

מבני נתונים ואלגוריתמים – תרגול #4

עצים

תרגיל:

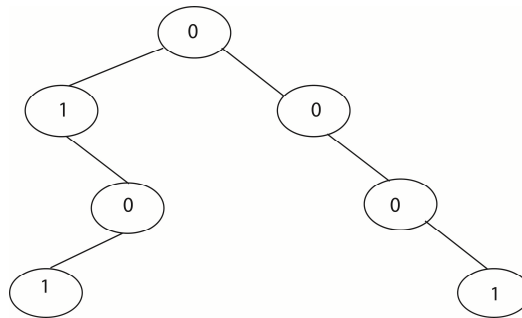
הציעו מבנה נתונים התומך בהכנסה, הוספה וחיפוש של מחרוזות ביטים ב- $O(k)$ אורך- k (המחרוזת).

- המבנה מתעלם מכפילויות.

פיתרון:

נשתמש בעץ בינארי. כל צומת יחזיק ערך אינפורמציה 0 או 1. כל מחרוזת תהיה מסלול בעץ – 0=שמאלה, 1=ימינה. אם מסיימים מסלול בעץ בצומת שהערך בו הוא 1 אז המחרוזת נמצאת במבנה.

דוגמא – העץ הבא מכיל את המחרוזות הבאות: 0,010, 111.



אלגוריתם הוספה: (רקורסיבי)

s - מצביע למחרוזת הביטים
 n - מספר הביטים שנשאר להוסיף
 T - מצביע לצומת בעץ

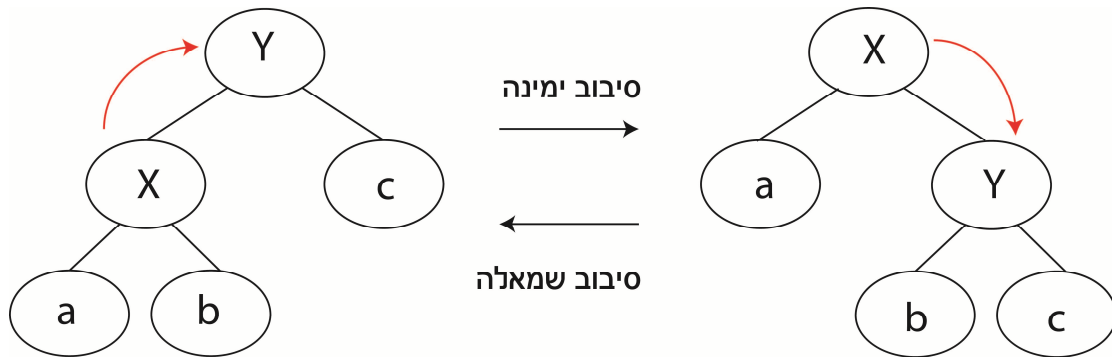
```
Insert(s, n, T):
  if n==0
    valid(T) = 1
  else if s==0
    next=Left(T)
    if next==null
      Left(T) = new node
    Insert(s+1, n-1, Left(T))
  else // s==1
    next=Right(T)
    if next==null
      Right(T) = new node
    Insert(s+1, n-1, Right(T))
  return
```

סיבוכיות: $T(k) = T(k-1) + O(1) = O(k)$
(הוכחה באינדוקציה).
← בבית – חיפוש והוצאה.

עצי AVL

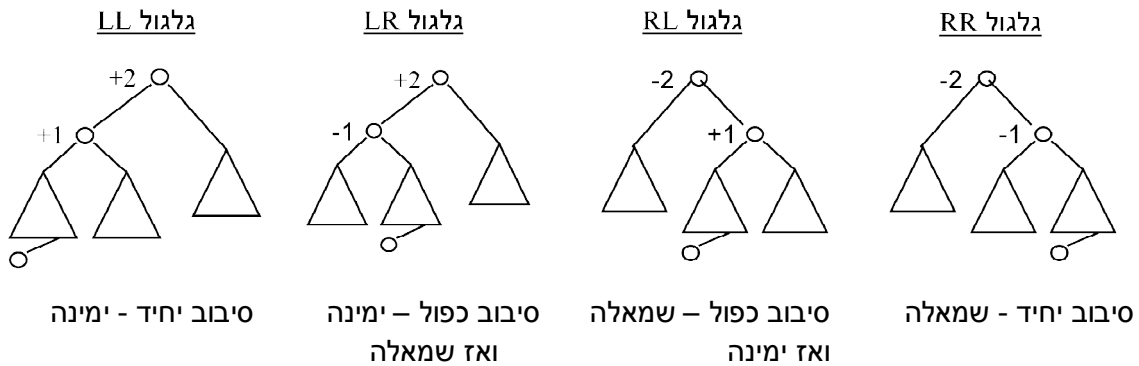
- עץ חיפוש מאוזן גובה - $H(T) = O(\log n)$
- עץ שבו ההפרש בין גובה תת-העץ הימני של צומת פנימי לתת-העץ השמאלי הוא לכל היותר 1.
- מוסיפים לכל צומת מקדם איזון: 0/1/-1.
- לאחר הוספת/מחיקת צומת, ייתכן שיהיה צורך לאזן מחדש. לכן נבצע סיבובים: (הצמתים בהם ייתכן שהופר האיזון הם לאורך מסלול ההוספה/מחיקה)

סיבוב פשוט:



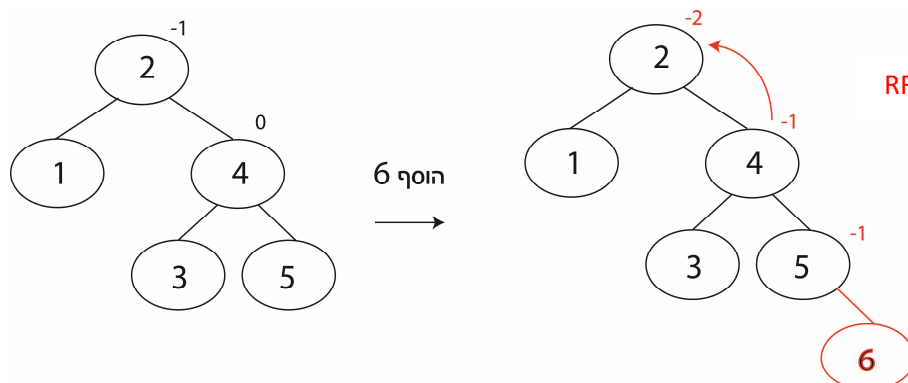
סוג ומספר הסיבובים תלויים בצורה שבה הופר האיזון - עדכון מקדמי האיזון מתבצע לאחר הוספת/מחיקת צומת כאשר עולים חזרה במעלה העץ.

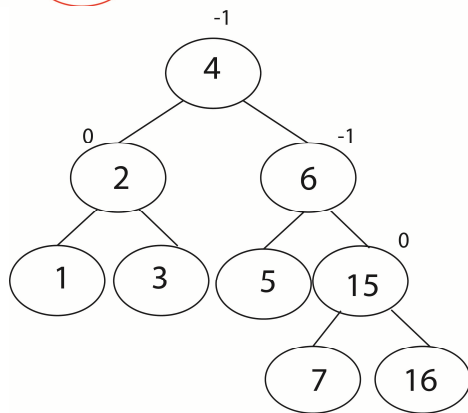
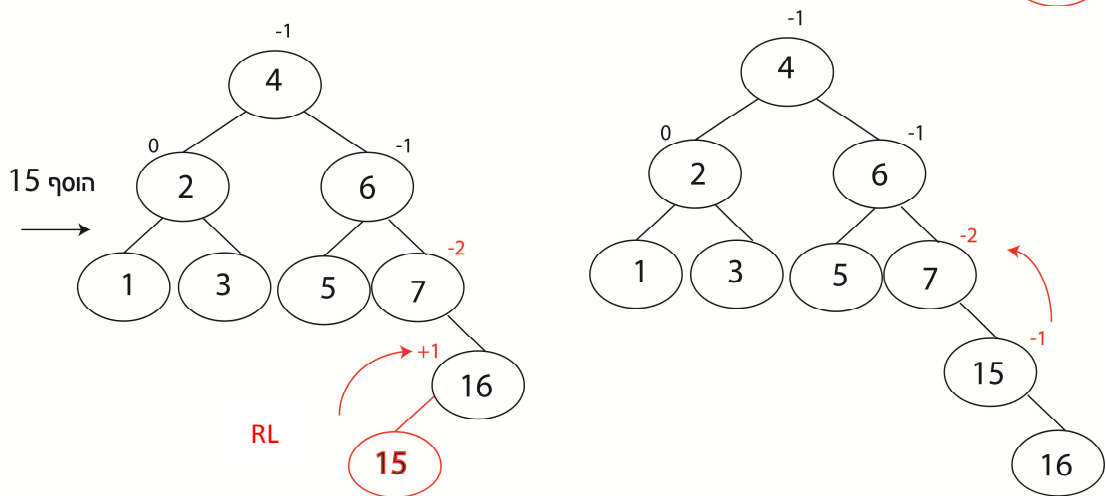
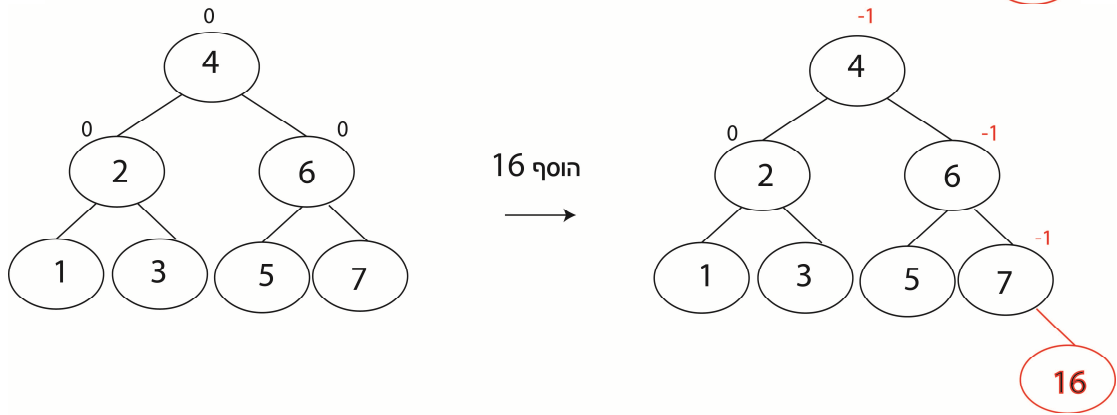
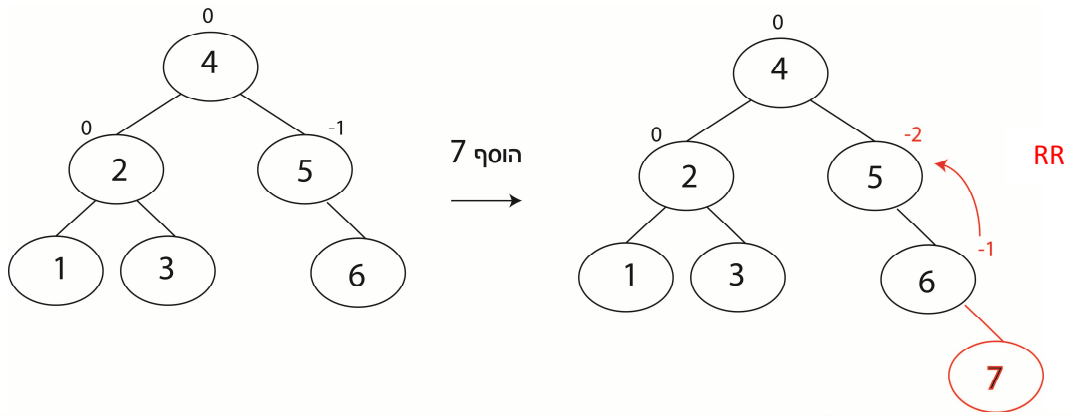
- אם באים משמאל בעץ - מוסיפים +1
- אם באים מימין בעץ - מחסירים -1



סיבוכיות: סיבובים - $O(1)$, עליה וירידה בעץ - $O(\log n)$.

דוגמא:



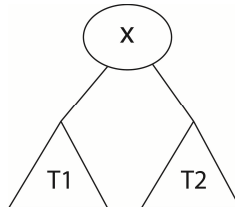


תרגיל:

נתונים 2 עצי AVL – T_1, T_2 וערך x כך ש- $T_1 < x < T_2$ (כלומר כל הערכים ב- T_1 קטנים ממש מ- x וכל הערכים ב- T_2 גדולים ממש מ- x). אחד את T_1 ו- T_2 לעץ AVL בצורה יעילה.

פיתרון:

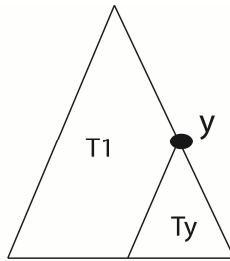
- חשב את גובהם של 2 העצים $h(T_1), h(T_2)$. $O(\log n_1 + \log n_2)$
- אם $h(T_1) = h(T_2)$ החזר:



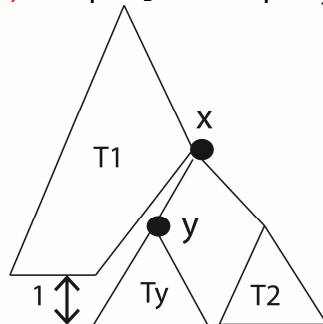
אחרת:

אם $h(T_1) > h(T_2)$:

- 1) חפש את תת העץ הימני של T_1 שגובהו $h(T_2)$. נקרא לתת-העץ הזה T_y ושורשו y . $O(\log n_1 - \log n_2)$



- 2) החלף את y ב- x , והוסף את T_y כבן שמאלי ו- T_2 כבן ימני. $O(1)$



- 3) חזור מ- x עד לשורש ותקן לפי הצורך. $O(\log n_1)$

אם $h(T_1) < h(T_2)$: כנ"ל הפוך.

סיבוכיות $O(\log n_1 + \log n_2) = O(\log(n_1 * n_2))$

תרגיל:

נתונים זוגות סדורים של מספרים. הציעו מבנה נתונים שתומך בפעולות הבאות (ב- $O(\log n)$):

- 1) הכנסת/הוצאת זוג
- 2) חיפוש לפי הקואורדינטה הראשונה או השניה
- 3) מעבר על זוגות ממיונים לפי הקואורדינטה הראשונה או השניה

פיתרון:

נשתמש בץ AVL. נחזיק 2 עצים. נכניס את הזוגות ל-2 העצים, שבראשון לפי הקואורדינטה הראשונה ובשני לפי השניה.

(1) הכנסה/חיפוש: $O(\log n) = 2O(\log n)$

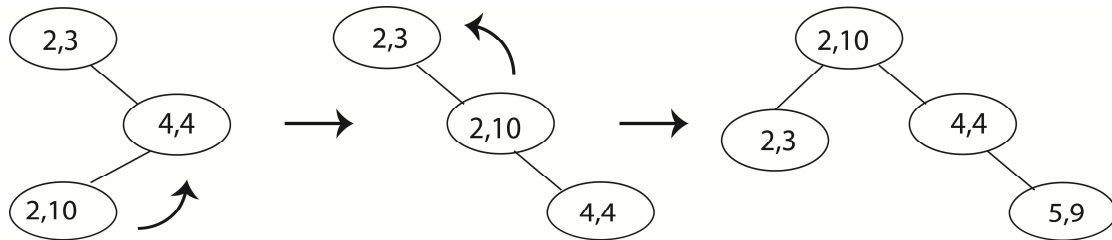
(2) חיפוש לפי קואורדינטה: בעץ המתאים לפי הקואורדינטה המתאימה $O(\log n)$.

(3) מעבר: בעץ המתאים, עוברים ב-inorder. $O(\log n)$.

דוגמא: (לפי הקואורדינטה הראשונה):

הוסף את הזוגות הבאים:

$(2,3), (4,4), (2,10), (5,9)$.



מעבר סדרתי: $(2,3), (2,10), (4,4), (5,9)$

אלגוריתמי מיון

מיוני השוואה:

- *Bubble sort*: עוברים על כל הזוגות ומחליפים ביניהם לפי הצורך. $O(n^2)$
- *Cocktail sort*: עושים *Bubble sort* ל-2 הכיוונים. $O(n^2)$
- *Insertion sort*: בכל איטרציה (מעבר על המערך) נלקח איבר ממערך הקלט, ומוכנס למקומו הנכון בתוך המערך הממוין שנבנה בחלק השמאלי של המערך. $O(n^2)$
- *Selection sort*: מוצאים את האיבר הנמוך ביותר ושמים אותו במקום הראשון, וכך הלאה. $O(n^2)$
- *Quick sort*: רקורסיבי – בוחרים באקראי איבר ציר, מעבירים לצד אחד את כל הגדולים ממנו, מצד שני את כל הקטנים ושמים את איבר הציר במקומו. כעת עושים שוב מיון מהיר עבור כל צד. $O(n^2)$ במקרה הגרוע אך בממוצע $O(\log n)$. **יציבות – תלוי במימוש.**
- *Merge sort*: ברקורסיה מחלקים את המערך ל-2, ומאחדים את 2 החלקים לפי הסדר. $O(n \log n)$
- *Heap sort*: מכניסים את כל האיברים לערימת מקסימום, וכל פעם מוצאים את השורש ומסדרים את הערימה מחדש. $O(n \log n)$ יציב.

- **יציבות של מיון**: מיון נקרא מיון יציב כאשר הוא שומר על הסדר של הנתונים לאחר המיון במידה ושניים זהים. דוגמה להבחנה בין מיון יציב ללא-יציב: אם רוצים למיין רשימת שמות על פי שם משפחה, אך אם שמות המשפחה זהים, למיין על פי השם הפרטי, אפשר למיין את המערך על פי שם פרטי ואחר כך למיין שוב על פי שם המשפחה. דבר זה יתאפשר רק אם המיון הוא יציב, אך אם הוא אינו יציב, אזי יכול להיות שהסדר הפנימי של השמות הפרטיים ייהרס.

תרגיל

נתונה רשימה של זוגות $(a_1, b_1), \dots, (a_n, b_n)$, כאשר a_i מציון תאריך לידה של לטאה ו- b_i תאריך פטירה. הציעו אלגוריתם שהפלט שלו הוא המספר המקסימלי של לטאות שחיו באותו זמן.

פיתרון:

- הרעיון: נמיין את תאריכי הלידה והפטירה ביחד תוך כדי שמירה על הנתון לידה/פטירה.
- לכל זוג (a_i, b_i) מגדיר 2 זוגות מספרים: $(a_i, 1)$, $(b_i, -1)$ (1 מציין לידה ו-1 מציין פטירה).
- נמיין לפי הקואורדינטה הראשונה ואז לפי השניה * $O(n \log n) = O(2n \log n)$
- כעת נעבור על הרשימה הממויינת לפי האלגוריתם הבא: $O(n)$

$max = 0$

$count = 0$

for each pair (x_i, y_i) in sorted list:

$count += y_i$

$max = \max(max, count)$

return max

$count$ – כמה חיים ביחד כרגע, max – הערך המקסימלי שחיו ביחד עד עכשיו

דוגמת ריצה:

הזוגות:

$(1,5), (2,3), (2,10), (3,10), (4,7)$

הזוגות החדשים ממוינים:

$(1,1) (2,1), (2,1) (3,1) (3,-1) (4,1) (5,-1) (7,-1) (10,-1) (10,-1)$

הלולאה:

i	1	2	3	4	5	6	7	8	9	10
Count	1	2	3	2	3	4	5	2	1	0
Max	1	2	3	3	3	4	4	4	4	4

• הלטאות שחיו בו-זמנית הם: $(1,5), (2,10), (3,10), (4,7)$.

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

* הערה של תרגיל חכם בכיתה 😊 - אם לא ממיינים גם לפי הקואורדינטה השניה, ייתכנו תוצאות שונות במקרים של לטאות שנולדו ונפטרו באותו יום. לדוגמא – אם מורידים את הזוג $(4,7)$, ו- $(3,-1)$ מופיע לפני $(3,1)$ נקבל מקסימום 3 לטאות. אבל אם $(3,1)$ מופיע לפני $(3,-1)$ נקבל מקסימום של 4 לטאות. לכן נמיין גם לפי הקואורדינטה השניה - תחשבו שלטאות נולדות בבוקר ונפטרות בלילה 😊.