



دانشگاه شهید بهشتی کرمان

درخت جستجو دودویی خود-متوازن

ساختمان های داده و الگوریتم

درخت جستجو دودویی

Binary search tree

Type tree

Invented 1960

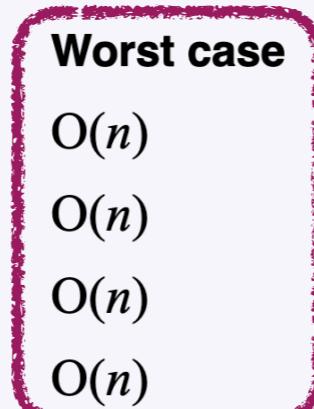
Invented P.F. Windley, A.D. Booth, A.J.T.
by Colin, and T.N. Hibbard

Time complexity in big O notation

Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

درخت جستجو دودویی

Binary search tree		
Type	tree	
Invented	1960	
Invented by	P.F. Windley, A.D. Booth, A.J.T. Colin, and T.N. Hibbard	
Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$



$$h = O(n)$$

درخت جستجو دودویی

هدف و تقابل ایده‌ها

- درخت ارتفاع متوازن
 - ارتفاع دو زیر درخت چپ و راست را برای هر راس برابر نگه می‌دارد.
- درخت اندازه متوازن
 - تعداد راس‌های دو زیر درخت چپ و راست را برای هر راس برابر نگه می‌دارد.
- درخت وزن متوازن
 - با توجه به تعدد استفاده از رئوس درختی را به صورت «آنلاین» به وجود می‌آورد که امید ریاضی هزینه دسترسی به گره‌ها کمینه شود.

درخت جستجو دودویی

انواع و مدل‌ها

- 2–3 tree
- AVL tree 
- Red–black tree
- Scapegoat tree
- Splay tree
- Tango tree
- ...

درخت جستجو دودویی

انواع و مدل‌ها

- 2–3 tree
- AVL tree
- Red–black tree
- Scapegoat tree
- Splay tree
- Tango tree
- ...

AVL درخت

Adelson-Velsky and Landis

AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION

G. M. ADEL'SON-VEL'SKI^{II} AND E. M. LANDIS

In the present article we discuss the organization of information contained in the cells of an automatic calculating machine. A three-address machine will be used for this study.

Statement of the problem. The information enters a machine in sequence from a certain reserve. The information element is contained in a group of cells which are arranged one after the other. A certain number (the information estimate), which is different for different elements, is contained in the information element. The information must be organized in the memory of the machine in such a way that at any moment a very large number of operations is not required to scan the information with the given evaluation and to record the new information element.

An algorithm is proposed in which both the search and the recording are carried out in $C \lg N$ operations, where N is the number of information elements which have entered at a given moment.

A part of the memory of the machine is set aside to store the incoming information. The information elements are arranged there in their order of entry. Moreover, in another part of the memory a "reference board" [1] is formed, each cell of which corresponds to one of the information elements.

The reference board is a dyadic tree (Figure 1a): each of its cells has no more than one left cell, and no more than one right cell subordinated to it. Direct subordination induces subordination (partial ordering). In addition, for each cell of the tree, all the cells which are subordinate to a left (right)

درخت AVL

Adelson-Velsky and Landis

AVL tree

Type tree

Invented 1962

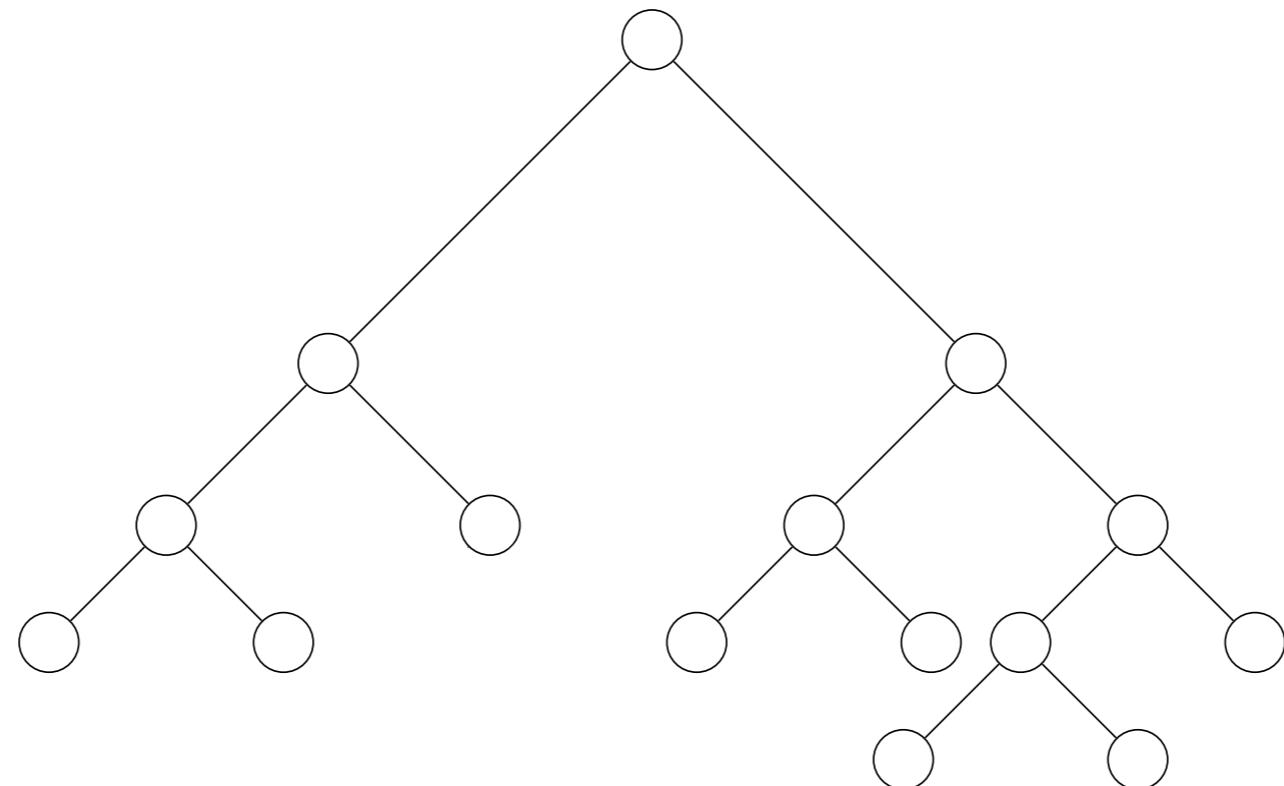
Invented Georgy Adelson-Velsky and Evgenii
by Landis

Time complexity in big O notation

Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$

درخت AVL

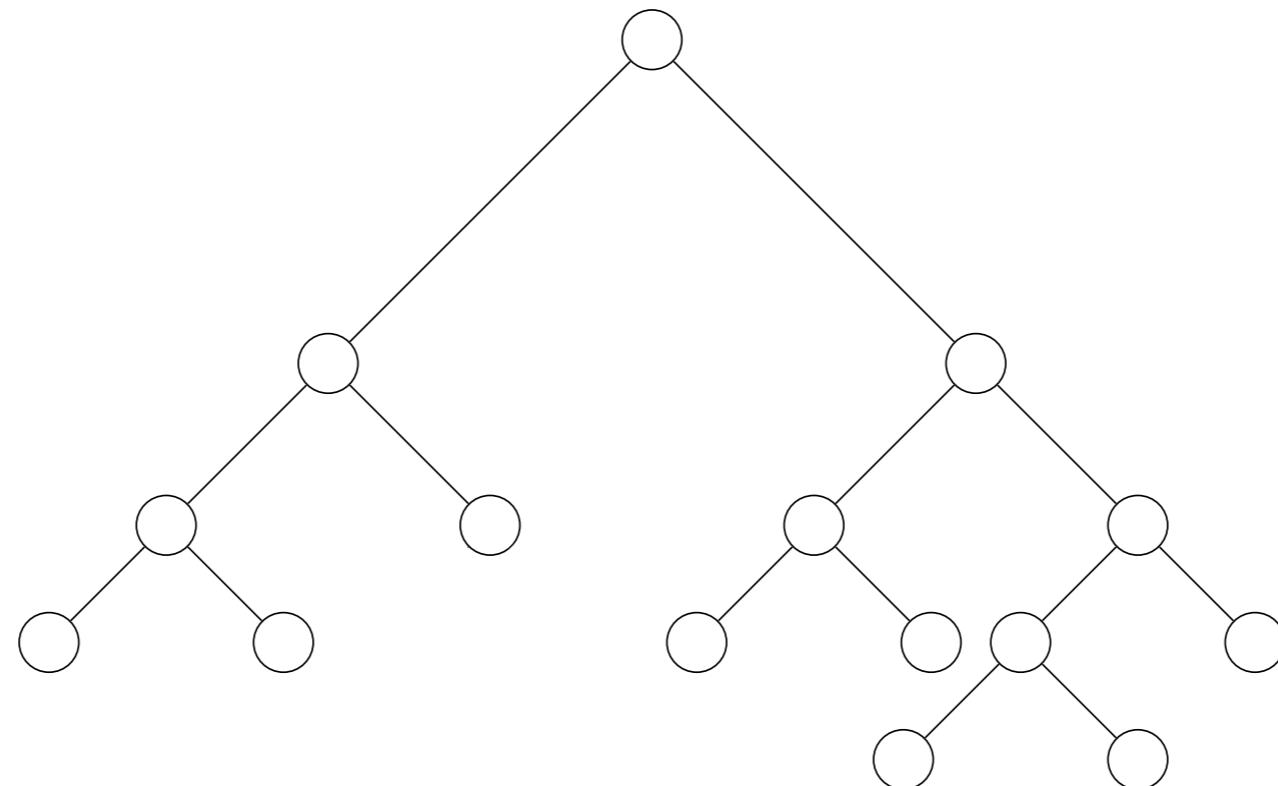
Adelson-Velsky and Landis



آیا این درخت متوازن ه ؟ متوازن نیست، تقریبا متوازن ه

درخت AVL

Adelson-Velsky and Landis

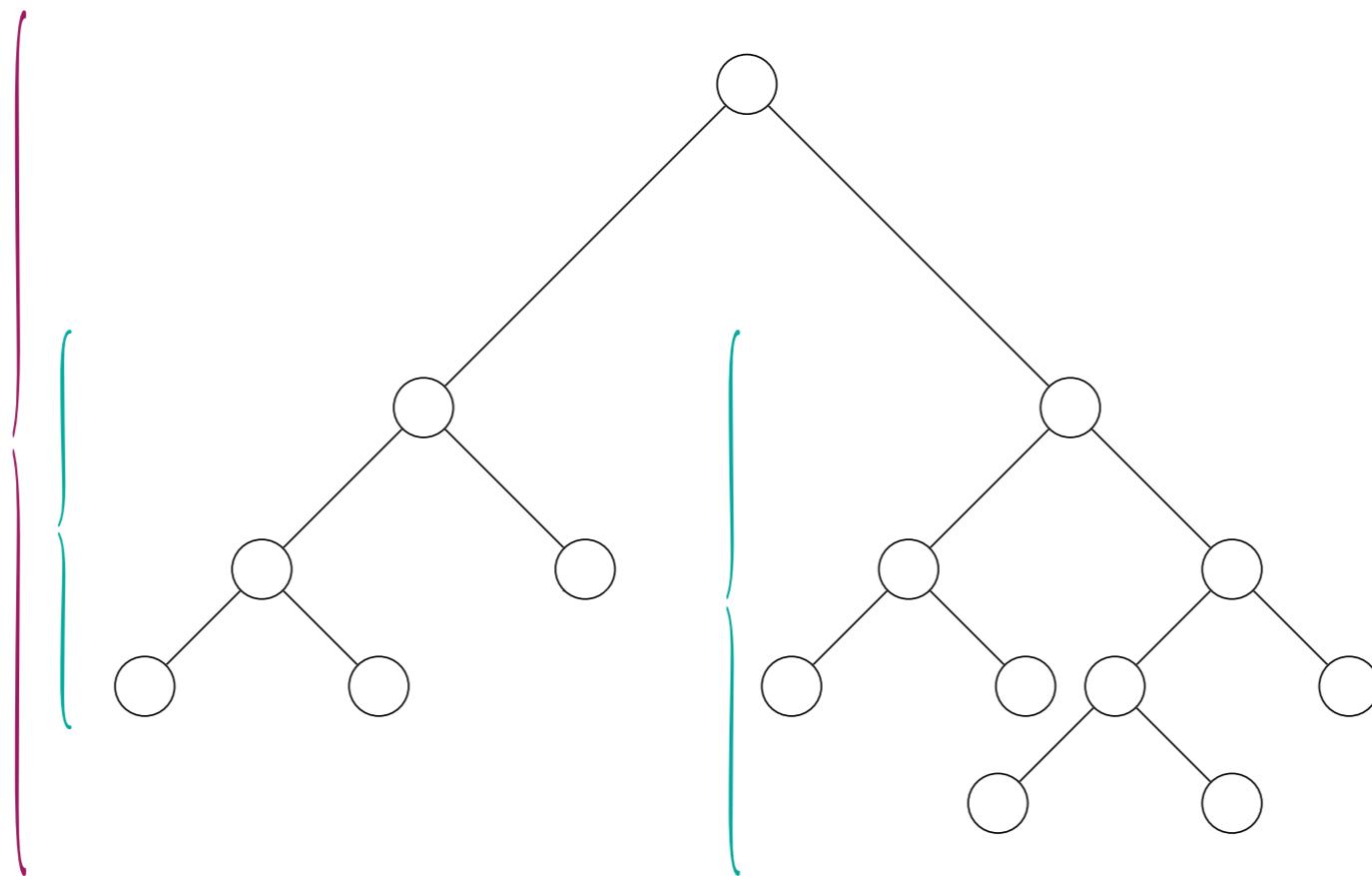


اصلًا متوازن یعنی چی؟

AVL درخت

Adelson-Velsky and Landis

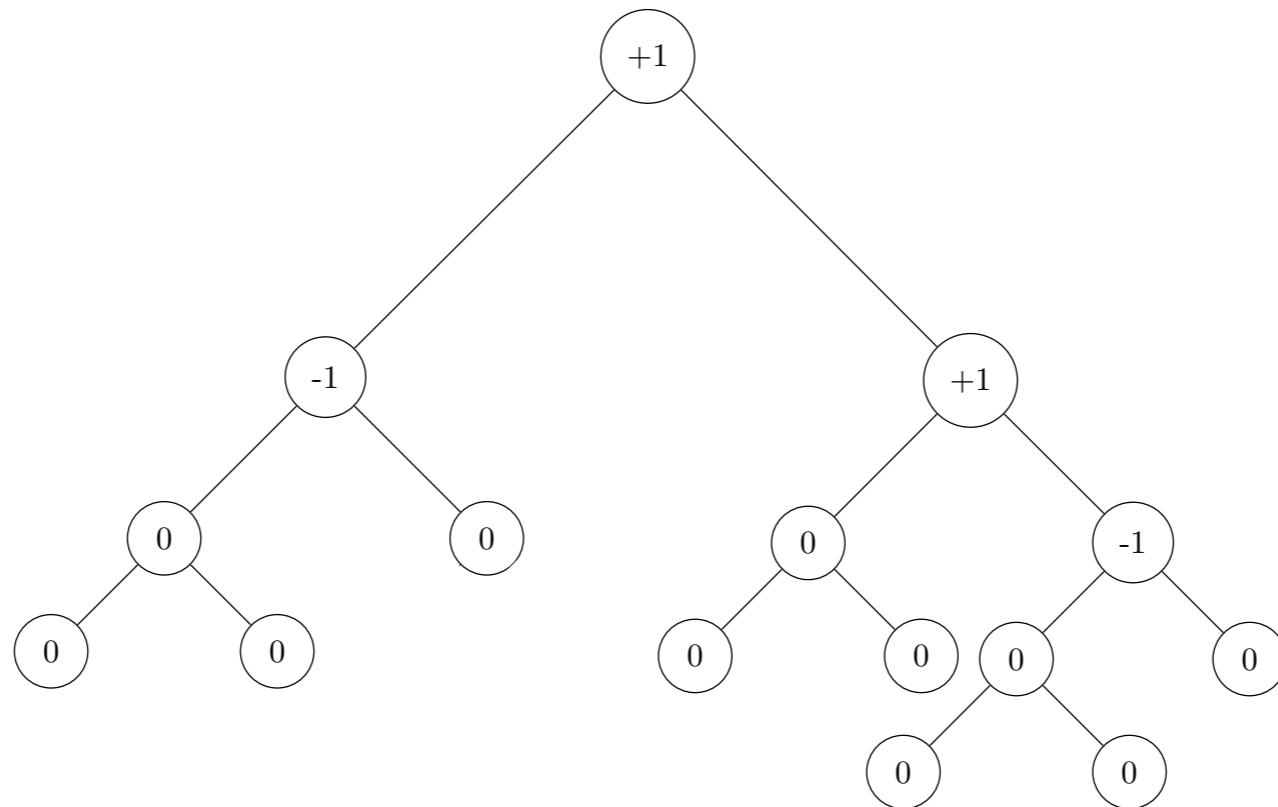
- $\text{BF}(X) := \text{Height}(\text{RightSubtree}(X)) - \text{Height}(\text{LeftSubtree}(X))$
- $\text{BF}(X) \in \{-1, 0, 1\}$



درخت AVL

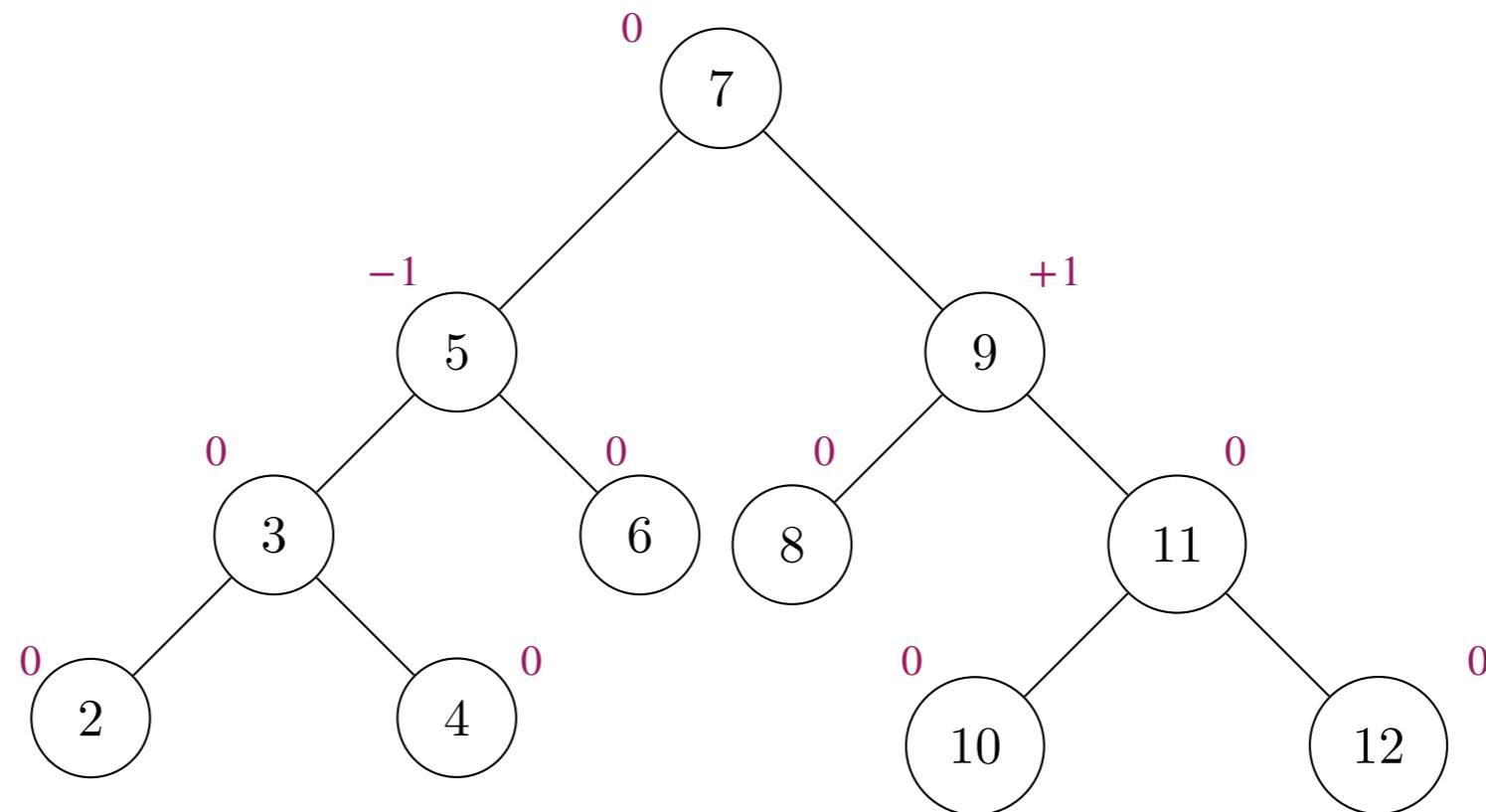
Adelson-Velsky and Landis

- $\text{BF}(X) := \text{Height}(\text{RightSubtree}(X)) - \text{Height}(\text{LeftSubtree}(X))$
- $\text{BF}(X) \in \{-1, 0, 1\}$



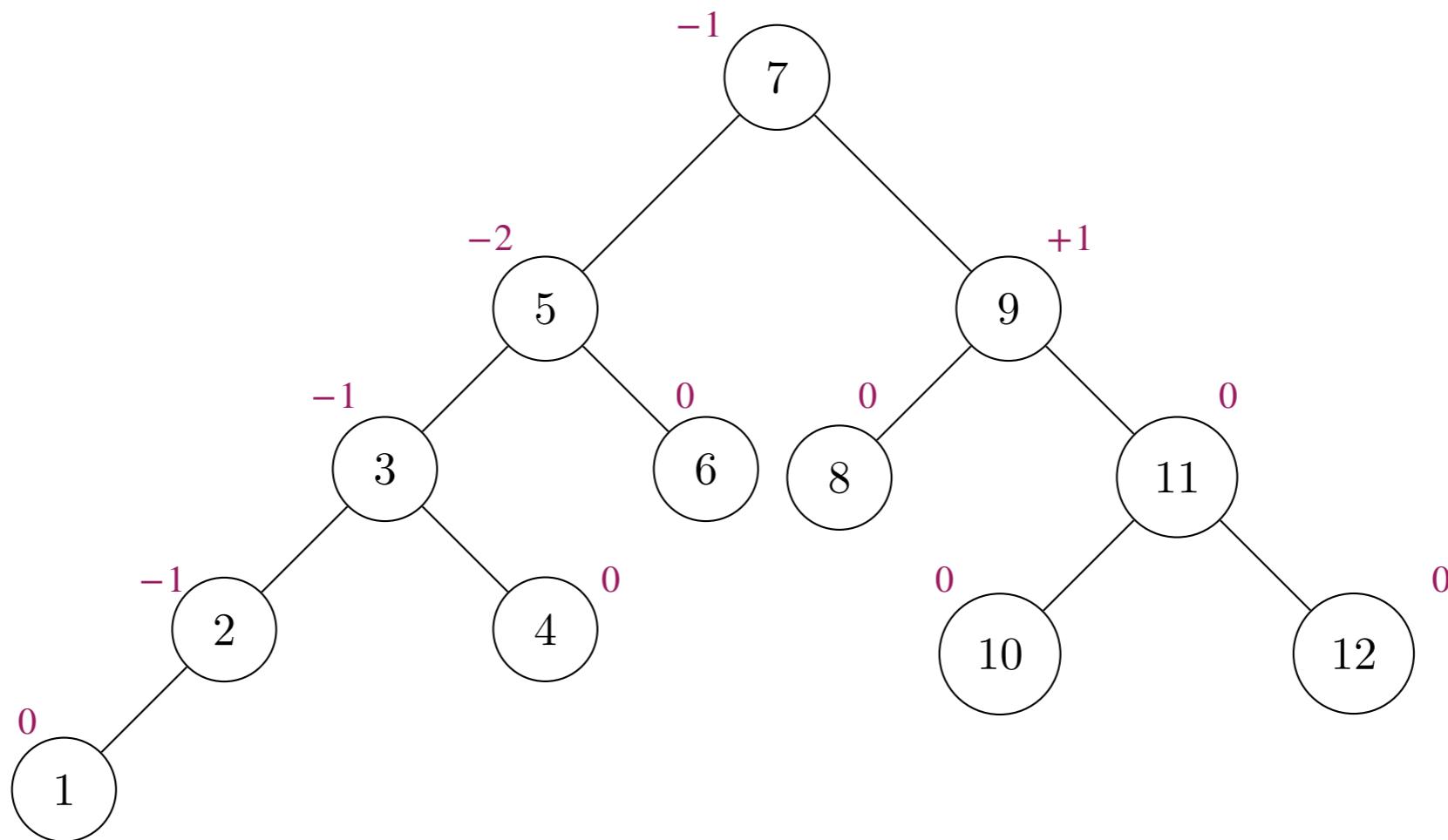
عملیات چرخش

در درخت زیر گره $x = 7$ رو درج می‌کنیم:



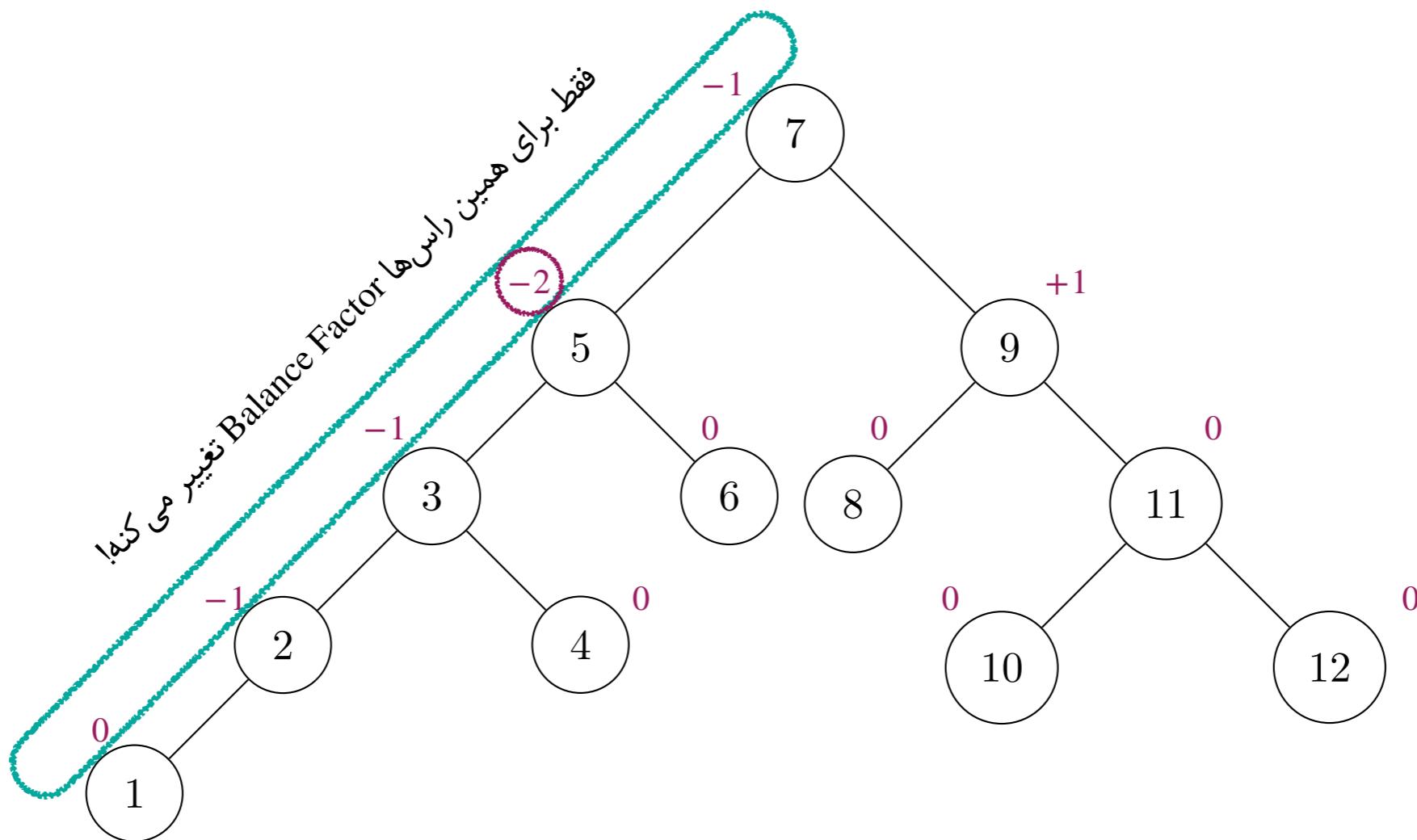
عملیات چرخش

در درخت زیر گره $x = 7$ رو درج می‌کنیم:



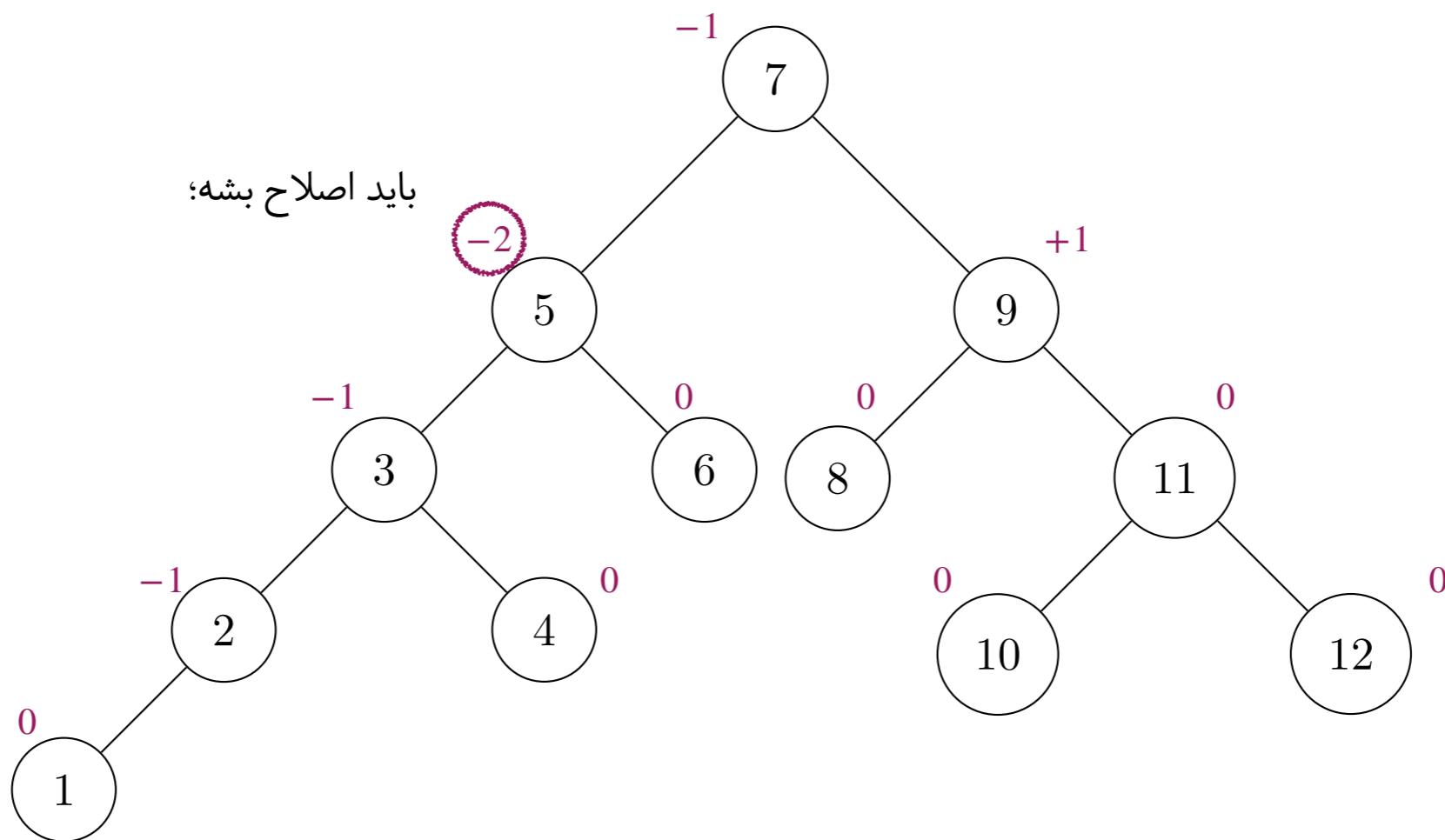
عملیات چرخش

فقط Balance Factor برای رئوس مسیر «ریشه» تا «گره درج شده» تغییر می‌کنه!



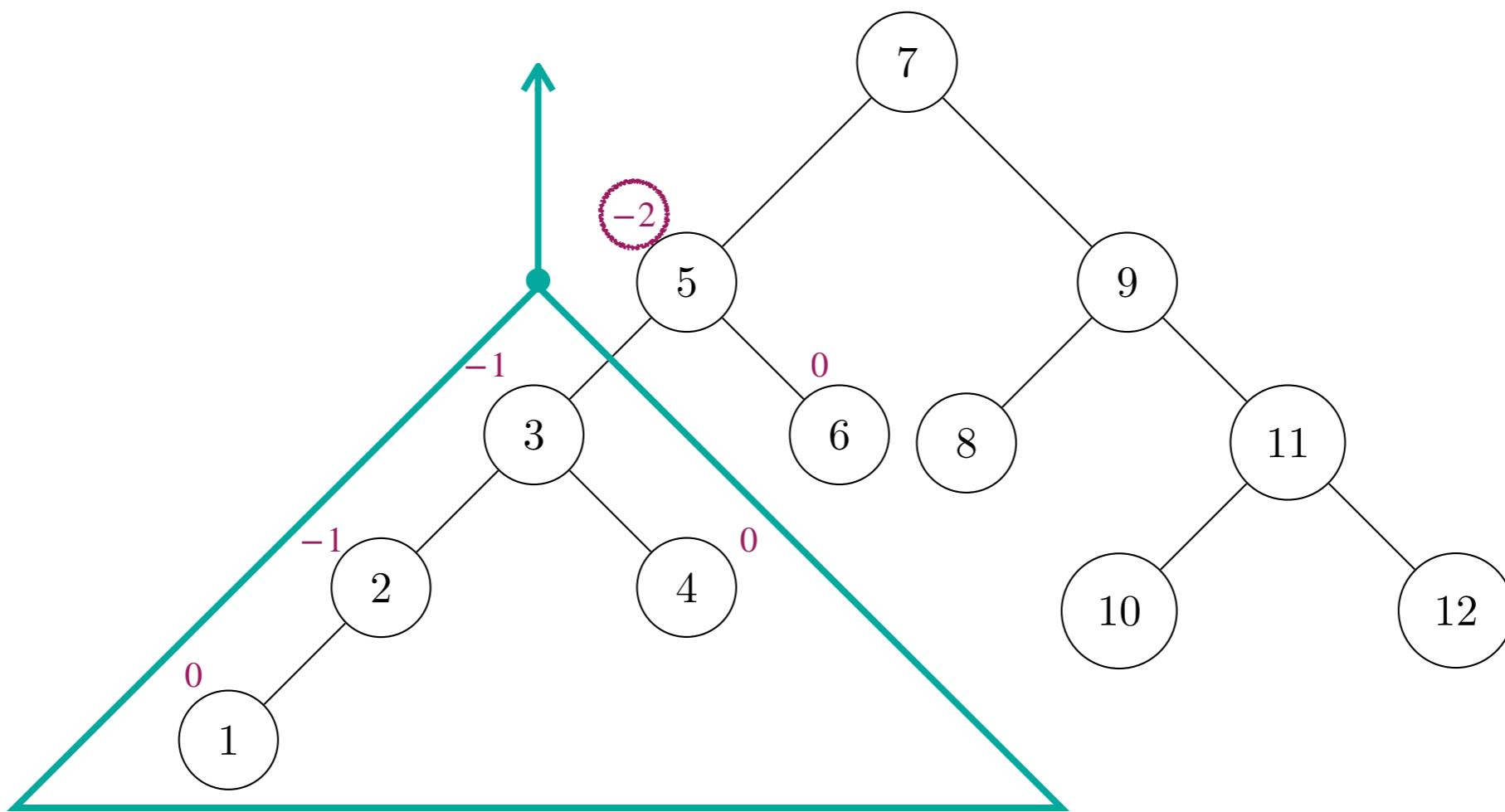
عملیات چرخش

برای تمام این رئوس که «ثابت توازن» آنها $\{1, 0, -1\}$ نیست، باید از عملیات چرخش استفاده کرد.



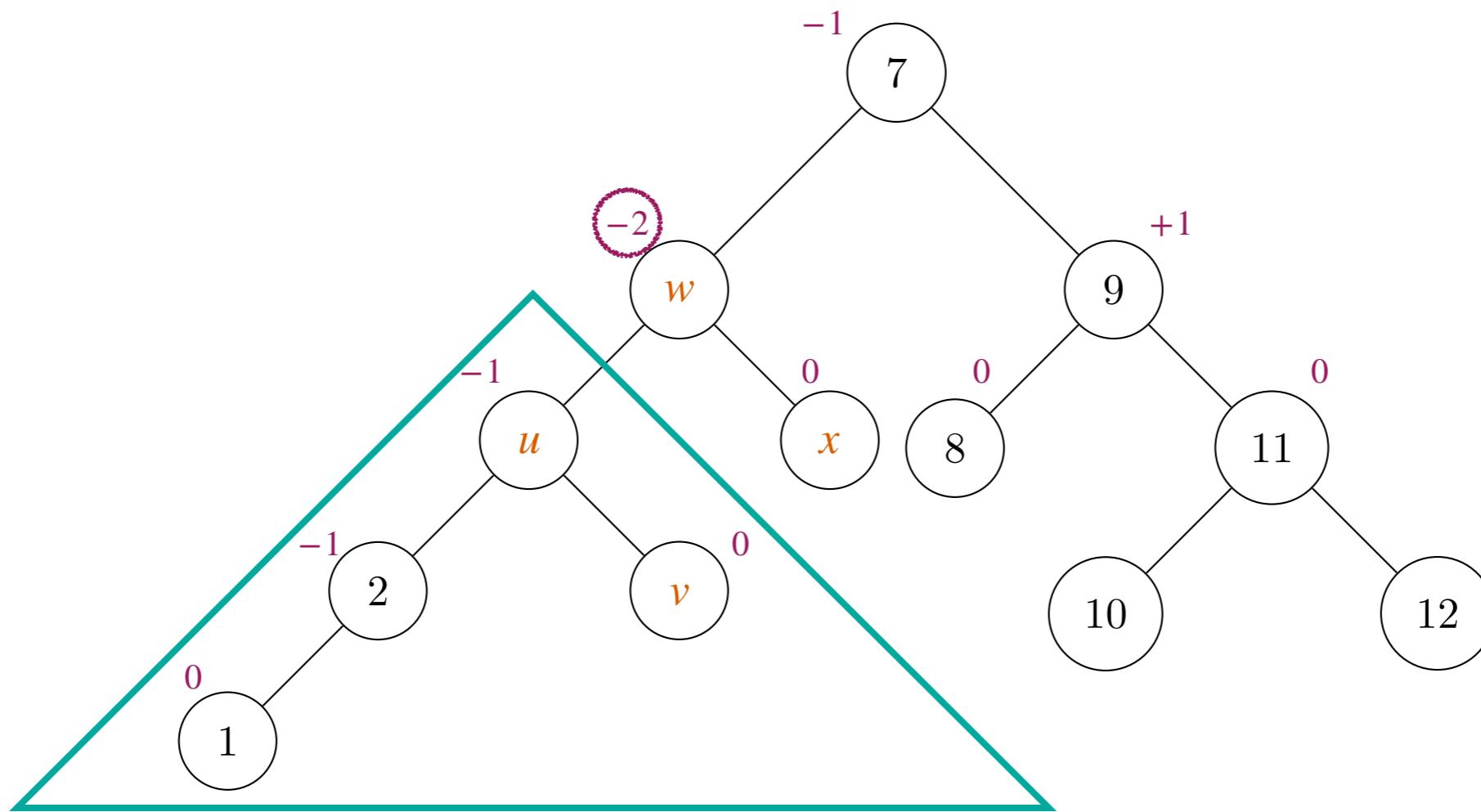
عملیات چرخش

ایده چرخ آن است که «زیر درخت با ارتفاع بیشتر» را بالاتر قرار دهیم.



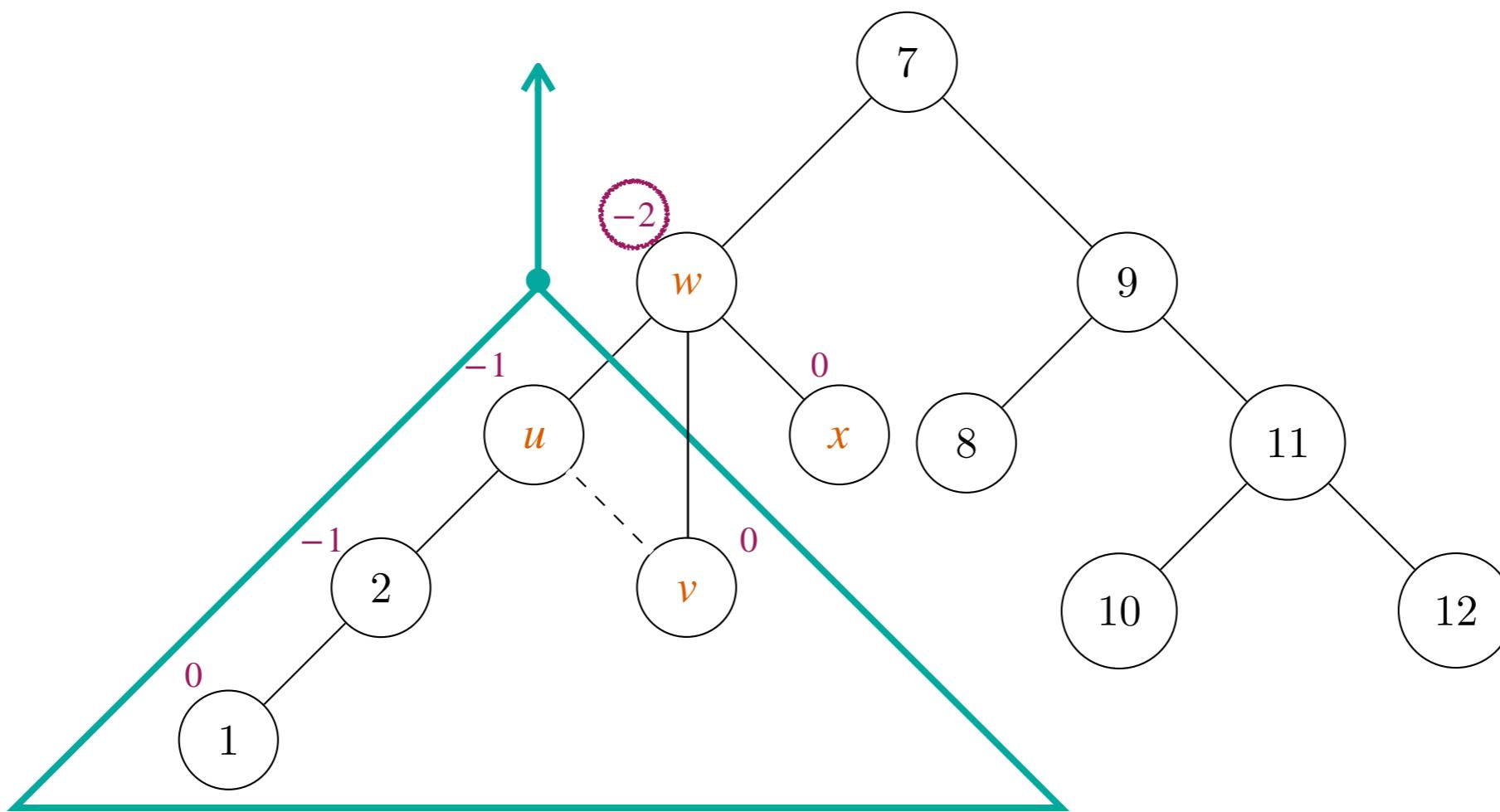
عملیات چرخش

با توجه به تعریف درخت جستجو دودویی داریم:



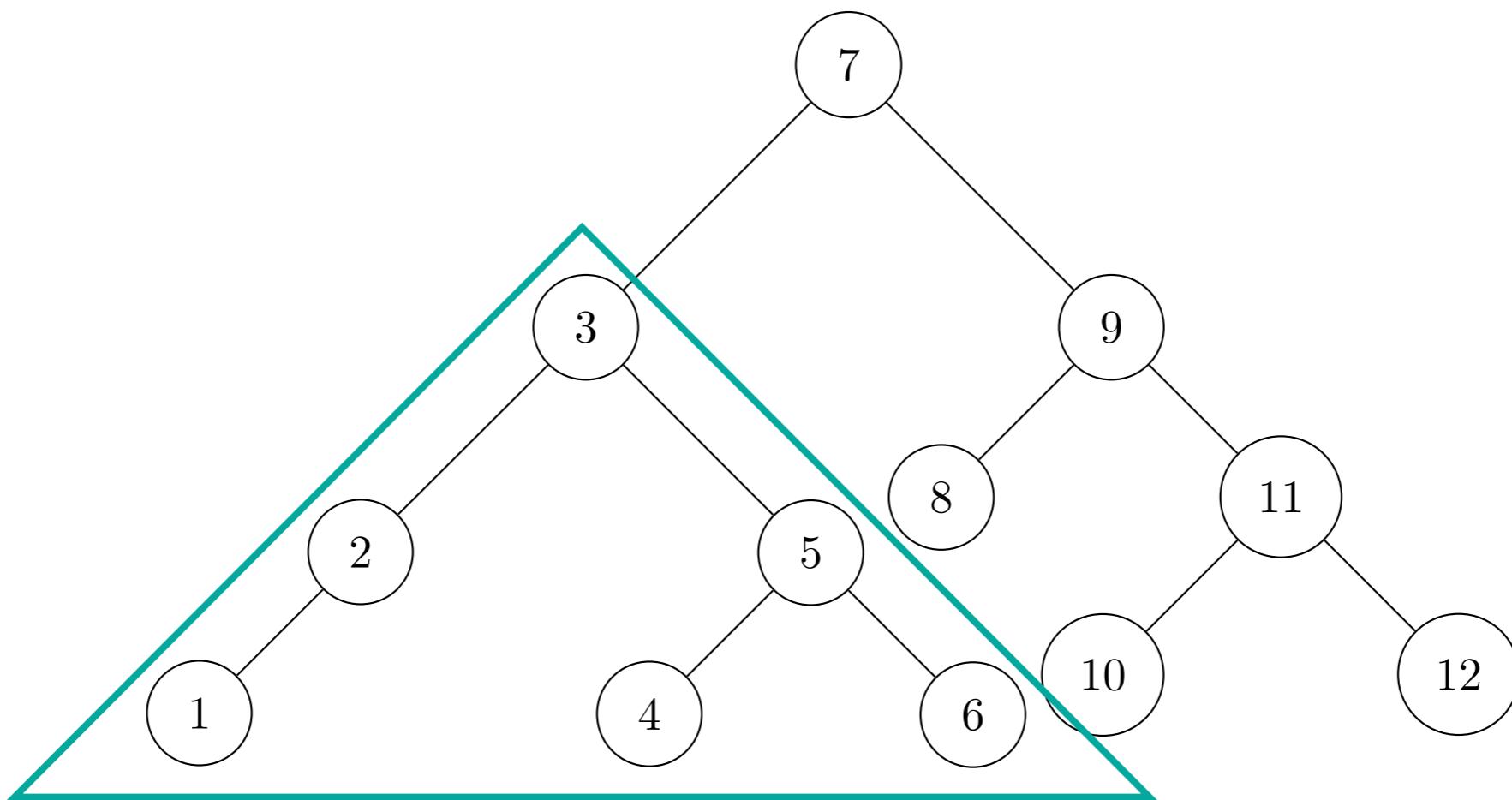
عملیات چرخش

با توجه به تعریف درخت جستجو دودویی داریم:



عملیات چرخش

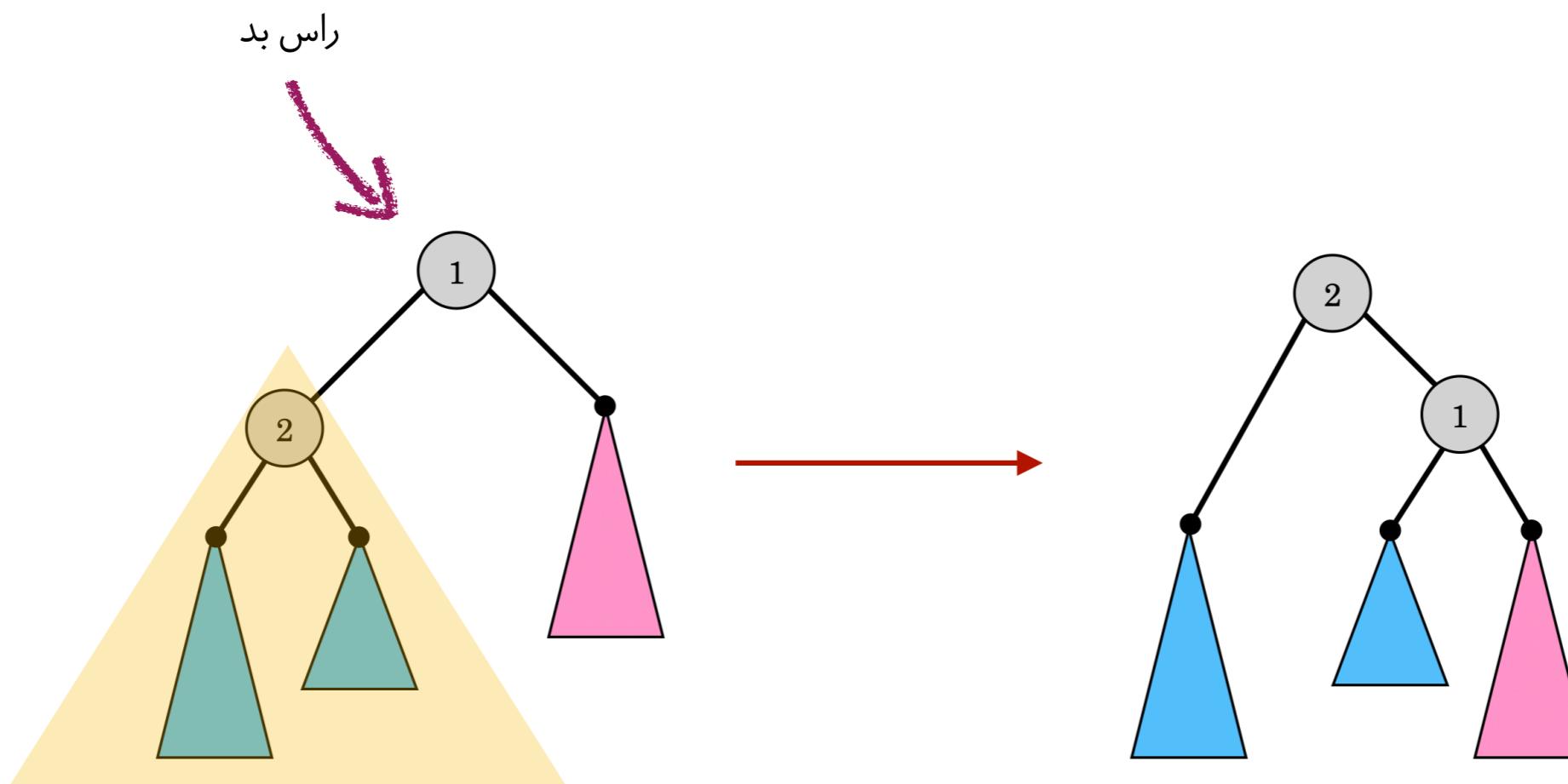
و درخت متوازن شد.



$$u \leq v \leq w \leq x$$

عملیات چرخش

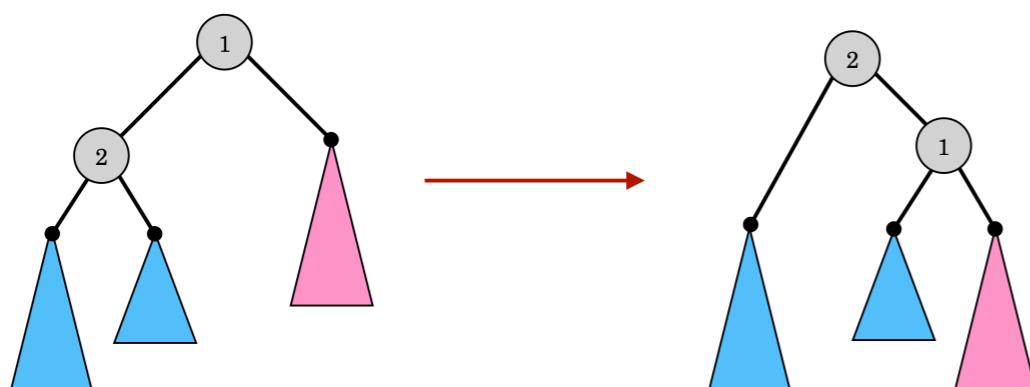
در یک نگاه



عملیات چرخش

تحلیل مرتبه زمانی

- برای هر راس v عملیات چرخش تنها ۳ اشاره‌گر را تغییر می‌دهد.
- پس از مرتبه $\mathcal{O}(1)$ زمان می‌برد.
- تنها برای روئوس موجود در مسیر «ریشه» تا «گره تغییر یافته» باید اجرا شود.
- طبق آنچه می‌دانیم، تعداد آنها $\mathcal{O}(\log n)$ است.
- پس در مجموع مرتبه زمانی ما برای یک عملیات بهروزرسانی $\mathcal{O}(h + \log n)$ خواهد بود.
- فراموش نشود که $.h = \mathcal{O}(\log n)$



```

def insert_node(self, root, key):
    # Find the correct location and insert the node
    if not root:
        return TreeNode(key)
    elif key < root.key:
        root.left = self.insert_node(root.left, key)
    else:
        root.right = self.insert_node(root.right, key)

    root.height = 1 + max(self.getHeight(root.left),
                          self.getHeight(root.right))

    # Update the balance factor and balance the tree
    balanceFactor = self.getBalance(root)
    if balanceFactor > 1:
        if key < root.left.key:
            return self.rightRotate(root)
        else:
            root.left = self.leftRotate(root.left)
            return self.rightRotate(root)

    if balanceFactor < -1:
        if key > root.right.key:
            return self.leftRotate(root)
        else:
            root.right = self.rightRotate(root.right)
            return self.leftRotate(root)

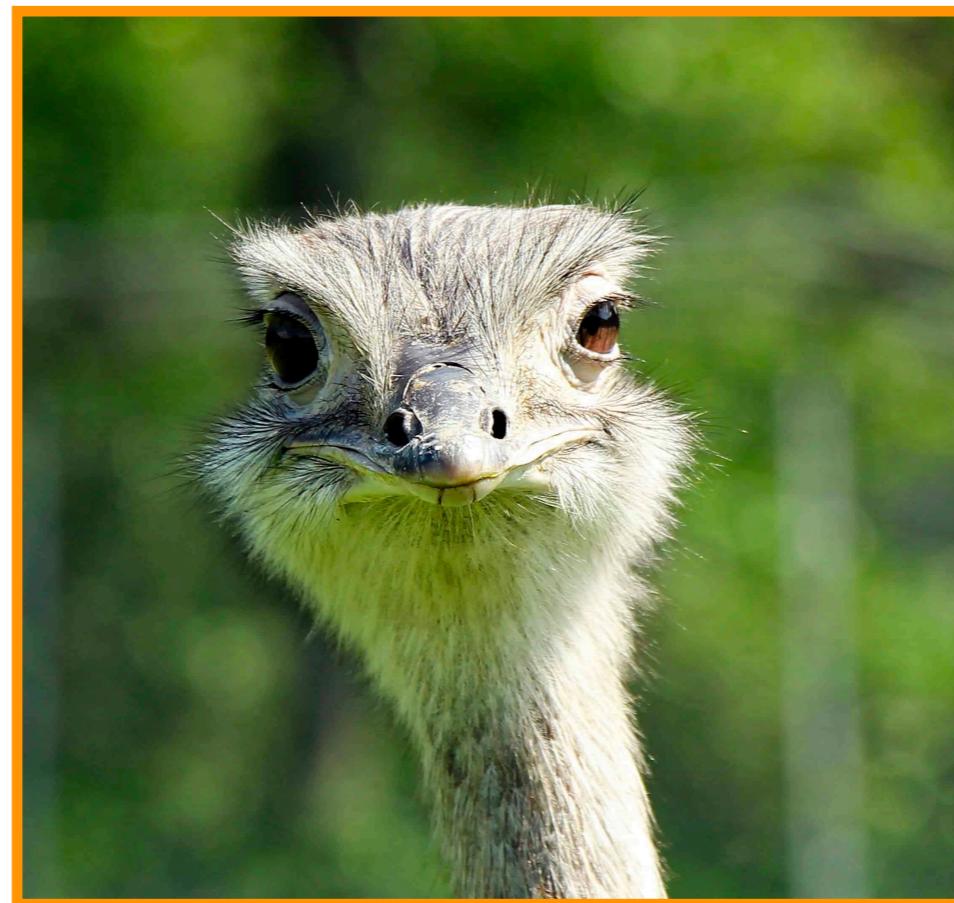
    return root

```

زیر درخت راست عمیق تره

زیر درخت چپ عمیق تره

سؤال؟



چند سوال.

- AVL trees are not weight-balanced?
- Hash tables versus binary trees
- Rotation distance Problem