תיעוד

<u>המחלקה AVLNode:</u>

כל עצם במחלקה הזו מהווה צומת אפשרית בעץ חיפוש בינארי או עצי AVL, לכל עצם ששייך למחלקה הזו ישנם 7 שדות והם:

1. key : המפתח של הצומת, המפתחות הינם ייחודיים כלומר לכל צומת יש מפתח ששונה מהצמתים האחרים בעץ. טיפוס int.
2. info: מחרוזת שמכילה מידע כלשהו שאנו צריכים לשמור בצומת. טיפוס String.
3. size: מכיל את כמות הצמתים שנמצאים בתת עץ כאשר השורש הוא הצומת הנוכחית, כולל את הצומת הנוכחי. טיפוס int.
4. height: מכיל את הגובה של הצומת הזו. טיפוס int.
5. left: מצביע לבן השמאלי של הצומת הזו. טיפוס IAVLNode.
6. right: מצביע לבן הימני של הצומת הזו. טיפוס IAVLNode.
7. parent: מצביע להורה של הצומת הזו. טיפוס IAVLNode.

<u>המתודות של המחלקה:</u>

- getKey(): מחזירה את הערך של השדה key בסיבוכיות זמן O(1).
- getValue(): מחזירה את הערך של השדה info בסיבוכיות זמן O(1).
- getHeight(): מחזירה את הגובה של הצומת ע"י חישוב המקסימום בין הגובה של הבן הימני לגובה של הבן השמאלי של הצומת, ומוסיפה לערך זה אחד ,כלומר מחזירה את הערך של השדה height בסיבוכיות זמן O(1). (לא קוראת ל getHeight של הבנים שלה, אלה משתמשת ישירות בשדה height שלהם).
- getSize(): מחזירה את הערך של השדה size בסיבוכיות זמן O(1).
- getLeft(): מחזירה את הערך של השדה left בסיבוכיות זמן O(1).
- getRight(): מחזירה את הערך של השדה right בסיבוכיות זמן O(1).
- getParent(): מחזירה את הערך של השדה parent בסיבוכיות זמן O(1).
- setLeft(IAVLNode node), setRight(IAVLNode node) : הפונקציה מעדכנת את השדה right/left של הצומת כך שיכלול את הצומת שהתקבלה כפרמטר, אם הצומת שווה ל-null היא מעדכנת אותה להיות צומת ווירטואלית. בנוסף לכך, היא קוראת לפונקציית עזר בשם updateHeightSize כאשר היא מעדכנת את הערך של הגובה והגודל של הצומת הנוכחית בהינתן הבן החדש. הסיבוכיות שלה הינה O(1) ונציין את זה בהמשך. לכן הפעולה מבצעת שינוי של פוינטירים ופעולה נוספת בעלות O(1) ולכן הסיבוכיות שלה הינה O(1).
- setParent(IAVLNode node): הפונקציה מעדכנת את השדה parent של הצומת כך שיכלול את הצומת שהתקבלה כפרמטר,הפעולה מבצעת שינוי של פוינטיר ולכן הסיבוכיות שלה הינה O(1).
- setHeight(int Height): הפונקציה מעדכנת את השדה height של הצומת הנוכחית. הסיבוכיות הינה O(1).
- setSize(int size): הפונקציה מעדכנת את השדה size של הצומת הנוכחית. הסיבוכיות הינה O(1).
- isRealNode(): מחזירה True לכל צומת שהוא לא עלה חיצוני (VirtualNode). O(1).

פונקציות עזר למחלקה זו:

updateSizeHeight(): מעדכנת את ה-height של הNode להיות המקסימום בין הheights של הבנים שלו פלוס אחד וגם מעדכן את ה-size של הNode להיות סכום הsizes של הבנים שלו פלוס אחד, הסיבוכיות הינה O(1).

BFCalc(): מחשבת ומחזירה את ה-balance factor של הNode שהוא הheight של הבן השמאלי פחות הheight של הבן הימני, הסיבוכיות הינה O(1).

המחלקה AVLTree:

כל עצם במחלקה הזו מהווה עץ חיפוש בינארי מסוג AVL .

למחלקה הזו יש 3 שדות :

1. Root: השדה הזה הוא מסוג IAVLNode ומייצג את שורש של העץ.
2. Min: השדה הזה הוא מסוג IAVLNode ומייצג את הNode המינימאלי בעץ.
3. Max: השדה הזה הוא מסוג IAVLNode ומייצג את הNode המקסימאלי בעץ.

המתודות של המחלקה AVLTree:

- empty(): הפונקציה מחזירה true אם העץ הינו ריק, אחרת היא מחזירה false כלומר הפונקציה בודקת אם השדה של השורש שווה ל-null או עלה חיצוני וסיבוכיות זמן בריצה הוא O(1).
- search(int k): הפונקציה מחפשת אם יש צומת עם המפתח הנתון k כך שהיא מחזירה את הערך value שלו אם הוא נמצא בעץ, אחרת היא מחזירה null .
שיטת המימוש: הפונקציה קורת לפונקציית עזר searchNode(int k) שמחזירה הNode שאנו מחפשים או null אם הוא לא קיים (נסביר יותר בהמשך), ואז מחזירה את השדה value של הNode סיבוכיות זמן: הפונקציה יורדת בכל פעם רמה אחת בעץ, ובגלל שהעץ הינו עץ AVL עם גובה O(logn) אזי סיבוכיות הזמן הינה O(logn).
- min(): הפונקציה הזו קוראת לשדה min שהוא מסוג IAVLNode אם השדה שווה ל- null אז הפונקציה מחזירה null, אחרת הפונקציה קורת ל-getValue() על השדה min ומחזירה את הערך וסיבוכיות זמן הריצה הוא O(1) בגלל שכבר שמור לנו את הערך המינימאלי בשדה min, לכן אנחנו רק נבקש את השדה הרלוונטי לצומת השמורה בשדה וזה עולה O(1).
- max(): הפונקציה הזו קוראת לשדה max שהוא מסוג IAVLNode אם השדה שווה ל- null אז הפונקציה מחזירה null, אחרת הפונקציה קורת ל-getValue() על השדה max ומחזירה את הערך וסיבוכיות זמן הריצה הוא O(1) בגלל שכבר שמור לנו את הערך המקסימאלי בשדה max, לכן אנחנו רק נבקש את השדה הרלוונטי לצומת השמורה בשדה וזה עולה O(1).
- keysToArray():
שיטת המימוש:
קוראים לפונקציית עזר (nodeToArray) שמחזירה מערך ממוין לפי המפתחות בשיטת in order, ואחר כך, מאתחלים מערך מטיפוס int באורך של המערך שקיבלנו, ועוברים דרך לולאה על כל הצמתים ששמורות במערך ושומרים את ערך המפתח שלהם במערך שלנו, ואחרי סיום הלולאה מחזירים אותו. $O(n) + O(n) = O(n)$ – כי קריאה לפונקציית העז עולה O(n).
infoToArray(): *שיטת המימוש:*

קוראים לפונקציית עזר (nodeToArray) שמחזירה מערך ממוין לפי המפתחות בשיטת in order, ואחר כך, מאתחלים מערך מטיפוס String באורך של המערך שקיבלנו, ועוברים דרך לולאה על כל הצמתים ששמורות במערך ושומרים את הערך שלהם במערך שלנו, ואחרי סיום הלולאה מחזירים אותו. $O(n) + O(n) = O(n)$ – כי קריאה לפונקציית העז עולה O(n).

- size(): הפונקציה הזו קורת לשדה root שהוא מסוג IAVLNode אם השדה שווה ל- null אז הפונקציה מחזירה null, אחרת הפונקציה קורת ל-getSize() על השדה root ומחזירה את הערך וסיבוכיות זמן הריצה הוא O(1).
- getRoot(): הפונקציה מחזירה את השורש של הפונקציה ששמור כשדה, סיבוכיות פעולה O(1).
- insert(int k ,String s): הפונקציה מוסיפה צומת בעלת מפתח k וערך i לעץ, ומחזירה את מספר פעולות האיזון שנדרשו כדי לתקן את העץ.

שיטת המימוש: קוראים לפונקצית העזר searchNode עם המפתח הנתון, כדי לבדוק אם הוא נמצא בעץ כבר או לא, אם כן נמצא, מחזירים -1 ומסתיימת הפעולה, אם לא אז נבצע חיפוש למקום המתאים לצומת החדש, חיפוש זה מתבצע כמו חיפוש בינארי, בכל פעם אנו יורדים רמה עד שמגיעים לצומת ששווה ל-null ושם אנו מוסיפים את הצומת החדש כבן הימני או השמאלי (ההוספה עולה O(1)) לצומת שלפני האחרונה שהגענו אליה בלולאה ולאחר ההוספה של הצומת החדש שימוש בפונקציות: setleft() ו- setright() ו- setparent() בהתאם, ואז קוראים לפונקציית עזר rebalanceInsert(IAVLNode x) שהיא דואגת לאזן את העץ החל מצומת הנתון X.

אחרי האיזון, קוראת לשתי הפונקציות updateSizeHeight() ו- updateMinMax() שמעדכניים את הגובה והוגדל של שורש העץ, וגם את הערכים המינימאליים ומקסימאליים, כך שהכל יהי תקין ומחזירה את מספר פעולות האיזון שהתבצעו בסה"כ בשלב תיקון העץ וסיבוכיות הזמן של הפונקציה הינה מתבטאת על ידי:

$$O(\log n) + O(\log n) + O(\log n) + O(1) + 2 * O(\log n) = O(\log n)$$

- delete(int k):

שיטת מימוש: קוראים לפונקציית עזר searchNode כדי לבדוק אם הצומת נמצאת בעץ, אם לא מחשירים -1 ומסתיימת הפעולה, אם כן נשמור את הצומת הפונקציה ונבדוק אם יש לה בן ימני \ שמאלי, ולפי זה מחליטים איך למחוק אותו, במצב שאין בן ימני \ שמאלי או יש רק אחד אנו מבצעים שינוי פוינטירים שעולה O(1) ומוחקים את הצומת, אחרת אנו מחפשים את העוקב של הצומת שנרצה למחוק בעזרת הפונקציה findSuccessor(IAVLNode x) ואז מוחקים אותו פיזית מהעץ, ומכניסים אותו במקום הצומת שנרצה באמת למחוק, לאחר מכן, קוראים לפונקציית עזר rebalanceDelete(IAVLNode x) כאשר מקבלת את ההורה של הצומת שנמחקה ומתחילה פעולת אישון משם, נוכיח בהמשך שהסיבוכיות שלה היא O(log n). אחרי המחיקה קוראים לפונקציות updateSizeHeight ו- updateMinMax כדי לעדכן שורש העץ והשדים שלו.

לכן סה"כ קיבלנו שהסיבוכיות של פעולת המחיקה היא:

$$O(\log n) + O(\log n) + O(\log n) + O(1) + 2 * O(\log n) = O(\log n)$$

- join(IAVLNode x, AVLTree t):

שיטת מימוש: בהתחלה בודקת אם אחד משני העצים הוא null אם כן אז פשוט מוסיפה את x לעץ מבניהם הלא ריק ע"י שימוש בפונקציה insert (כי אם מוסיפים את הצומת הנתון לעץ ואז מבצעים איזון זה בדיוק לעשות פעולת הכנסה רגילה) ומבצעת עדכון ל-root אם צריך, אם שניהם ריקים אז העץ המתקבל הוא עץ שיש בו רק את x שהוא השורש של העץ, אחרת (כאשר שני העצים לא ריקים) אנו בודקים איזו עץ משניהם הוא בעל ה-height הגדול ביוצר מ-1 מהשני, וקוראים לו longTree והשני shortTree .

אם ההבדל בין ה-heights של שניהם קטן שווה 1 אז מעדכנים את x להיות השורש ושני השורשים של העצים הופכים להיות הבנים של x בהתאם, אחרת יורדים בעץ הגדול ביותר בצורה מתאימה לצורך (כלומר יורדים בבנים ימיניים אם העץ בעל ערכים קטנים מ-X, אחרת יורדים בבנים השמאליים) עד שנגיע למקום שבו ההבדל בין הגבהים הוא 1 או 0 (נקרא לאיבר

זה xson)ושם מוסיפים את x כך שהבנים שלו הם shortTree.root ו-xson וה-parent של x מתעדכן להיות xson.parent ובסוף הפונקציה מאזנים את העץ ע"י שימוש בפונקציות rebalanceInsert(int x) כי למדנו שאיזון פעולת join הוא בדיוק איזון של הכנסה, ו- updateSizeHeight() ו- updateMinMax() בהתאם, סיבוכיות זמן הריצה היא:

$$O(|t.rank - tree.rank|)$$

כי יורדים בעץ הגדול מהם בדיוק longTree.rank-shortTree.rank ובפעולת האיזון עולים אותו מספר של רמות בעץ עד שהוא עץ מאוזן AVL.

- split(int x):

שיטת מימוש: קוראים לפונקציה searchNode כדי לדעת איפה נמצא הצומת שצריך לפצל לפיו.

בפונקציה הזו אנחנו מתחילים באתחול 4 מערכים, שנים מהם מטיפוס AVLTree אחד מהווה תתי עצים בעלי ערכים קטנים מ-X, והשני מהווה תתי עצים בעלי ערכים גדולים מ-X. ושנים מטיפוס IAVLNode, אחד מהווה צמתים שלפניהם נבצע פעולת join בין תתי עצים ששמרנו.
נעבור בלולאה מהצומת שמצאנו עד השורש. לכל הורה בודקים אן הוא צריך להיות בתת עץ שמאלי או ימיני, אם שמאלי, אש נשמור את התת עץ השמאלי שלו במערך מתאים, ונשמור את ההורה עצמו גם במערך מתאים. ובאותה צורה גם שומרים תת עץ ימיני.
אחרי שידענו עכשיו מה הם העצים שצריך למזג אותם, ודרך אילו צמתים, אז נקרא לפעולות join באופן מתאים.

הוכחנו בשיעור סיבוכיות: O(log n)
וזה נכון במימוש שלנו כי במקרה הכי גרוע, הצומת היא עלה ואז בלולאה ראשונה עולים log n רמות, אחר כך מתקיים שסכום כל פעולות המיזוג לא יעלה על log n, הוכחנו בשיעור.

<u>פונקציות עזר:</u>

updateMinMax(): הפונקציה הזו מעדכנת את ה-min/max של הNode ע"י לעבור על המסלול הכי שמאלי וגם כן הכי ימני עד שנגיע לעליים, הסיבוכיות הינה כגובה העץ.

$$O(\log n) + O(\log n) = O(\log n)$$

nodeToArray (): הפונקציה הזו נקצה מערך בגודל ה-size של השורש ומתחילה באיבר המינימאלי בעץ ע"י קריאה לשדה min, מכניסה אותו למערך וקורת לפונקציה findSuccessor(IAVLNode x) n-1 פעמים, החל מהאיבר המינימאלי, ומכניסה כל איבר שתחזיר אותו אותו למערך, נמשיך בצעדים אלו עד שיתמלא המערך, סיבוכיות זמן הריצה הוא O(n) עבור n שהוא מספר האיברים בעץ.

searchNode(int k): הפונקציה מתחילה מהשורש של העץ, משווה בכל פעם בין הkey של הצומת לבין k שהתקבל כפרמטר, אם המפתח של הצומת גדול מ k אנו פונים שמאלה, אם המפתח של הצומת קטן מ k אנו פונים ימינה, אם הם שווים אנו מחזירים את הצומת. אם הלולאה הסתיימה ללא ערך מוחזר אנו נחזיר null, הסיבוכיות הינה כגובה העץ O(logn).

leftRotation(IAVLNode x): הפונקציה מקבלת עץ וצומת בעץ הזה, כך שצריך לעשות סיבוב שמאלי לצומת הזה, הפונקציה מבצעת שינוי פוינטירים של מספר צמתים כך שבסוף אנו מקבלים את הסיבוב שרצינו, סיבוכיות פעולה O(1).

rightRotation(IAVLNode x): הפונקציה מקבלת עץ וצומת בעץ הזה, כך שצריך לעשות סיבוב ימני לצומת הזה, הפונקציה מבצעת שינוי פוינטירים של מספר צמתים כך שבסוף אנו מקבלים את הסיבוב שרצינו, סיבוכיות פעולה O(1).

rebalanceInsert(IAVLNode x): הפונקציה הזו מתחילה מהצומת הנתונה כפרמטר, מחשבים את ה-factor balance של האבא שלה ובודקים אם הוא 2 או -2 אם כן אז בודקים את ה-bf של הבן

הימני\השמאלי שלה (תלוי במסלול שעולים בו) ומבצעים את הרוטציה המתאימה ע"י שימוש בפונקציות leftRotation או rightRotation כדי לאזן את העץ, בסוף אנו משתמשים עוד פעם בפונקציה ()updateSizeHeight כדי לעדכן את ה-size וה-height של הצמתים ששינו את המקום שלהם וגם את הפונקציה הסיבוכיות הינה כאורך העץ O(logn), כי במקרה הכי גרוע נצרך לעשות רוטציות לכל צומת מהעלה עד השורש, אבל רוטציות עולות O(1) ולכן זה יהיה שווה ל- O(log n).

rebalanceDelete(IAVLNode x): הפונקציה הזו מתחילה מהצומת הנתונה כפרמטר, מחשבים את ה-balance factor של הצומת הנתונה, ונבדוק אם הוא שווה ל- 2 או 2- אם כן אז בודקים את ה-bf של הבן הימני\השמאלי שלה, ומבצעים את הרוטציה המתאימה ע"י שימוש בפונקציות leftRotation או rightRotation כדי לאזן את העץ, בסוף אנו משתמשים עוד פעם בפונקציה updateSizeHeight() כדי לעדכן את ה-size וה-height של הצמתים ששינו את המקום שלהם הסיבוכיות הינה כאורך העץ O(logn). (, כי במקרה הכי גרוע נצרך לעשות רוטציות לכל צומת מהעלה עד השורש, אבל רוטציות עולות O(1) ולכן זה יהיה שווה ל- O(log n).

findSuccessor(IAVLNode x): הפונקציה מחפשת את העוקב של הצומת הנתונה כפרמטר ומחזירה אותו, כלומר אם יש בן ימני לצומת אזי הולכים אליו ואז כל הדרך שמאלה עד שמקבלים null אם אין בן ימני לצומת אזי הולכים בכל פעם להורה של הצומת הנוכחית עד הפנייה הראשונה ימינה ומחזירים את מה שמקבלים. סיבוכיות הפעולה O(logn).

<u>המחלקה VirtualNode:</u>

המחלקה הזו מהווה את העליים החיצוניים של העץ, ויורשת מהמחלקה AVLNode את הפונקציות שלה חוץ מ-4 שהיא דורשת:

1. ()getsize: מחזירה 0 שהוא ה-size של כל עלי חיצוני.
2. ()getheight: מחזירה 1- שהוא ה-height של כל עלי חיצוני.
3. ()getkey: מחזירה 1- שהוא ה-key של כל עלי חיצוני.
4. ()isRealNode: מחזירה False תמיד.

מדידות

שאלה 1:

סעיף א':

מספר סידורי	כמות חילופים בניסוי 1	כמות חילופים בניסוי 2	עלות חיפושים בניסוי 1	עלות חיפושים בניסוי 2
1	25233814	49995000	165476	231332
2	100314558	199990000	333446	502660
3	224483963	449985000	544822	788084
4	399425938	799980000	786010	1085316
5	622424559	1249975000	964873	1386164
6	900129135	1799970000	1163528	1696164
7	1230575228	2449965000	1398706	2010628
8	1605950916	3199960000	1559538	2330628
9	2025048605	4049955000	1954460	2650628
10	2508691634	4999950000	2092500	2972324

<u>החלפות למערך הפוך:</u> נניח שהמערך באורך n.

כל איבר במערך, כשנגיע אליו נצטרך לשים אותו במקום 0 במרעך, בגלל שהמערך הפוך, כלומר, נזיז האיבר שבמקום 1 למקום 0, אחר כך, האיבר שבמקום 2 למקום 0 וכו'..

לכן כמות ההחלפות היא: $1+2+3+\cdots+(n-1)=\frac{(n-1)\left(1+(n-1)\right)}{2}=\frac{n^2-n}{2}$

(סכום סדרה חשבונית)

וזה מתהווה במספר ההחלפות שקיבלנו בניסוי 2, לכן המדידות מתנהגות כצפוי.

<u>החלפות למערך אקראי:</u> מערך שמוסדר אקראי מהווה מצב "בינוני" של החלפות, כאשר מצב "הכי גרוע" הוא כאשר המערך הפוך (כלומר מה שהוסבר למעלה), ומצב "הכי טוב" הוא כאשר המערך ממוין.

ציפינו שמספר ההחלפות למערך אקראי יהיה קירוב לממוצע ההחלפות של המצב הכי גרוע והמצב הכי טוב.

נסמן H_i את מספר החילופים לאיבר מספר i, סיבוכיות הפעולה שווה ל $n+H=n+\sum_i Hi$ כאשר n מהווה סריקה לכל איברי המערך.

כדי לחשב ממוצע ההחלפות, החשב את התוחלת, מתקיים מלינאריות:

$$E(n+H)=n+E\left(\sum_i H_i\right)=n+\sum_i E(H_i)$$

כאשר תוחלת לכל i מהווה סכום האיברים שבמקום קטן מ-i אבל גדולים מהערך שנמצא בו.

כלומר:

$$E(H_i)=E\left(\sum_{j<i} f_{i,j}\right)=\sum_{j<i}\frac{1}{2}$$

סכום כל האפשריות של i,j הוא שווה לסיכוי לבחירת 2 דברים שונים מתוך n אפשריות, כלומר nC2, לכן מתקיים: $E(H_i) = \frac{1}{2} * \frac{n^2-n}{2} = \frac{n^2-n}{4}$ שזה שווה לחצי מההחלפות למערך הפוך, בהתאם לציפיות ולתוצאות בטבלה.

<u>עלות החיפוש בהכנסה הפוכה:</u> חסם עליון לאורך המסלול ביו האיבר המקסימלי למיקום ההכנסה החדש יהיה שווה ל - $2\log(H_i)$ כאשר H_i הוא מספר ההחלפות להכנסת האיבר מס' i. אם ביצענו H_i החלפות, לכן זה אומר שמהאיבר המקסימלי עד מקום ההכנסה ישנן בדיוק H_i איברים, לכן תת עץ שהם נמצאים בה היא מגובה $\log(H_i)$, והמסלול הכי ארוך בין שני איברים בתת עץ מגובה כזה הוא $2\log(H_i)$.

אזי, מתקיים שסכום עלויות החיפוש (אורך המסלולים לכל הכנסה) הוא:

$$\sum_{i=1}^{n-1} 2\log(H_i)$$

$$2\sum_{i=1}^{n-1}\log(i) \le 2n\log\left(\frac{\sum_{i=1}^{n-1} i}{n}\right) = 2n\log\left(\frac{n-1}{2}\right)$$

שהו חסם עליון הדוק ככל האפשר לעלות החיפוש ב- AVL עם הכנסה הפוכה וזה קרוב מאוד לתוצאות שקיבלנו בטבלה.

<u>עלות החיפוש בהכנסה אקראית:</u>

נמצא תוחלת לאורך המסלול באופן דומה לסעיף קודם:

$$E\left(2\sum_i \log(H_i)\right) \le 2\sum_i \log(E(H_i))$$

מצאני את $E(H_i)$ בסעיף קודם, לכן נציב ונקבל:

$$2\sum_i \log\big(E(H_i)\big) = 2\sum_i \log(\frac{1}{2}\binom{n}{2}) = 2n\log(\frac{n^2-n}{4})$$

זהו חסם עליון הדוק ככל האפשר לעלות החיפוש בהכנסה אקראית לעץ – AVL.

סעיף ב':

מצאנו בסעיף א' שעלות חיפוש היא לכל היותר $2n\log(\frac{n^2-n}{4})$. בפעולת insert לעץ AVL ראשית מבצעים חיפוש, אחר כך מוסיפים (O(1)), ואחרי ההוספה מבצעים איזון לעץ כדי שיחזור לענות על תנאי עץ AVL. הוכחנו בשיעור וגם בחלק התיעוד שעלות האיזון היא לא יותר מ- O(log n). לכן, סיבוכיות Insert היא:

$$O(2n\log\left(\frac{n^2-n}{4}\right) + O(\log n)) = O(nlog(n^2))$$

שאלה 2:

מספר סידורי	עלות join ממוצע בניסוי 1	עלות join מקסימלי בניסוי 1	עלות join ממוצע בניסוי 2	עלות join מקסימלי בניסוי 2
1	2.90	6	2.46	14
2	2.5	6	3.3	19
3	2.71	5	2.84	16
4	2.71	10	2.5	17
5	3	9	3.15	17
6	2	6	2.64	18
7	2.78	7	2.62	18
8	2.86	5	2.37	19
9	3	5	2.6	18
10	2.6	6	2.64	18

<u>עלות ממוצעת לבחירת מפתח אקראי:</u> נניח שמספר הצמתים בעץ הוא n, לכן, הסיכוי לבחור איבר מגובה k, כאשר מתקיים k <= log n הוא $\frac{2^{k-1}}{n}$, עלות פעולת split היא כגובה העץ, כמו שהוכחנו בשיעור, ובמקרה שלנו תת העץ הוא בגובה k, לכן אנחנו מבצעים k פעולות join, ומתקיים שעלות פעולת ה- join הממוצעת הוא קבוע (2 או 3), כלומר O(1). נחשב את התוחלת:

$$E(average\ join) = \sum_{k=1}^{\log n} \frac{2^{k-1}}{n} * c = \frac{c}{2n} \sum_{k=1}^{\log n} 2^k = \frac{c}{2n} * (2n-1) = \frac{c(2n-1)}{2n}$$

קיבלנו שהעלות קבועה ותלויה ב- c לכל גודל העץ, c הוא הערך בין (2,3) וזה קרוב לתוצאות שקיבלנו.

<u>עלות מקסימלית לבחירת מפתח אקראי:</u> כמו שהוכחנו בשיעור, סיבוכיות פעולת split תלויה במספר פעולות הjoin, בחירה אקראית לצומת היא יכולה להיות או המקרה הכי טוב, בלי join בכלל (כלומר split לשורש) או מקרה הכי גרוע, שעולה O(log n). לכן חסם עליון טריוויאלי (אבל לא הכי הדוק) הוא O(log n). נכון שזה קצת יותר גדול מהתוצאות שקיבלנו, אבל זה נכון כי ההסתברות למקרה הכי גרוע היא יותר נמוכה מההסתברות למקרה סביר יחסית, בגלל זה תוצאות העלות המקסימלית לבחירה אקראית הם איפשהו במחצית המקרה הכי גרוע.

<u>עלות ממוצעת לבחירת מפתח מקסימלי לתת עץ שמאלי:</u> האיבר שמבקשים ממנו לעשות split עליו הוא ה predecessor של השורש, והוא בהכרח אין לא בנים ימניים. לכן אם העץ הוא בגובה log n אזי מבציעים (log n)-1 פעולות join, בעלות קבועה לכל אחת, אלו הן פעולות ה- join של תתי עצים עם ערכים קטנים מהצומת שבחרנו (במקרה הכי גרוע זה שווה ל-2 כי ככה מוגדר עץ AVL). ובנוסף, מבציעים עוד פעולת join אחרונה שהיא בין תת עץ ימני והשורש, ובמקרה הכי גרוע היא שווה ל log n. סה"כ נבצע log n פעולות join, ונחשב את הממוצע שלהם:

$$average(join) = \frac{(\log n - 1) * 2 + \log n * 1}{\log n} = \frac{3\log n - 2}{\log n} = 3 - \frac{2}{\log n}$$

וזה מאוד קרוב לתוצאות שקיבלנו.

<u>עלות מקסימלית לבחירת מפתח מקסימלי לתת עץ שמאלי:</u> ה- join המקסימלי כמו שפרטנו למעלה הוא עולה log n והוא קורה כאשר עושים join לשורש עם תת עץ ימני, הוא ההפרש בין גובה האיבר הכי ימני והשורש, כלומר, $(\log n - 1) - (-1) = \log n$, ואפשר לראות את זה בתוצאות שלנו.