותר פעמיים – פעם כדי להגיע לצומת המיועדת, ופעם אחת כשנצא מהצומת המיועדת החוצה לעבר האיבר הבא. מכיוון שבעץ בעל n איברים יש n-1 קשתות נקבל שהעלות הכוללת של הפעולה שלנו היא $O(n)$.

הפונקציה ()size:

תכלית: הפונקציה מחזירה את גודל העץ.

אופן פעולה: הפונקציה מחזירה את שדה ה-size של העץ.

סיבוכיות: $O(1)$.

הפונקציה ()getRoot:

תכלית: הפונקציה מחזירה מצביע לשורש העץ.

אופן פעולה: הפונקציה מחזירה את שדה ה-root של העץ.

סיבוכיות: $O(1)$.

הפונקציה public AVLTree[] split(int x):

תכלית:

הפונקציה מקבלת ערך x המציין מפתח כלשהו בעץ, ומחזירה מערך של 2 עצי AVL תקינים המכילים את כל המפתחות הקטנים מ-x בעץ המקורי ואת כל המפתחות הגדולים מ-x בהתאמה.

אופן פעולה:

תחילה נחפש את הצומת בעץ עם המפתח x. כעת נאתחל 4 עצים: עץ עבור המפתחות הגודלים מ-x (יאותחל כתת העץ הימני של צומת x), עץ עבור המפתחות הקטנים מ-x (יאותחל כתת העץ השמאלי של צומת x), ועוד 2 עצי עזר לפעולות ה-join שנעשה אחד לעץ של הגדולים ואחד לעץ של הקטנים.

כעת נבצע את האלגוריתם שראינו בכיתה לפונקצית split: נעבור מהצומת x עד לשורש העץ בלולאה.

- בכל פעם שעלינו להורה מבן ימני – נבצע פעולת join של העץ של הקטנים עם תת העץ השמאלי של ההורה וצומת ההורה.
- בכל פעם שעלינו להורה מבן שמאלי – נבצע פעולת join של העץ של הגדולים עם תת העץ הימני של ההורה וצומת ההורה.

בכל פעם נאתחל את עצי העזר שלנו בתתי העצים שאנחנו רוצים לעשות איתם join.

לבסוף נבצע פעולות updateMin(), updateMax() לעץ של הגדולים ולעץ של הקטנים כדי לתחזק פרמטרים אלו בעצים שקיבלנו.

נחזיר מערך של שני עצים אלו.

סיבוכיות: $O(\log n)$.

ניתוח הסיבוכיות יהיה זהה לניתוח הסיבוכיות שעשינו בכיתה לפונקציה *split* כולל $O(\log n)$ עבור פעולות עדכון המינימום והמקסימום בסוף הפעולה. נקבל בסה"כ סיבוכיות של $O(\log n)$.

<u>הפונקציה (join(IAVLNode x, AVLTree t :</u>

תכלית:

הפונקציה מאחדת את העץ שעליו נקראה, יחד עם צומת x ועץ נוסף t. לאחר הקריאה לפונקציה על עץ מסוים, המצביע לעץ המקורי כעת יצביע לעץ המאוחד. הפונקציה מחזירה חסם על הסיבוכיות של ביצוע הפעולה.

אופן פעולה:

הפונקציה ראשית מתייחסת למצב שבו אחד מהעצים המקוריים ריק, שכן במצב כזה, לפי תנאי הקדם של הפונקציה, הצומת החדש x יכנס כמינימום או מקסימום בעץ שאינו ריק.

במקרה שבו שני העצים לא ריקים, הפונקציה מבצעת את אלגוריתם join כפי שראינו בכיתה וקוראת לפונקציה balanceAfterInsertion לאחר ההכנסה של הצומת x החדש.

בכל מקרה, לאחר איחוד שני העצים עם הצומת x, הפונקציה דואגת לעדכן את השדות של העץ המקורי שעליו נקראה הפונקציה כך שכעת המצביע של העץ המקורי יצביע לעץ החדש המאוחד.

לבסוף הפונקציה מחזירה חסם על סיבוכיות הפעולה, שכפי שראינו בכיתה, הוא הפרש הגבהים של שני העצים ועוד 1.

סיבוכיות: $O(|t1.rank - t2.rank| + 1)$.

כפי שראינו סיבוכיות הפעולה join על שני עצי t1,t2 AVL היא: $O(|t1.rank - t2.rank| + 1)$.

תיעוד פונקציות העזר תחת המחלקה AVLTree

<u>הפונקציה updateRoot(AVLTree tree) :</u>

תכלית:

הפונקציה מעדכנת את שדה השורש של העץ כך שיצביע לשורש האמיתי של העץ.

אופן פעולה:

הפונקציה מתחילה מבצעת לולאה החל מהצומת שמוגדר כרגע כ-root. בכל איטרציה הפונקציה קוראת להורה של הצומת הנוכחי. הפונקציה מעדכנת את שדה ה-root של העץ להיות הצומת העליון ביותר (האחרון לפני הורה שהוא null).

סיבוכיות: $O(\log n)$

חוץ מהלולאה הפונקציה מבצעת עבודה קבועה. בכל איטרציה של הלולאה הפונקציה מבצעת עבודה קבועה ולכן העבודה במסגרת הלולאה היא כמספר האיטרציות. כלומר, כגובה העץ לכל היותר. נקבל סיבוכיות זמן במקרה הגרוע של $O(\log n)$ כגובה של עץ AVL תקין.

<u>הפונקציה getSize() :</u>

תכלית:

הפונקציה מחזירה את גודל העץ.

אופן פעולה:

הפונקציה מחזירה את שדה ה-size של העץ.

סיבוכיות: $O(1)$.

<u>הפונקציה getMaximum() :</u>

תכלית:

הפונקציה מחזירה מצביע לצומת בעל הערך המקסימלי בעץ.

אופן פעולה:

הפונקציה מחזירה את השדה maximum של העץ.

סיבוכיות: $O(1)$.

הפונקציה ()getMinimum :

תכלית:

הפונקציה מחזירה מצביע לצומת בעל הערך המקסימלי בעץ.

אופן פעולה:

הפונקציה מחזירה את השדה minimum של העץ.

סיבוכיות: $O(1)$.

הפונקציה treePosition(int key, IAVLNode curr) :

תכלית:

הפונקציה מחזירה את הצומת בעל המפתח key בתת העץ ששורשו curr אם הוא קיים. אחרת, הפונקציה תחזיר את ההורה של הצומת המתאים לו היה קיים בעץ.

אופן פעולה:

אם הפונקציה קיבלה null, היא תחזיר null. הפונקציה מבצעת חיפוש במורד העץ כפי שראינו בכיתה. תנאי העצירה של לולאת החיפוש הוא כאשר הגענו לצומת בעל המפתח הרצוי או כאשר הגענו לעלה מדומה.

סיבוכיות: $O(\log n)$

בכל צעד של החיפוש, הפונקציה יורדת רמה אחת בעץ ומבצעת מספר קבוע של פעולות, שכל אחת לוקחת זמן קבוע. פרט ללולאת החיפוש, הפונקציה מבצעת מספר קבוע של פעולות שכל אחת לוקחת זמן קבוע. מכיוון שגובה עץ AVL הוא $O(\log n)$, הסיבוכיות של הפונקציה היא $O(\log n)$.

הפונקציה (getSuccessor(IAVLNode node :

תכלית:

הפונקציה מחזירה את ה-successor של הצומת node (בהנחה שמדובר בצומת שאינו עלה מדומה) אם הוא קיים, ואחרת מחזירה null.

אופן פעולה:

אם node הוא עלה מדומה, נחזיר null. אחרת, נבצע את האלגוריתם שראינו בכיתה למציאת successor תוך שימוש בפונקציית העזר findMinNode (סיבוכיות $O(\log n)$) על הבן הימני של הצומת במידה וקיים.

סיבוכיות: $O(\log n)$

הפונקציה מבצעת מספר קבוע של פעולות שלוקחות זמן קבוע, פרט לפעולות הבאות.

לולאת ה-while שבכל איטרציה עולה רמה בעץ. הסיבוכיות שלה היא $O(\log n)$, כגובה של עץ AVL.

הקריאה לפונקציה findMinNode. סיבוכיות הפונקציה היא $O(\log n)$ כפי שמפורט בתיעוד שלה בהמשך המסמך. בסה"כ סיבוכיות הפונקציה אם כך היא $O(\log n)$.

הפונקציה (getPredecessor(IAVLNode node :

תכלית:

הפונקציה מחזירה את ה-predecessor של הצומת node (בהנחה שמדובר בצומת שאינו עלה מדומה) אם הוא קיים, ואחרת מחזירה null.

אופן פעולה:

אם node הוא עלה מדומה, נחזיר null. אחרת, נבצע את האלגוריתם שראינו בכיתה למציאת predecessor תוך שימוש בפונקציית העזר findMaxNode (סיבוכיות $O(\log n)$) על הבן השמאלי של הצומת במידה וקיים.

סיבוכיות: $O(\log n)$

הפונקציה מבצעת מספר קבוע של פעולות שלוקחות זמן קבוע, פרט לפעולות הבאות.

לולאת ה-while שבכל איטרציה עולה רמה בעץ. הסיבוכיות שלה היא $O(\log n)$, כגובה של עץ AVL.

הקריאה לפונקציה findMaxNode. סיבוכיות הפונקציה היא $O(\log n)$ כפי שמפורט בתיעוד שלה בהמשך המסמך. בסה"כ סיבוכיות הפונקציה אם כך היא $O(\log n)$.

<u>הפונקציה private IAVLNode deleteNode(IAVLNode node, String typeNode, String dirNode):</u>

תכלית: הפונקציה מוחקת את הצומת האמיתי בעץ *node* לפי כללי המחיקה שלמדנו בכיתה. הפונקציה מחזירה את ההורה של הצומת שנמחקה בפעולה (*null* במקרה שנמחק השורש). אם הצומת שנמחקת היא בינרית אז הצומת שתוחזר היא ההורה של ה-*successor* של *node*.

אופן הפעולה: מחיקת צומת מעץ לפי המצבים שתוארו בכיתה.

סיבוכיות: $O(\log n)$ במקרה הגרוע (כאשר נמחק צומת בינרית כך שפעולת *getSuccessor* תעלה עבורו $O(\log n)$. בשאר המקרים הסיבוכיות תהיה קבועה.

<u>הפונקציה private static void updateSizeNodesFromNode(IAVLNode node):</u>

תכלית: הפונקציה מעדכנת את רשומת sizeNode לכל צומת בעץ החל מהצומת node כלפי מעלה עד לשורש העץ, לפי האינווריאנטה הבאה:

node.sizeNode = node.left.sizeNode + node.right.sizeNode + 1

אופן הפעולה:

הפעולה מבצעת לולאה מהצומת node עד לשורש העץ, ובכל פעם קוראת לפונקציה updateSizeNode(IAVLNode node) אשר מיישמת את האינווריאנטה על הצומת.

סיבוכיות: $O(root.height\ -\ node.height\ +\ 1)$

הפעולה מבצעת איטרציות כהפרש בין הגובה של הצומת *node* לגובה העץ (במעלה הדרך אל השורש).

בכל איטרציה מבוצעת קריאה לupdateSizeNode שסיבוכיות הזמן שלה היא $O(1)$.

<u>הפונקציה private static void updateSizeNode(IAVLNode node):</u>

תכלית:

עדכון מקומי של השדה sizeNode של הצומת node לפי האיווריאנטה הבאה:

node.sizeNode = node.left.sizeNode + node.right.sizeNode + 1

ואם הצומת אינה אמיתי אז הערך הנ"ל מאותחל ל-0.

אופן הפעולה: השמת הערך הנדרש.

סיבוכיות: $O(1)$

כל הפעולות קבועות.

הפונקציה ()updateMin :

תכלית:

הפונקציה מעדכנת את שדה ה-minimum של העץ להיות אכן הצומת בעל המפתח המינימלי.

אופן פעולה:

הפונקציה קוראת ל-findMinNode על השורש של העץ (סיבוכיות $O(\log n)$).

סיבוכיות: $O(\log n)$.

הפונקציה ()updateMax :

תכלית:

הפונקציה מעדכנת את שדה ה-maximum של העץ להיות אכן הצומת בעל המפתח המינימלי.

אופן פעולה:

הפונקציה קוראת ל-findMaxNode על השורש של העץ (סיבוכיות $O(\log n)$).

סיבוכיות: $O(\log n)$.

הפונקציה findMinNode(IAVLNode node):

תכלית:

הפונקציה מחזירה מצביע לצומת בעל המפתח המינימלי בתת העץ ששורשו node.

אופן פעולה:

אם node הוא null הפונקציה מחזירה null. אחרת, הפונקציה יורדת שמאלה מ-node עד שהיא נתקלת בצומת שאינו אמיתי ומחזירה את ההורה שלו.

סיבוכיות: $O(\log n)$.

חוץ מלולאת ה-while, הפונקציה מבצעת מספר קבוע של פעולות שכל אחת לוקחת זמן קבוע.

בלולאת ה-while, הפונקציה יורדת בכל פעם רמה אחת בתת העץ ומבצעת עבודה קבועה. העומק המירבי של תת עץ של עץ AVL תקין הוא $O(\log n)$ ולכן זו סיבוכיות הפונקציה.

הפונקציה findMaxNode(IAVLNode node):

תכלית: הפונקציה מחזירה מצביע לצומת בעל המפתח המקסימלי בתת העץ ששורשו node.

אופן פעולה: אם node הוא null הפונקציה מחזירה null. אחרת, הפונקציה יורדת ימינה מ-node עד שהיא נתקלת בצומת שאינו אמיתי ומחזירה את ההורה שלו.

סיבוכיות: $O(\log n)$.

חוץ מלולאת ה-while, הפונקציה מבצעת מספר קבוע של פעולות שכל אחת לוקחת זמן קבוע. בלולאת ה-while, הפונקציה יורדת בכל פעם רמה אחת בתת העץ ומבצעת עבודה קבועה. העומק המירבי של תת עץ של עץ AVL תקין הוא $O(\log n)$ ולכן זו סיבוכיות הפונקציה.

הפונקציה private String getKindOfNode(IAVLNode node):

תכלית: הפונקציה מחזירה מחרוזת המכילה סיווג של הצומת node לפי הסוג: עלה, אונרי או בינרי, ולפי היחס שלו כבן של ההורה שלו: שורש, בן ימני או בן שמאלי.

אופן הפעולה: הפונקציה מבצעת בדיקות ישירות כדי לסווג את היחס של node להורה שלו וקובעת כך אם node הוא שורש, בן ימני להורה שלו או בן שמאלי להורה שלו. כמו כן, הפונקציה קוראת לפעולות isUnary(IAVLNode node) ו- isLeaf(IAVLNode node) כדי לקבוע את הסיווג השני לסוג הצומת.

הפונקציה מחזירה בהתאם את אחת המחרוזות הבאות:

Leaf_root / Leaf_right / Leaf_left / Unary_root / Unary_right / Unary_left / Binary_root / Binary_right / Binary_left.

הפונקציה מחזירה מחרוזת ריקה אם node הוא null או צומת שאינה אמיתית.

סיבוכיות: $O(1)$ – כל הפעולות קבועות וכל הקריאות לפונקציות הן בסיבוכיות קבועה.

הפונקציה private boolean isUnary(IAVLNode node):

תכלית: הפונקציה בודקת ומחזירה ערך בוליאני מתאים אם הצומת node היא צומת אמיתי מסוג צומת אונרית.

אופן הפעולה: בדיקה ישירה לפי הגדרה של צומת אונרית כצומת עם בן אחד (אמיתי) בדיוק.

סיבוכיות: $O(1)$ – כל הפעולות קבועות.

הפונקציה *private boolean isLeaf(IAVLNode node)*:

תכלית: הפונקציה בודקת ומחזירה ערך בוליאני מתאים אם הצומת node היא צומת אמיתי מסוג עלה.

אופן הפעולה: בדיקה ישירה לי הגדרה של עלה כצומת אמיתי ששני הבנים שלו אינם צמתים אמיתיים (או לחילופין, ללא בנים אמיתיים).

סיבוכיות: $O(1)$ – כל הפעולות קבועות.

הפונקציה rotateRight(IAVLNode currSon) :

תכלית: הפונקציה מבצעת סיבוב קשת ימינה.

אופן פעולה:

הפונקציה פועלת לפי האלגוריתם שראינו בכיתה תוך תחזוקת השדות המתאימים ולבסוף קוראת לפונקציה updateSizeNode (סיבוכיות ($O(1)$)) עבור שני הצמתים שסובבנו ביניהם.

סיבוכיות: $O(1)$

הפונקציה מבצעת עבודה קבועה וקוראת לפונקציה updateSizeNode שגם היא מבצעת עבודה קבועה.

הפונקציה rotateLeft(IAVLNode currSon) :

תכלית: הפונקציה מבצעת סיבוב קשת שמאלה.

אופן פעולה:

הפונקציה פועלת לפי האלגוריתם שראינו בכיתה תוך תחזוקת השדות המתאימים ולבסוף קוראת לפונקציה updateSizeNode (סיבוכיות ($O(1)$)) עבור שני הצמתים שסובבנו ביניהם.

סיבוכיות: $O(1)$

הפונקציה מבצעת עבודה קבועה וקוראת לפונקציה updateSizeNode שגם היא מבצעת עבודה קבועה.

הפונקציה demote(IAVLNode) :

תכלית: הפונקציה מבצעת demote.

אופן פעולה:

הפונקציה מנמיכה את שדה ה-height של הצומת ב-1 בעזרת שימוש בפונקציות getHeight ו setHeight. שתיהן מסיבוכיות $O(1)$.

סיבוכיות: $O(1)$

הפונקציה promote(IAVLNode) :

תכלית: הפונקציה מבצעת promote.

אופן פעולה:

הפונקציה מגדילה את שדה ה-height של הצומת ב-1 בעזרת שימוש בפונקציות getHeight ו setHeight. שתיהן מסיבוכיות $O(1)$.

סיבוכיות: $O(1)$

הפונקציה balanceAfterInsertion(IAVLNode newParent) :

תכלית: הפונקציה מבצעת את פעולת האיזון במקרה של הכנסה רגילה של צומת ובמקרה של הכנסת צומת לעץ במהלך פעולת join. הפונקציה מחזירה את מספר פעולות האיזון שנדרשו.

אופן פעולה:

הפונקציה מקבלת את ההורה לצומת החדש שהוכנס. אם ההורה הוא null הפונקציה לא מבצעת שום פעולת איזון ולכן מחזירה 0. אחרת, הפונקציה נכנסת ללולאה של פעולות איזון כלפי מעלה כל עוד מתקיים שלהורה הנוכחי יש הפרשי גבהים לא חוקיים כלפי הבנים שלו. הפונקציה משתמשת בפונקציית העזר isLegalAVLNode (סיבוכיות $O(1)$).

בכל איטרציה של הלולאה הפונקציה בודקת את כל אחד מה case-ים שראינו בהרצאה, כולל בדיקה של המצבים הסימטריים להם. בנוסף, יש case שלא ראינו בהרצאה אך אנו יכולים לפגוש במהלך הכנסה בעת ביצוע join והוא המקרה שבו בין ההורה לצומת החדש שהוכנס יש הפרש גבהים של 0, הבן השני של ההורה בהפרש 2 ולצומת החדש שהוכנס יש שני בנים עם הפרש דרגות של 1.

במקרה כזה ראינו לנכון לבצע סיבוב בין הצומת החדש לאביו, promote לצומת החדש והמשך בלולאה.

סיבוכיות: $O(\log n)$

הפונקציה עולה לכל היותר את כל גובה העץ, ובכל רמה מבצעת עבודה קבועה. לכן סיבוכיות הפונקציה היא כגובה של עץ AVL תקין, $O(\log n)$.

הפונקציה <u>private int balanceAfterDeletion(IAVLNode parentOfDeleted):</u>

תכלית:

הפונקציה מקבלת הורה של צומת שנמחקה מהעץ ומבצעת פעולות לאיזון העץ כדי שיישאר עץ AVL תקין לאחר המחיקה. הפעולה מחזירה את מספר פעולות האיזון שנדרשו לצורך איזון העץ.

אופן פעולה:

פעולות האיזון מבוצעות תחילה בהתאם להפרשי הדרגות של הצומת parentOfDeleted לבנים שלו.

הפעולה מבצעת איזון של העץ בהתאם למקרים שלמדנו בכיתה ע"י פעולות demote, promote, rotateLeft, rotateRight וממשיכה באיזון כלפי מעלה ובבחינת המקרים כלפי מעלה בעץ אם נדרש.

סיבוכיות: $O(\log n)$.

כל הפעולות שמבוצעות במהלך הלולאה שעוברת במקרה הצורך על העץ עד לשורש הן קבועות. לכן נקבל במקרה הגרוע בו האיזון יסתיים רק בשורש והתחלנו בצומת שקרובה לעלים בעץ - $\log n$ איטרציות.

הפונקציה <u>private boolean isLegalAVLNode(IAVLNode node):</u>

תכלית:

הפונקציה מחזירה ערך בוליאני בהתאם לבדיקה אם הצומת node היא צומת תקינה בעץ AVL בהתאם להפרש הגבהים שלה עם הבנים שלה. כלומר, תחזיר true אם הפרש הגבהים של הצומת עם הבנים שלה הוא מאחד הסוגים: 1-1, 1-2, 2-1. אחרת תחזיר false.

הנחה מקדימה: הצומת node היא צומת אמיתי.

אופן הפעולה: בדיקה ישירה תוך שימוש בפונציות ()getHeight של הצומת ושל הילדים שלה.

סיבוכיות: $O(1)$ – כל הפעולות קבועות.

חלק ניסויי / תיאורטי

שאלה 1

סעיף א׳ –

מספר סידורי i	מספר חילופים במערך ממוין- הפוך	עלות החיפושים במיון AVL עבור מערך ממוין-הפוך	מספר חילופים במערך מסודר אקראית	עלות החיפושים במיון AVL עבור מערך מסודר אקראי
1	1,999,000	38,884	985,865	34,084
2	7,998,000	85,764	4,026,195	77,017
3	31,996,000	187,524	16,016,746	175,119
4	127,992,000	407,044	63,873,480	377,296
5	511,984,000	878,084	255,654,903	813,583

סעיף ב׳ –

ניתוח תיאורטי של מספר החילופים:

במערך ממוין הפוך בגודל n לאיבר במקום ה-i יש $n-i-1$ איברים אחריו והם כולם קטנים ממנו.

לכן, מספר החילופים במערך ממוין הפוך בגודל n הוא:

$$(n-1)+(n-2)+\cdots+(1)+(0)=\sum_{k=0}^{n-1}k=\frac{n(n-1)}{2}$$

ניתוח תיאורטי של עלות החיפושים:

בכל פעולת חיפוש אנחנו מתחילים מהאיבר המקסימלי של העץ ומגיעים למיקום של האיבר המינימלי בעץ כי המערך ממוין הפוך. לכן, עלות כל חיפוש תהיה פעמיים גובה העץ הנוכחי.

נציע חסם עליון עבור עלות החיפושים במקרה של מערך ממוין הפוך.

נצפה אם כך, שעלות החיפוש הגדולה ביותר תתקבל כאשר העץ הכי גדול, כלומר בהכנסה האחרונה.

גובה העץ בהכנסה האחרונה הוא $\log n$ בגלל שמדובר בעץ AVL תקין.

נקבל חסם עליון לעלות הכוללת של פעולות החיפוש כשנסכום עבור n פעולות החיפוש את העלות המקסימלית. נקבל: $O(\,n\cdot\log n)$.

כעת נציע חסם תחתון עבור עלות החיפושים במקרה של מערך ממוין הפוך.

נסתכל על עלות החיפוש במצב שכבר הכנסנו לעץ $\frac{n}{2}$ איברים מהמערך.

במצב זה גובה העץ הוא $\log\frac{n}{2} = \log n - 1$.

כל פעולת הכנסה נוספת תעלה לפחות $\log\frac{n}{2}$ (כי גובה העץ לא יקטן החל מנקודה זו). לכן, נוכל לקבל חסם תחתון עבור העלות הכוללת של פעולות החיפוש, אם נסכום עלות מינימלית של $\log\frac{n}{2}$ עבור $\frac{n}{2}$ ההכנסות שנותרו. נקבל חסם תחתון של: $\frac{n}{2}\cdot\log\frac{n}{2} = \frac{1}{2}(n\cdot\log n) - \frac{n}{2} = \Omega(n\cdot\log n)$.

בסה״כ נקבל שהעלות הכוללת של פעולות החיפוש שנבצע בעת ההכנסות של האיברים לעץ ממערך ממוין הפוך היא: $\theta(n\cdot\log n)$.

<u>סעיף ג׳ –</u>

ראשית נסתכל על מספר החילופים.

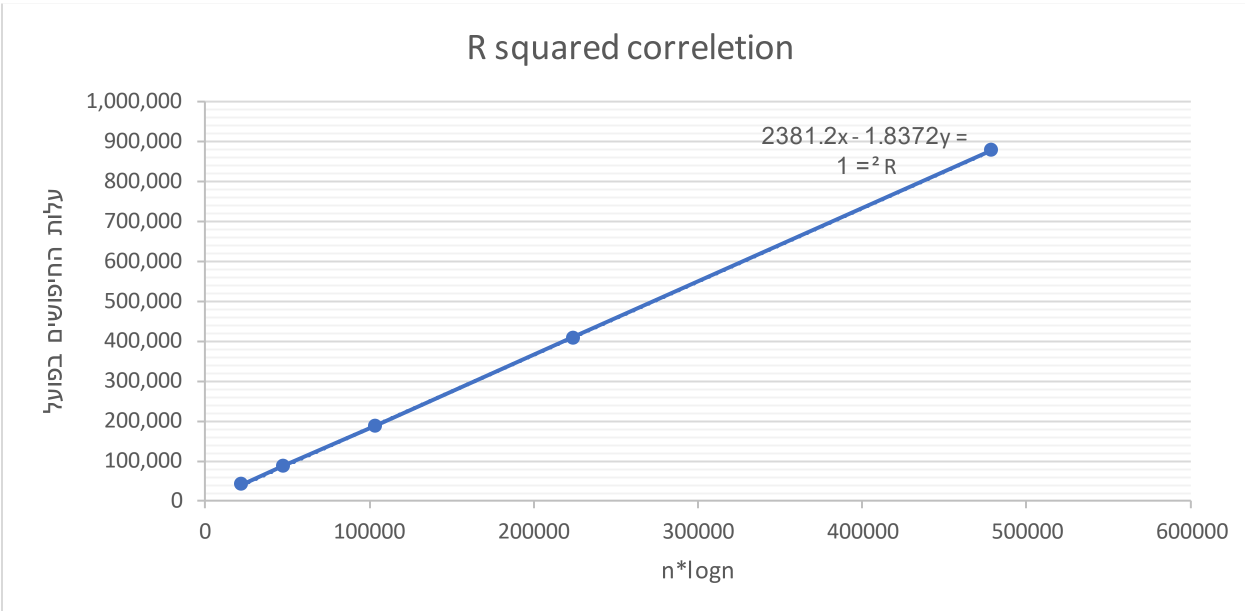
מספר סידורי i	גודל המערך n	מספר חילופים במערך ממוין- הפוך	מספר החילופים לפי החישוב שלנו $\frac{n(n-1)}{2}$
1	2000	1,999,000	$\frac{2000(2000-1)}{2}$ = 1,999,000
2	4000	7,998,000	$\frac{4000(4000-1)}{2}$ = 7,998,000
3	8000	31,996,000	$\frac{8000(8000-1)}{2}$ = 31,996,000
4	16,000	127,992,000	$\frac{16000(16000-1)}{2}$ = 127,992,000
5	32,000	511,984,000	$\frac{32000(32000-1)}{2}$ = 511,984,000

כעת נסתכל על מידת ההתאמה בנוגע לניתוח של עלות החיפושים.

לאקסל נכניס את הטבלה הבאה. השורה במקום ה-i מלמעלה מייצגת ערך n המחושב לפי ערך ה-i.

מספר סידורי i	גודל המערך n	עלות החיפושים במיון AVL עבור מערך ממוין-הפוך	עלות החיפושים לפי החסם שלנו $n\cdot\log n$
1	2000	38,884	$2000\log 2000 = 21{,}931.57$
2	4000	85,764	$4000\log 4000 = 47{,}863.14$
3	8000	187,524	$8000\log 8000 = 103{,}726.27$
4	16,000	407,044	$16000\log 16000 = 223{,}452.55$
5	32,000	878,084	$32000\log 32000 = 478{,}905.10$

נציג כעת את גרף המתאם שקיבלנו המתאר את הטבלה לעיל. ערך ה- R^2 שקיבלנו בחישוב ישיר בעזרת ה-excel הוא: $0.999993569 \approx 1$.



נראה שיש התאמה חזקה מאוד בין החסם האסימפטוטי שחישבנו בסעיף הקודם לבין הנתונים שהתקבלו בניסוי בפועל.

<u>סעיף ד׳ –</u>

נגדיר את h_i להיות מספר האיברים שקודמים לאיבר ה-i במערך וגדולים ממנו. באופן זה $\sum_{i=0}^{n-1} h_i = h$.

הגדרה זו של h_i תואמת את הגדרת מספר החילופים כפי שראינו בסעיף א׳.

ראשית ננסה לחשב את עלות החיפוש עבור ההכנסה של האיבר ה-i במערך בהתחשב בערך h_i.

בעת חיפוש המיקום להכנסת האיבר ה-i במערך נצטרך ״לדלג״ על לכל היותר h_i איברים שהכנסנו קודם לכן, כלומר, לעלות מהמקסימום לכל היותר $\log h_i$ רמות.

על כן, נוכל לחשב חסם עליון עבור עלות המיון באופן הבא:

$$\sum_{i=0}^{n-1}\log h_i = \log\left(\prod_{i=0}^{n-1} h_i\right) = \log\left(\left(\left(\prod_{i=0}^{n-1} h_i\right)^{\frac{1}{n}}\right)^n\right) \le \log\left(\left(\frac{\sum_{i=0}^{n-1} h_i}{n}\right)^n\right) = n\cdot\log\left(\frac{h}{n}\right)$$

במעבר של אי השוויון השתמשנו באי שוויון הממוצעים ובמונוטוניות הלוגריתם. בשאר המעברים השתמשנו בתכונות של לוגריתם.

כלומר, קיבלנו חסם עליון של: $O\left(n\cdot\log\left(\frac{h}{n}\right)\right)$.

סעיף ה' –

הסיבה נתבקשנו למצוא חסם עליון בלבד ולא חסם $\theta(\cdot)$ היא שעבור מספר חילופים h מסויים, נוכל למצוא התנהגות אסימפטוטית שונה כחסם תחתון.

לדוגמה, נסתכל על שני המערכים הבאים.

במערך הראשון המחצית הראשונה של האיברים ממוינים בסדר עולה והמחצית השנייה בסדר יורד.

$$\left[\frac{n}{2}, \frac{n}{2} - 1, \dots, 1, \frac{n}{2} + 1, \frac{n}{2} + 2, \dots, n\right]$$

במערך השני המחצית הראשונה של האיברים ממוינים בסדר הפוך והמחצית השנייה ממוינים בסדר עולה.

$$\left[1, 2, \dots, \frac{n}{2}, n, n - 1, \dots, \frac{n}{2} + 1\right]$$

בשני המצבים h זהה.

<u>שאלה 2</u>

<u>סעיף א׳ –</u>

מספר סידורי i	עלות join **ממוצע** עבור split **אקראי**	עלות join **מקסימלי** עבור split **אקראי**	עלות join **ממוצע** עבור split של **האיבר** המקסימלי בתת העץ השמאלי	עלות join **מקסימלי** עבור split של **האיבר** המקסימלי בתת העץ השמאלי
1	2.4	6	2.56	11
2	2.9	5	2.5	12
3	2.42	5	2.73	14
4	2.29	6	2.69	15
5	2.42	5	2.2	16
6	2.24	4	2.4	17
7	1.87	4	2.22	18
8	2.16	6	2.65	20
9	2.47	5	2.48	21
10	2.58	5	2.56	22

<u>סעיף ב׳ –</u>

תשובה:

נצפה שעלות join ממוצע עבור split אקראי תהיה כ-2.5.

הסבר:

הפרש הדרגות הממוצע בין צומת להורה שלה בעץ AVL הוא כ-1.5 (הפרש הדרגות המותר הוא 1 או 2 בין בן להורה). אם נסכום את העלות הכוללת של כל פעולות ה-join שנעשה עבור split מסוים, נקבל שהיא שווה לדרגה של שורש העץ פחות הדרגה של הצומת המקורי ועוד 1 (בהתבסס על האופן שבו הגדרנו סיבוכיות join). לכן העלות הכוללת של כל פעולות ה-join היא כמספר הרמות שנעבור דרכן כפול $\approx 1.5 + 1 = 2.5$. נקבל עלות כוללת של כ- $2.5 \log n$.

מספר פעולות ה-join שנבצע היא כמספר הצמתים שנעבור דרכם בטיפוס למעלה מהצומת המקורי עד לשורש העץ, כלומר, כ- $\log n$.

לכן, אם נבצע חישוב ממוצע עבור פעולת join אחת, נקבל שהיא תעלה כ- ≈ 2.5.

הסיבה לכך שבשני התרחישים התוצאות דומות היא שבשני התרחישים עבדנו עם עצי AVL תקינים ולכן ההסבר לעיל תקף עבור שניהם.

נציין שבשני התרחישים קיבלנו תוצאות שקרובות לממוצע שחישבנו בגלל שבשני התרחישים אנו מצפים לקבל צומת בגובה מאוד נמוך בעץ, מה שיטיב עם הערך הממוצע שחישבנו לפעולת join.

הרוב המוחלט של הצמתים נמצא ברמות הנמוכות של העץ ולכן בעת בחירת צומת אקראי נצפה שהוא יהיה נמוך יחסית. בנוגע לתרחיש השני, מעצם הגדרת הצומת המבוקש כאיבר מקסימלי בתת העץ השמאלי של השורש, נצפה שהוא יהיה צומת נמוך גם כן.

<u>סעיף ג׳ –</u>

העלות של join מקסימלי בתרחיש שבו ביצענו split על האיבר המיוחד שבחרנו הוא כגובה העץ, כלומר:

$$\log(n) = \log\left(1000 * 2^i\right) = \log(1000) + \log\left(2^i\right) \approx 10 + i$$

הסיבה לכך היא שמהאופן שבו בחרנו את האיבר המיוחד, כ-predecessor של השורש, נובעים הדברים הבאים:

1. האיבר שבחרנו יהיה בגובה 0 או 1 מתוקף היותו מקסימום בתת עץ AVL תקין.
2. כל הצמתים בעץ עם מפתחות הגדולים מהצומת שבחרנו יהיו בתת העץ הימני של השורש.
3. במהלך ביצוע ה-split, כדי להגיע מהאיבר המיוחד שבחרנו לשורש, נבצע עליות מצד ימין בלבד, פרט לעלייה האחרונה שתהיה עלייה מצד שמאל אל השורש עצמו.
4. אם כך, בפעולת ה-join האחרונה נאלץ לבצע פעולת join של עץ ריק (כי לא נאספו צמתים הגדולים מהאיבר המיוחד) יחד עם תת העץ הימנו של השורש. עלות פעולה זו תהיה גובה העץ, כהפרש הגבהים של שני תתי העצים הללו.
5. נבחין כי עלות פעולת join המקסימלית היא לכל היותר גובה העץ המקורי. הסיבה לכך היא שעלות פעולת join מוגדרת כהפרש הגבהים של שני תתי העצים ושני תתי העצים שלהם הפרש הגבהים הגדול ביותר, הם תת עץ מידי של השורש יחד עם עץ ריק.

הניתוח התיאורטי אכן תואם את הממצאים בטבלה.

סעיף ד׳ –

עלות ה join המקסימלי במהלך פעולת split יהיה כהפרש הגבהים הגדול ביותר בין שני תתי עצים שנבצע עליהם join. נקבל את ההפרש הגדול ביותר עבור הרצף הגדול ביותר של פניות לכיוון מסוים במהלך העלייה כלפי מעלה. לכן, חסם אסימפטוטי על עלות join זה יהיה גודל הרצף הארוך ביותר של פניות לכיוון מסוים.

על מנת לענות על השאלה נרצה לחשב שני גדלים:

1. את תוחלת הגובה של צומת אקראי בעץ אקראי בעל n צמתים.
2. את התוחלת על גודל הרצף הארוך ביותר של פניות לכיוון מסוים.

עבור החישוב הראשון:

נחשב את התוחלת.

$$\frac{1}{n}\cdot\log n+\frac{2}{n}\cdot(\log n-1)+\cdots+\frac{2^i}{n}\cdot(\log n-i)+\cdots+\frac{2^{\log n-1}}{n}\cdot(\log n-\log n)=$$

$$=\frac{\log n}{n}\cdot\left(1+2+4+\cdots+\frac{n}{2}\right)-\frac{1}{n}\left(2\cdot 1+2^2\cdot 2+2^3\cdot 3+\cdots+n\cdot\log n\right)=$$

$$=\frac{\log n}{n}\left(\frac{n}{2}-1\right)-\frac{1}{n}(2n\cdot\log n-2n+2)=O(1)$$

קיבלנו שהתוחלת של גובה של צומת אקראי בעץ AVL תקין הוא מספר קבוע. נצפה שגובה זה יהיה סביב הרמה הנמוכה ביותר בעץ.

הדבר מתיישב עם העובדה שככל שיורדים ברמות של עץ AVL תקין, (שהוא יחסית מאוזן), יש יותר צמתים באופן מעריכי. אם כך, שוב נצפה שגובה של צומת אקראי יהיה סביב הרמה הנמוכה ביותר בעץ.

עבור החישוב השני:

ראינו בקורס שעבור בהסתברות שעבור k הטלות מטבע, כשכל תוצאה מתקבלת בסיכוי שווה, תוחלת אורך הרצף הארוך ביותר היא בסדר גודל של: $\log k$

נוכל להשליך את חישוב זה על הבעיה הנוכחית ולהסיק כי עבור k פניות שונות, ימינה ושמאלה, שנבחרות באקראי בהסתברות שווה, תוחלת הרצף הגדול ביותר של פניות לכיוון מסוים היא $\log k$.

אם כך, עבור גובה ממוצע 0 של צומת אקראי בעץ בגובה $O(\log n)$, אורך המסלול שנבצע מהצומת האקראי לשורש יהיה $O(\log n)$ ולכן התוחלת של הרצף הארוך ביותר של פניות לכיוון מסוים היא $O(\log\log n)$.

החסם מתיישב עם הטבלה, מכיוון שעבור גדלי העצים בשאלה, $\log\log n$ הוא מספר כמעט קבוע לכן, התוצאות מתיישבות אחת עם השנייה.