***username1 - ohadelhayani***

***id1 - 206843328***

***name1 - Ohad Libai***

***username2 - shahafkurman***

***id2 - 318513843***

***name2 - Shahaf Kurman***

***מסמך תיעוד הקוד***

המחלקות שמומשו הן:

1. ***AVLNode***
2. ***AVLTreeList***

נעבור על הפונקציות לפי הסדר שבו הן כתובות בקוד ונתאר אותן.

***המחלקה AVLNode :***

* **פונקציית ה-constructor :**

מקבלת ערך value ויוצרת אובייקט מטיפוס AVLNode . לאובייקט AVLNode 6 שדות: - **value** המכיל את הערך הרצוי.

-**size** המעיד על כמות הצמתים שנמצאים מתחת לצומת בעץ (כולל הצומת עצמו).

-**height** המעיד על גובהו של הצומת בעץ.

- **left** המשמש כפוינטר לבן השמאלי.

- **right** המשמש כפוינטר לבן הימני.

- **parent** המשמש כפוינטר לאבא של הצומת

* כעת נעבור לתאר את קבוצת הפונקציות הבסיסיות במחלקה זו. פונקציות אלה פונות לשדה של הצומת ומבצעות שאילתה או שינוי הערך בשדה שלו.

**פונקציות get** : מטרתן לשלוף את הערך של השדה בצומת. למשל, getParent מחזירה את ההורה של הצומת. פונקציה נוספת בקטגוריית השאילתות היא isRealNode אשר מספקת מענה לשאלה האם מדובר בצומת אמיתי בעץ או בצומת דמה שהוכנס לצורך נוחות המימוש.

**פונקציות set** : מבצעות את השינוי הרצוי בשדה של הצומת. למשל, setValue מבצעת השמה לשדה הערך של הצומת.

שני הסוגים רצים ב-O(1) .

למחלקה זו הוספנו פונקציות החיוניות לביצוע הדרישות, להלן פירוט.

* פונקציית ה-**successor** ופונקציית ה-**predecessor** : פונקציית סימטריות במהותן. הראשונה מחזירה את הצומת העוקב על-פי סדר האינדקסים ברשימה

והשנייה מחזירה את האיבר הקודם לצומת שעליו מופעלת הפונקציה. המימוש נעשה בדומה למתואר בהרצאות. פונקציות אלה מסייעות בביצוע פעולות של הכנסה ומחיקה, מציאת איבר ראשון ואחרון בעץ, חיפוש איברים ברשימה ועוד.

סיבוכיות הזמן שלהן במקרה הגרוע ביותר יהיה O(h) כאשר h מסמל את גובה העץ. כיוון שהעץ הוא AVL , על-פי הנלמד h=log(n) .

* **inOrderArray** : פונקציית עזר רקורסיבית שמטרתה להחזיר את הרשימה בסדר ממוין. הפונקציה למעשה עושה in-order על העץ. מופעלת כאשר קוראים לשירות listToArray . סיבוכיות הזמן שלה תהיה O(n) היות שהיא עוברת על כל האיברים בעץ.
* **maintainSizeHeight** : פונקציה שתפקידה לתחזק את שדות ה-size וה-height של הצמתים בעץ לאחר פעולות כמו הכנסה, מחיקה, איחוד ופיצול רשימות. הפונקציה מחזירה 1 אם נעשה שינוי בגובה של הצומת כתוצאה מפעולה כמו הכנסה או מחיקה בעץ, ו-0 אם השינוי לא השפיע על גובהו של הצומת. בכל קריאה לפונקציה, תיקונים, אם נדרשים, של הגובה והגודל עולים O(1) מאחר שנעשית בדיקה באמצעות המצביעים לבנים.
* **הפונקציות leftRotation ו- rightRotation**  : פונקציות האחראיות לגלגולים בעץ על-מנת לשמר את תכונת האיזון. פונקציית rotate בהתאם ל-balance factor של הצומת מנתבת לגלגול המתאים. מתודות אלו נעזרות גם כן ב-maintainSizeHeight. פונקציות אלה מבצעות שינויי פוינטרים. כמו כן פועלות בזמן קבוע.
* **פונקציית rotate** : פונקציית עזר המהווה רכיב באיזון העץ. לפונקציה זו קוראים במידה שהתברר שצריך לבצע גלגול בעץ. פונקציה זו פועלת גם כן בזמן קבוע.
* **fixTree** : הפונקציה המרכזית האחראית על התיקונים בעץ. פונקציה זו מקבלת את הצומת שממנה צריך להתחיל לתקן את העץ. אחראית על תחזוק השדות size ו-height ומשתמשת בפונקציות גלגול כדי לאזן את העץ. הפלט שלה אצור בצומת המכיל מידע על כמות השינויים שבוצעו בעץ (כפי שהוגדרו בתרגיל) וגם את השורש של העץ לאחר כל התיקונים. ב-worst case הפונקציה תתחיל לתקן מהעלה ותעלה עד לשורש ב-O(h) עבור h גובה העץ.
* **join** : המקבלת 3 צמתים ומבצעת איחוד של שני עצים. הצומת self יהיה הצומת המקשר בין שני עצים כאשר שני השורשים של העצים מועברים כארגומנטים. מבצעת שינויי פוינטרים תוך מציאת הפרש גבהים מתאים כפי שהוסבר בכיתה. בפונקציה זו עושים שימוש פעולות ה-split וה-concat . תחזיר לבסוף את השורש של העץ שמוזג, אך לא לפני שתבצע תיקונים באמצעות קריאה ל-fixTree . סה"כ תעבוד ב-O(log(n)) .

***המחלקה AVLTreeList :***

* **פונקציית הבנאי**:

לפונקציית הבנאי לא מעבירים ארגומנטים. קריאה לבנאי מייצרת רשימה ריקה. לכל יצירת אובייקט מאותחלים דיפולטית ל- 3 השדות הערך None .

3 השדות הם : - **root** המהווה מצביע לשורש של עץ.

-**firstElem** המחזיק את האיבר הראשון ברשימה על-פי סדר ממוין.

-**LastElem** המחזיק את האיבר האחרון ברשימה על-פי סדר ממוין.

למחלקה זו הוספנו את המתודות *treeSelect*, *UpdateFirst/Last* .

* **tree-select** : פונקציה זו מספקת את הצומת במיקום ה-i ברשימה. אופן המימוש דומה למתואר בכיתה. כשצריך להשיג מצביע לצומת בעץ קוראים לה. למשל ב-retrieve נעשה שימוש בפונקציה זו. גם בתחילת הקוד להכנסה ולמחיקה. פונקציה רקורסיבית המשתמשת בעיקר בשדה ה-size של הצמתים ורצה ב-O(logn) .
* **UpdateFirst, UpdateLast** : פונקציות תחזוקה לשדות firstElem, lastElem של העץ. נעשה בהן שימוש בעת הכנסה, פיצול ולמעשה בכל פעם שנעשה שינוי בעץ שעלותו היא log(n) וזאת על-מנת לא לפגוע בסיבוכיות של פעולות אחרות הלוקחות זמן קבוע. פונקציות אלה מאפשרות להחזיק מצביעים לאיבר הראשון ולאיבר האחרון ברשימה ובכך מאפשרות אליהם גישה בזמן קבוע. בכך פונקציות כמו first ו-last וכן שימושים פנימיים כמו ב-concat נעשים ב-O(1) . עלות הפעולה שלהן תהיה O(logn) .
* **empty** : מחזירה ערך בוליאני באשר לשאלה האם הרשימה המוצגת לא מכילה איברים כלל. O(1)
* **retrieve:**  מחזירה את את ערכו של האיבר שנמצא במיקום ה-i ברשימה. קוראת ל-tree-select הרקורסיבית ומחזירה את ערכו של הצומת שקיבלה מ- treeSelect . סיבוכיות הזמן היא כיעילות הפעולה של treeSelect - O(logn) .
* **insert :** מבצעת הכנסה של איבר בעל ערך val במיקום ה-i . מחזירה את כמות השינויים שהתחוללו בעקבות ההכנסה. נחפש את המיקום שבו רוצים לבצע הכנסה באמצעות tree-select , נתפור פוינטרים ונבצע תיקון על העץ באמצעות קריאה ל- fixTree . פונקציה זו יורדת ב-log(n) , משנה פוינטרים ב-O(1) ועולה חזרה למעלה כדי לעדכן את שדות ה-size וה-height ומבצעת גלגולים מתוך fixTree , שבתורה קוראת ל-rotate , העושה שימוש ב-leftRotate וב-rightRotate וכמובן ב-maintainSizeHeight ב-O(logn) .
* **delete :** שתפקידה למחוק את האיבר במיקום ה-i מהעץ. תחילה נמצא את המיקום למשל על ידי קריאה ל-treeSelect . נבצע תפירה מתאימה של פוינטרים ונעלה חזרה לשורש כדי לבצע תיקונים ולשמור על האיזון בעץ. בנוסף נעדכן את השדות של האיבר הראשון והאחרון ברשימה. הפונקציה תחזיר לבסוף את מספר השינויים שהתחוללו כתוצאה מהמחיקה. סה"כ O(logn) .
* **first, last :** קריאה להן מחזירה את ערך האיבר במקום הראשון ובמקום האחרון ברשימה. כיוון שאנו מתחזקים שדה firstElem, lastElem נקבל שסיבוכיות הזמן היא קבועה- נשלוף את האיברים באמצעות המצביע.
* **listToArray :** מציגה בפנינו את הרשימה בצורת מערך. נתבקשה הצגה של המערך בצורת list של פייתון ולכן המתודה תחזיר זאת. כלומר נקרא ל-inOrderArray עם רשימה ריקה ונבקש ממנה למלא אותה בסדר ממוין מתוך העץ. העלות כאן היא כמובן O(n) .
* **length :** מחזירה את גודל הרשימה ב-O(1) בזכות שדה ה-size
* **split**: מקבלת אינדקס ומפצלת את הרשימה באינדקס זה. תחזיר רשימה של פייתון ובו באינדקס 1 ישב האיבר שבו מפצלים, באינדקס 0 העץ בעל האינדקסים הקטנים מהאינדקס של האיבר שפוצל, באינדקס 2 העץ בעל האינדקסים הגדולים מהאיבר שבו מפצלים. משתמשת כמובן ב-join ועל-פי מצב הפוינטרים מעבירה את שורשי העצים T1 ו-T2 אליה. בסוף התהליך נעדכן את האיבר הראשון והאחרון של כל רשימה ב-updateFirst, updateLast. ניתוח נאיבי ולא מעמיק יאמר שהעלות שלה על-ידי ביצוע פעולות של join יעלה log^2(n) אך למעשה היא פועלת ב-log(n) (הסבר ניתן בהרצאה).
* **concat** : מחברת שתי רשימות לכדי רשימה אחת. תחילה מסירים את האיבר במקום האחרון ברשימה, אליו יש גישה ב-O(1) . לאחר מכן מבצעים פעולת join אחת ומגישים את הפרש הגבהים של העצים טרם המיזוג. סיבוכיות תלויה ב-join הפועלת ב-log(h) כאשר h מייצג את גובה העץ לאחר המיזוג.
* **search** : שירות זה מחזיר את האינדקס בו נמצא הערך val . נתחיל באיבר הראשון ברשימה ונתקדם in-order על העץ באמצעות פעולת ה-successor שתופעל כל פעם על הצומת הנוכחי. worst case יהיה להגיע לאיבר האחרון ואז O(n) .
* **getRoot** : מחזירה את השורש של העץ. O(1)

***הסוף***

**שאלת חקר 1:**

|  |  |  |  |
| --- | --- | --- | --- |
| **מספר סידורי i** | **ניסוי 1 - הכנסות** | **ניסוי 2 - מחיקות** | **ניסוי 3 - הכנסות ומחיקות לסירוגין** |
| **1** | 4,948 | 1,921 | 1,765 |
| **2** | 9,892 | 3,774 | 3,343 |
| **3** | 19,851 | 7,574 | 6,649 |
| **4** | 39,718 | 15,271 | 13,107 |
| **5** | 79,067 | 30,622 | 26,489 |
| **6** | 158,940 | 61,141 | 53,011 |
| **7** | 317,579 | 122,526 | 106,802 |
| **8** | 635,743 | 245,260 | 213,547 |
| **9** | 1,272,110 | 489,898 | 424,587 |
| **10** | 2,542,742 | 978,256 | 853,427 |

**עמודה מספר 1 - הכנסות:** הביטוי האסימפוטטי התואם הוא f(n) = 2.5n . בכל פעם שהגדלנו את מספר האיברים בעץ פי 2, מספר פעולות האיזון שנדרשו גדל פי 2. כלומר מספר פעולות האיזון במקרה של הכנסות רנדומיות הוא לינארי בגודל האיברים בעץ. באקסל למשל נראה קוו מגמה ישר.

**עמודה מספר 2 – מחיקות:** מתנהג בדומה (ולא במפתיע) לפעולות של הכנסה: הגדלת מספר האיברים בעץ מגדילה את מספר השינויים שהתחוללו בעץ (גבהים ופעולות רוטציה) באופן לינארי. הביטוי האסימפוטטי התואם הוא f(n) = n . שוב, באקסל ניווכח בקוו מגמה ישר.

**עמודה מספר 3 – הכנסות ומחיקות לסירוגין:** מספר זה גדל גם כן באופן לינארי כתלות בגודל הקלט. הביטוי המתאים הינו f(n) = 1.75n , וכפועל יוצא של הכנסה-מחיקה לסירוגין מהווה מעין ממוצע של שתיהן.

**שאלת חקר 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 1.44 | 5 | 1.5 | 12 |
| 2 | 1.45 | 8 | 1.63 | 13 |
| 3 | 1.75 | 8 | 1.66 | 14 |
| 4 | 1.69 | 4 | 1.46 | 16 |
| 5 | 1.46 | 3 | 1.375 | 17 |
| 6 | 1.53 | 5 | 1.53 | 18 |
| 7 | 1.33 | 3 | 1.73 | 19 |
| 8 | 1.36 | 5 | 1.44 | 20 |
| 9 | 1.61 | 8 | 1.73 | 22 |
| 10 | 1.57 | 5 | 1.61 | 22 |

**ניתוח join ממוצע לשני התרחישים:**

באופן כללי נפתח בכך שככל שהעץ יותר מאוזן כך עלות ה-join פוחתת. בשל תכונת ה-AVL , הפרש הגובה של שני צמתים המהווים שורשים של תתי עצים (בן ימני ובן שמאלי) הוא לכל היותר 2 , על כן ה-join הממוצע אמור להיות חסום מלמעלה על-ידי 2 + h/numsOfJoins , כאשר h מסמל את גובה העץ, numsOfJoins הוא מספר האיחודים. כל עוד לא נוצרו עצי פיבונאצ'י בכל תת עץ, שזהו מקרה מאוד מיוחד שבו הפרשי הגבהים בין הבנים של כל צומת בעץ הוא הגדול ביותר, החסם על join ממוצע אמור להיות 2. מהתוצאות עולה שהעלות נעה בטווח המצופה שבין 1 ל-2 (עקרונית אפשר שיהיה גם פחות מ-1, ואז העץ ייחשב אפילו מאוזן יותר, עם פחות הבדלים בהפרשי הגבהים). פעולת ה-join יורדת למטה בגובה של תת עץ גבוה יותר עד אשר מגיעה לגובה זהה (עד כדי 1+ או 1-) לתת העץ השני. אם למשל אנו במקרה של עלייה מהאינדקס הקבוע הרי שעבור מיזוג על העץ הגדול נקבל בכל פעם מיזוג מינימלי – הפרשי גבהים של 2 לכל היותר - והדבר ימשוך את הממוצע למטה, ובעלייה האחרונה לשורש יהיה מקסימלי – גובה העץ- ויעלה את הממוצע. תכונת הגבהים המאוזנים נשמרת לכל אורכו ורוחבו של העץ, ומשום כך נטען שקיים חסם על סכום הפרשיהגבהים הכולל של שני סוגי המיזוגים. עבור אינדקס שנבחר (רנדומית) נקבל סכומים שונים של הפרשי מיזוג שינועו בטווח שבין 0 לכמות המיזוגים המקסימלית. על כן בנוסף הממוצע של התרחיש הראשון יכול להיות גבוה, נמוך או שווה לממוצע של התרחיש השני. כי גם אם עלינו במעלה העץ מספר רצוף של פעמים בכיוון מסוים ובכך הפרשי הגבהים היו קטנים יחסית, מתישהו נצטרך לפנות בכיוון אחר על העץ, ואז "נשלם" בהפרש גבהים גדול יותר. אם נמצע על פני כמות המיזוגים (או אם נרצה כמות העליות לשורש) נקבל ממוצעים קרובים. ואכן ניתן להיווכח בקרבה שבין הממוצעים בשני המקרים, עד כדי כך שב- i=6 הממוצע היה זהה.

**join מקסימלי:**

נפתח בכך שעלות ה-join המקסימלי תתקבל כאשר הגענו לשורש- שם נחבר בינו ובין הגובה של העלים בעץ, כלומר עלות כגובה העץ. מבחינה תיאורטית עלינו לראות עלייה מגמתית באשר ל-join המקסימלי עם בחירת ה-predecessor של השורש. כיוון שמגדילים בכל פעם את גודל העץ גדל גם גובהו ולכן נצפה לעלות join גדולה יותר. מכיוון שמובן לנו שה-join המקסימלי מתקבל מהגובה של תת העץ הימני ועוד 1 נשער שמספר זה יהיה לוגריתמי במספר האיברים שהוכנסו לעץ (לפי בסיס ɸ בהוכחה על החסם של הגובה בעץ AVL) . הגובה של עץ הנוצר מסדרה של הכנסות אקראיות וגם נעשה עליו איזון ישאף ל- log(n) , ואכן ניתן לראות בטבלה שכאשר הגדלנו את הקלט פי 2 (שזהו בערך בסיס הלוגריתם) העץ נהיה עמוק יותר ברמה אחת. ובמילים אחרות עלות ה-join המקסימלי גדלה ב-1. בחירת האינדקס במיקום זה לא נעשתה בכדי, אולם יכולנו לבחור באופן שקול לחלוטין את ה-successor של השורש ולקבל תוצאות זהות.

**שאלת חקר 3**

**מספר פעולות האיזון בממוצע**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| מספר סידורי i | עץ AVL -הכנסה להתחלה | עץ ללא מנגנון איזון- הכנסה להתחלה | עץ AVL- סדרה מאוזנת | עץ ללא מנגנון איזון- סדרה מאוזנת | עץAVL - סדרה אקראית | עץ ללא מנגנון איזון- סדרה אקראית |
| 1 | 2.974 | 499.5 | 0.991 | 0.991 | 2.511 | 2.484 |
| 2 | 2.986 | 999.5 | 0.995 | 0.995 | 2.4635 | 2.438 |
| 3 | 2.98 | 1499.5 | 0.996 | 0.996 | 2.47 | 2.4373 |
| 4 | 2.9925 | 1999.5 | 0.997 | 0.997 | 2.49 | 2.421 |
| 5 | 2.9938 | 2499.5 | 0.9984 | 0.9984 | 2.46 | 2.4432 |
| 6 | 2.9945 | 2999.5 | 0.998 | 0.998 | 2.45 | 2.416 |
| 7 | 2.995 | 3499.5 | 0.998 | 0.998 | 2.45 | 2.49 |
| 8 | 2.996 | 3999.5 | 0.9985 | 0.9985 | 2.48 | 2.4926 |
| 9 | 2.99 | 4499.5 | 0.999 | 0.999 | 2.47 | 2.406 |
| 10 | 2.99 | 4999.5 | 0.9991 | 0.9991 | 2.49 | 2.4593 |

**סדרה אקראית:**

אנו מניחים שבהכנסות אקראיות מספר פעולות האיזון הדרושות פוחת, וזאת משום שהכנסות אקראיות בד"כ לא מביאות לשינויי גבהים משמעותיים ולכן לא נדרש לגלגולים או לעדכוני גבהים. בנוסף, ההכנסה נשארת אקראית לכל אורך הבדיקה ולכן בממוצע פעולות האיזון חיות באותו טווח בשני העצים. ציפייה זו אכן מתיישבת עם התוצאות: מצד אחד מכיוון שעץ AVL מאזן את עצמו, מספר פעולות האיזון לעומת עץ רגיל הינו קטן יותר. מצד שני מכך שההכנסה היא אקראית גם לעץ רגיל לא נדרשות יחסית הרבה פעולות איזון, ובכל מקרה בשני העצים אין תלות בגודל העץ על פעולות האיזון אלא רק על גובהו שחסום לוגריתמית על-פי משפט מההרצאות.

**סדרה מאוזנת:**

נרצה למלא את העץ כך שלכל צומת יהיו שני בנים, ובנוסף למלא את כל הרמות בעץ. במצב זה נצפה שהההכנסה הראשונה לצומת שעדיין אין לו בנים תשפיע על כל האבות שלו ואילו בהכנסה השנייה מספר העדכונים לא ישתנה כלל. מובן שעלינו לצפות שבממוצע מספר האיזונים לא אמור להשתנות בין הגדלים השונים בעץ, שכן ההכנסה נעשית תמיד באותו אופן ובאופן שמצריך הכי פחות תחזוק. הדבר זהה לחלוטין בעץ AVL שבהכנסה זו מנגנון האיזון שלו מעוקר ומשולל פעילות. בסדרה כזו של הכנסות עץ AVL ועץ BST רגיל זהים במובן של פעולות איזון.

**סדרה של הכנסות להתחלה:**

מצפים למספר פעולות האיזון ב-AVL הרב ביותר מבין השלושה. באופן הזה נעשים בכל הכנסה שנייה לכל היותר פעולות גלגול. בעץ רגיל הדבר יתבטא ביצירת שרוך כך שבכל פעם שמכניסים מעדכנים את כל הצמתים עד לשורש. באופן לא מפתיע, מספר פעולות האיזון בעץ רגיל במקרה זה היא הגבוהה ביותר וגם קבועה וידועה מראש. כשמדובר בעץ רגיל, הכנסה זו יוצרת עץ כל-כך לא מאוזן עד כדי שינוי סדרי גודל משמעותיים לעומת ההכנסות האחרות.

**עומק הצומת המוכנס בממוצע**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| מספר סידורי i | עץ AVL -הכנסה להתחלה | עץ ללא מנגנון איזון- הכנסה להתחלה | עץ AVL- סדרה מאוזנת | עץ ללא מנגנון איזון- סדרה מאוזנת | עץAVL - סדרה אקראית | עץ ללא מנגנון איזון- סדרה אקראית |
| 1 | 9.967 | 499.5 | 7.978 | 7.978 | 9.192 | 10.58 |
| 2 | 10.971 | 999.5 | 8.977 | 8.977 | 10.1445 | 12.482 |
| 3 | 11.631 | 1499.5 | 9.63 | 9.63 | 10.846 | 14.44 |
| 4 | 11.973 | 1999.5 | 9.9765 | 9.9765 | 11.2327 | 13.19 |
| 5 | 12.359 | 2499.5 | 10.362 | 10.362 | 11.5012 | 13.87 |
| 6 | 12.632 | 2999.5 | 10.635 | 10.635 | 11.791 | 14.8825 |
| 7 | 12.828 | 3499.5 | 10.83 | 10.83 | 12.023 | 14.5 |
| 8 | 12.974 | 3999.5 | 10.976 | 10.976 | 12.25 | 15.077 |
| 9 | 13.178 | 4499.5 | 11.179 | 11.179 | 12.45 | 15.96 |
| 10 | 13.36 | 4999.5 | 11.36 | 11.36 | 12.55 | 15.6545 |

**סדרה אקראית של הכנסות:**

הכנסה באופן אקראי מביאה לכך שבתוחלת מספר ההשוואות שיבוצעו מהשורש ועד העלים בעץ ללא מנגנון איזון תהיה לוגריתמית, כפי שזה המצב בעץ עם מנגנון איזון. זאת על-פי משפט שנלמד בהרצאות. ואמנם, ניתן לראות מסדרת הניסויים שבממוצע העומק של צומת חדש בעץ בשני סוגי העצים הוא עם ערכים קרובים, ולכאורה לעץ AVL אין יתרון משמעותי על פני עץ BST רגיל. שני העצים מתנהגים באופן דומה עם שינויים קלים כשההכנסה אקראית, כמצופה.

**סדרה מאוזנת:**

כנאמר קודם- הכנסה דטרמיניסטית זו מכריחה עץ מאוזן. בכך אין שוני מבחינת העומקים של הצמתים החדשים שנוצרים מאחר שגם לא נעשים שום גלגולים בעץ וגם כי ממלאים את כל הרמות בעץ in-order . צפינו שהכנסה זו במהותה תיצור עץ כך שהצמתים המוכנסים לתוכו בעלי עומק מינימלי בכל פעם.

**סדרה של הכנסות להתחלה:**

סדרה זו מניבה שרוך בעץ BST רגיל. הדבר גורם לכך שהעומק של כל צומת חדש יהיה מקסימלי. מספר זה יהיה זהה למספר העדכונים על הגבהים. בניגוד לכך, עץ AVL יגלגל את הצמתים כל הזמן ימינה כדי להתאזן וכך למעשה בכל הכנסה העומק של צומת חדש יהיה תלוי פחות בסוג ההכנסה, שבמקרה הזה תהיה כל הזמן למקום ה-0 ברשימה. נציין שכצפוי מהכנסה דטרמיניסטית לתחילת הרשימה יווצרו לנו עצי פיבונאצ'י שיבטאו את המספר הרב ביותר של criminals-AVL.