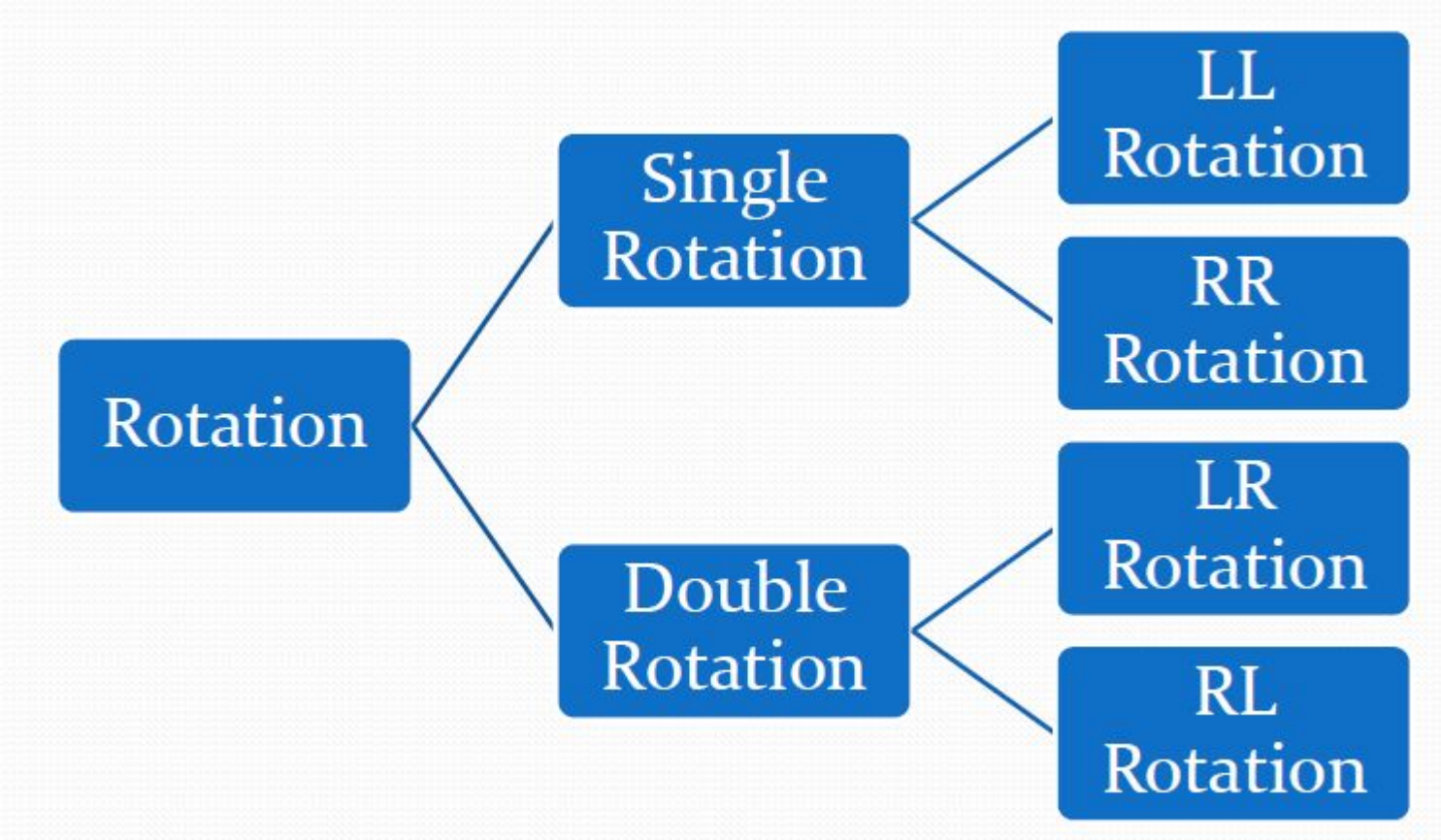
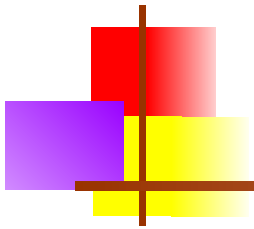
AVL-Tree

Types of Rotation

Rotation- To switch children and parents among two or three adjacent nodes to restore balance of a tree.





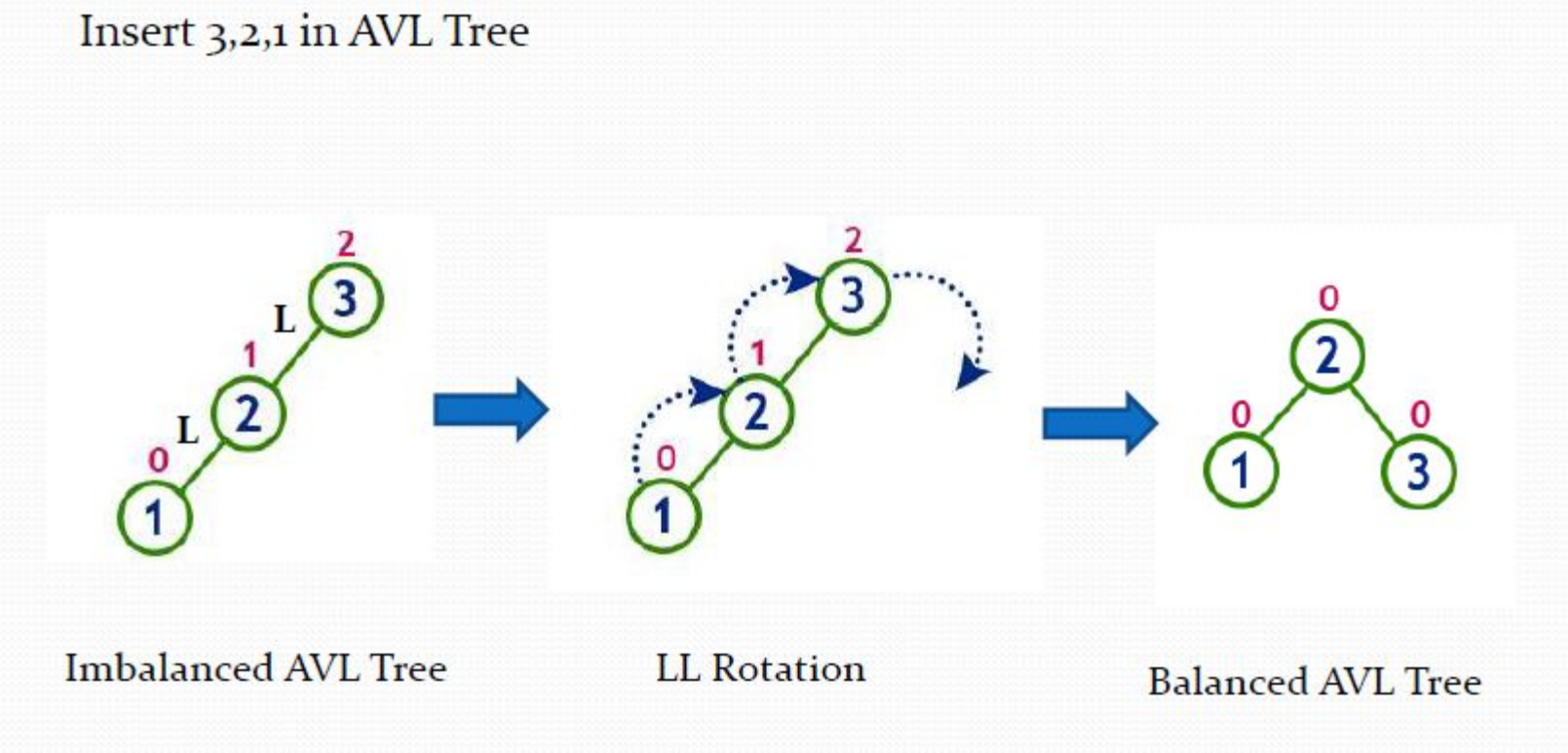
Types of Rotation

- Single Rotation is applied when imbalanced node and child has same sign of BF (in the direction of new inserted node).
  - LL Rotation is applied in case of +ve sign. It mean left

    tree is heavy and so LL rotation is done.

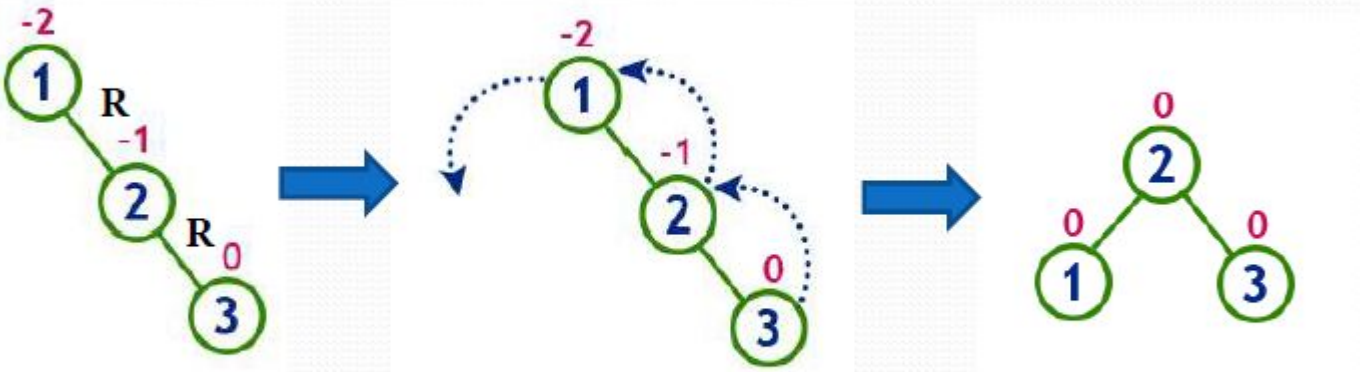
    RR Rotation is applied in case of -ve sign. It mean right tree is heavy and so RR rotation is done.
- Double Rotation is applied when imbalanced node and child has different signs of BF (in the direction of new inserted node).

LL Rotation


Insert 3,2,1 in AVL Tree
2
3
L
1
2
L
0
1
Imbalanced AVL Tree
2
3
1
2
0
1
LL Rotation
0
2
0
0
1
3
Balanced AVL Tree

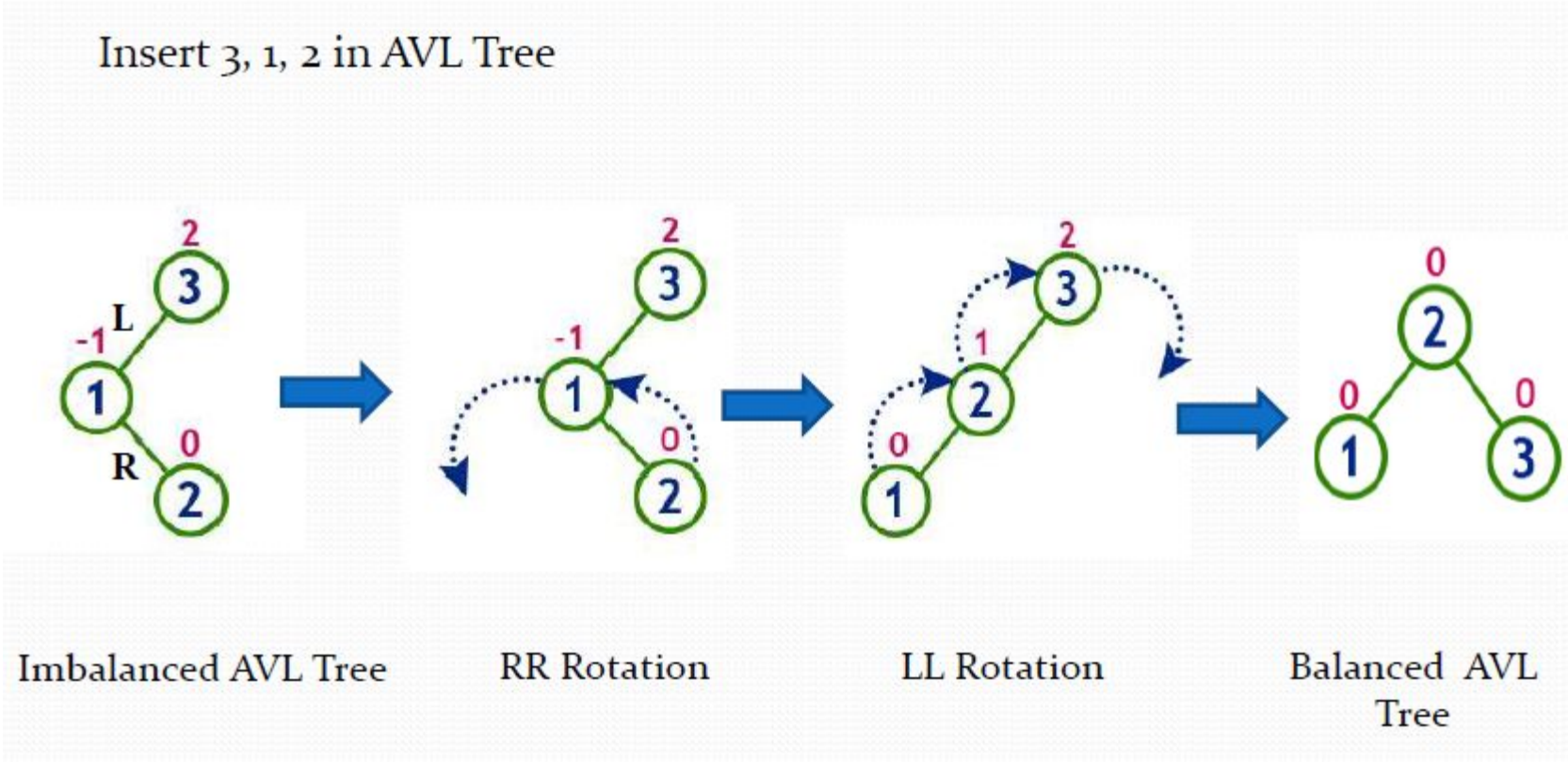
RR Rotation

Insert 1,2,3 in AVL Tree

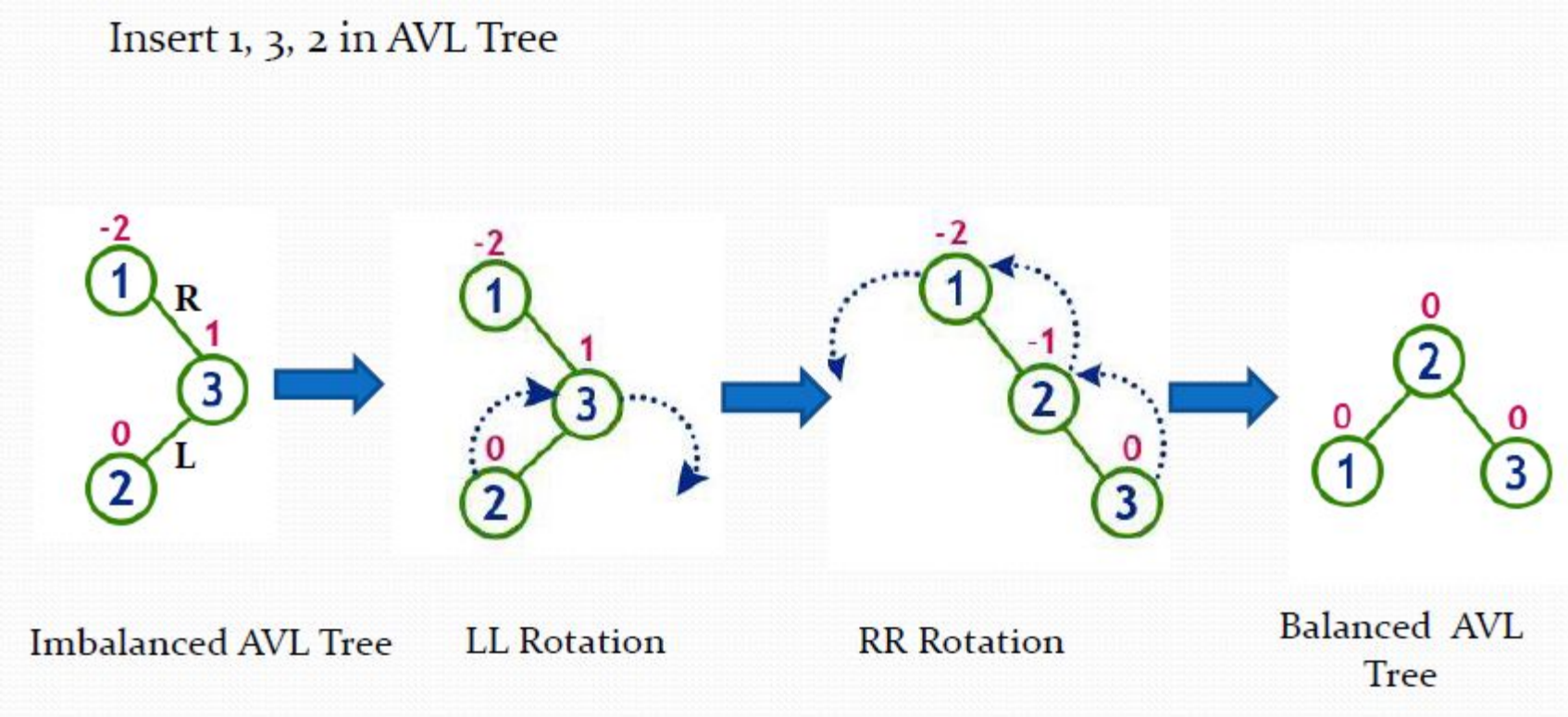


Imbalanced AVL Tree RR Rotation Balanced AVL Tree

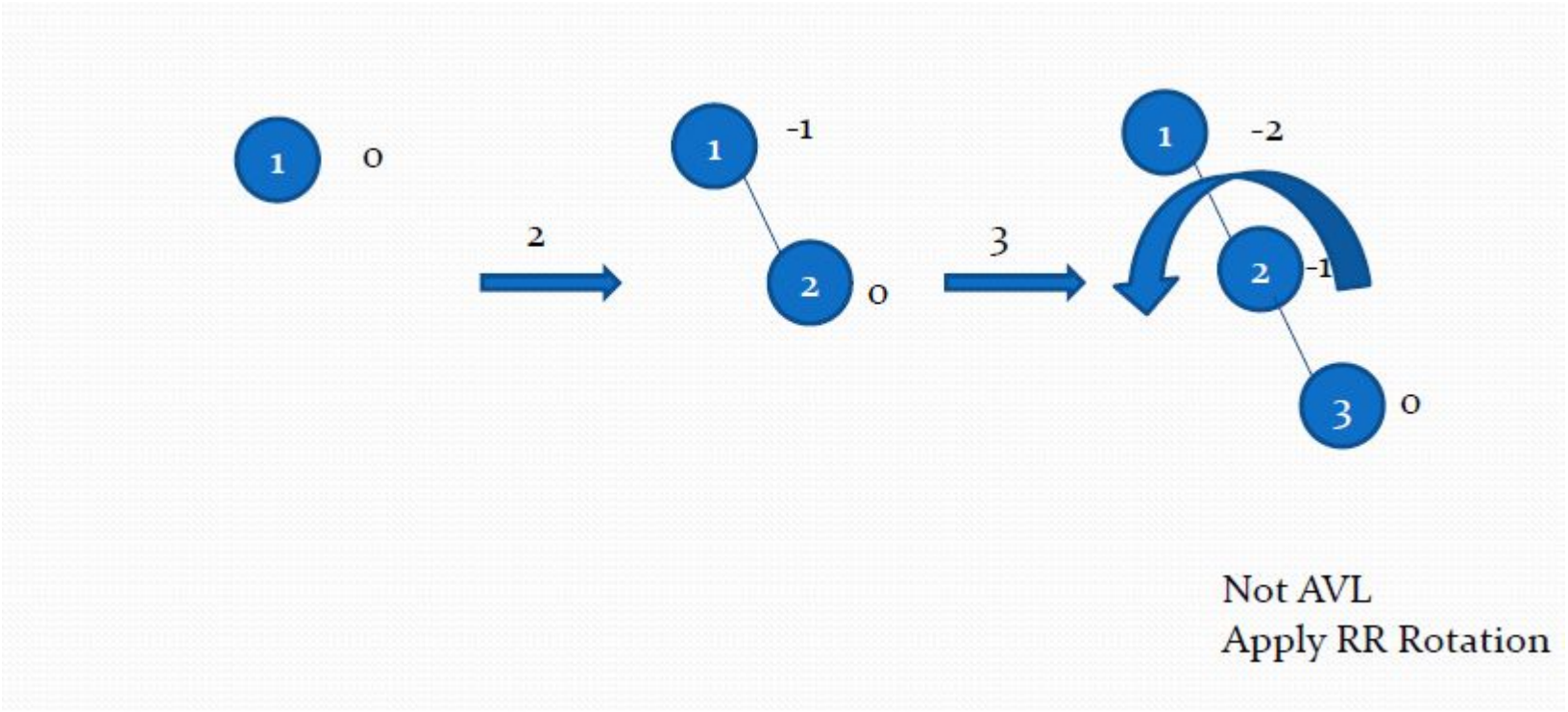
LR Rotation


Insert 3, 1, 2 in AVL Tree
2
3
-1
L
1
0
R
2
Imbalanced AVL Tree
2
3
-1
1
0
2
RR Rotation
2
3
1
2
0
1
LL Rotation
0
2
0
0
1
3
Balanced AVL Tree

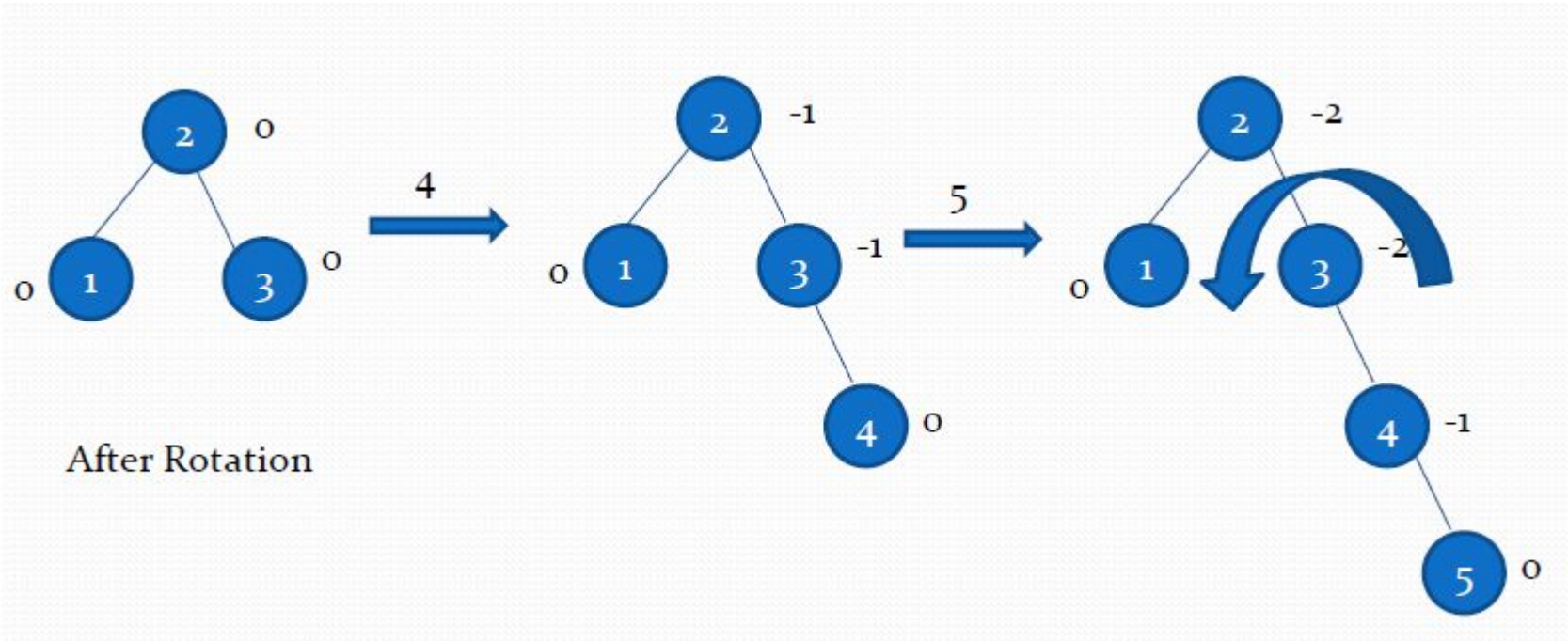
RL Rotation


Insert 1, 3, 2 in AVL Tree
-2
1
R
1
3
0
2
L
-2
1
1
3
0
2
-2
1
-1
2
0
3
0
2
0
0
1
3
Imbalanced AVL Tree
LL Rotation
RR Rotation
Balanced AVL Tree

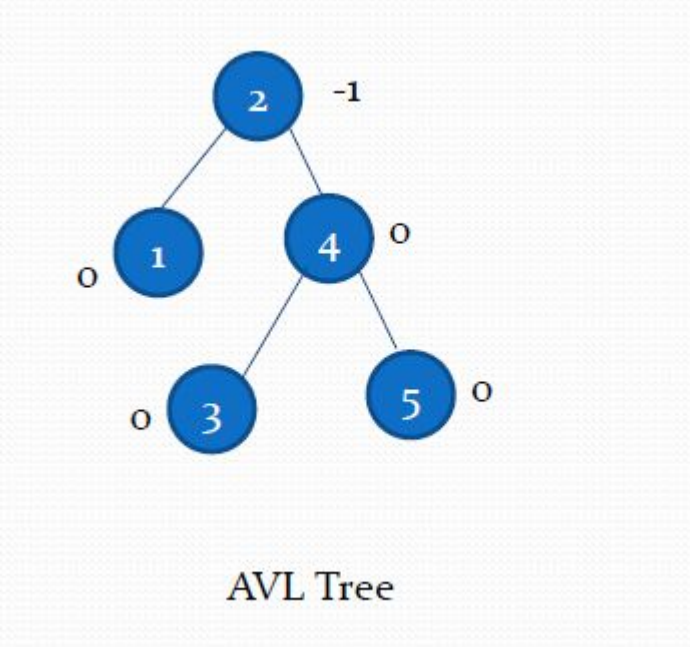
Construct a AVL Tree by inserting from 1 to 5 numbers



Construct a AVL Tree by inserting from 1 to 5 numbers

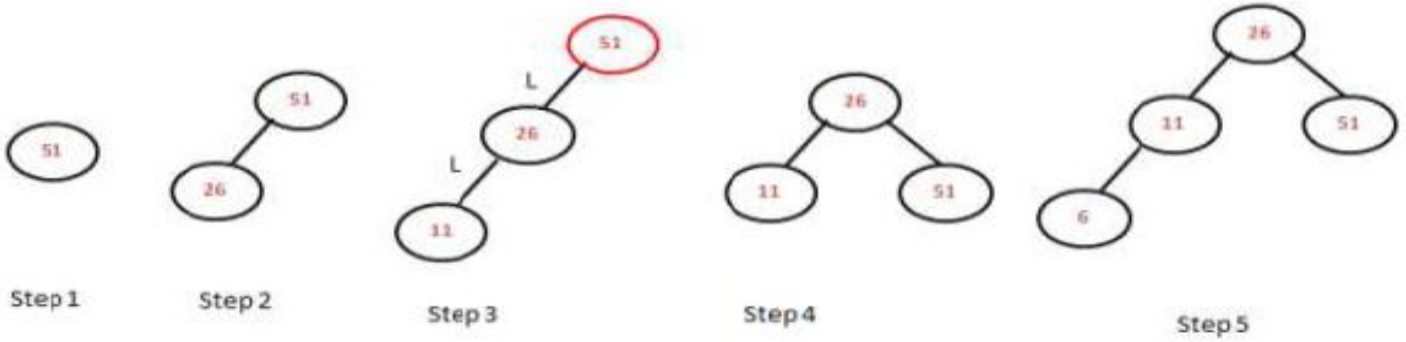


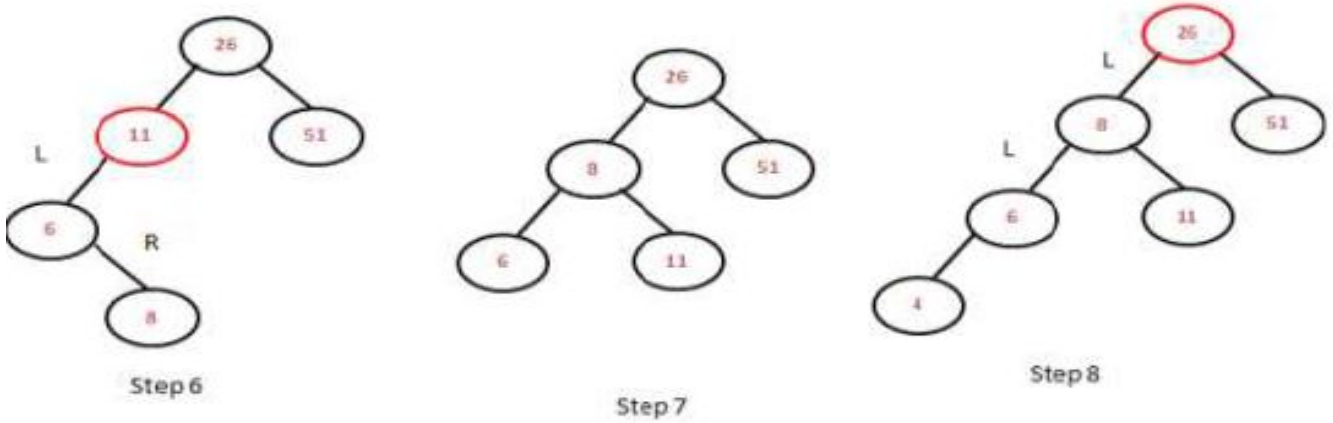
Construct a AVL Tree by inserting from 1 to 5 numbers

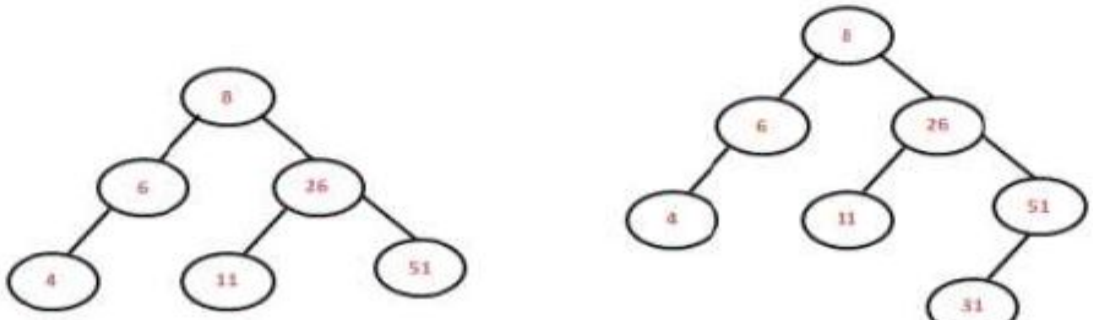


AVL Tree

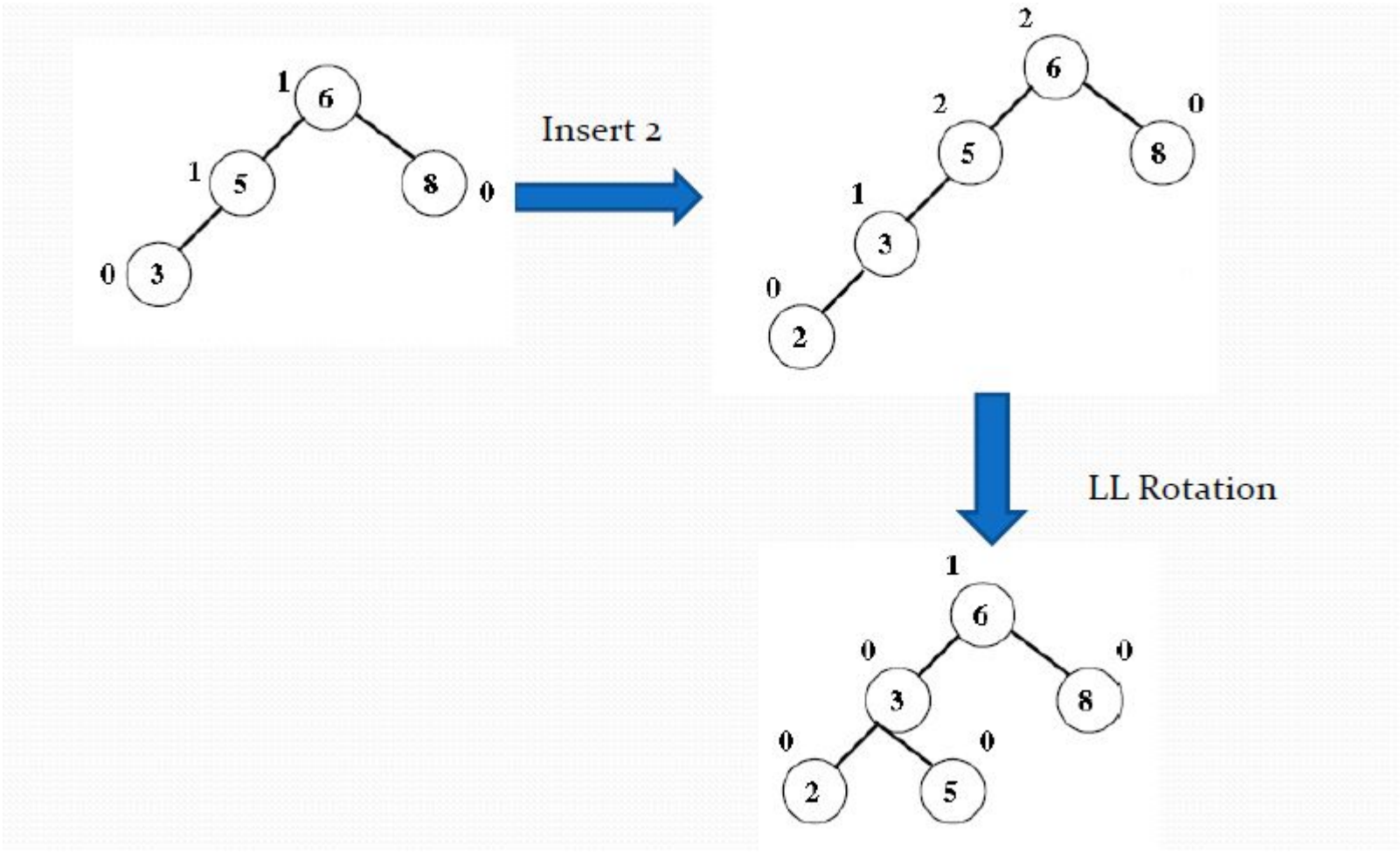
Construct AVL Tree with data items: 51, 26, 11, 6, 8, 4, 31



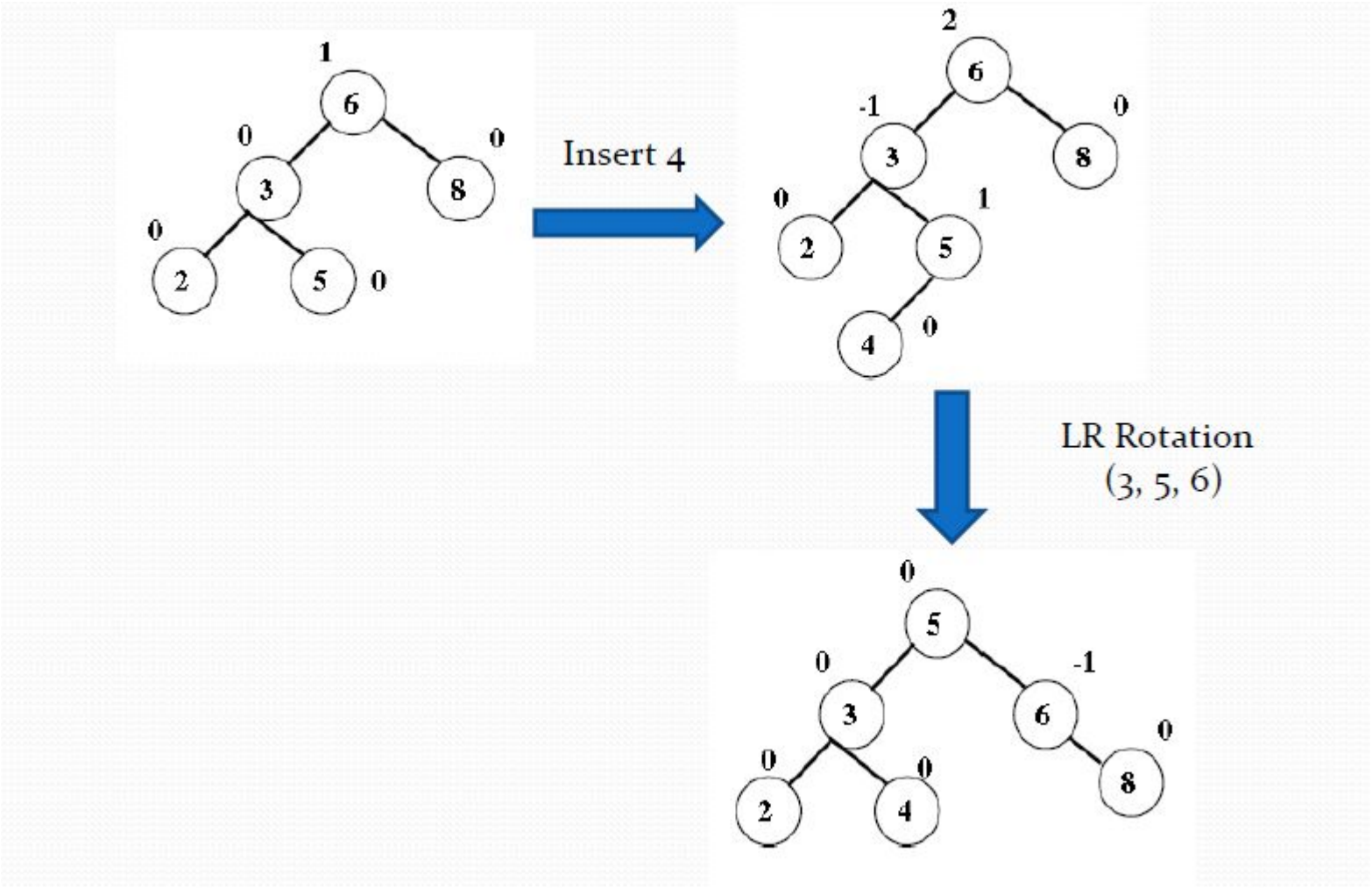




Insertion in AVL Tree

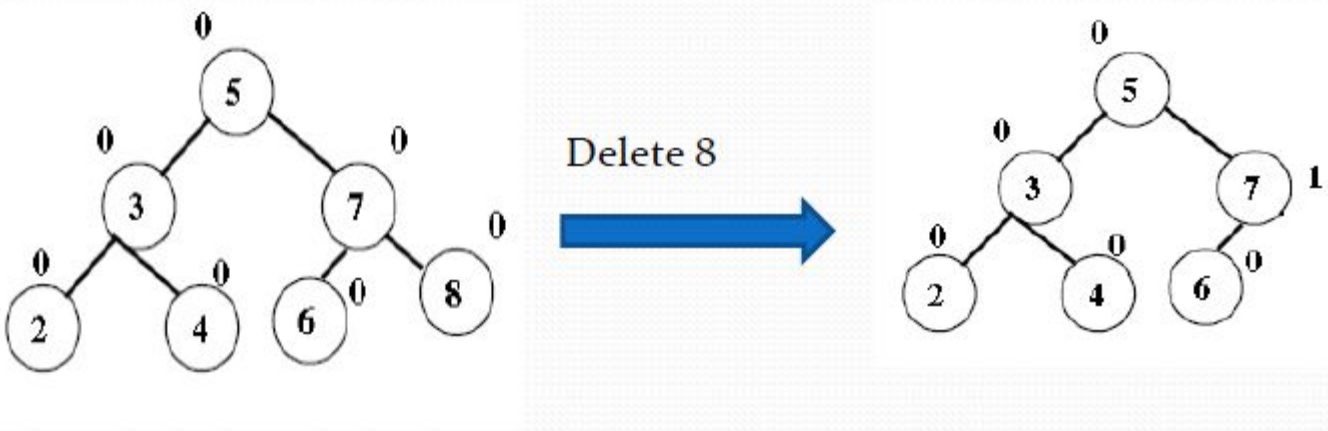


Insertion in AVL Tree



Deletion in AVL Tree

It is also possible to delete an item from AVL Tree.

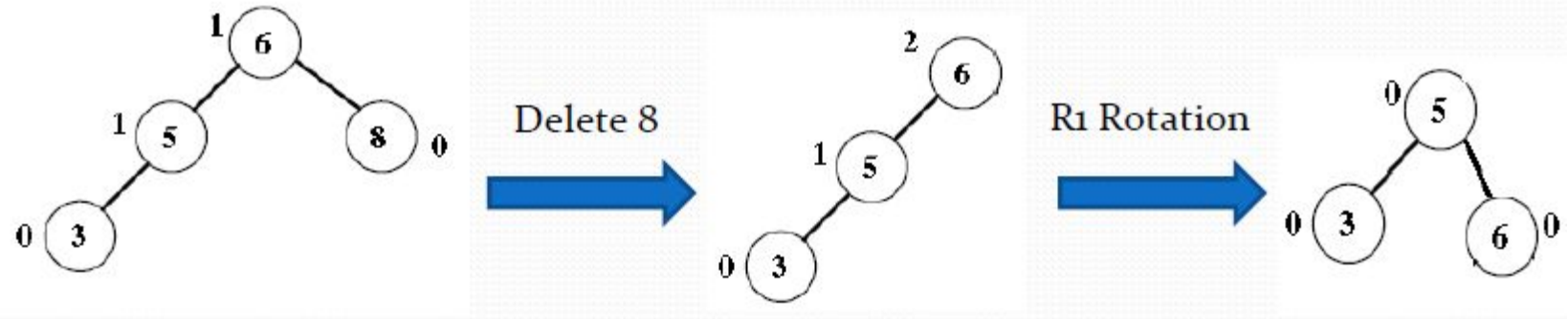


Balanced AVL

Balanced AVL

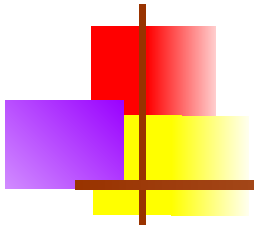
Deletion in AVL Tree

Just like insertion, deletion can cause an imbalance, which will need to be fixed by applying one of the four rotations.



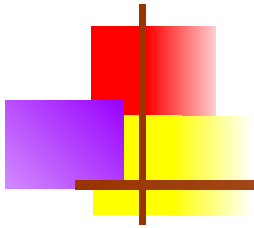
Balanced AVL

Imbalanced AVL



Deletion in AVL Tree

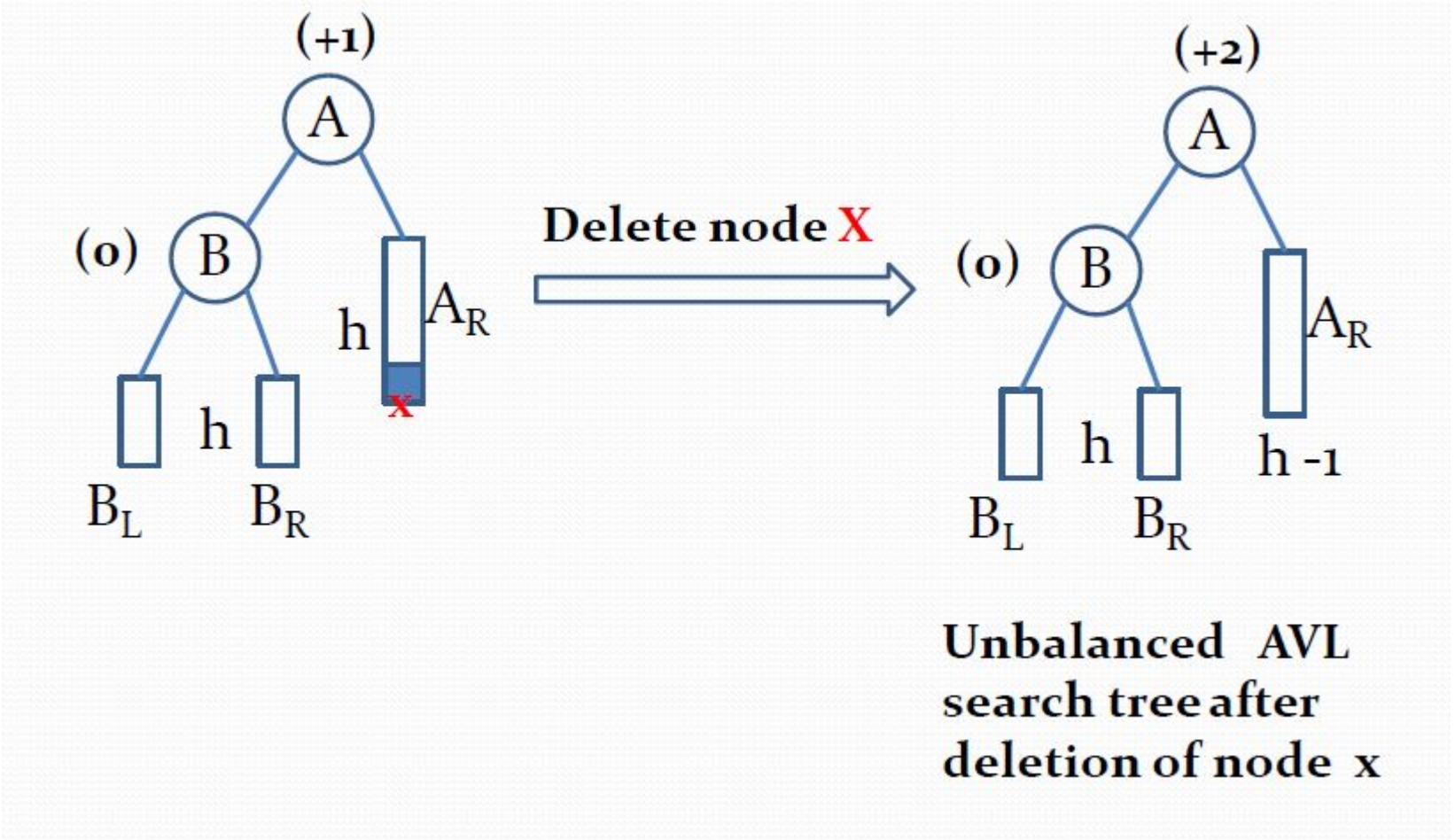
- The deletion is also the same as in BST. However, in imbalanced tree due to deletion, one or more rotations need to be applied to balance the AVL tree.
- The Right(R) imbalance is classified into R0, R1, R-1
- The Left(L) imbalance is classified into L0, L1, L-1



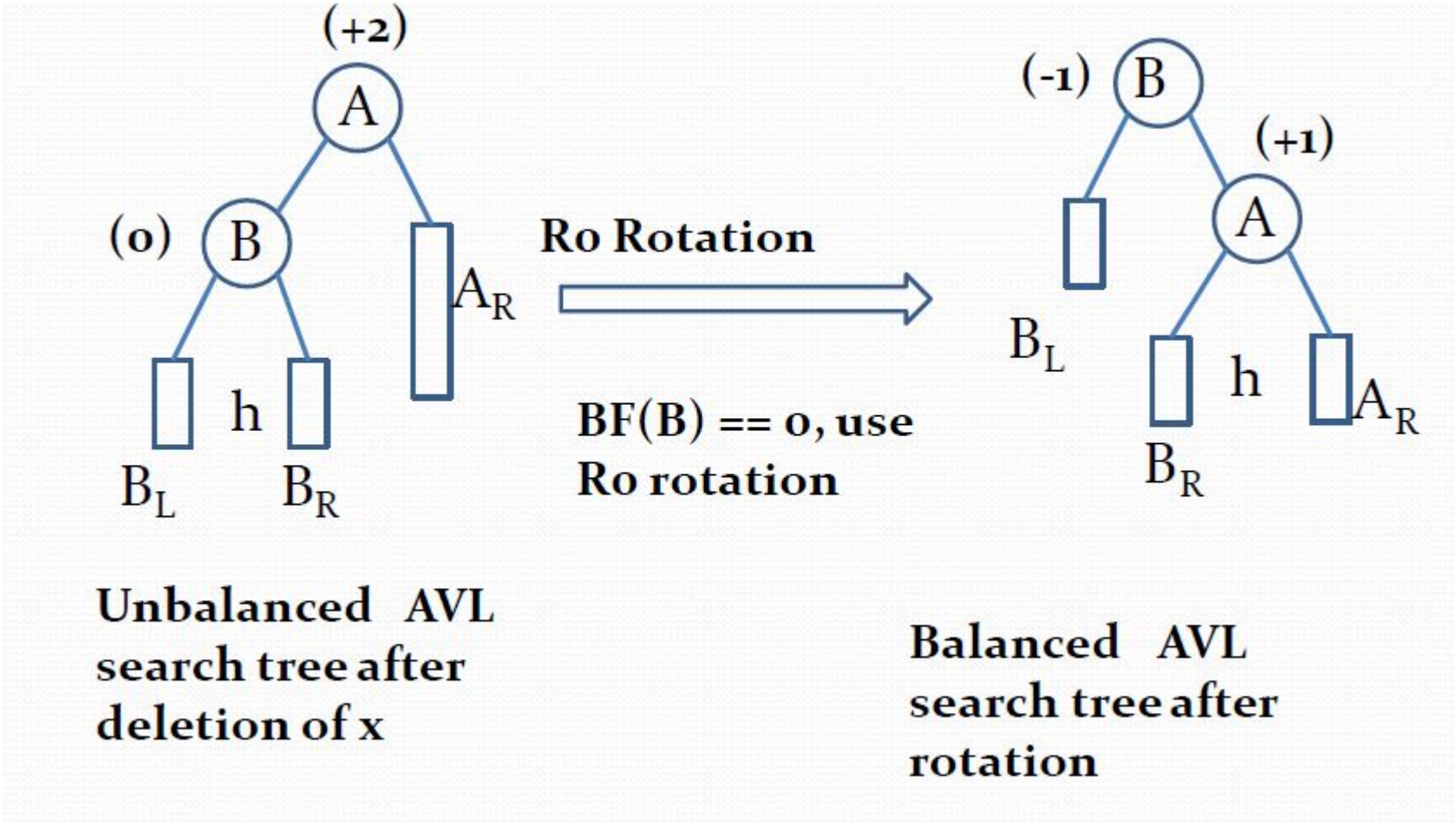
Deletion in AVL Tree

- LL Rotation is same to R0 and R1
- RR Rotation is same to L0 and L-1
- LR Rotation is same to R-1
- RL Rotation is same to L1

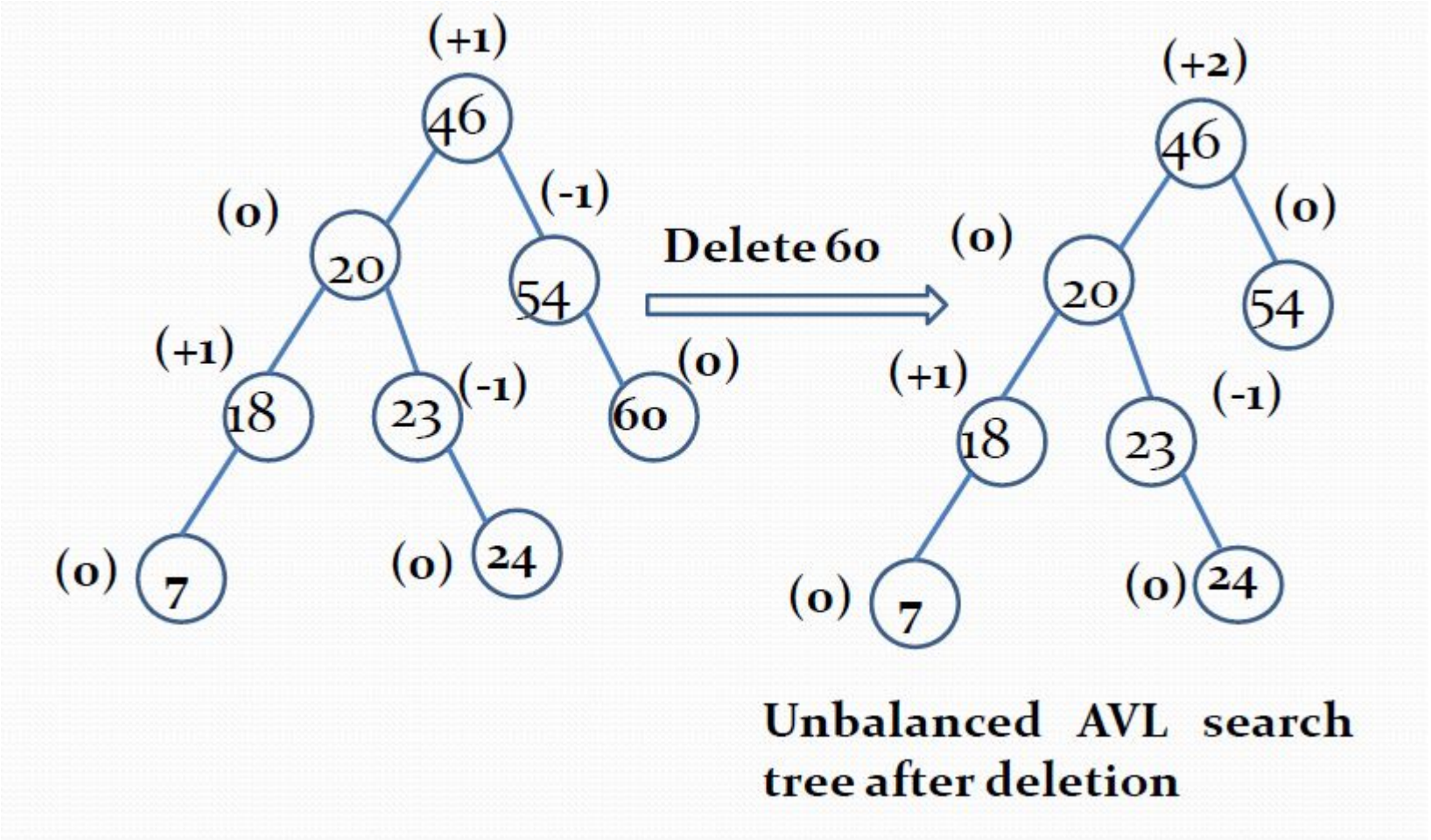
R0 Rotation


(+1)
A
(o)
B
h
A_R
x
h
B_L
B_R
Delete node X
(+2)
A
(o)
B
A_R
h
h -1
B_L
B_R
Unbalanced AVL search tree after deletion of node x

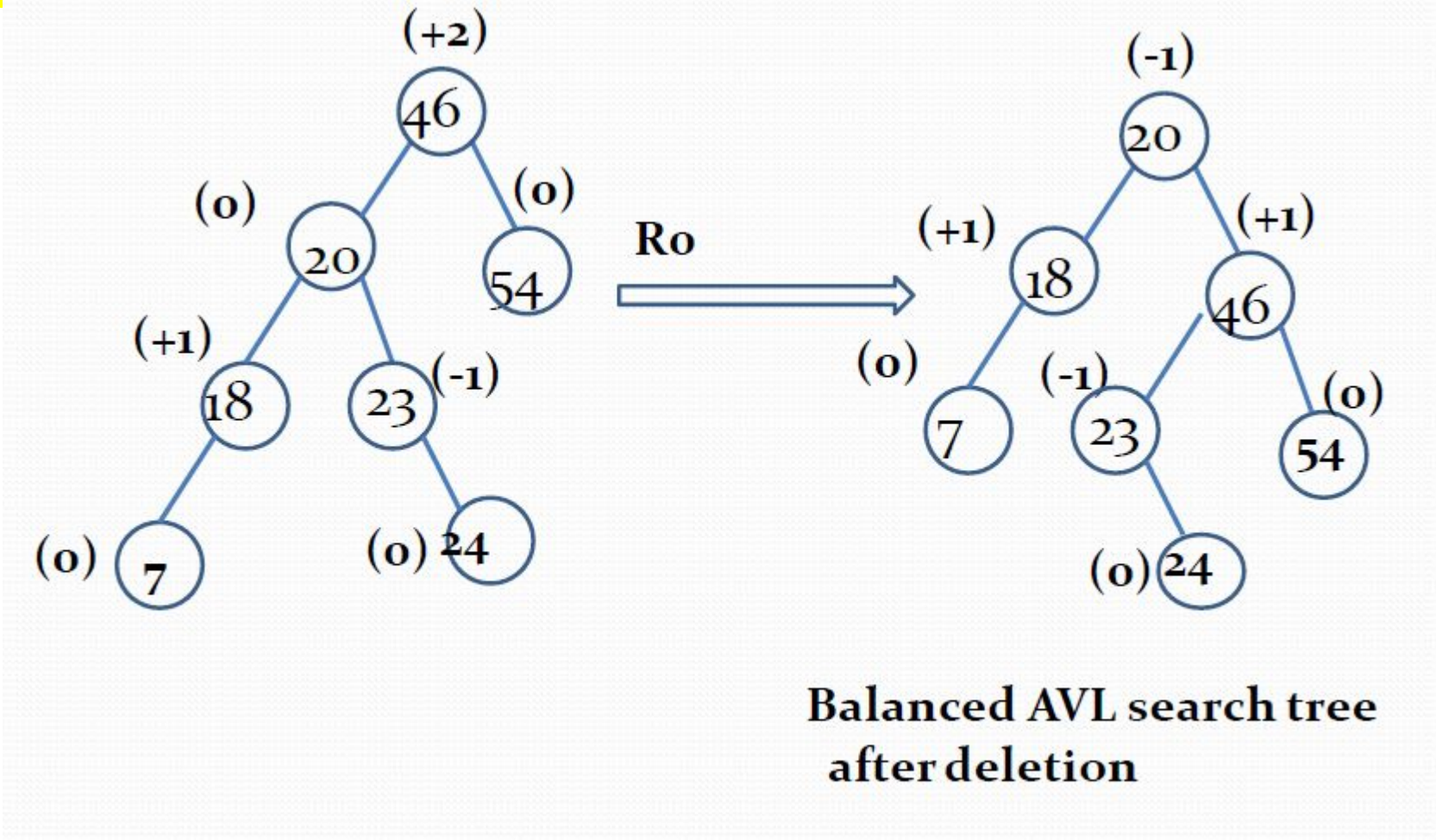
R0 Rotation


(+2)
A
(o)
B
A_R
h
B_L
B_R
Ro Rotation
BF(B) == o, use
Ro rotation
(-1)
B
(+1)
A
B_L
h
A_R
B_R
Unbalanced AVL
search tree after
deletion of x
Balanced AVL
search tree after
rotation

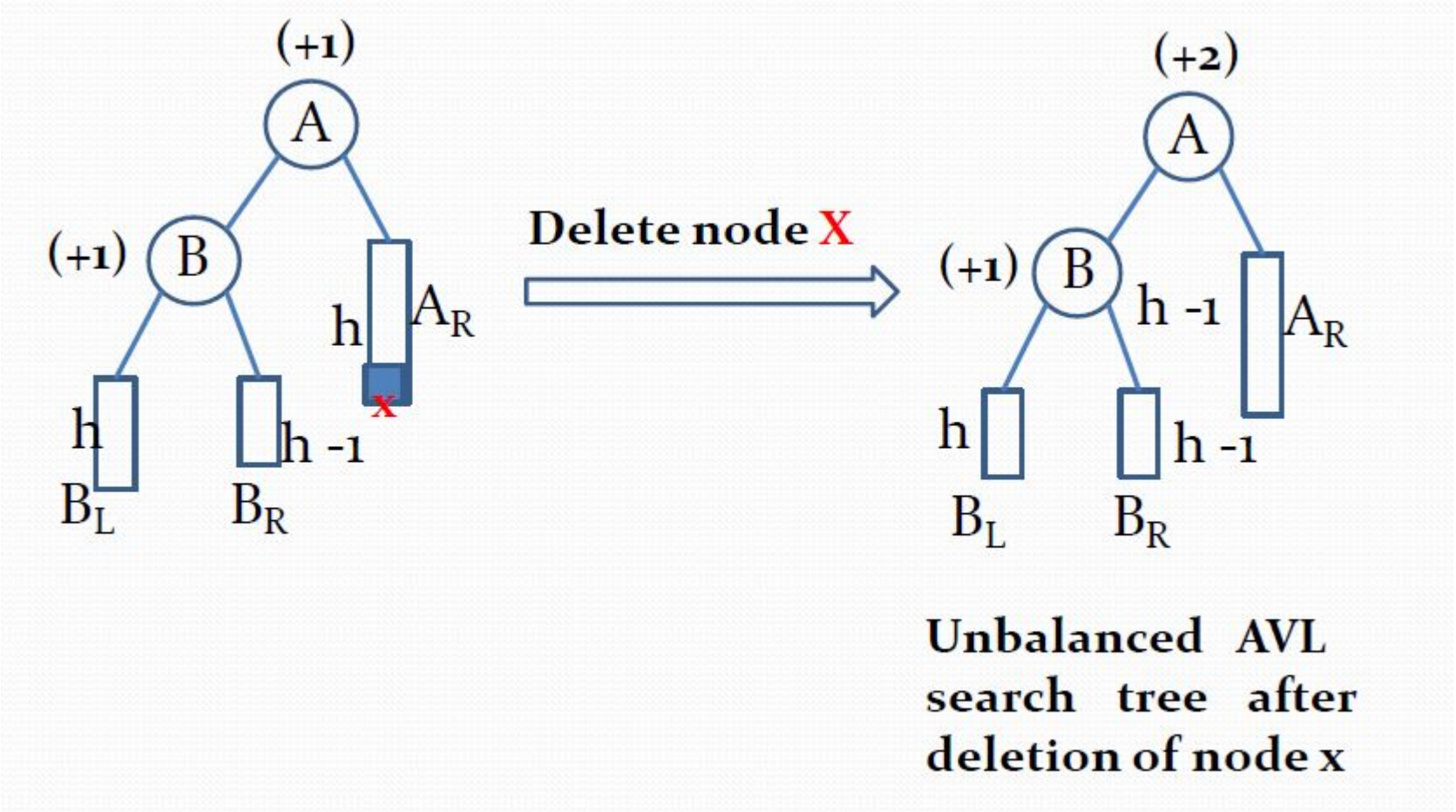
R0 Rotation Example



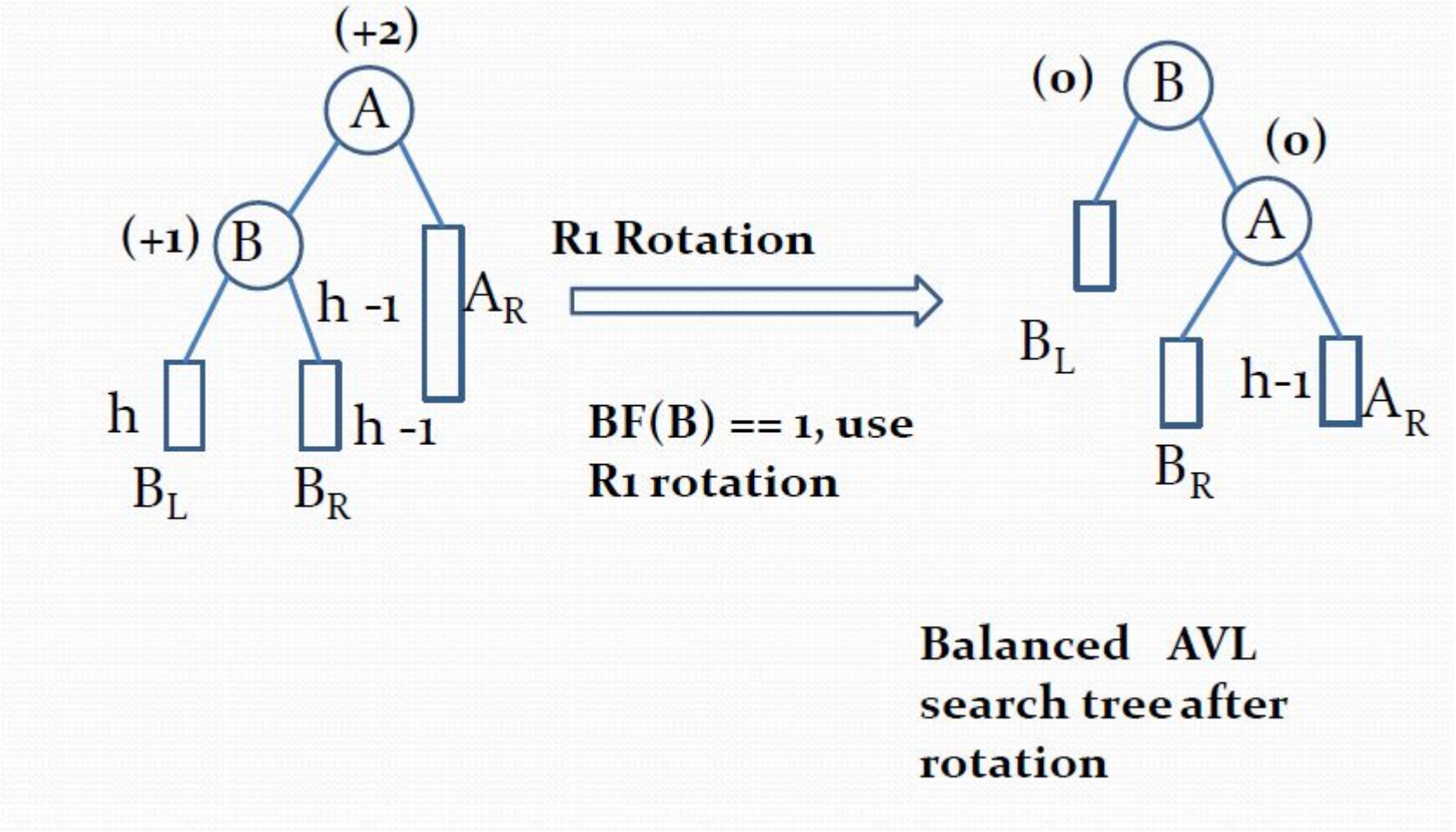
R0 Rotation Example



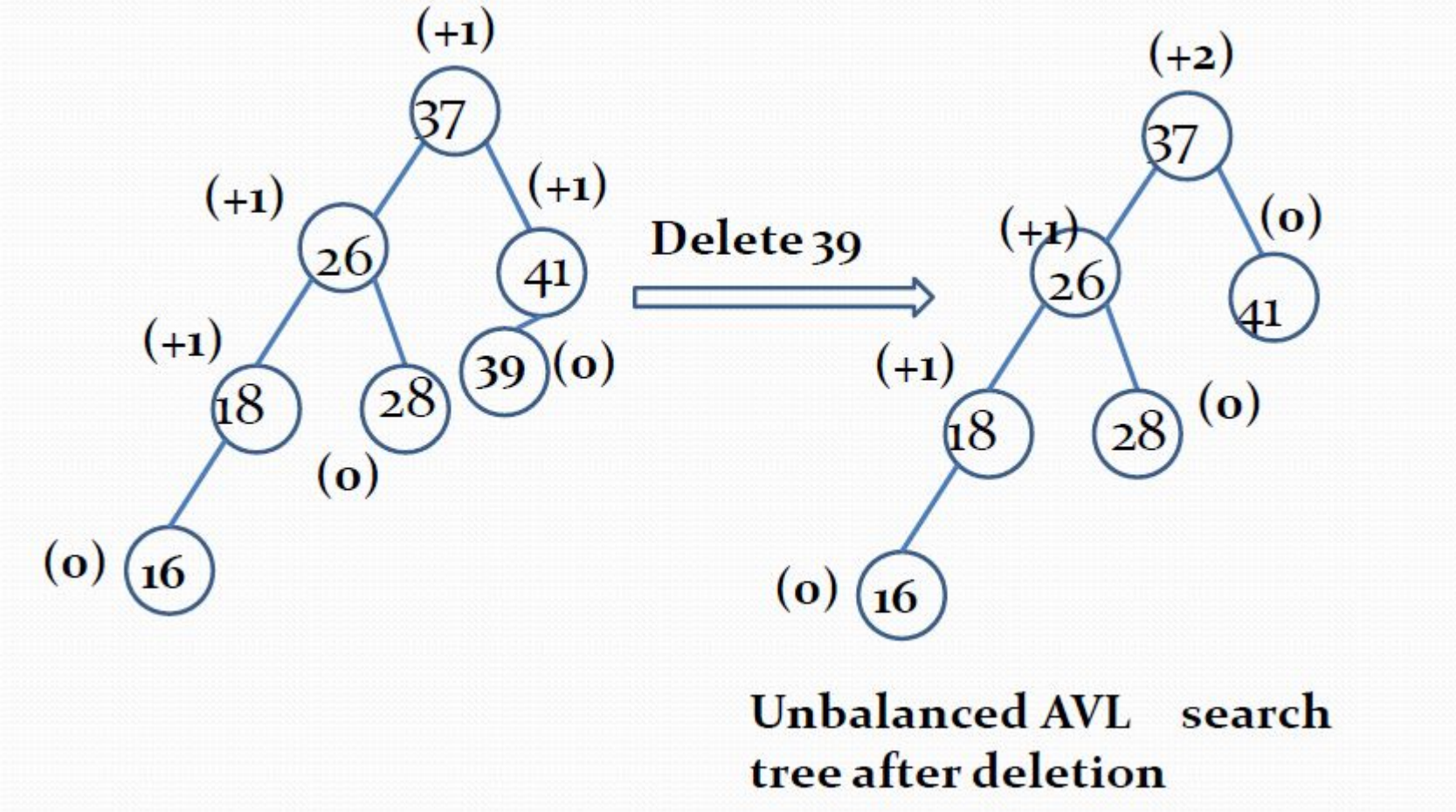
R1 Rotation


(+1)
A
(+1)
B
h
A_R
x
h
h -1
B_L
B_R
Delete node X
(+2)
A
(+1)
B
h -1
A_R
h
h -1
B_L
B_R
Unbalanced AVL search tree after deletion of node x

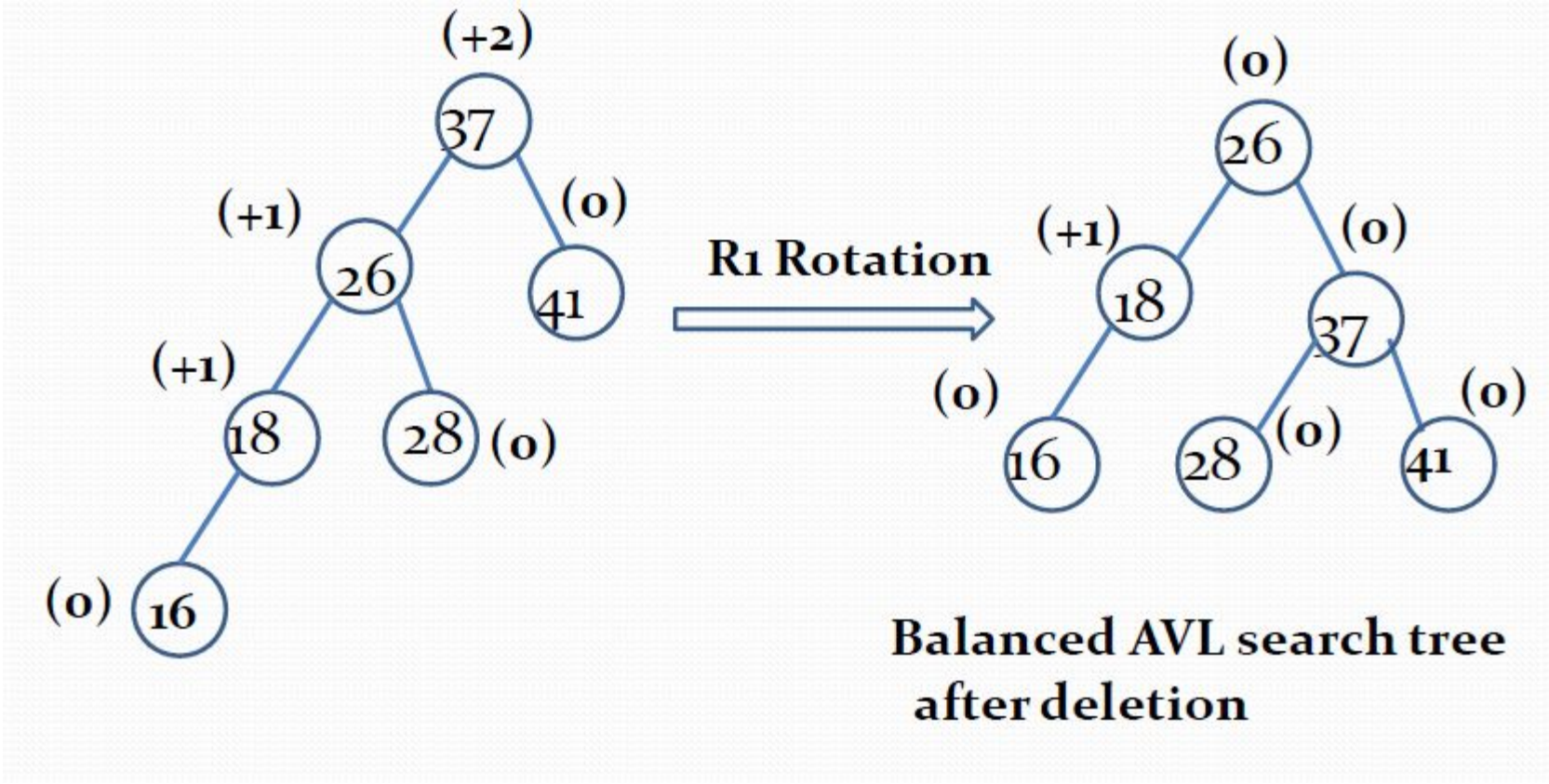
R1 Rotation


(+2)
A
(+1)
B
h -1
A_R
h
h -1
B_L
B_R
R1 Rotation
BF(B) == 1, use
R1 rotation
(0)
B
(0)
A
B_L
h-1
A_R
B_R
Balanced AVL
search tree after
rotation

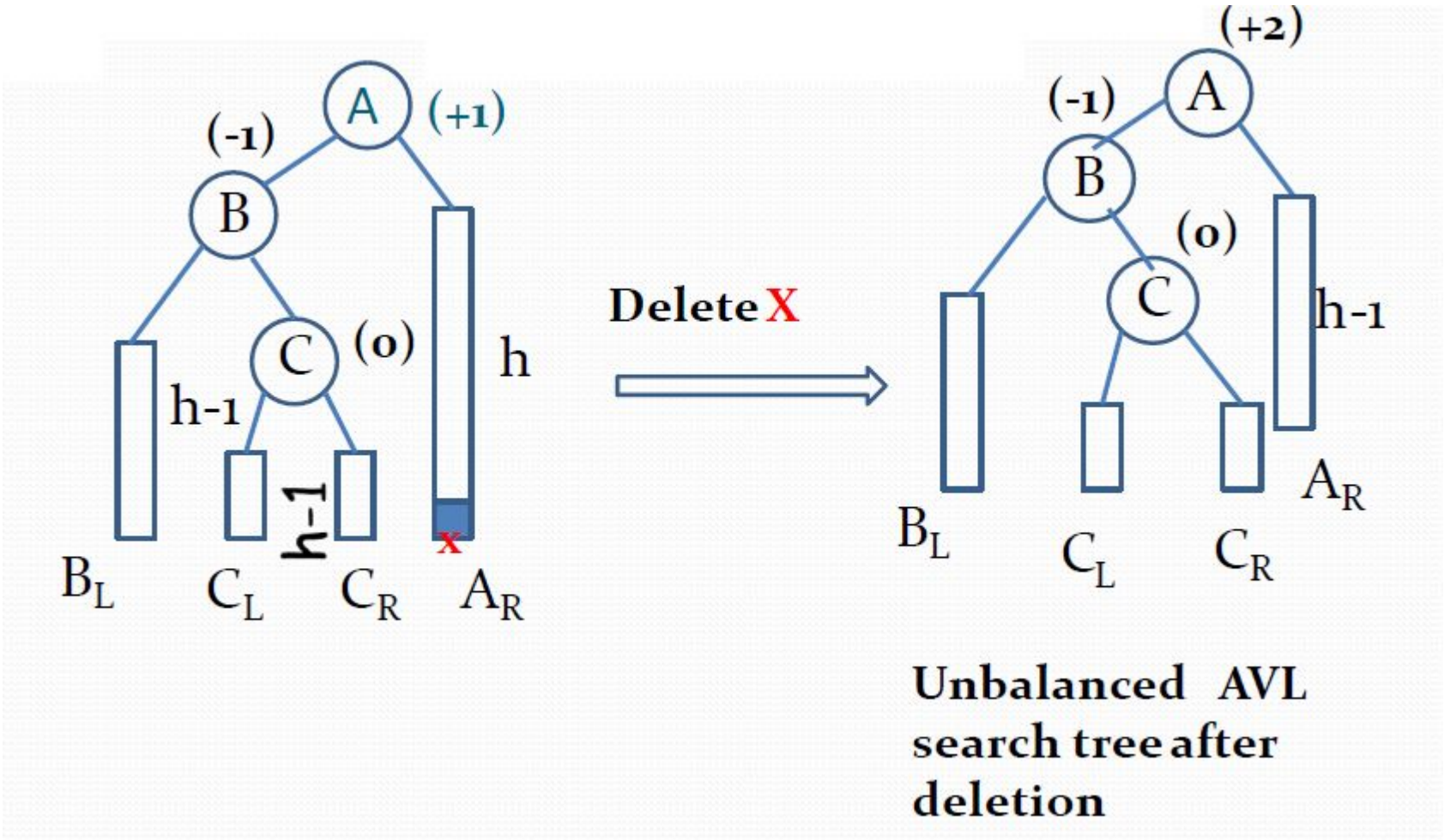
R1 Rotation Example



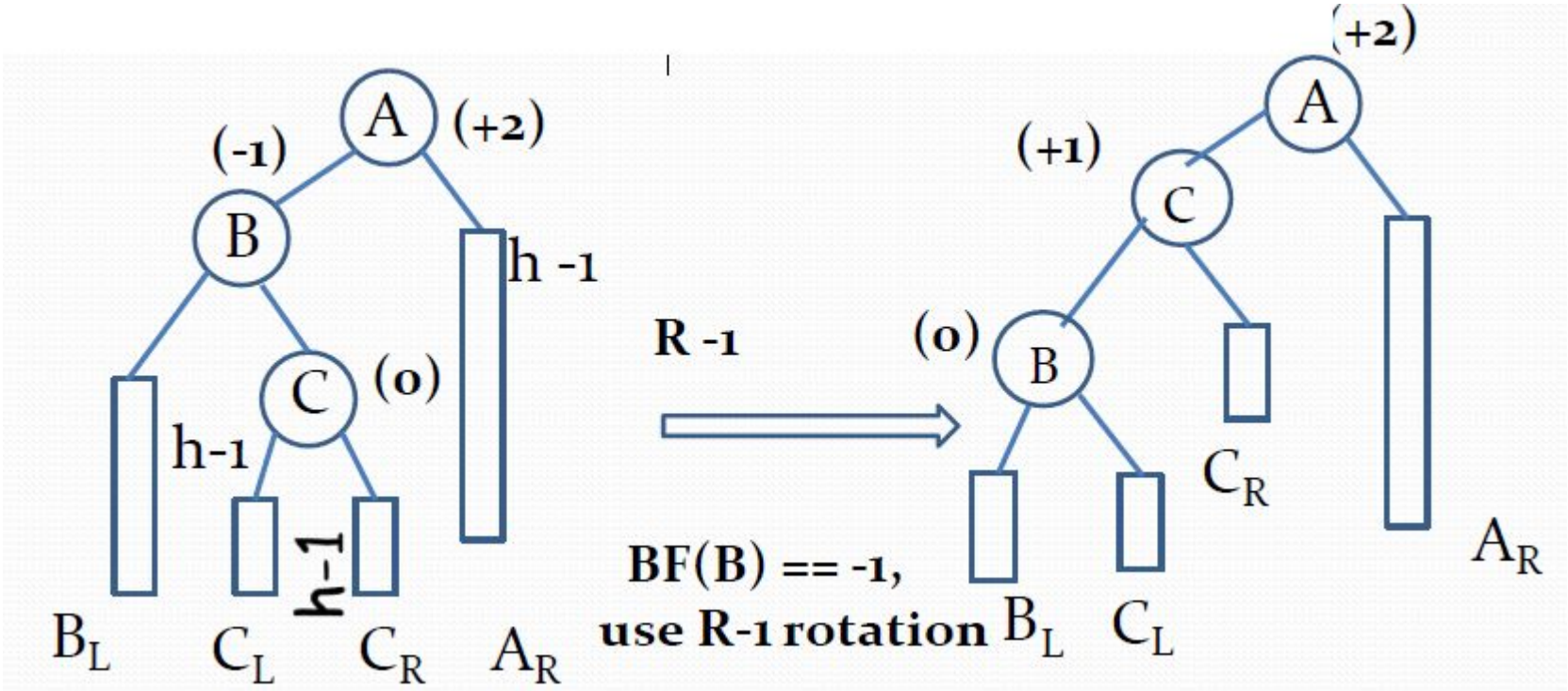
R1 Rotation Example



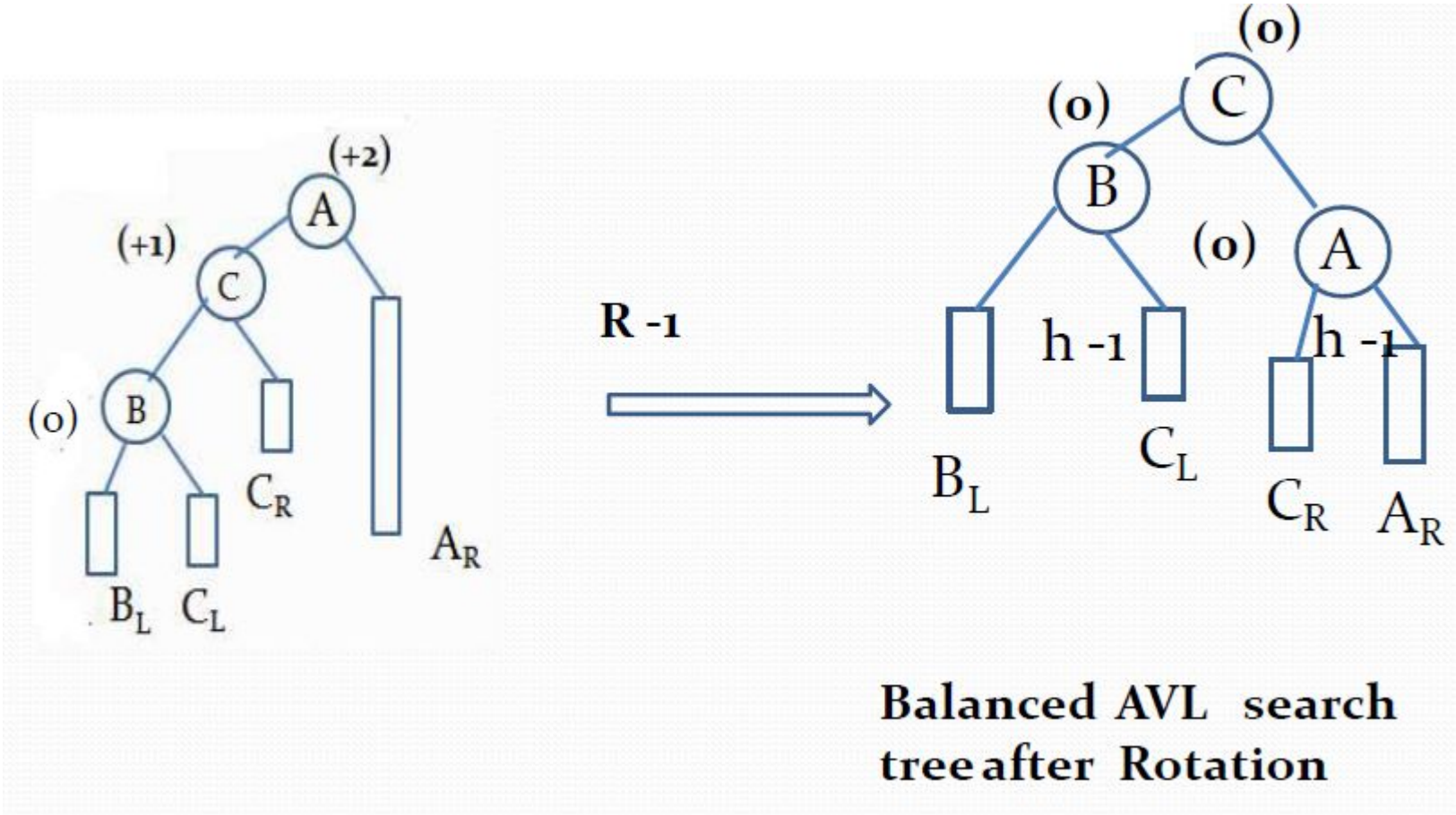
R-1 Rotation


A
(-1)
(+1)
B
C
(o)
h
h-1
h-1
x
B_L
C_L
C_R
A_R
Delete X
(+2)
(-1)
A
B
(o)
C
h-1
A_R
B_L
C_L
C_R
Unbalanced AVL search tree after deletion

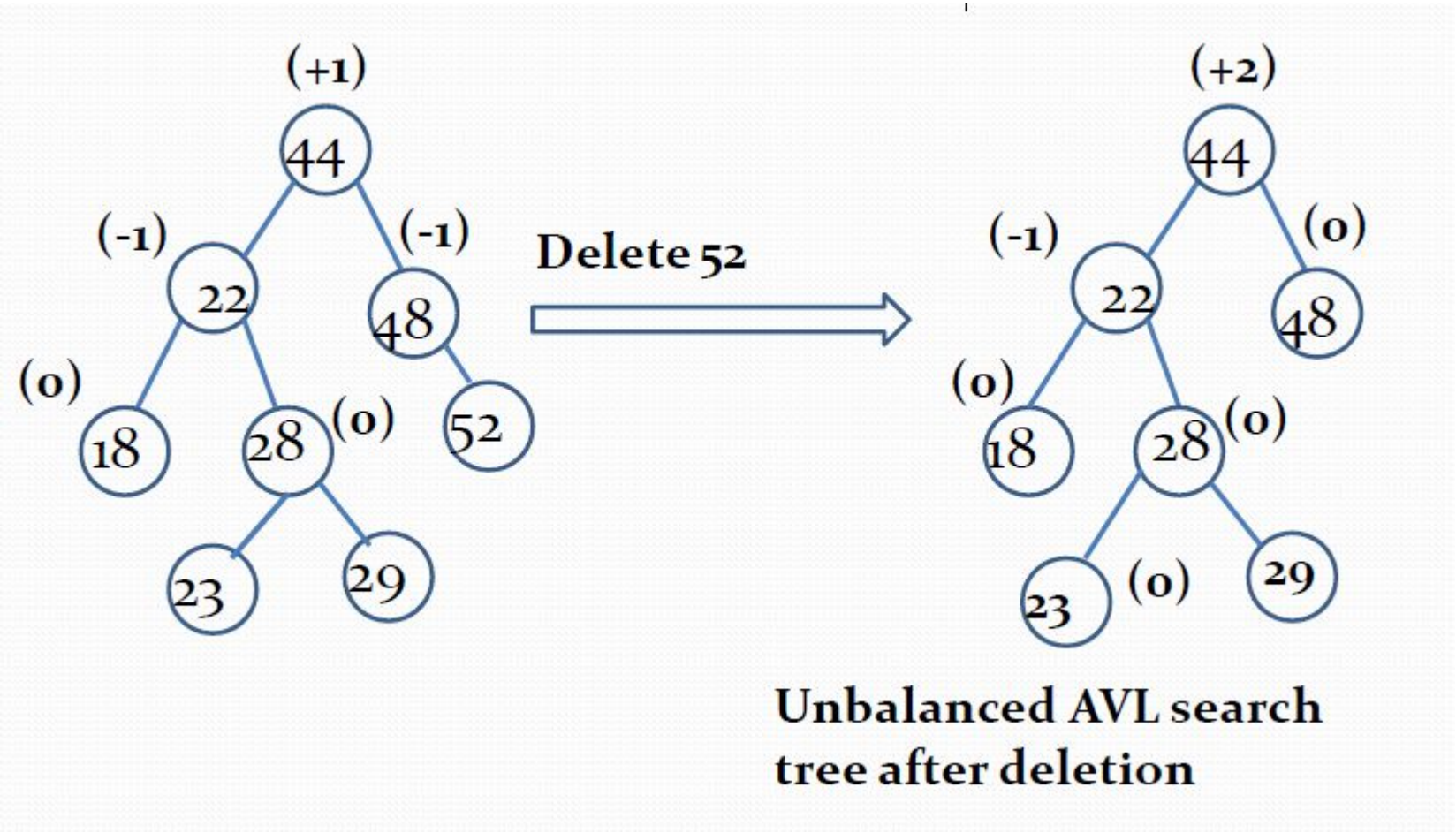
R-1 Rotation


A
(-1)
(+2)
B
h -1
C
(o)
h-1
h-1
B_L
C_L
C_R
A_R
R -1
BF(B) == -1,
use R-1 rotation
(+2)
A
(+1)
C
(o)
B
C_R
A_R
B_L
C_L

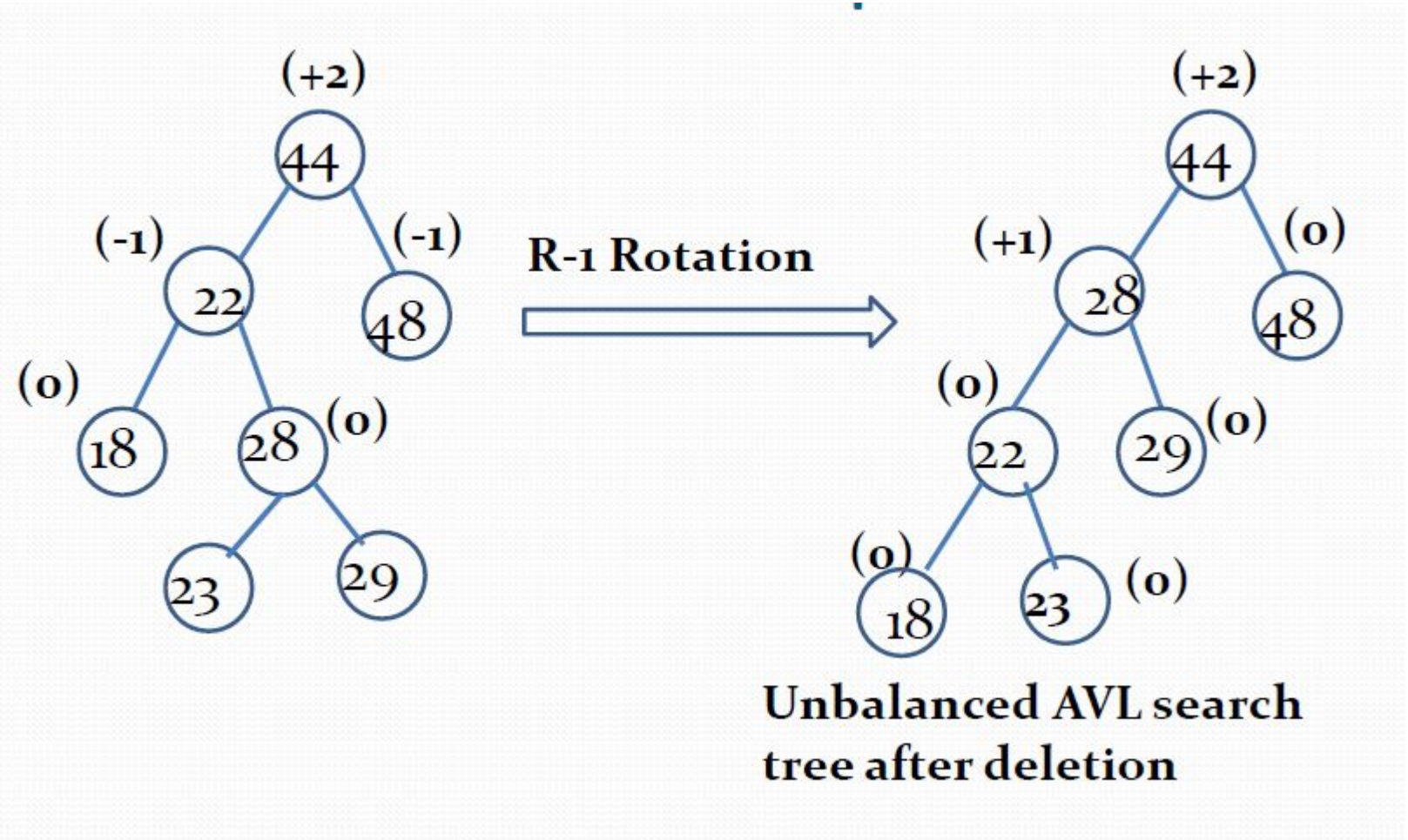
R-1 Rotation


(+2)
A
(+1)
C
(o)
B
C_R
A_R
B_L
C_L
R -1
(o)
C
(o)
B
(o)
A
h -1
h -1
B_L
C_L
C_R
A_R
Balanced AVL search tree after Rotation

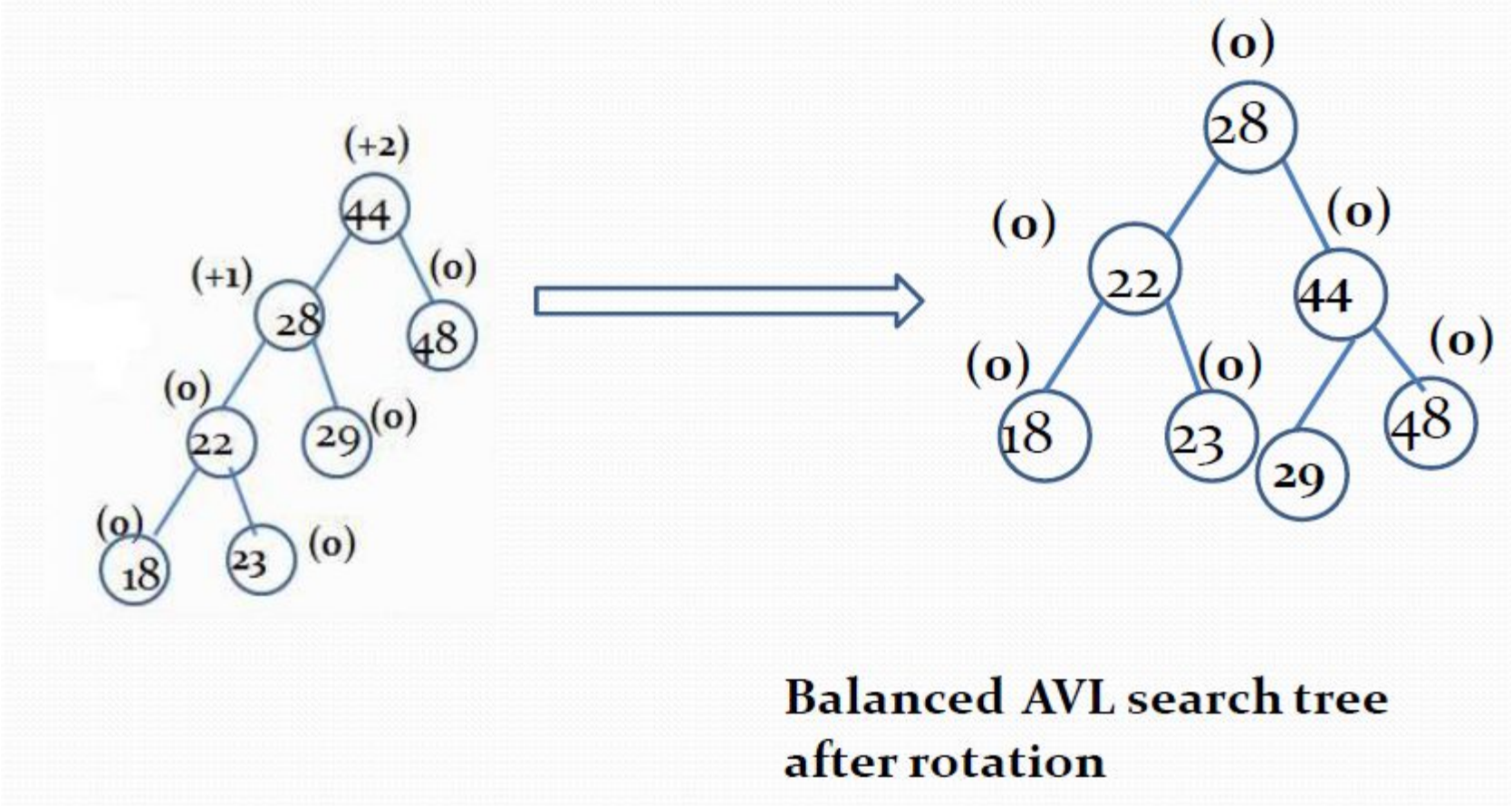
R-1 Rotation Example



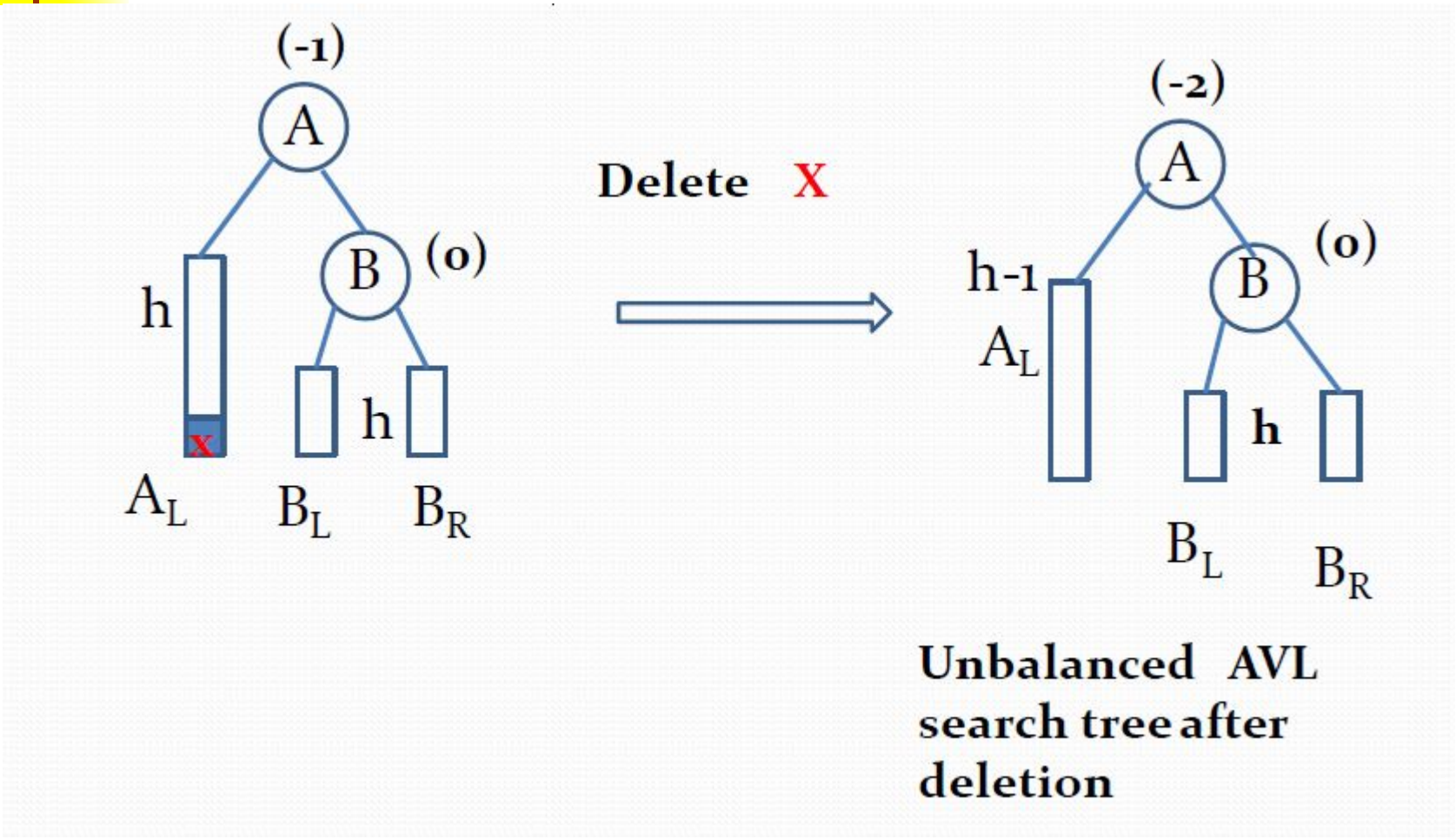
R-1 Rotation Example



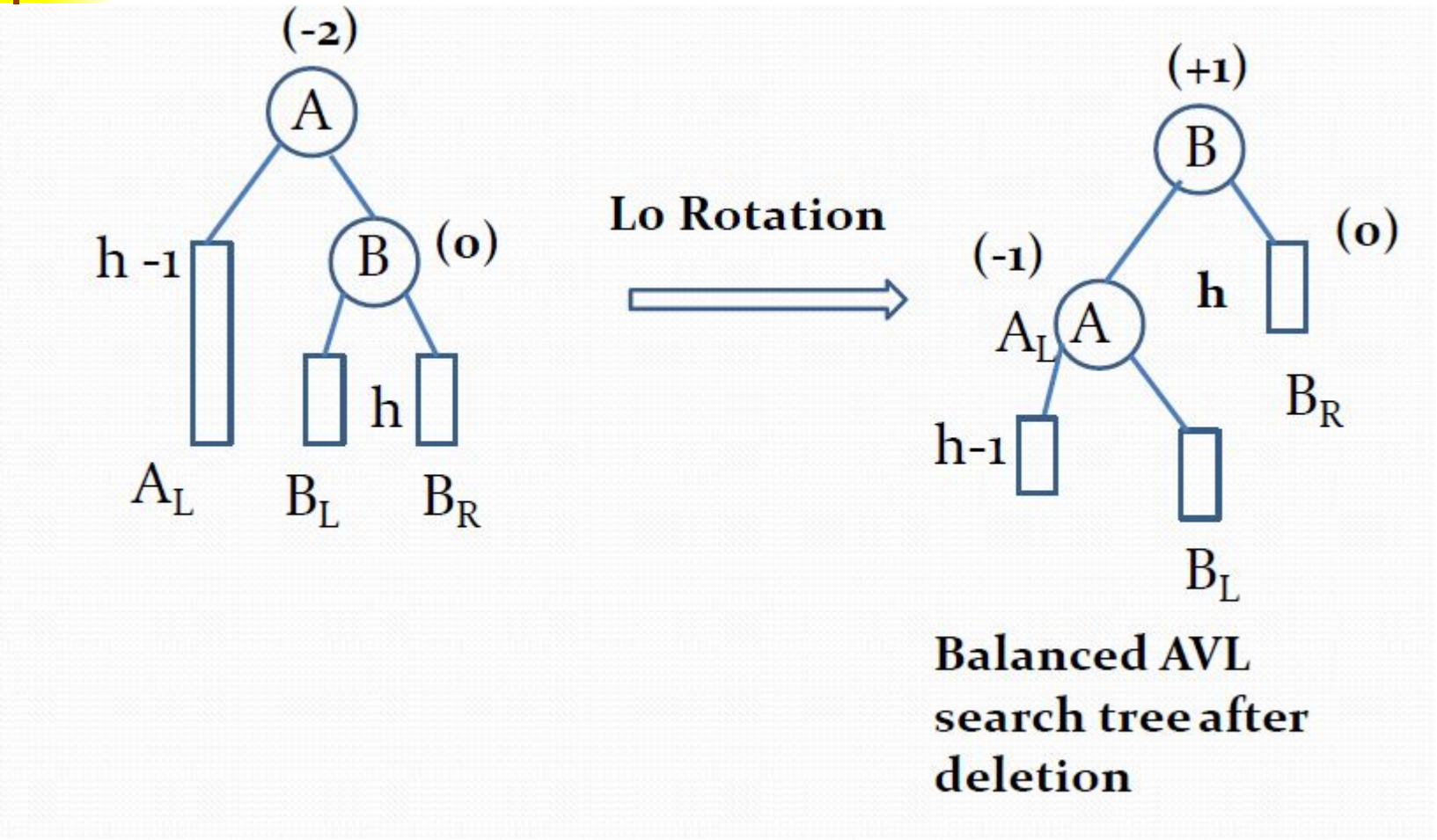
R-1 Rotation Example



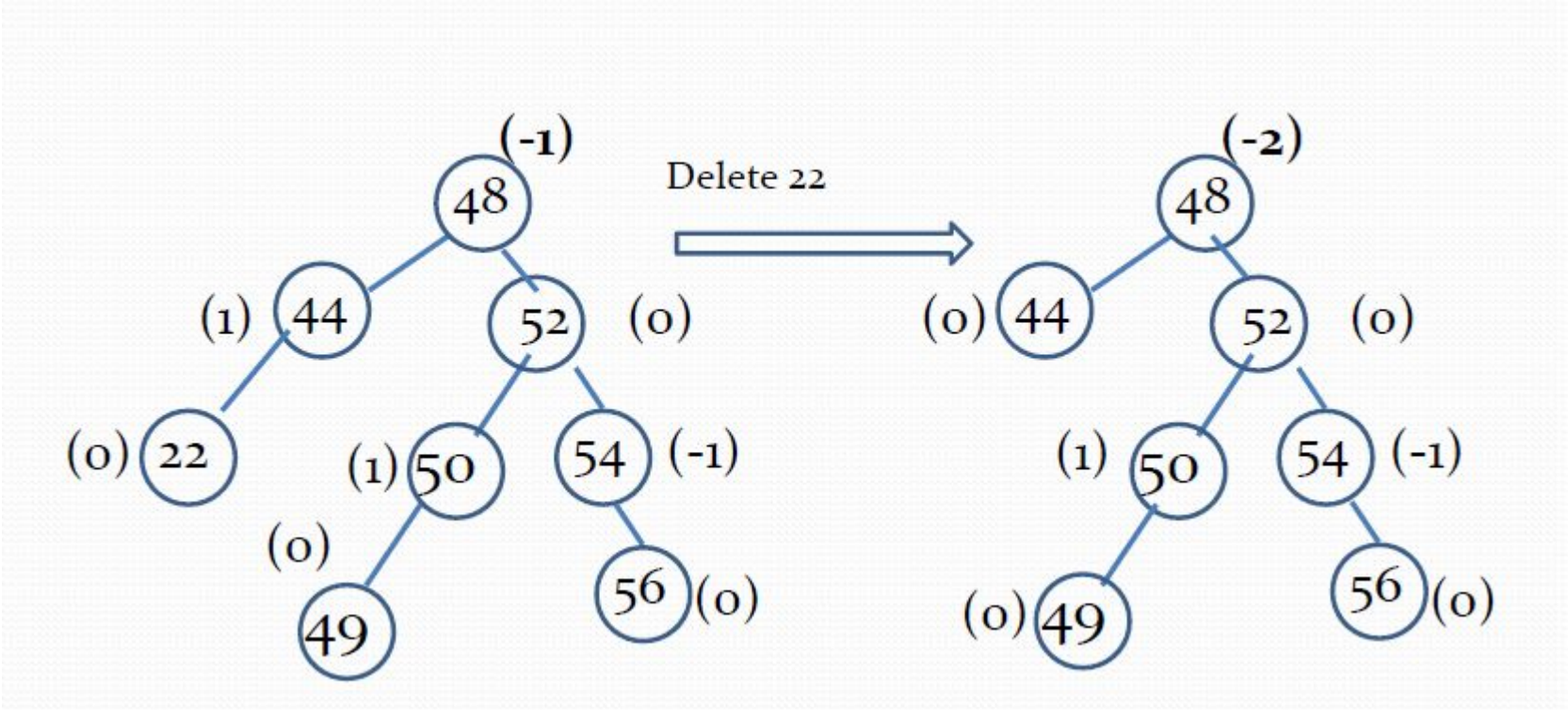
L0 Rotation


(-1)
A
B
(o)
h
h
x
A_L
B_L
B_R
Delete X
(-2)
A
B
(o)
h-1
A_L
h
B_L
B_R
Unbalanced AVL search tree after deletion

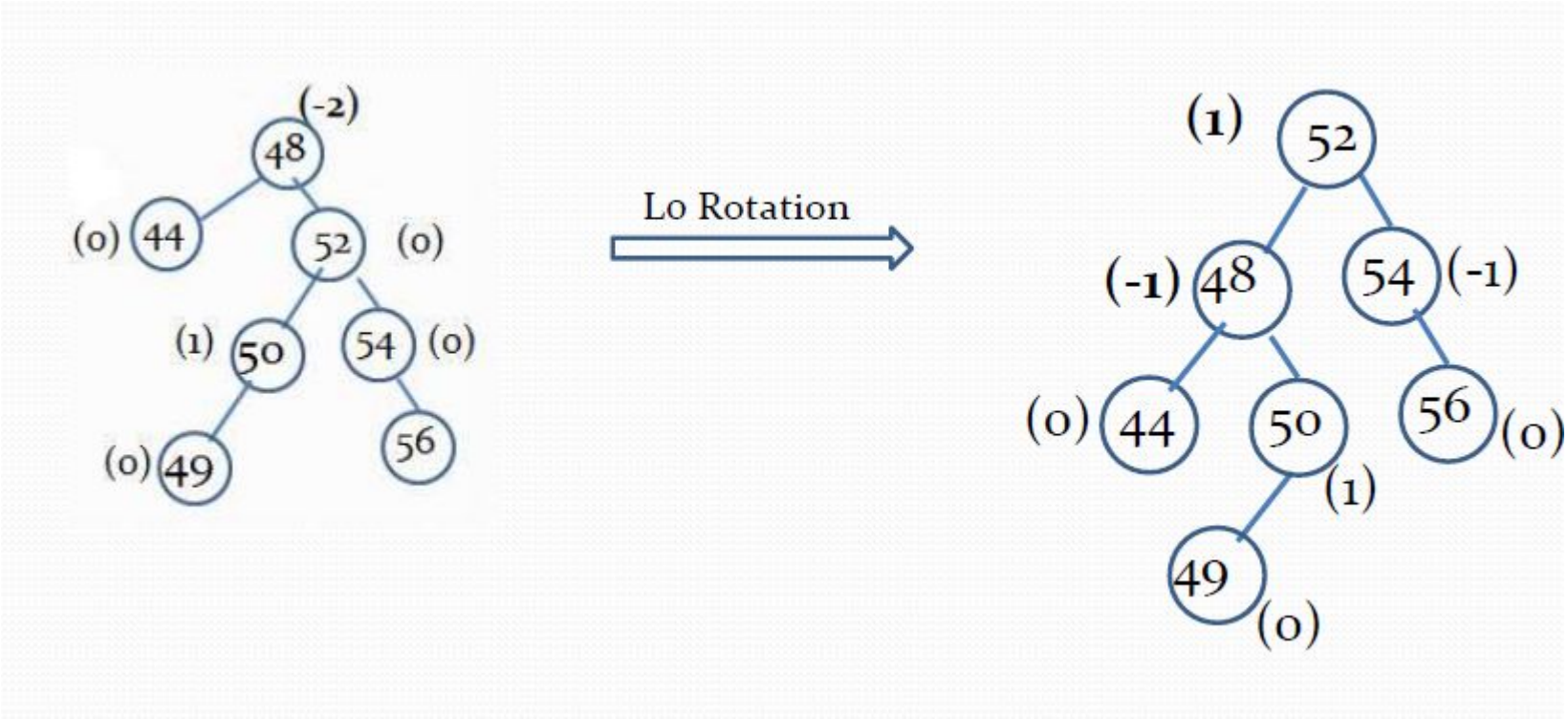
L0 Rotation


(-2)
A
h -1
B
(0)
h
A_L
B_L
B_R
Lo Rotation
(+1)
B
(-1)
h
(0)
A_L
A
B_R
h-1
B_L
Balanced AVL search tree after deletion

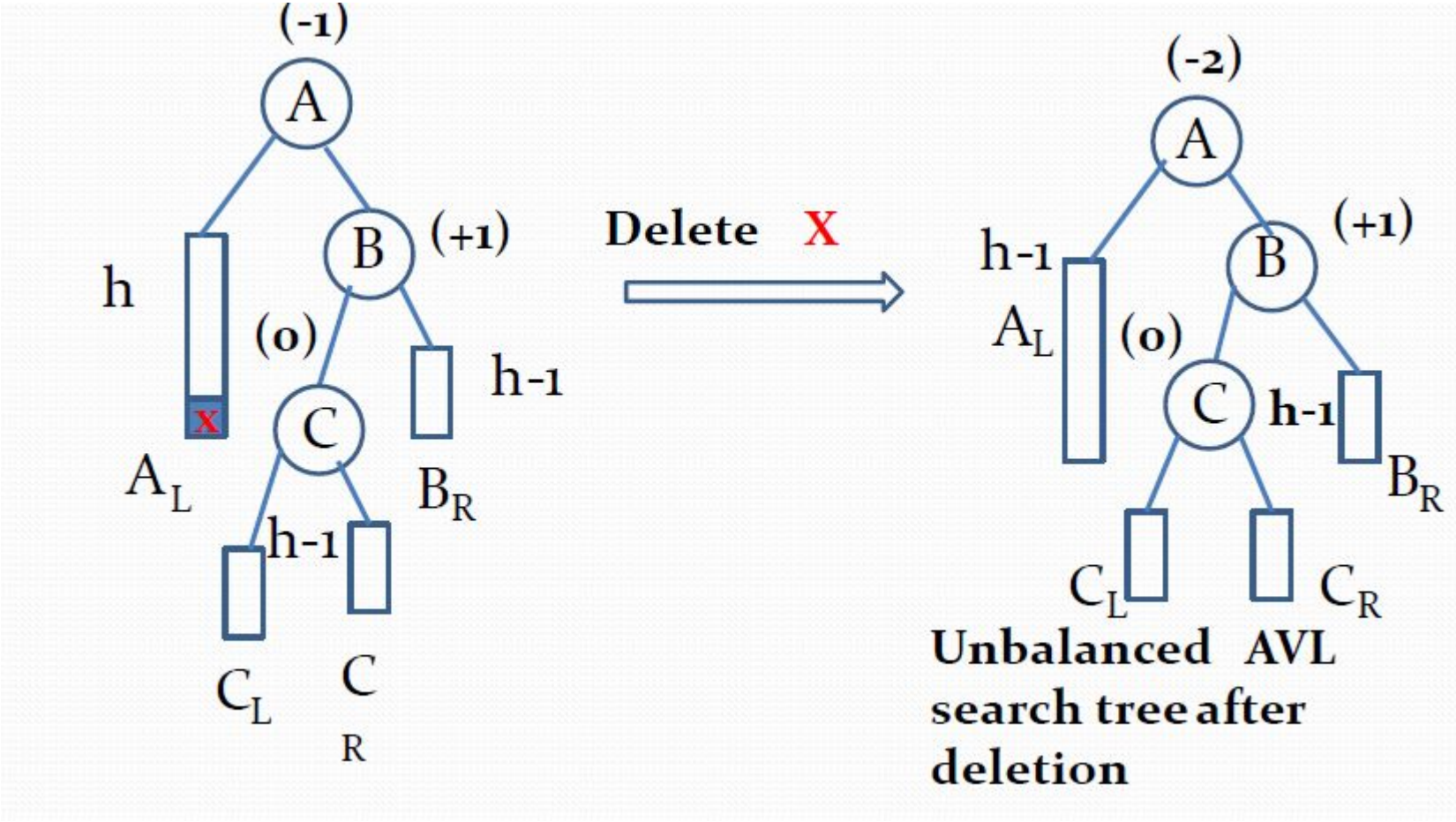
L0 Rotation


(-1)
48
(1) 44
52 (0)
(0) 22
(1) 50
54 (-1)
(0)
49
56 (0)
Delete 22
(-2)
48
(0) 44
52 (0)
(1) 50
54 (-1)
(0) 49
56 (0)

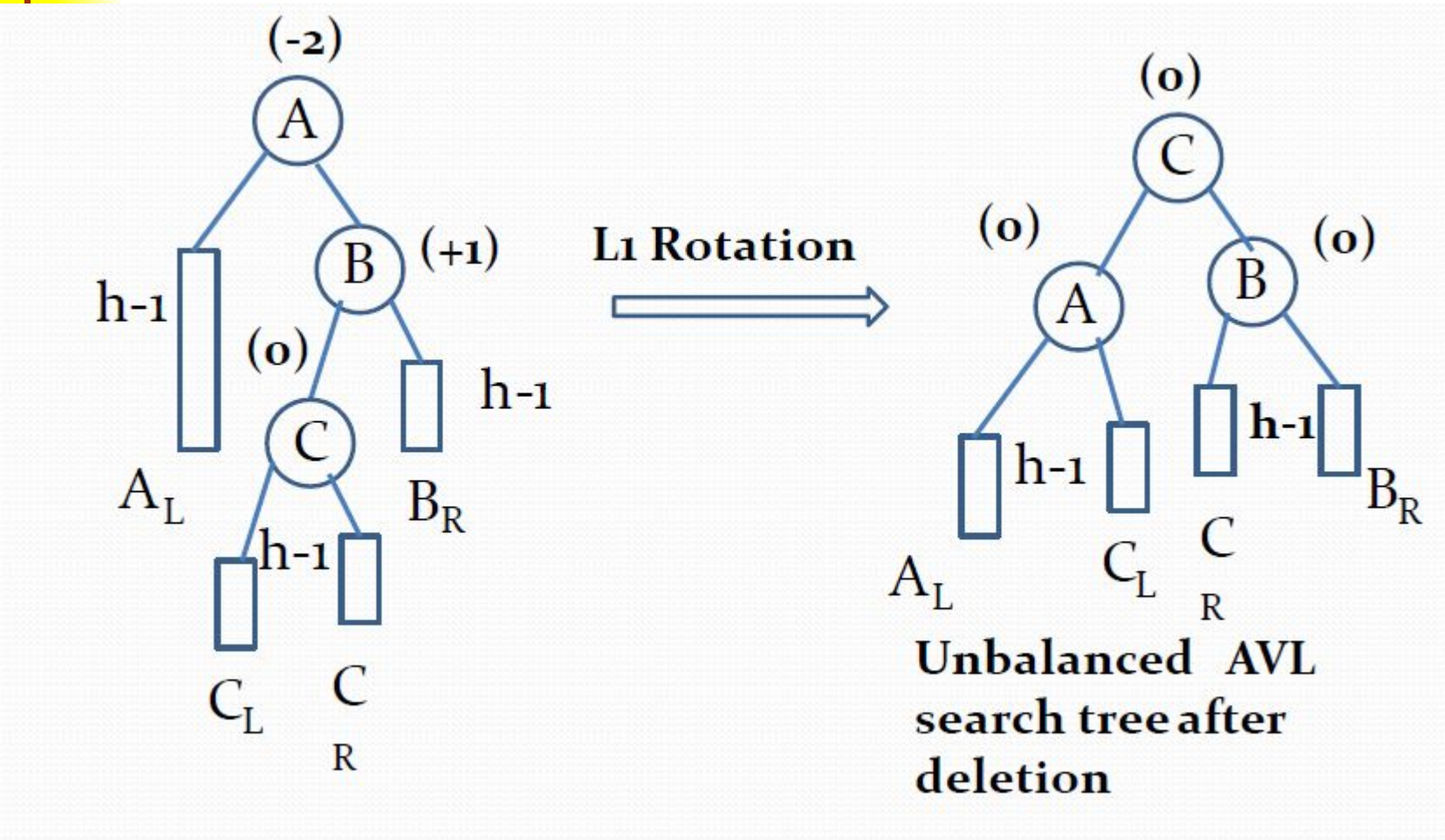
L0 Rotation


(-2)
48
(0) 44
52 (0)
(1) 50
54 (0)
(0) 49
56
Lo Rotation
(1)
52
(-1) 48
54 (-1)
(0) 44
50
56 (0)
(1)
49
(0)

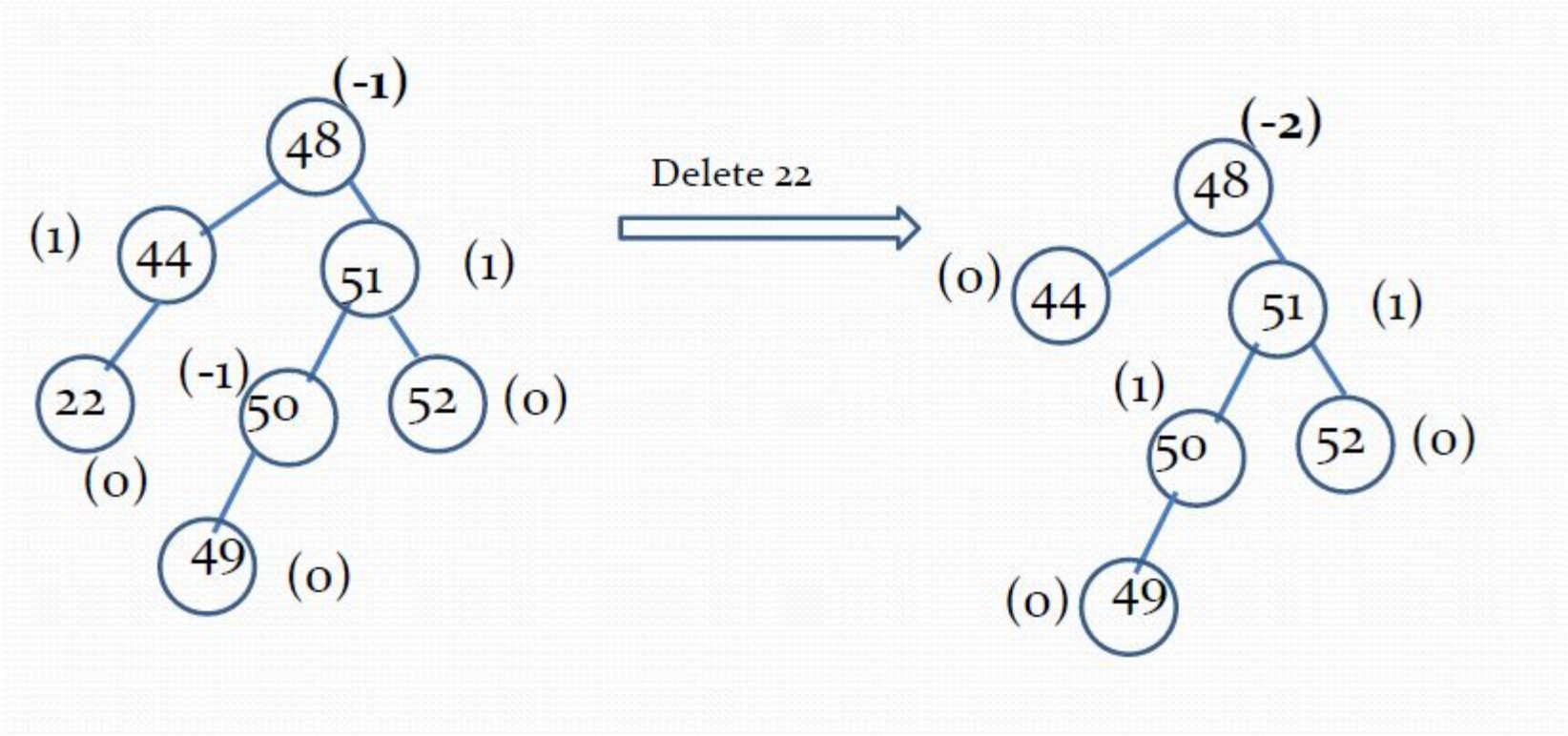
L1 Rotation


(-1)
A
B
(+1)
h
(o)
h-1
x
C
A_L
B_R
h-1
C_L
C
R
Delete X
(-2)
A
h-1
B
(+1)
A_L
(o)
C
h-1
B_R
C_L
C_R
Unbalanced AVL search tree after deletion

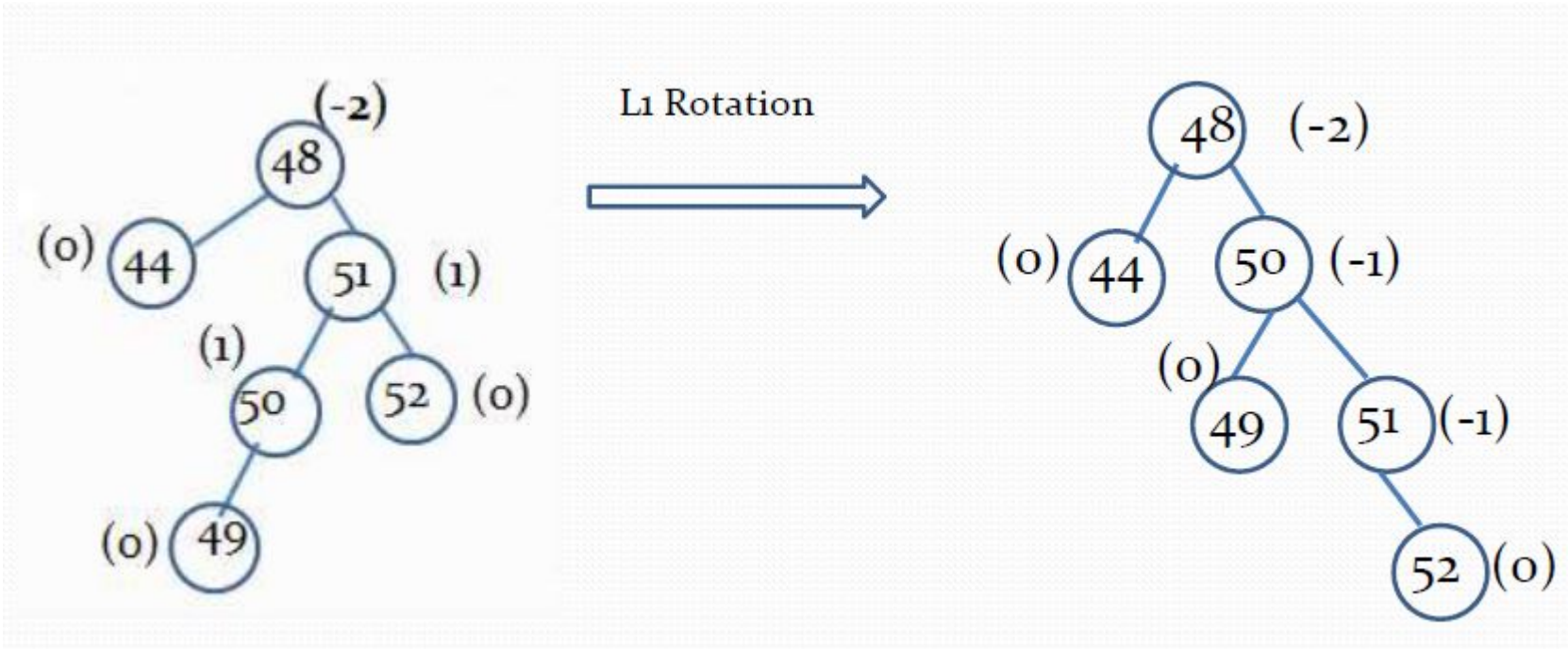
L1 Rotation


(-2)
A
B
(+1)
h-1
(0)
C
h-1
A_L
B_R
h-1
C_L
C
R
L1 Rotation
(0)
C
(0)
A
B
(0)
h-1
h-1
B_R
A_L
C_L
C
R
Unbalanced AVL search tree after deletion

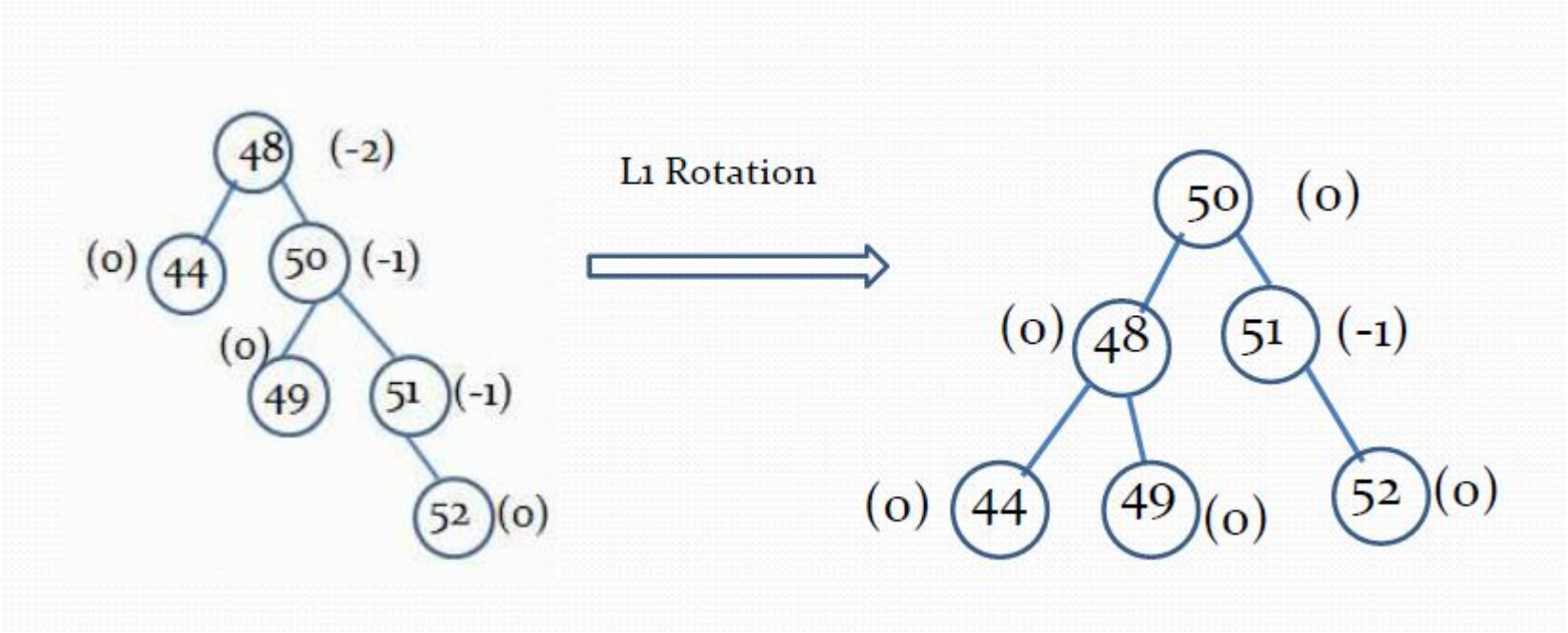
L1 Rotation



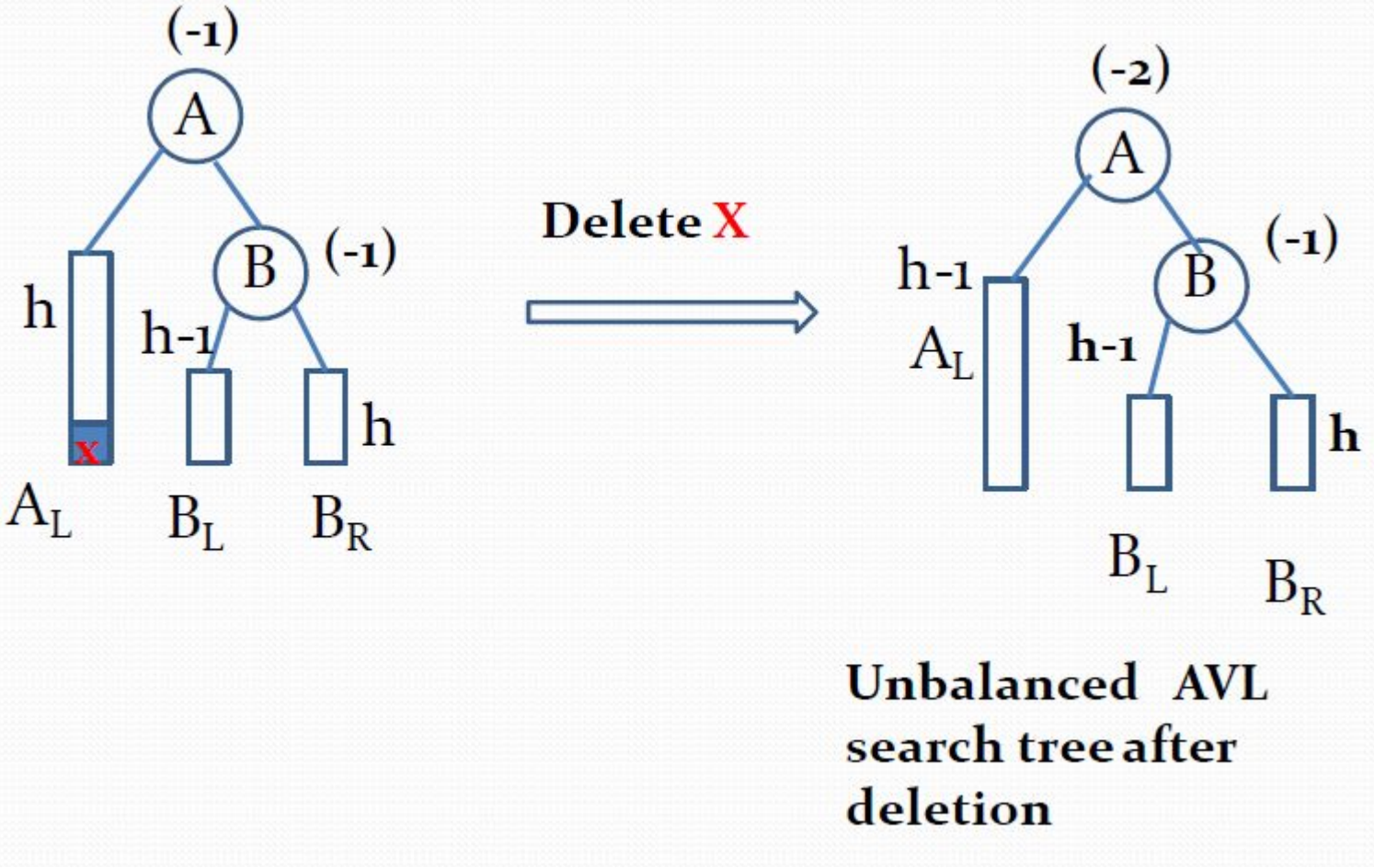
L1 Rotation



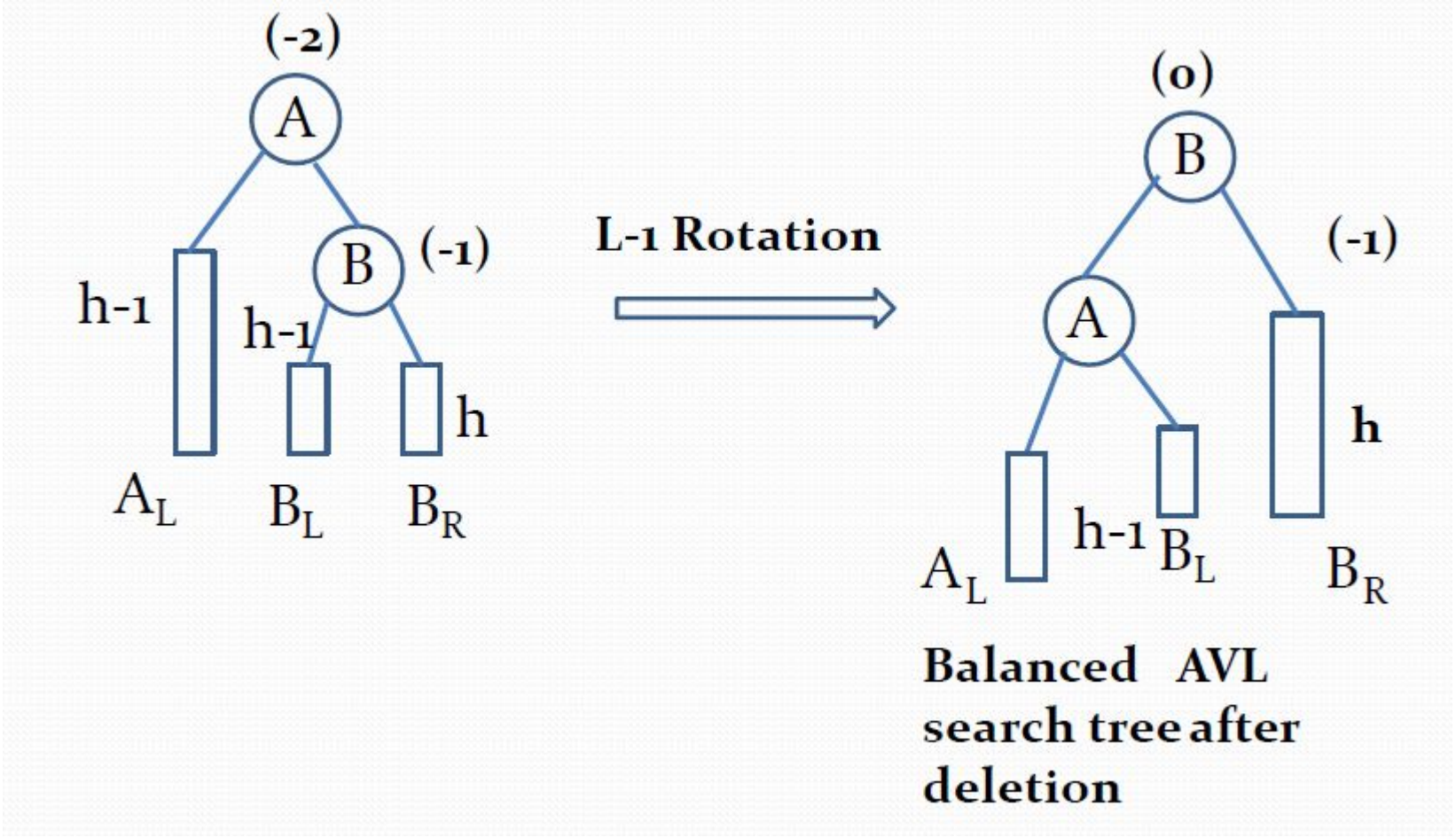
L1 Rotation



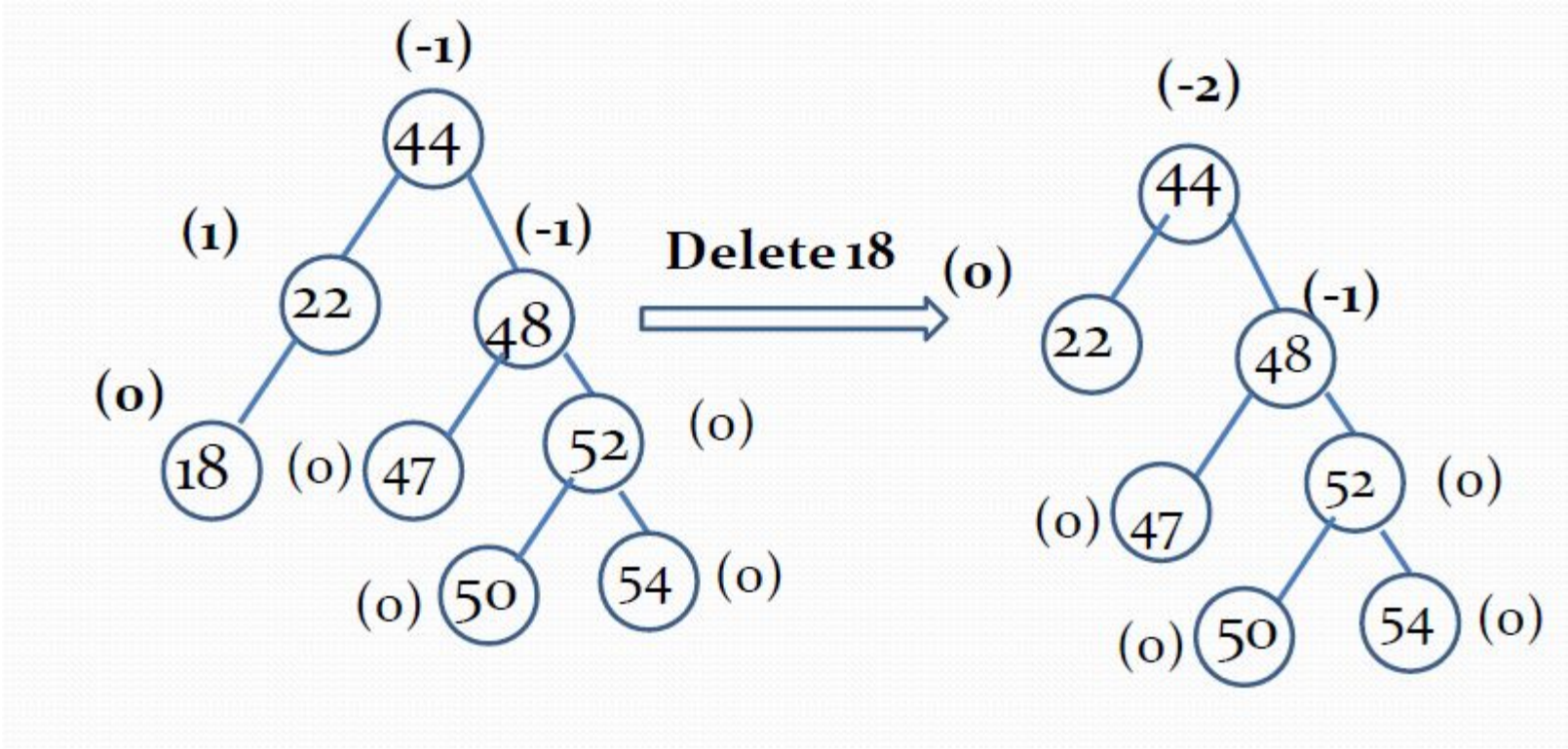
L-1 Rotation


(-1)
A
B
(-1)
h
h-1
h
x
A_L
B_L
B_R
Delete X
(-2)
A
h-1
B
(-1)
A_L
h-1
h
B_L
B_R
Unbalanced AVL search tree after deletion

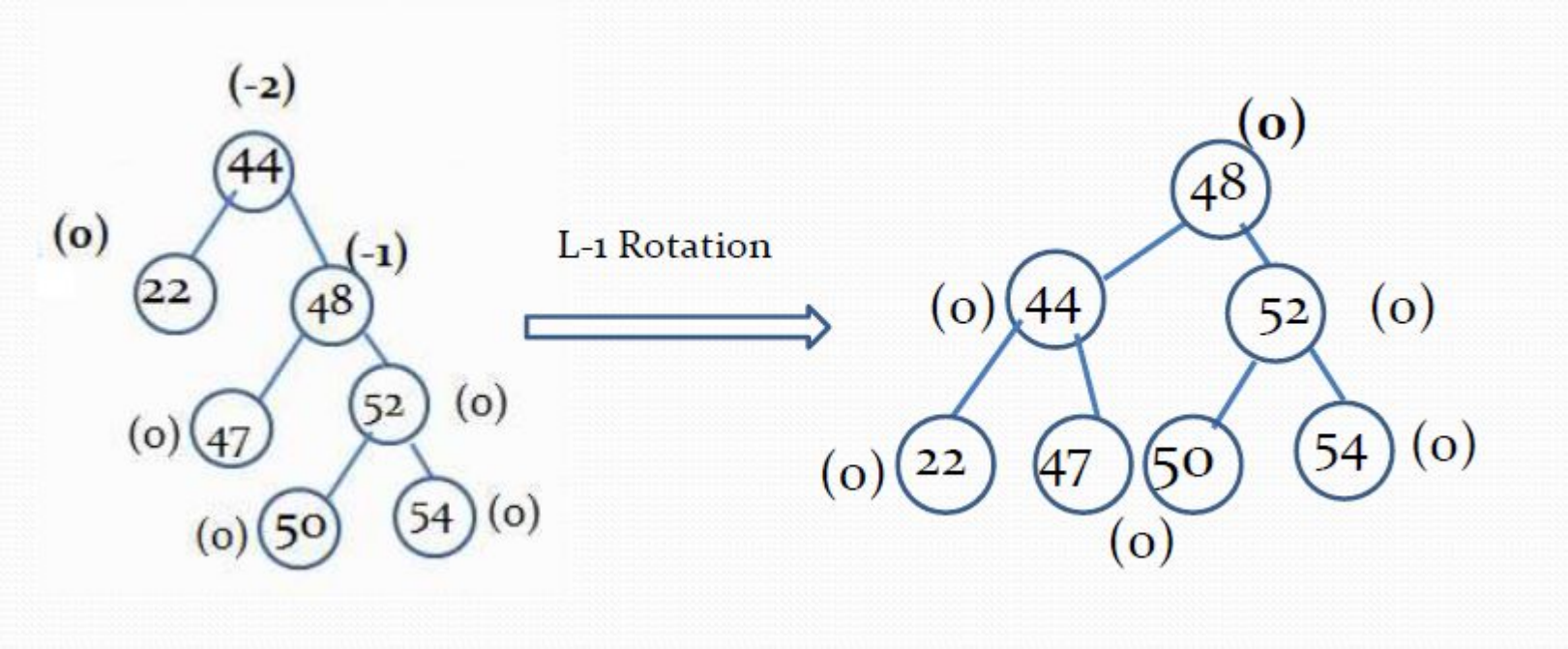
L-1 Rotation


(-2)
A
B
(-1)
h-1
h-1
h
A_L
B_L
B_R
L-1 Rotation
(0)
B
(-1)
A
h
h-1
A_L
B_L
B_R
Balanced AVL search tree after deletion

L-1 Rotation


(-1)
44
(1)
(-1)
22
48
(0)
18
(0)
47
52
(0)
(0)
50
54
(0)
Delete 18
(0)
(-2)
44
22
(-1)
48
(0)
47
52
(0)
(0)
50
54
(0)

L-1 Rotation


(-2)
44
(0)
22
(-1)
48
(0)
47
52
(0)
(0)
50
54
(0)
L-1 Rotation
(0)
48
(0)
44
52
(0)
(0)
22
47
50
54
(0)
(0)