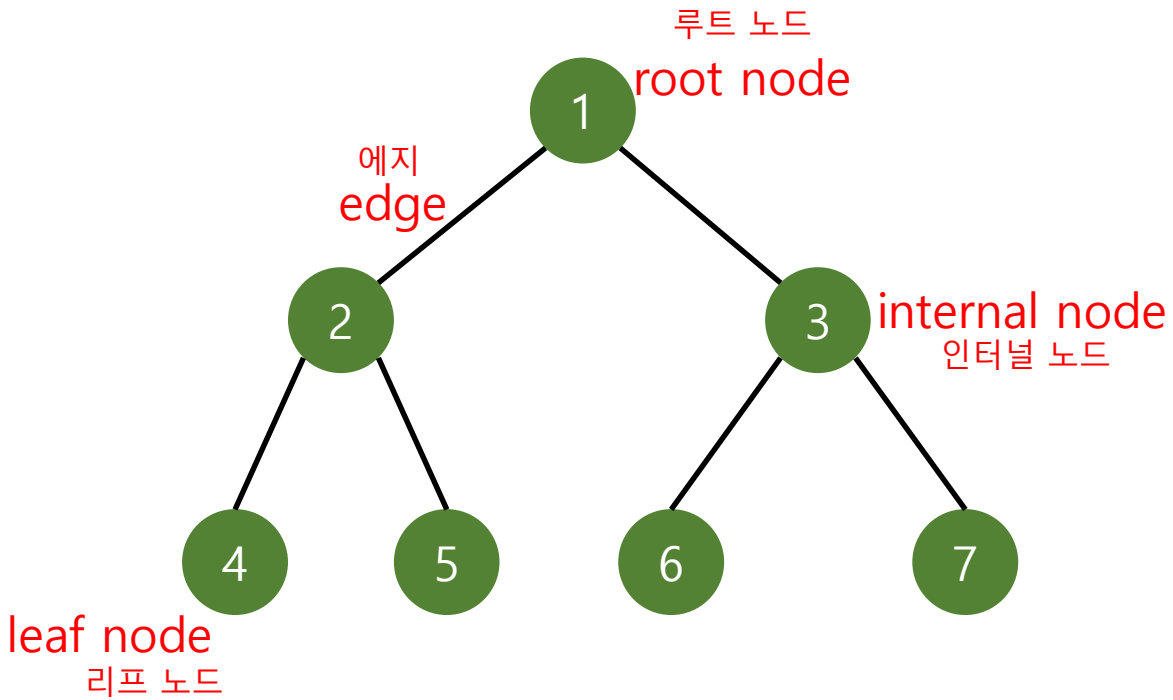
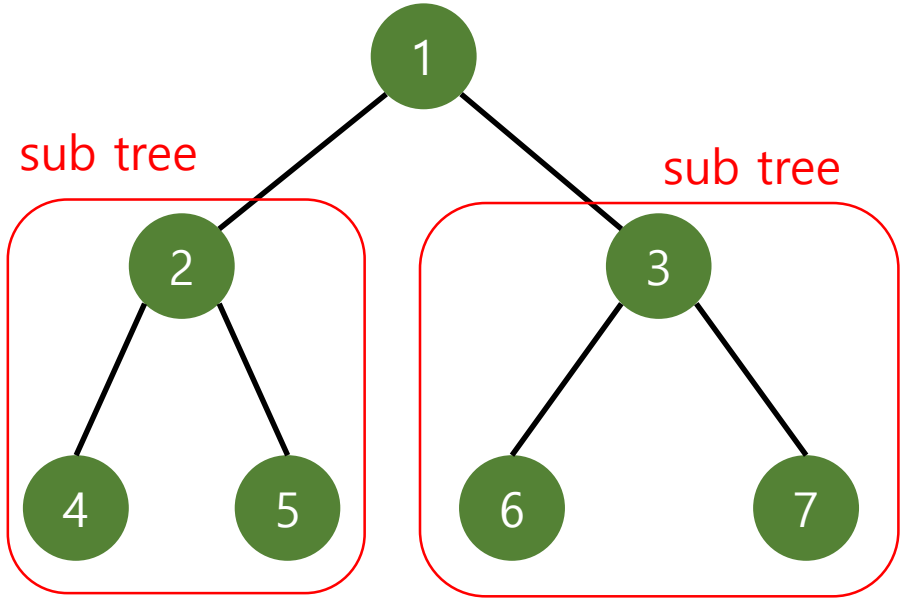
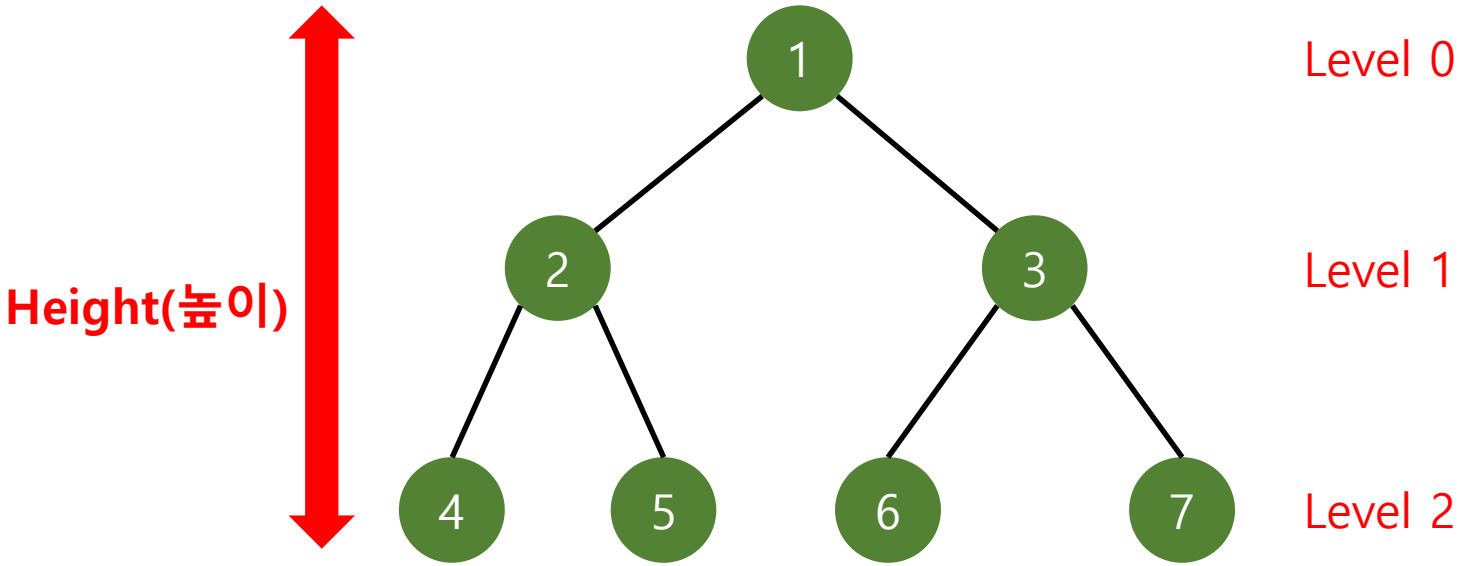
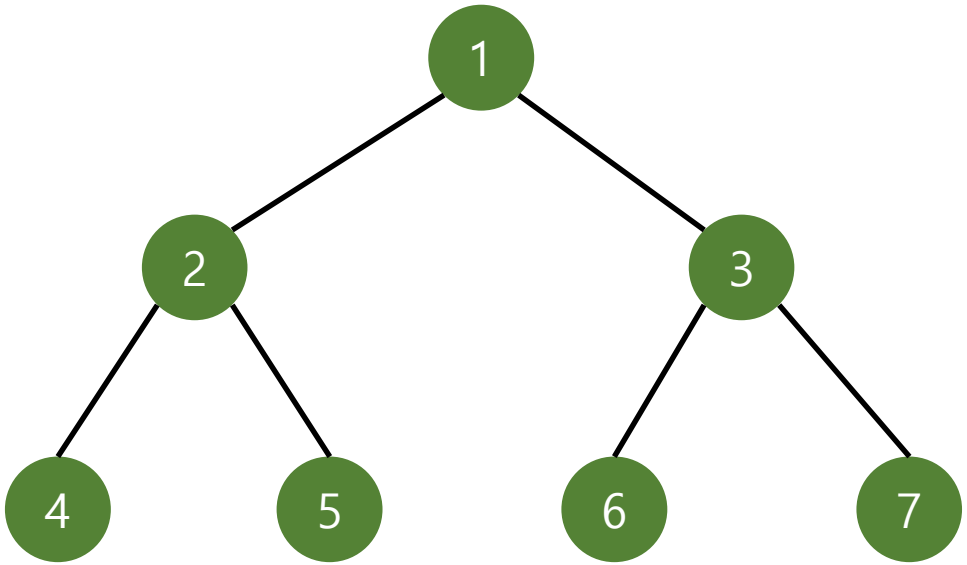
Tree

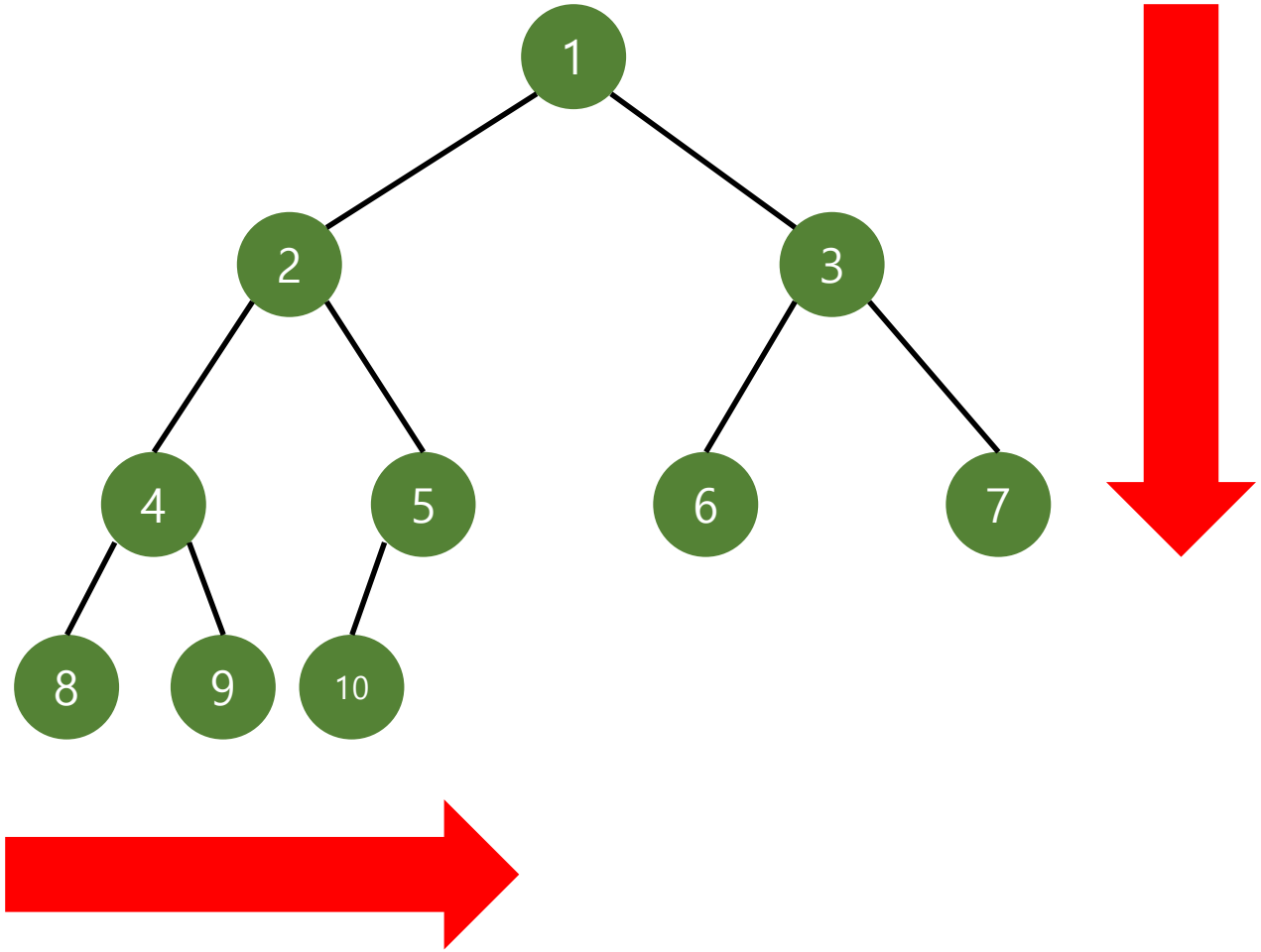
Connected acyclic graph

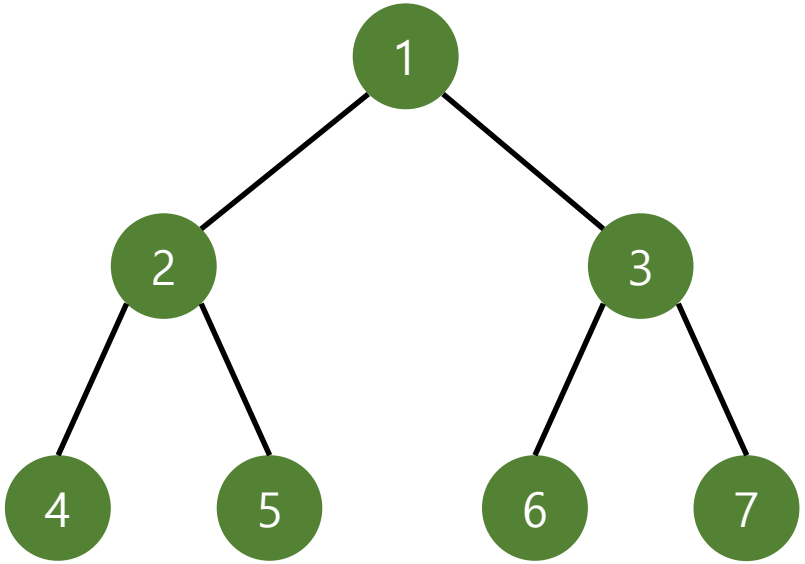

루트 노드
root node
1
에지
edge
2
3
internal node
인터널 노드
4
5
6
7
leaf node
리프 노드

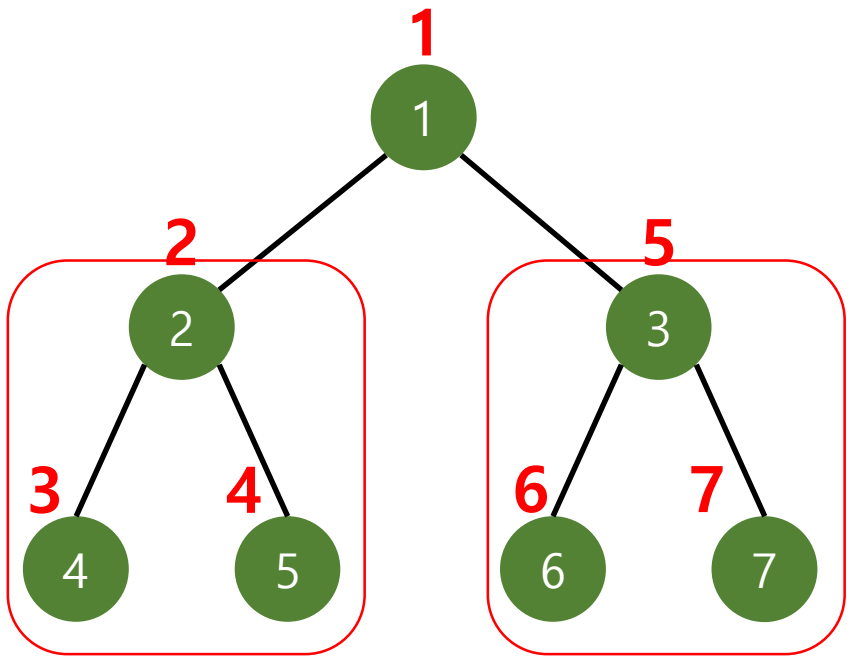

1
sub tree
sub tree
2
3
4
5
6
7

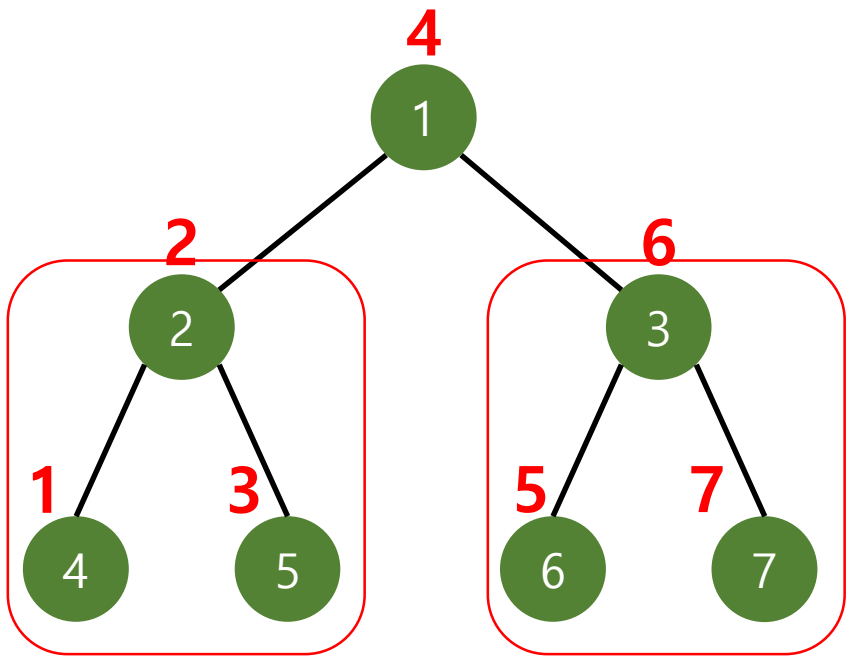

Height(높이)
1
2
3
4
5
6
7
Level 0
Level 1
Level 2

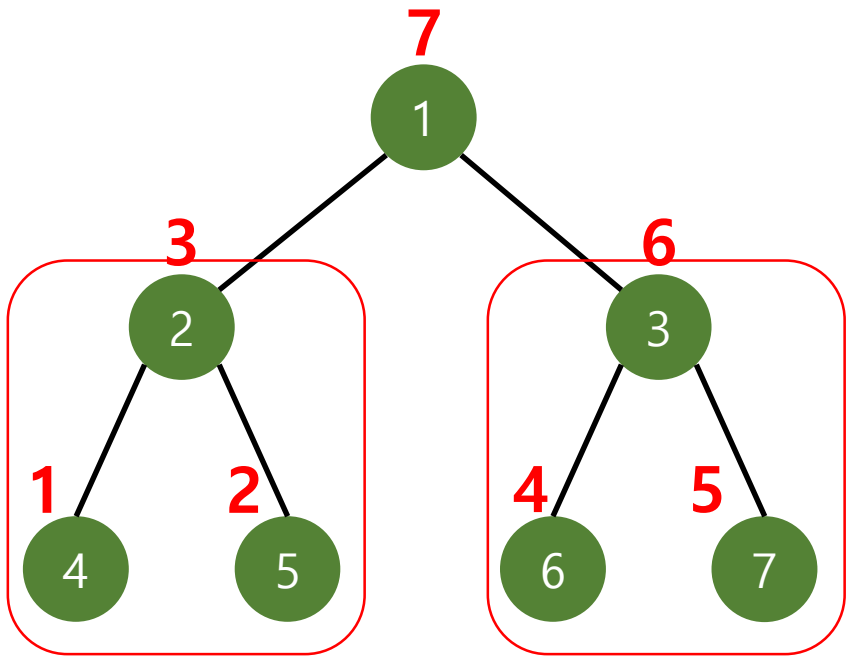

1
2
3
4
5
6
7


1
2
3
4
5
6
7
8
9
10


1
2
3
4
5
6
7

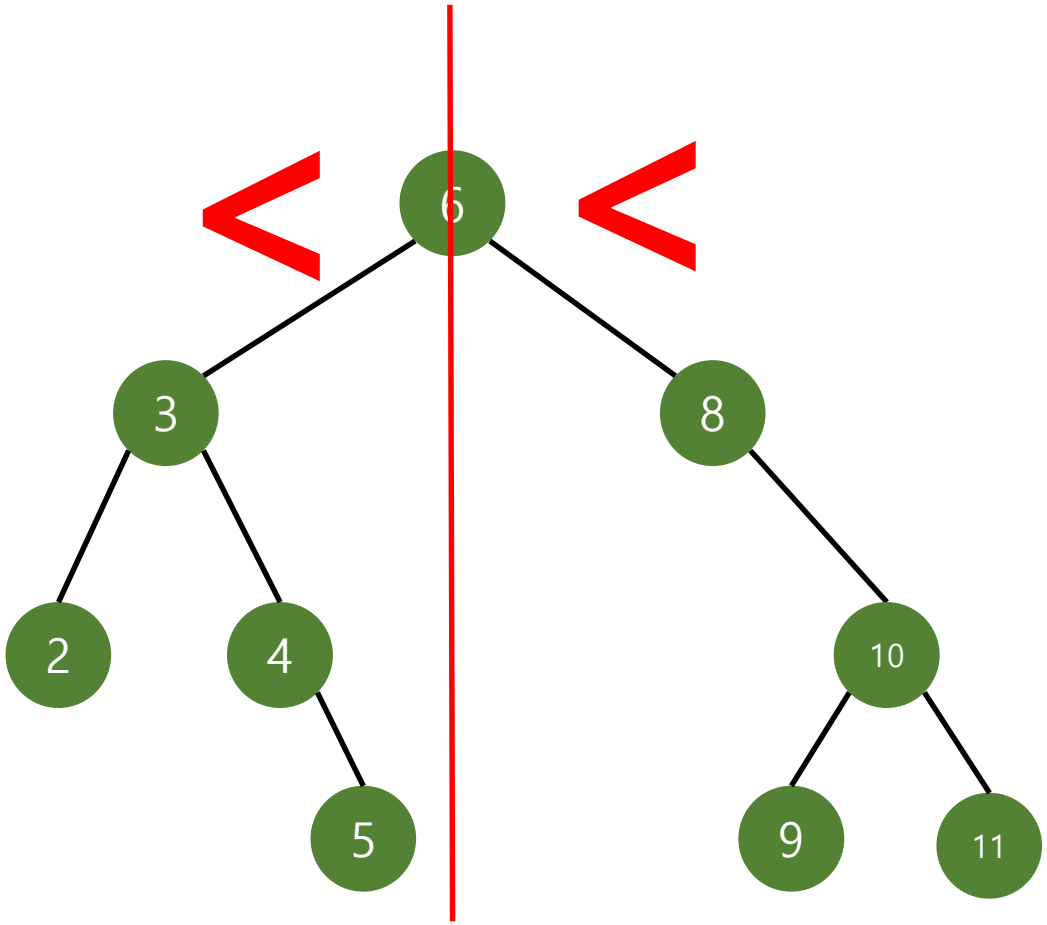

1
1
2
5
2
3
3
4
6
7
4
5
6
7

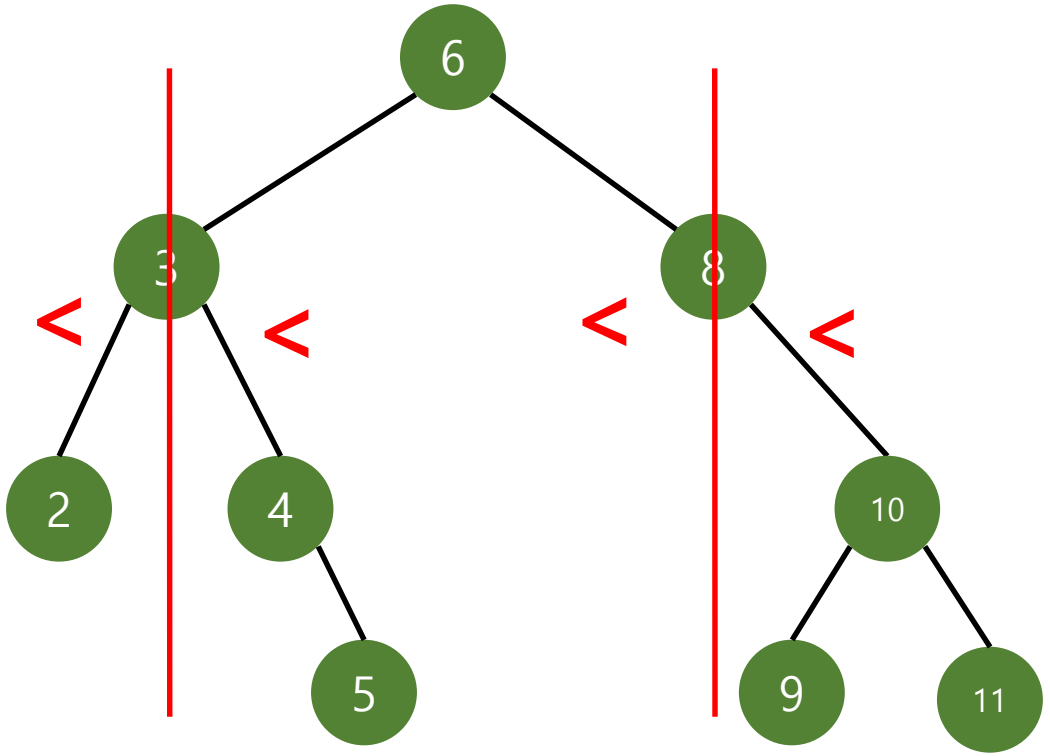

4
1
2
6
2
3
1
3
5
7
4
5
6
7

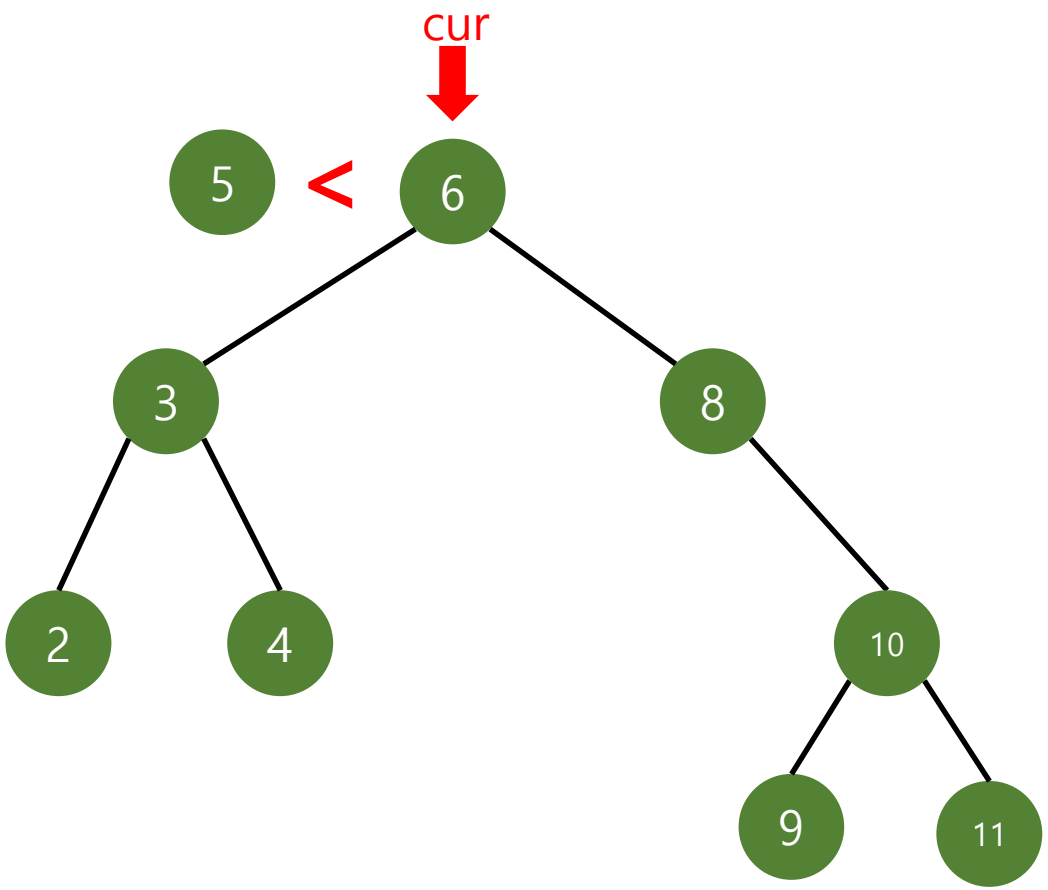

7
1
3
6
2
3
1
2
4
5
4
5
6
7

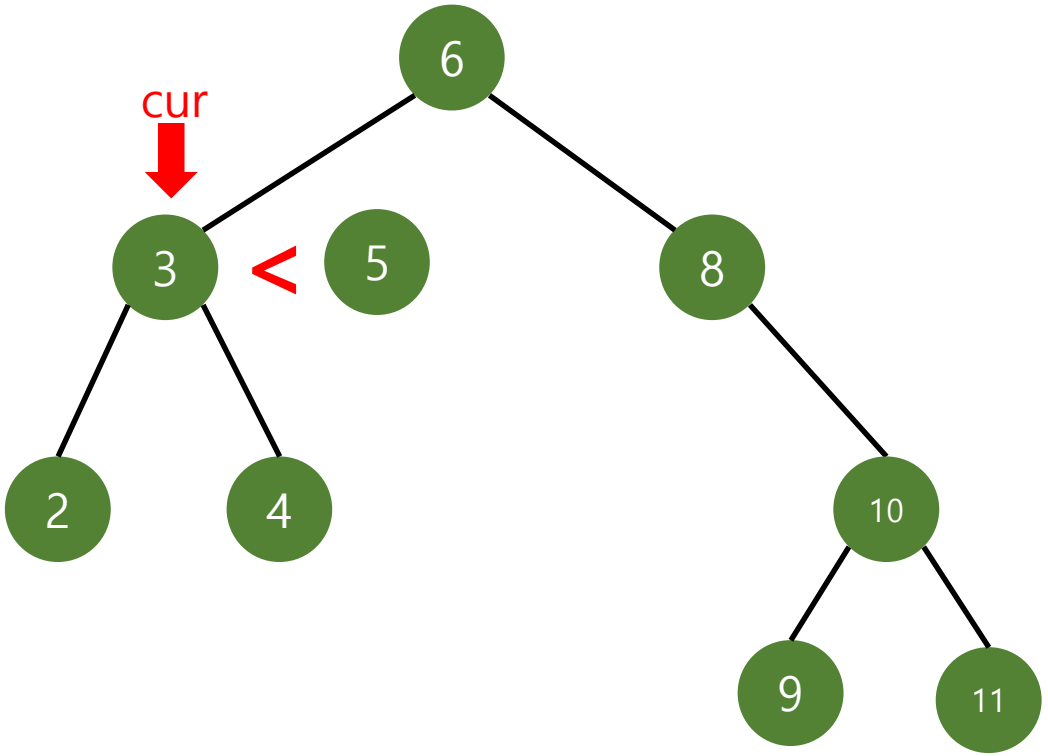
BST

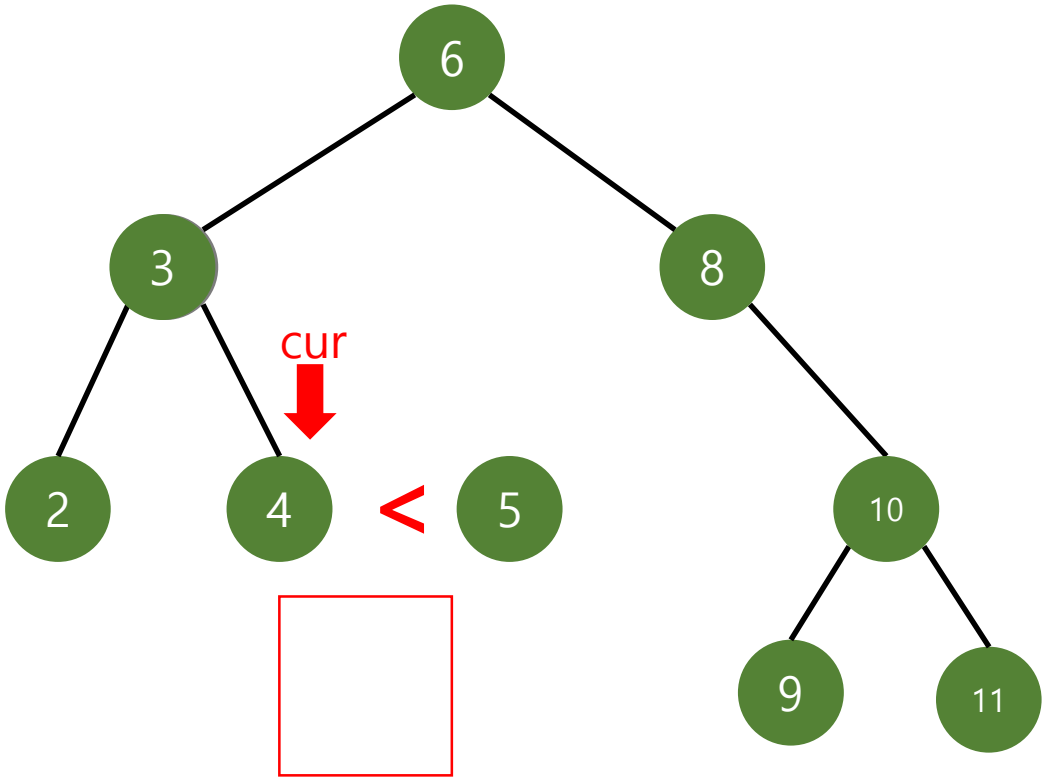
Binary Search Tree

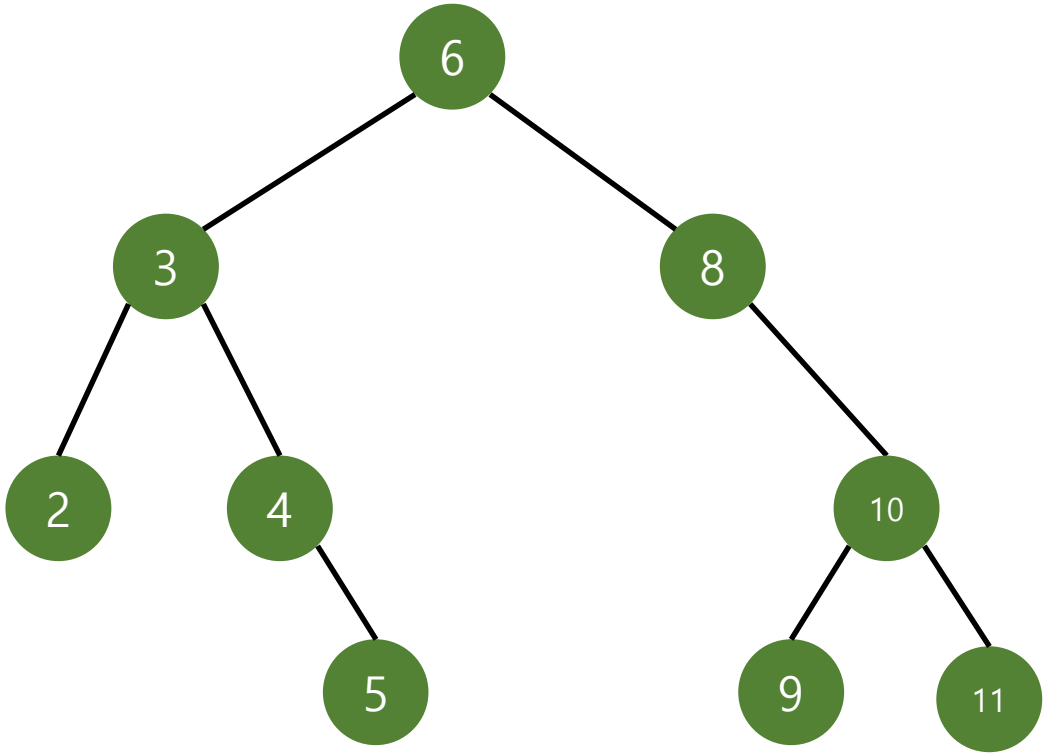

6
<
<
3
8
2
4
10
5
9
11

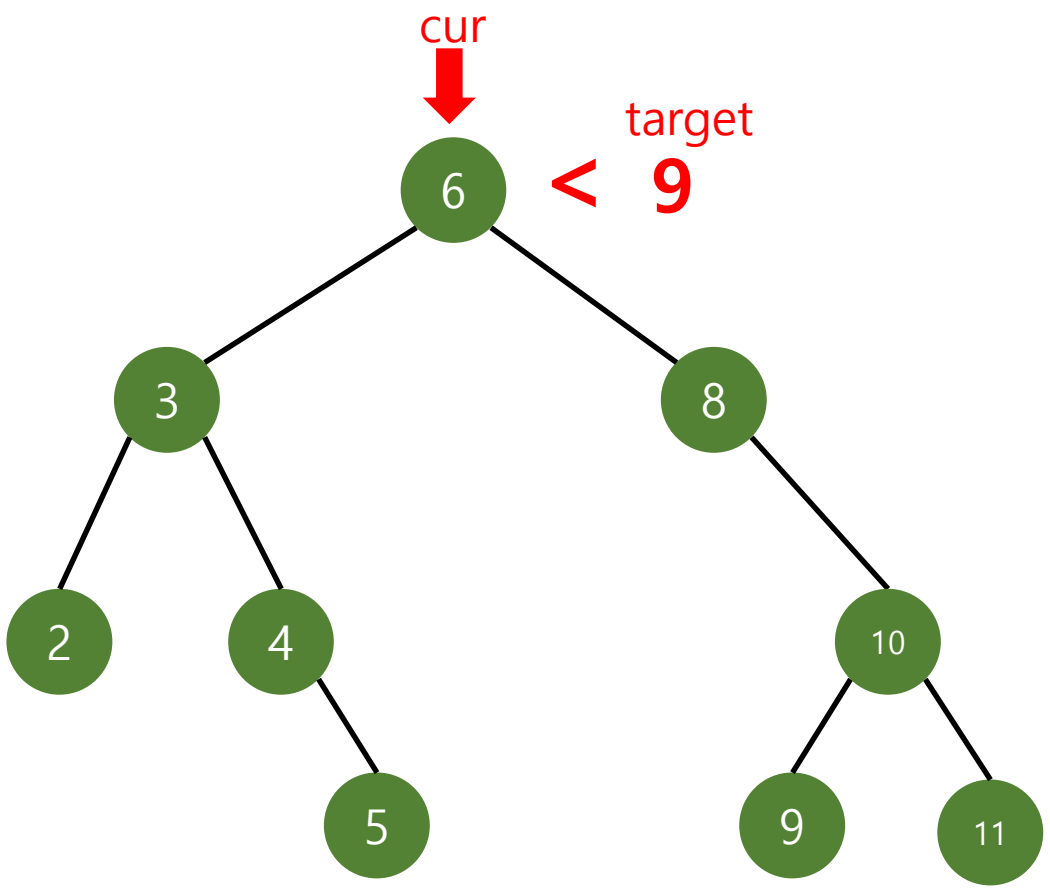

6
3
8
<
<
<
<
2
4
10
5
9
11

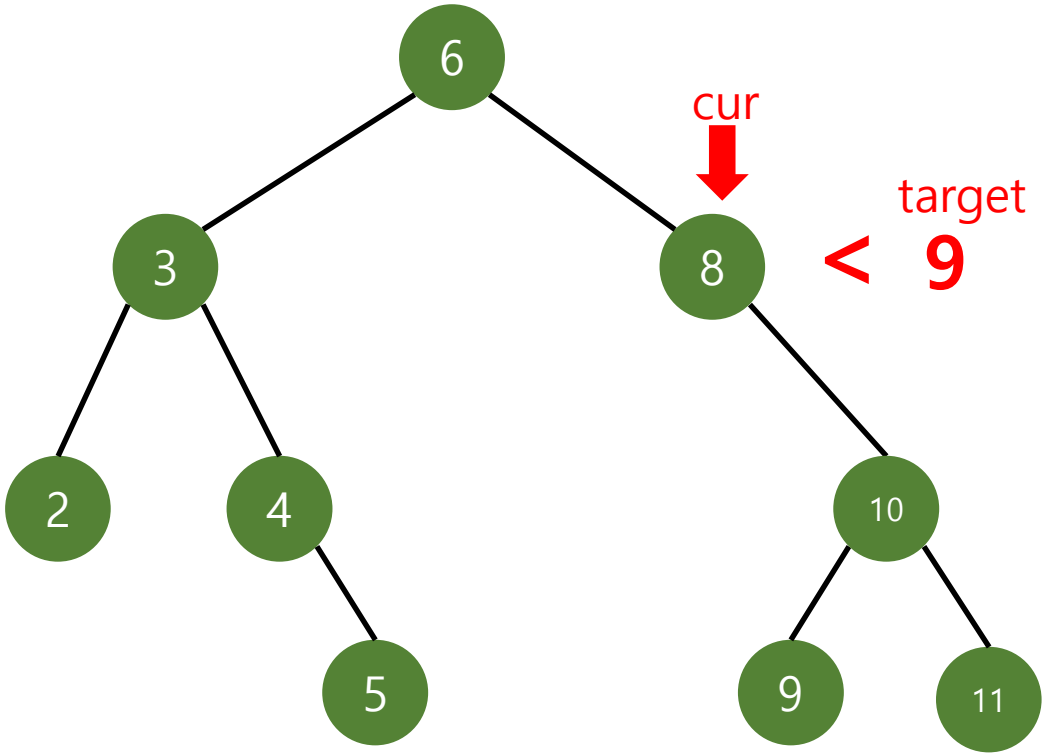

cur
5
<
6
3
8
2
4
10
9
11

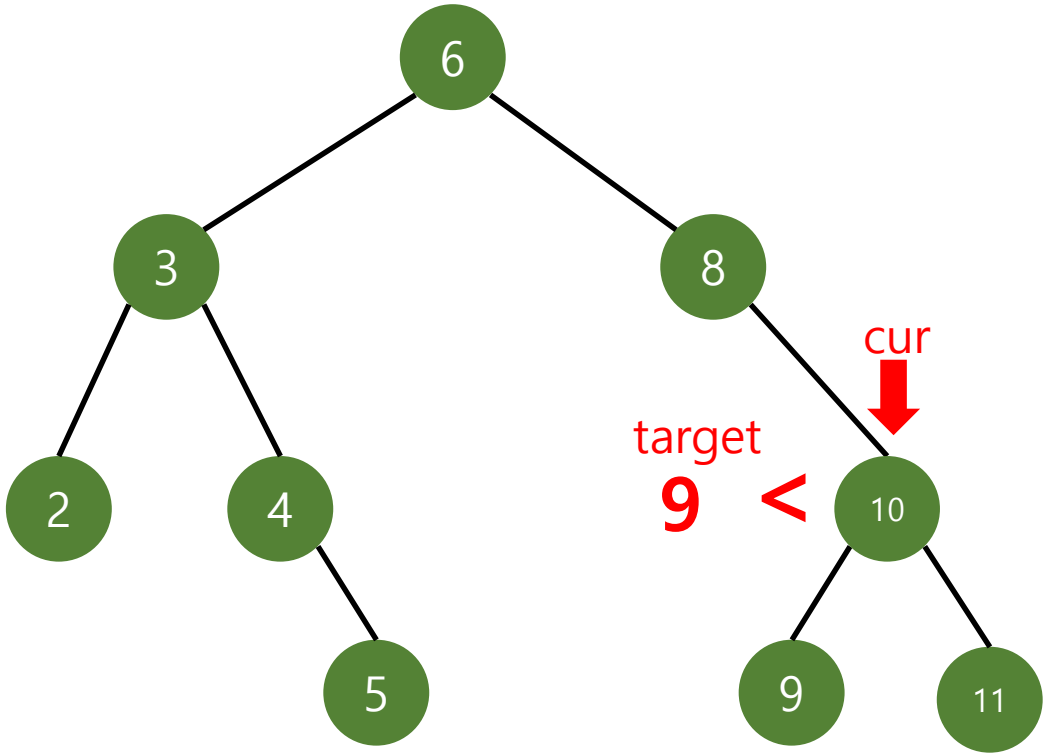

6
cur
3
<
5
8
2
4
10
9
11

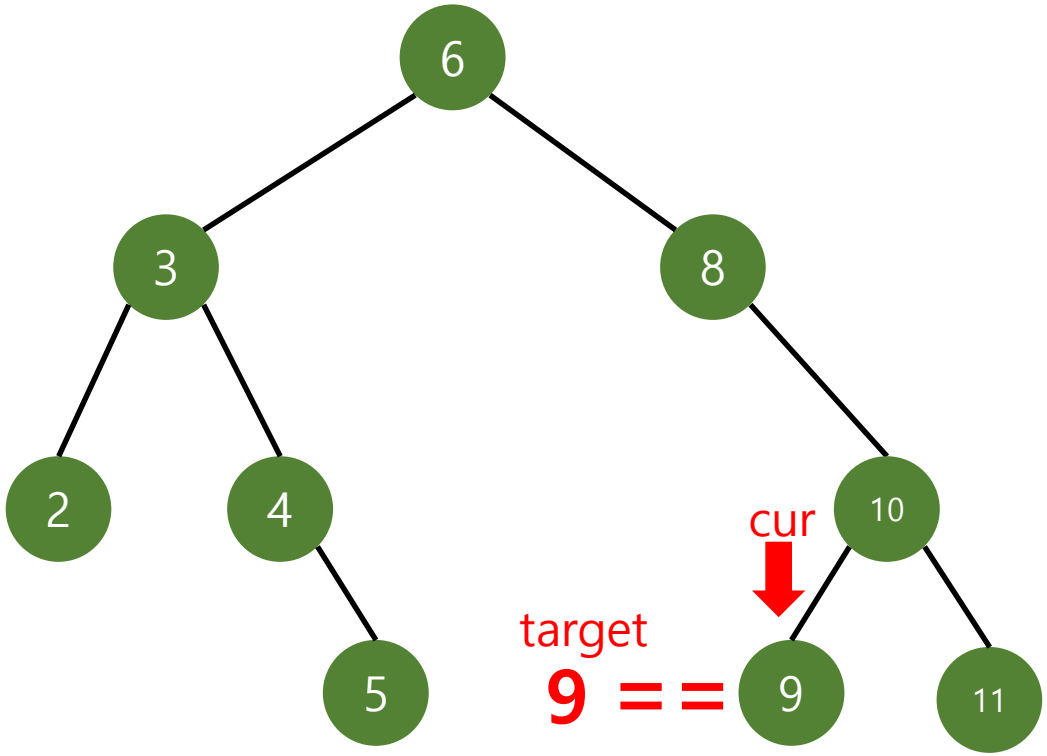

6
3
8
cur
2
4
<
5
10
9
11

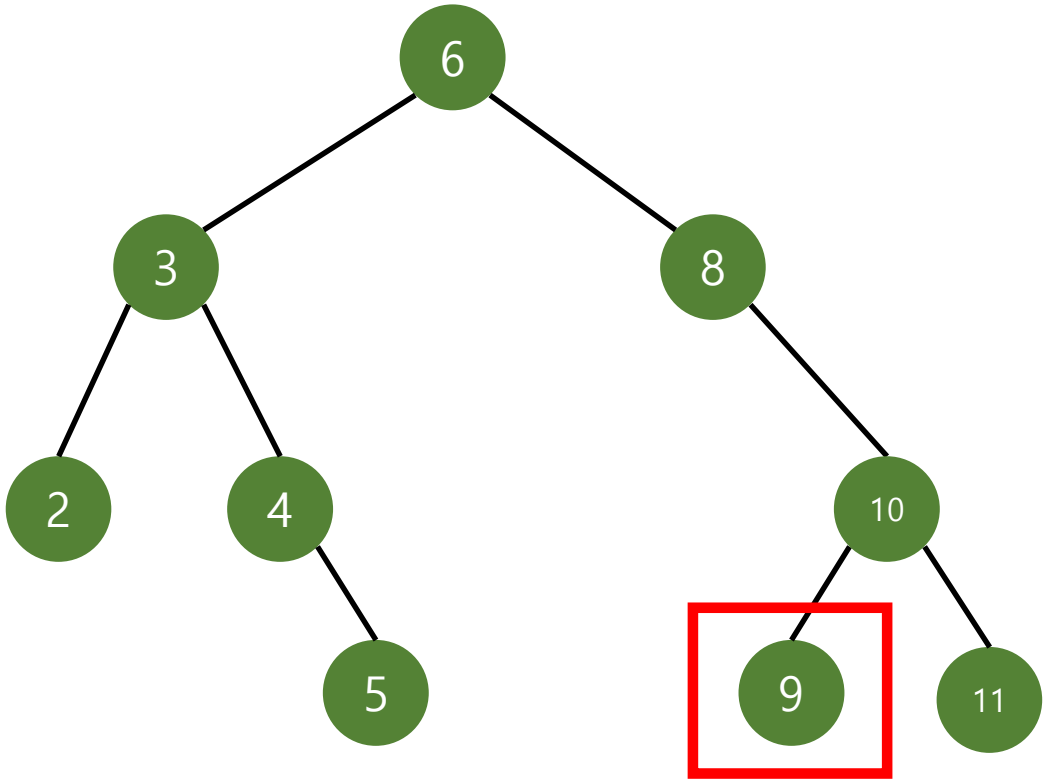

6
3
8
2
4
10
5
9
11

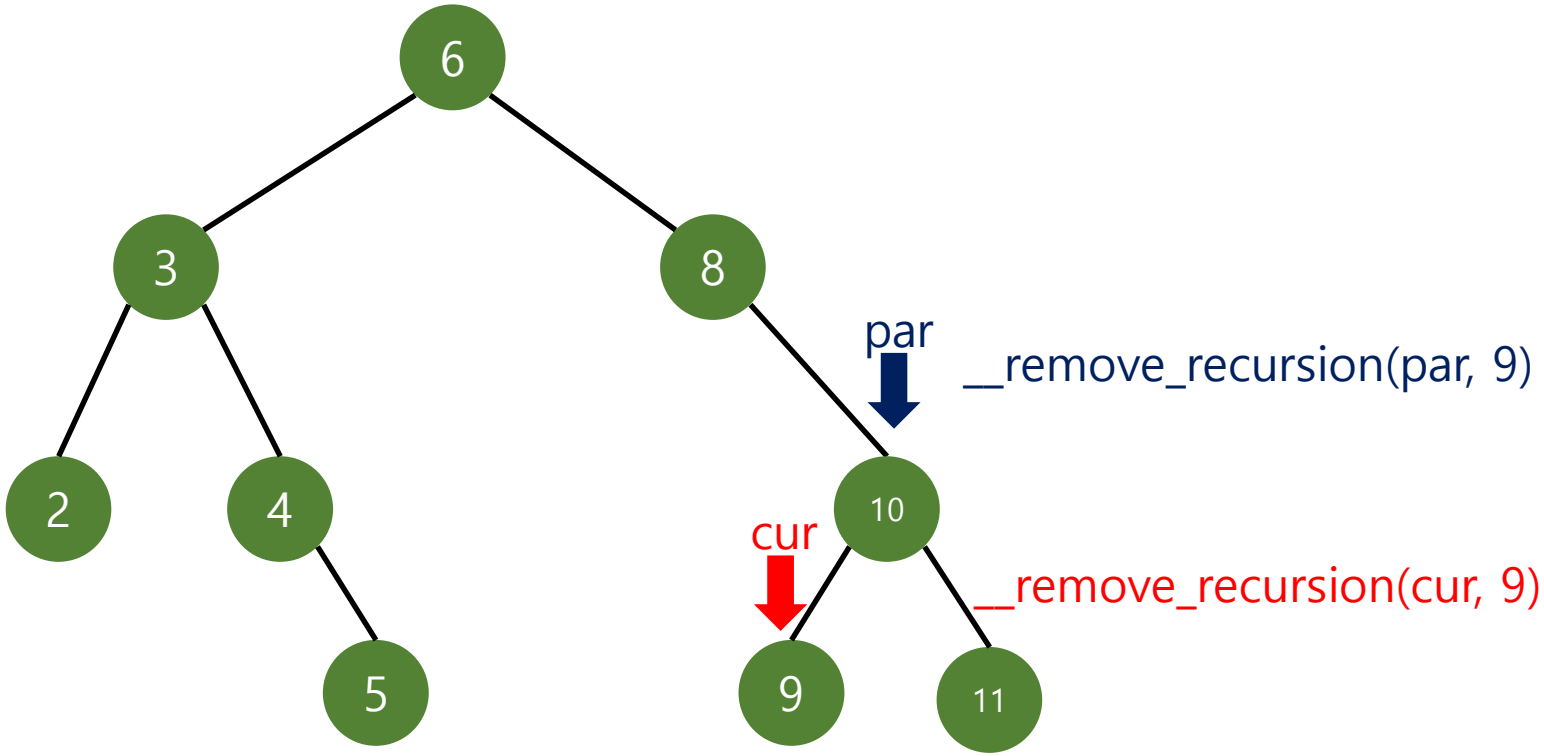

cur
target
6
< 9
3
8
2
4
10
5
9
11


6
cur
target
3
8
< 9
2
4
10
5
9
11

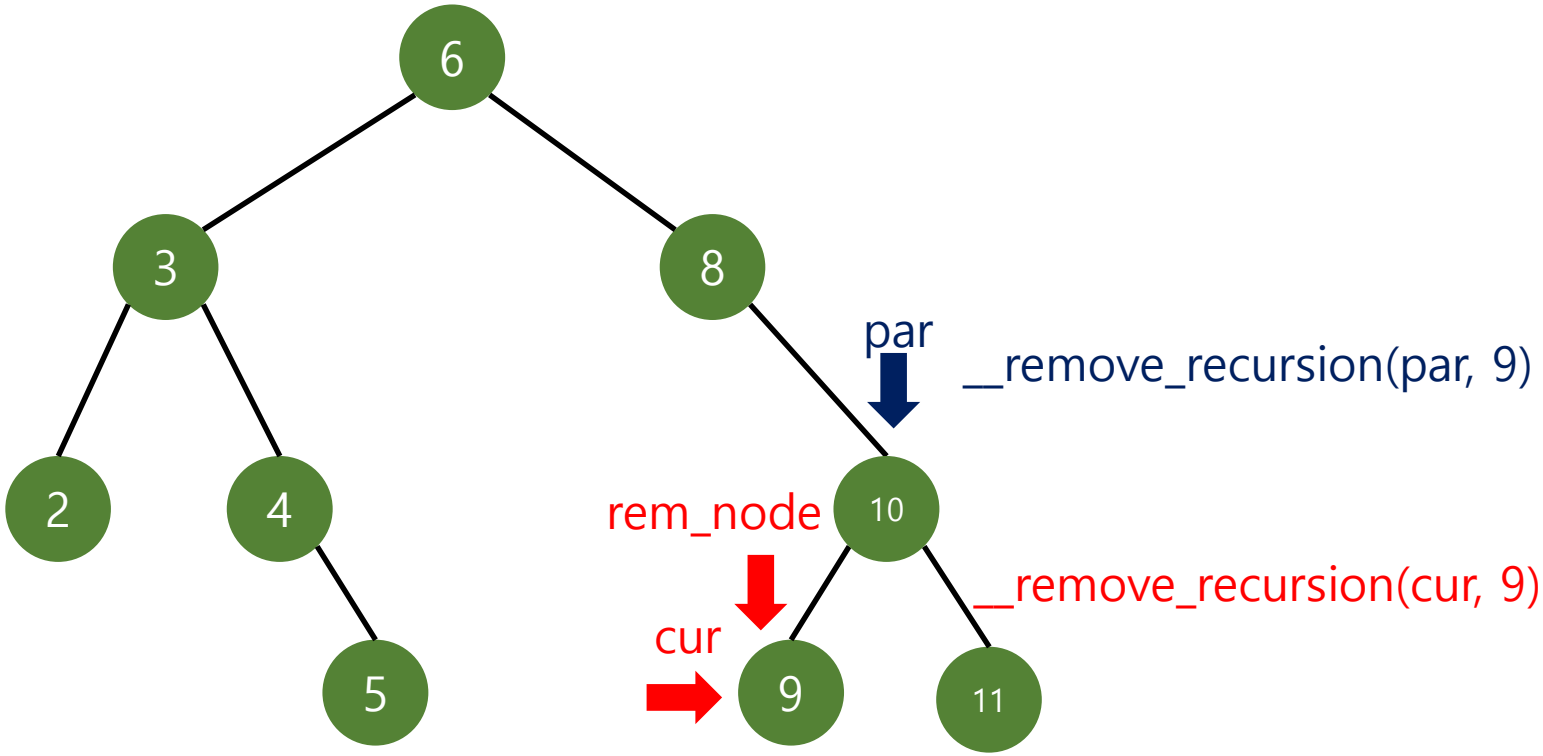

6
3
8
cur
target
9
<
10
2
4
5
9
11


6
3
8
2
4
10
cur
target
9 == 9
5
11

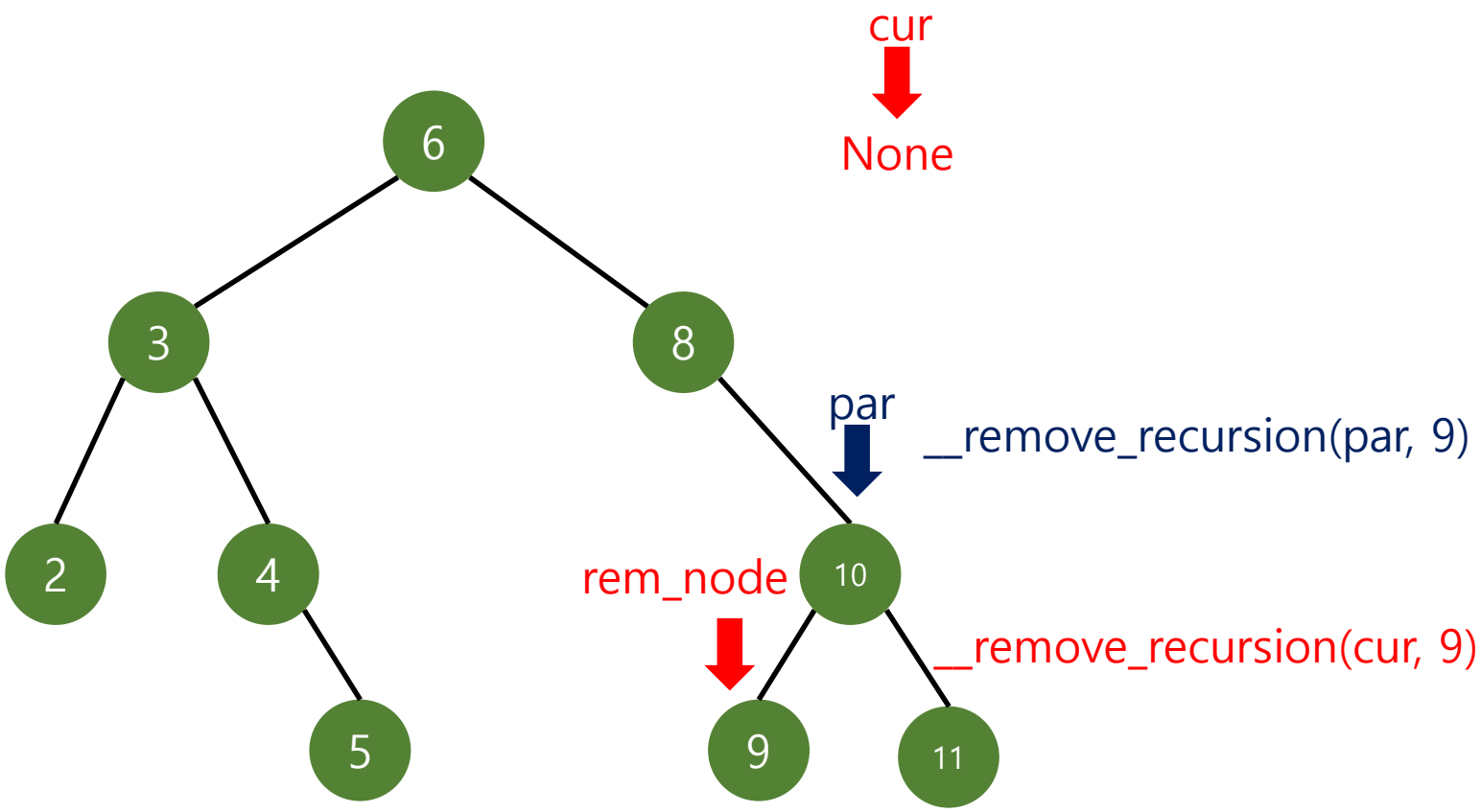

6
3
8
2
4
10
5
9
11


6
3
8
2
4
10
5
9
11
par
__remove_recursion(par, 9)
cur
__remove_recursion(cur, 9)

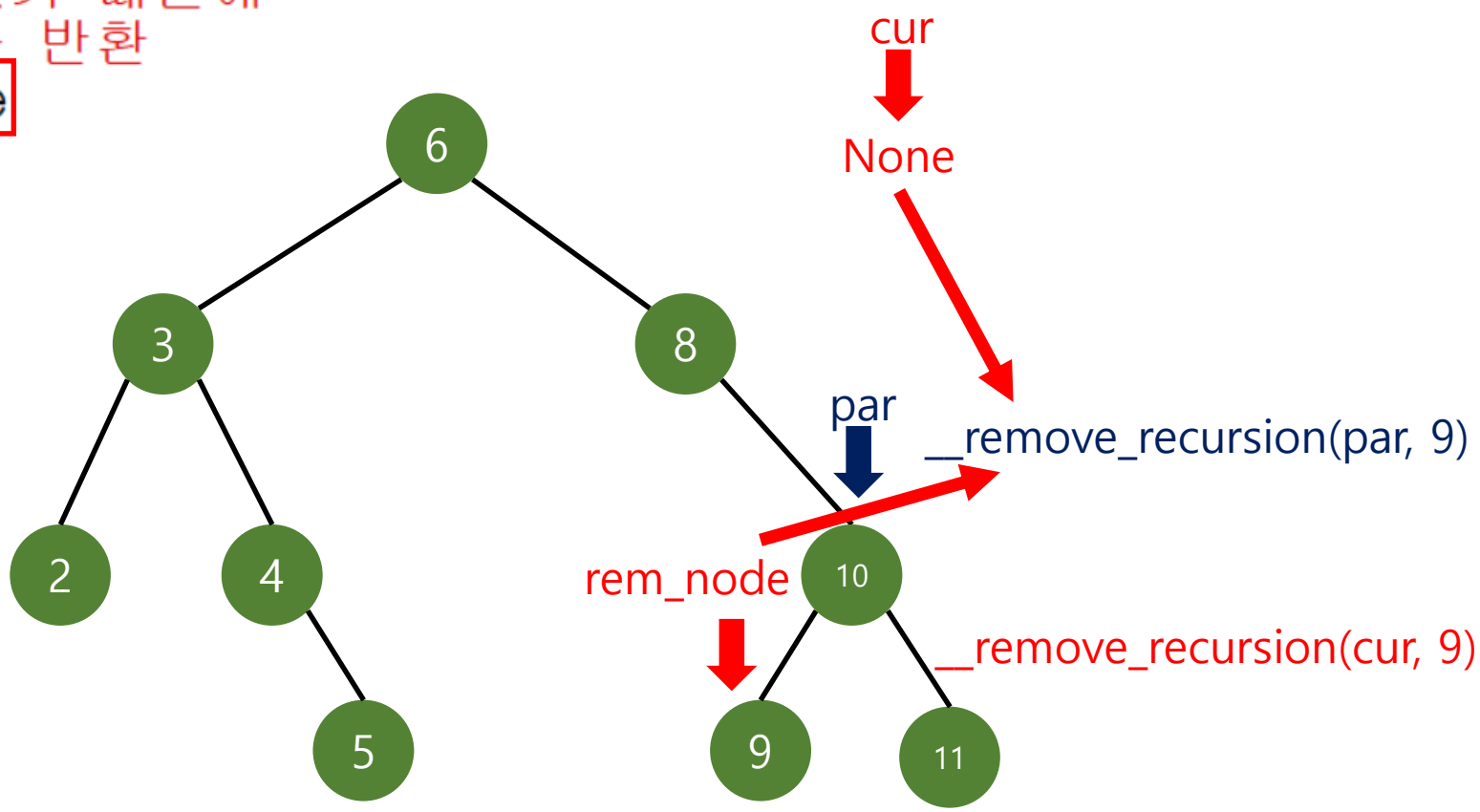
```
else:
    #3. 리프 노드일 때
    if not cur.left and not cur.right:
        rem_node = cur
        cur = None
```



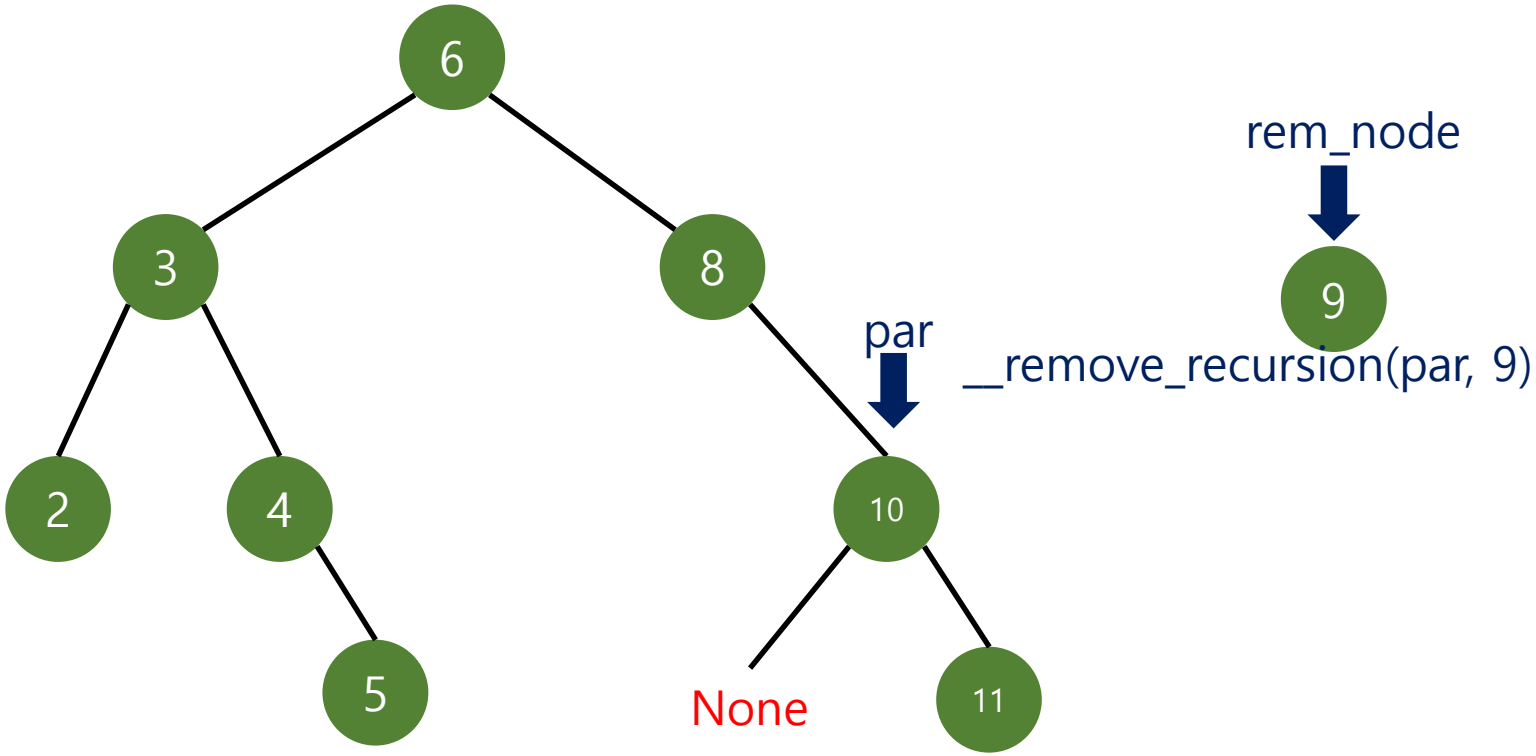
```
else:
    #3. 리프 노드일 때
    if not cur.left and not cur.right:
        rem_node = cur
        cur = None
```

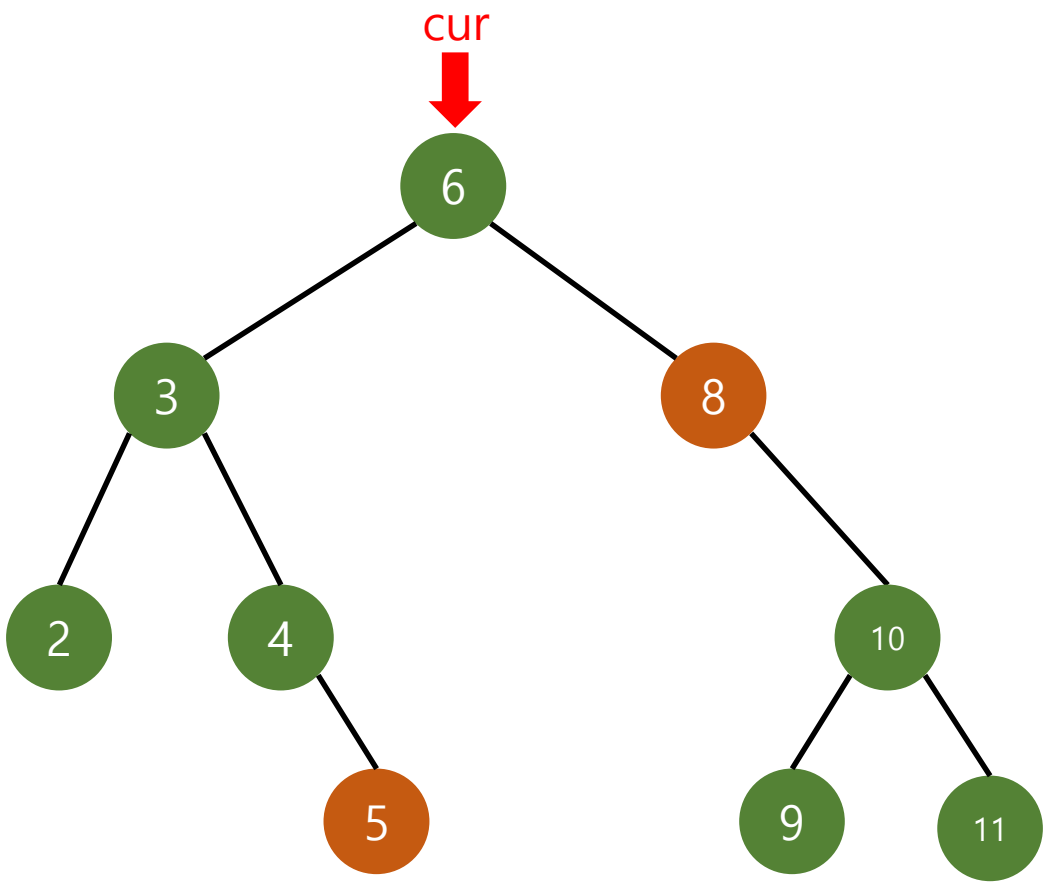


```
#삭제 노드가 루트 노드일 경우
#루트가 변경될 수 있기 때문에
#삭제 후 현재 루트를 반환
return cur, rem_node
```



```
#target 데이터가 노드 데이터보다 작으면
#노드의 왼쪽 자식에서 target을 지운다.(재귀 함수 호출)
elif target < cur.data:
    cur.left, rem_node = W
        self.remove_recursion(cur.left, target)#1
```




cur
6
3
8
2
4
10
5
9
11