<u>תרגיל בית 1 מבני נתונים</u>

id1: 322520255

name1: Itamar Ben Nun

username1: itamarbennun

id2: 316061787

name2: Tal Malka

username2: talmalka2

<u>חלק מעשי</u>

<u>מחלקת AVLNode</u>

מחלקה זו מייצגת צומת בעץ AVL.

<u>שדות מחלקה</u>

- <u>key :</u> המפתח של הצומת.
- <u>value :</u> הערך המשויך למפתח.
- <u>right :</u> מצביע לילד הימני.
- <u>left :</u> מצביע לילד הימני.
- <u>parent :</u> מצביע על ההורה של הצומת.
- <u>height :</u> גובה הצומת בעץ.

<u>פעולות מחלקה</u>

<u>__init__()</u>

יוצרת צומת חדש בעץ עם מפתח, ערך והפניות ראשוניות לילדים וירטואליים.

סיבוכיות: $O(1)$, הפעולות כולן הן השמות פשוטות.

<u>is_real_node()</u>

בודקת אם הצומת הוא "אמיתי" ולא צומת וירטואלי. צומת אמיתי הוא זה שמפתחו שונה מ-None. מחזירה true/false בהתאם.

סיבוכיות: $O(1)$, הקריאה מתבצעת בשדה יחיד בצומת.

<u>search_helper(k)</u>

מבצעת חיפוש רקורסיבי בעץ החל מהצומת הנוכחי, תוך השוואת מפתח הצומת למפתח המבוקש. אם המפתח שווה, הפונקציה מחזירה את הצומת. אם המפתח קטן, הפונקציה מחפשת בתת-העץ השמאלי. אם המפתח גדול, הפונקציה מחפשת בתת-העץ הימני. מחזירה את הצומת של הצומת עם המפתח, או בן וירטואלי בו היה אמור להיות המפתח. כמו כן מחזירה את אורך החיפוש כמספר הקשתות של מסלול החיפוש +1.

סיבוכיות: $O(\log n)$, בעץ AVL מאוזן העומק המרבי לוגריתמי ביחס למספר האיברים.

מחלקת AVLNode

מחלקה זו מיישמת עץ AVL.

שדות מחלקה

- root : מצביע לשורש העץ.
- min : מצביע לצומת עם המפתח המינימלי.
- max : מצביע לצומת עם המפתח המקסימלי.
- tree_size : מספר הצמתים האמיתיים בעץ.

פעולות מחלקה

__init__()

מאתחלת עץ ריק עם שורש וירטואלי, ללא צמתים אמיתיים, ועם מונה גודל השווה לאפס.

סיבוכיות : $O(1)$, הפעולות כולן הן השמות פשוטות.

search(k)

מבצעת חיפוש מפתח בעץ על ידי קריאה ל-search_helper מהשורש. אם המפתח נמצא, מחזירה את הערך, אם חוזר בן וירטואלי מחזירה None. מחזירה את אורך החיפוש כמספר הקשתות של מסלול החיפוש +1.

סיבוכיות : $O(\log n)$, מאחר שהחיפוש מבוצע בעץ מאוזן.

finger_search(k)

מבצעת חיפוש למפתח בעץ מהצומת המקסימלי על ידי קריאה ל-finger_search_helper. אם המפתח נמצא מחזירה מצביע לצומת, אם חוזר בן וירטואלי מחזירה None. מחזירה את אורך החיפוש כמספר הקשתות של מסלול החיפוש +1.

סיבוכיות : $O(\log n)$, משתמשת ב-finger_search_helper.

finger_search_helper(k)

מבצעת חיפוש למפתח בעץ מהצומת המקסימלי על ידי טיפוס במעלה העץ עד שמגיע לצומת עם מפתח קטן/שווה למפתח שמחפשת או עד השורש. משם מבצעת חיפוש רגיל. מחזירה את הצומת עם המפתח, או בן וירטואלי בו היה אמור להיות המפתח. כמו כן מחזירה את אורך החיפוש כמספר הקשתות של מסלול החיפוש +1.

סיבוכיות : $O(\log n)$, מאחר שהחיפוש מבוצע בעץ מאוזן ומוגבל לגובה העץ.

successor(x)

מוצאת את הצומת שאחריו במפתח (המפתח הקטן ביותר הגדול מהמפתח הנוכחי). אם לצומת יש תת-עץ ימני, המפתח הבא יהיה המפתח הקטן ביותר בתת-העץ הזה. אחרת, הפונקציה נעה למעלה בעץ עד למציאת צומת הורה מתאים.

סיבוכיות : $O(\log n)$, מאחר שהחיפוש מוגבל לגובה העץ.

insert(k, v)

מבצעת חיפוש למפתח k בעץ, החל מהשורש, ומוצאת את מקום ההכנסה המתאים (עלה וירטואלי); מעדכנת את המפתח ואת הערך, יוצרת שני בנים וירטואליים חדשים ומאזנת את העץ. מחזירה שלשה שמכילה את המצביעה לצומת שנוצר, מספר הקשתות במסלול ההכנסה ומספר פעולות ה-promote שנעשו במהלך האיזון.

סיבוכיות: $O(\log n)$, במקרה הגרוע, משתמשת ב-search_helper ו-insert_de_facto בנפרד, אשר פועלות כל אחת בסיבוכיות $O(\log n)$ כפי שמתואר.

finger_insert(k, v)

מבצעת חיפוש למפתח k בעץ, החל מהמקסימום, ומוצאת את מקום ההכנסה המתאים (עלה וירטואלי); מעדכנת את המפתח ואת הערך, יוצרת שני בנים וירטואליים חדשים ומאזנת את העץ. מחזירה שלשה שמכילה את המצביעה לצומת שנוצר, מספר הקשתות במסלול ההכנסה ומספר פעולות ה-promote שנעשו במהלך האיזון.

סיבוכיות: $O(\log n)$, במקרה הגרוע, משתמשת ב-finger_search_helper ו-insert_de_facto בנפרד, אשר פועלות כל אחת בסיבוכיות $O(\log n)$ כפי שמתואר.

rebalance(x)

מבצעת איזון לעץ החל מצומת מסוים, בהתאם למקרים כפי שנלמדו בהרצאה. סופרת את מספר הפעמים בהם נדרש לבצע promote ומחזירה זאת.

סיבוכיות: $O(\log n)$, מאחר שהאיזון עשוי לדרוש מעבר על כל הצמתים מהעלים עד השורש, מעבר שמוגבל בגובה העץ. כמו כן משתמשת בפונקציות rotate_left ו-rotate_right שפועלות בזמן קבוע כפי שמתואר.

rotate_left(x)

מבצעת רוטציה שמאלה סביב צומת ע״י שינוי מצביעים ועדכון גבהים כפי שנלמד בהרצאה.

סיבוכיות: $O(1)$, הפעולות כולן הן השמות פשוטות.

rotate_right(x)

מבצעת רוטציה ימינה סביב צומת ע״י שינוי מצביעים ועדכון גבהים כפי שנלמד בהרצאה.

סיבוכיות: $O(1)$, הפעולות כולן הן השמות פשוטות.

insert_de_facto(x, e, k, v)

מבצעת בפועל את ההכנסה לעץ באמצעות הצומת שהתקבלה מהחיפוש ומבצעת איזון.

סיבוכיות: $O(\log n)$, ההכנסה בפועל היא ע״י פעולות השמה פשוטות, ומשתמשת ב-rebalance שפועלת בזמן $O(\log n)$ כפי שתואר.

delete(x)

מסירה צומת מהעץ, על פי החלוקה למקרים של מחיקה מעץ חיפוש בינארי, ומאזנת אותו כלפי מעלה. מבצעת איזון בהתאם למקרים כפי שנלמדו בהרצאה, בדומה ל-rebalance.

סיבוכיות: $O(\log n)$, מאחר שהמעבר על צמתים באיזון מוגבלים לגובה העץ. כמו כן משתמשת בפונקציות rotate_left ו-rotate_right שפועלות בזמן קבוע כפי שמתואר.

join(t, k, v)

מחברת את העץ הנוכחי עם עץ נוסף על ידי הכנסת מפתח חיבור בין השניים. הפעולה כוללת איזון מחדש של העץ המאוחד.

סיבוכיות: $O(\log n)$, אולם חסם יותר הדוק הוא הפרש הגבהים, האיזון מעלה מוגבל בגובה העץ הגבוה מבין השניים.

split(x)

מבצעת פיצול לעץ לפי צומת נתון, ע״י יצירת שני עצים אשר אחד עם כל המפתחות הגדולים מהמפתח של הצומת שניתן, והשני עם כל המפתחות שקטנים ממנו. מחזירה את שני העצים.

סיבוכיות: $O(\log n)$, כפי שהוכח בהרצאה.

set_min()

מעדכנת בעץ את המצביע לצומת עם המפתח המינימלי ע״י הליכה בבנים השמאליים עד לתחתית.

סיבוכיות: $O(\log n)$, ההליכה מוגבלת בגובה העץ.

set_max()

מעדכנת בעץ את המצביע לצומת עם המפתח המקסימלי ע״י הליכה בבנים הימניים עד לתחתית.

סיבוכיות: $O(\log n)$, ההליכה מוגבלת בגובה העץ.

avl_to_array()

מחזירה מערך שמייצג in-order walk על העץ – המפתחות בסדר ממוין עולה.

סיבוכיות: $O(n)$, מעבר על כל צמתי העץ.

max_node()

מחזירה את הצומת עם המפתח המקסימלי.

סיבוכיות: $O(1)$, המצביע שמור בעץ כשדה.

size()

מחזירה את גודל העץ – מספר הצמתים.

סיבוכיות: $O(1)$, הגודל שמור בעץ כשדה.

get_root()

מחזירה את שורש העץ.

סיבוכיות: $O(1)$, המצביע שמור בעץ כשדה.

insert_root(k, v)

מקרה קצה של insert, כאשר העץ ריק.

סיבוכיות: $O(1)$, הפעולות כולן הן השמות פשוטות.

חלק ניסויי

.1

מס"ד	עלות איזון במערך ממוין	עלות איזון במערך ממוין-הפוך	עלות איזון במערך מסודר אקראית	עלות איזון במערך עם היפוכים סמוכים אקראיים
1.	430	430	383.05	423.95
2.	873	873	776.2	867.3
3.	1760	1760	1570.4	1738.35
4.	3535	3535	3162.1	3498.05
5.	7086	7086	6329.25	7016.8
6.	14189	14189	12653.7	14037.9
7.	28396	28396	25338.85	28056.95
8.	56811	56811	50719.1	56153.5
9.	113642	113642	101372.7	112336.65
10.	227305	227305	202890.7	224675.3

כאשר מבצעים הכנסה, אנחנו מבצעים תחילה חיפוש לצומת שבה צריך להכניס את המפתח, כלומר יש לנו מצביע. ראינו בהרצאה שעלות ההכנסה amortize לצומת עם מצביע היא $O(1)$, ולכן עבור סדרה של n הכנסות עלות האיזונים היא $O(n)$ (מתיישב עם תוצאות הניסוי – לכל היותר $2n$). ראינו שגלגול (רגיל או כפול) היא פעולה טרמינלית, ולכן העלות שלה היא $O(1)$. כלומר לא משפיעה על החסם העליון התיאורטי של איזונים כולל גלגולים ולכן $O(n)$.

.2

מס"ד	מספר היפוכים במערך ממוין	מספר היפוכים במערך ממוין-הפוך	מספר היפוכים במערך מסודר אקראית	מספר היפוכים במערך עם היפוכים סמוכים אקראיים
1.	0	24531	12237.55	110.9
2.	0	98346	49575.25	221.3
3.	0	393828	195797.9	444.6
4.	0	1576200	786370.55	888.1
5.	0	6306576	3156659.35	1781.3

.3

מס"ד	עלות חיפוש במערך ממוין	עלות חיפוש במערך ממוין-הפוך	עלות חיפוש במערך מסודר אקראית	עלות חיפוש במערך עם היפוכים סמוכים אקראיים
.1	222	2695	2215.15	287.65
.2	444	6273	5251.0	581.05
.3	888	14317	12371.15	1159.15
.4	1776	32181	28420.45	2330.3
.5	3552	71461	64050.55	4657.6
.6	7104	157125	142773.7	9313.95
.7	14208	342661	310491.3	18640.2
.8	28416	742149	682672.75	37316.45
.9	56832	1597957	1475418.7	74597.15
.10	113664	3423237	3167710.1	149222.45

.4

I. לכל i, d_i מסמן את מספר האיברים לפני האינדקס ה-i כך שלכל $j > i$ מתקיים $A[i] > A[j]$, כלומר הגדרת ההיפוך. ולכן סך ההיפוכים הכולל הוא סכום ההיפוכים לכל איבר.

II. ב-finger_search מתחילים מהאיבר המקסימלי, עד שמגיעים לשורש או לצומת עם מפתח קטן/שווה מהמפתח שאנו מחפשים, והתת-עץ של הצומת הזה מכיל את כל האיברים שגדולים מהמפתח ה-i, כלומר d_i איברים. הגובה של העץ הזה חסום ע"י $O(\log{(d_i)})$, ובהתאם גם עלות ההכנסה של המפתח ה-i. במקרה שבו $d_i = 0$ עלות ההכנסה היא $O(1)$, ולכן עבור סדרת הכנסות של n איברים, עלות ההכנסה היא:

$$\sum_{i=1}^{n} O(\max{(1, \log{(d_i)})}) = \sum_{i=1}^{n} O(\log(d_i + 2)) = O(\log(\prod_{i=1}^{n}(d_i + 2)))$$

III. לפי א"ש הממוצעים מתקיים:

$$\prod_{i=1}^{n}(d_i + 2) \leq \left(\frac{\sum_{i=1}^{n}(d_i + 2)}{n}\right)^n \Rightarrow \log\left(\prod_{i=1}^{n}(d_i + 2)\right) \leq \log\left(\frac{\sum_{i=1}^{n}(d_i + 2)}{n}\right)^n =$$

$$n\log\frac{\sum_{i=1}^{n}(d_i + 2)}{n} = n\log\frac{I + 2n}{n} = O(n\log(\frac{I}{n} + 2))$$

הביטוי $\frac{I}{n}$ מייצג את ממוצע ההיפוכים, כלומר ה-d_i הממוצע, לכן עלות החיפוש הממוצעת היא כפי שראינו בסעיף II, תהליך ההכנסה של האיבר נחלק ל-2, כאשר בשלב הראשון מחפשים את תת-העץ שמכיל את d_i האיברים, שהוא בממוצע $\frac{I}{n}$, ולאחר מכן מחפשים בתת-העץ הזה, כלומר

העלות לוגריתמית ביחס למספר האיברים בו - $O(\log(\frac{I}{n}))$, ולכן עבור סדרה של n הכנסות, העלות היא $O(n\log(\frac{I}{n}+2))$.

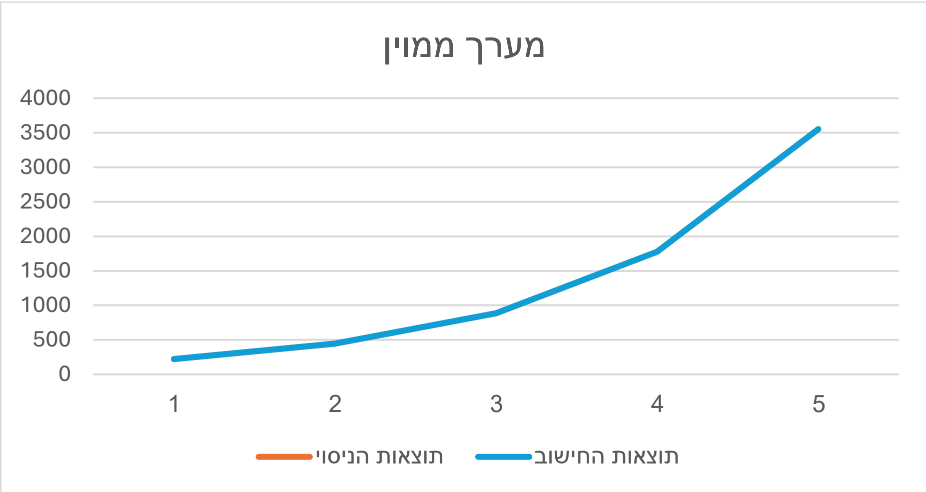
IV. נבחין כי במערך ממוין, מתקיים $I=0$, ולכן עלות החיפוש היא $n\cdot\log(2)=n\cdot 1=n$, שמשקף את תוצאות הניסוי בעמודה הימנית.

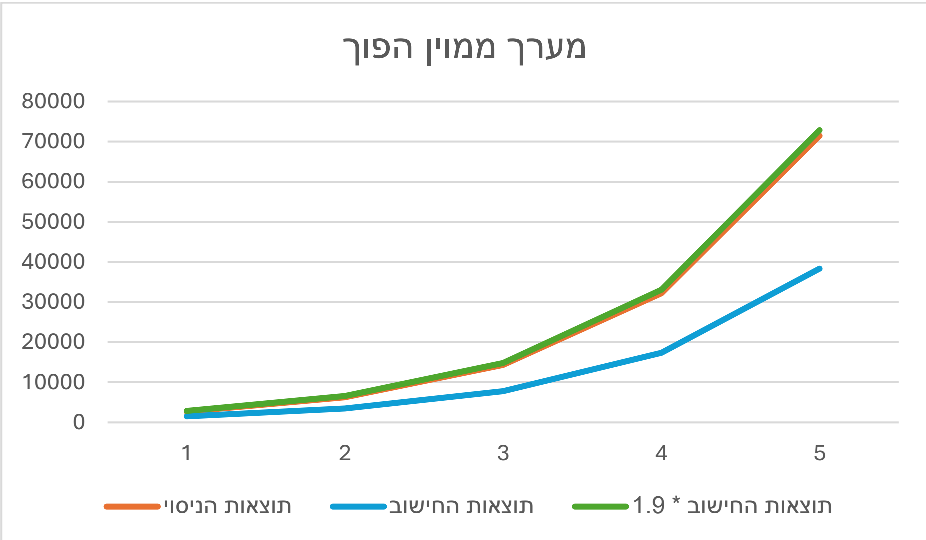
במערך הפוך, מתקיים $I=\binom{n}{2}=\frac{n(n-1)}{2}$, ולכן עלות החיפוש היא $n\cdot\log\left(\frac{n}{2}+1.5\right)$, מתנהג אסימפטוטית דומה לתוצאות העמודה השנייה עד כדי קבוע $c\approx 1.9$.

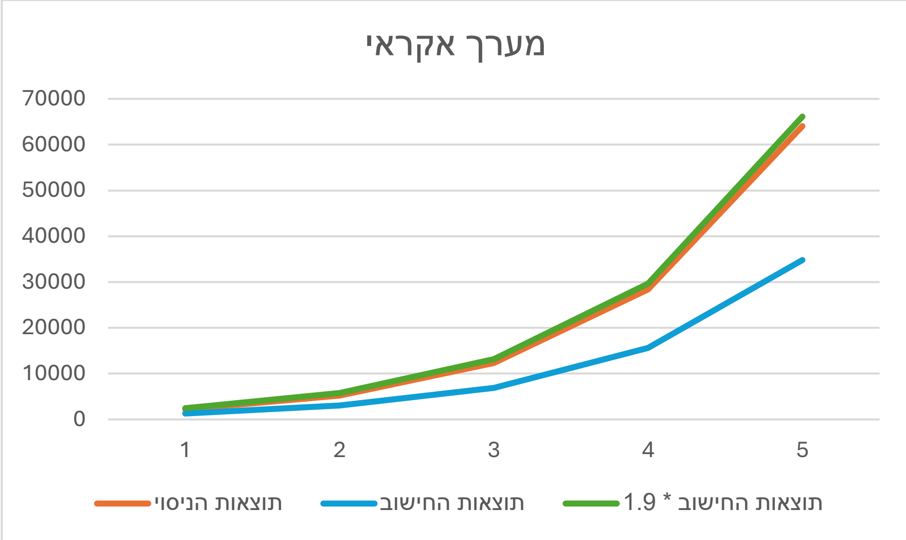
במערך אקראי, מספר ההחלפות הוא כחצי ממערך הפוך, ולכן תוצאות הניסוי דומות לשל מערך הפוך.

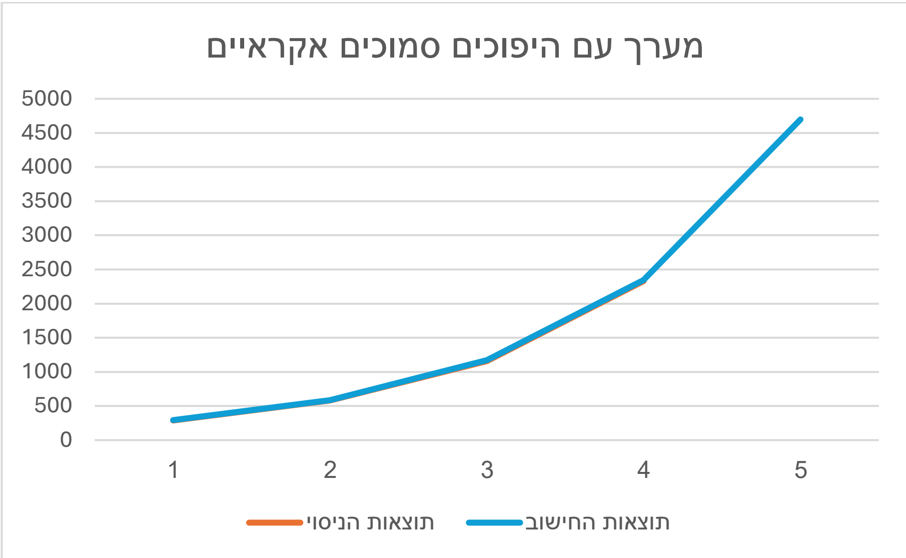
במערך בו יש חילופים אקראיים, מספר ההחלפות דומה למספר האיברים, ולכן תוצאות הניסוי דומות לשל מערך ממוין.

בתרשימים ניתן לראות על ציר ה-x את ערך i , בין 1 ל5. ציר ה-y מייצג את עלויות החיפוש. בתרשים הכתום ניתן לראות את תוצאות הניסוי, שלקוחות מטבלה 3. בתרשים הכחול ניתן לראות את תוצאות החישוב, בהתאם לנוסחה מסעיף 3 $n\log(\frac{I}{n}+2)$, כאשר I לקוח מטבלה 2, בהתאם לערך ה-i. בתרשים הירוק, במופיע בגרפים 2 ו3, ניתן לראות את תוצאות החישוב (מהתרשים הכחול) כפול c=1.9, וניתן לראות שגרף זה חוסם מלמעלה את הגרם הכתום.
*השתמשנו בלוג בבסיס 2 לצורך החישובים.






מערך אקראי
70000
60000
50000
40000
30000
20000
10000
0
1
2
3
4
5
תוצאות הניסוי
תוצאות החישוב
תוצאות החישוב * 1.9


מערך עם היפוכים סמוכים אקראיים
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
3
4
5
תוצאות הניסוי
תוצאות החישוב