

# حل تمرین سری پنجم (هرم)

ساختمان داده ها و الگوریتم



# سوال ۱.

دکتری ۹۳

تعداد برگ‌های یک max-heap با  $n$  عنصر حداقل چندتا است؟

$n - 1$  (۱)

$n - 2$  (۲)

$\lceil \frac{n}{2} \rceil$  (۳)

$\lceil \frac{n}{3} \rceil$  (۴)

# سوال ۱.

گزینه ۳.

تعداد برگ‌های یک max-heap با  $n$  عنصر حداقل چندتا است؟

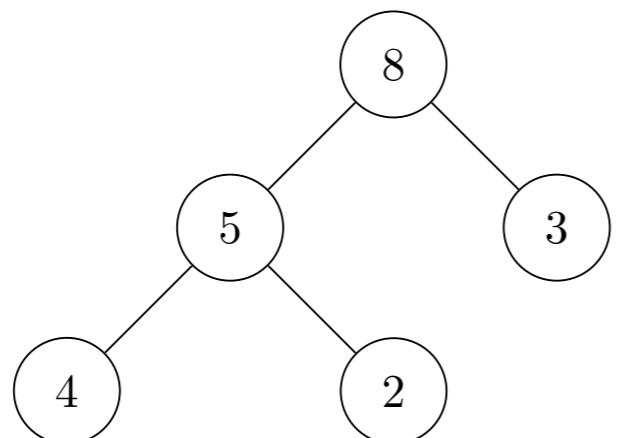
در فصل‌های قبلی دانستیم  $n_0 + n_2 + 1 = n_0$ ; از طرفی heap یک درخت

دودویی کامل است بنابراین  $\{0,1\}^{n_1}$ . پس داریم:

$$\begin{cases} n_1 \in \{0,1\} \\ n_2 + 1 = n_0 \\ n_0 + n_1 + n_2 = n \end{cases} \implies n_0 \sim \frac{n}{2}$$

$$n - 1 \quad (1)$$

$$n - 2 \quad (2)$$



$$\lceil \frac{n}{2} \rceil \quad (3)$$

$$\lceil \frac{n}{3} \rceil \quad (4)$$

## سوال ۲.

دکتری ۹۲

بیشینه تعداد مقایسه‌هایی که لازم است تا بتون یک max-heap را به یک min-heap با  $n$  گره تبدیل کرد؟

$\mathcal{O}(n)$  (۱)

$\mathcal{O}(\log n)$  (۲)

$\mathcal{O}(n + \log n)$  (۳)

$\mathcal{O}(n \log n)$  (۴)

## سوال ۲.

### گزینه ۱.

بیشینه تعداد مقایسه‌هایی که لازم است تا بتوان یک min-heap را به یک max-heap با  $n$  گره تبدیل کرد؟

فرض کنید که min-heap شما یک آرایه نامرتب است؛ ساختن یک max-heap از یک آرایه به طور درجا از  $O(n)$  زمان خواهد برد.

### ساختن هرم

در اینجا می‌خواهیم با استفاده از روال MAX-HEAPIFY یک آرایه‌ی  $A[1\dots n]$  را به یک Max Heap تبدیل کنیم.

روال Build-Max-Heap در درخت حرکت کرده و با استفاده از MAX-HEAPIFY آن را به صورت درجا به Max Heap تبدیل می‌کند. (توجه داشته باشید ابتدا  $i$  با اندیس آخرین گره‌ای که برگ نیست مقداردهی می‌شود.)

```
def Build_Max_Heap(A):
    heap.size=len(A)
    st=(heap.size)//2
    for i in [st, st-1, st-2, ..., 1]:
        MAX_HEAPIFY(i)
    return 'Built.'
```

شماره آخرین گره غیر برگ

توجه: روال Build-Max-Heap زمان  $O(n)$  صرف می‌کند.

### سوال ۳.

ارشد ۹۵

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

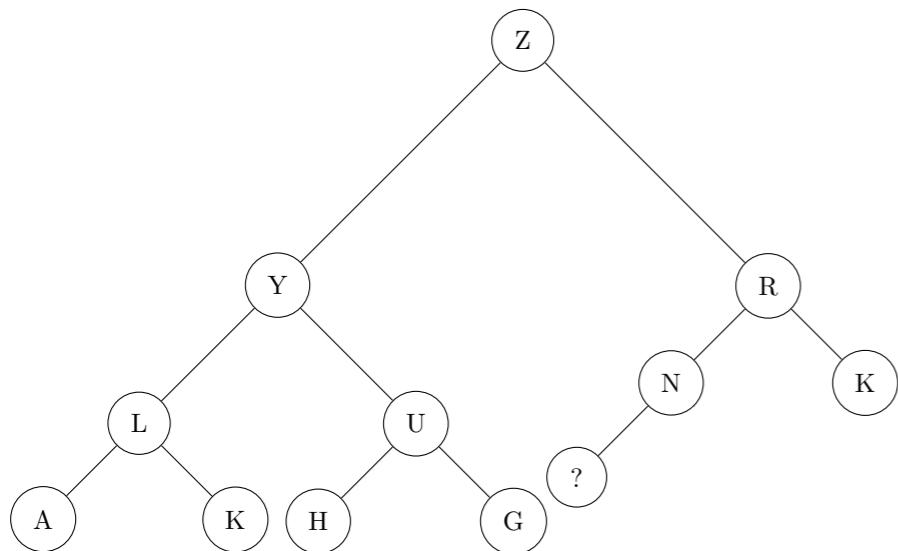
عمل حذف بیشینه روی هرم (الف) انجام شده است و درخت (ب) ساخته شده است. داده گره‌ای که با (؟) مشخص شده چه می‌تواند باشد؟

۱) یکی از I, J, K, L, M

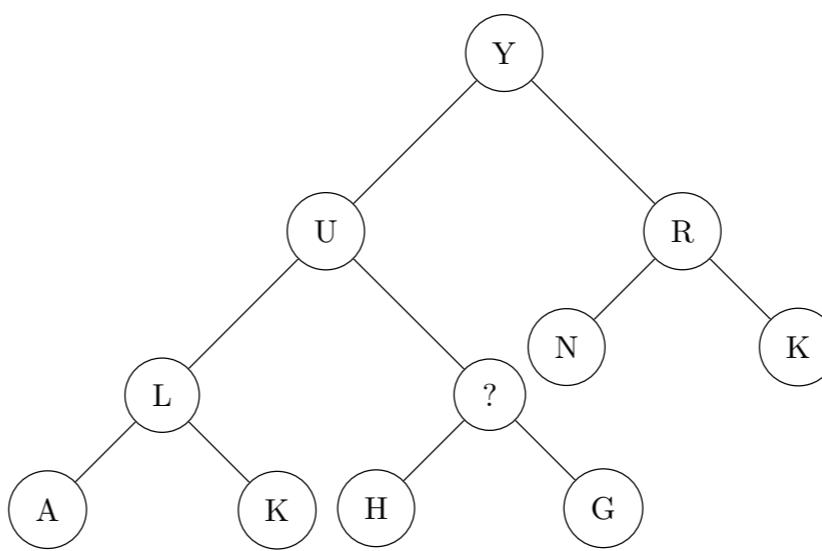
۲) یکی از H, I, J, K, L, M, N

I (۳)

M (۴)



(الف)



(ب)

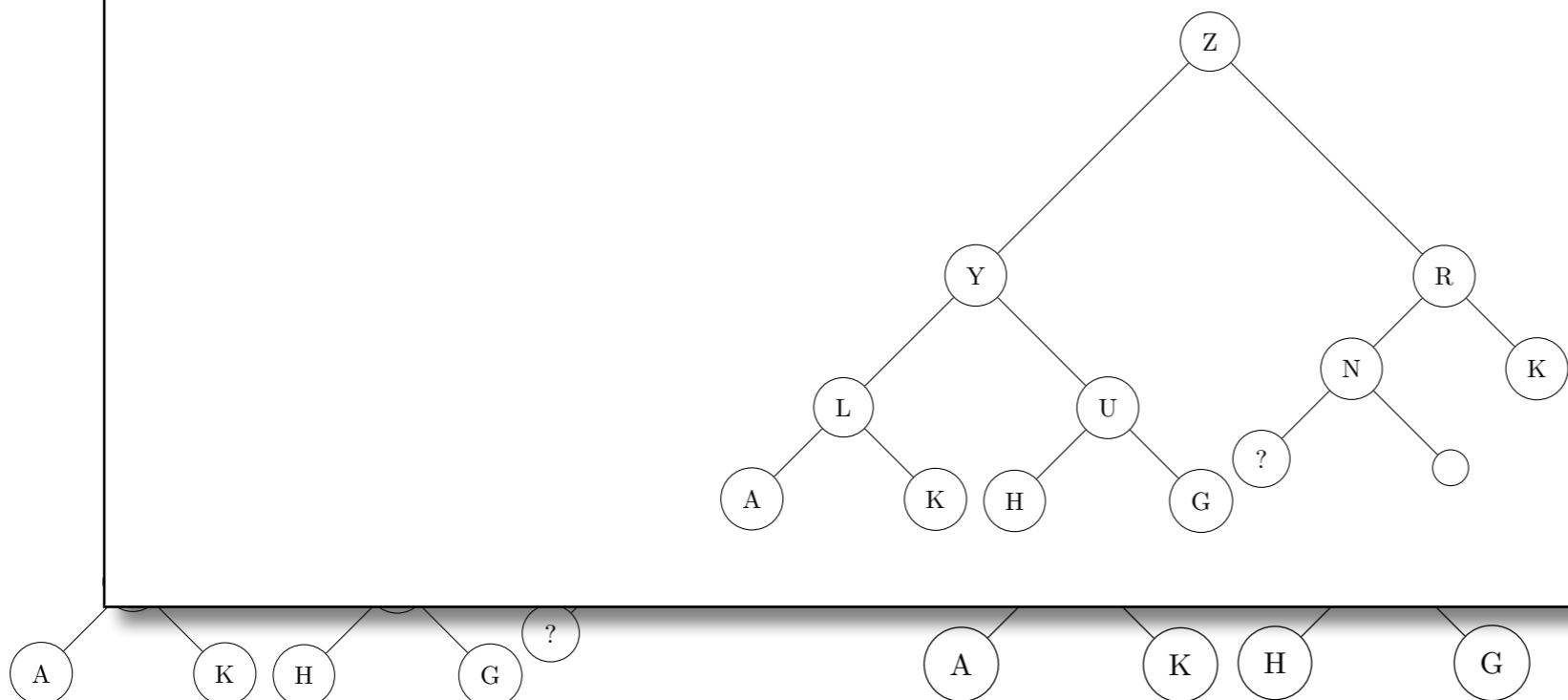
### سوال ۳.

ارشد ۹۵

عمل حذف بیشینه روی هرم (الف) انجام شده است و درخت (ب) ساخته شده است. داده گرهای که با

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

(?) م



(الف)

(ب)

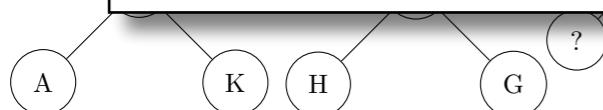
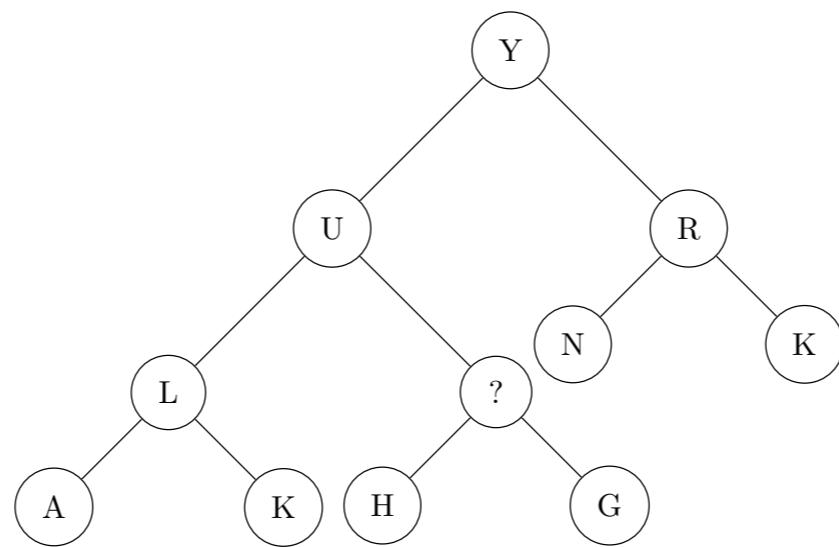
### سوال ۳.

گزینه ۲.

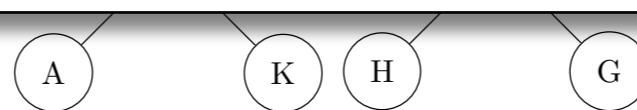
عمل حذف بیشینه روی هرم (الف) انجام شده است و درخت (ب) ساخته شده است. داده گره‌ای که با

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

(?) م



(الف)



(ب)

## سوال ۴.

ارشد ۹۲

در زیر پس از یک عمل حذف و تنظیم دوباره، خانه‌ی با اندیس شماره ۲ حاوی چه کلیدی خواهد بود؟

1	2	3	4	5	6	7	8	9	10	11	12	13
50	40	35	25	23	35	20	20	18	17	10	10	25

۴۰ (۱)

۳۰ (۲)

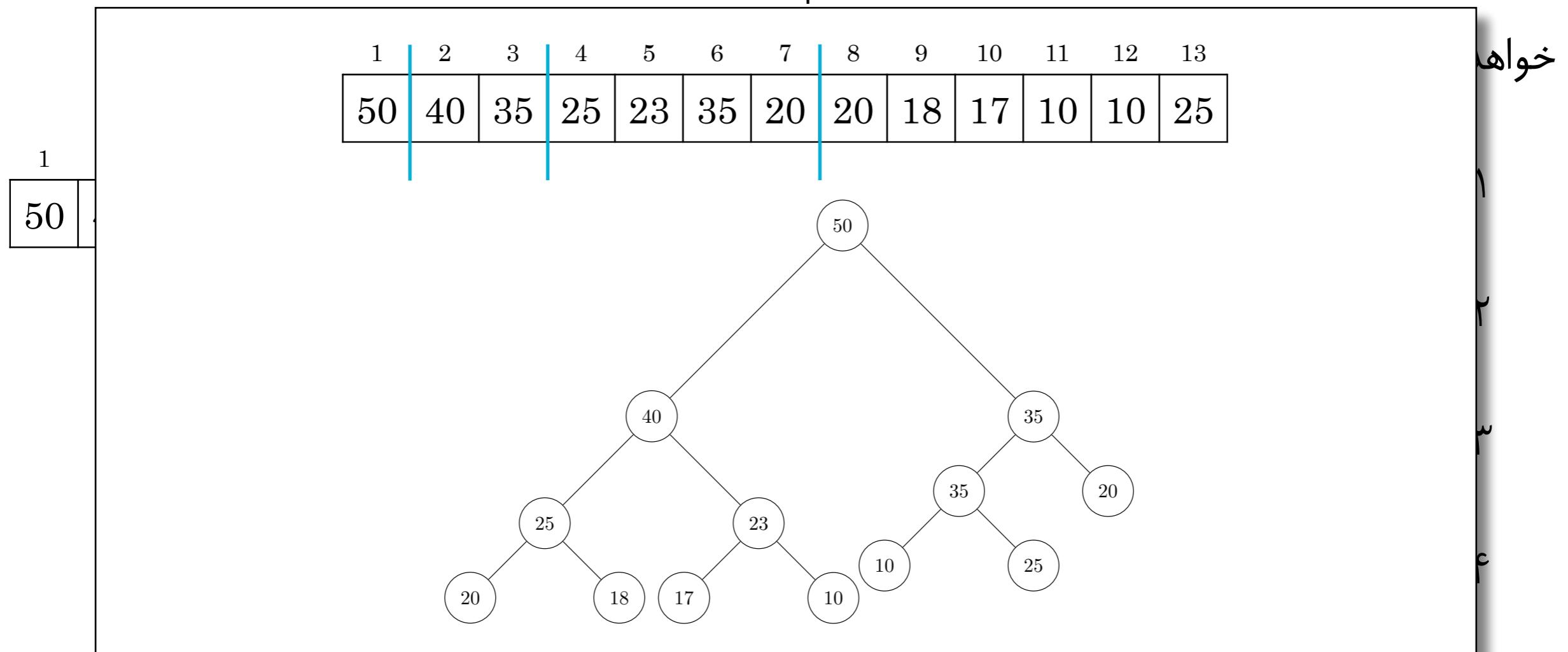
۲۵ (۳)

۲۳ (۴)

## سوال ۴.

ارشد ۹۲

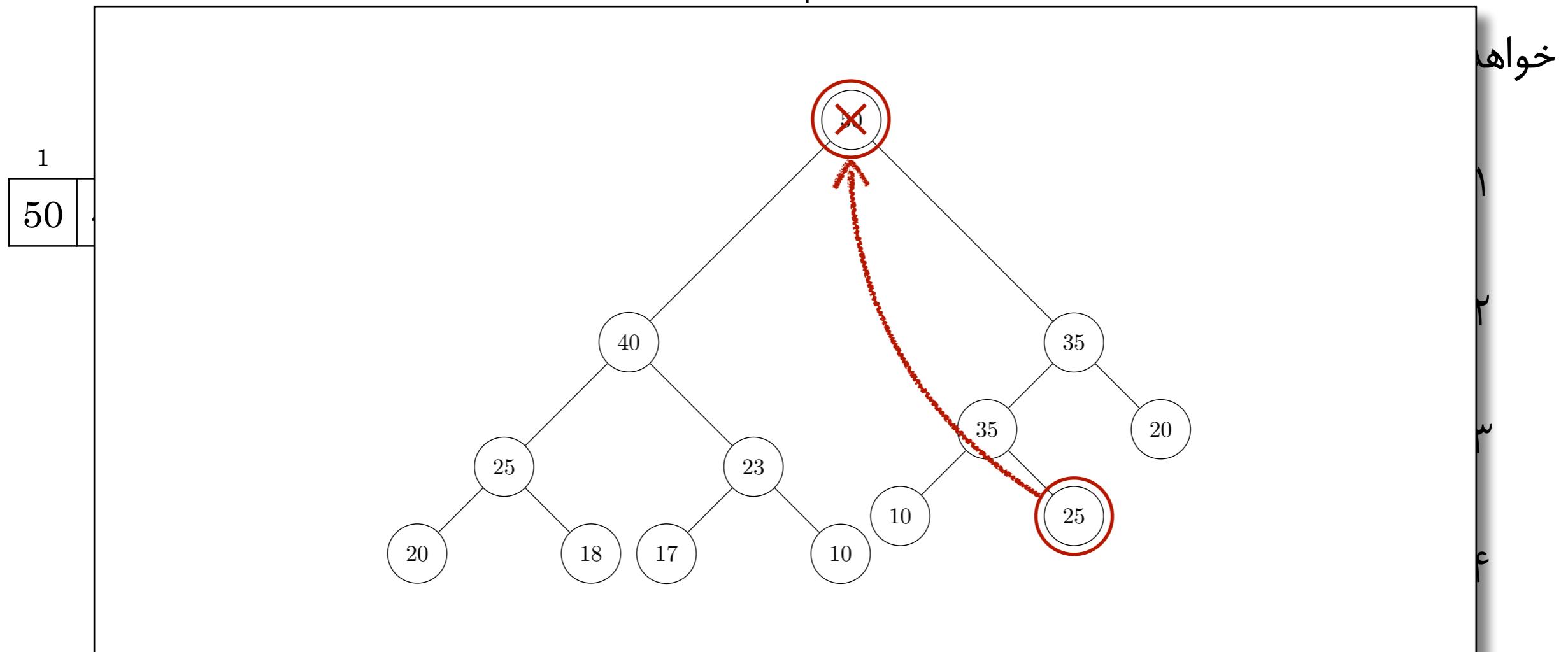
در زیر پس از یک عمل حذف و تنظیم دوباره، خانه‌ی با اندیس شماره ۲ حاوی چه کلیدی



## سوال ۴.

ارشد ۹۲

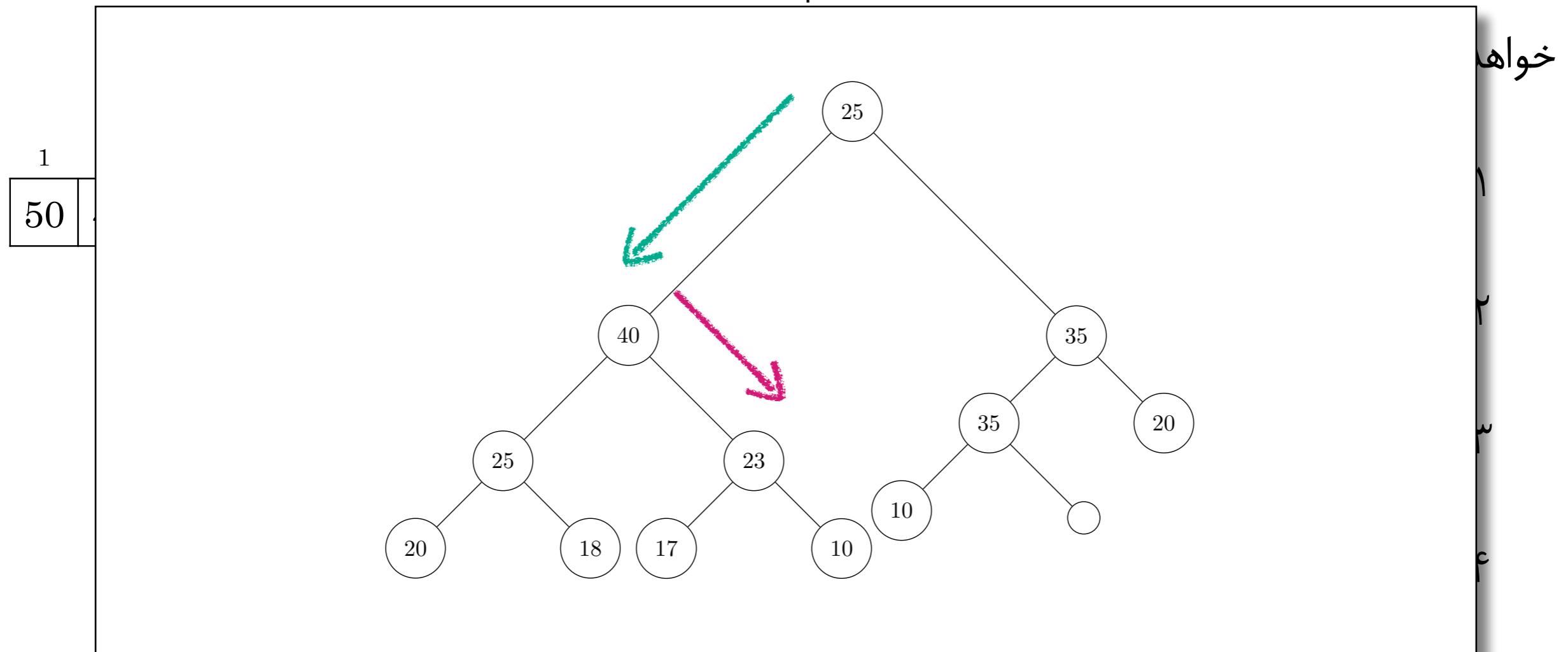
در زیر پس از یک عمل حذف و تنظیم دوباره، خانه‌ی با اندیس شماره ۲ حاوی چه کلیدی



## سوال ۴.

ارشد ۹۲

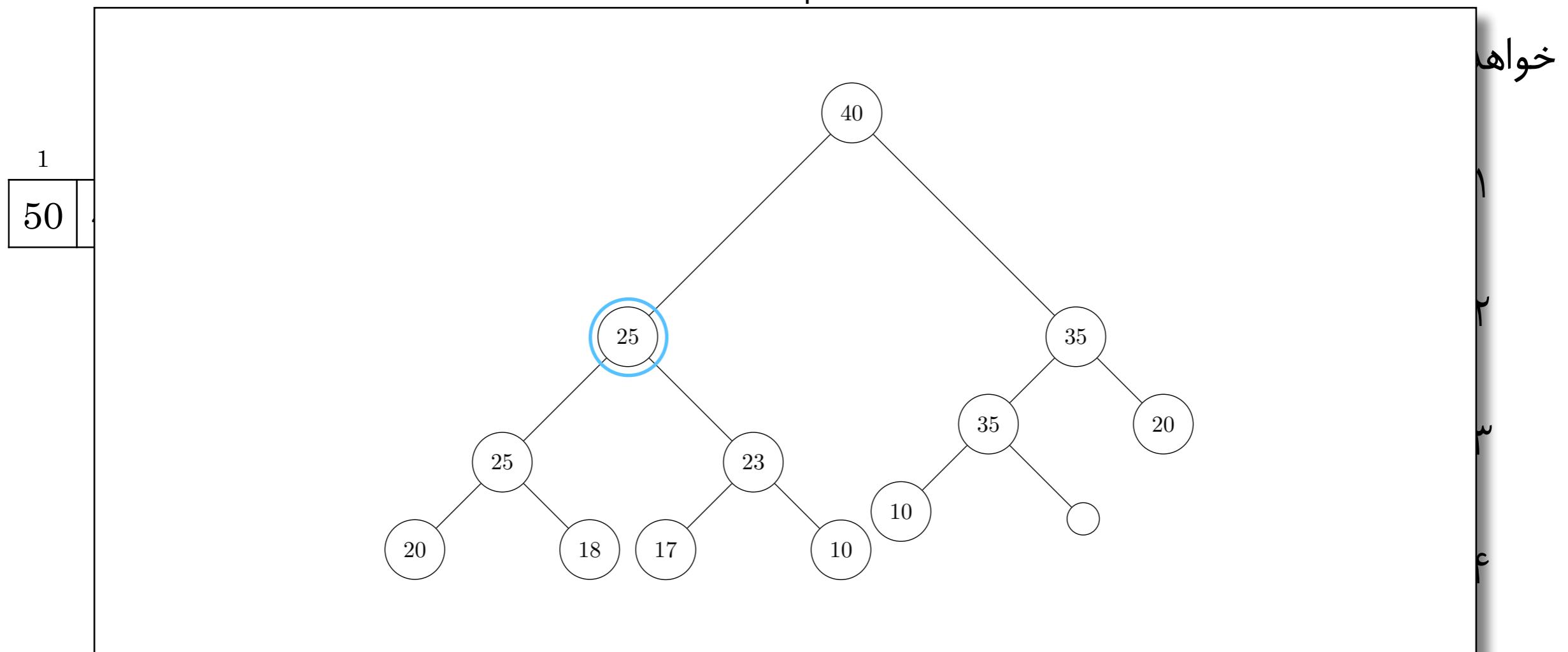
در زیر پس از یک عمل حذف و تنظیم دوباره، خانه‌ی با اندیس شماره ۲ حاوی چه کلیدی



## سوال ۴.

گزینه ۳.

در زیر پس از یک عمل حذف و تنظیم دوباره، خانه‌ی با اندیس شماره ۲ حاوی چه کلیدی



## سوال ۵.

ارشد ۹۲

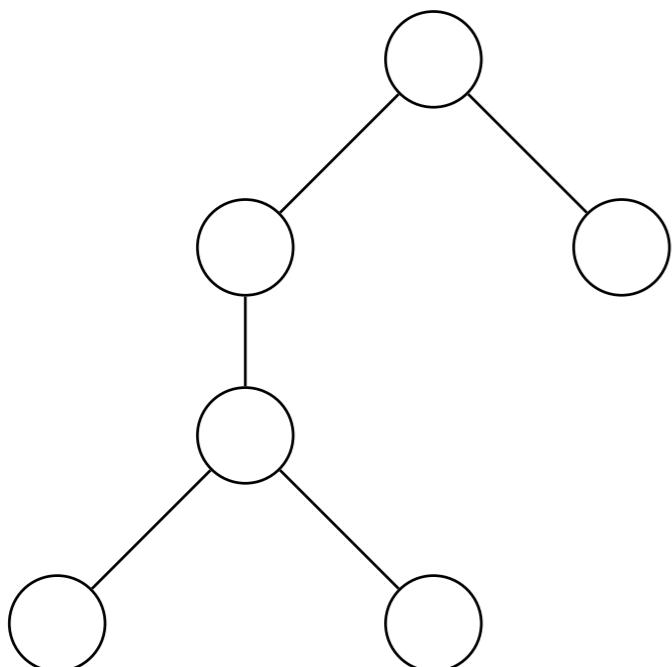
به چند طریق می‌توان اعداد ۱ تا ۶ را بدون تکرار در گره‌های درخت زیر برچسب گذاری کرد؛ بطوری که عدد هر گره از عدد فرزندانش بزرگتر باشد؟

۱۰ (۱)

۱۵ (۲)

۱۲ (۳)

۸ (۴)



## سوال ۵.

ارشد ۹۲

به چند طریق می‌توان اعداد ۱ تا ۶ را بدون تکرار در گره‌های درخت زیر برچسب گذاری کرد؛ بطوری

۶ از همه اعداد بزرگتر است، پس تنها در ریشه می‌تواند قرار گیرد.

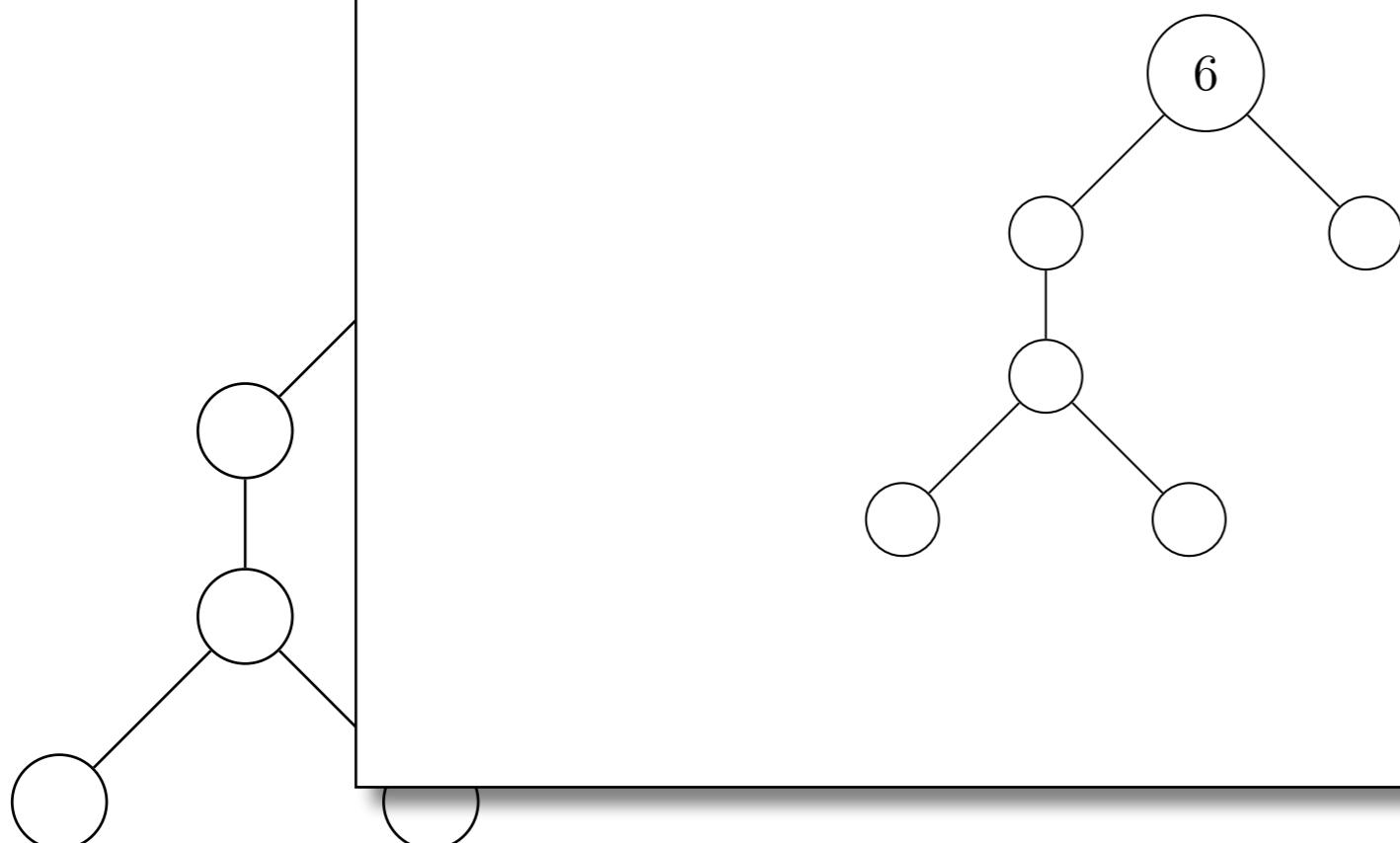
که عدد هر گره از

۱۰ (۱)

۱۵ (۲)

۱۲ (۳)

۸ (۴)



## سوال ۵.

ارشد ۹۲

به چند طریق می‌توان اعداد ۱ تا ۶ را بدون تکرار در گره‌های درخت زیر برچسب گذاری کرد؛ بطوری

از بین ۵ عدد باقی مانده یکی را باید انتخاب کنیم.

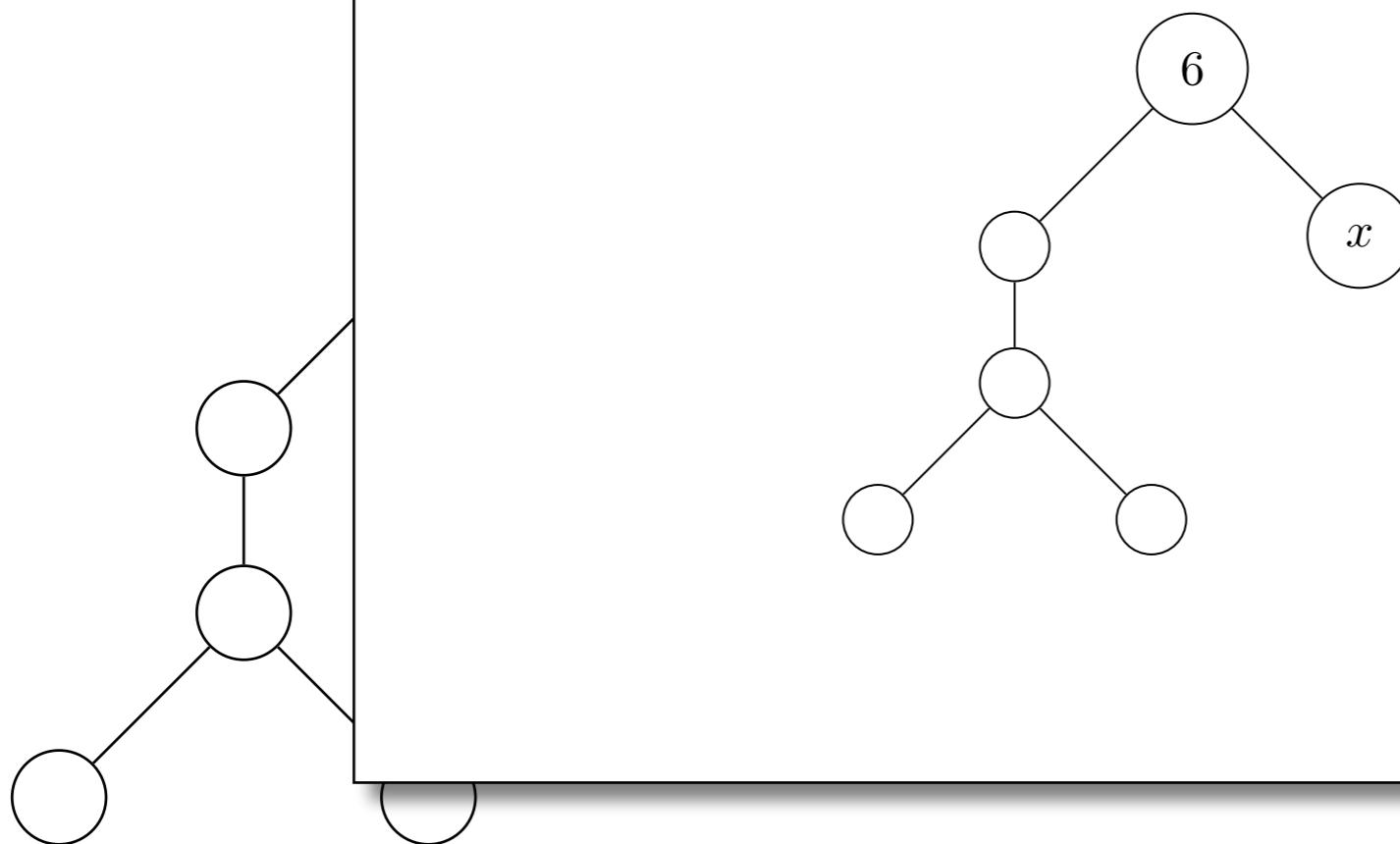
که عدد هر گره از

۱۰ (۱)

۱۵ (۲)

۱۲ (۳)

۸ (۴)



## سوال ۵.

ارشد ۹۲

به چند طریق می‌توان اعداد ۱ تا ۶ را بدون تکرار در گره‌های درخت زیر برچسب گذاری کرد؛ بطوری

که عدد هر گره از

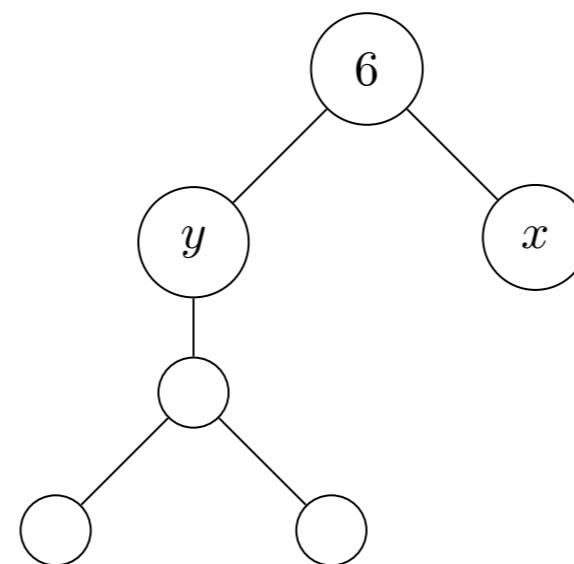
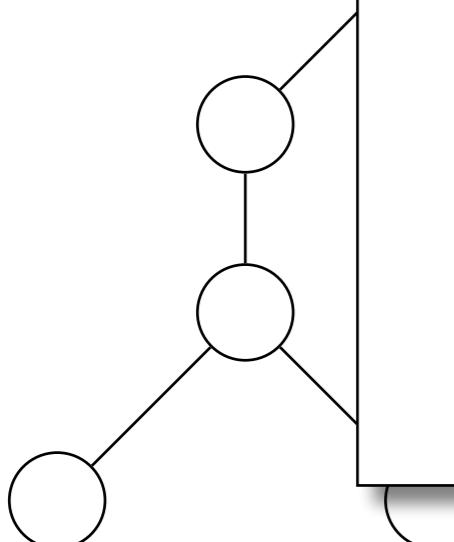
۱۰ (۱)

۱۵ (۲)

۱۲ (۳)

۸ (۴)

و باید برابر با بیشینه ۴ عدد باقی‌مانده باشد.



## سوال ۵.

ارشد ۹۲

به چند طریق می‌توان اعداد ۱ تا ۶ را بدون تکرار در گره‌های درخت زیر برچسب گذاری کرد؛ بطوری

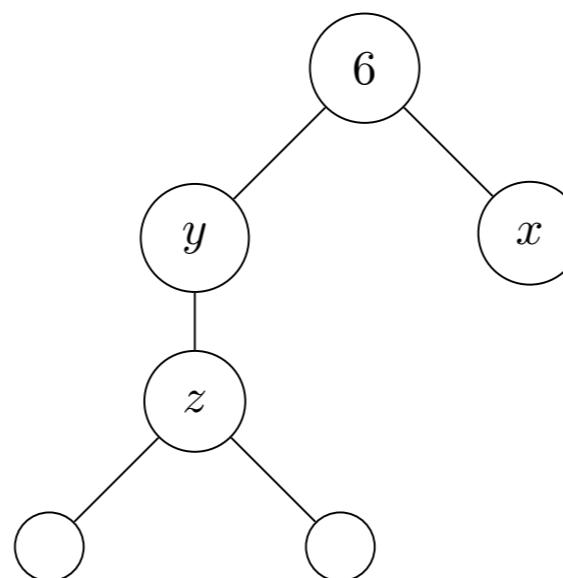
که عدد هر گره از

۱۰ (۱)

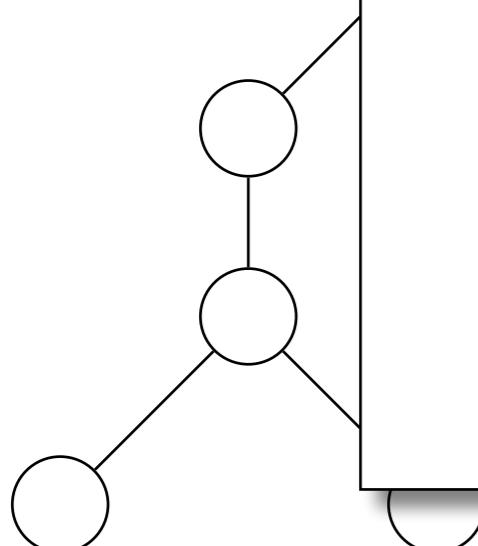
۱۵ (۲)

۱۲ (۳)

۸ (۴)



به طور مشابه،  $z$  نیز برابر با بیشینه ۳ عدد باقی‌مانده است.



## سوال ۵.

گزینه ۱.

به چند طریق می‌توان اعداد ۱ تا ۶ را بدون تکرار در گره‌های درخت زیر برچسب گذاری کرد؛ بطوری

که عدد هر گره از

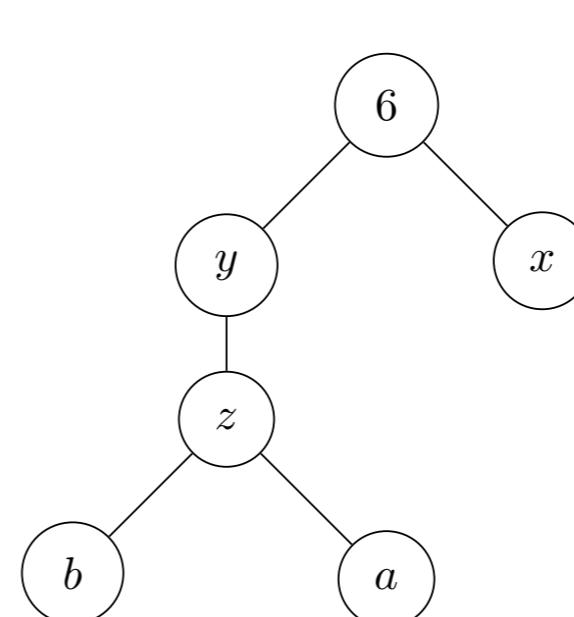
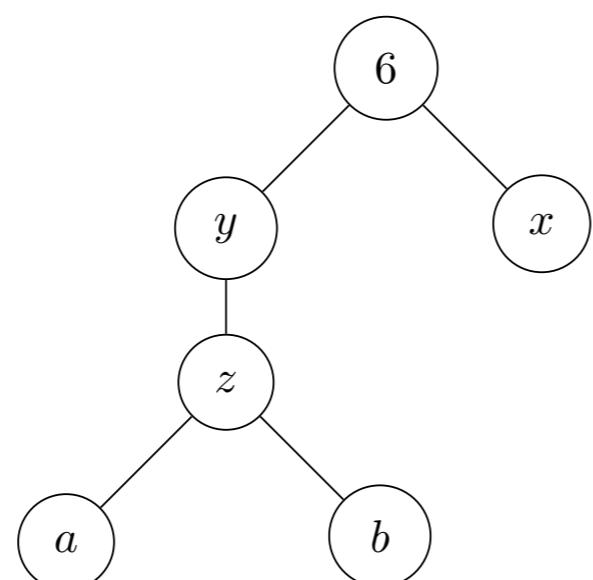
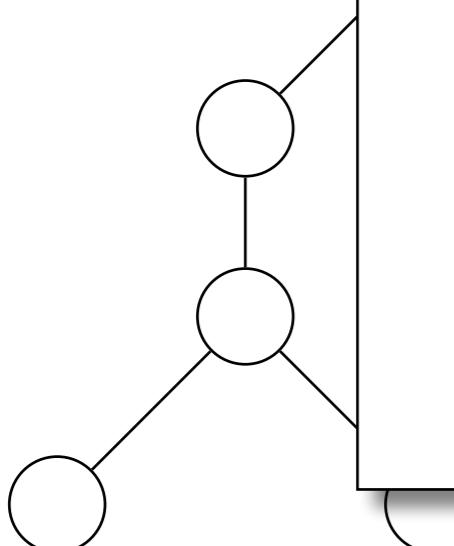
۱۰ (۱)

۱۵ (۲)

۱۲ (۳)

۸ (۴)

دو عدد باقی مانده به دلخواه می‌توانند قرار گیرند.



# سوال؟

جادب نیکو

وقت خالی شدم نیست هنوز؟! همه‌ی حافظه‌ام، پُر شده است.





دانشگاه شهید بهشتی کرمان

# درخت جستجو دودویی خود-متوازن

ساختمان های داده و الگوریتم

# درخت جستجو دودویی

## Binary search tree

Type tree

Invented 1960

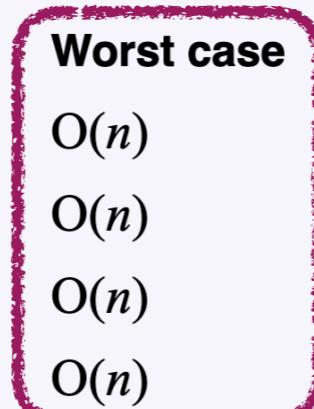
Invented P.F. Windley, A.D. Booth, A.J.T.  
by Colin, and T.N. Hibbard

### 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<sup>II</sup> 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	$O(n)$	$O(n)$
-------	--------	--------

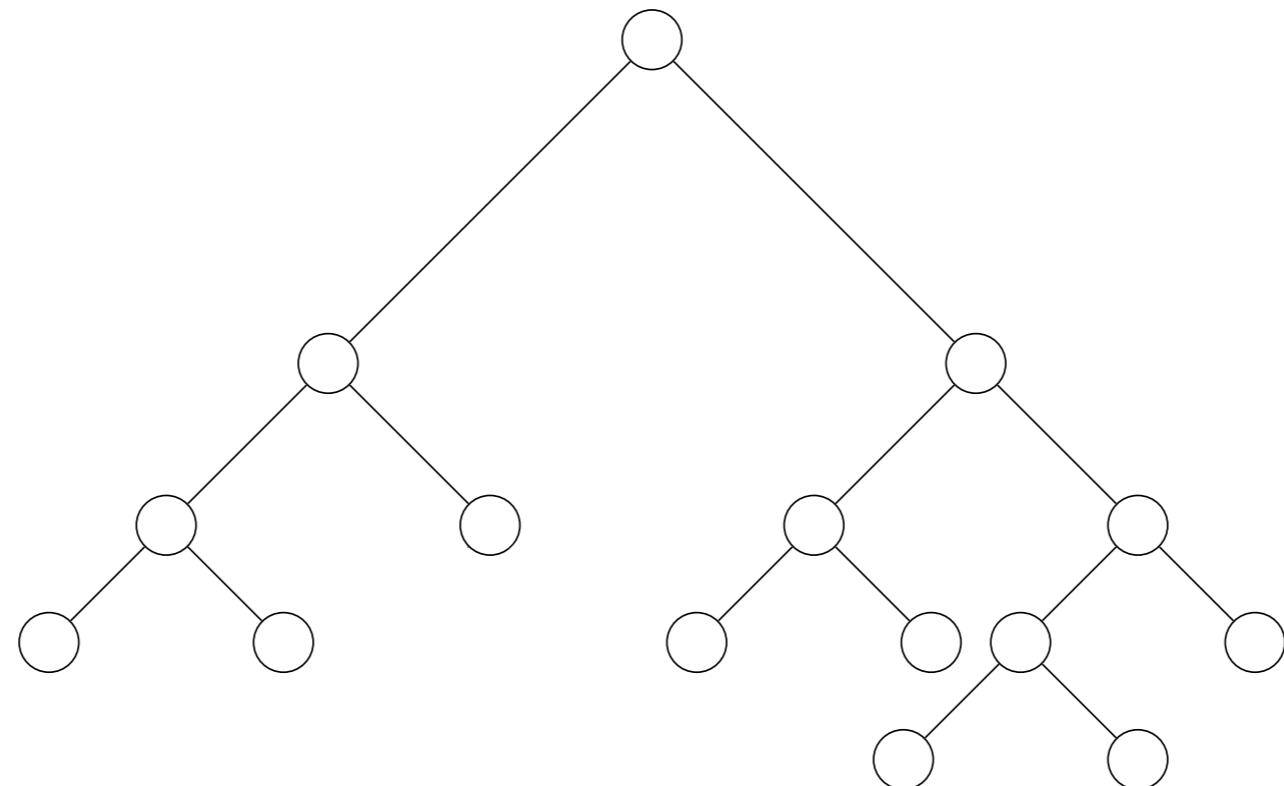
Search	$O(\log n)$	$O(\log n)$
--------	-------------	-------------

Insert	$O(\log n)$	$O(\log n)$
--------	-------------	-------------

Delete	$O(\log n)$	$O(\log n)$
--------	-------------	-------------

# درخت AVL

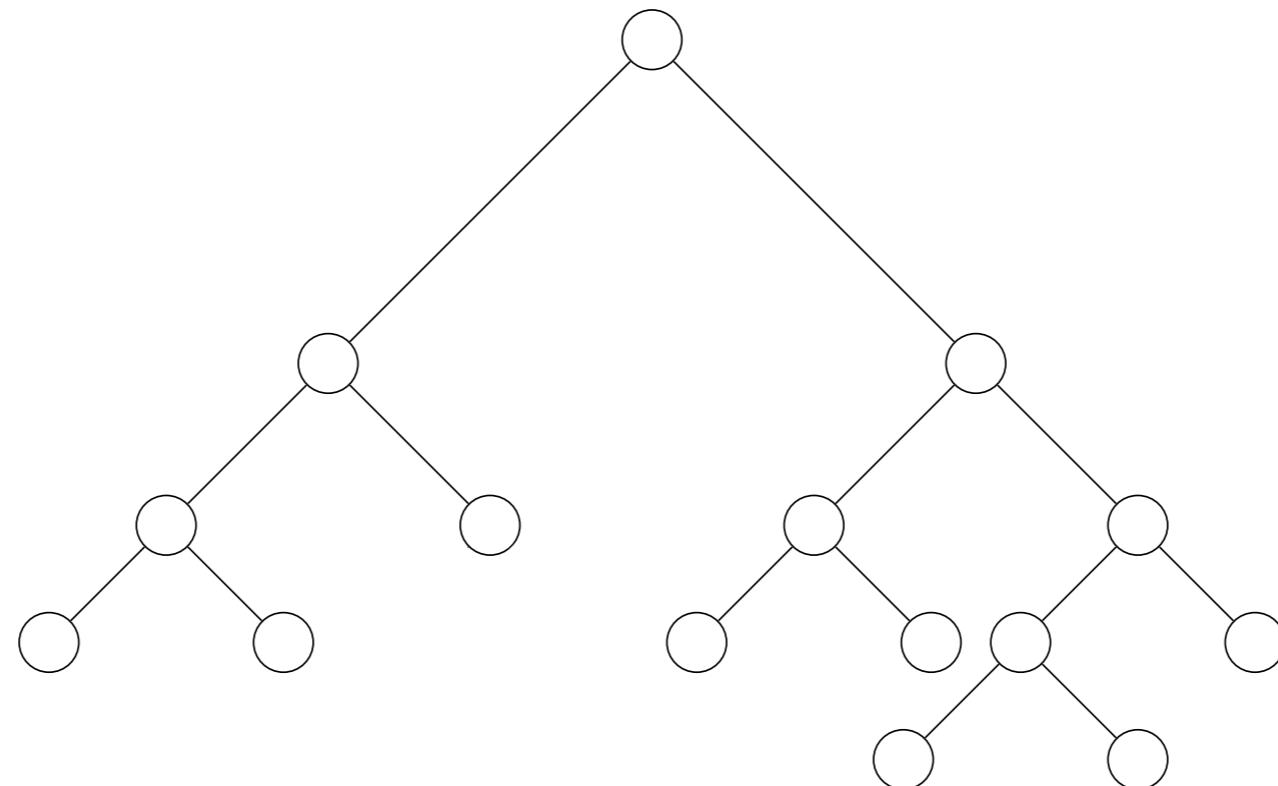
Adelson-Velsky and Landis



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

# درخت AVL

Adelson-Velsky and Landis

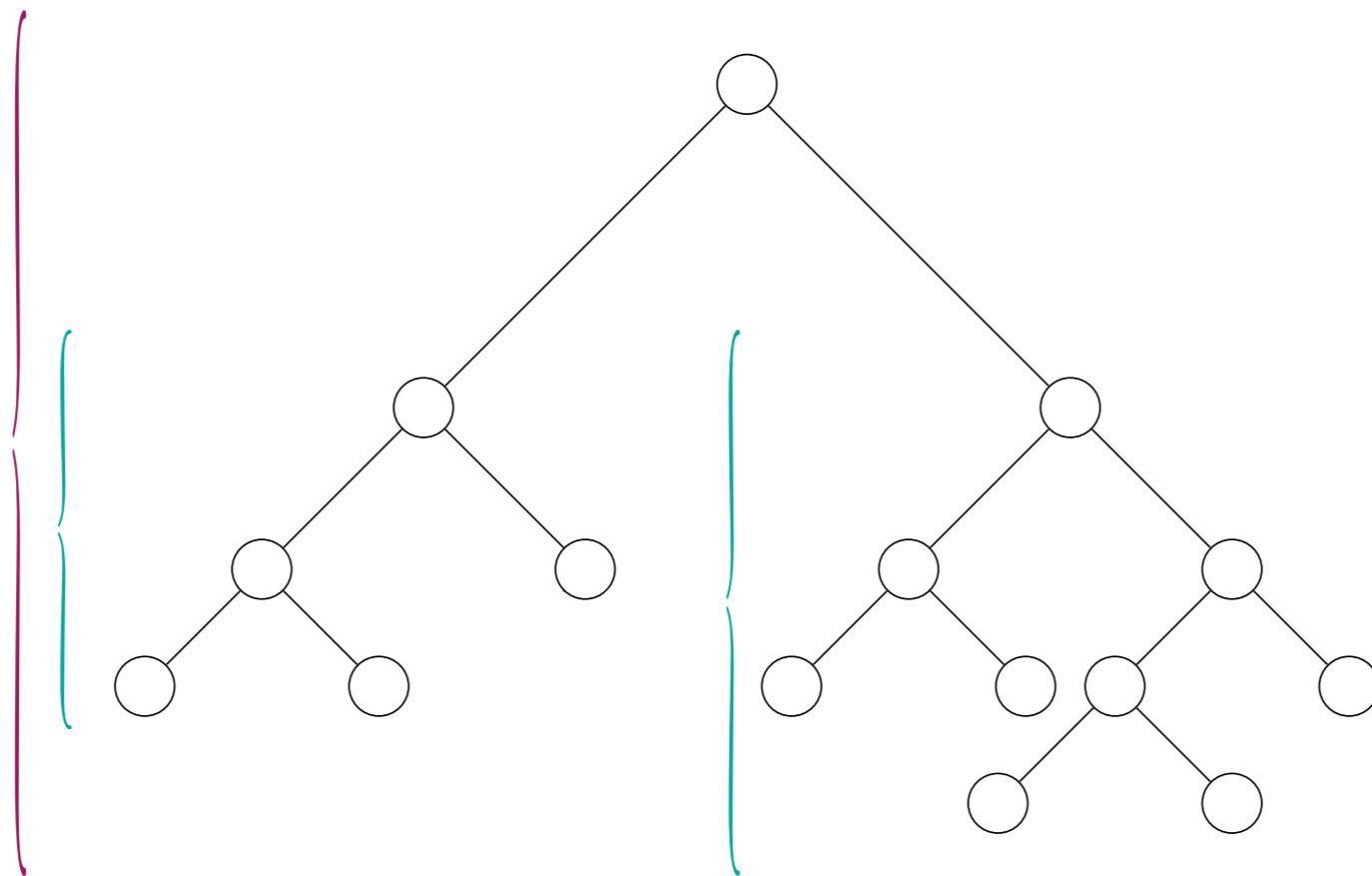


اصلًا متوازن یعنی چی؟

# AVL درخت

Adelson-Velsky and Landis

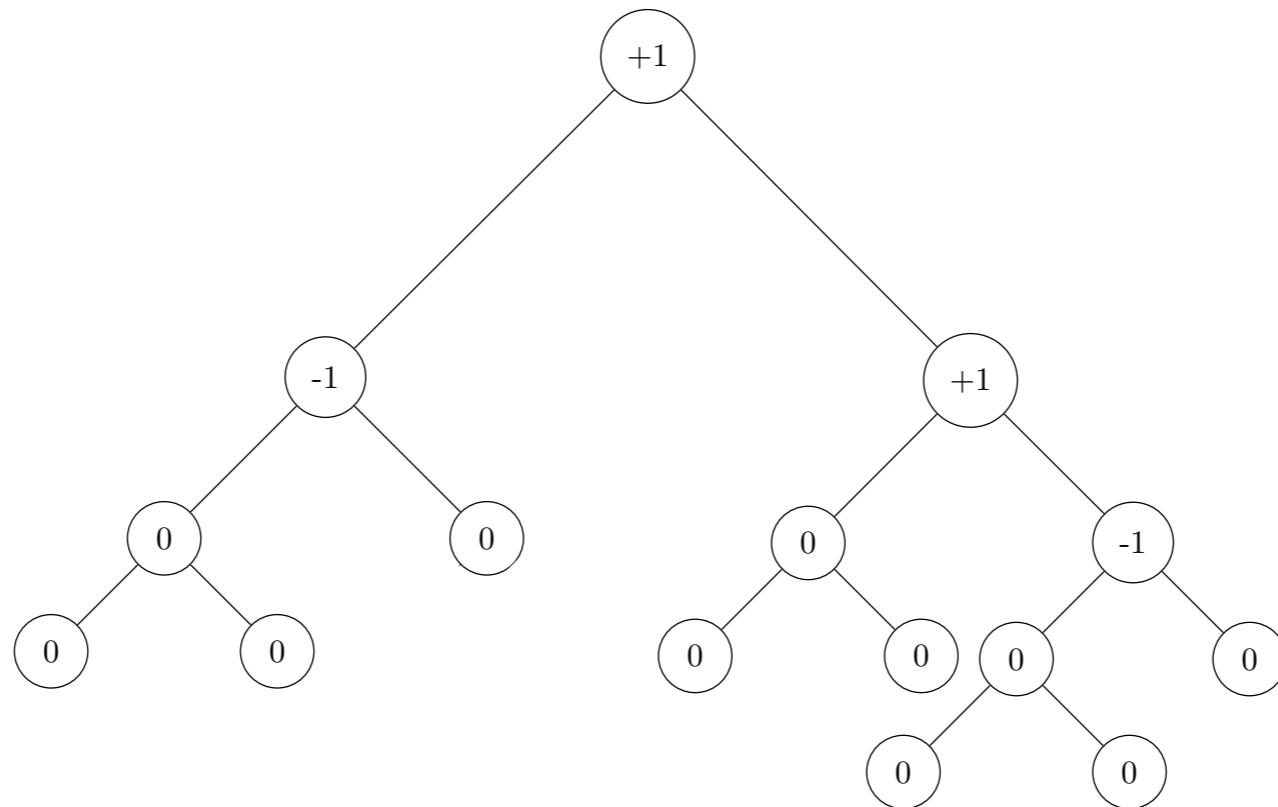
- $\text{BF}(X) := \text{Height}(\text{RightSubtree}(X)) - \text{Height}(\text{LeftSubtree}(X))$
- $\text{BF}(X) \in \{-1,0,1\}$



# AVL درخت

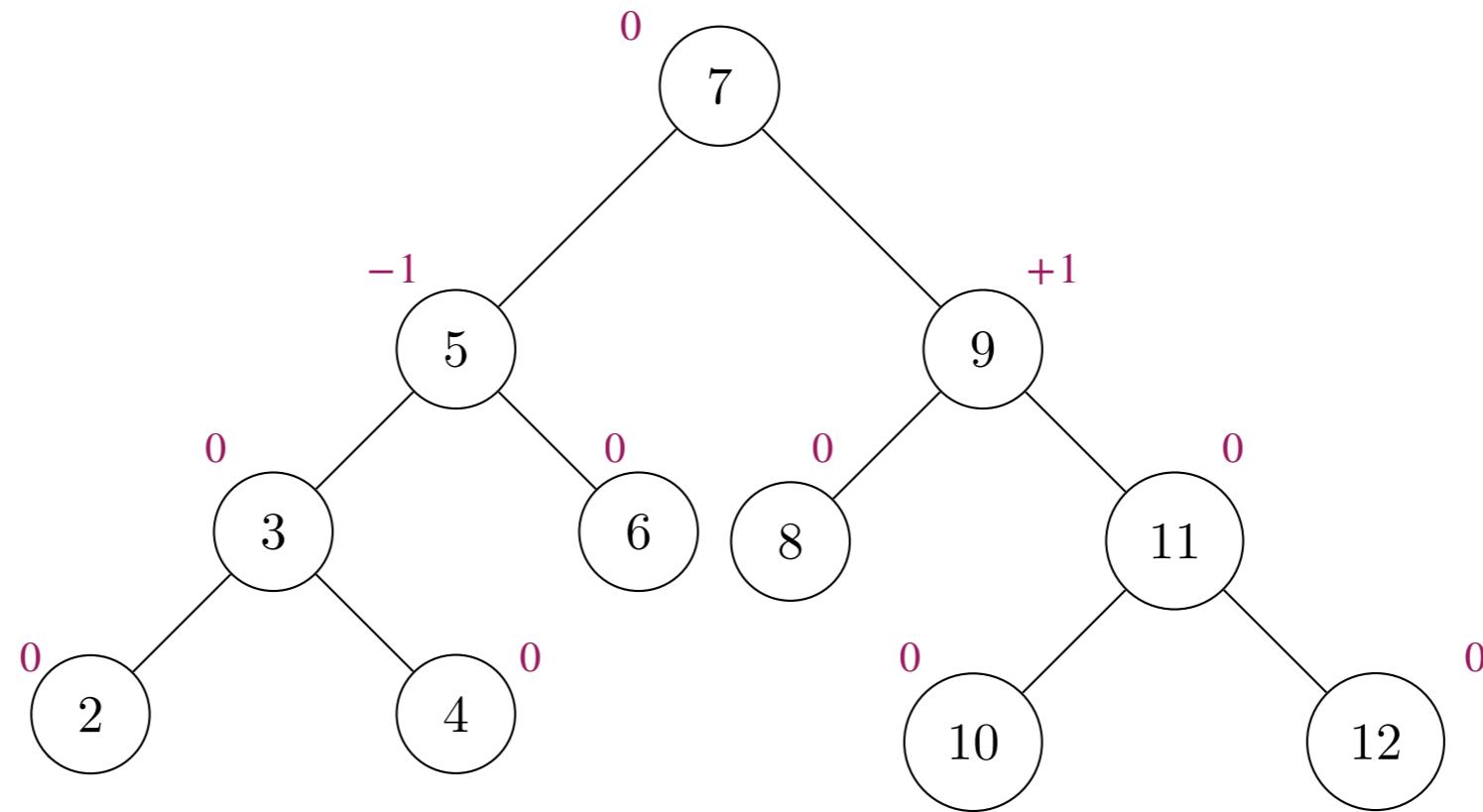
**Adelson-Velsky and Landis**

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



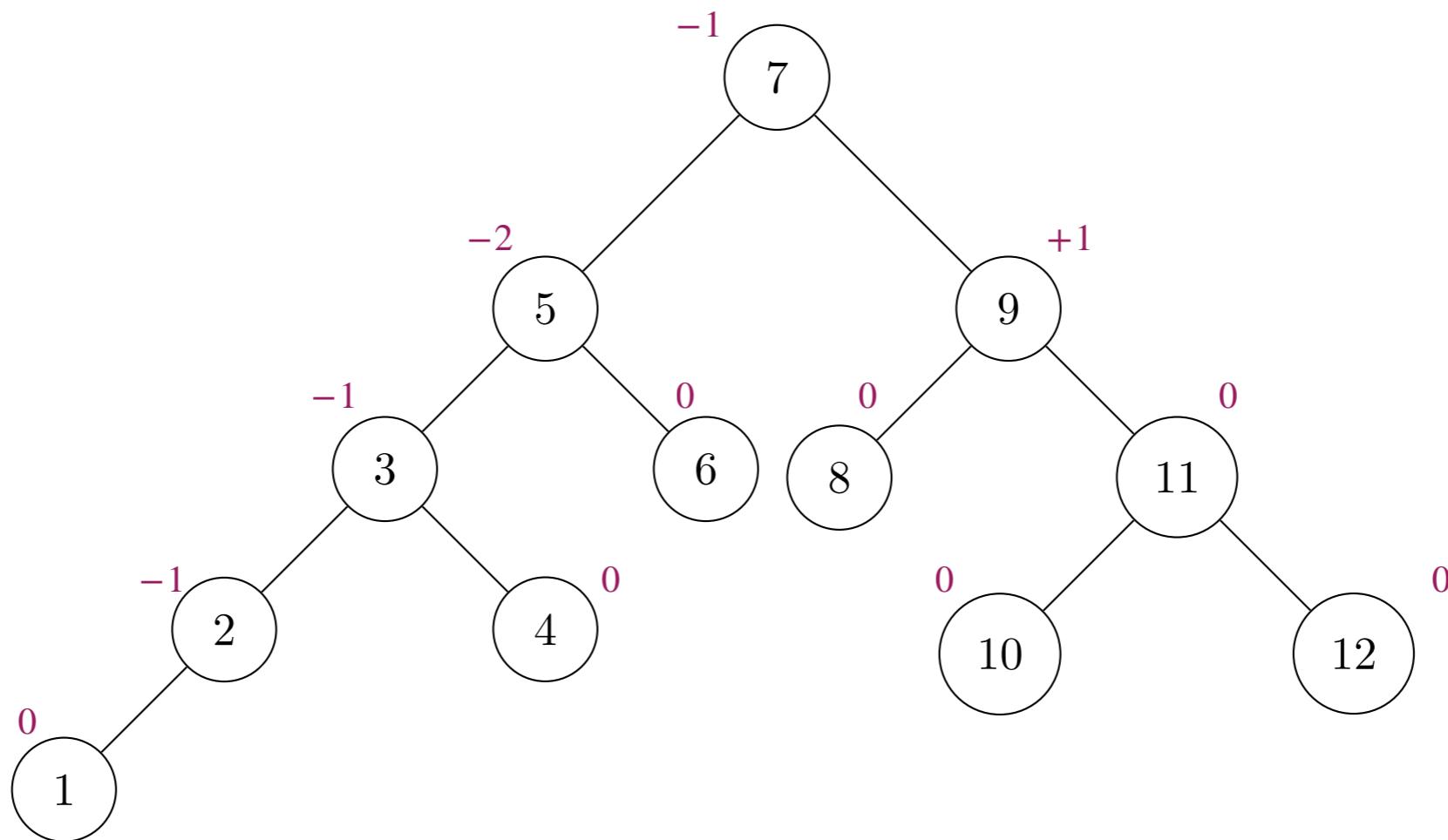
# عملیات چرخش

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



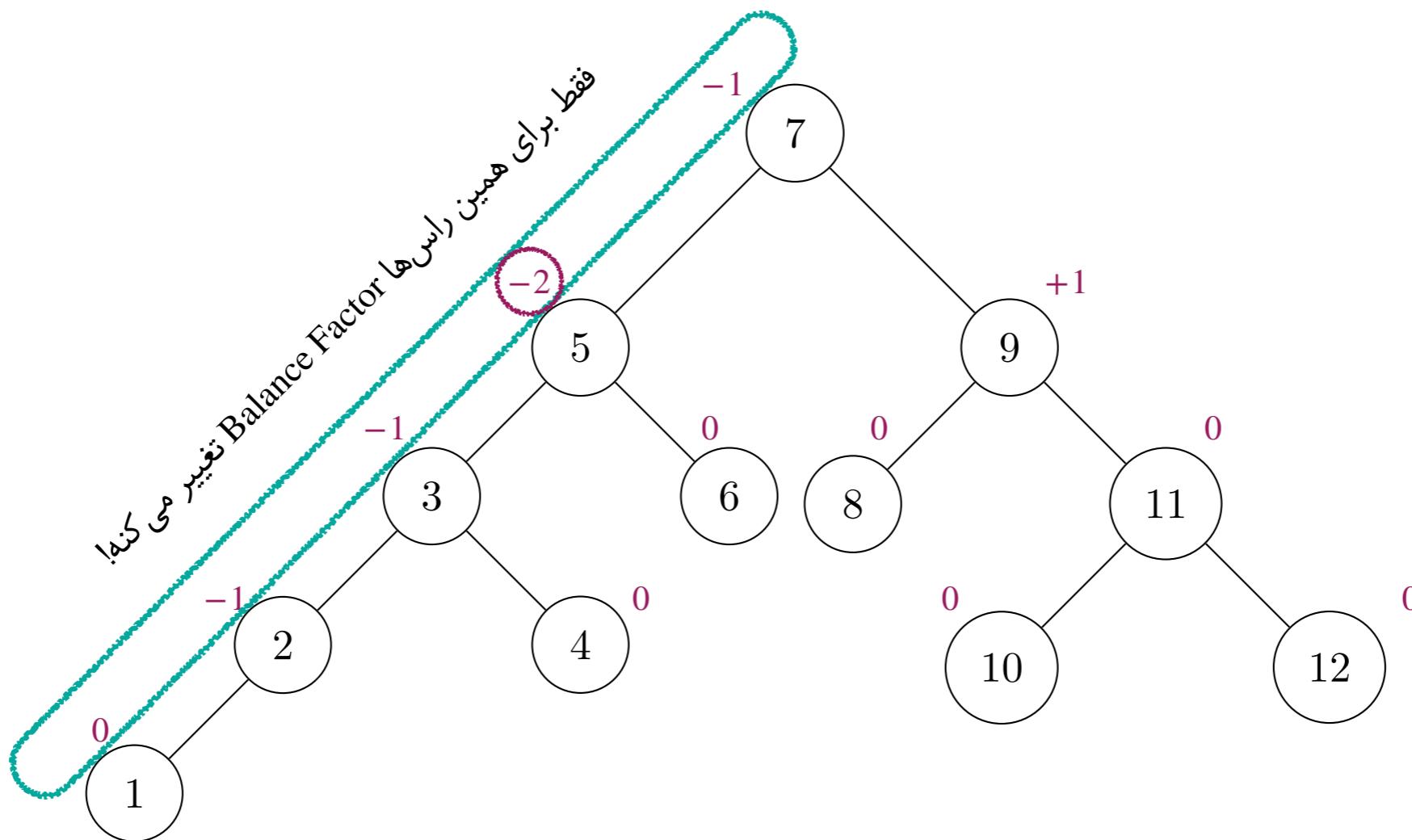
# عملیات چرخش

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



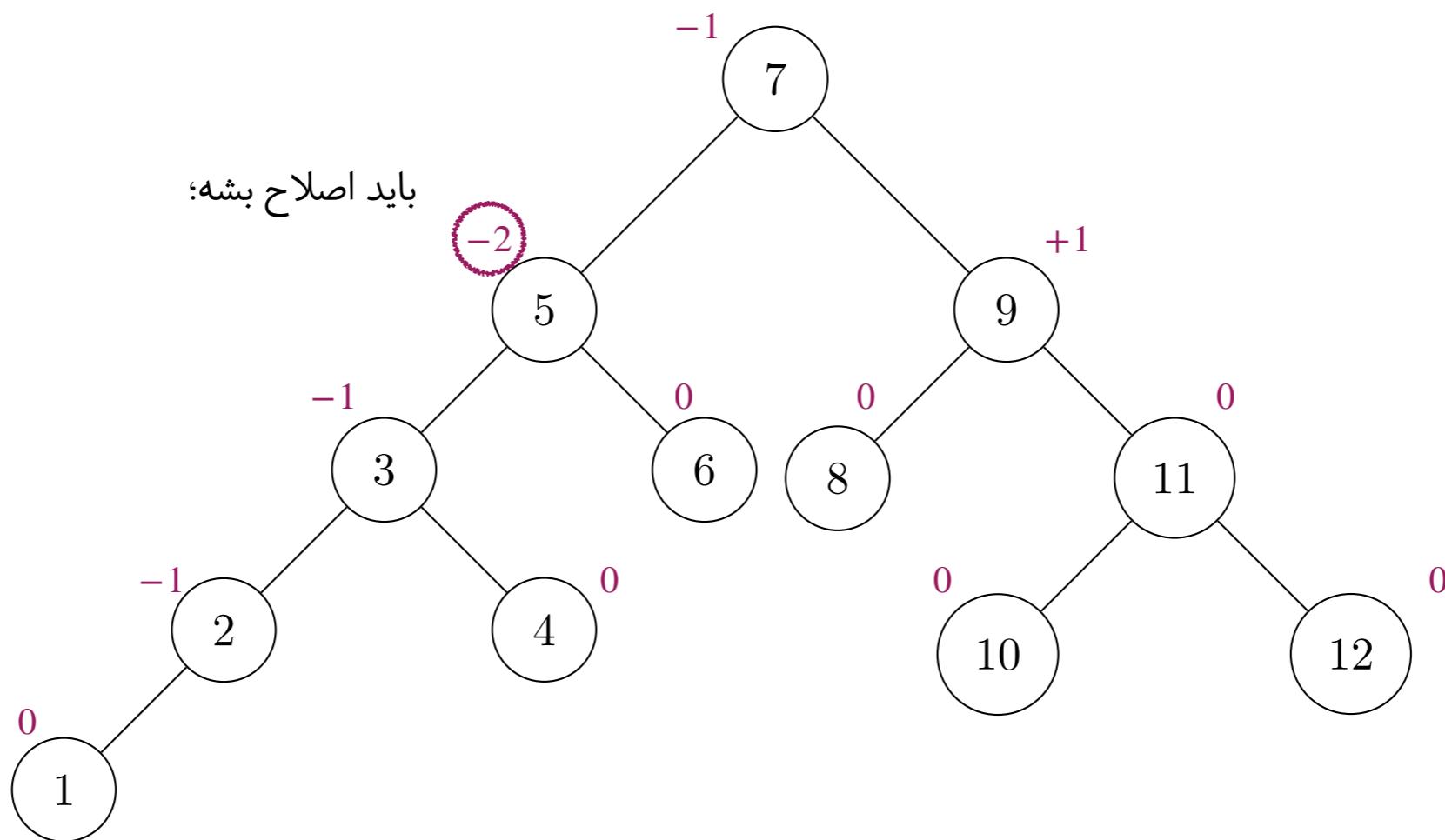
# عملیات چرخش

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



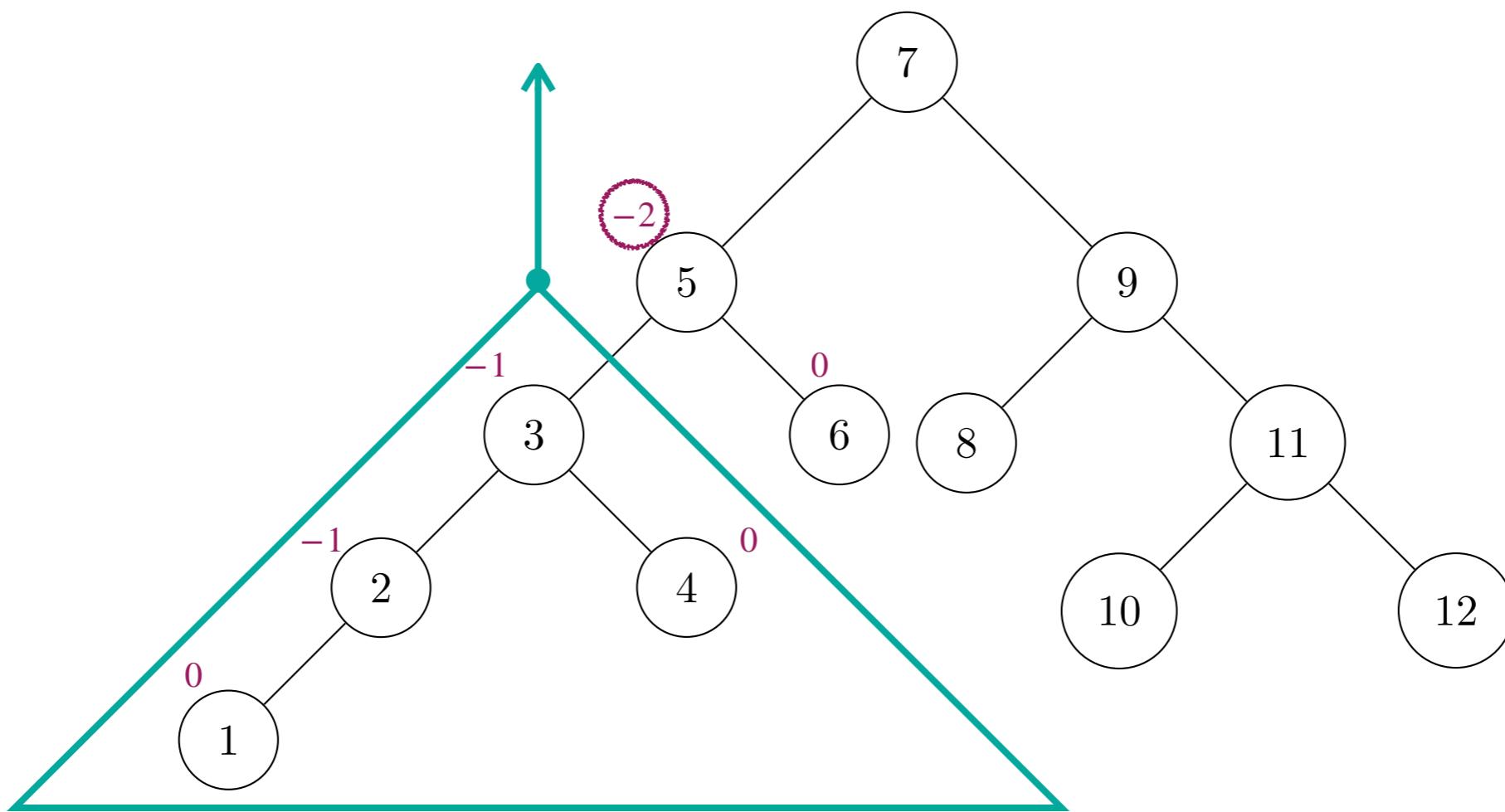
# عملیات چرخش

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



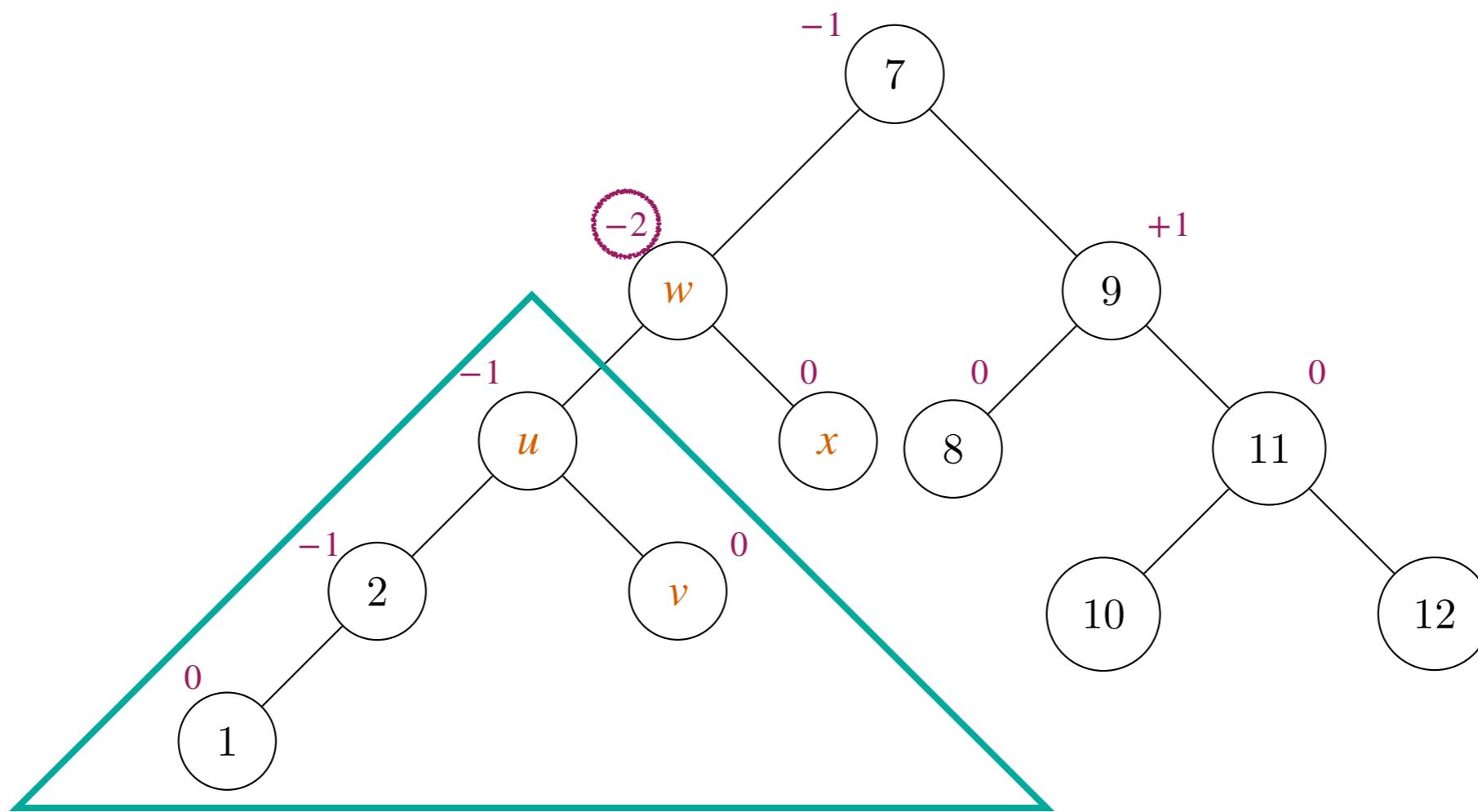
# عملیات چرخش

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



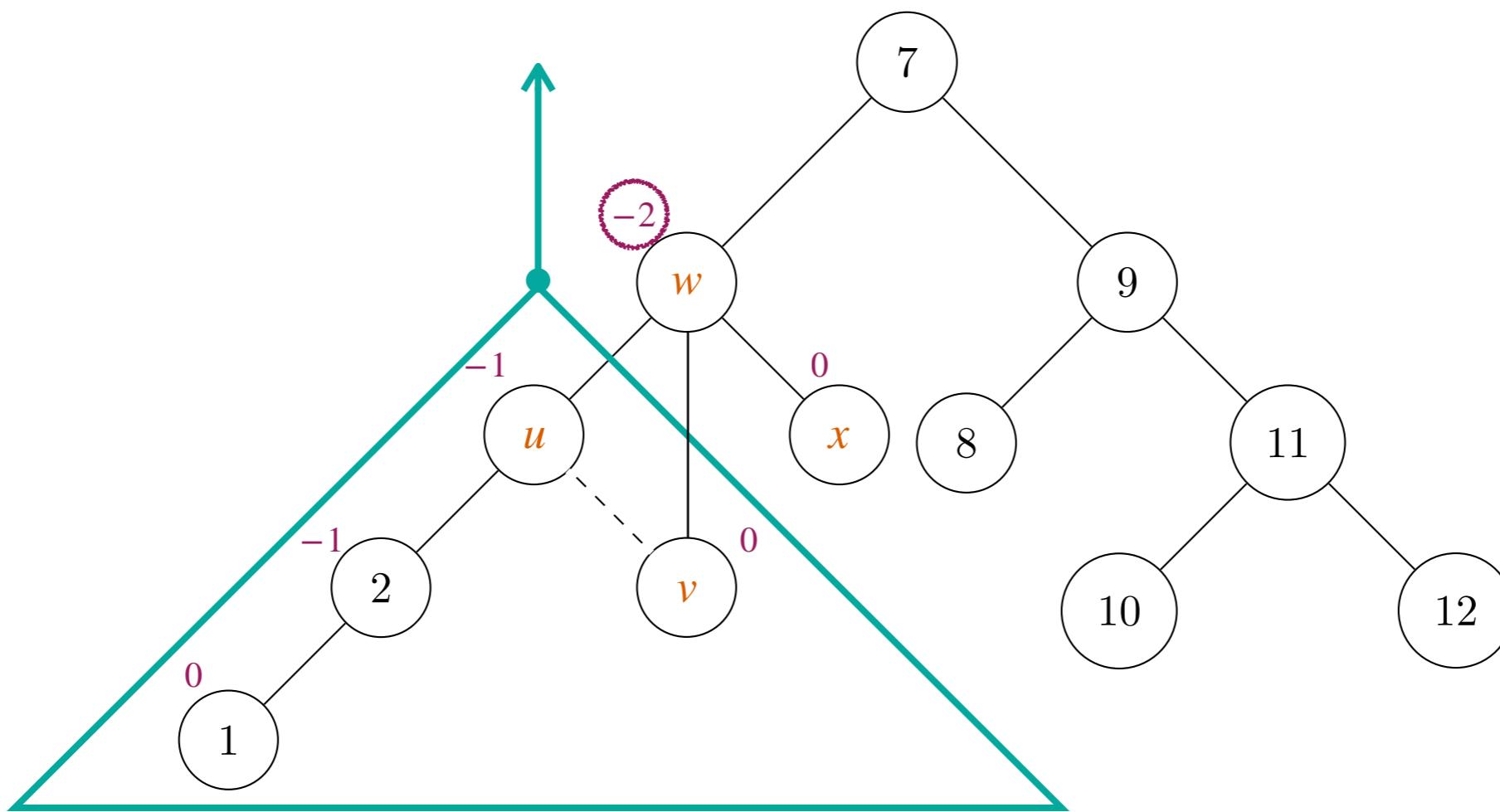
# عملیات چرخش

با توجه به تعریف درخت جستجو دودویی داریم:



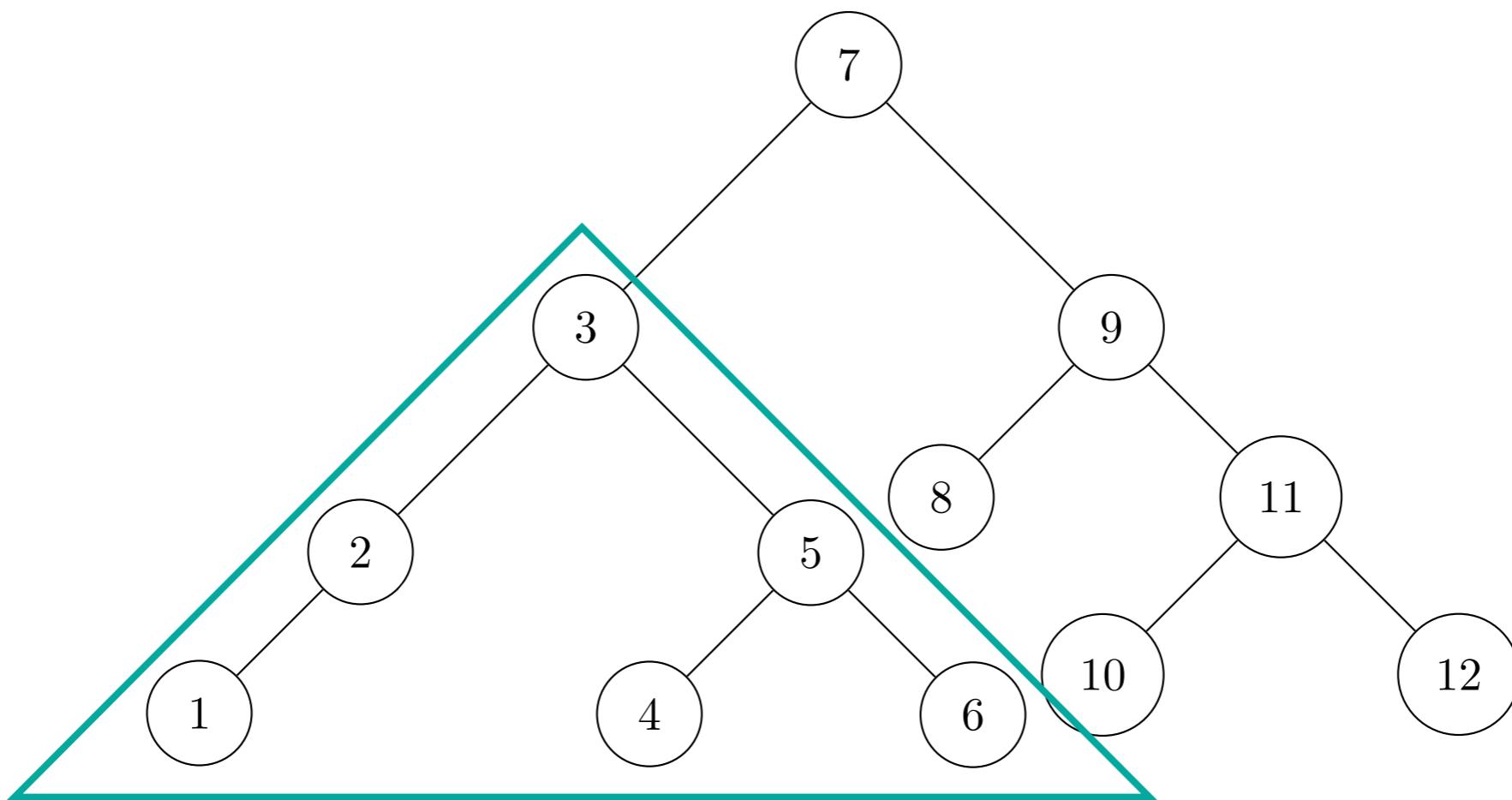
# عملیات چرخش

با توجه به تعریف درخت جستجو دودویی داریم:



# عملیات چرخش

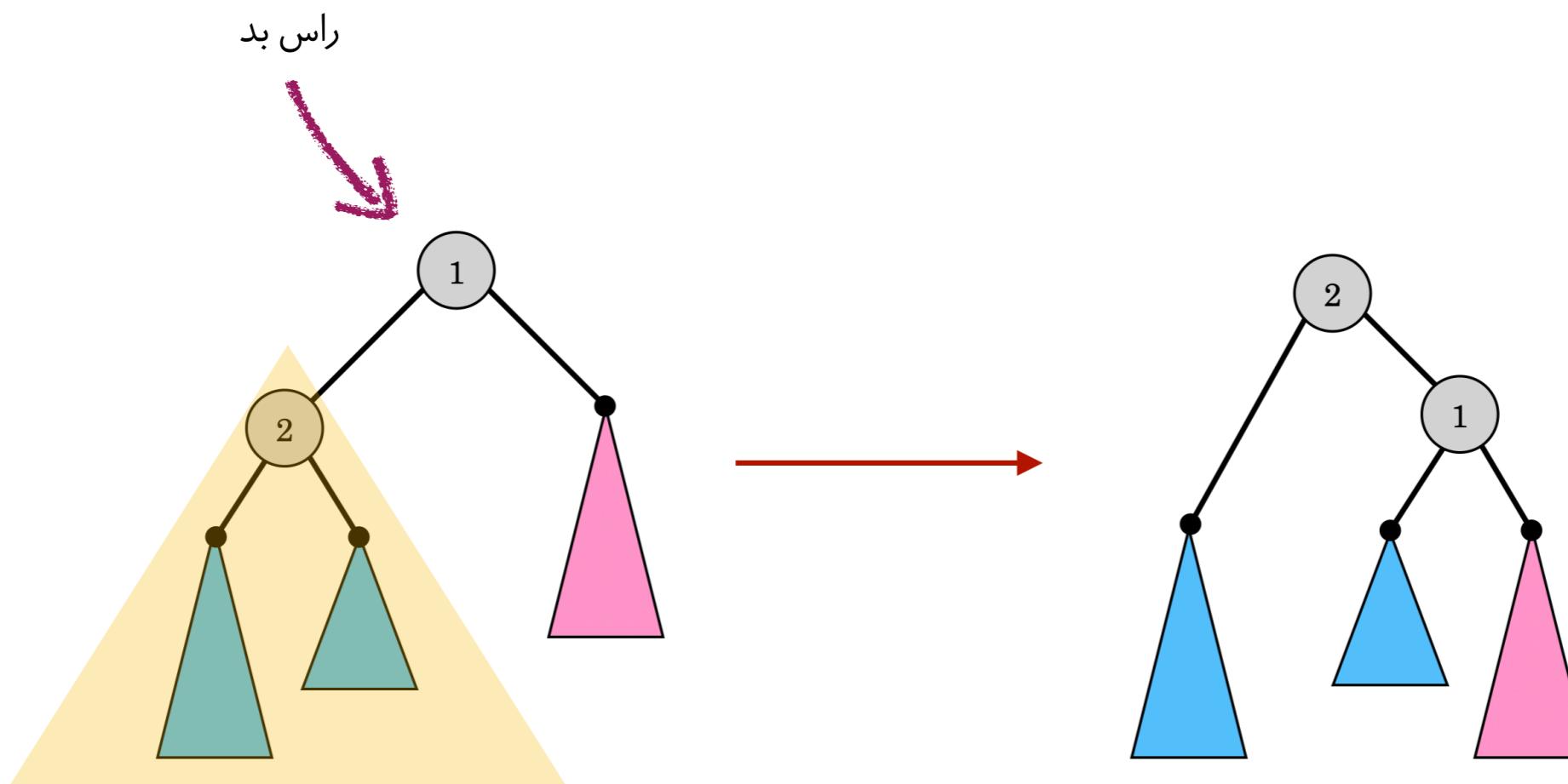
و درخت متوازن شد.



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

# عملیات چرخش

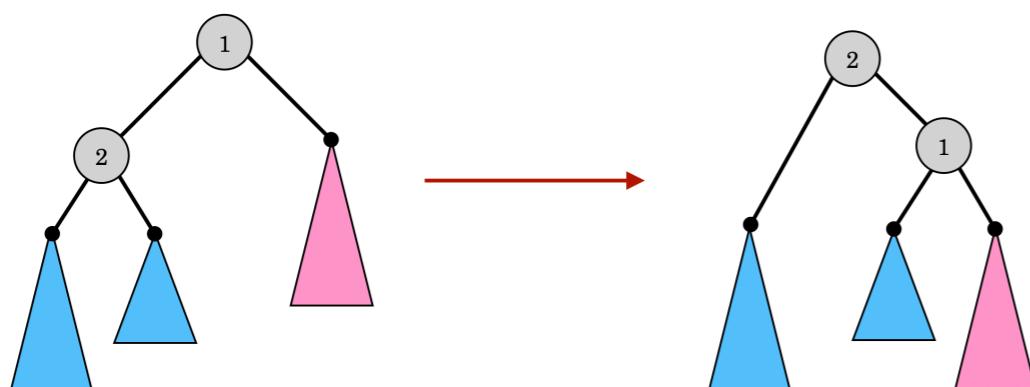
در یک نگاه



# 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)

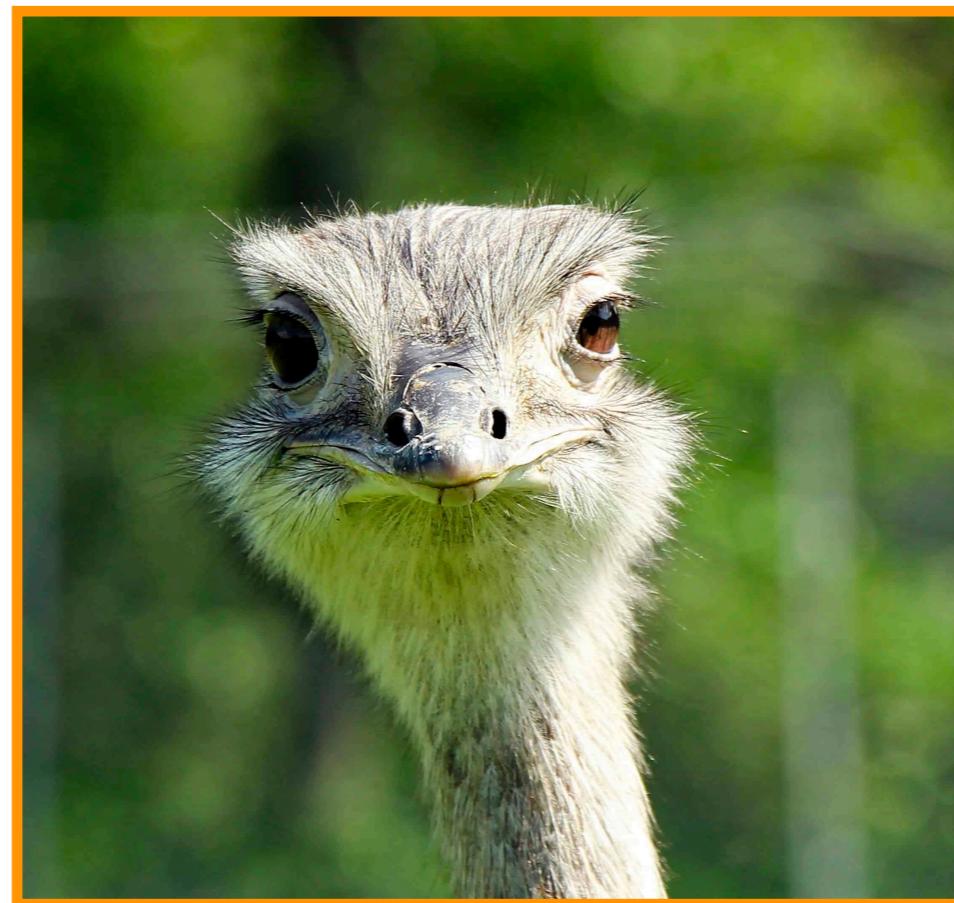
    return root

```

زیر درخت چپ عمیق تره

زیر درخت راست عمیق تره

سؤال؟



## چند سوال.

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