אוניברסיטת תל אביב

סמסטר ב' תשפ"ג

**מבני נתונים - פרויקט מספר 1 - עץ דרגות**

ליאור צמח 212258990

אלה בר 207768987

# שאלה 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **מספר סידורי i** | **מספר חילופים במערך ממוין-הפוך** | **עלות מיון AVL עבור מערך ממוין-הפוך** | **מספר חילופים במערך מסודר אקראית** | **עלות מיון AVL עבור מערך מסודר אקראי** | **מספר החילופים במערך כמעט ממוין** | **עלות מיון AVL עבור מערך כמעט ממוין** |
| 1 | 4498500 | 61824 | 2247182 | 54514 | 448500 | 48967 |
| 2 | 17997000 | 135655 | 8927405 | 123720 | 897000 | 105326 |
| 3 | 71994000 | 295318 | 36002268 | 264774 | 1794000 | 218004 |
| 4 | 289788000 | 638645 | 143000428 | 589300 | 3588000 | 443483 |
| 5 | 1151976000 | 1373300 | 574544650 | 1258040 | 7176000 | 894362 |

אופן מימוש חישוב החילופים:

לשם מציאת מספר החילופים במערך עבדנו לפי האלגוריתם הבא:

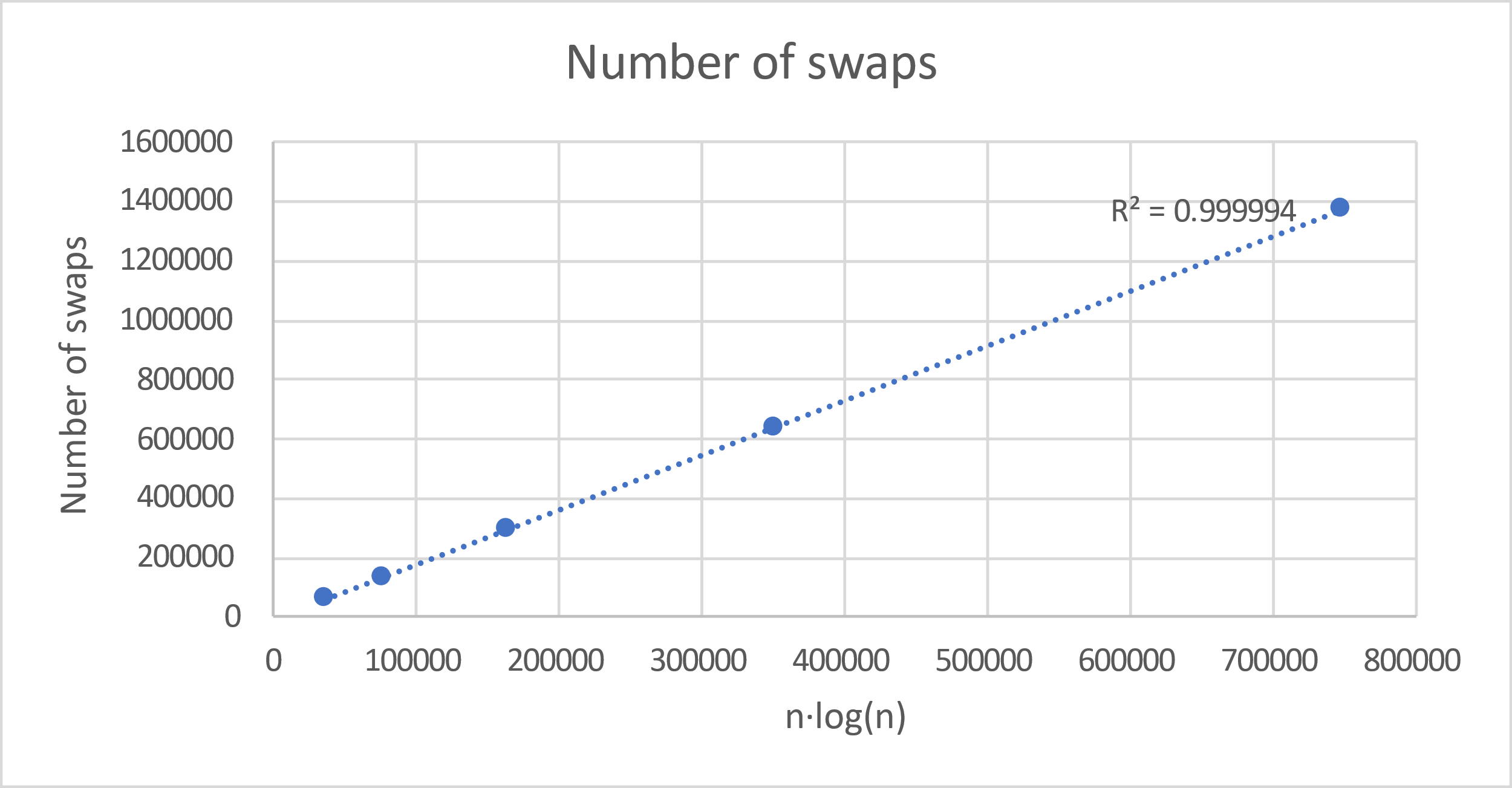
עבור כל צומת שהכנסנו לעץ בדקנו את ערך ה-(value-rank) של הצומת הנ"ל. סכמנו ערך זה על פני כל הצמתים שהוכנסו לעץ. הסכום שהתקבל הוא מספר אי הסדרים שבמערך.

1. ניתוח תיאורטי של מספר החילופים ועלות החיפושים של ה AVL במקרה של מערך ממוין-הפוך:

מספר החילופים במערך ממוין הפוך נתון ע"י הנוסחה: . זאת מכיוון שאם נספור את החילופים ע"פ כמה חילופים יוצר האיבר הגדול בזוג, נקבל שהאיבר n יוצר n-1 חילופים, האיבר n-1 יוצר n-2 חילופים, וכן הלאה עד האיבר 2 שיוצר חילוף 1, והאיבר 1 שיוצר אפס חילופים עקב היותו האחרון והכי קטן. לכן -  *כאמור לעיל.*

נתאר חסם הדוק עבור עלות החיפושים: כפי שלמדנו בכיתה עלות חיפוש של צומת בעץ בעל n צמתים היא מסיבוכיות ולכן נסכום על פני כל הצמתים ונקבל:

(זאת מכיוון שבשלב הi- אכן קיימים i איברים בעץ ולכן חיפוש מיקום הכנסת האיבר ה-i לוקח זמן).



נביט בגרף המצורף לעיל. הגרף מתאר את עלות מיון עץ ה- AVL עבור מערך ממוין - הפוך כתלות בפונקציה nlog(n).

קיבלנו גרף לינארי (ישר) מה שמעיד על כך שעלות מיון זה היא אכן מסיבוכיות כפי שציפינו.

כלומר, ישנה התאמה בין הערכים בטבלה מסעיף א' והניתוח מסעיף ב'.

בנוסף לכך, מדד ה- הוא 0.999994 מה שמעיד על איכות קירוב גבוהה מאוד.

# שאלה 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי i | עלות join **ממוצע**  עבור split **אקראי** | עלות join **מקסימלי** עבור split **אקראי** | עלות join **ממוצע** עבור split של **האיבר** מקסימלי בתת העץ השמאלי | עלות join **מקסימלי** עבור split של **איבר** מקסימלי בתת העץ השמאלי |
| 1 | 3.25 | 7 | 2.6 | 13 |
| 2 | 3.0 | 7 | 3.0 | 15 |
| 3 | 3.0 | 5 | 2.19666666666665 | 17 |
| 4 | 2.615384 | 5 | 3.076923 | 18 |
| 5 | 2.6 | 7 | 2.857142= | 18 |
| 6 | 2.428571= | 5 | 3.07142857 | 20 |
| 7 | 2.307692 | 6 | 3.06666 | 21 |
| 8 | 2.555= | 6 | 2.473684 | 22 |
| 9 | 2.105263 | 7 | 2.578947 | 23 |
| 10 | 22.555= | 6 | 2.684210 | 25 |

1. ניתוח תיאורטי של העלות של join ממוצע לשני הניסויים –

נסתכל על ה"פוטנציאל" של צד ימין ושמאל בהתאמה: בעת ביצוע join ימני או שמאלי, הפוטנציאל המתאים יהווה חסם עליון להפרש הגבהים בין העצים שיש לאחד. בכל צעד (טיפוס מצומת לאביו), סכום הפוטנציאלים יכול לגדול לכל היותר ב4 (הסבר מטה ע"י ניתוח כל אחד מהמקרים הגרועים האפשריים בתמונות). כאשר מתבצע join, העבודה הדרושה תהיה הפוטנציאל המתאים ועוד 1, ולאחר מכן הפוטנציאל מתאפס. לכן, סך העבודה שתתבצע היא כמספר פעולות האיחוד, ועוד הפוטנציאל בעת ביצוע כל פעולת איחוד. מספר פעולות האיחוד הוא כעומק הצומת, וכמוסבר מטה, סכום הפוטנציאלים הוא כ-4 כפול עומק הצומת. לכן, העלות הממוצעת חסומה ע"י 4.5 (גדילת הפוטנציאל בכל צעד, ועוד חצי כי רק צד אחד עובר איחוד). כלומר החסם האסימפטוטי לjoin ממוצע הוא O(1) – לא תלוי ב-n. זה גם מוכיח כי סיבוכיות split היא כעומק הצומת הנבחר.

למען שלמות ההסבר, נציג הסבר אלטרנטיבי: נשתמש ללא הוכחה בעובדה שעלות פיצול היא כעומק הצומת – נסמנו d. כיוון שמתבצעות בדיוק d פעולות איחוד, הסיבוכיות הממוצעת שלהן חייבת להיות קבועה. אם הסיבוכיות הממוצעת הייתה , אזי סיבוכיות הפיצול הייתה בסתירה.

*ניתוח זה מתאים עם הממצאים, כיוון שקל לראות שהם לא מושפעים מ-*i *וקרובים ל-2.5 (הממוצע בין עלות איחוד עצים בהפרש גובה 1, לעלות איחוד עצים בהפרש גובה 2. נדמה שאלו המקרים הנפוצים ביותר.)*

1. *ניתוח תיאורטי של העלות של* join *מקסימלי:*

*עבור צומת אקראי, עלות איחוד מקסימלית תתקבל במקרה הבא: במהלך הטיפוס, טיפסנו כמה פעמים לאותו הכיוון (ימינה או שמאלה), ואז החלפנו. זאת מכיוון שבעת החלפת הכיוון, הפוטנציאל של הצד אליו לא טיפסנו גבוה, ולכן פעולת האיחוד תהיה יקרה. הגרלת צומת אקראי בעץ שקולה להגרלת מסלול אקראי מהשורש אליו, אשר שקולה להגרלת רצף בינארי באורך כלשהו (בממוצע אורך זה יהיה העומק הממוצע של צומת בעץ, שהוא* ). לכן עלות האיחוד היקר ביותר תהיה כאורך הממוצע *של רצף (אחדות או אפסים) מקסימלי, אשר הוא בעצמו ב -של לוג אורך המחרוזת (עובדה ידועה, וידאנו אמפירית). כלומר הערך שנוכל לצפות לו הוא בערך , ולכן ערך זה לא מושפע ע"י* n *בטווחים שלקחנו.*

*עבור הצומת המקסימלי בתת העץ השמאלי, קל לראות שה*join *הכי יקר יתרחש בין הבן הימני (הוירטואלי) של הצומת ממנו התחלנו, לבין תת-העץ הימני של השורש (ביחד עם השורש בתפקיד הצומת המגשר). גובה הבן הוירטואלי מוגדר להיות 1-, וגובה תת-העץ הימני הוא ב*. לכן נקבל שפעולת האיחוד הכי יקרה תעלה עבודה.

ניתוח זה מתיישב עם הממצאים, כיוון שניתן לראות שהם עולים בצורה בערך לינארית ביחס ל-i, כאשר כלומר .

AVLTree

שדות: שורש העץ, מצביע לצומת המקסימלי

תיאור הפונקציות במחלקה AVLTree (אופן פעולתן וסיבוכיות זמן הריצה):

|  |  |
| --- | --- |
| **פעולה** | **תיאור** |
| search(self,key) | הפונקציה מבצעת חיפוש בינארי לאיבר בעל מפתח key. היא מחזירה מצביע לצומת המתאים אם קיים, אחרת מחזירה None.  סיבוכיות זמן: O(logn) |
| rotate\_left(self,node) | הפונקציה מבצעת רוטציה לצד שמאל כפי שנלמד בהרצאה  סיבוכיות זמן: O(1) |
| rotate\_right(self,B) | הפונקציה מבצעת רוטציה לימין כפי שנלמד בהרצאה  סיבוכיות זמן: O(1) |
| rotate\_leftright(self, node) | הפונקציה מבצעת רוטציה 'שמאל-ימין' כפי שנלמד בכיתה  סיבוכיות זמן: O(1) |
| rotate\_rightleft(self, n) | הפונקציה מבצעת רוטציה 'ימין-שמאל' כפי שנלמד בכיתה  סיבוכיות זמן: O(1) |
| perform\_rotation(self, node) | הפונקציה בודקת האם דרוש גלגול לצומת node הנתון (ע"פ התנאים המתוארים באלגוריתם ההכנסה), וקוראת לאחת מ-4 הפונקציות לעיל בהתאם.  סיבוכיות זמן: O(1) |
| perform\_rotation\_delete(self, node) | בדיוק כנ"ל, רק שהפונקציה משתמשת בתנאים המתוארים באלגוריתם המחיקה.  סיבוכיות זמן: O(1) |
| create\_node\_bst(self, key, val, cur) | הפונקציה מתחילה בצומת cur, מבצעת חיפוש בינארי בעץ עד למציאת מקום הכנסת המפתח val, והופכת אותו מצומת וירטואלי לאמיתי.  סיבוכיות זמן: במקרה הגרוע O(log n), ובמדויק O(height of cur) |
| find\_start\_fingertree(self, key) | הפונקציה מתחילה בצומת המקסימלי בעץ, ומחפשת את הצומת לאורך המסלול הימני מהשורש, שהוא האב הקדמון המשותף המינימלי של המקסימום ושל מיקום ההכנסה של צומת בעל מפתח key. פונקציה זאת מוצאת את cur הנשלח לcreate\_node\_bst כמתואר לעיל.  סיבוכיות זמן: במקרה הגרוע O(log n),  ובמדויק O(height of returned value) |
| insert(self, key, val, use\_finger = True) | פונקציה זו מכניסה צומת בעל מפתח key וערך val לעץ. אם הדגל use\_finger הוא True, חיפוש מקום ההכנסה מבוצע ע"י שתי הפונקציות לעיל, אחרת חיפוש המיקום מבוצע ע"י create\_node\_bst המקבל בתור cur את שורש העץ. לאחר מכן הפונקציה מאזנת את הוריו של הצומת החדש ע"פ האלגוריתם הנלמד בכיתה.  סיבוכיות זמן: O(logn) |
| delete(self, node, balance=true) | הפונקציה מוחקת צומת מהעץ ומאזנת את הוריו, ע"פ האלגוריתם הנלמד בכיתה.  סיבוכיות זמן: O(logn) |
| successor(self, node) | הפונקציה מוצאת את הצומת בעל המפתח העוקב למפתח של הצומת node.  סיבוכיות זמן: O(logn) |
| predecessor(self, node) | הפונקציה מוצאת את הצומת בעל המפתח הקודם למפתח של הצומת node.  סיבוכיות זמן: O(logn) |
| avl\_to\_array(self) | הפונקציה מחזירה מערך ממוין (ע״פ המפתחות) של האיברים במילון כאשר כל איבר מיוצג ע״י זוג סדור של (key, value).  סיבוכיות זמן: O(n) |
| size(self) | הפונקציה מחזירה את כמות הצמתים בעץ.  סיבוכיות זמן: O(1) |
| split(self, node) | הפונקציה מפצלת את העץ self לשני עצים, אחד המכיל את כל הצמתים להם מפתחות הקטנים משל הצומת הנתון node, ואחד המכיל את כל הצמתים להם מפתחות גדולים משל הצומת הנתון. הפיצול מבוצע ע"פ האלגוריתם הנלמד בכיתה.  סיבוכיות זמן: O(logn) |
| join(self, tree, key, val, update\_maximum = True) | הפונקציה מאחדת את העץ הנתון tree לתוך העץ self (בהינתן שהמפתח המינימלי באחד קטן מהמקסימלי באחר), תוך שימוש בצומת חדש בעל מפתח key וערך val בתור צומת מגשר, בעזרת האלגוריתם הנלמד בכיתה.  החיפוש של מיקום המיזוג מתחיל משורש העץ הגבוה, והתיקונים מתחילים ממיקום המיזוג. לכן סיבוכיות הזמן היא O(height difference), הפרש הגבהים בין העצים. |
| rank(self, node) | הפונקציה מחזירה את דרגת הצומת הנתון בעץ, כלומר מספר הצמתים הקטנים ממנו. האלגוריתם מבוצע עם חיפוש בינארי בעץ, ולכן:  סיבוכיות זמן: במקרה הגרוע O(log n), ובמדויק O(depth of node) |
| select(self,i) | הפונקציה מוצאת את הצומת שדרגתו היא i, בעזרת חיפוש בינארי במורד העץ.  סיבוכיות זמן: במקרה הגרוע O(log n), ובמדויק O(depth of returned value) |
| get\_root(self) | הפונקציה מחזירה את שורש העץ.  סיבוכיות זמן: O(1) |

AVLNode

שדות: שורש העץ, מצביע לצומת המקסימלי

תיאור הפונקציות במחלקה AVLNode

(אופן פעולתן וסיבוכיות זמן הריצה):

|  |  |
| --- | --- |
| **פעולה** | **תיאור** |
| get\_key(self) | מחזיר את המפתח של הצומת, או None אם הצומת הוא וירטואלי.  סיבוכיות זמן: O(1) |
| get\_value(self) | מחזיר את הinfo של הצומת או None אם הצומת הוא וירטואלי.  סיבוכיות זמן: O(1) |
| get\_left(self) | מחזיר את הבן השמאלי של הצומת, או None אם אין כזה.  סיבוכיות זמן: O(1) |
| get\_right(self) | מחזיר את הבן הימני של הצומת, או None אם אין כזה.  סיבוכיות זמן: O(1) |
| get\_parent(self) | מחזיר את ההורה של הצומת, או None אם אין כזה.  סיבוכיות זמן: O(1) |
| get\_height(self) | מחזיר את גובה הצומת, 1- עבור צומת וירטואלי.  סיבוכיות זמן: O(1) |
| get\_size(self) | מחזיר את גודל תת-העץ של הצומת, 0 עבור צומת וירטואלי.  סיבוכיות זמן: O(1) |
| set\_key(self, key) | הפונקציה מעדכנת את המפתח של הצומת להיות key  סיבוכיות זמן: O(1) |
| set\_value(self, value) | הפונקציה מעדכנת את הערך של הצומת להיות value  סיבוכיות זמן: O(1) |
| set\_left(self, node) | הפונקציה מעדכנת את הבן השמאלי של הצומת להיות node  סיבוכיות זמן: O(1) |
| set\_right(self, node) | הפונקציה מעדכנת את הבן הימני של הצומת להיות node  סיבוכיות זמן: O(1) |
| set\_parent(self, node) | הפונקציה מעדכנת את ההורה של הצומת להיות node  סיבוכיות זמן: O(1) |
| update\_height(self) | הפונקציה מעדכנת את שדה הגובה של הצומת  סיבוכיות זמן: O(1) |
| set\_height(self, h) | הפונקציה משנה את שדה הגובה של הצומת הנוכחי להיות h  סיבוכיות זמן: O(1) |
| update\_size(self) | הפונקציה מעדכנת את שדה ה-size של הצומת  סיבוכיות זמן: O(1) |
| set\_size(self, s) | הפונקציה משנה את שדה הגובה של הצומת size להיות s  סיבוכיות זמן: O(1) |
| get\_BF(self) | הפונקציה מחזירה את ה-BF של הצומת כפי שהוגדר בכיתה  סיבוכיות זמן: O(1) |
| realize(self, key, value) |  |
| is\_real\_node(self) | מחזיר TRUE אם הצומת מייצג צומת אמיתי בעץ (קרי: צומת שאינו וירטואלי). |
| inorder(self, process\_func, index) | פונקציה זו מקבלת את הפונקציה process\_func(node, index) => None אשר מורצת על כל צומת בעץ, כך שindex הוא דרגת הצומת. תהליך זה קורה בעזרת טיול in-order בעץ, ושמירה על אינדקס נכון. |