

מבני נתונים

תרגול 3
ליאור שפירא

עצי חיפוש בינאריים

- ראינו כי עלות פעולה היא $O(h)$
- בהינתן עץ בעל n צמתים, גובהו המינימאלי יהיה $O(\log n)$
- כיצד נגרום לעץ להיות מאוזן?



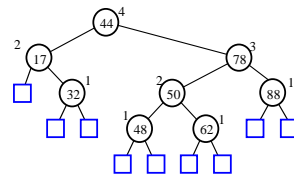
עצי חיפוש בינאריים מאוזנים

גובה של עץ AVL

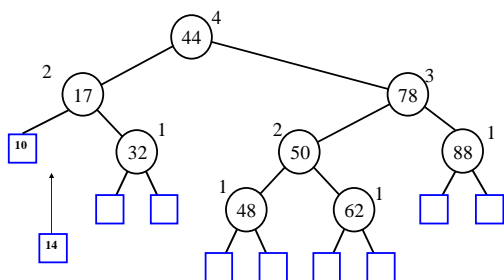
- למה: גובה של עץ T המחזיק n מפתחות הוא $O(\log n)$ הוכחה:
 - נמצא את $n(h)$, מס' מינימלי של צמתים פנימיים בעץ AVL בעל גובה h
 - בבירור $n(1)=1, n(2)=2$
 - עבור $n \geq 3$, עץ AVL בעל גובה h מכיל את השורש, תת עץ בגובה $h-1$ ותת עץ בגובה של לפחות $h-2$
 - ר"א: $n(h) \geq 1 + n(h-1) + n(h-2)$

עצי AVL 1962 / G.M. Adelson-Velskii / E.M. Landis

- עץ חיפוש בינארי מאוזן (מבחינת גובה)
- עבור כל צומת פנימית v של עץ T
- גובה תתי העצים מתחת ל- v יכול להיות שונה בעד 1



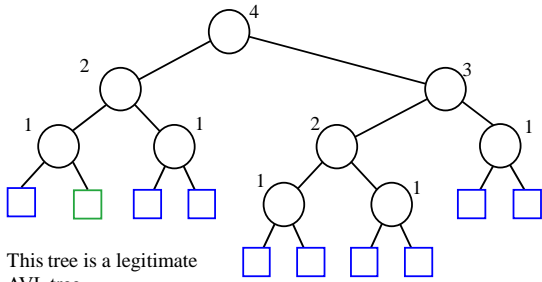
Insertion



גובה של עץ AVL

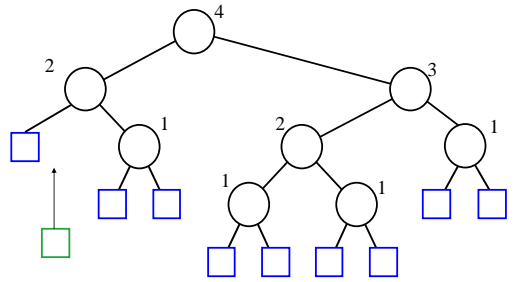
$$\begin{aligned}
 n(h-1) > n(h-2) &\Rightarrow n(h) \geq 2n(h-2) \\
 n(h) &\geq 2n(h-2) \\
 n(h) &\geq 4n(h-4) \\
 \dots & \\
 n(h) &\geq 2^i n(h-2i) \\
 \Rightarrow n(h) &\geq 2^{h/2-1} && \text{פתרון הסדרה} \\
 \Rightarrow h &\leq 2 \log n(h) + 2 && \text{לוג לשני הצדדים} \\
 \Rightarrow h &= O(\log n) && \text{מש"ל}
 \end{aligned}$$

צומת פנימית חדשה

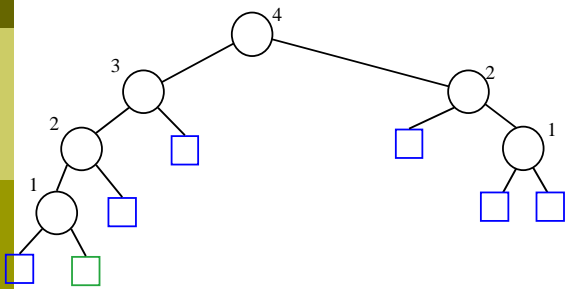


This tree is a legitimate AVL tree...

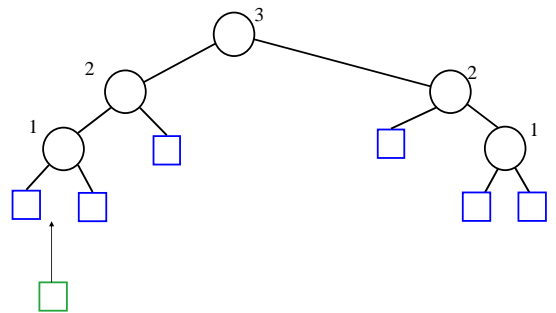
נתעלם מהערכים, נתמקד באיזון העץ



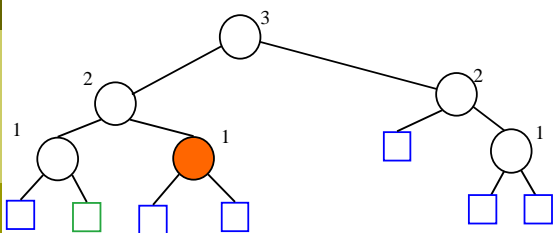
נוסיף את הצומת החדש



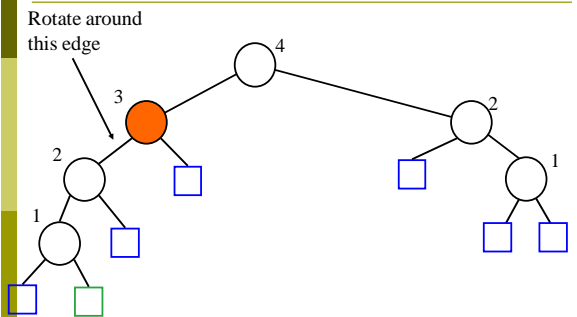
עוד דוגמה



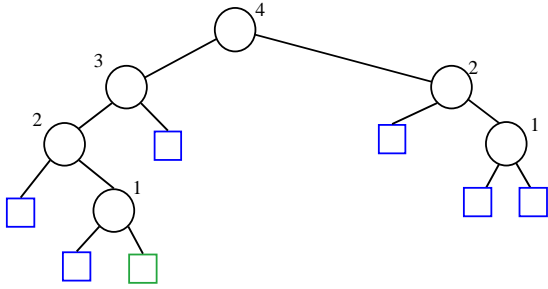
נתקן את השגיאות



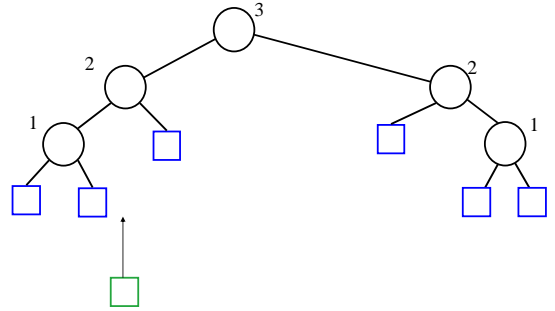
הפרנו את חוקיות העץ



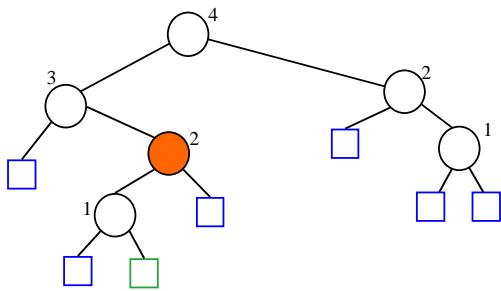
נוסיף את הצומת החדש



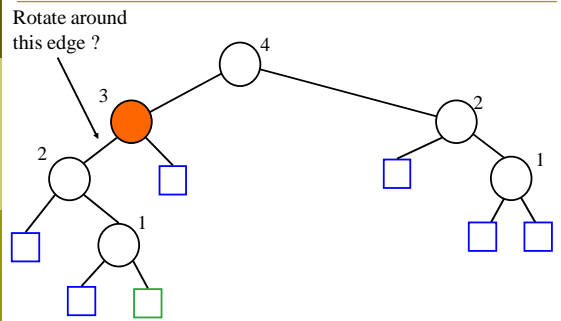
עוד דוגמה



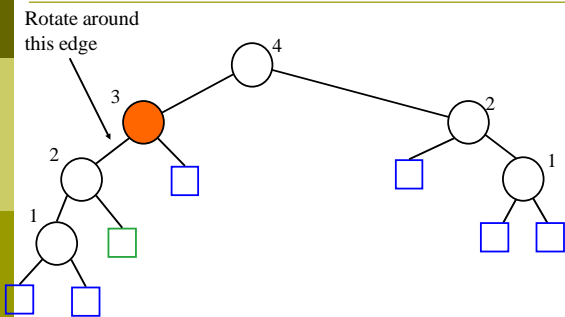
הפעם הרוטציה לא עוזרת



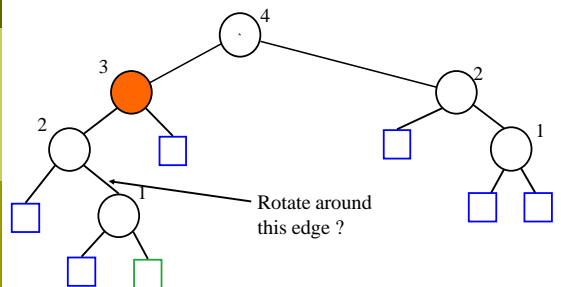
שוב קיבלנו הפרה של החוקים



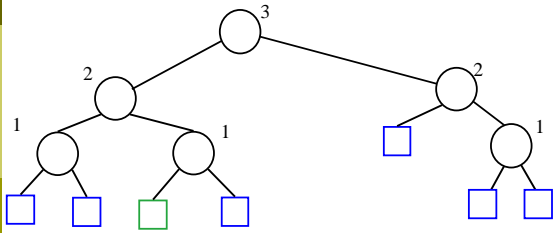
חזרנו למקרה הקודם



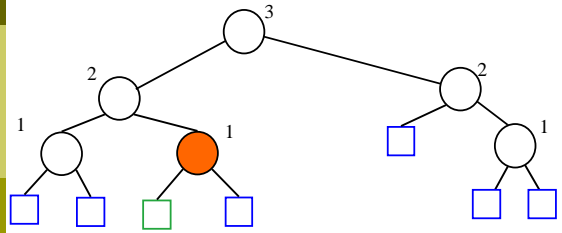
מה ניתן לעשות?



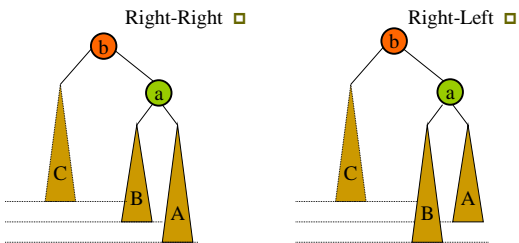
אז מה האלגוריתם הכללי?



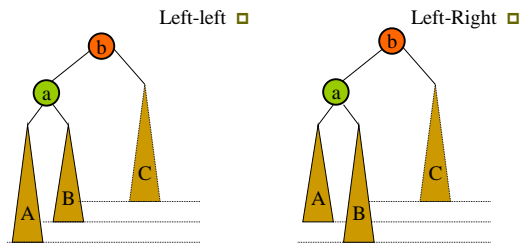
תיקנו את הפרת החוקים



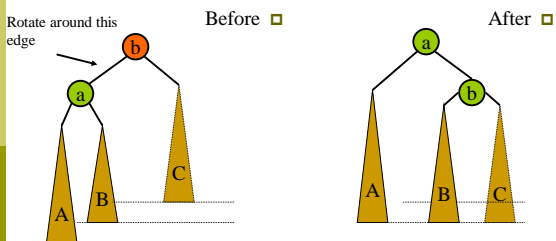
ותמונות המראה שלהם



סוגי חוסר שיווי משקל



Left-left → single rotation



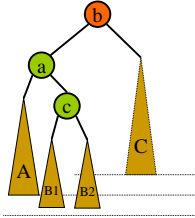
נסתכל על הבעיה לוקאלית

שני עקרונות מנחים

- Imbalance will only occur on the path from the inserted node to the root (only these nodes have had their subtrees altered - local problem)
- Rebalancing should occur at the *deepest unbalanced node* (local solution too)

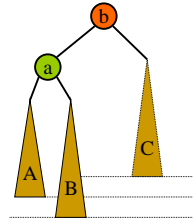
Lets look at this more carefully

• Before:

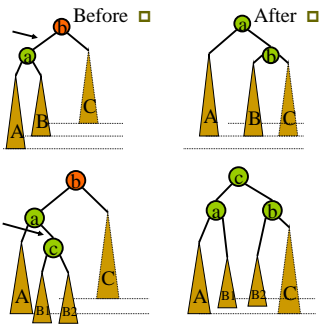


Left-Right fixing

• Before:



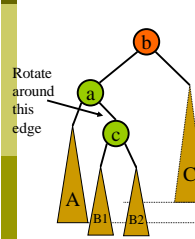
בהוספת צומת חוסר האיזון "מוגבל"



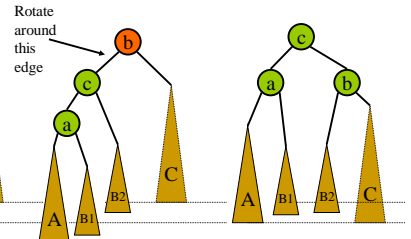
□ שימו לב שבשני המקרים של חוסר איזון בהוספת צומת, לאחר rebalancing, גובה תת העץ שהשוורש שלו הוא הצומת שהייתה בחוסר איזון, נשאר אותו דבר! ז"א שלאחר איזון (הטציה או שתיים) אין צורך להמשיך ולבדוק חוסר איזון יותר גבוה בעץ, כי בוודאות אין כזה!

We need two rotations here...(double rotation)

• Before:



• After:



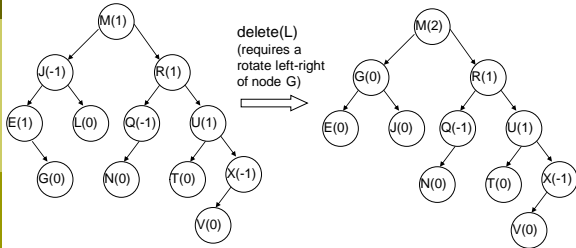
AVL Tree Delete

- An AVL delete is similar to a regular binary tree delete
 - search for the node
 - remove it
 - zero children: replace it with null
 - one child: replace it with the only child
 - two children: replace it with right-most node in the left subtree

סיכום

- We fix the first node x on the way up where there is a violation
- After the fix, x is at the same height as it was before, so no nodes further up towards the root will need to be updated.
- Can implement using just **2 bits** per node for rebalancing (the difference between the heights of the children)

AVL Tree Delete

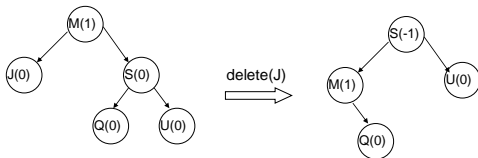


Notice that even after fixing J, M is still out of balance

AVL Tree Delete

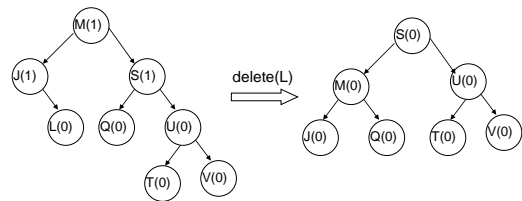
- Complications arise from the fact that deleting a node can unbalance a number of its ancestors
 - insert only required you find the first unbalanced node
 - delete will require you to go all the way back to the root looking for imbalances
 - Must balance any node with a ± 2 balance factor (+2 the left sub-tree is 2 levels deeper, -2 the right sub-tree is 2 levels deeper)

Rotating Nodes



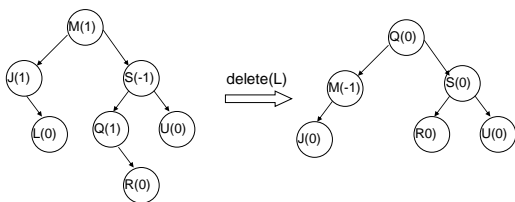
Deleting a node in the left sub-tree (M's balance becomes 2).
Need to rotate M with its right sub-tree.
S's balance factor is 0 before rotate.
Just do a rotate left of node S.
Notice that the height of the tree *does not* change in this case.

Rotating Nodes



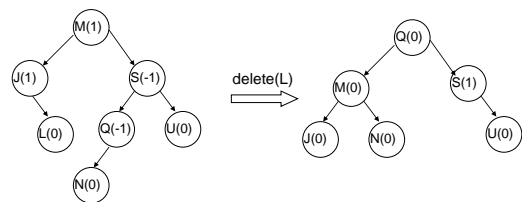
Deleting a node in the left sub-tree (M's balance becomes 2).
Need to rotate M with its right sub-tree.
S's balance factor is 1 before rotate.
Just do a rotate left of node S.
Notice that the height of the tree *does* change in this case.

Rotating Nodes



Deleting a node in the left sub-tree (M's balance becomes 2).
Need to rotate M with its right sub-tree.
S's balance factor is -1 before rotate.
Need to do a right rotate of Q with S and then a left rotate of Q with M.
Notice that the height of the tree changes.

Rotating Nodes



Deleting a node in the left sub-tree (M's balance becomes 2).
Need to rotate M with its right sub-tree.
S's balance factor is -1 before rotate.
Need to do a right rotate of Q with S and then a left rotate of Q with M.
Notice that the height of the tree changes.

לינקים מעניינים

- [AVL Trees on Wikipedia](#)
- [AVL Tree applet](#)
- [Nice Red-Black tree demo](#)
- [Another AVL/RB tree applet](#)

זמני ריצה

- a single restructure is $O(1)$
 - using a linked-structure binary tree
- find is $O(\log n)$
 - height of tree is $O(\log n)$, no restructures needed
- insert is $O(\log n)$
 - initial find is $O(\log n)$
 - Restructuring up the tree, maintaining heights is $O(\log n)$
- remove is $O(\log n)$
 - initial find is $O(\log n)$
 - Restructuring up the tree, maintaining heights is $O(\log n)$

The End