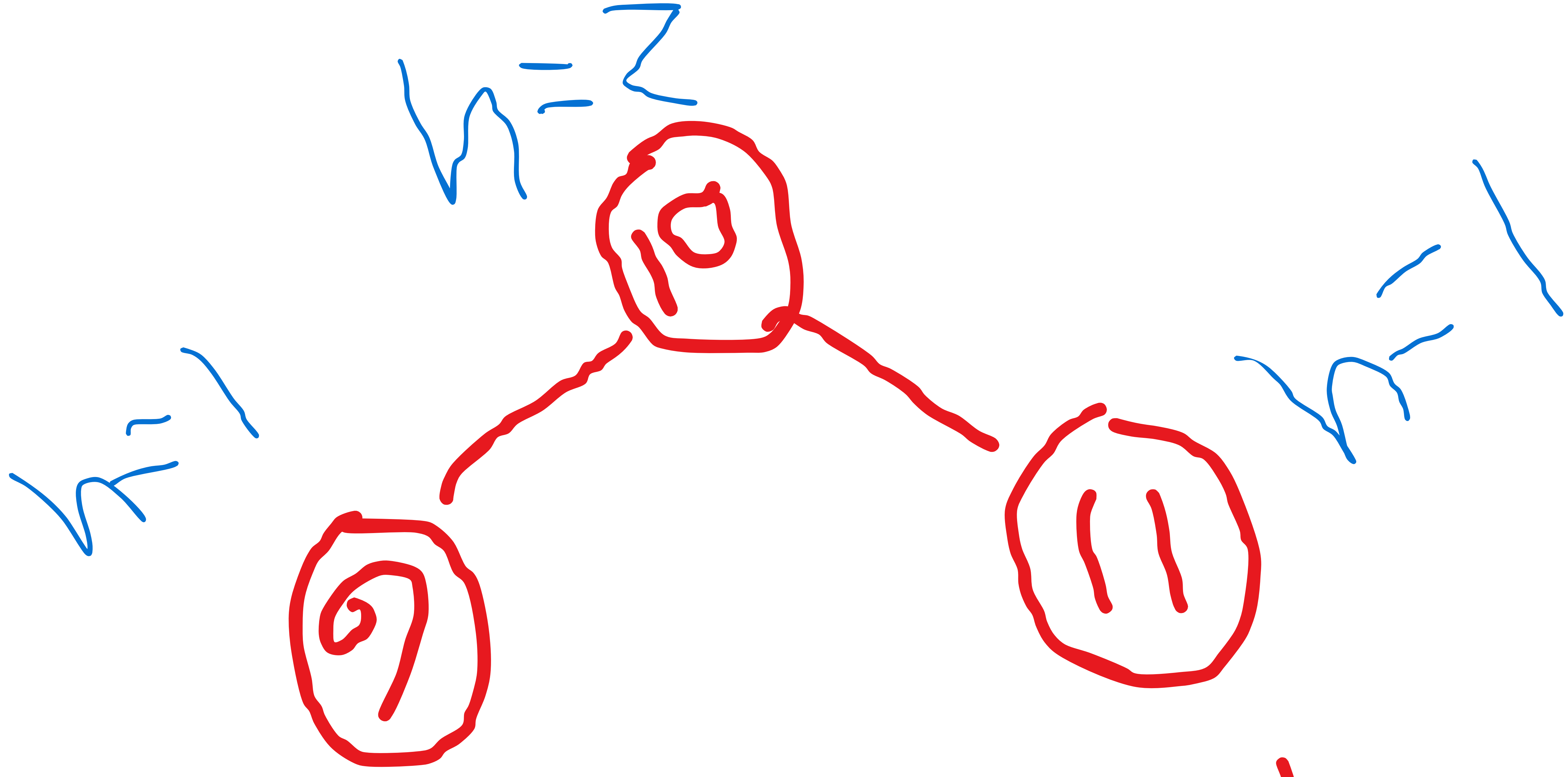
AVL Tree

Invented by the Adelson- Velky and Landis

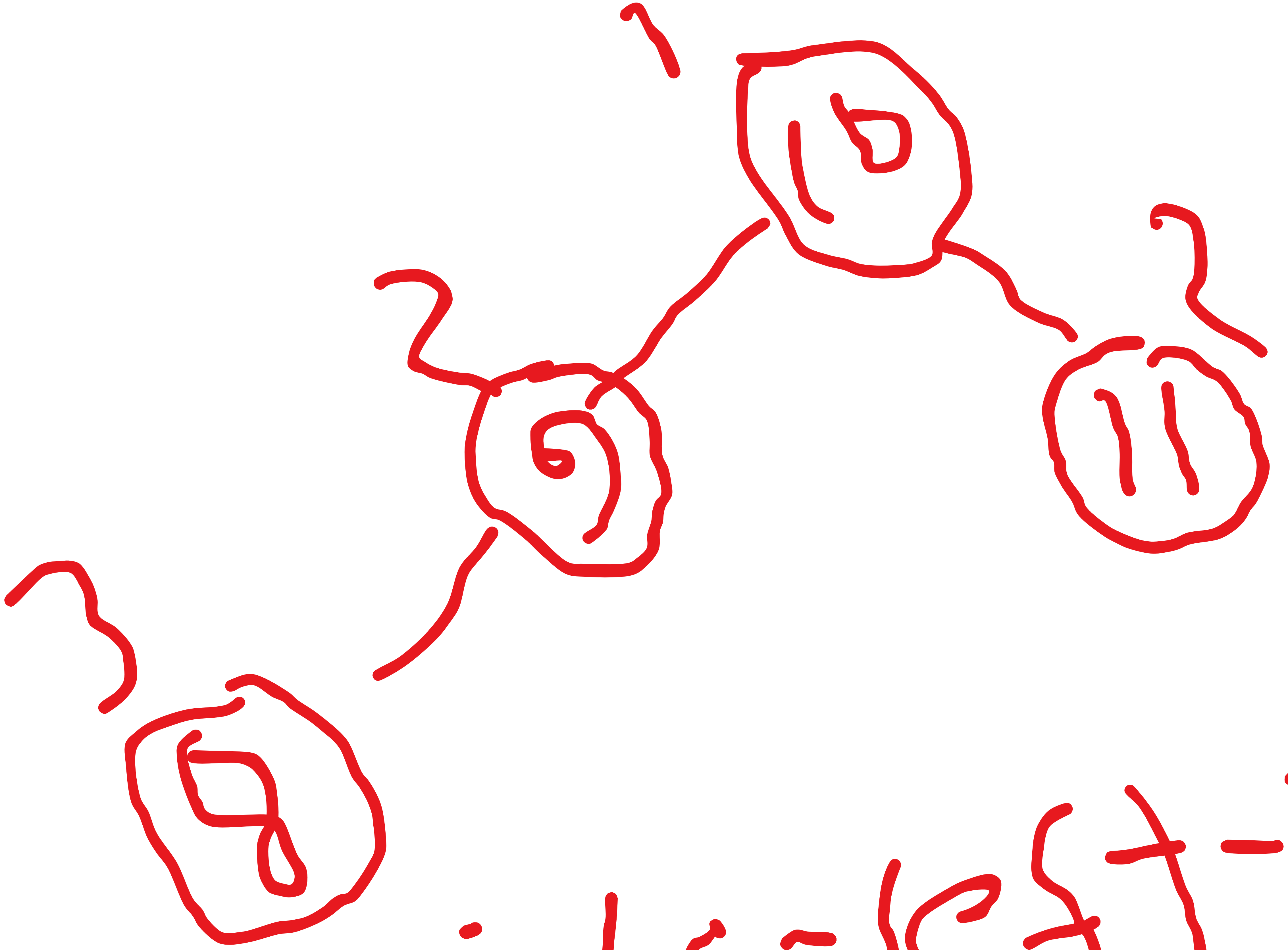
Is a self balancing binary search tree

What is self balancing binary search tree?

It maintains a property where the heights of the two child subtrees of any node differ by at most one, thus ensuring that the tree remains balanced.

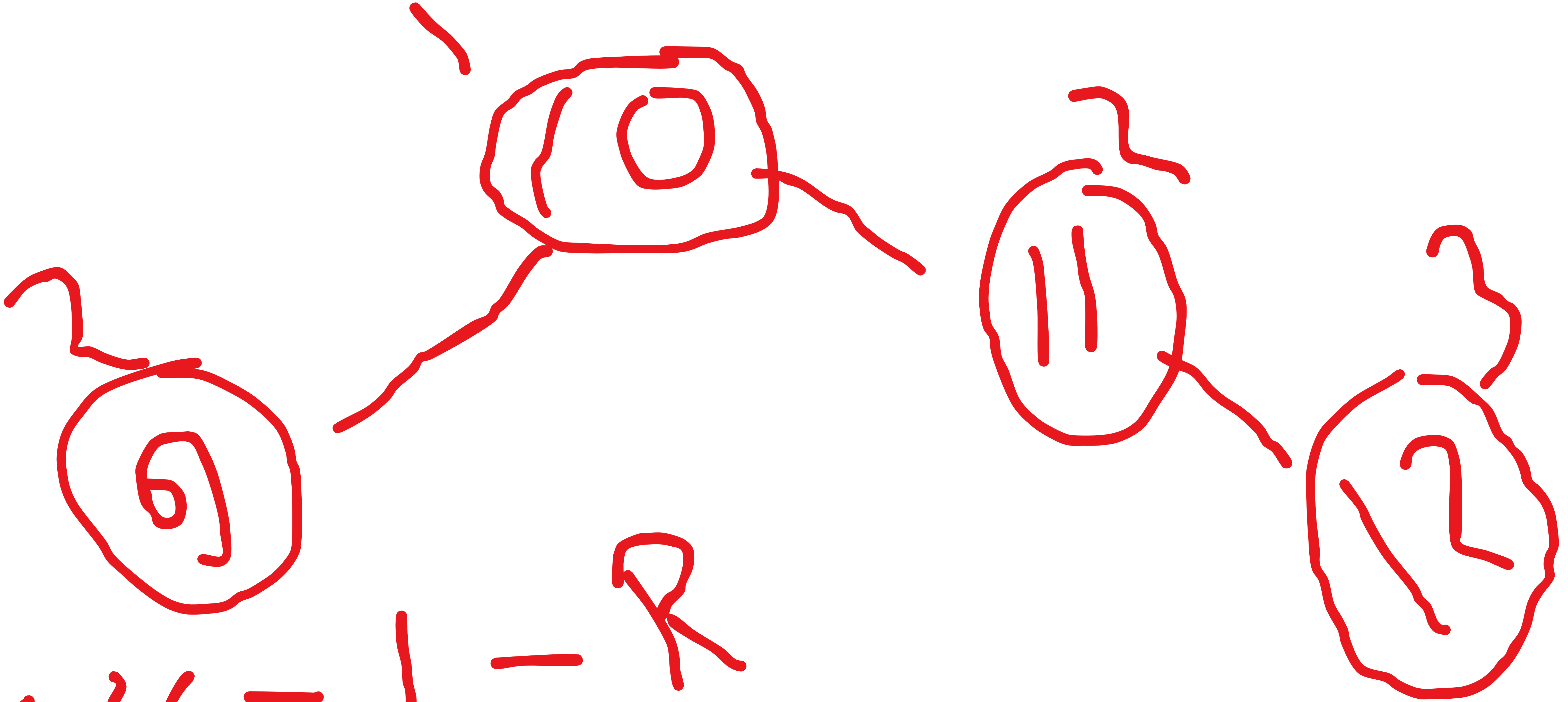


$$\text{diff} = \text{left} - \text{right}$$
$$= 1 - 1 = 0$$



diff = left - right

= 2 - 1 = 1



$$diff = L - R$$
$$= 1 - 2 = -1$$

So, the difference between the left side height and right side height or the balance of the tree will be -1 or 0 or 1. If the tree will be balancee based on height.

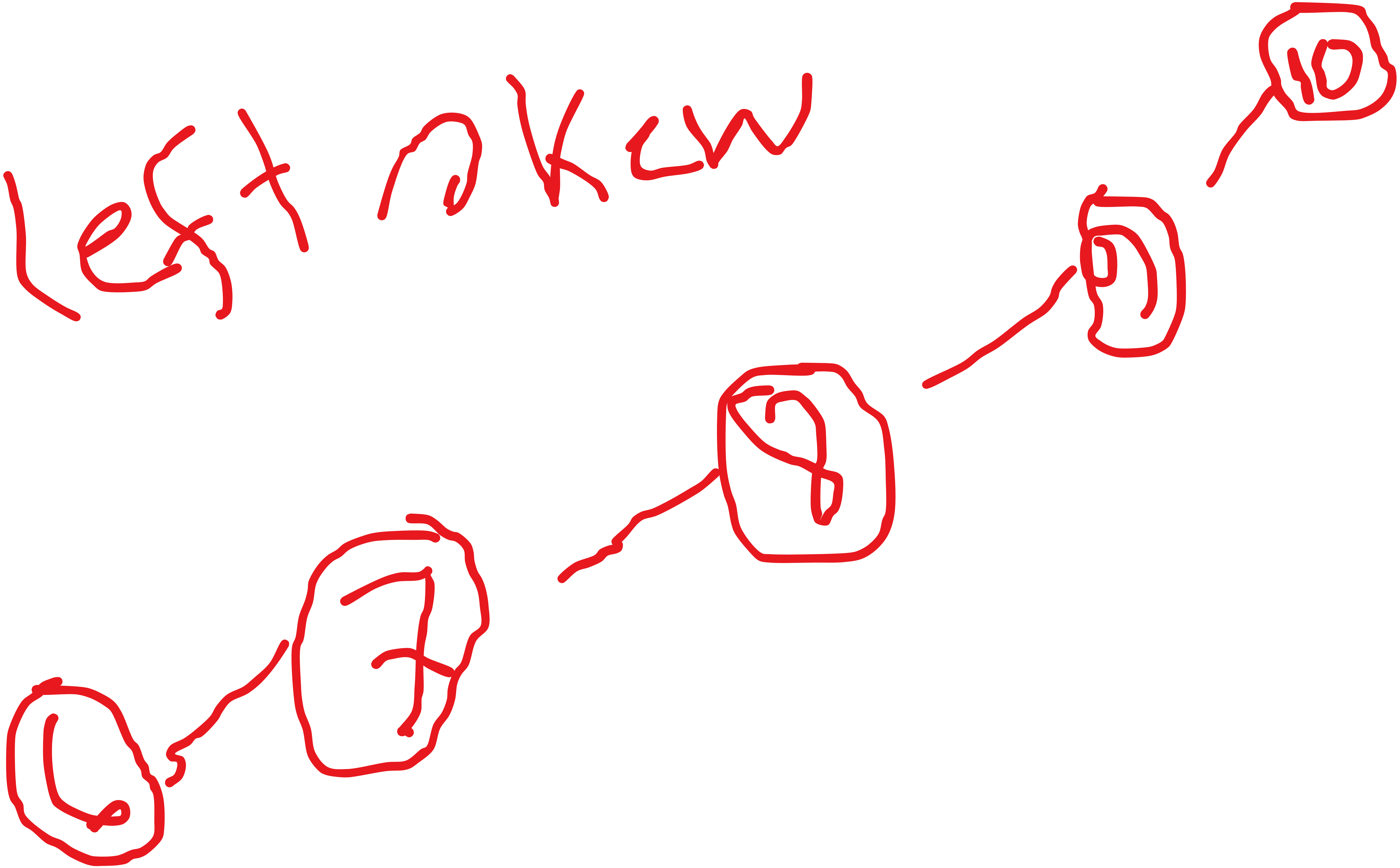
Why we use the AVL tree?

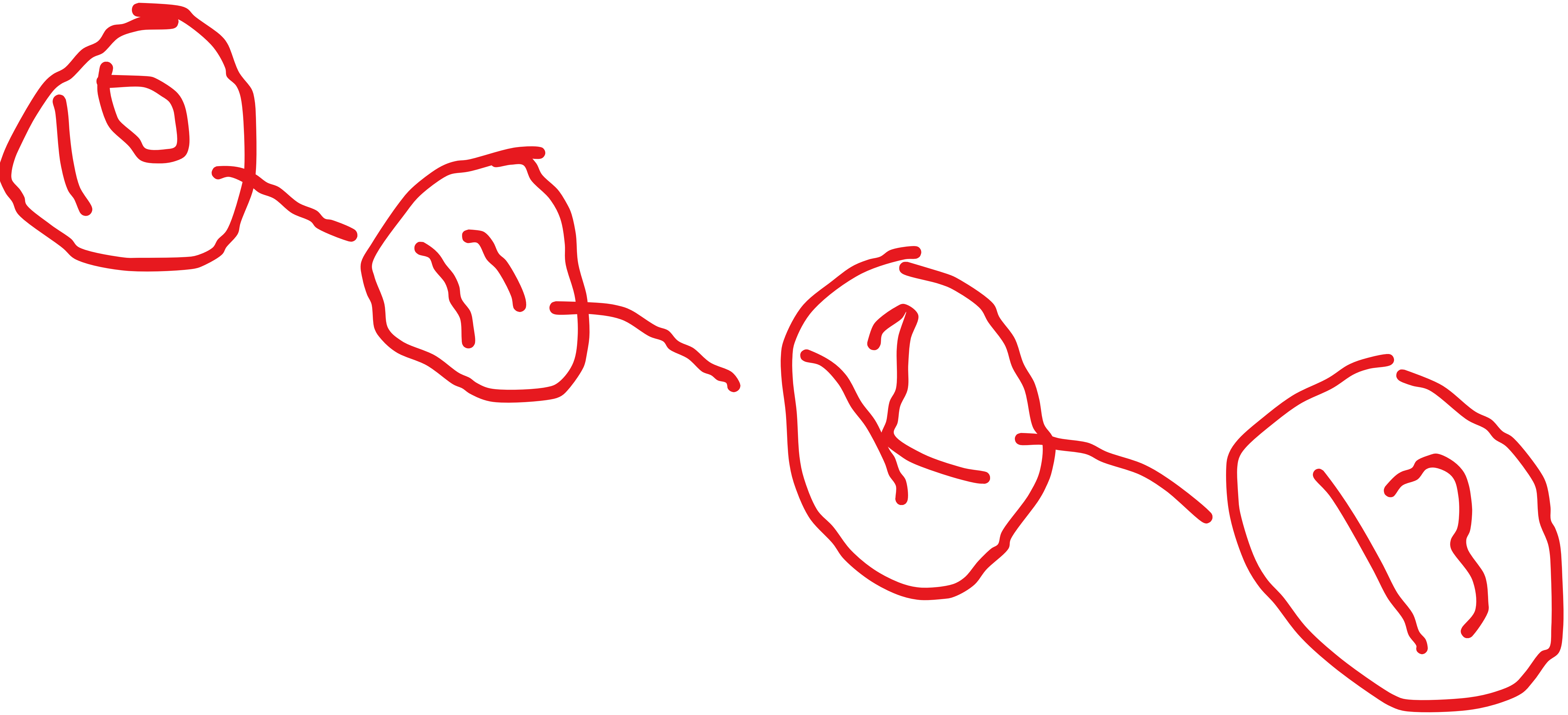
The main advantage of AVL trees is that they provide guaranteed logarithmic time complexity for basic operations like insertion, deletion, and search. This makes them efficient for applications where the data set is frequently modified or accessed.

This is widely used when we need faster addition, deletion or searching on some dynamic data.This will give me much better performance than binary search tree.

Compare with Binary Search Tree

left skew

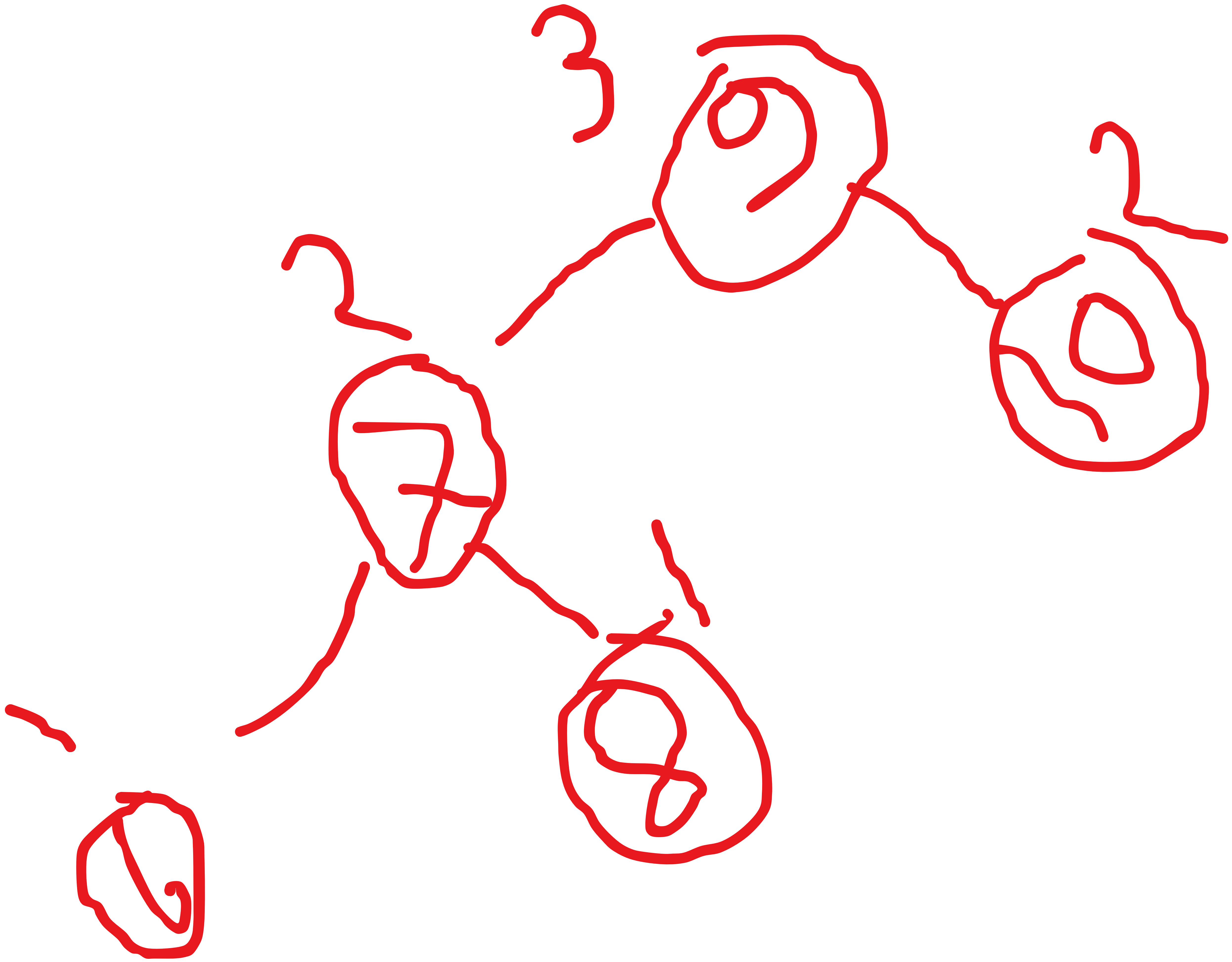




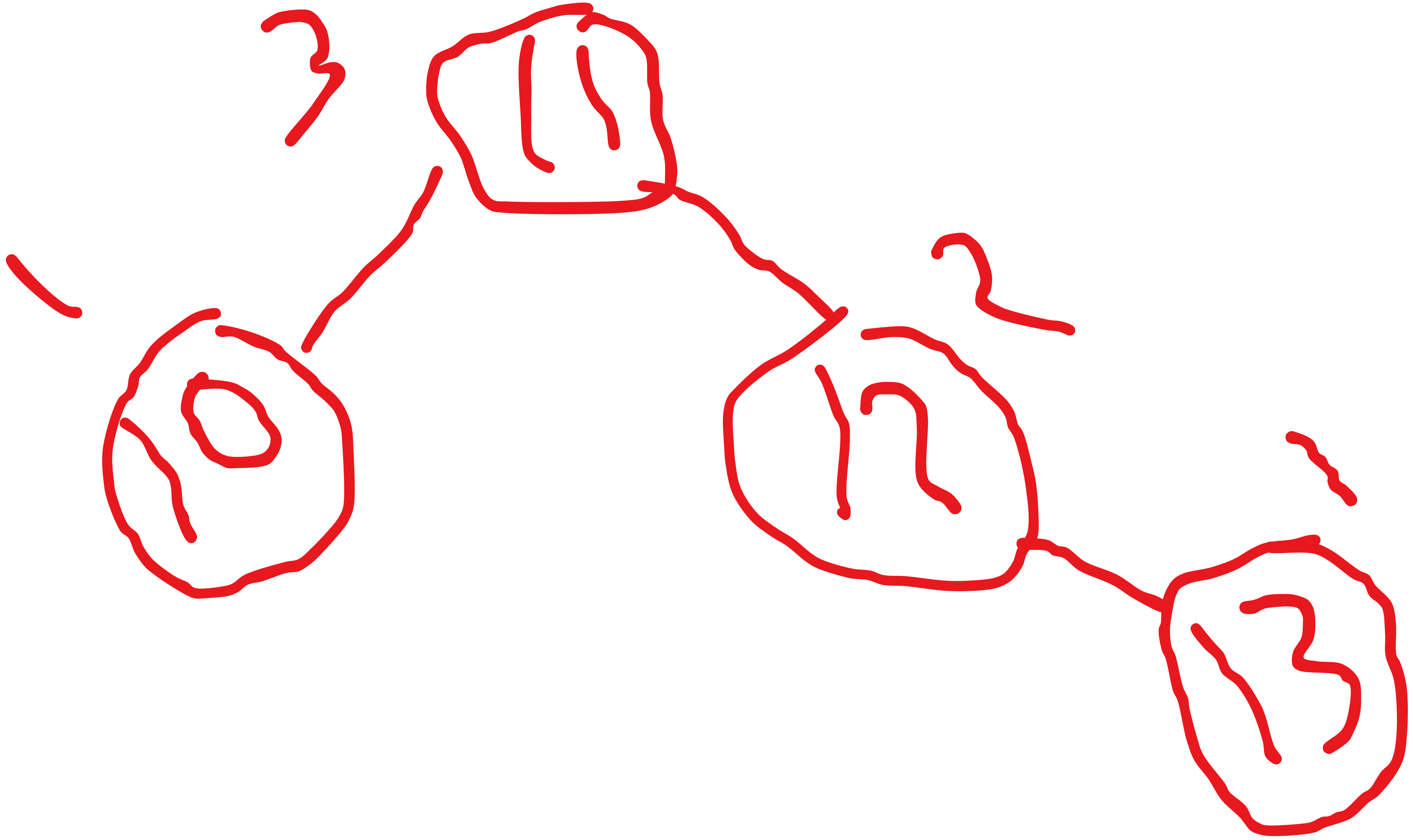
right skew

This both sitution BST will take O(n) complexity to search or make any addition or deletion operation.

But AVL tree will ensure O(log(n)) complexity via this sitution.



3
2
7
2
8


3
11
1
10
2
12
1
13

How can it balance the trees?

Via Rotations

How many types of rotation it has?

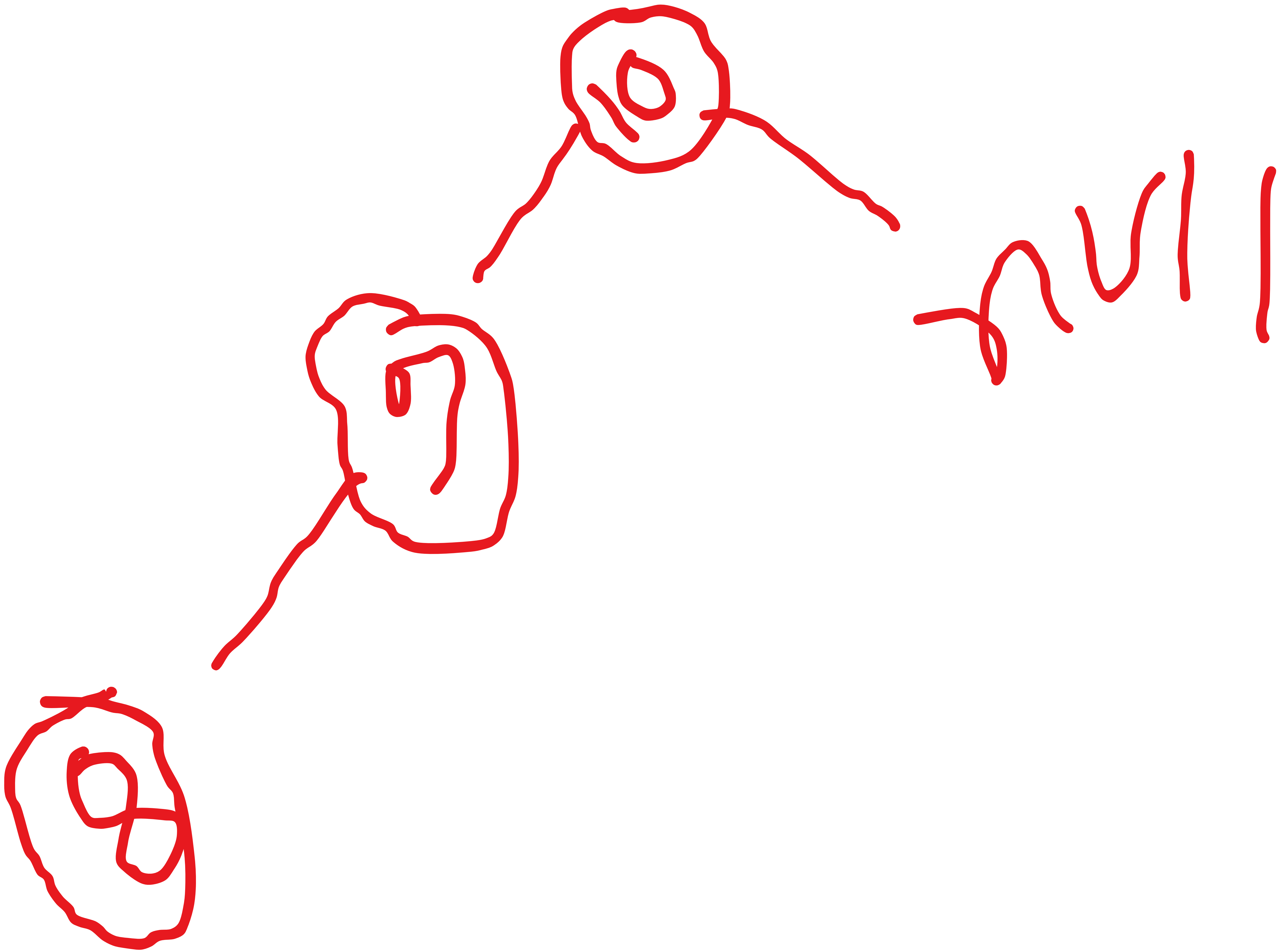
Basically two-way

- Left Rotation
- Right Rotation

When we will use them?

It’s mainly vary on the balance of the tree.

When the balance of the tree is > 1, in that sitution the tree will look like this way.

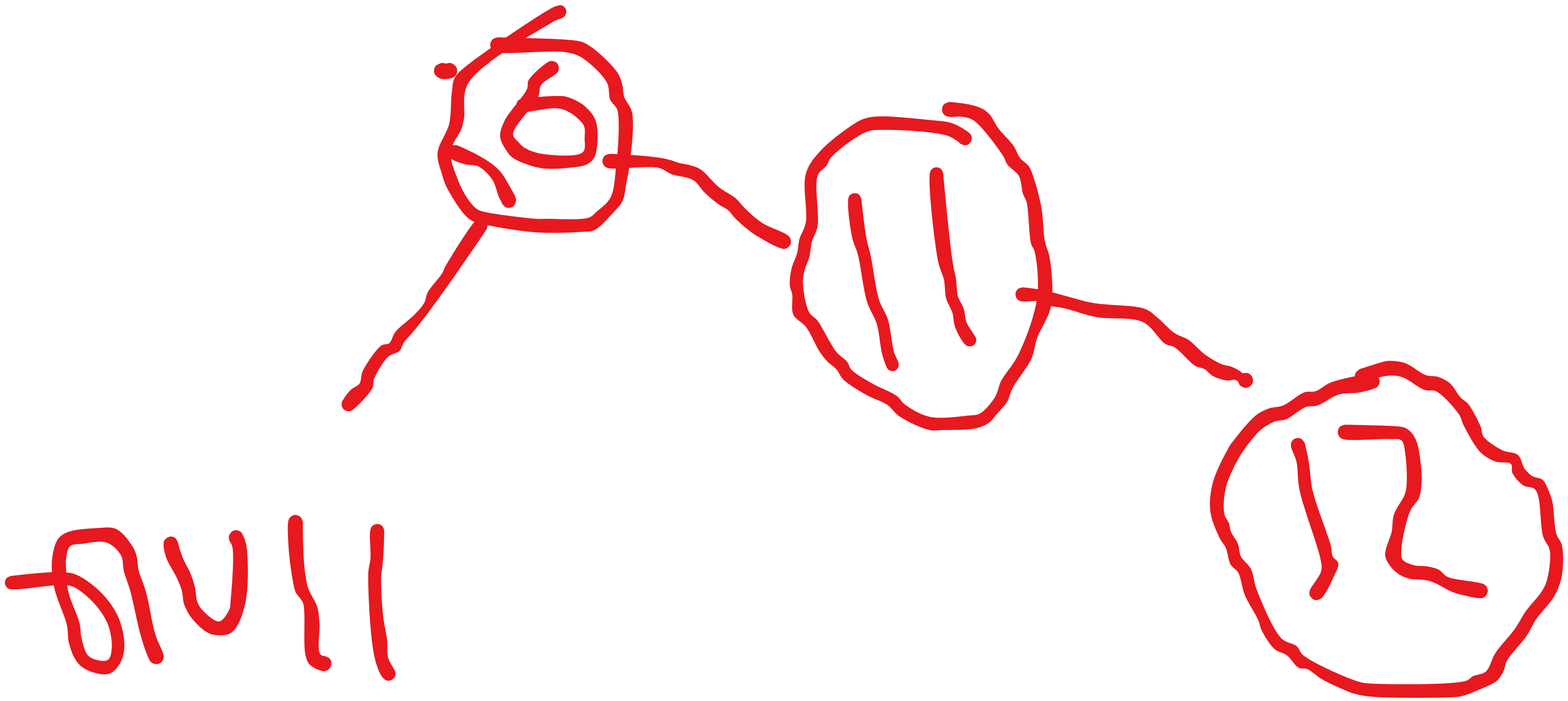


10
7
8
null

This is also called the left heavy situation

In this sitution we have to use the right rotation.

If the balance factor of the tree is ≤ -2, then the sitution will look like this way.

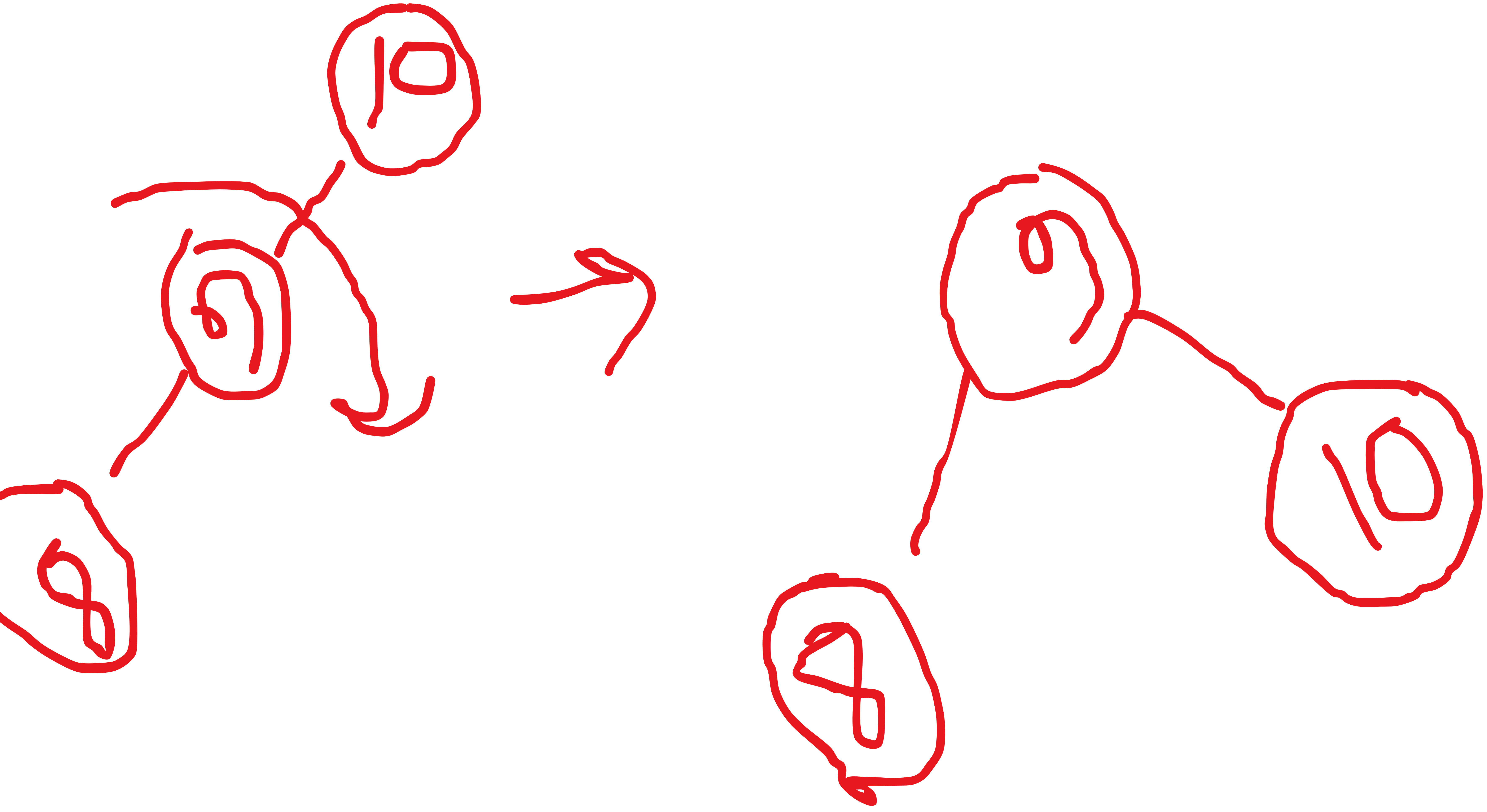


6
11
12
null

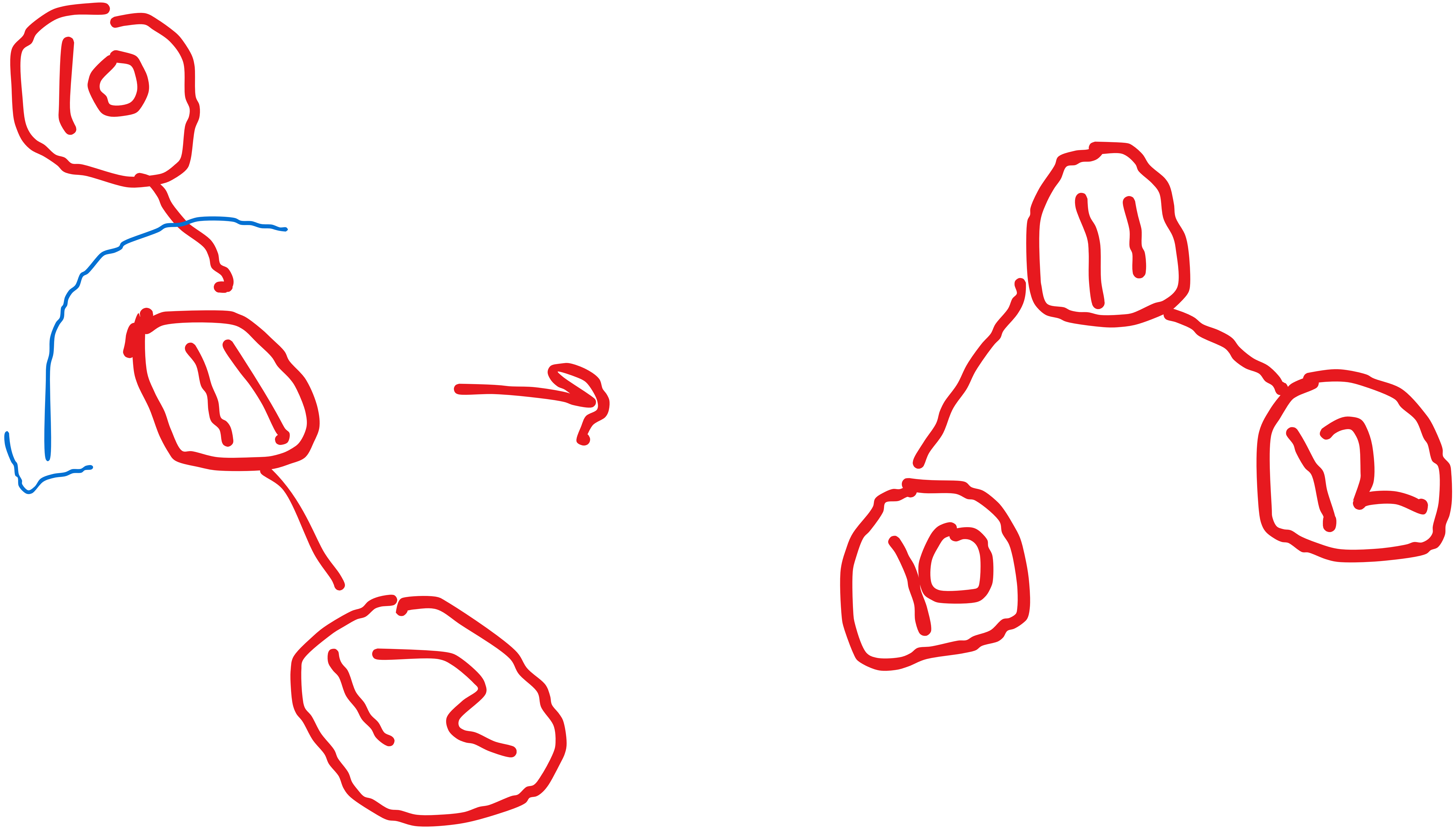
This is called the right heavy sitution

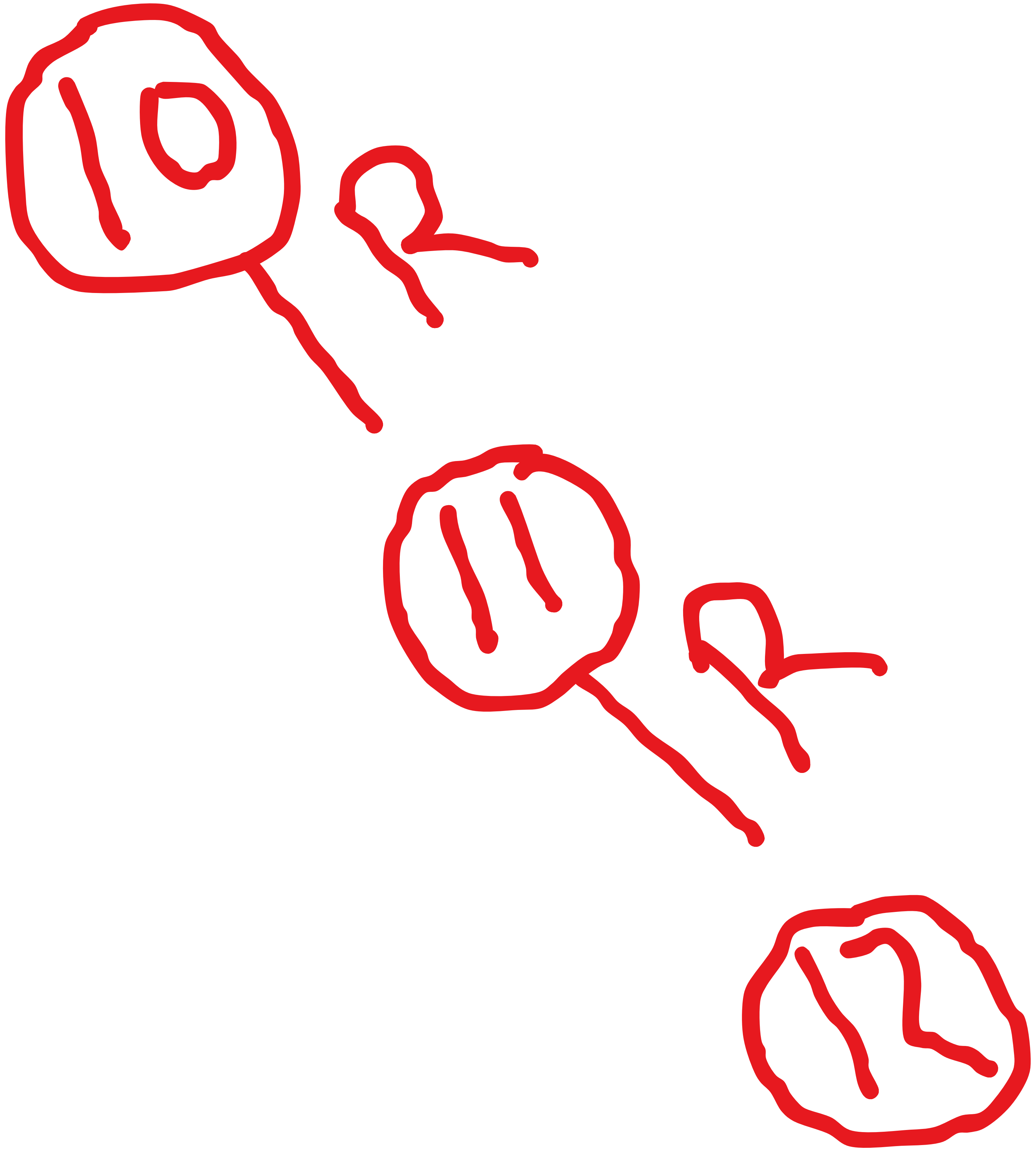
In this sitution we need to perform the left rotation.

After performing the right rotation the tree will look like this.

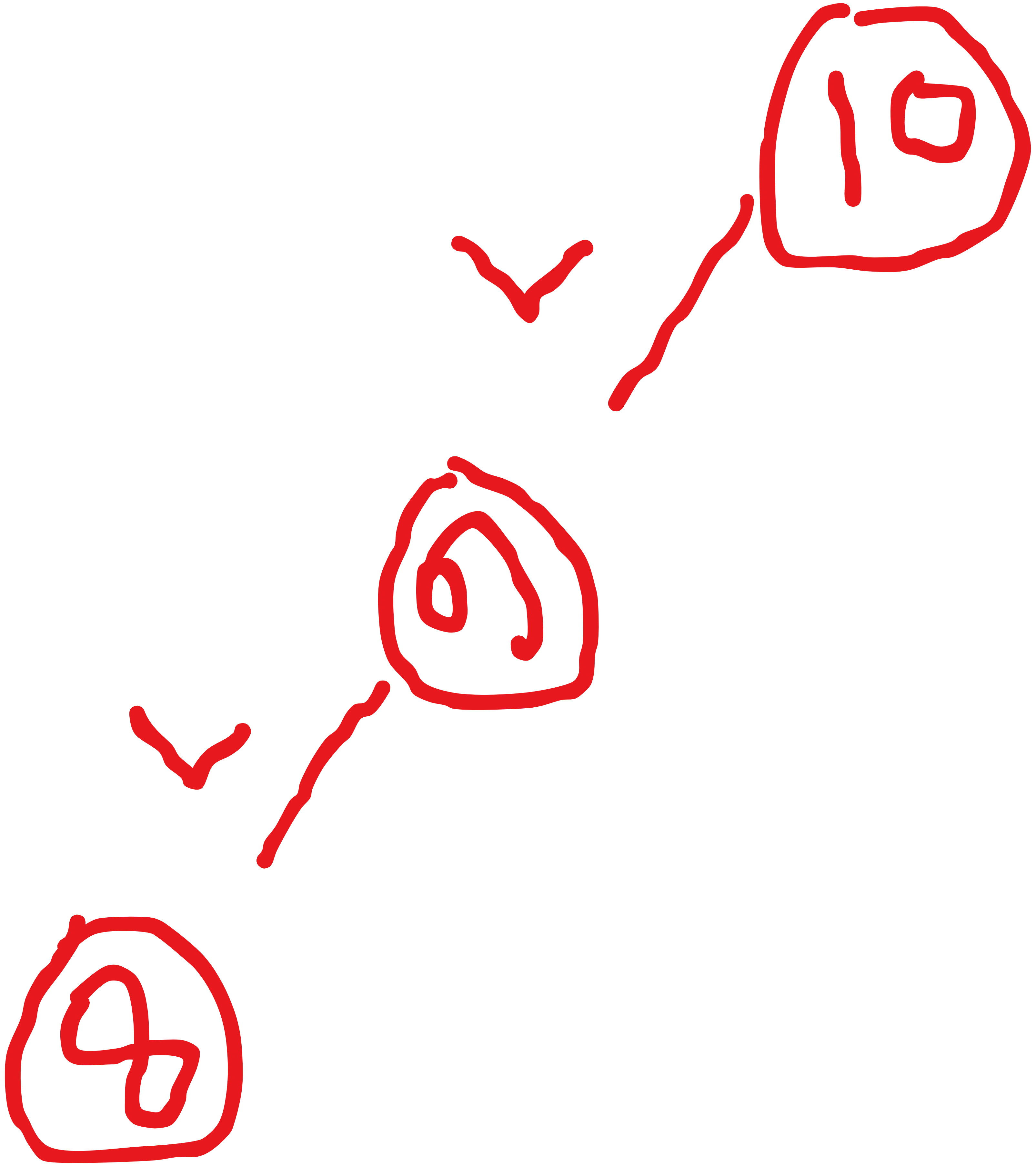

10
9
8
9
8
10

After performing the left rotation the tree will look like this


10
11
12
11
10
12


10
R
11
R
12

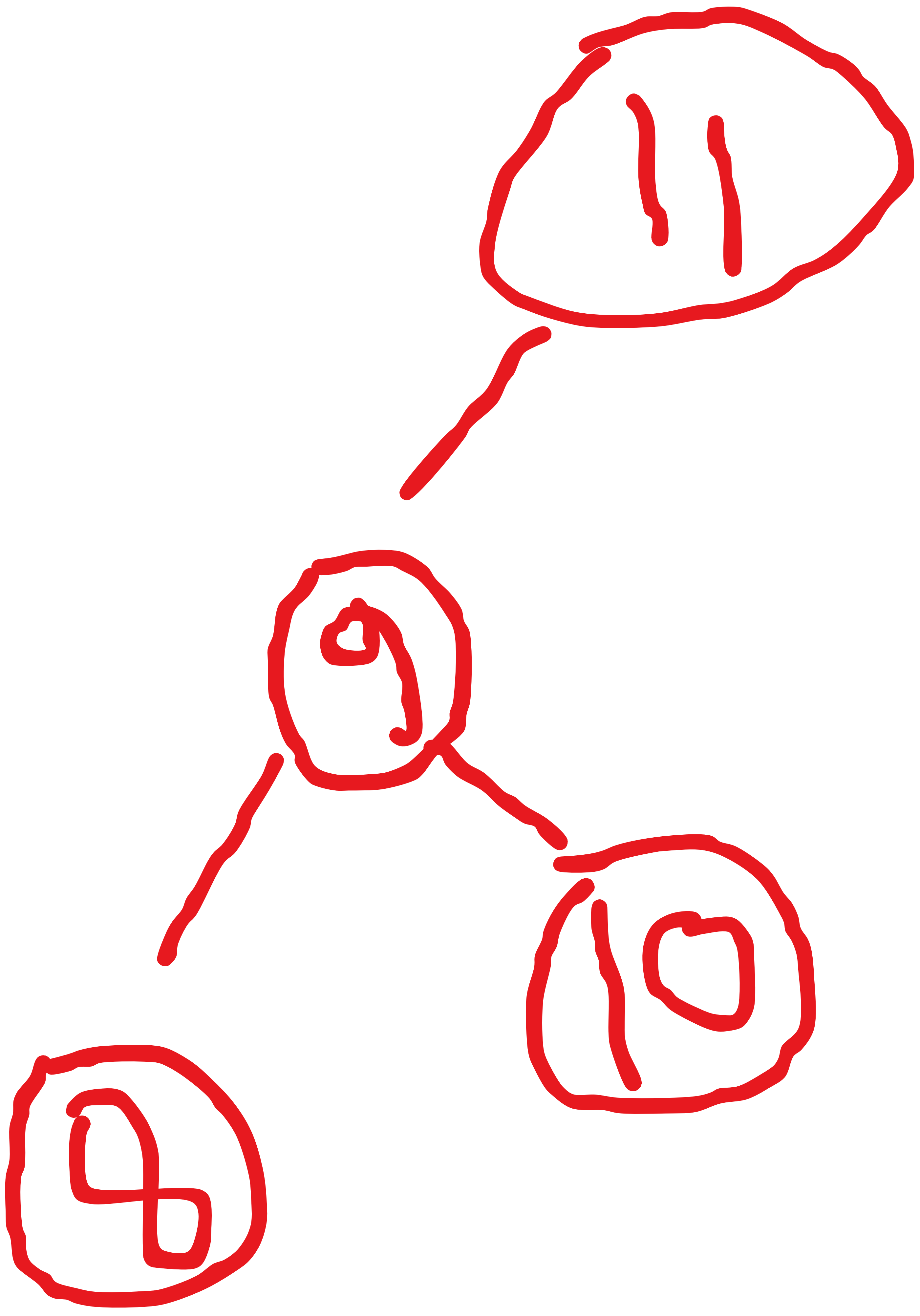
This is called the Right-Right sitution

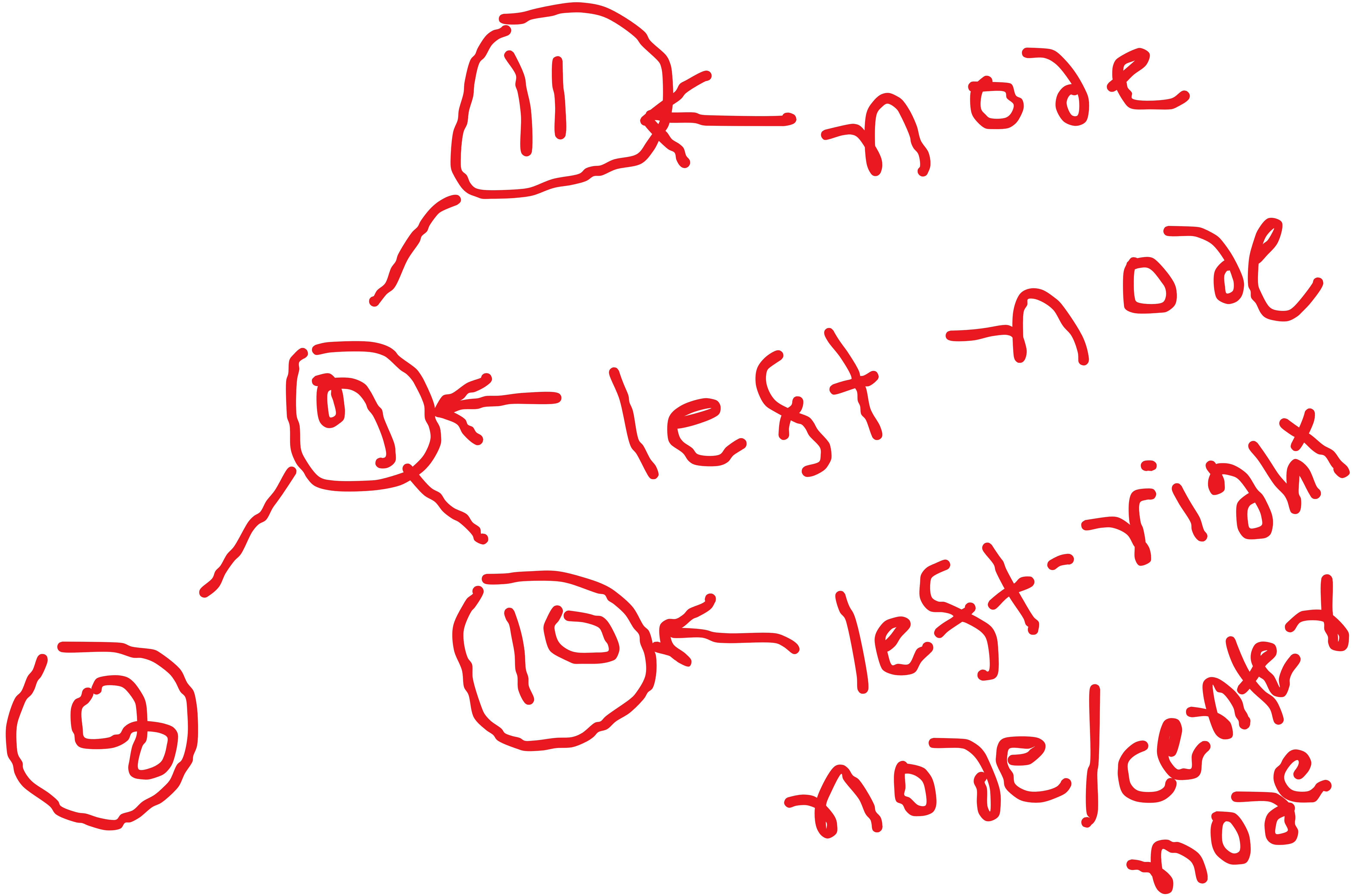

10

This is called the left - left sitution

- **we mainly can perform the right rotation in this L-L sitution**
- **We mainly can perform the left rotaton in this R-R situation.**

So, how we can perform the Right rotation?


11
9
8
10


11
node
9
left node
10
left-right node/center node

Set this node as the right node of the left node of this main node.

11

node

9

left node

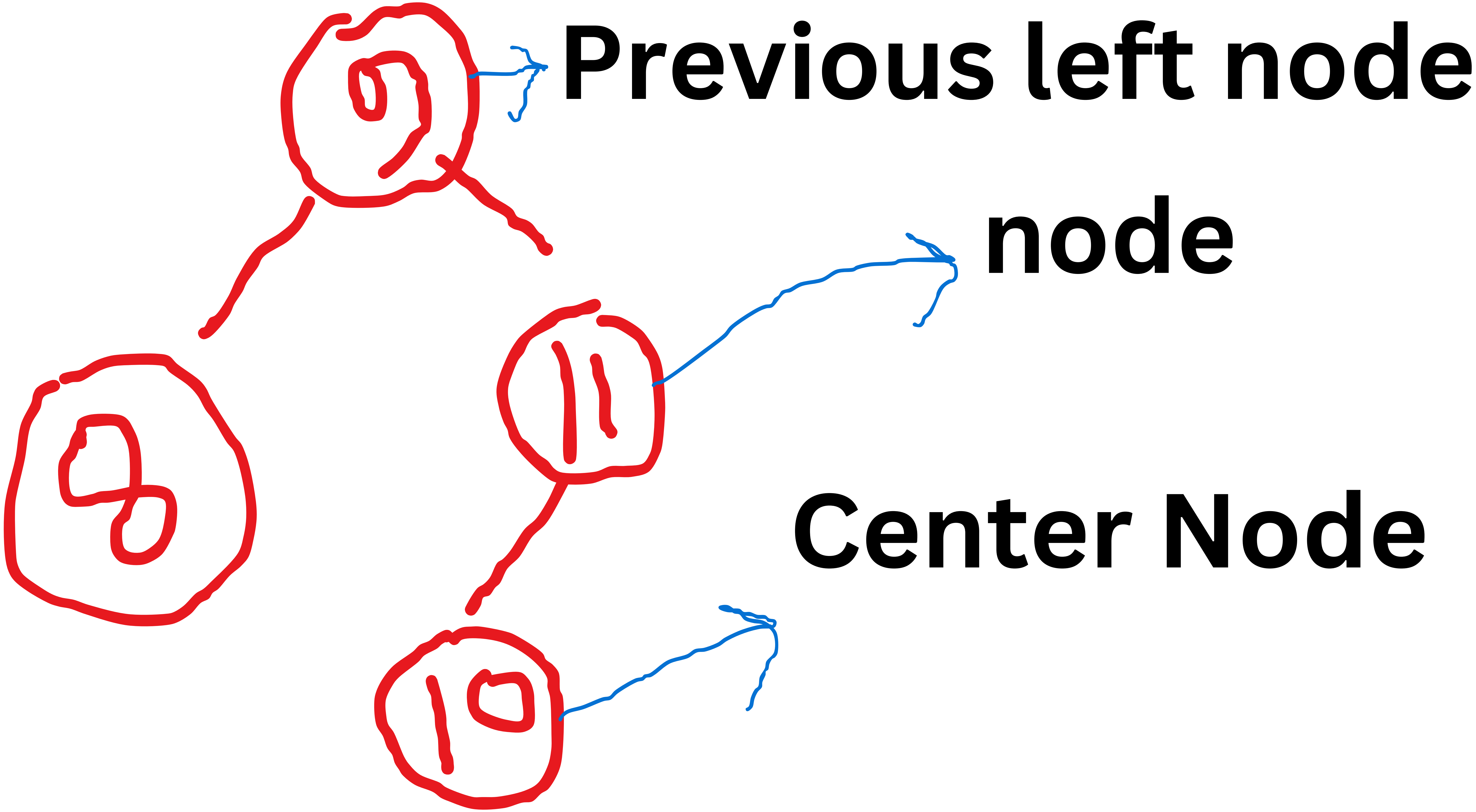
10

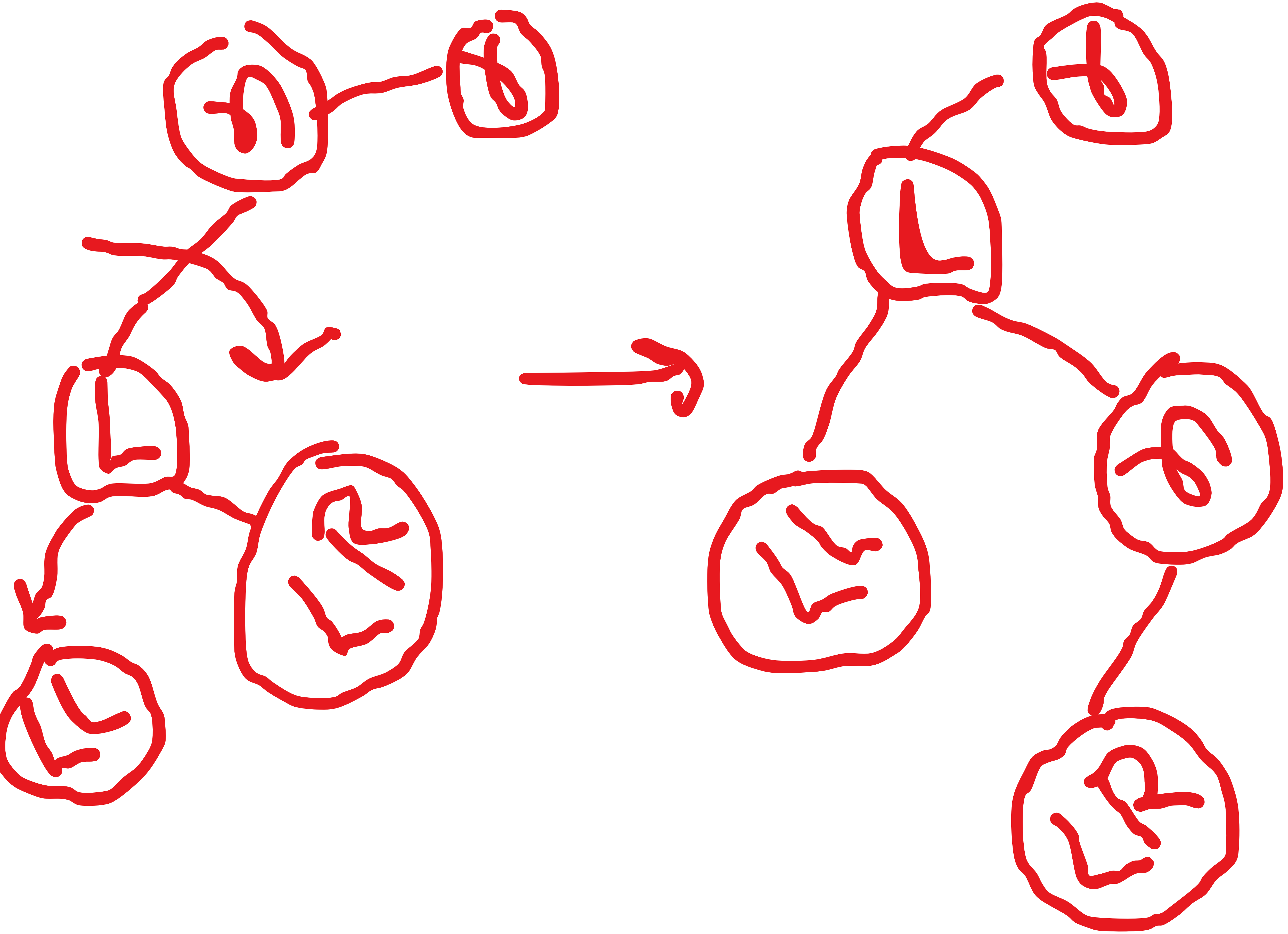
left-right node

8

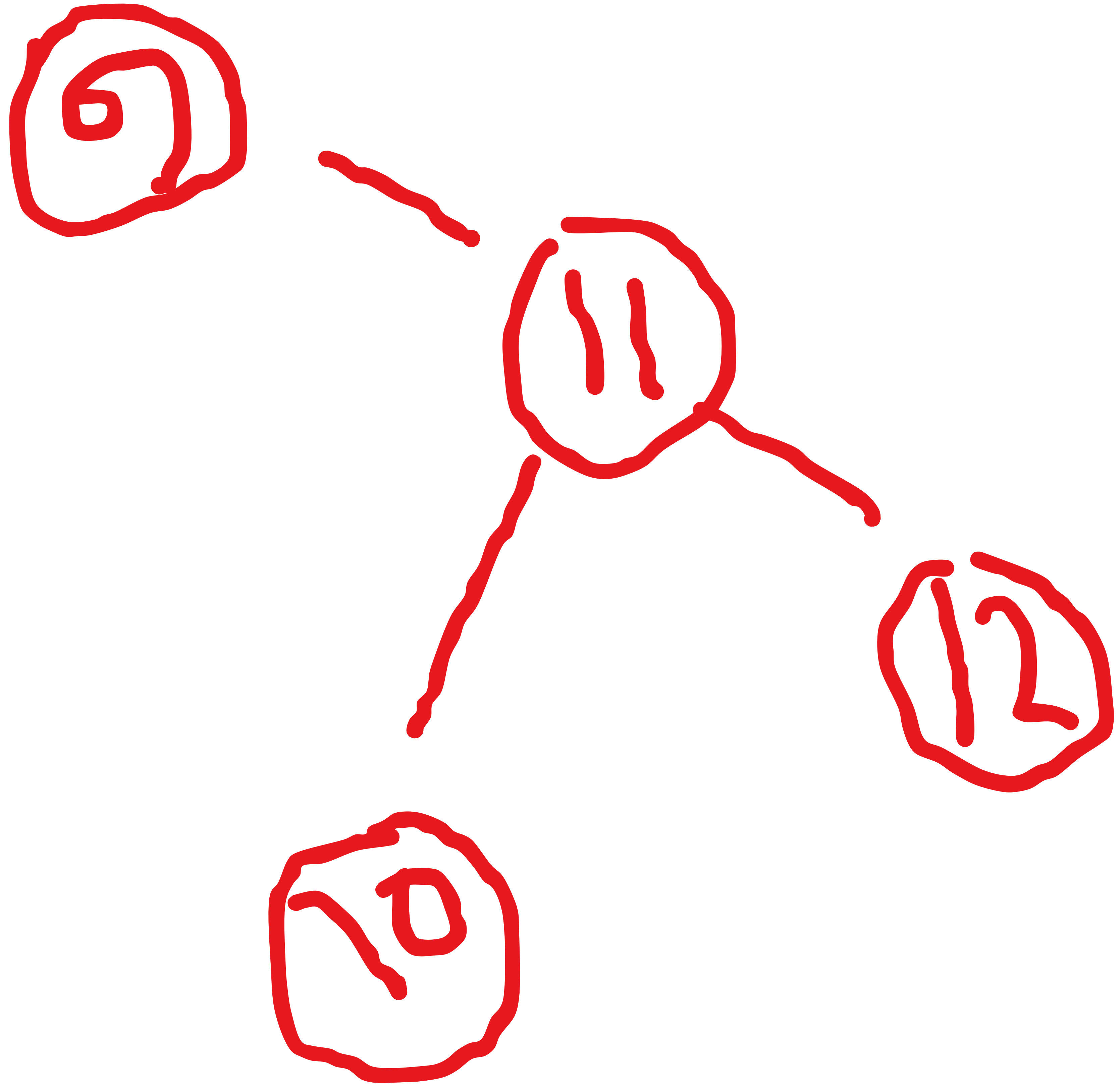
Set this left right node

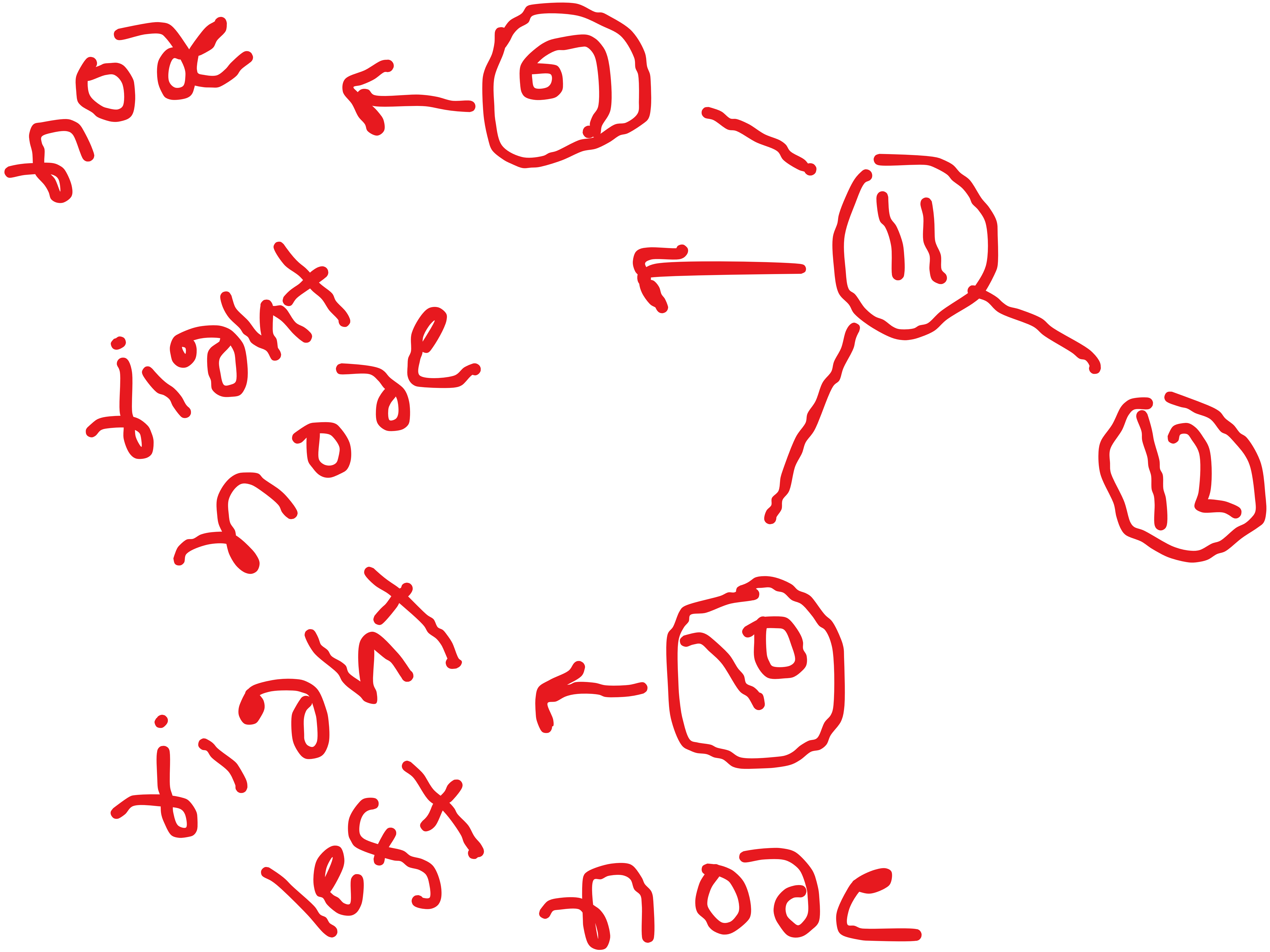
node as the left of the main node.


9
8
11
10
Previous left node
node
Center Node

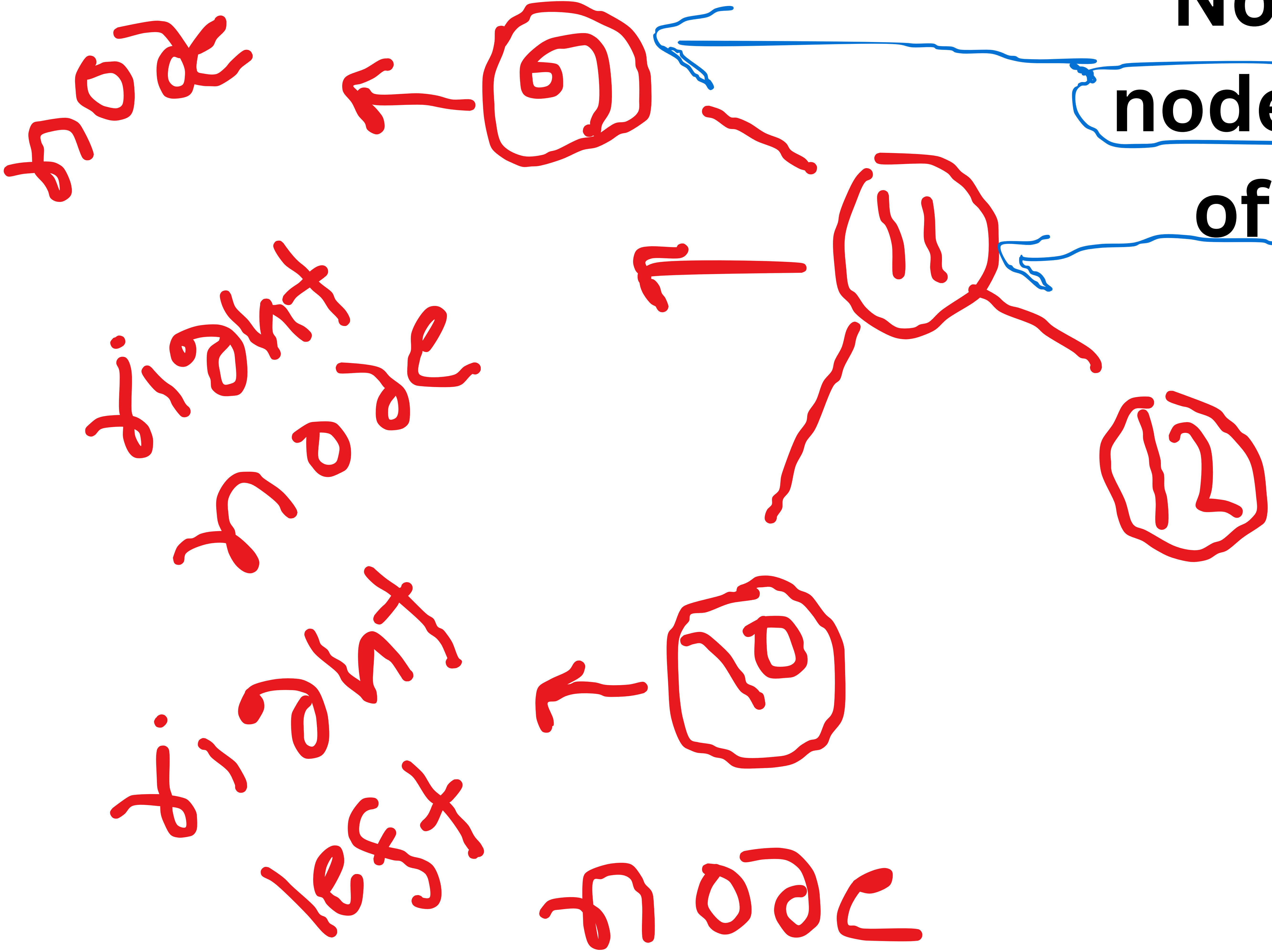

L
LL
LR
L
LL
LR

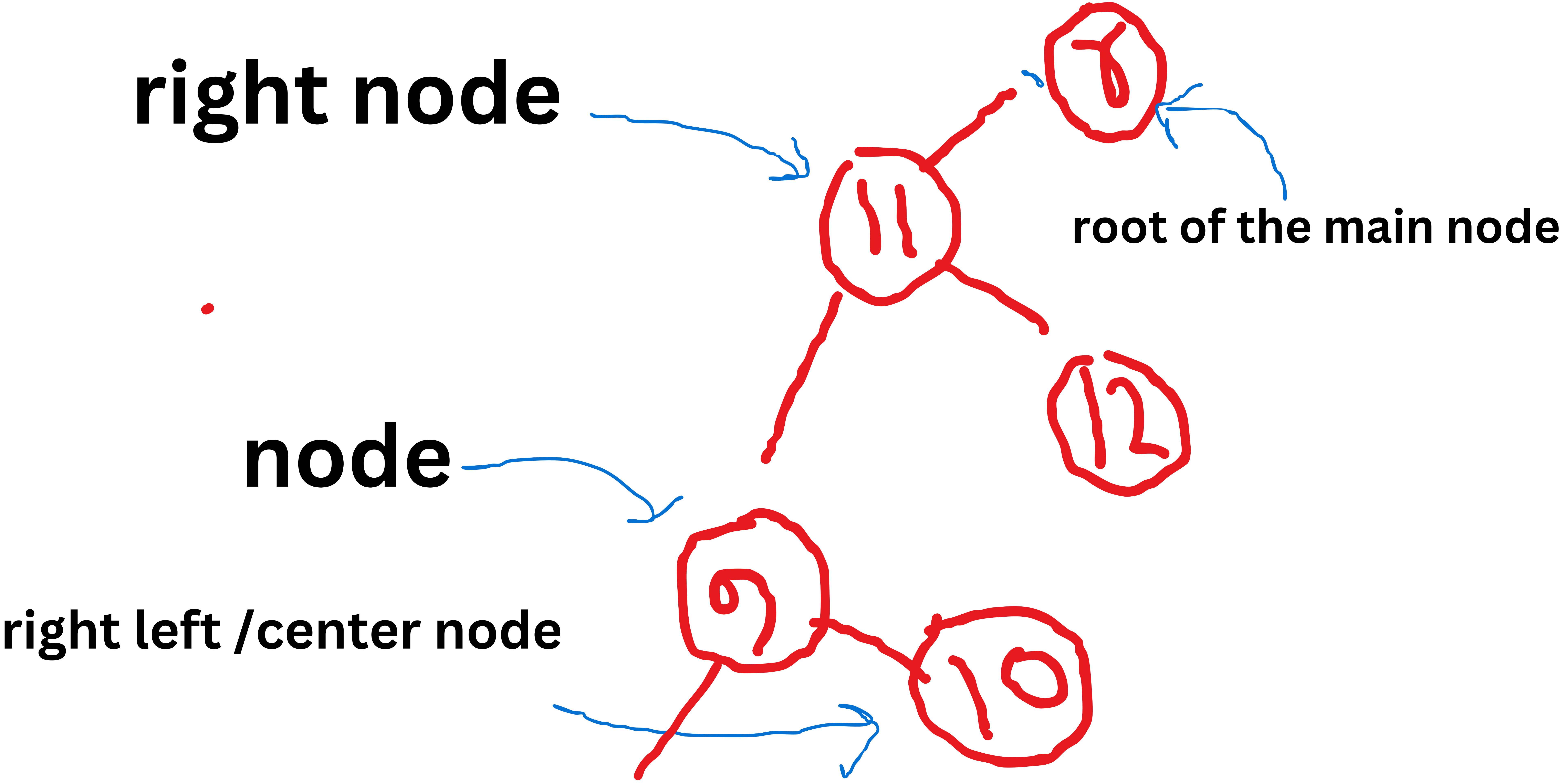
Now how can we perform the left rotation?


9
11
10
12

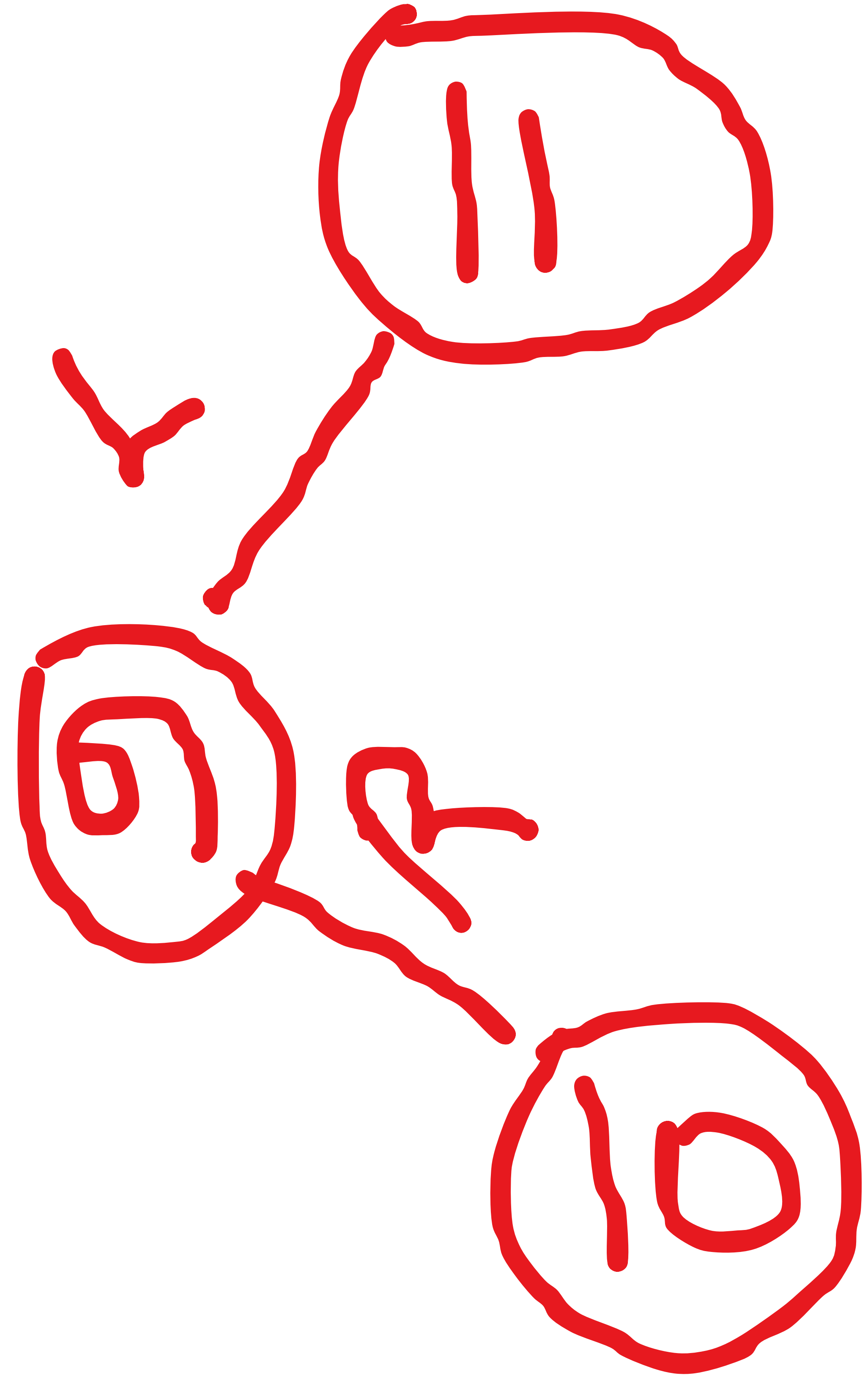

node
9
11
right node
12
10
right left
node

Now set this node as the left of the right node.

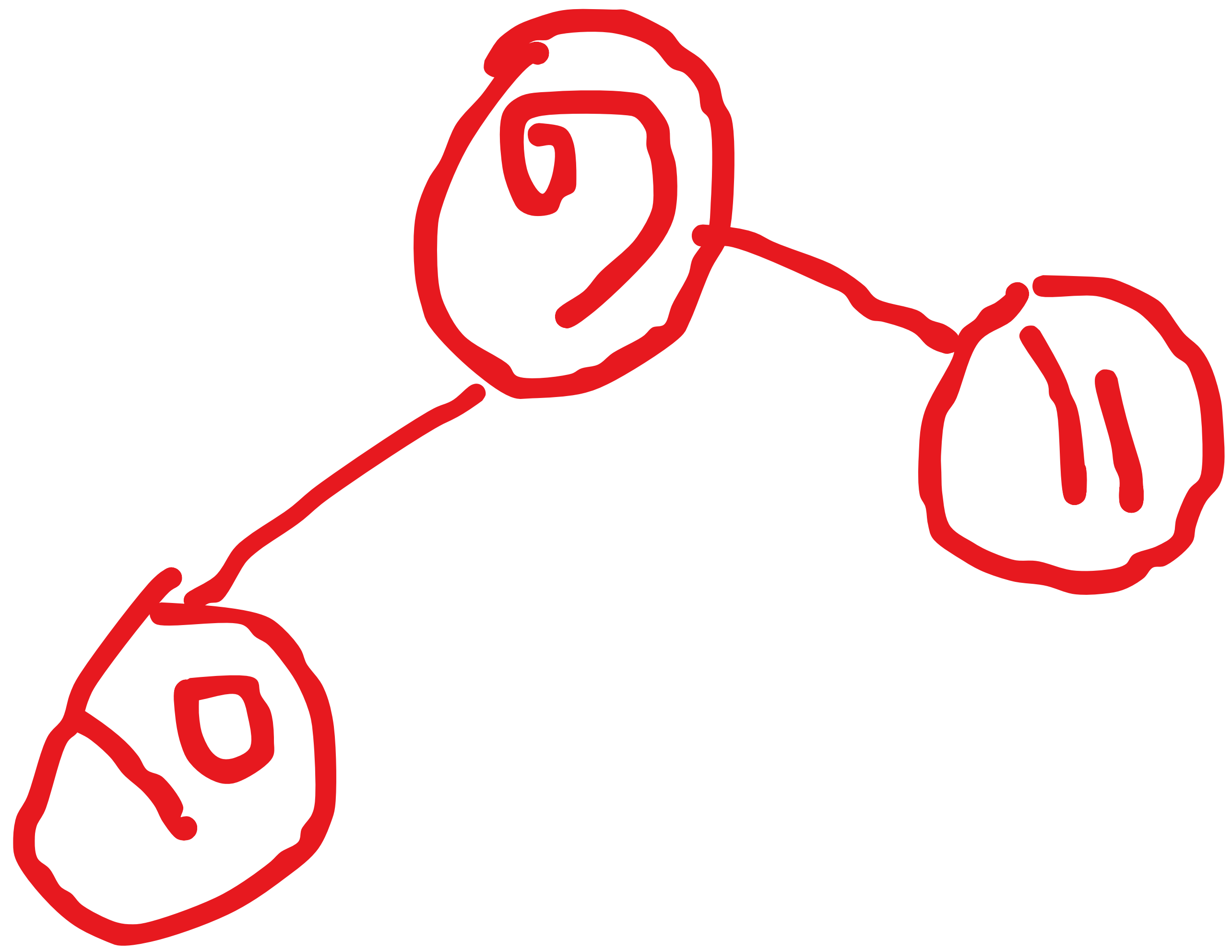



right node
root of the main node
11
12
node
9
right left /center node
10

But, there is a spetial case at here.

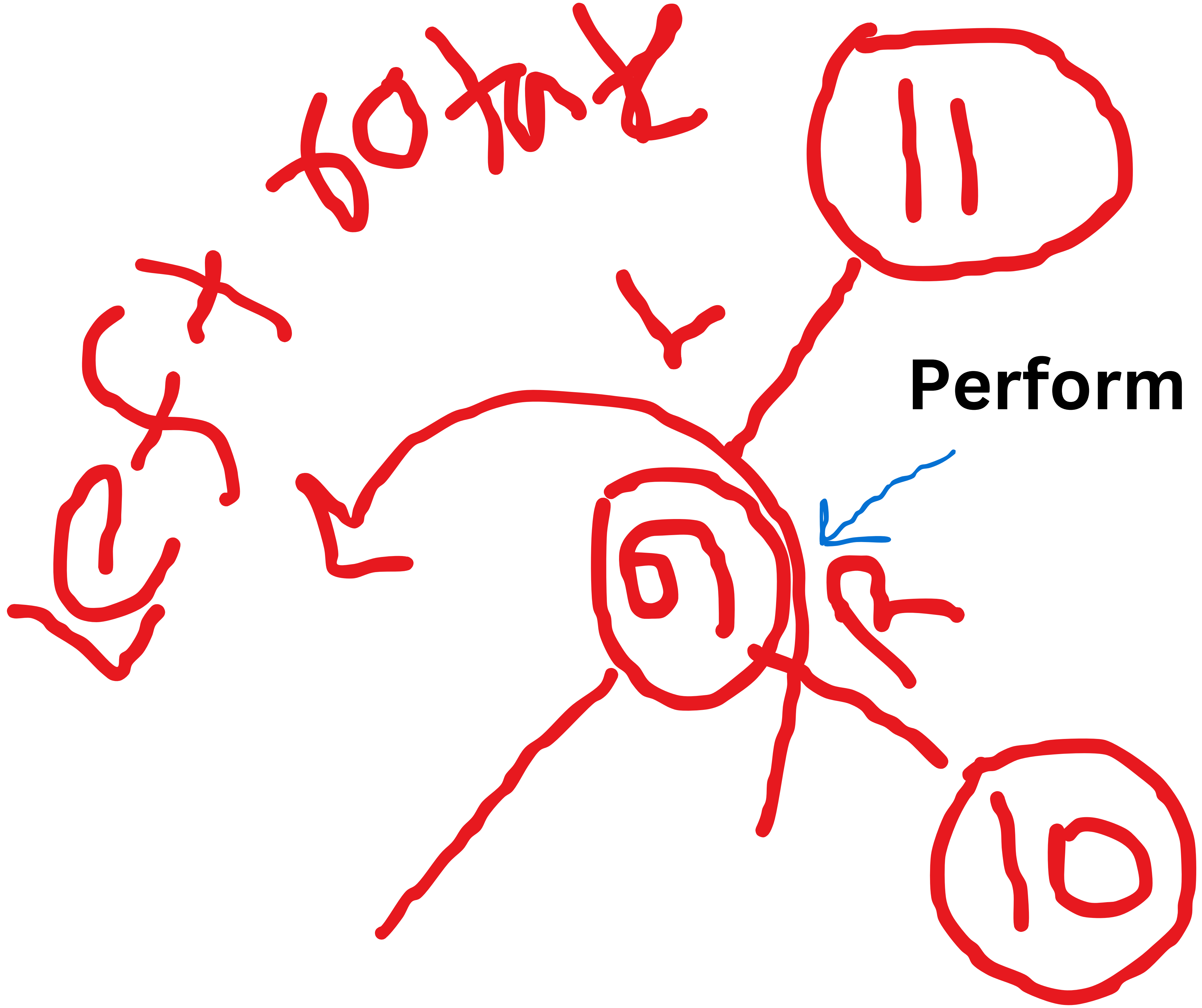

11
L
9
R
10

If, we perform the right rotation at here.

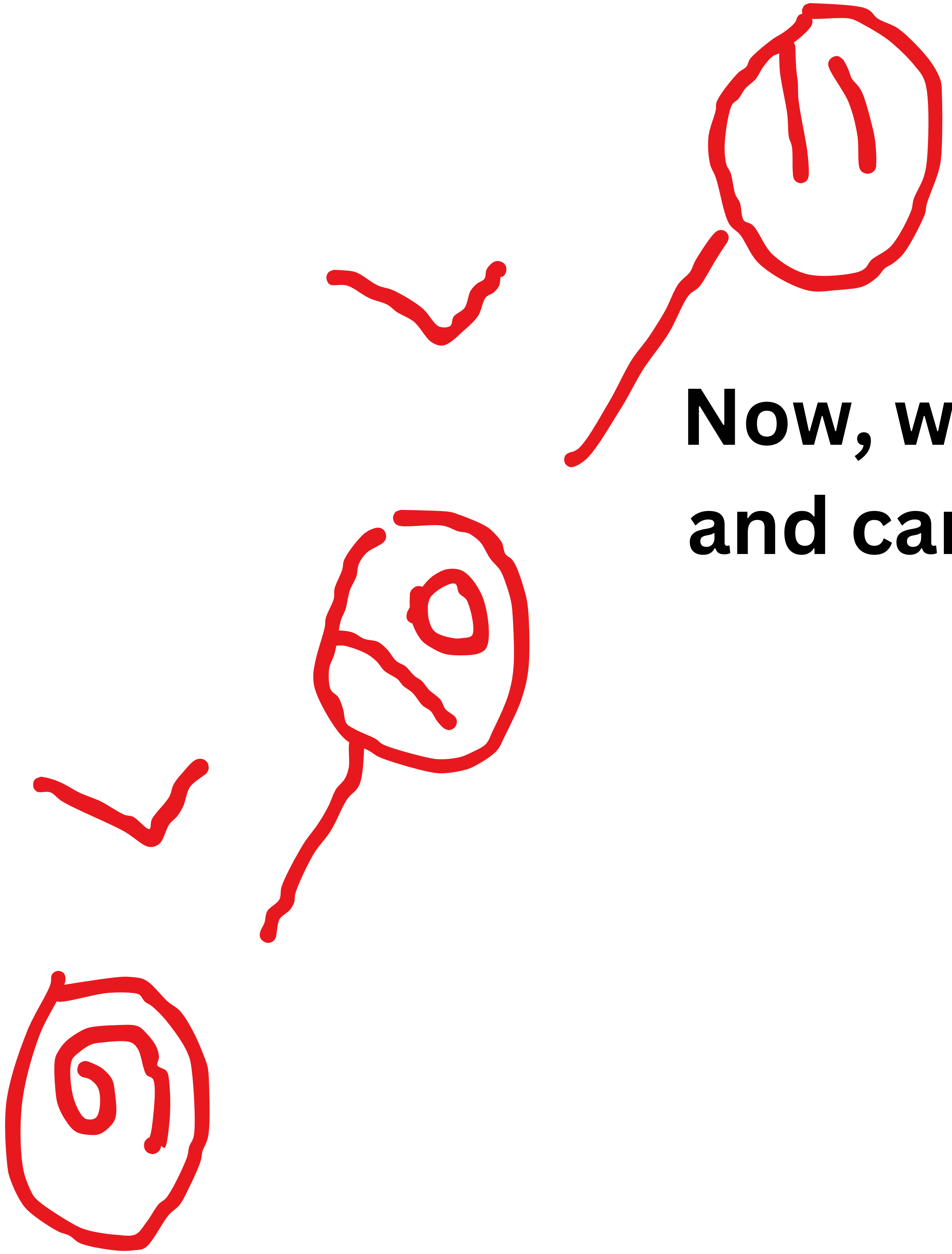


Basically this sitution violoate the rules of BST.

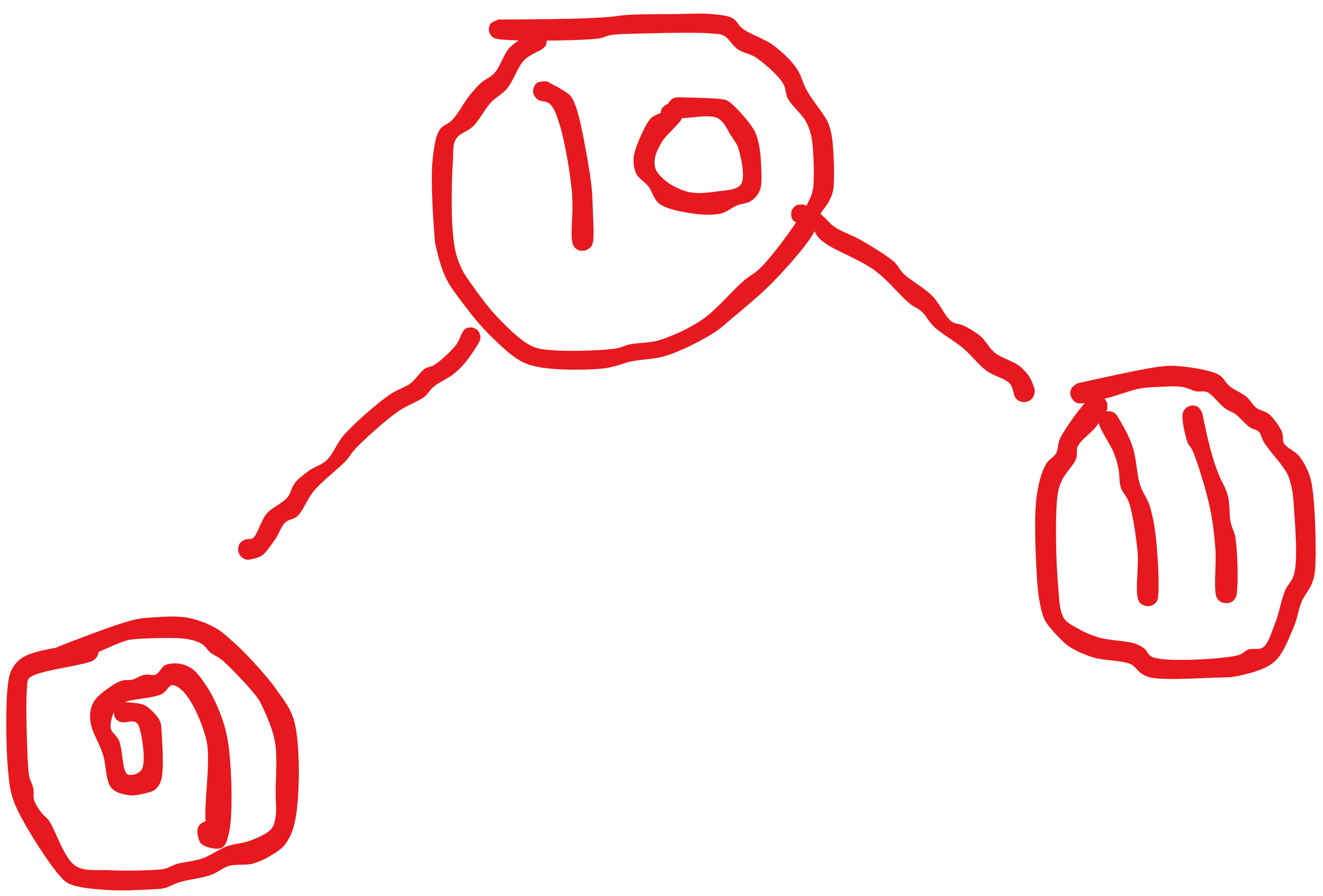
So, how can we solve this issue?



Perform the left rotation on the left node.

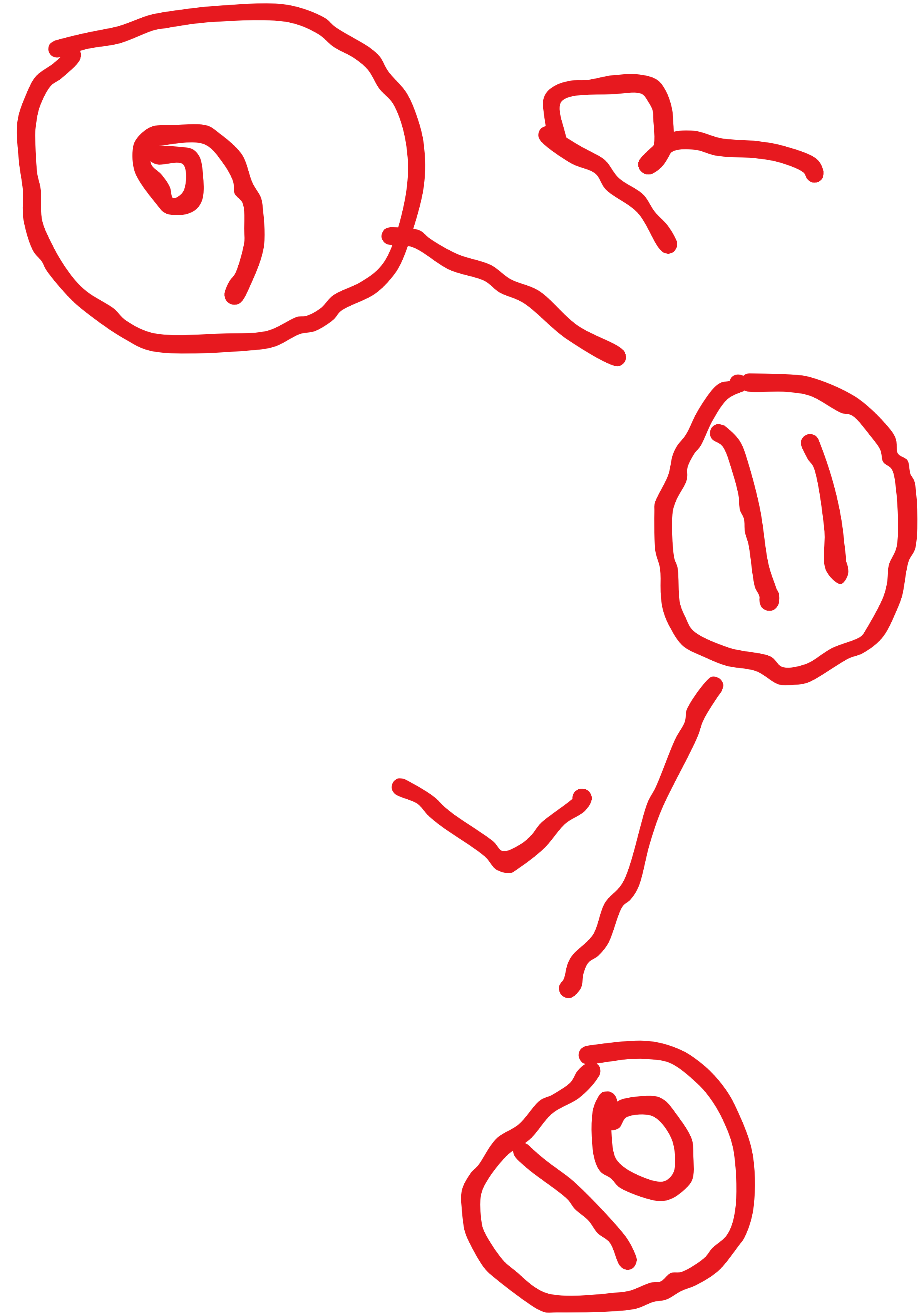


Now, we get the wanted L-L sitution and can perform the right rotate on the node



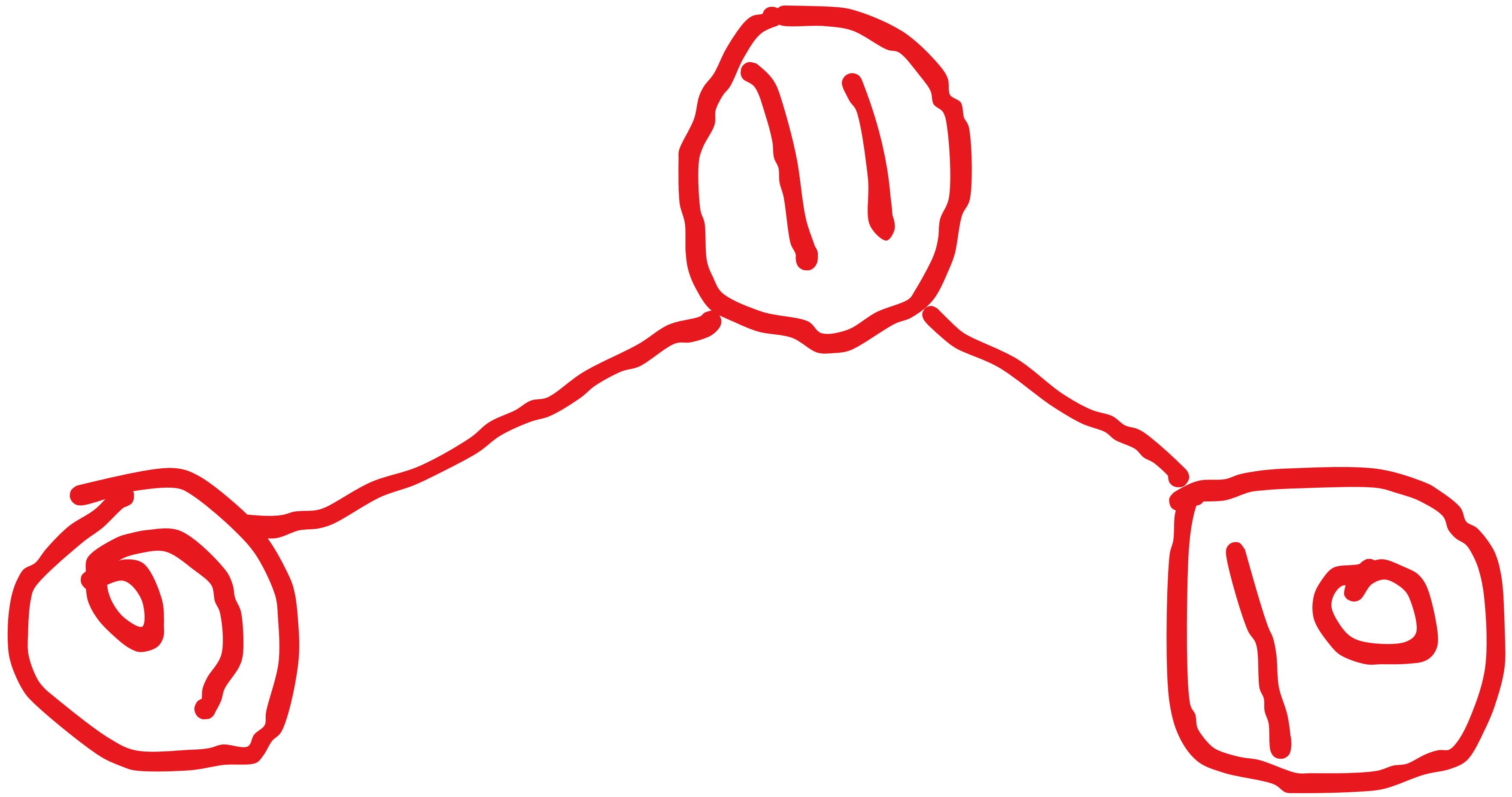
After perform the right rotation

We have an another sitution at here.


9
R
11
L
10

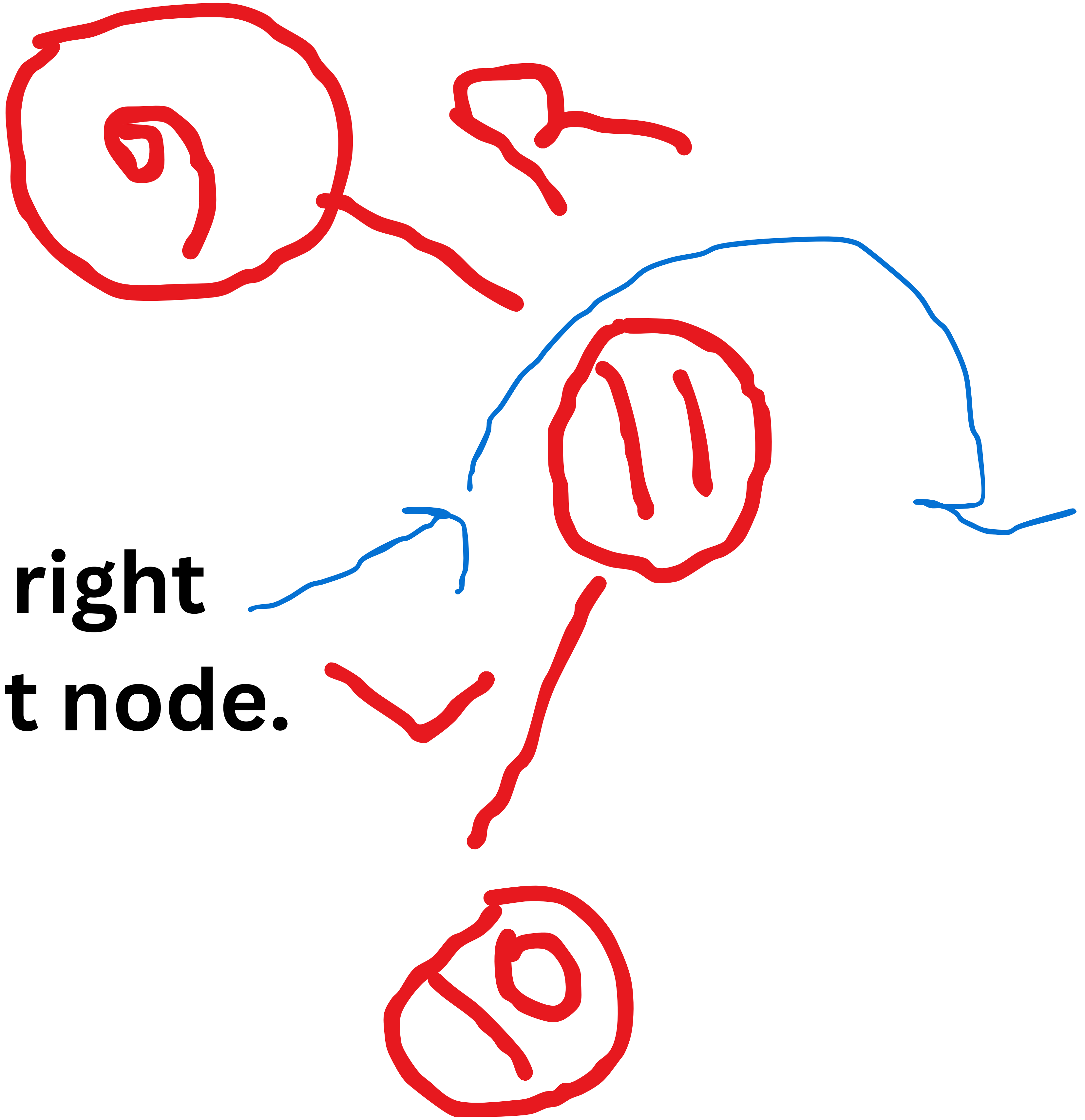
This is called the Right - left sitution.

If we perform the left rotation on here.

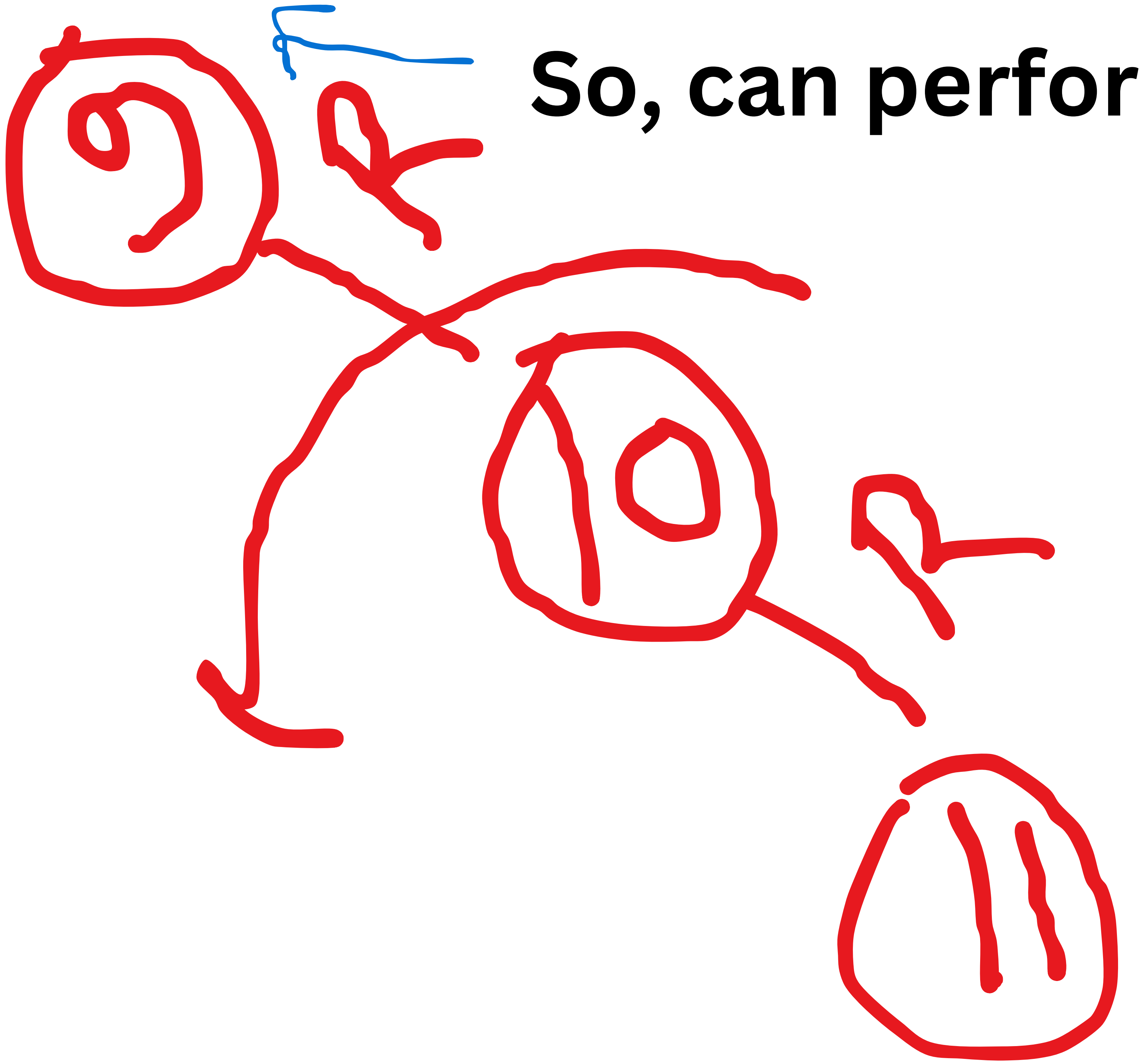


This violate the BST tree rules.

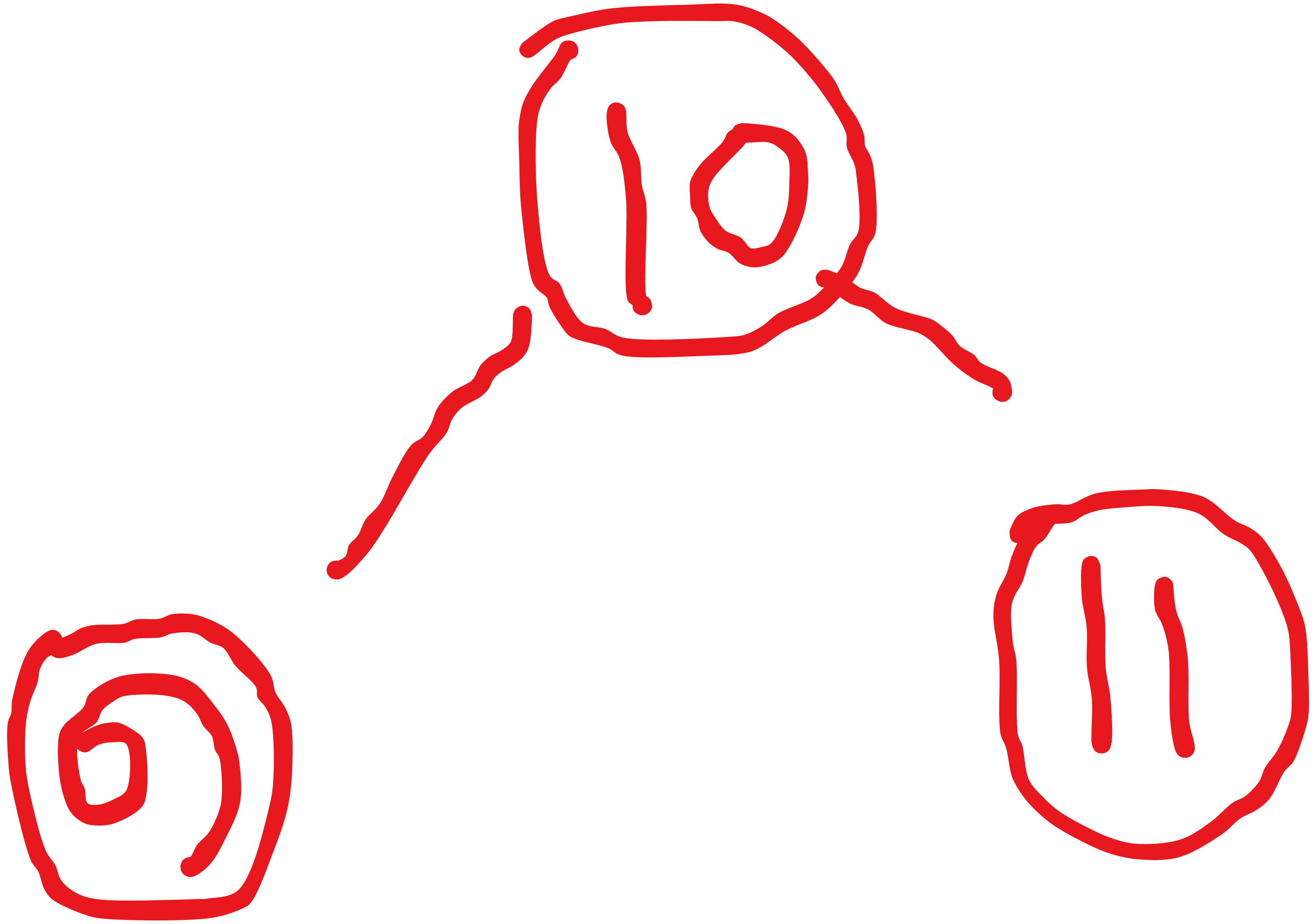
First perform the right rotation on the right node.



So, can perform the left rotation.



Now, we get the right-right sitution.


10
9
11

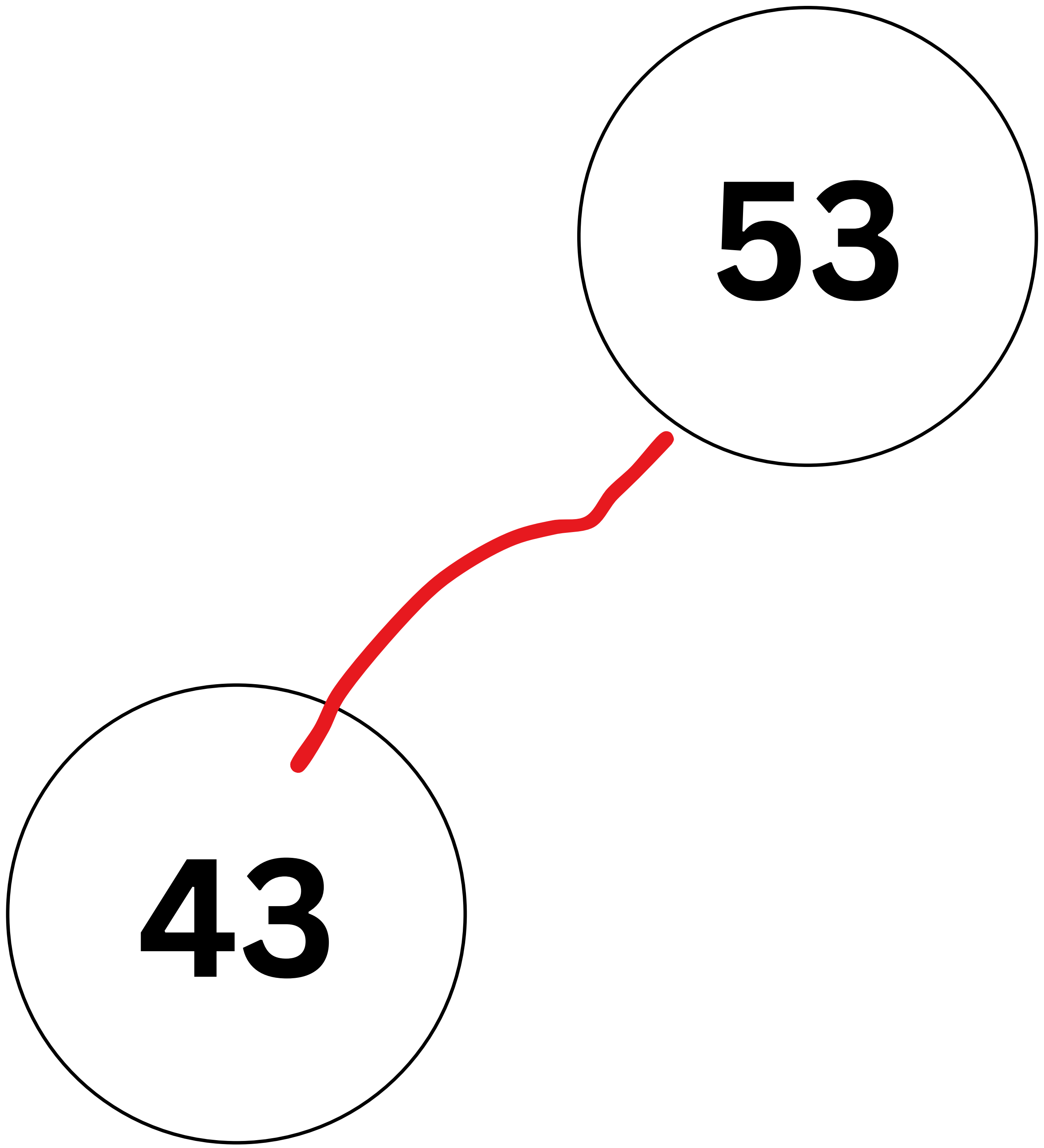
- **If the left-right sitution occur then perform the left rotation on the left node. 0 > balance(left)**
- **if the right-left sitution occur then perform the right rotation on the right node.balance(right) > 0**

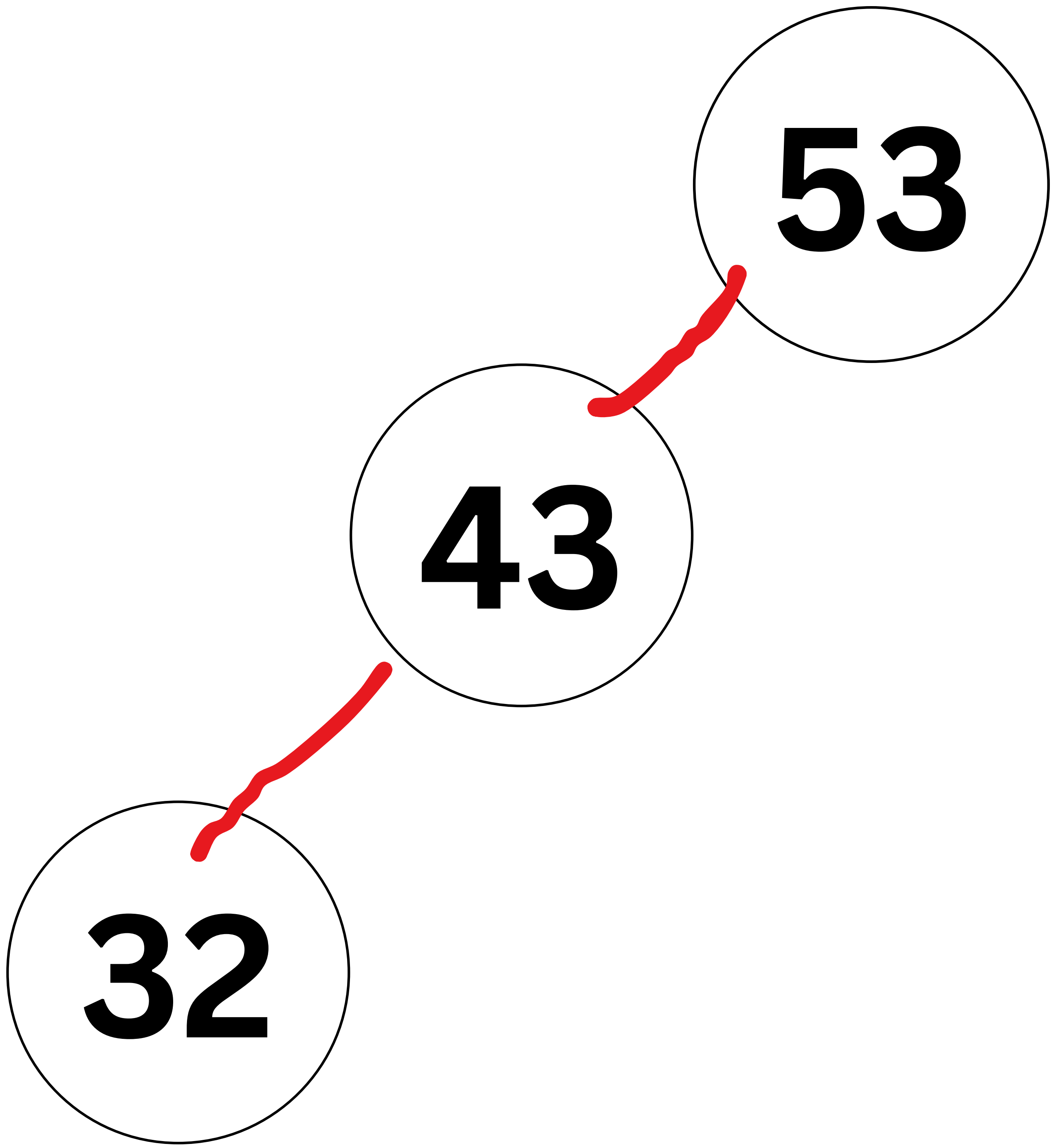
- Then mainly the left-left sitution occur and can perform the right rotation.
- Then mainly the right-right sitution occur and can perform the left rotation.

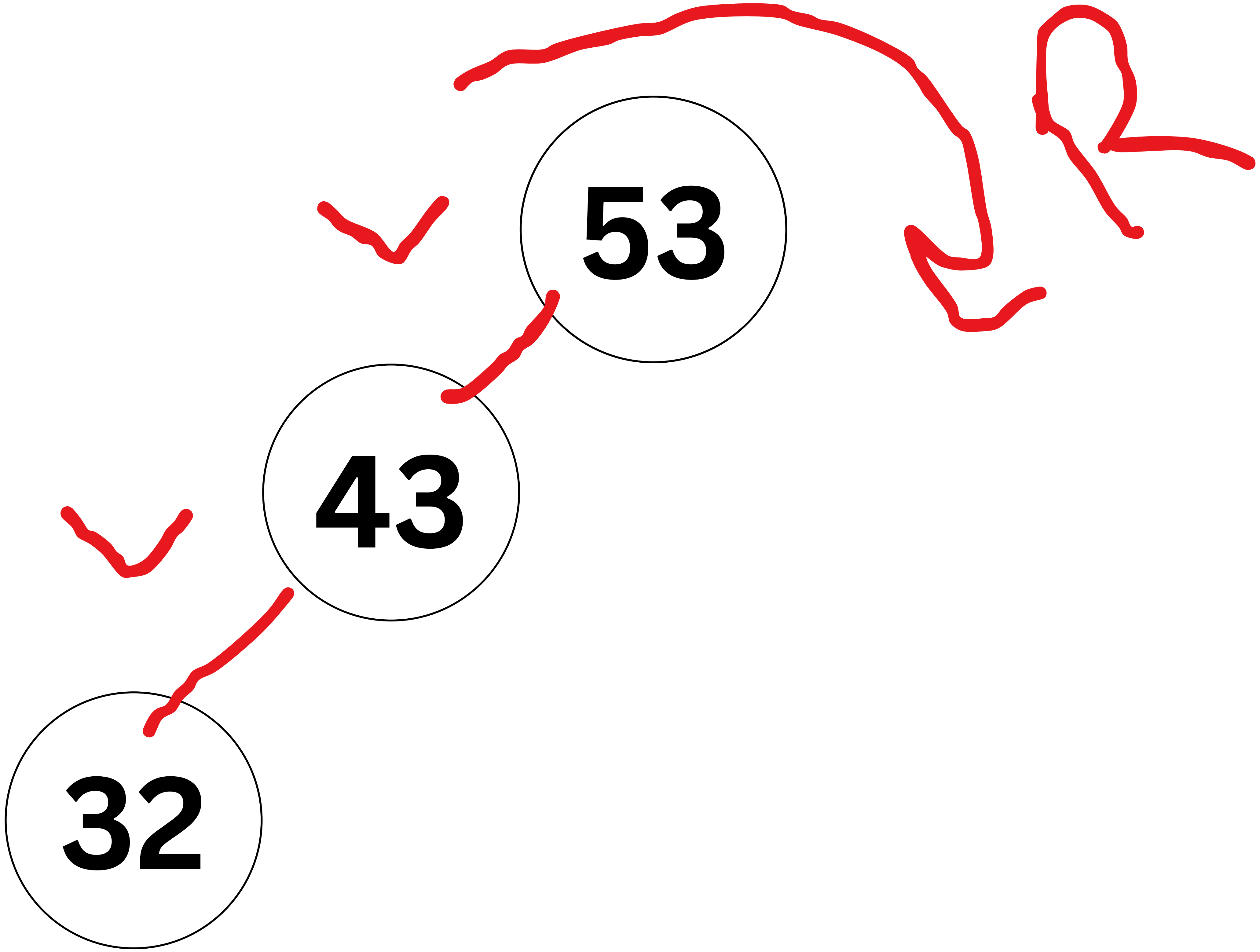
Let’s Solve an example.

53, 43, 32, 12, 23, 33, 70, 60,65, 83,
10, 9, 2

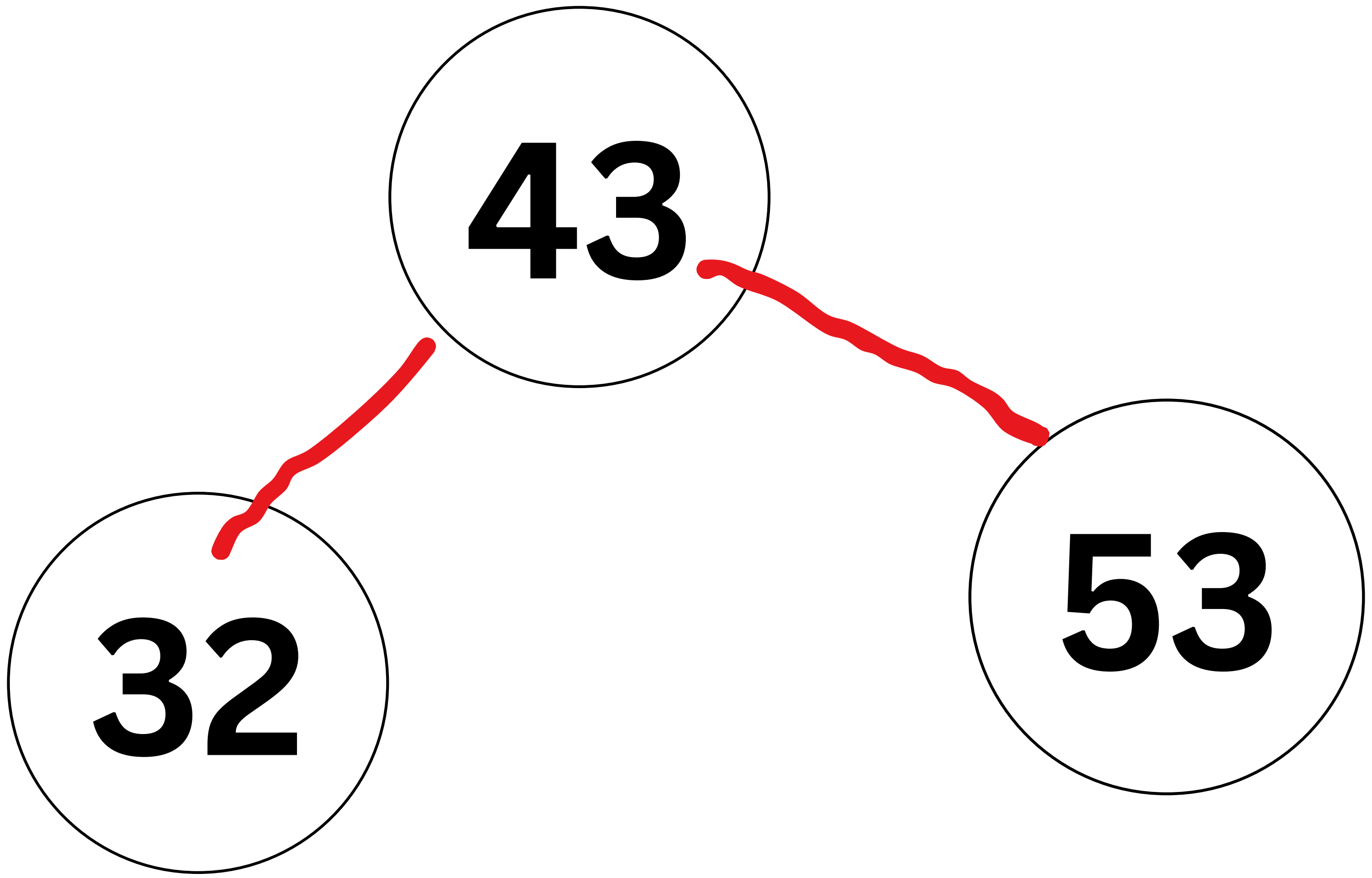
53

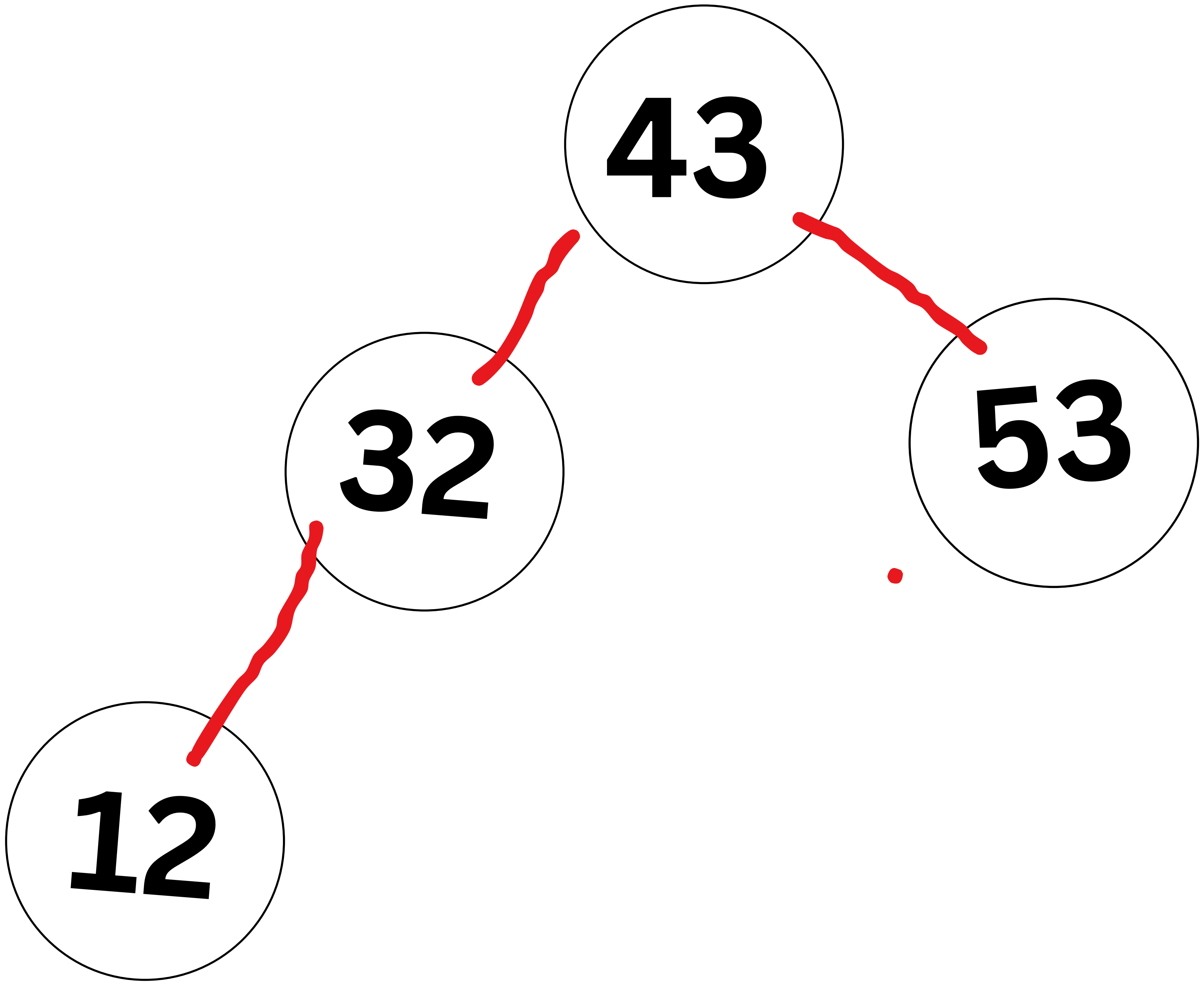

53
43

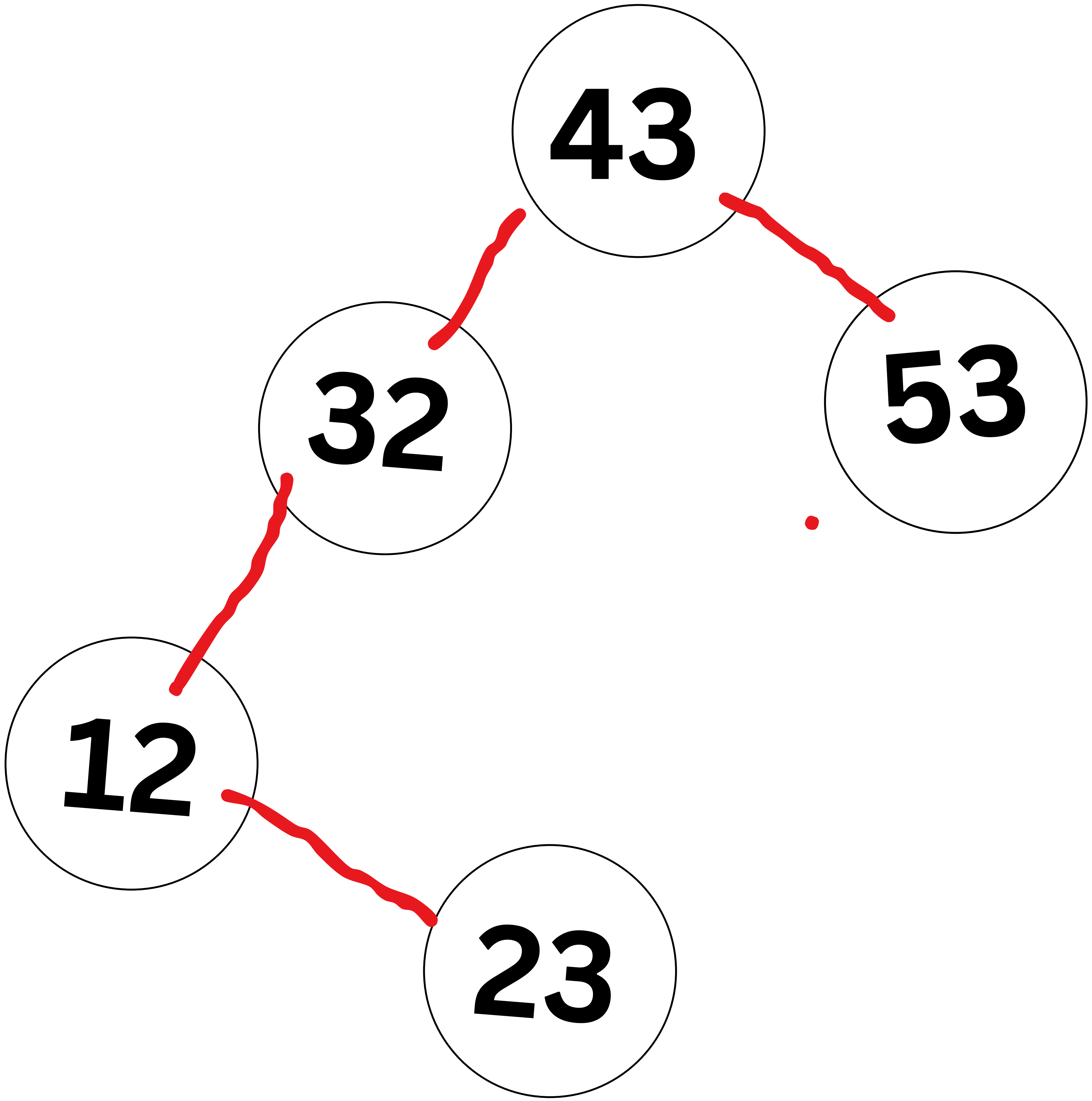

53
43
32



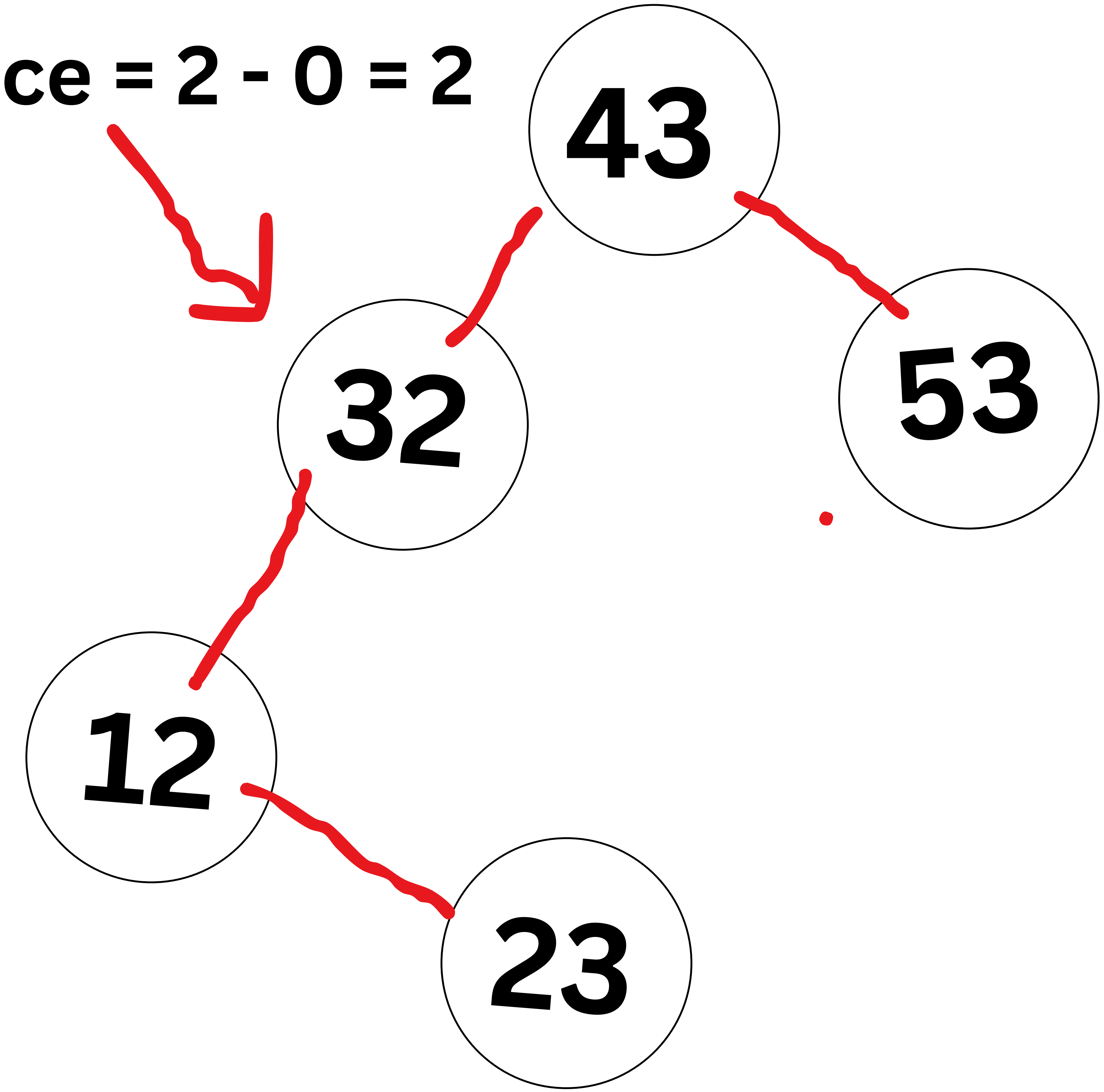
53
43
32
R


43
32
53

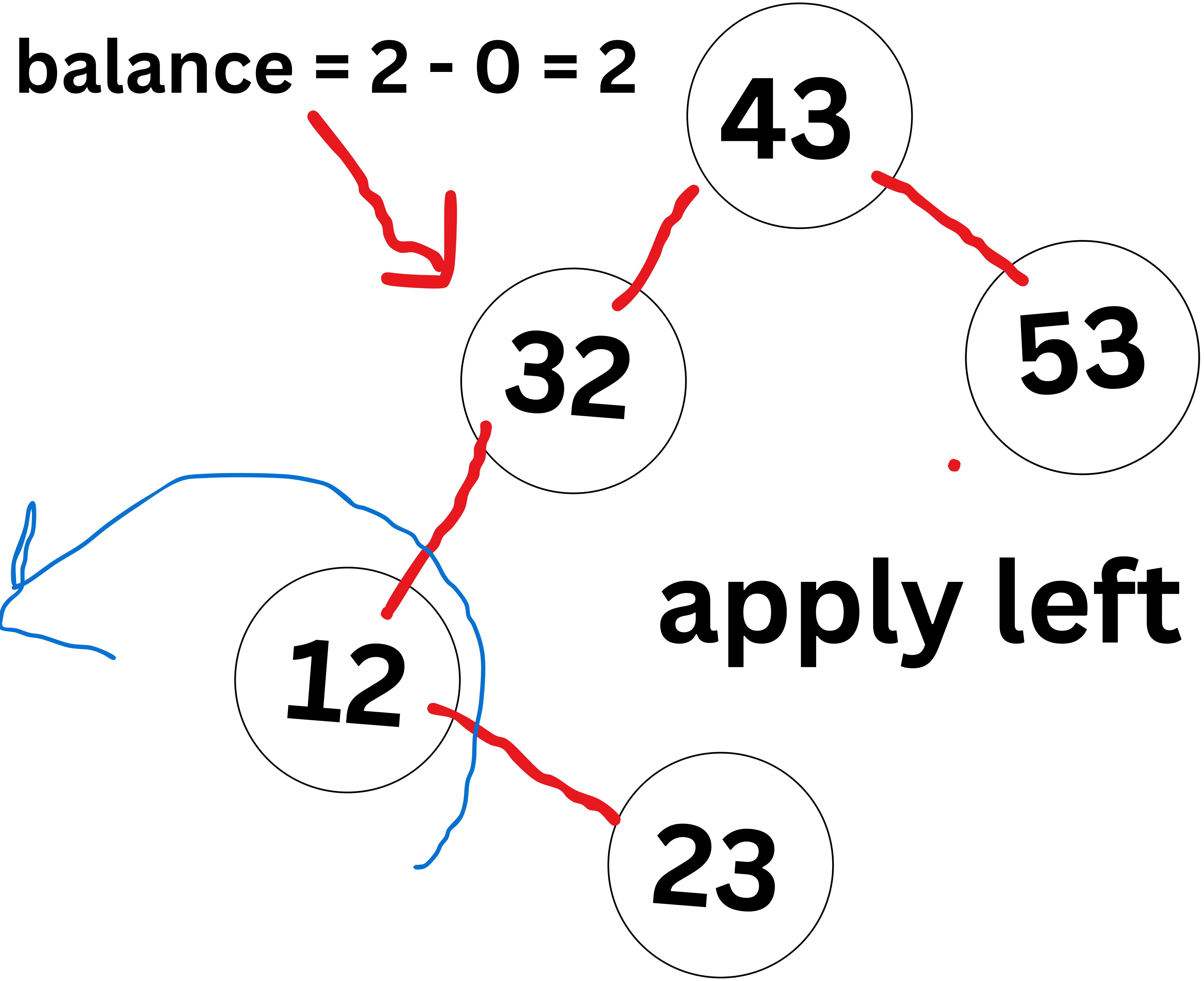

43
32
53
12


43
32
53
12
23

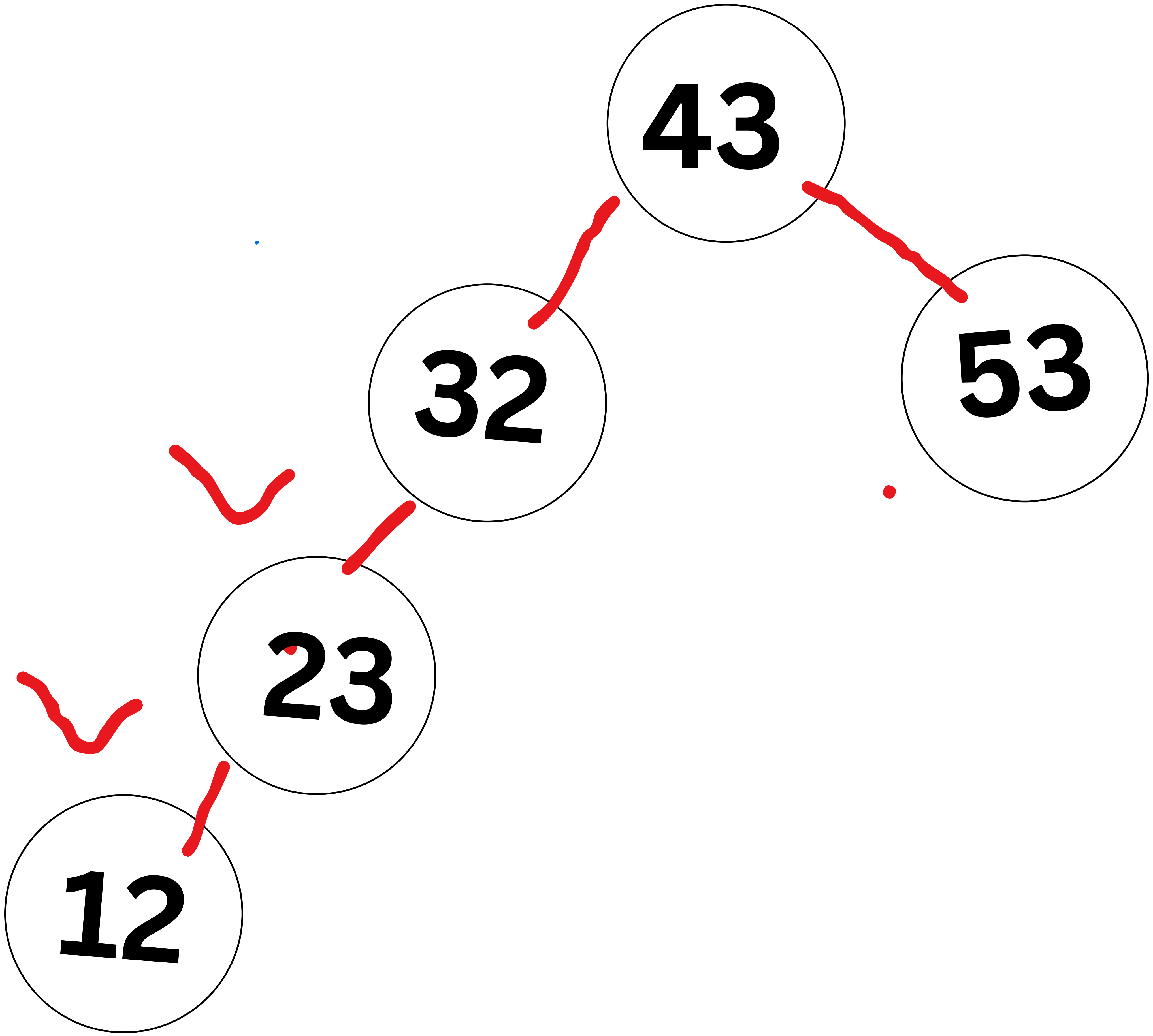
balance = 2 - 0 = 2

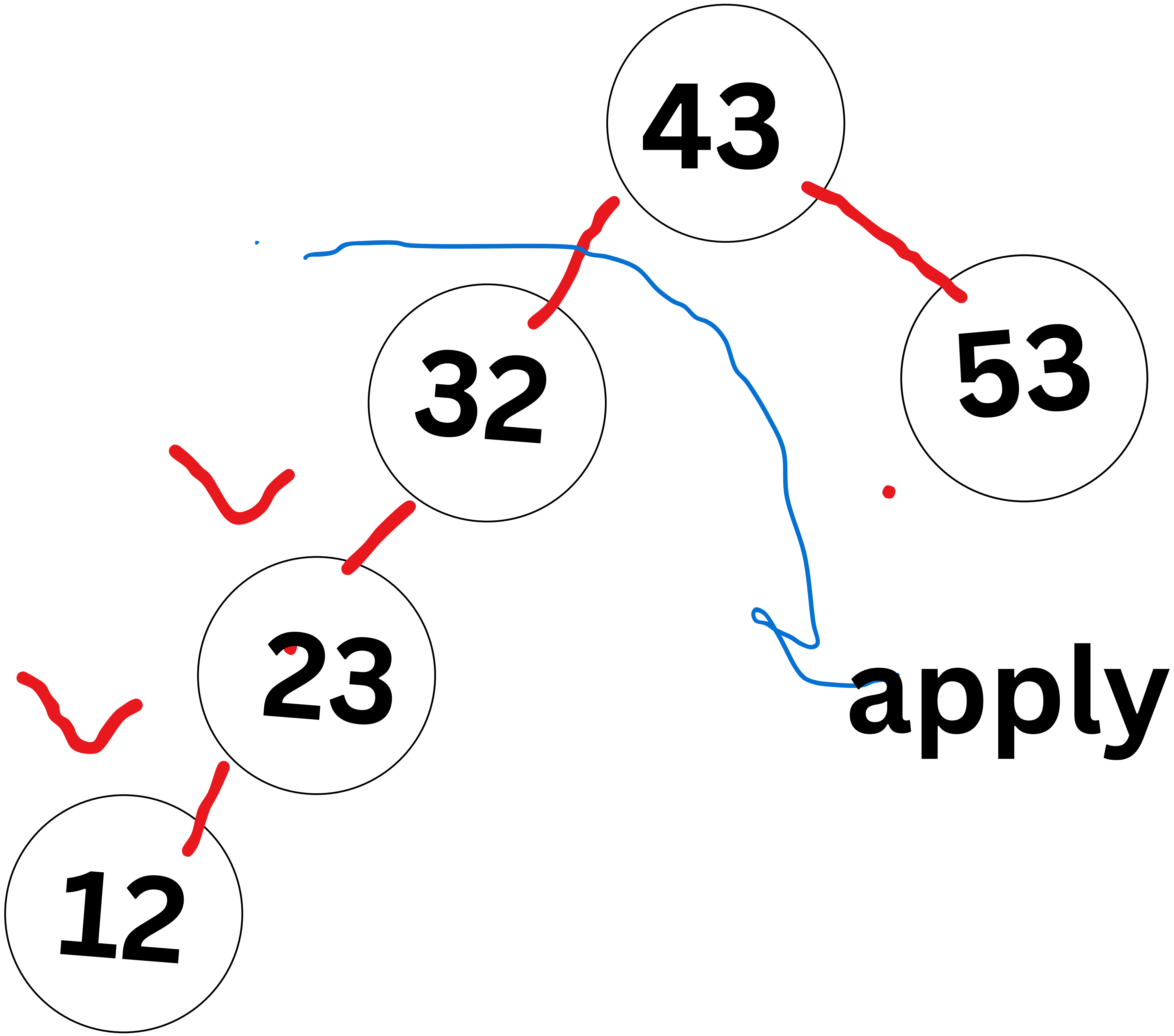


balance = 2 - 0 = 2

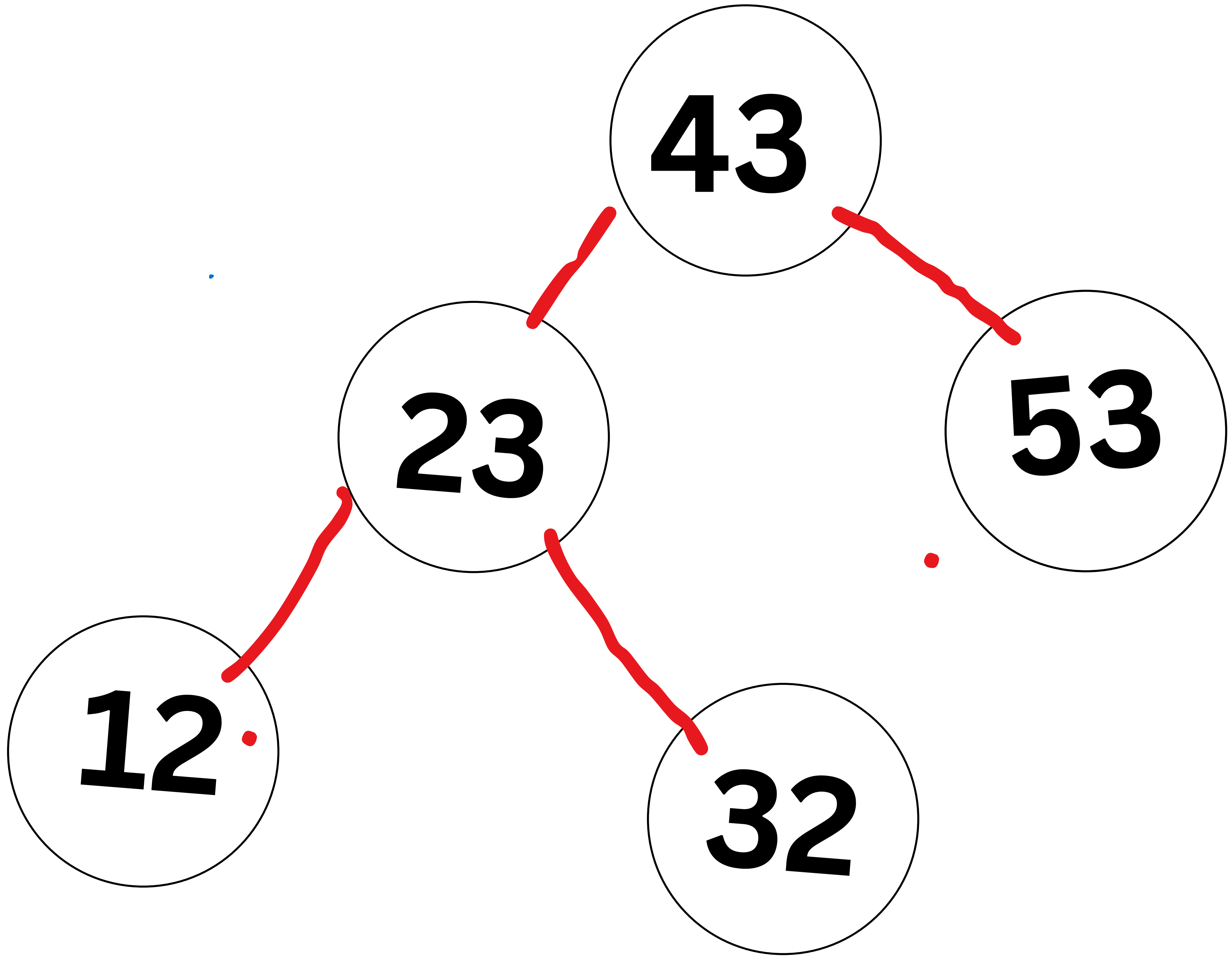


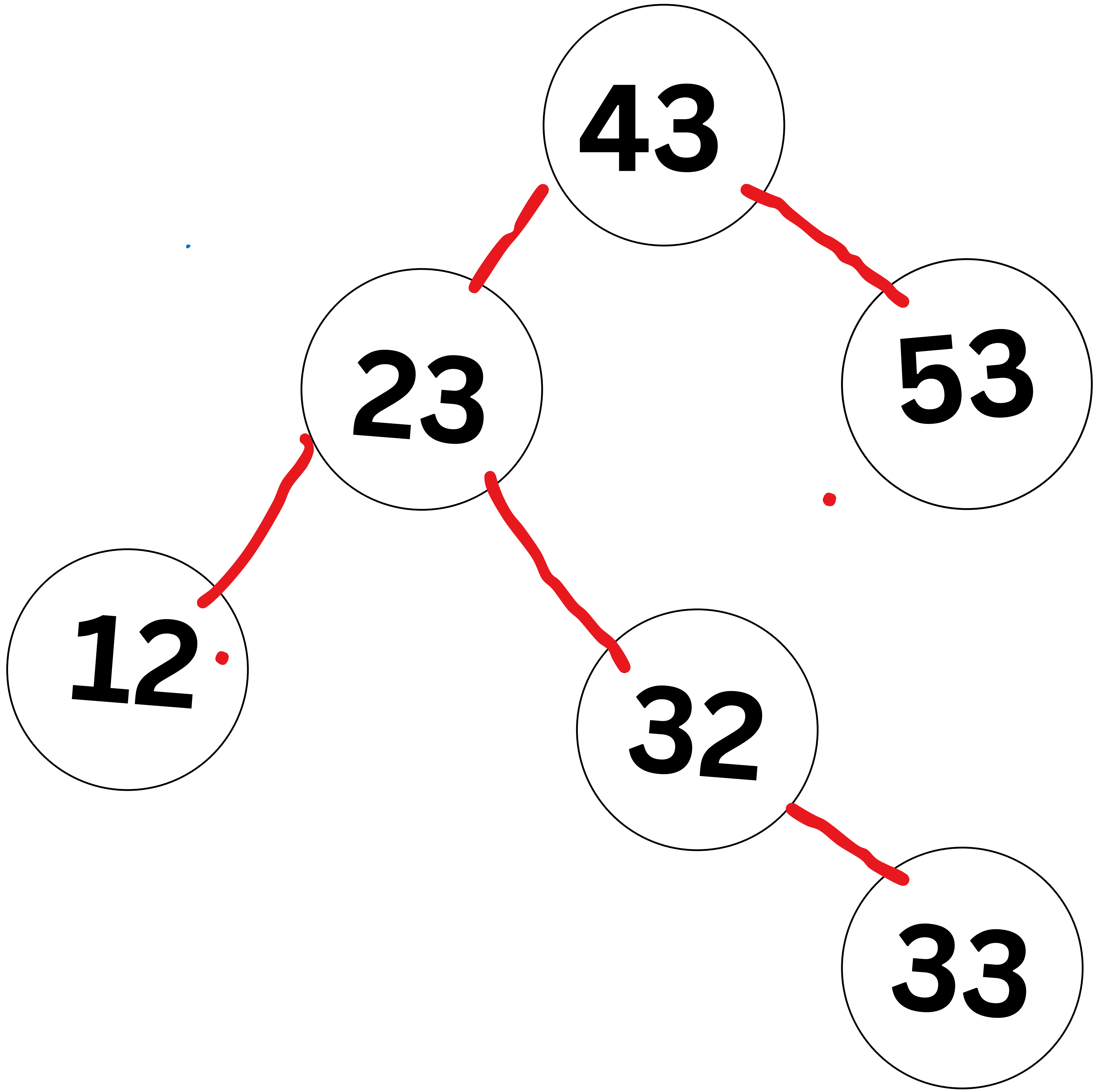
apply left on the 12


43
32
53
23
12

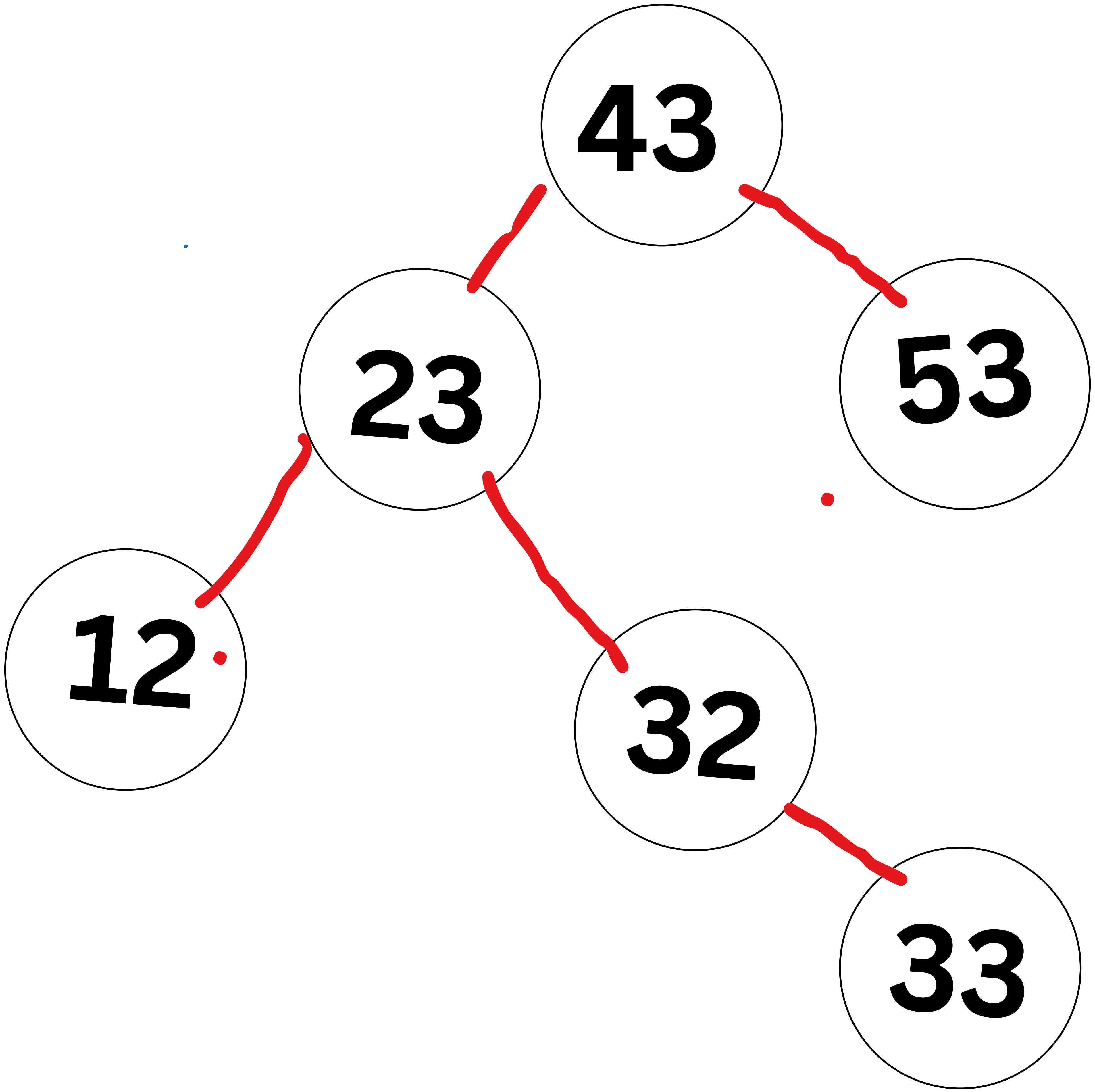


apply right on 32

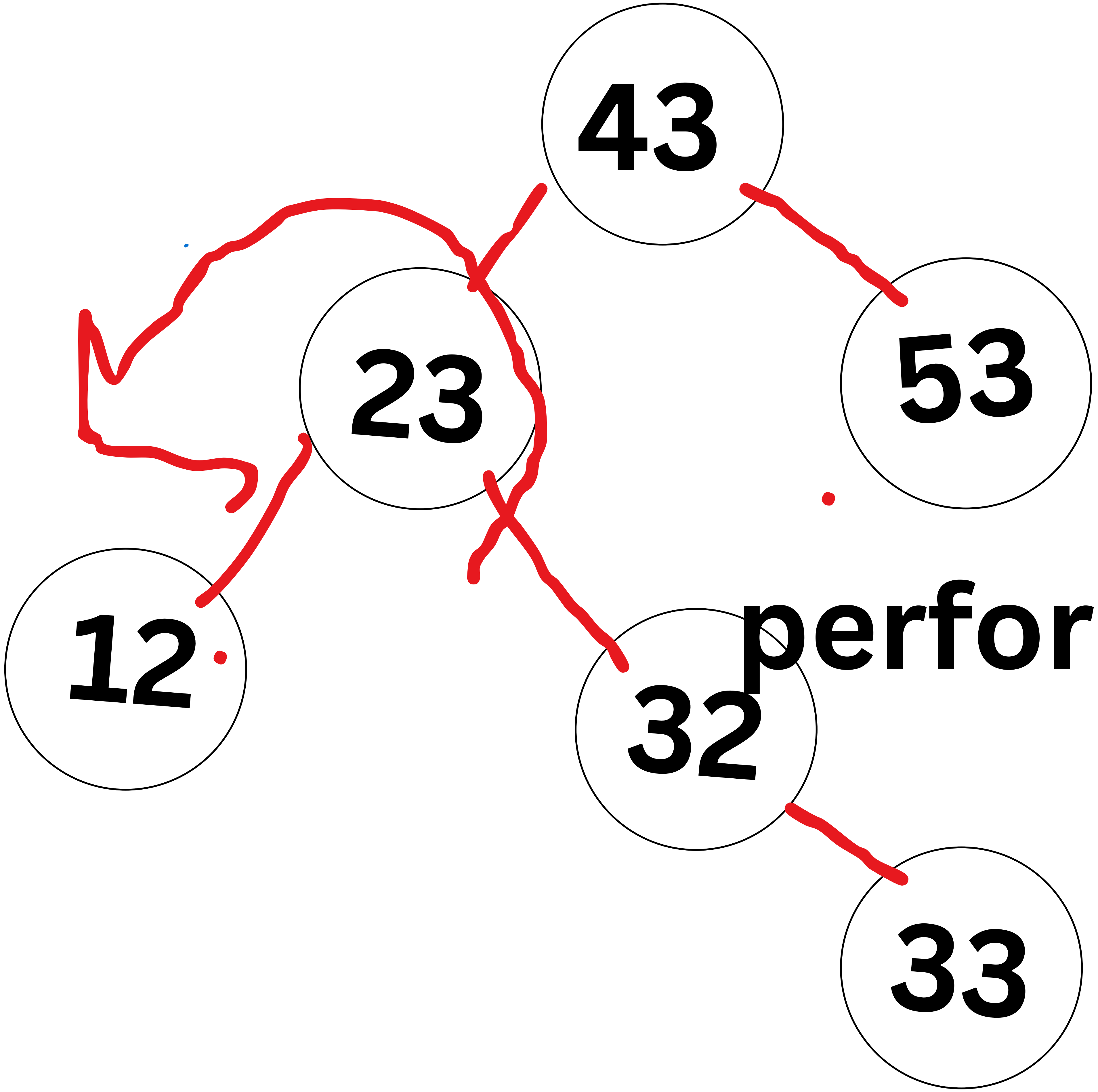

43
23
53
12
32


43
23
53
12
32
33

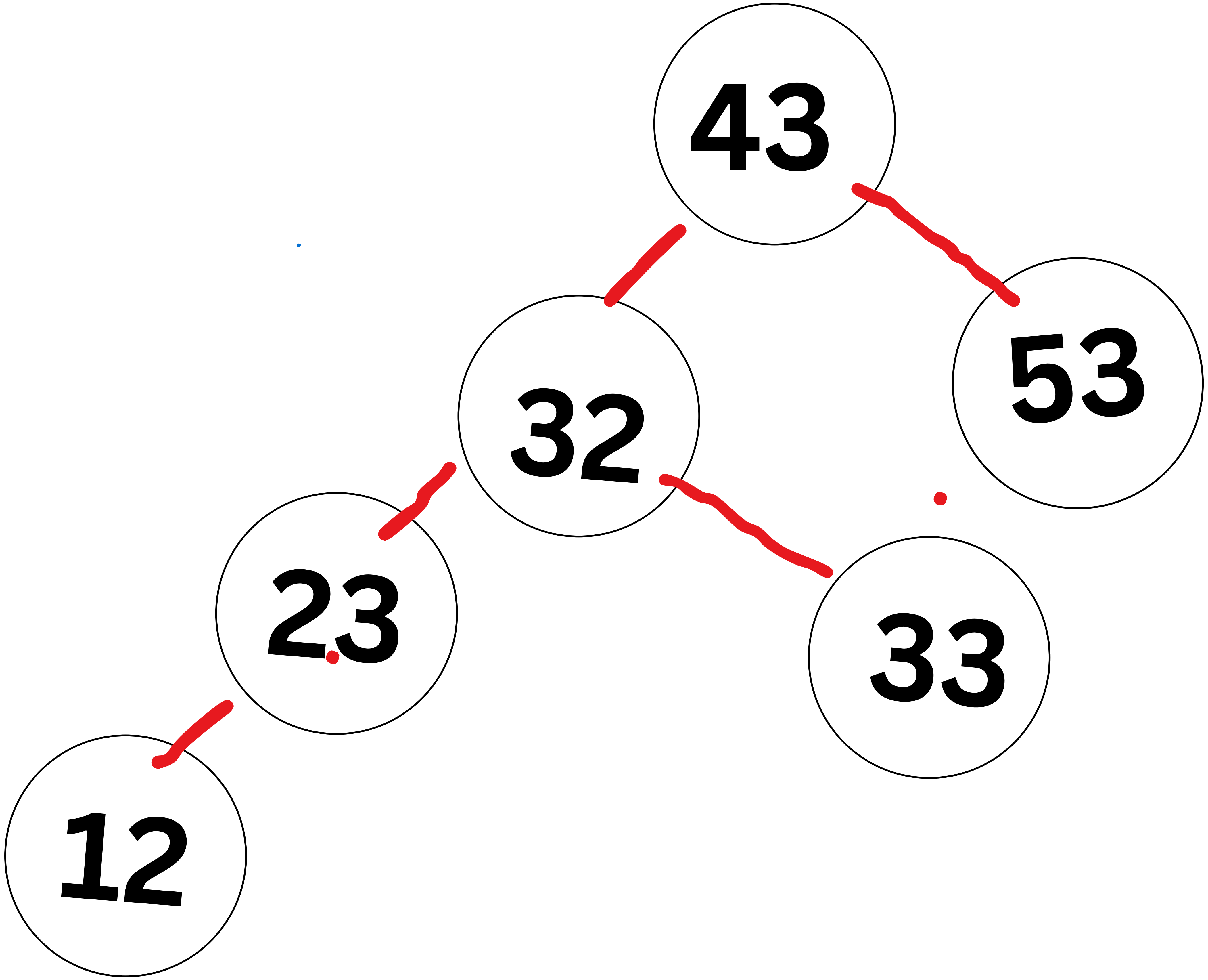
balance = 3 - 1 = 2



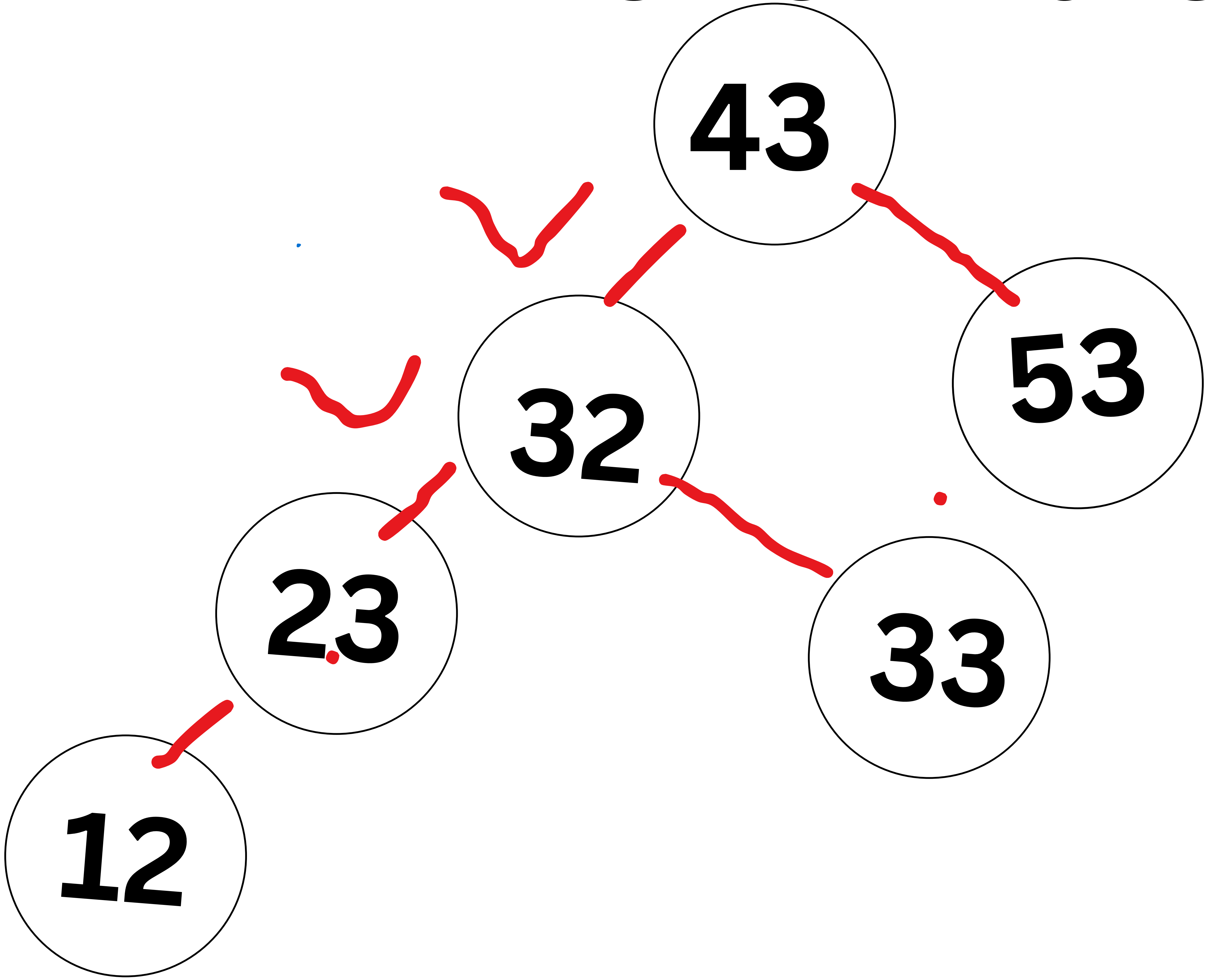
balance = 3 - 1 = 2

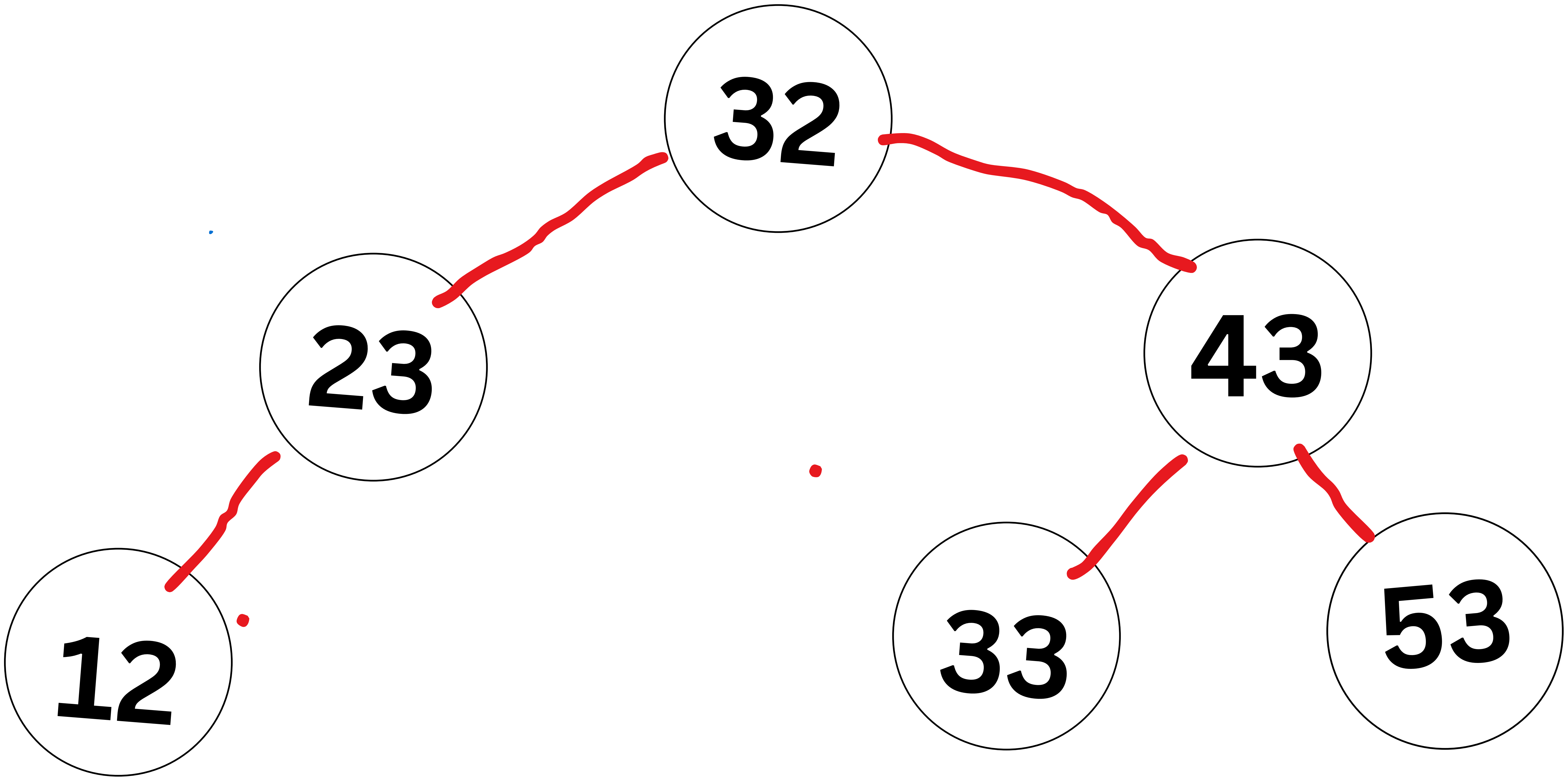


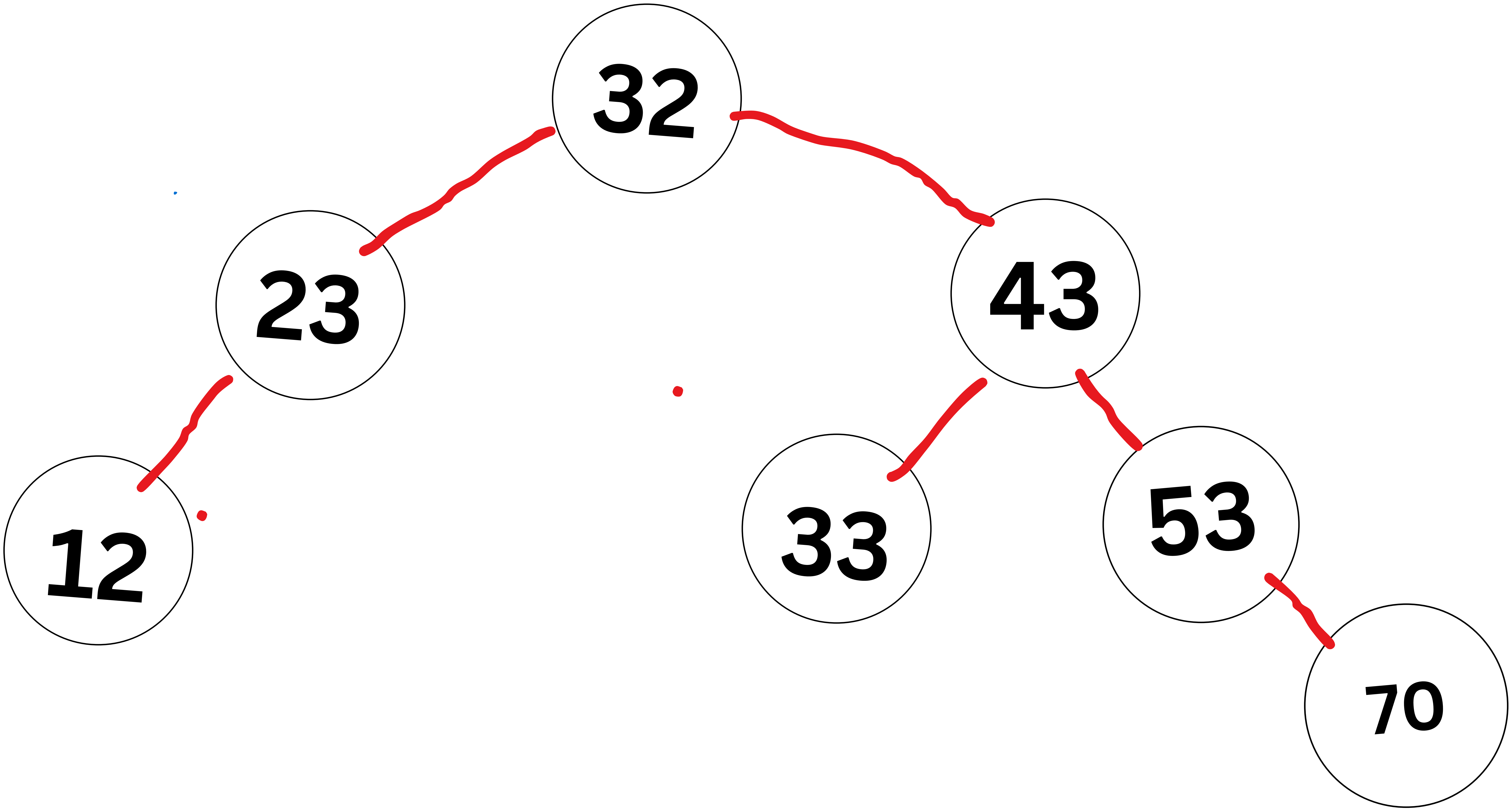
perform left on the 23

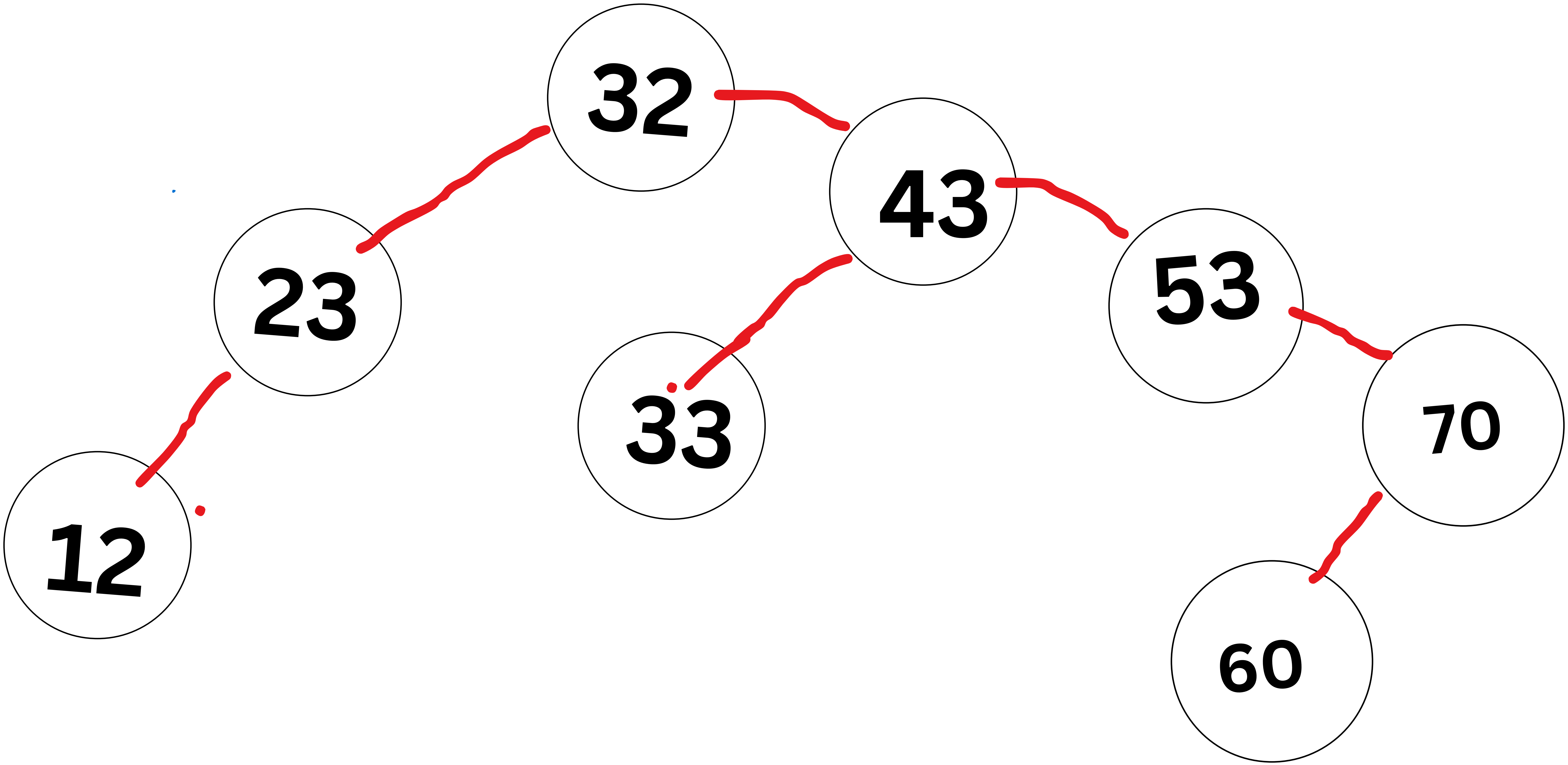

43
32
53
23
33
12

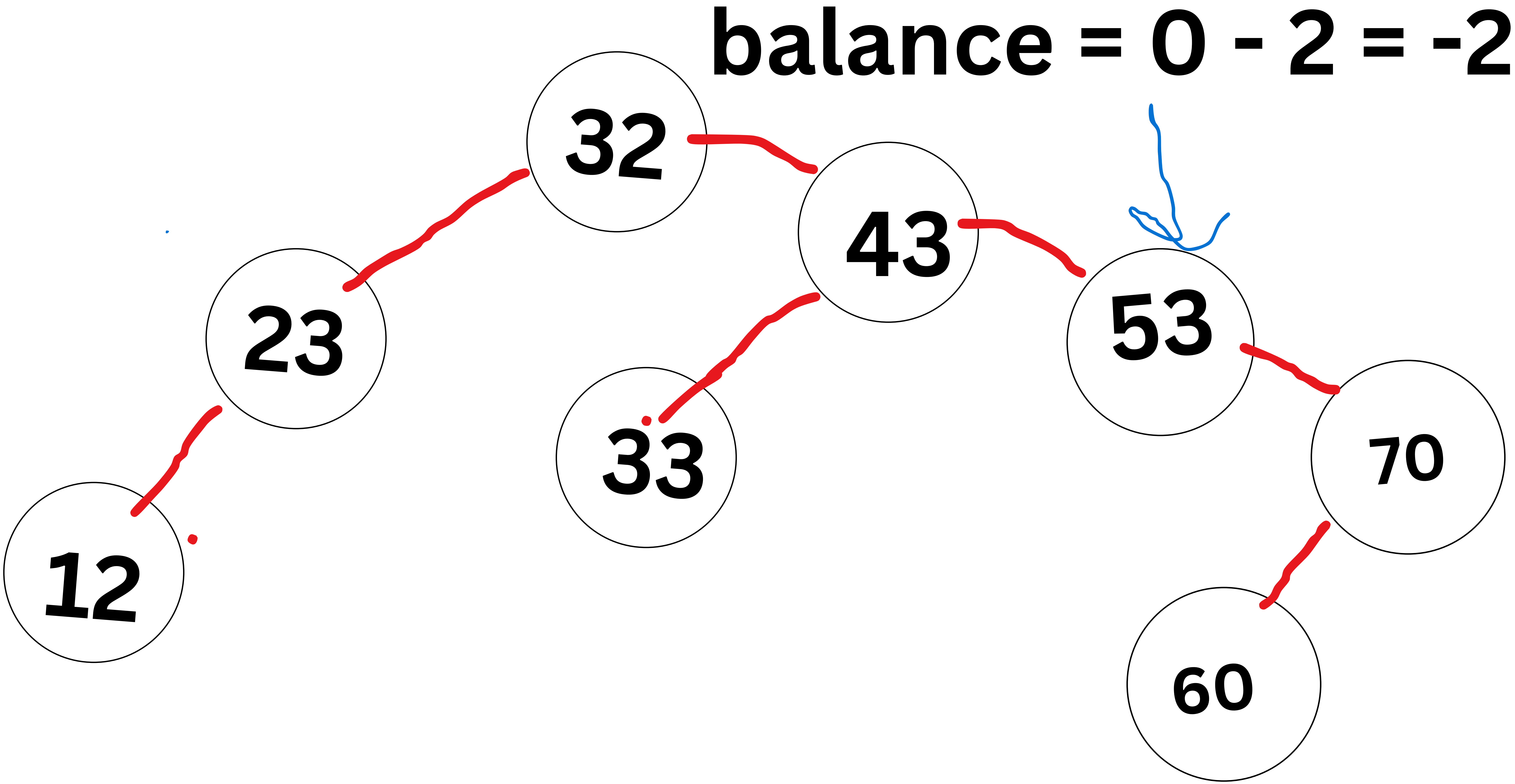
Perform the right on 43



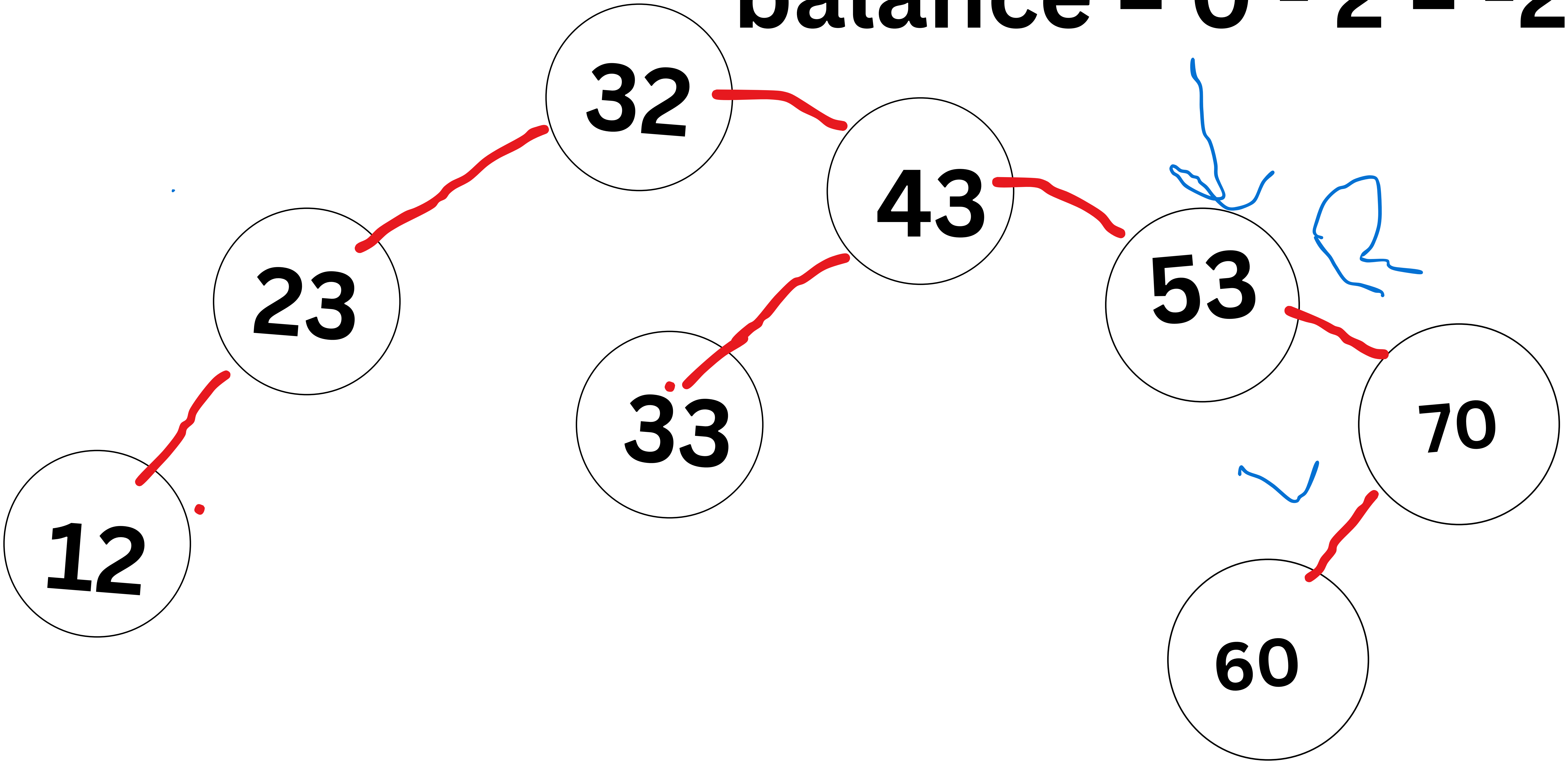

32
23
43
12
33
53


32
23
43
12
33
53
70

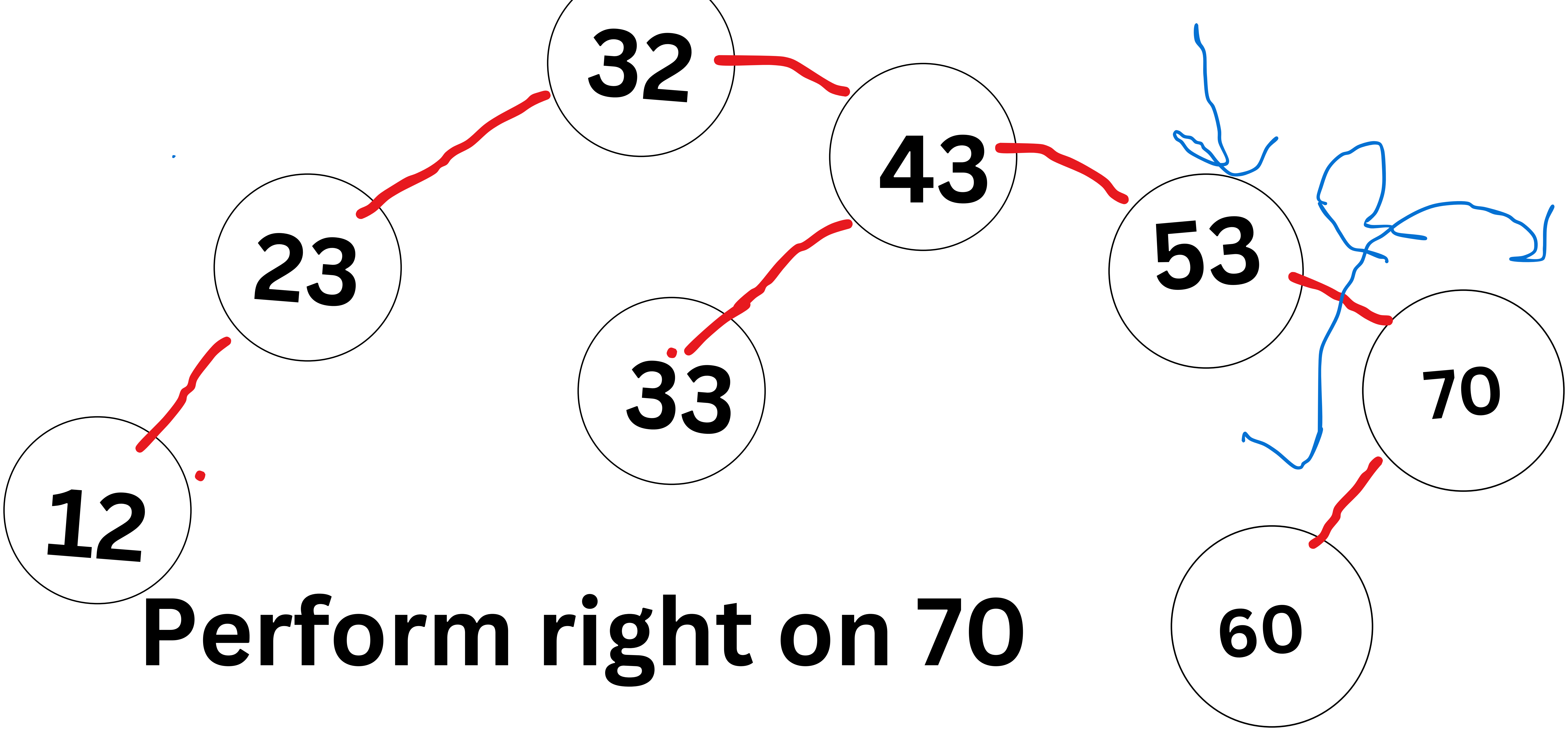

32
23
43
12
33
53
70
60


balance = 0 - 2 = -2
32
23
43
12
33
53
70
60

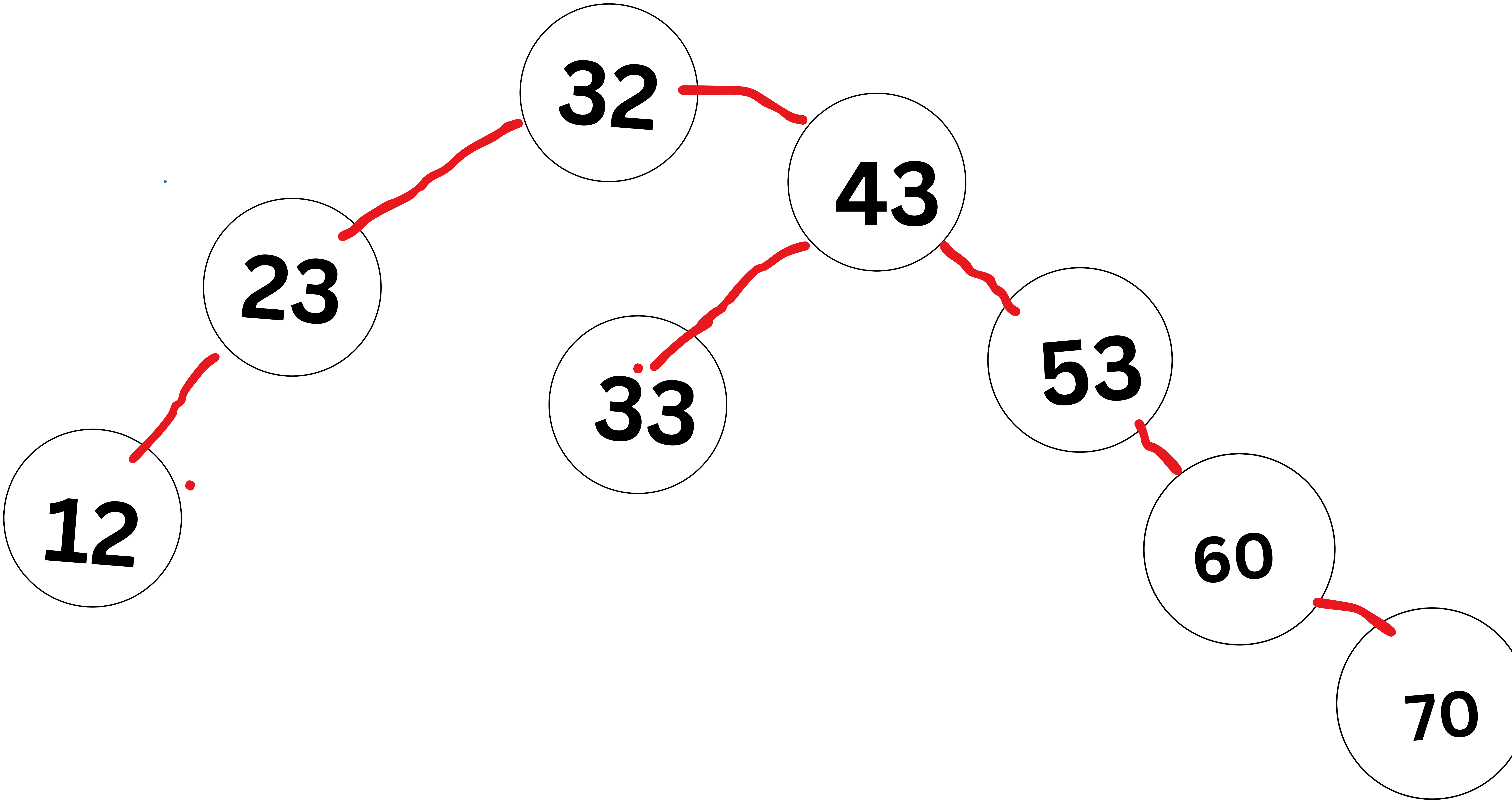
balance = 0 - 2 = -2

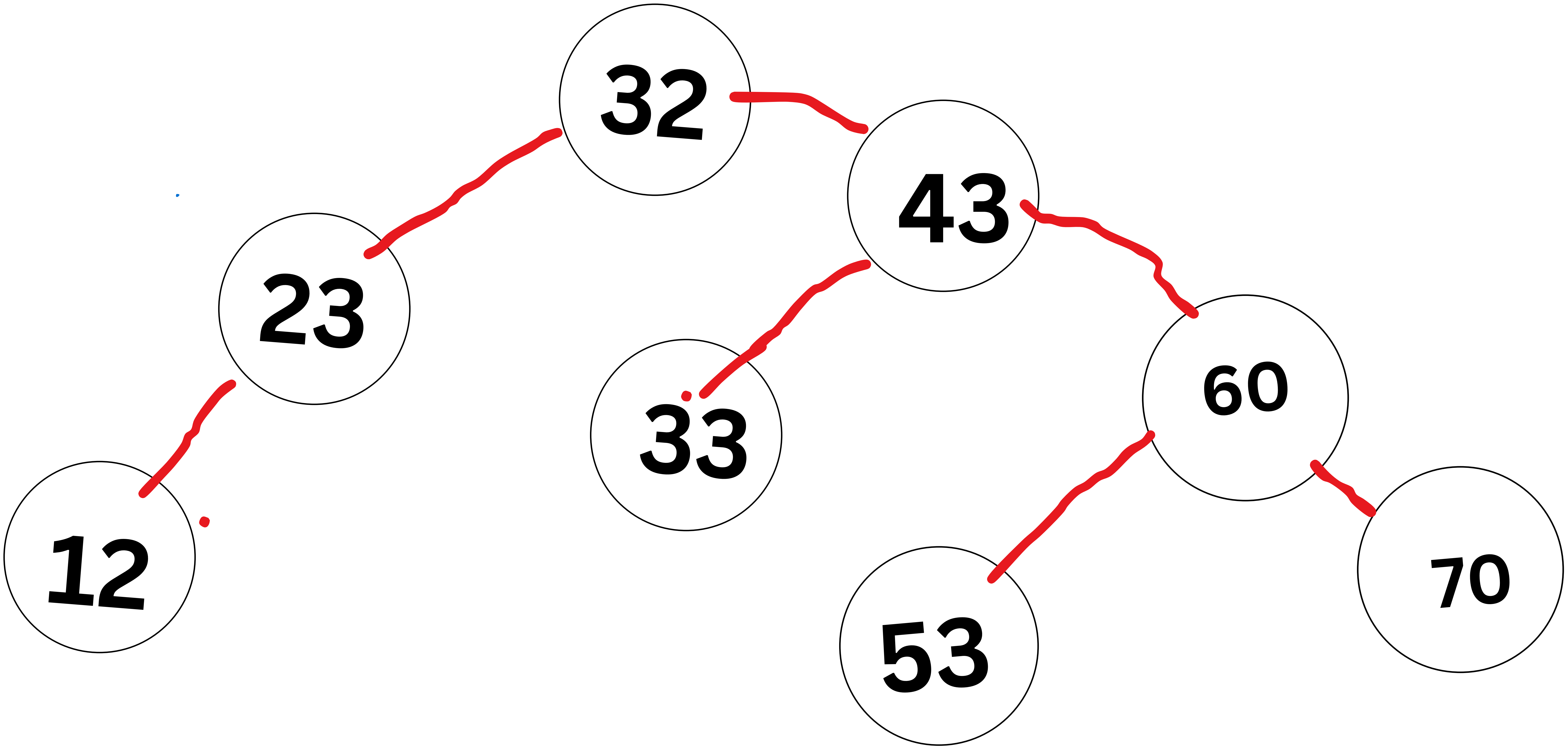


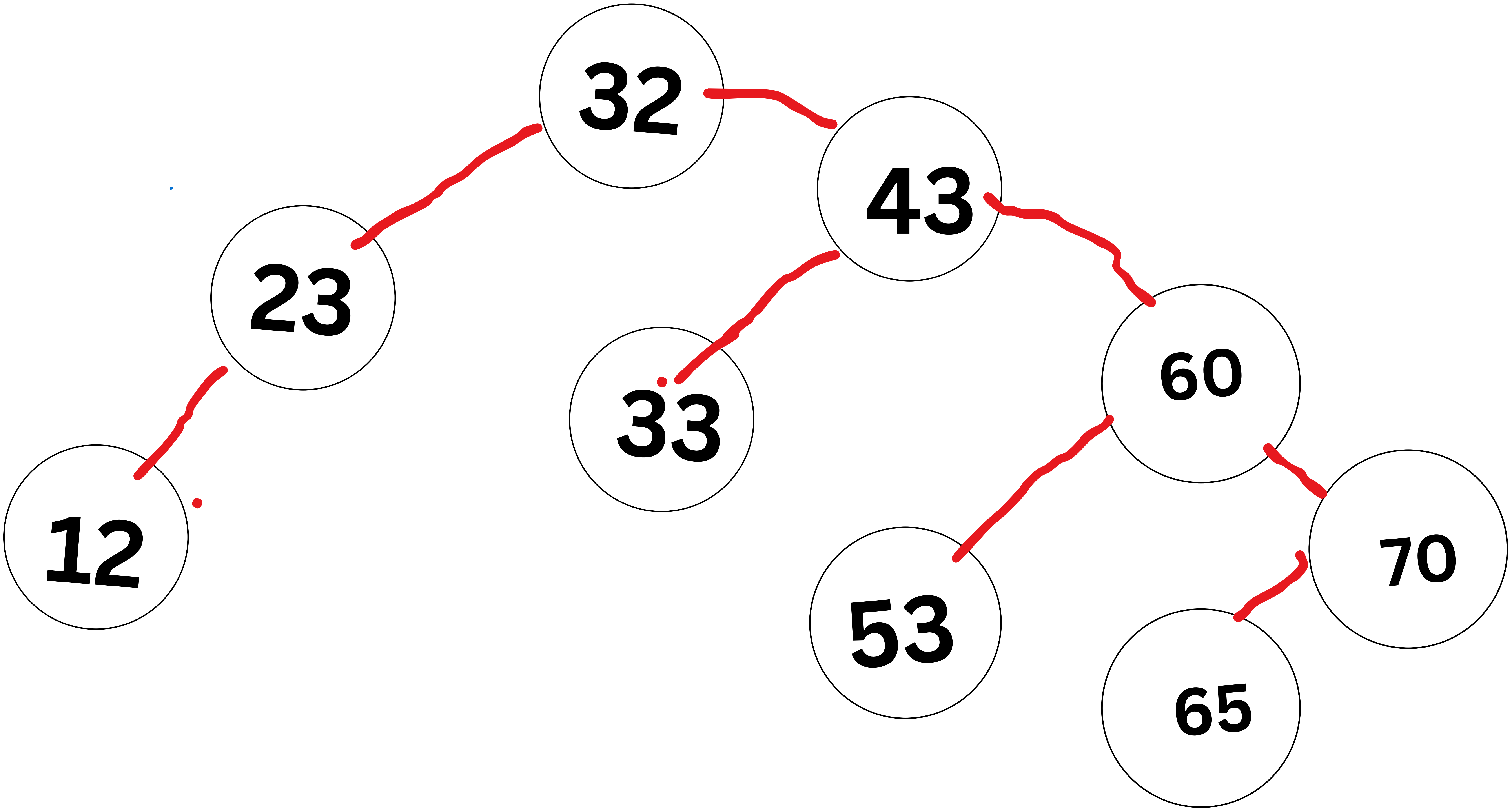
balance = 0 - 2 = -2

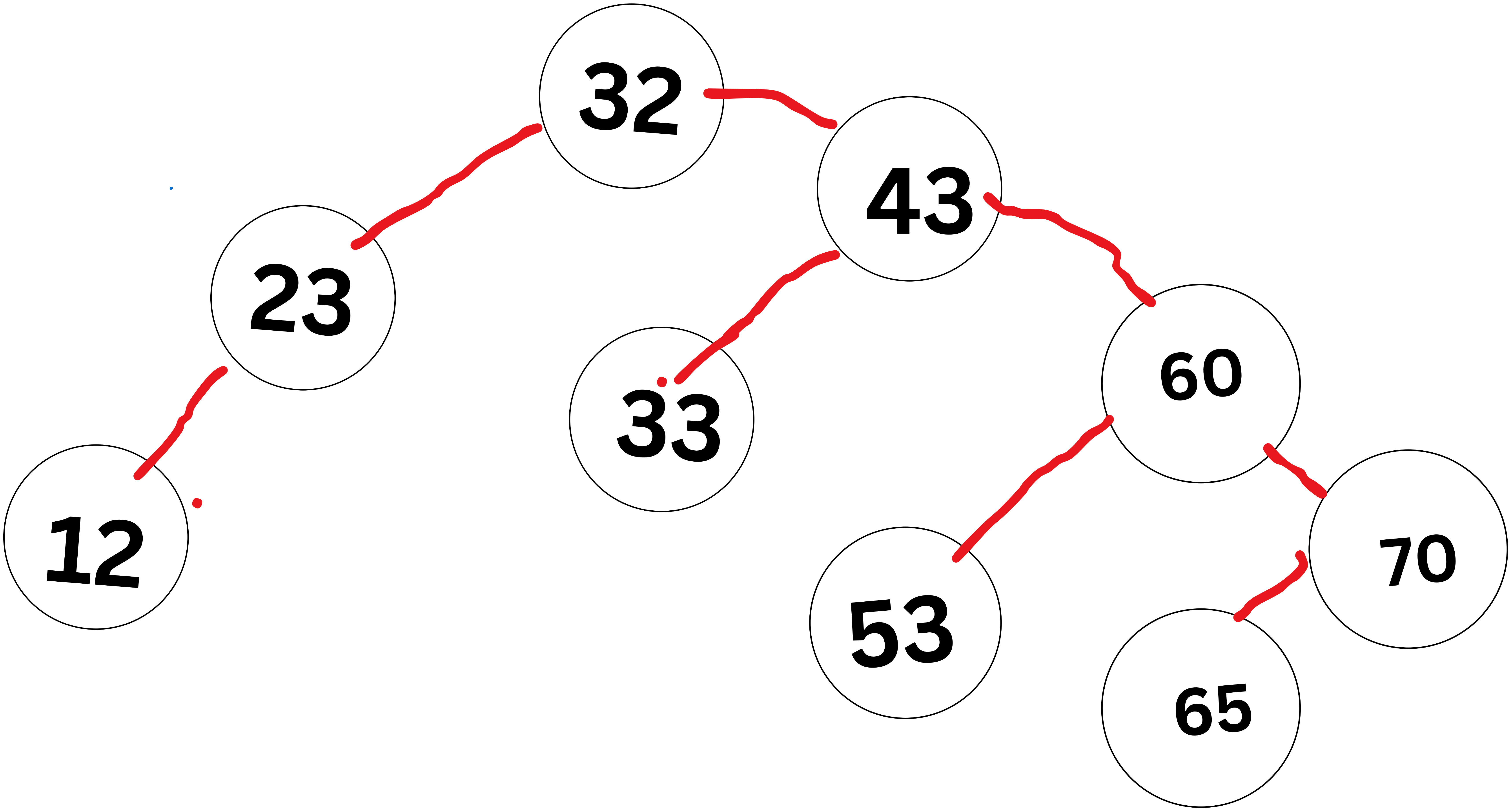


Perform right on 70

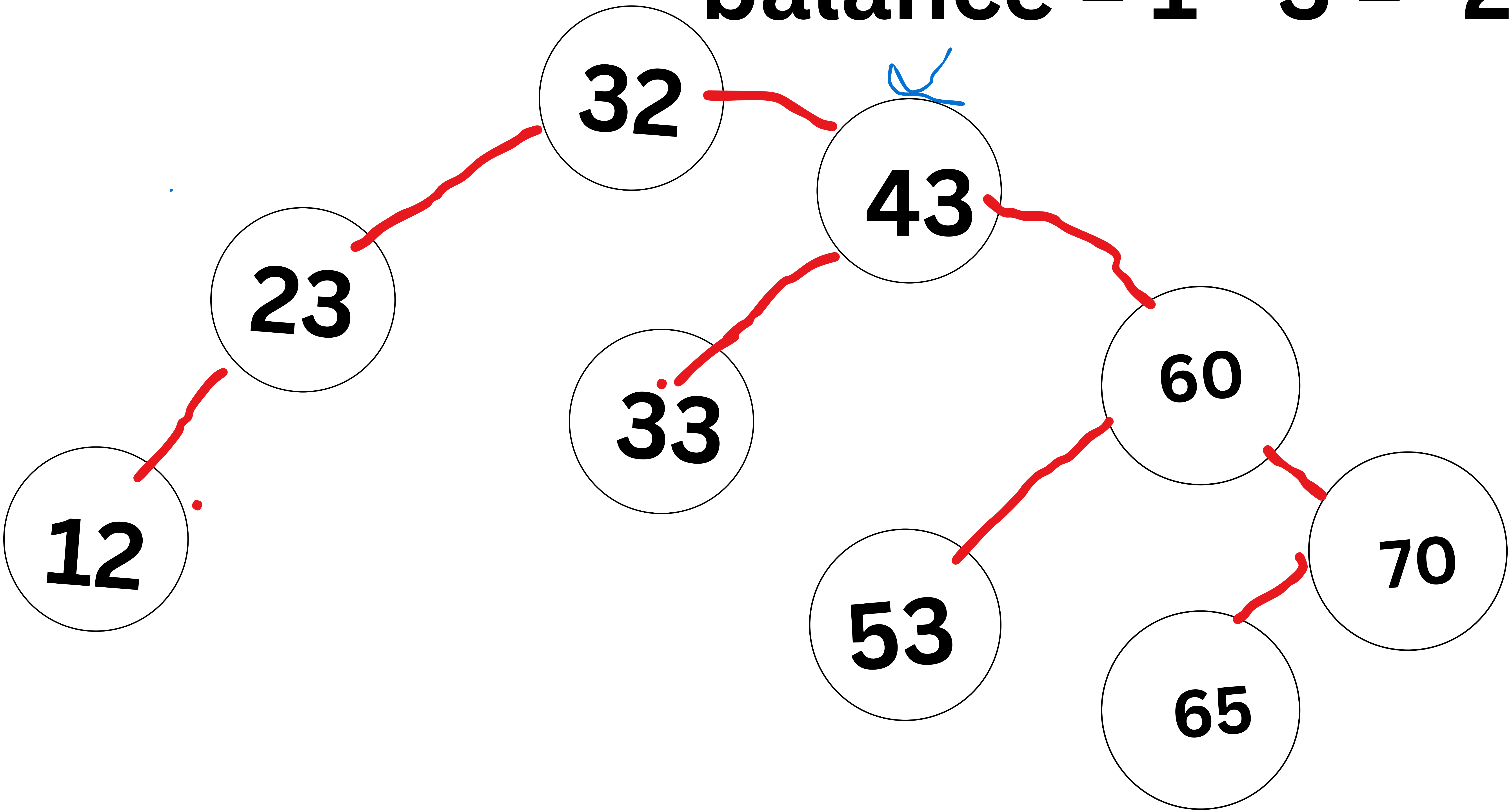

32
23
43
12
33
53
60
70


32
43
23
33
60
12
53
70

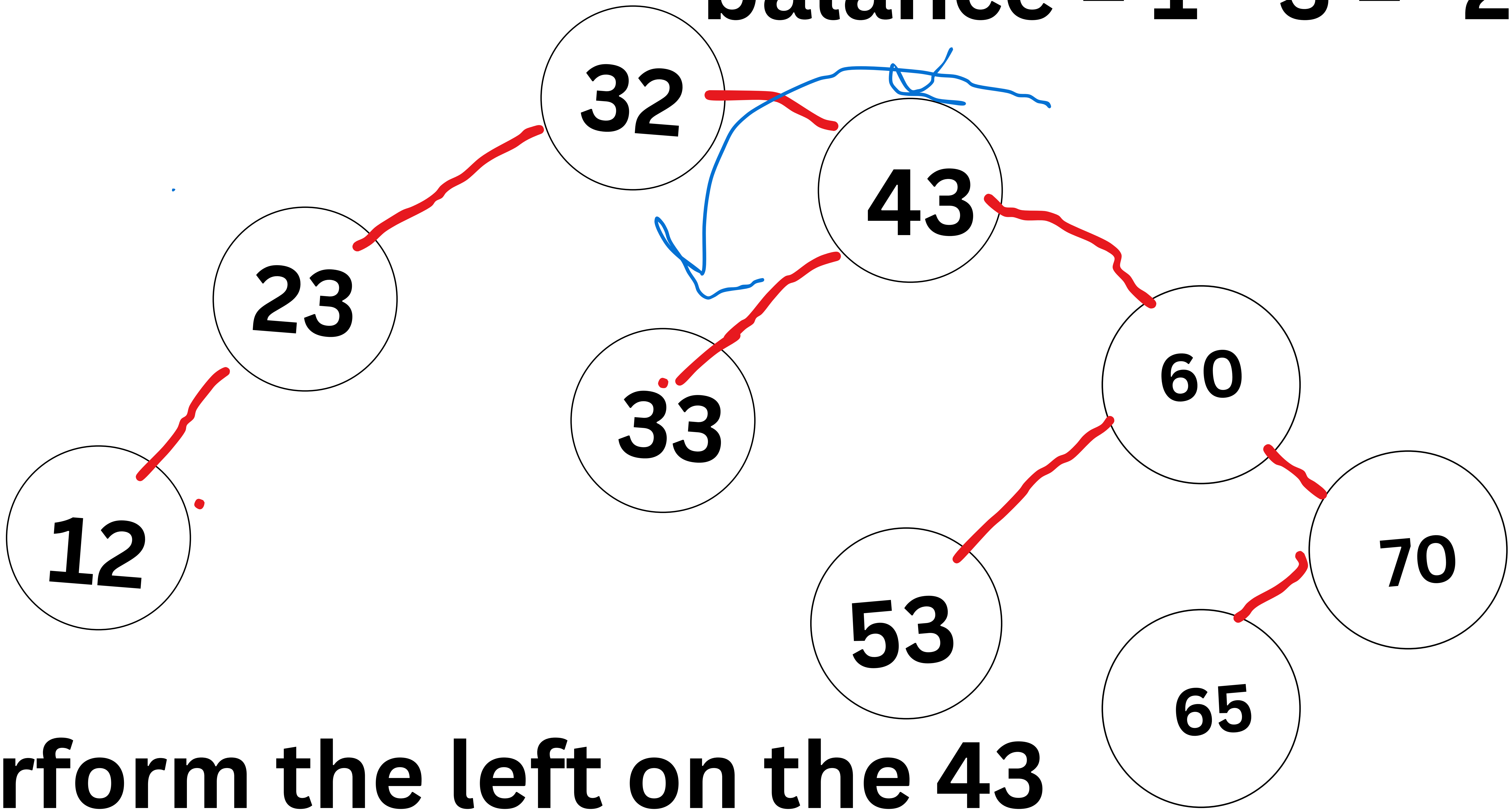

32
43
23
33
60
12
53
70
65


32
23
43
12
33
60
53
70
65

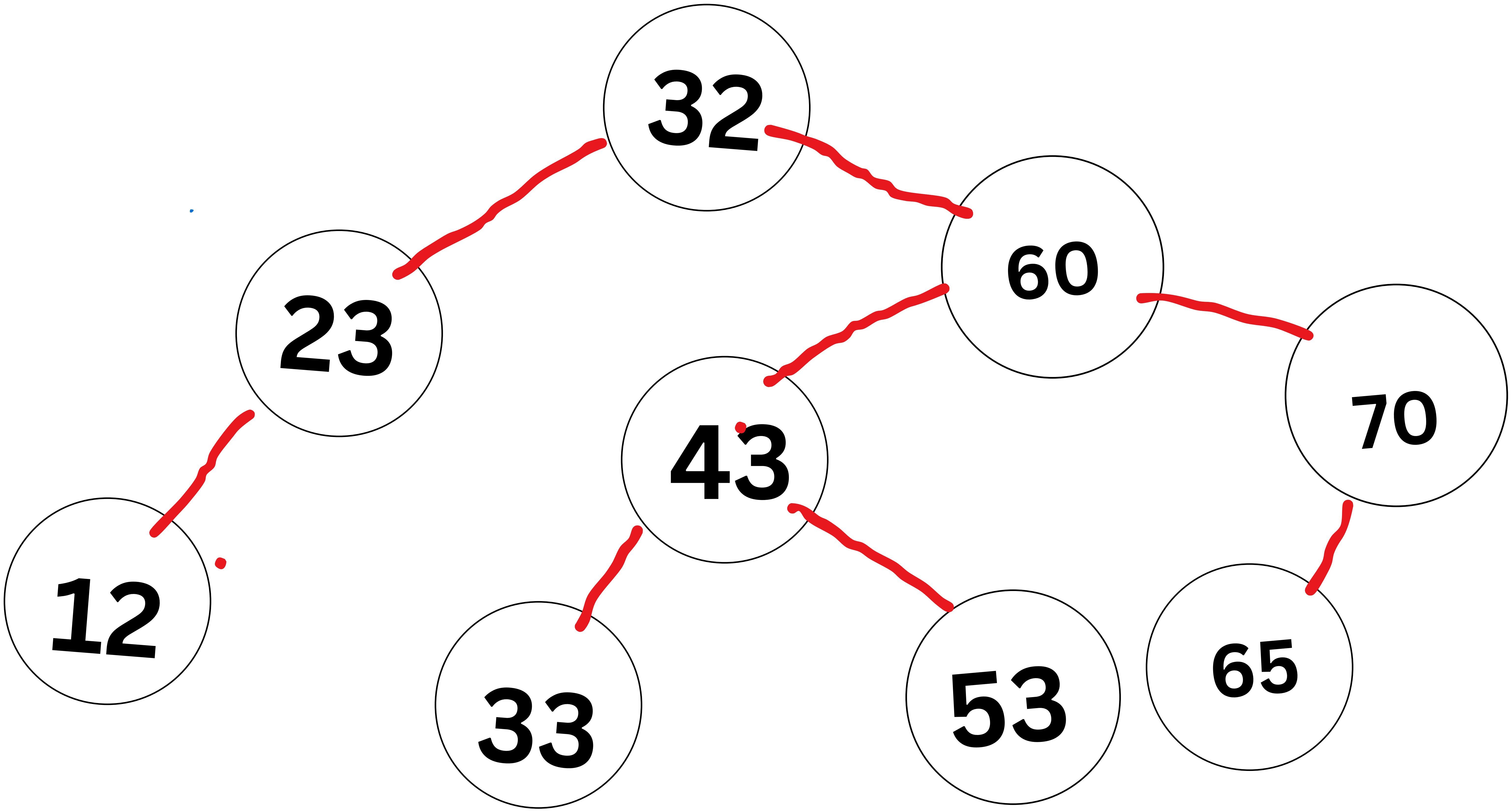
balance = 1 - 3 = -2

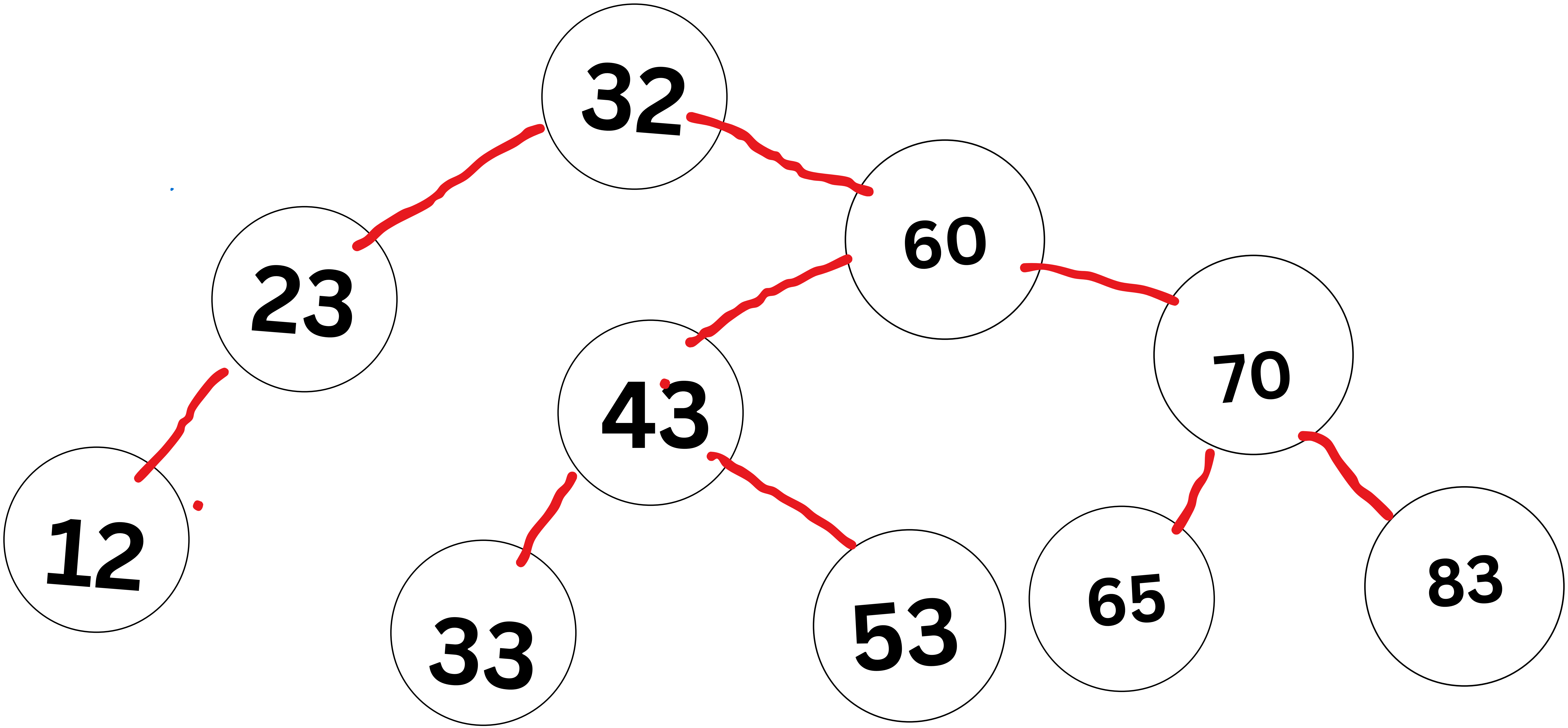


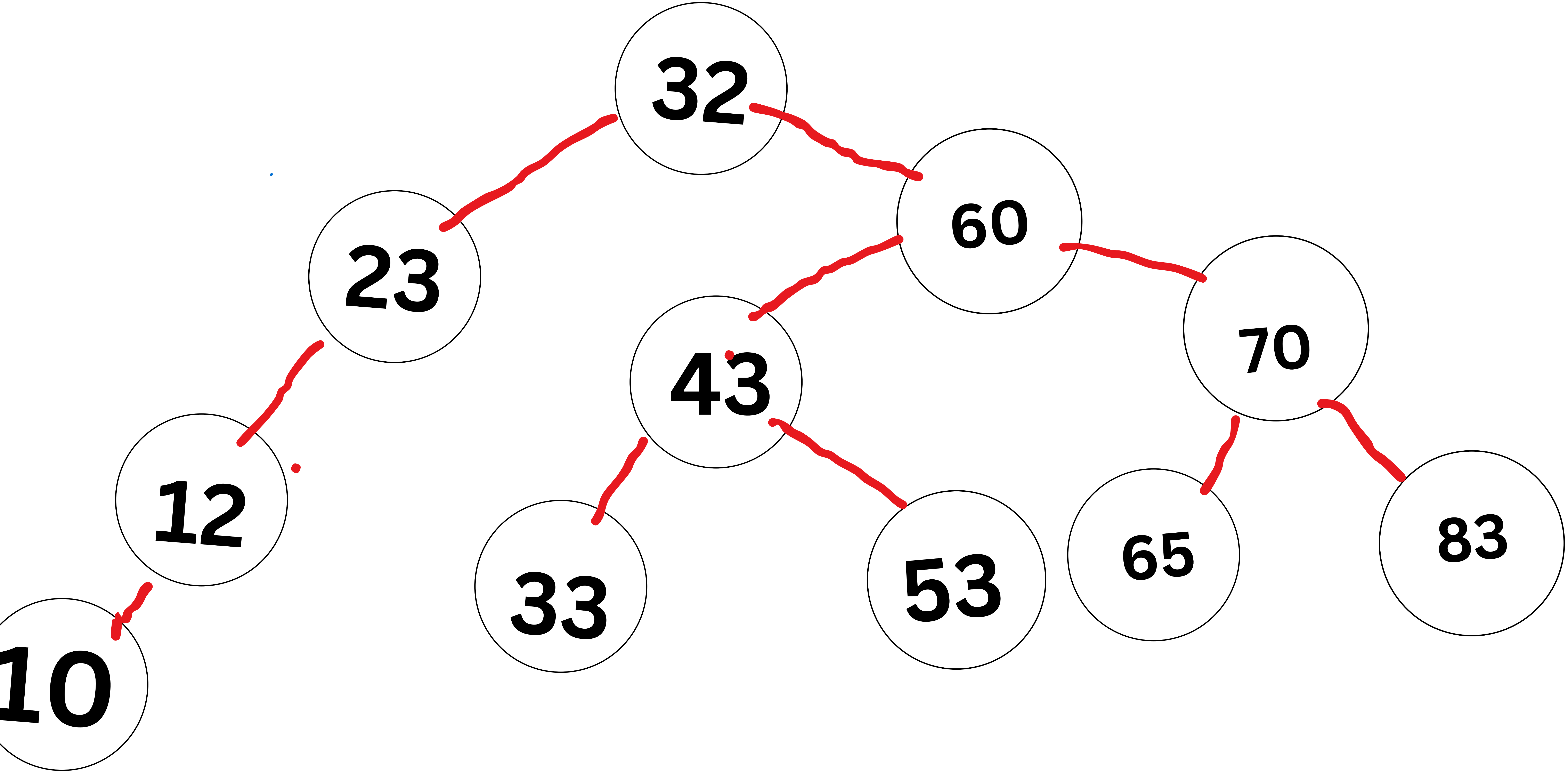
balance = 1 - 3 = -2

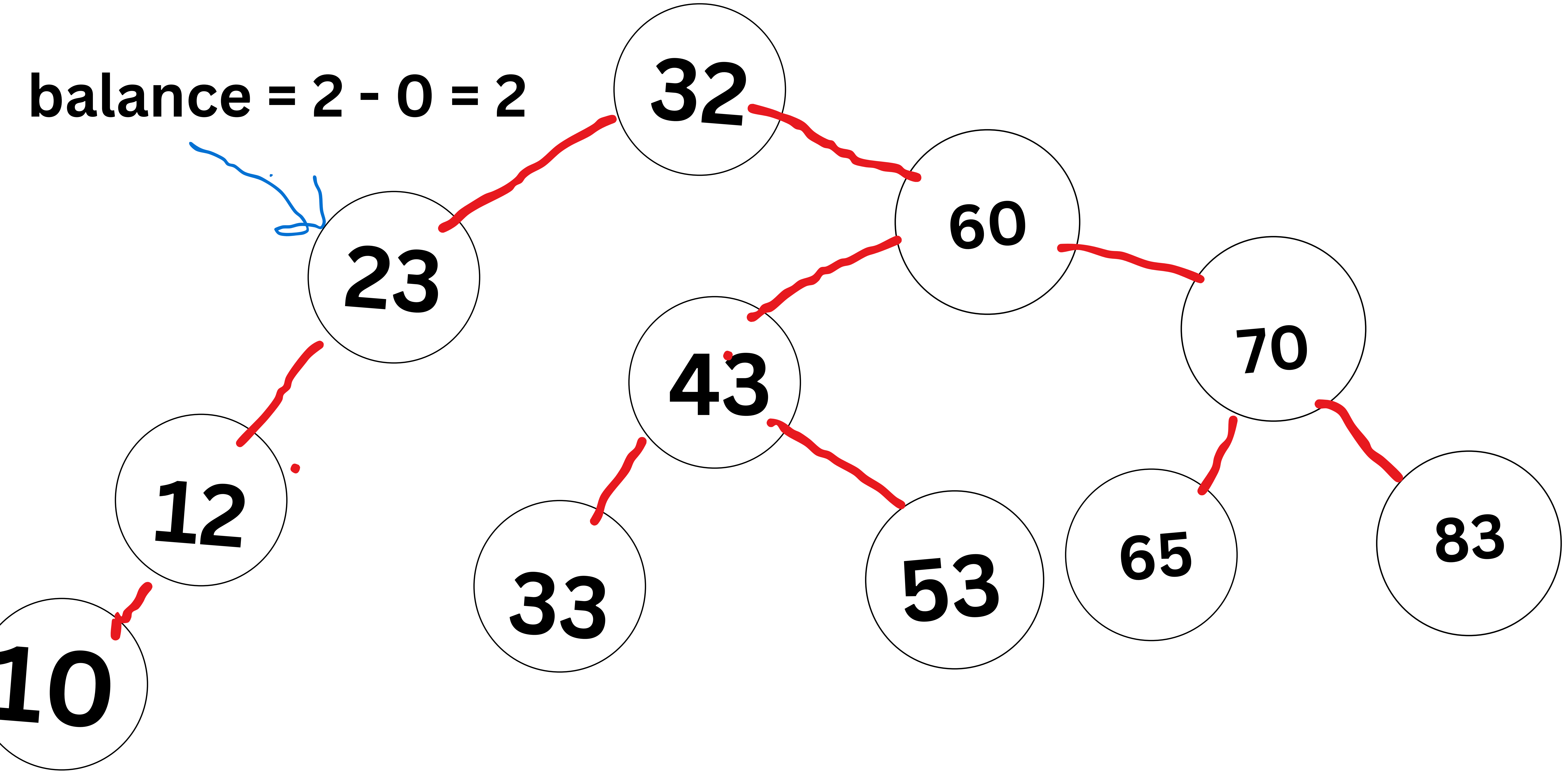


perform the left on the 43


32
23
60
12
43
70
33
53
65

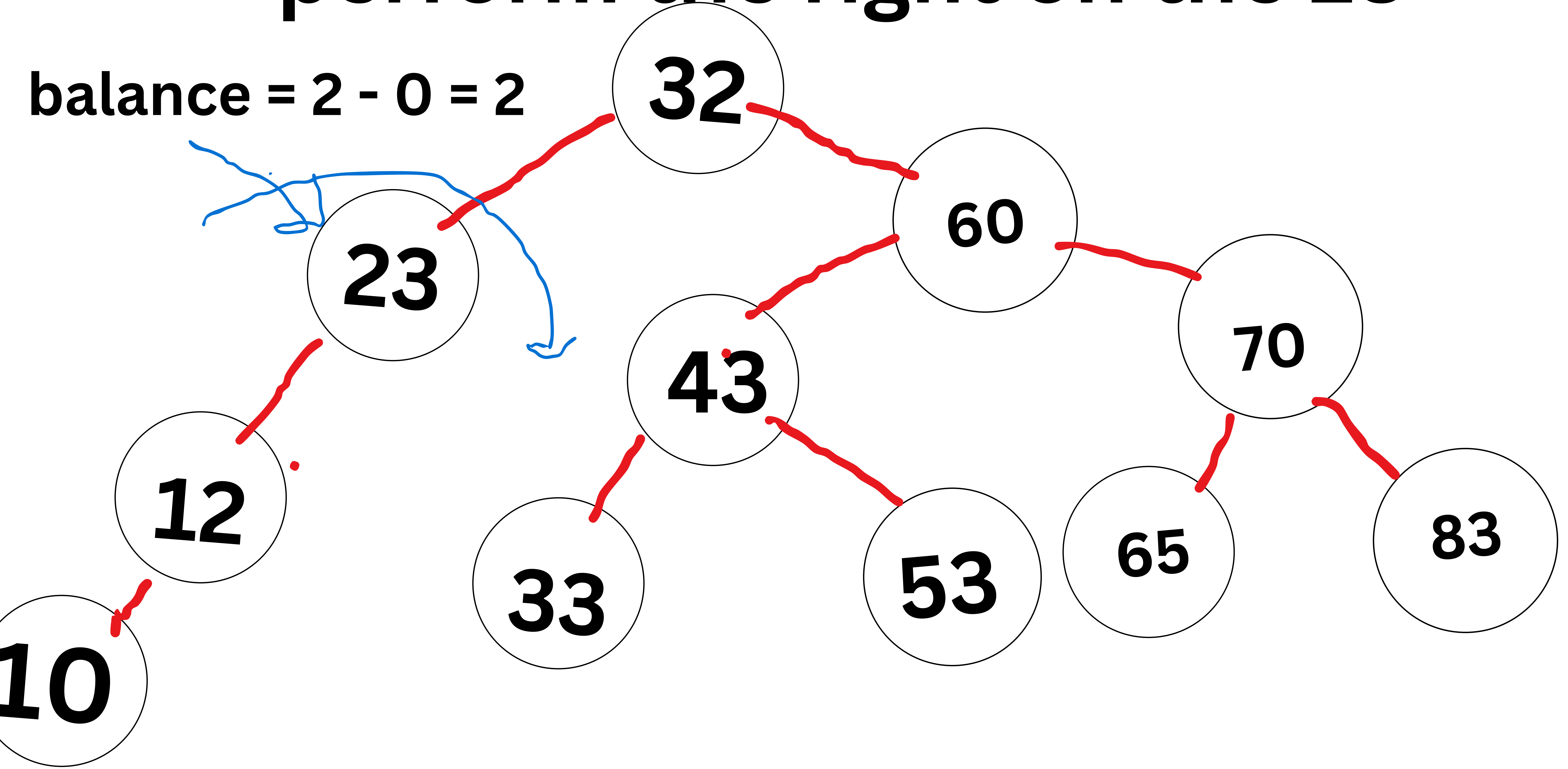

32
23
60
12
43
70
33
53
65
83

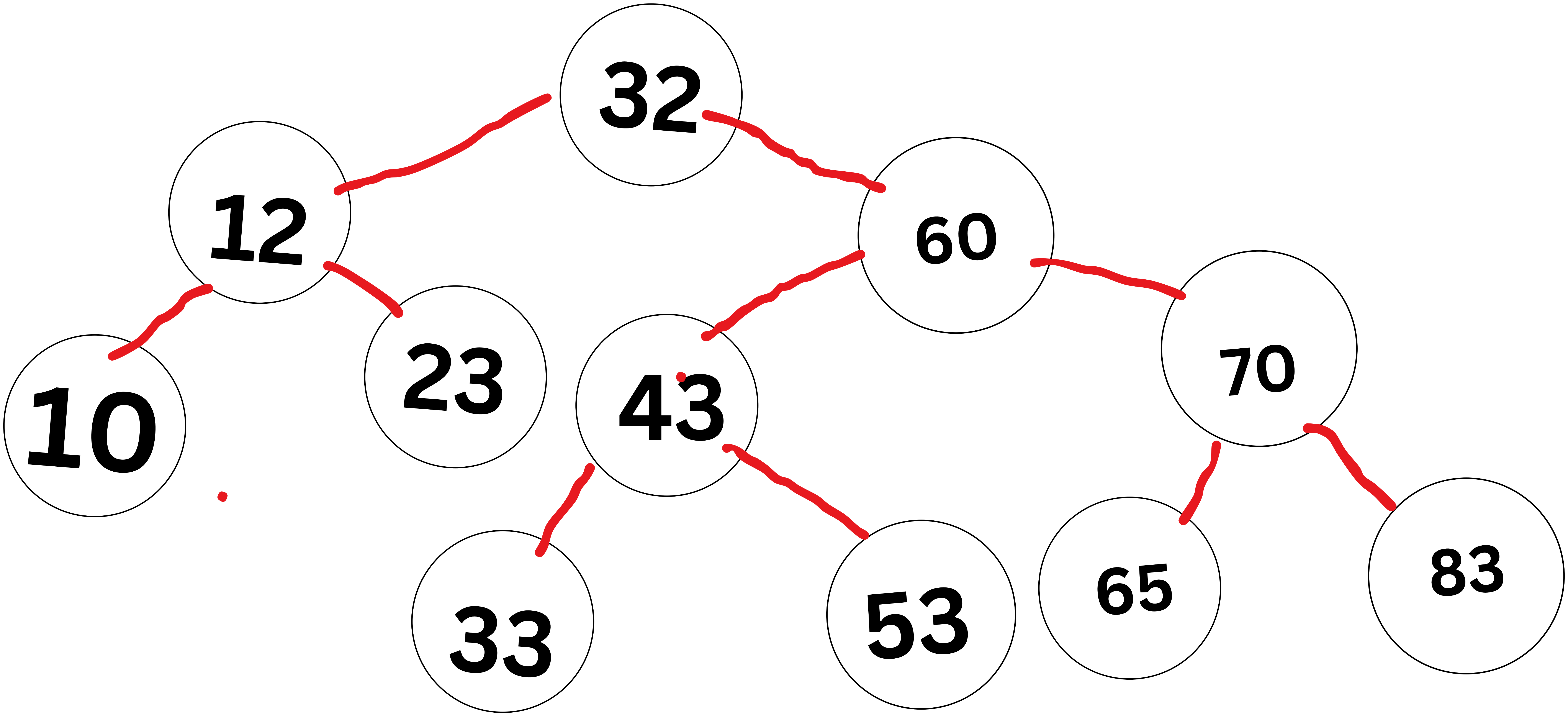

32
23
60
43
70
12
33
53
65
83
10

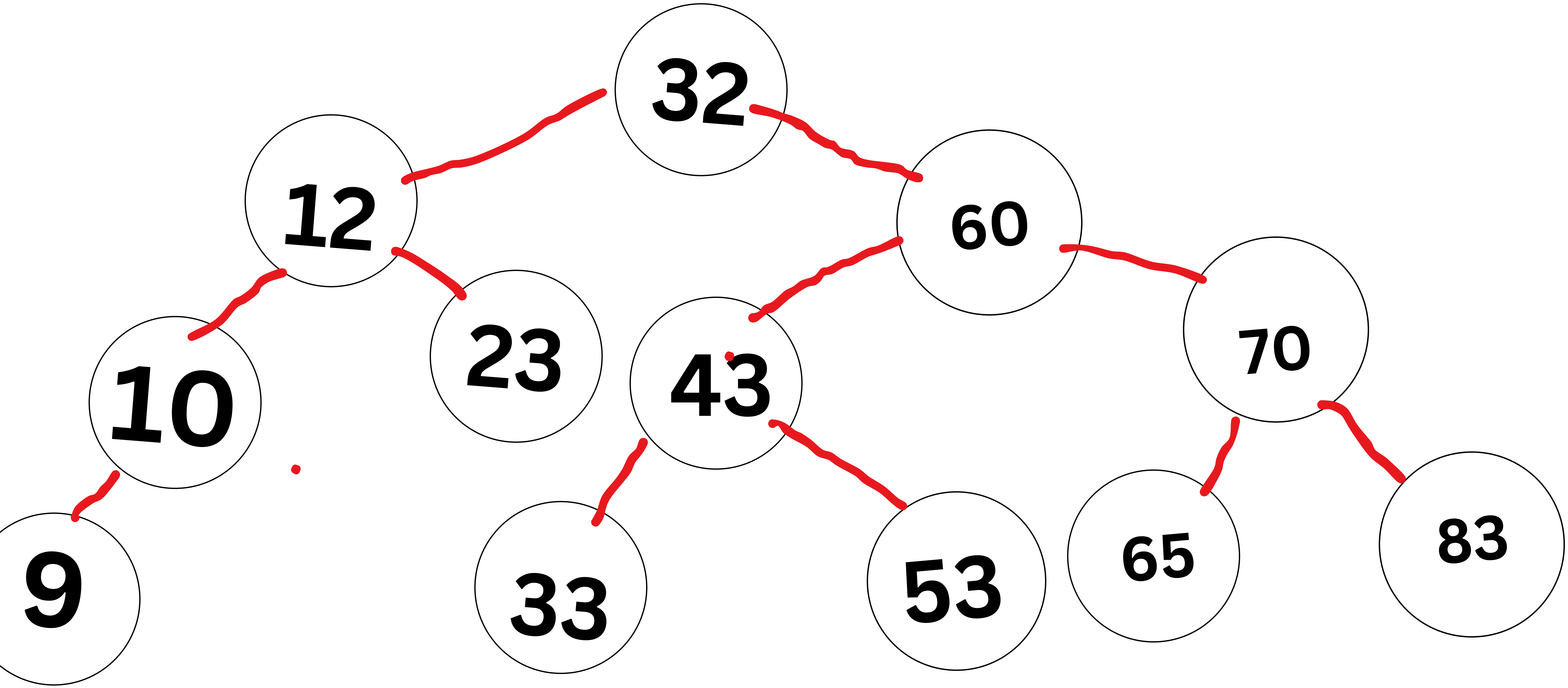

balance = 2 - 0 = 2
32
23
60
12
43
70
10
33
53
65
83

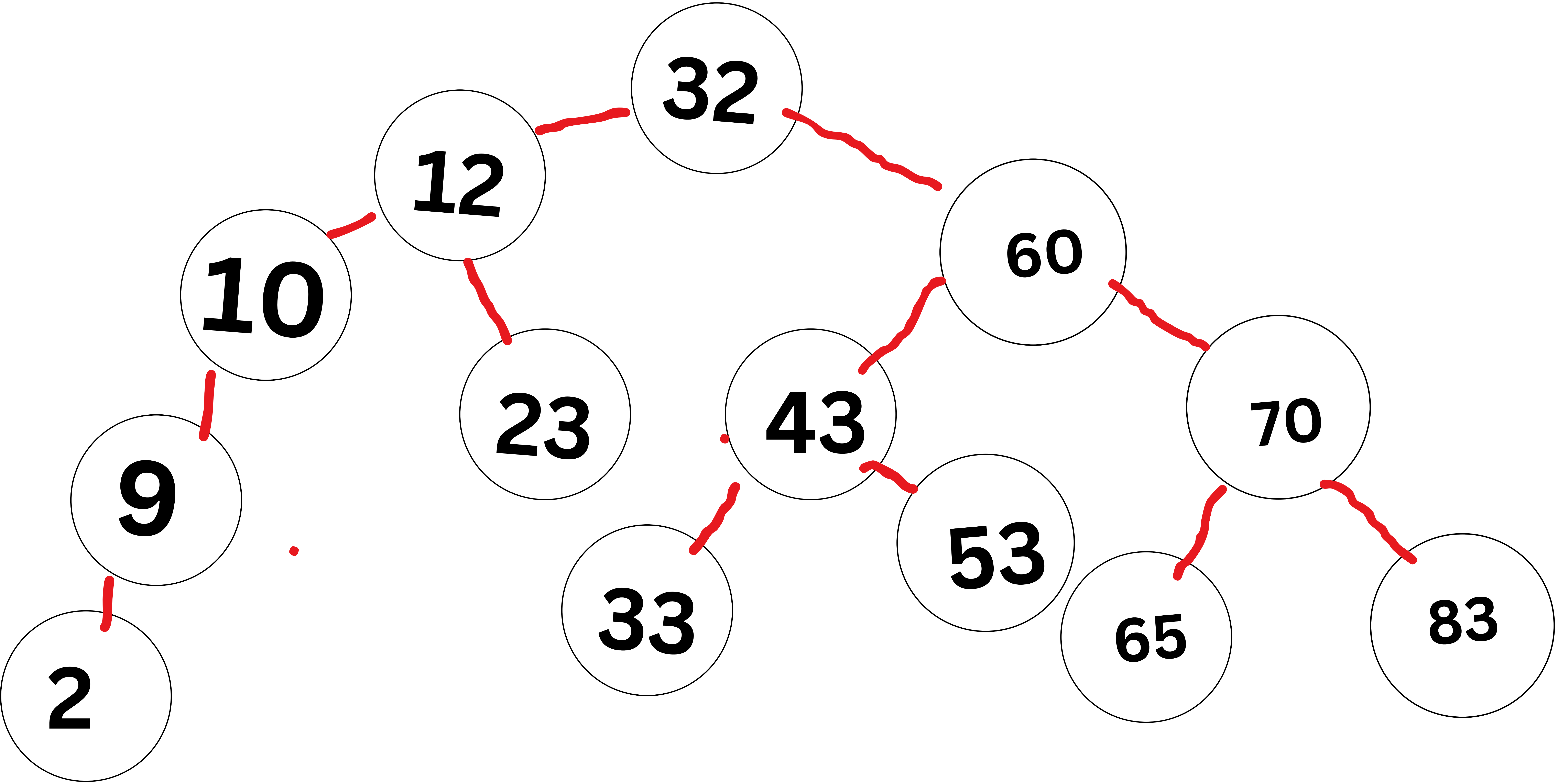
perform the right on the 23

balance = 2 - 0 = 2




32
12
60
10
23
43
70
33
53
65
83

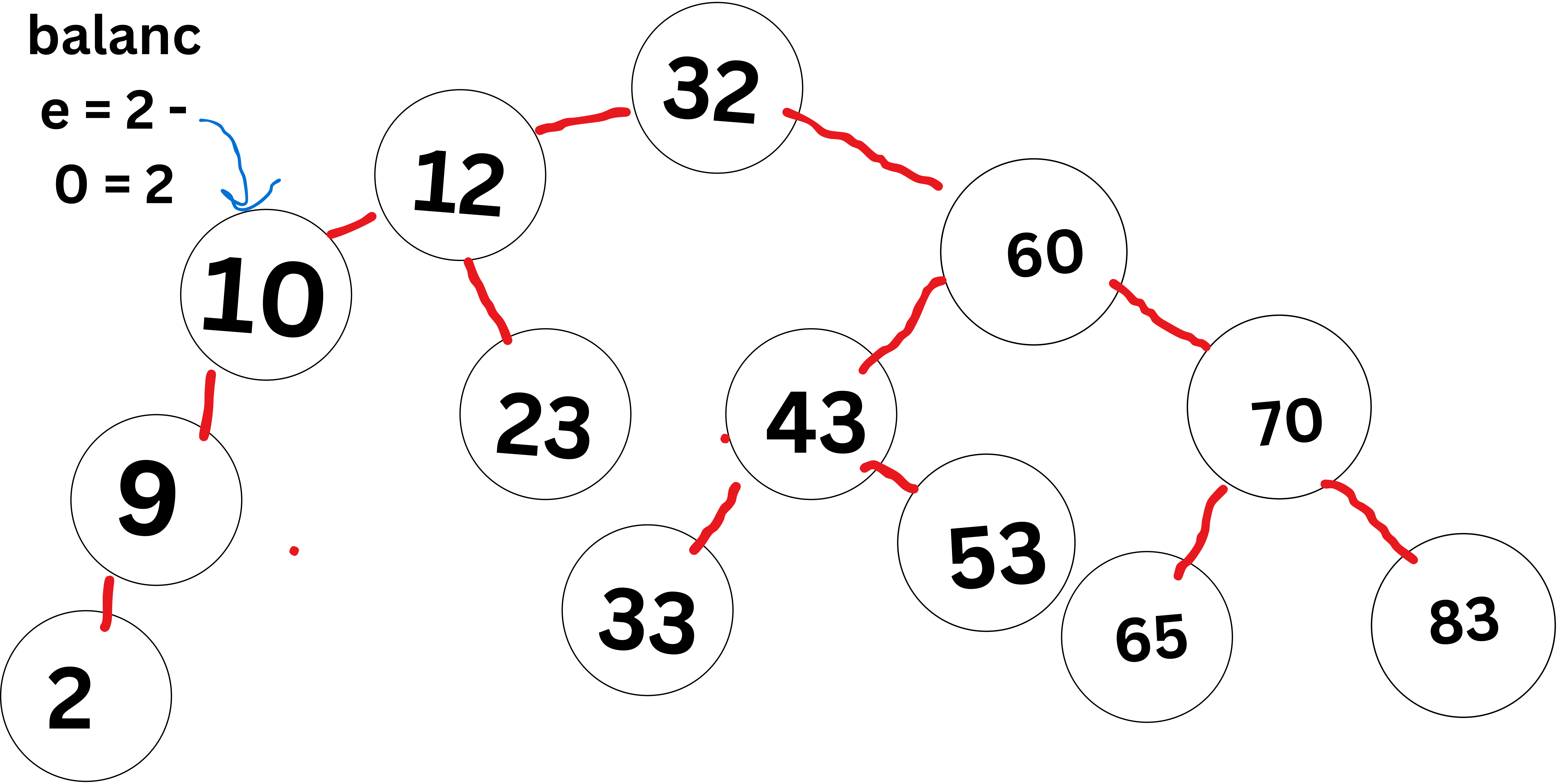

32
12
60
10
23
43
70
9
33
53
65
83


32
12
60
10
23
43
70
9
33
53
65
83
2

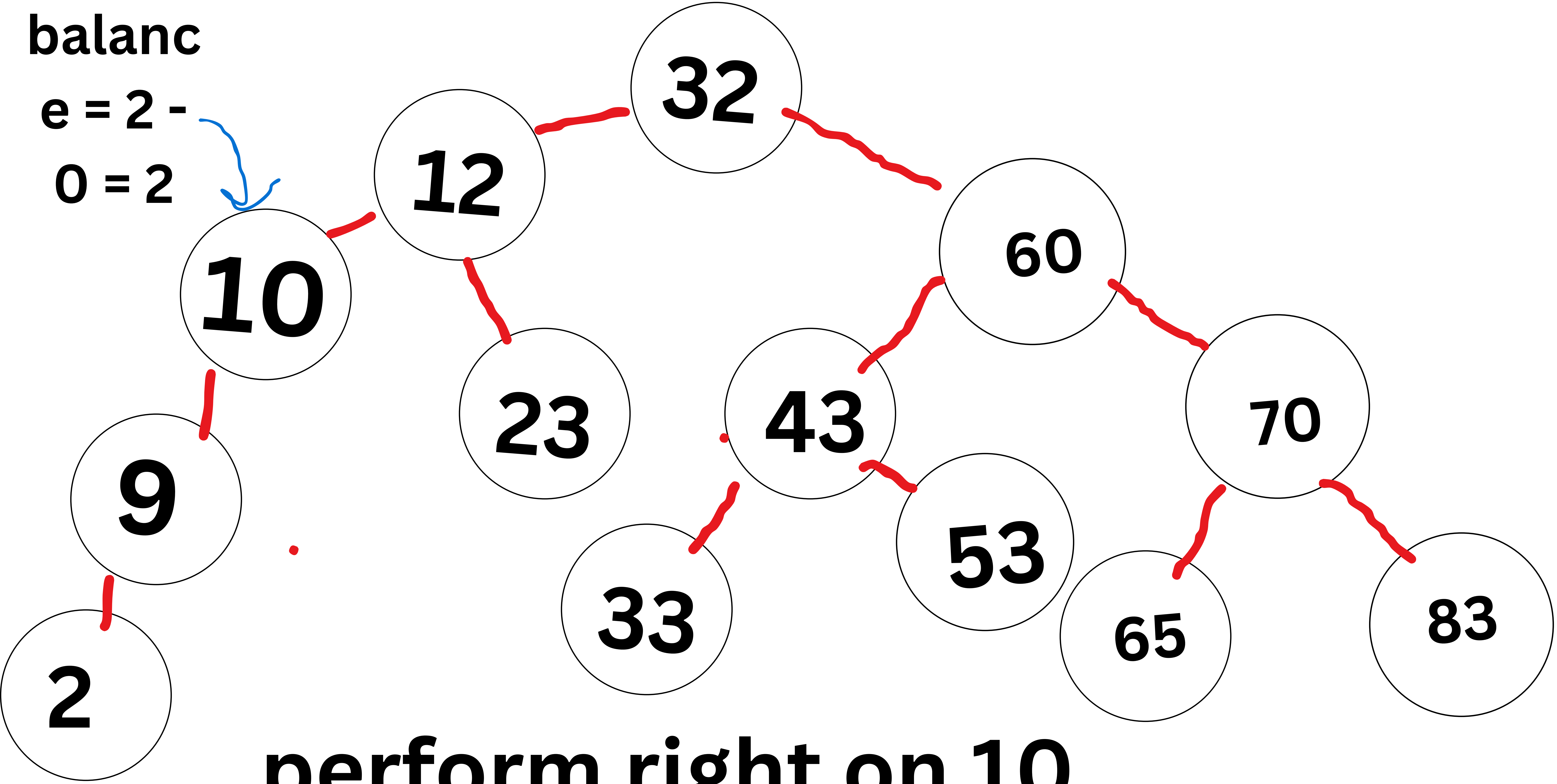
balanc

e = 2 -

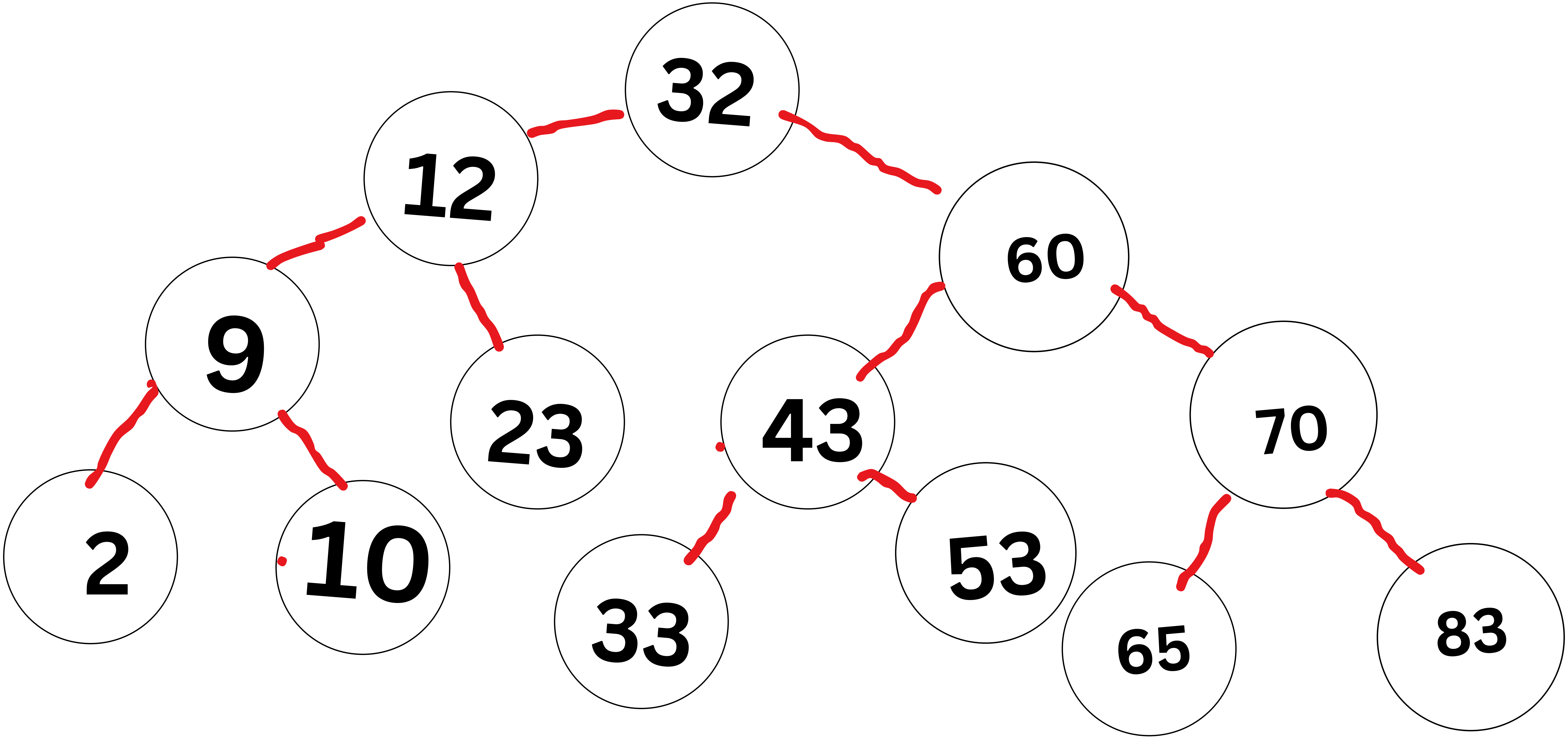
0 = 2



balanc
e = 2 -
0 = 2



perform right on 10


32
12
60
9
23
43
70
2
10
33
53
65
83

Now our tree is a fully avl tree

deletion operation is same as the binary search tree. Just e have to perform this balancing operation fter perform the deletion operation.

Let’s Go to the implementation.