TREE ADT

BST

AVL

TREE ADT

MOTIVATIONS

Lists - Linear
Trees - Logarithmic

File Systems

Arithmetic Expressions

Compiler Designs

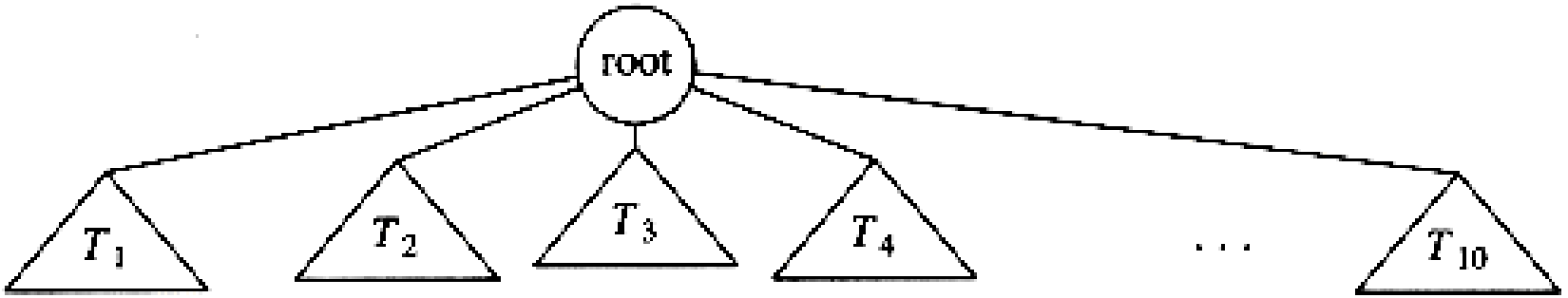
TREE

A connected graph with no cycles.

TREE

A tree consists of a distinguished node r (the root), and zero or more sub trees, T_1, T_2 …, T_k each of whose roots are connected by a directed edge to r.

TREES



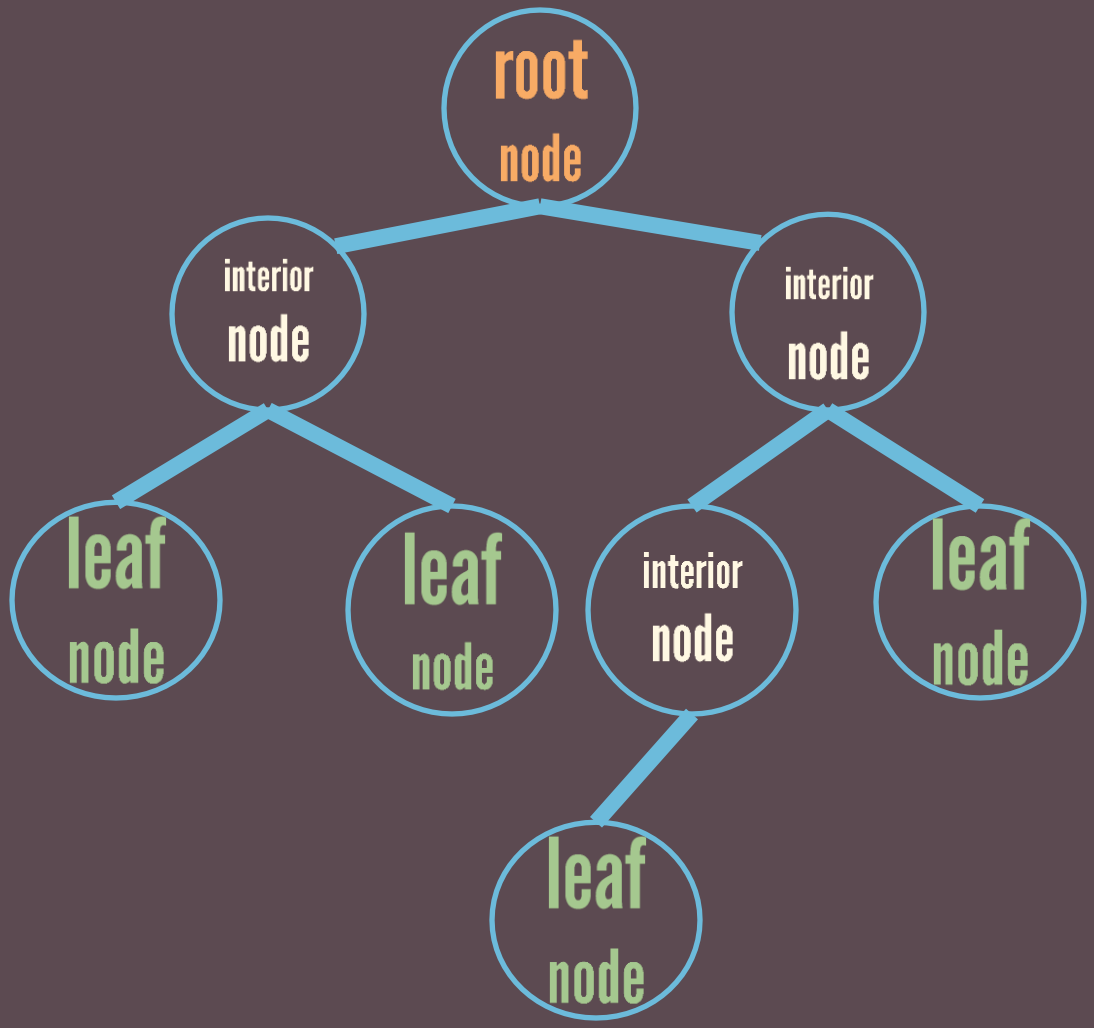
TREE THEOREMS

Any two vertices are connected by a unique path.

TREE
THEOREMS

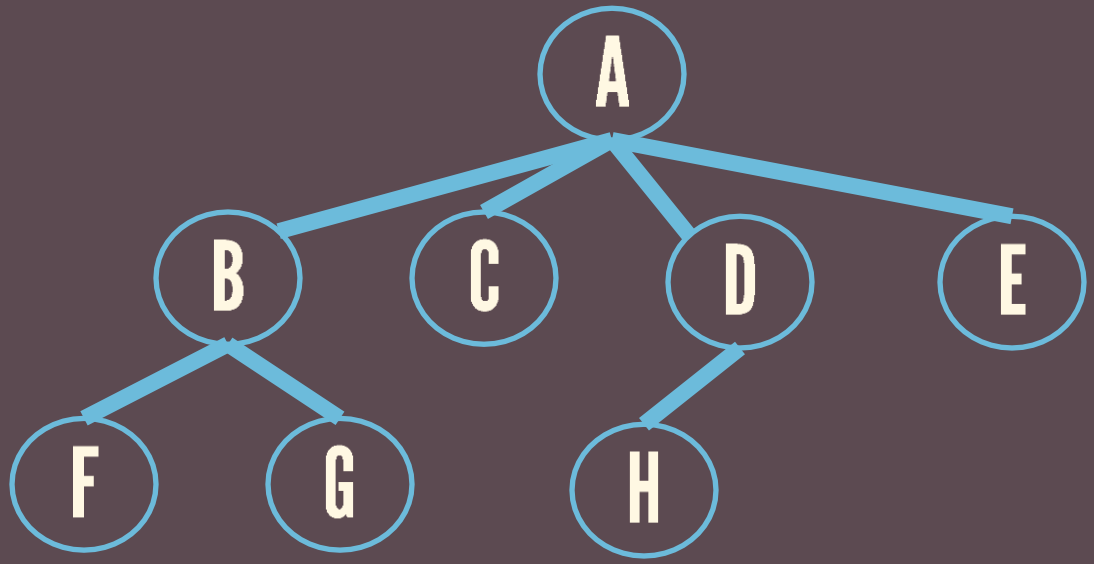
The number of edges in a tree is |V(G)| - 1

ROOT INTERIOR NODE LEAF NODE



SIBLINGS PARENT CHILD GRANDCHILD

ANCESTORS DESCENDANTS



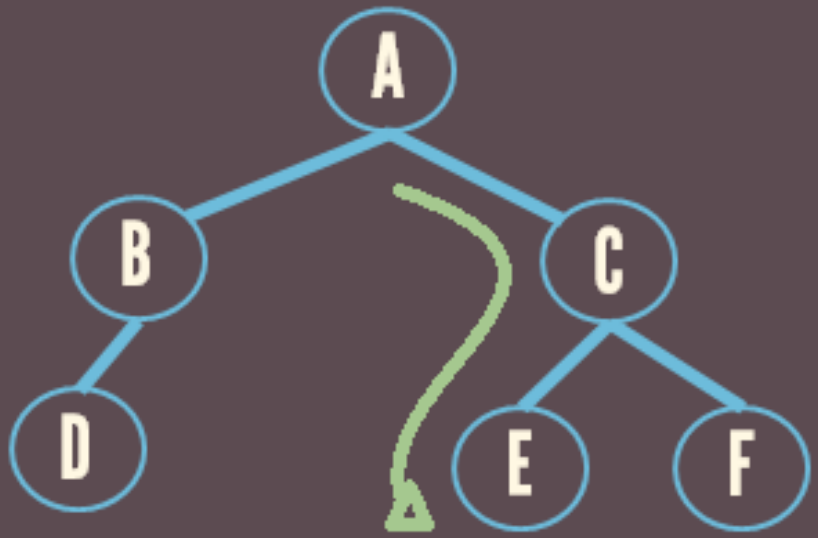
PATH

from node n_1 to n_k

Sequence of nodes n_1, n_2, … n_k such that n_i is the parent of n_{i+1}

$1 \leq i \leq k$

PATH



A C E

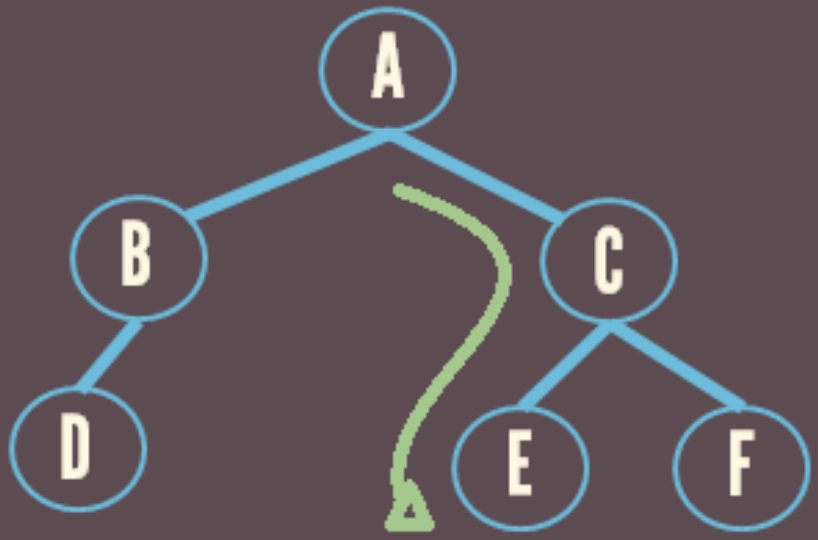
length of the

PATH

The number of edges on the path.

length of the

PATH



A C E : 2

HEIGHT

of node n_i

The height of node n_i is the longest path from n_i to a leaf.

Height of A = 2

Height of B = 0

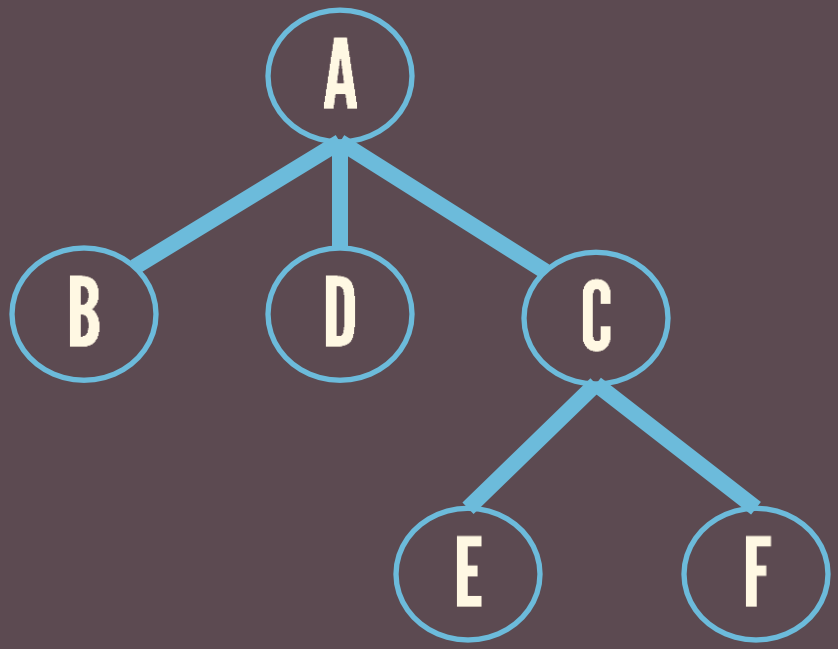
Height of D = 0

Height of C = 1

Height of E = 0

Height of F = 0

Height of the tree = 2



DEPTH

of node n_i

The length of the unique path from the root to n_i

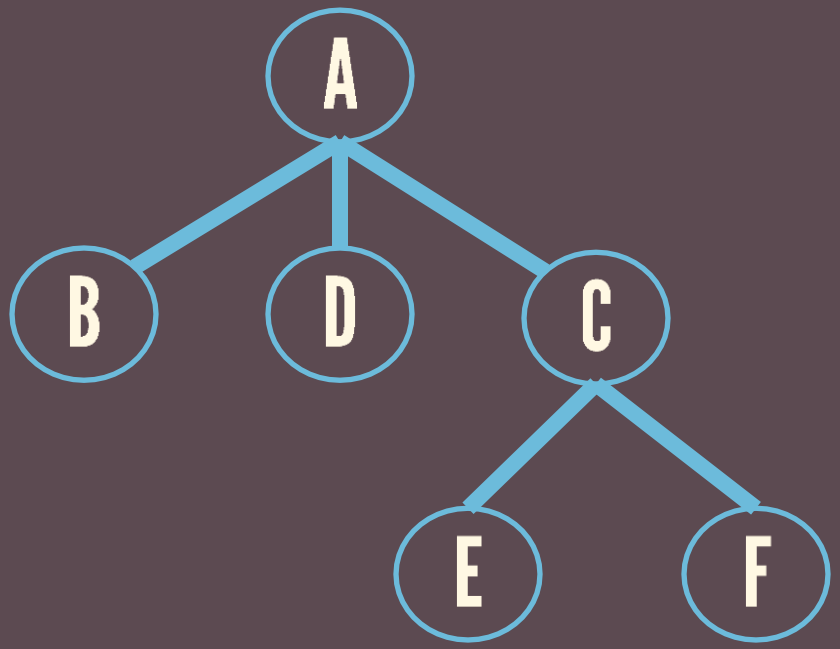
DEPTH

LEVEL

0

1

2



0

1

2

IMPLEMENTATIONS

LINKED REPRESENTATION

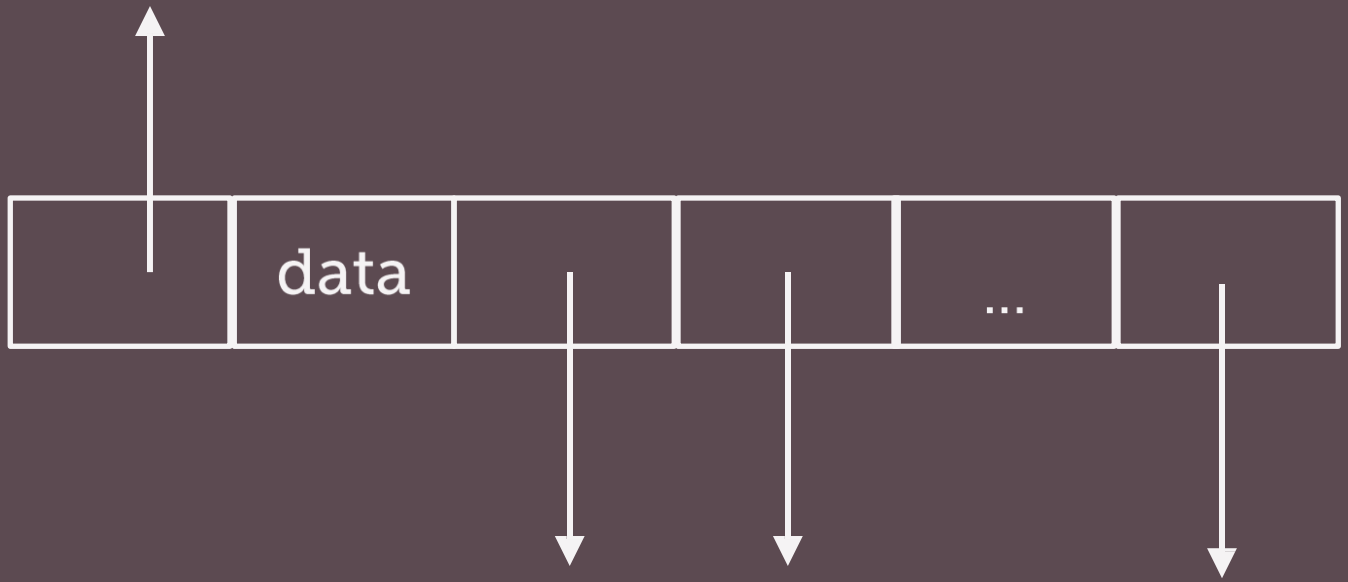
FIRST CHILD, NEXT SIBLING REPRESENTATION

LINKED REPRESENTATION

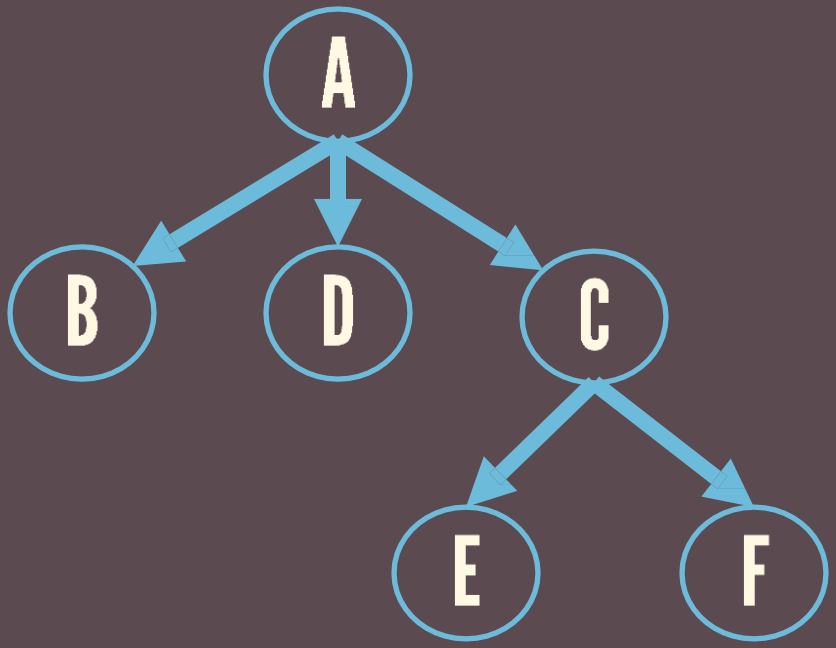
Each node besides its data has a pointer to each child of the node.

```
class TreeNode:
  def __init__(self, data):
    self.data = data
    self.parent = None
    self.child1 = None
    self.child2 = None
    ...
    self.childk = None
```

LINKED REPRESENTATION



LINKED REPRESENTATION



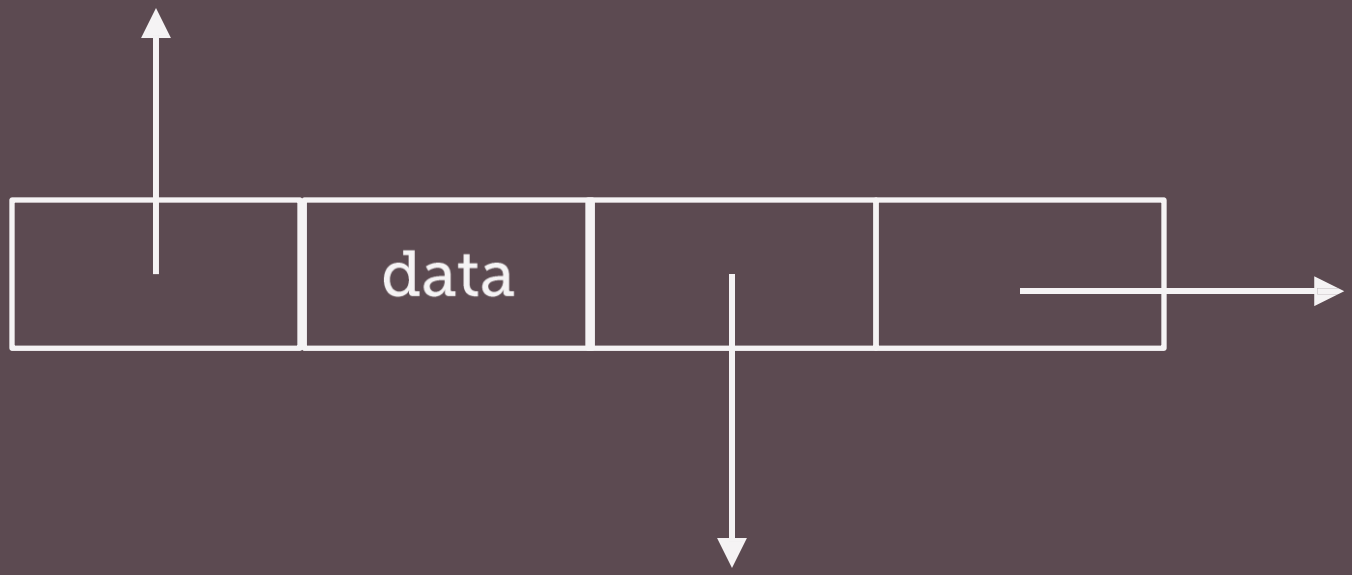
FIRST CHILD, NEXT SIBLING REPRESENTATION

Keep the children of each node in a linked list of tree nodes.

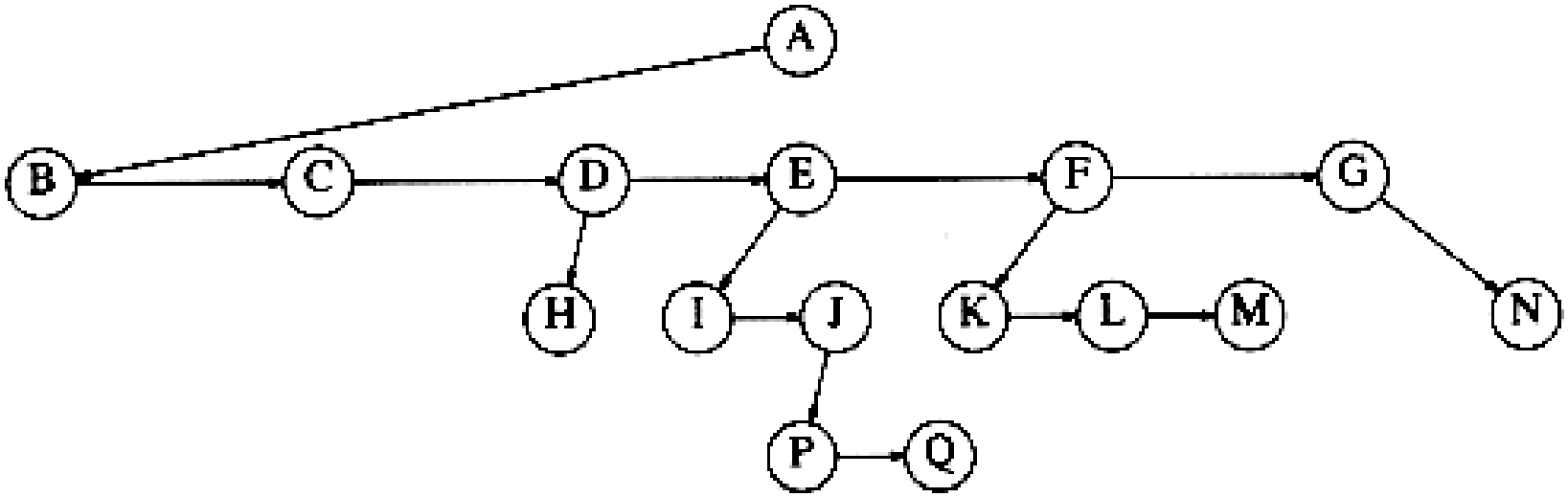
```
class TreeNode:
  def __init__(self, data):
    self.data = data
    self.parent = None
    self.firstChild = None
    self.nextChild = None
```

FIRST CHILD, NEXT SIBLING

REPRESENTATION



FIRST CHILD, NEXT SIBLING REPRESENTATION



BINARY TREES

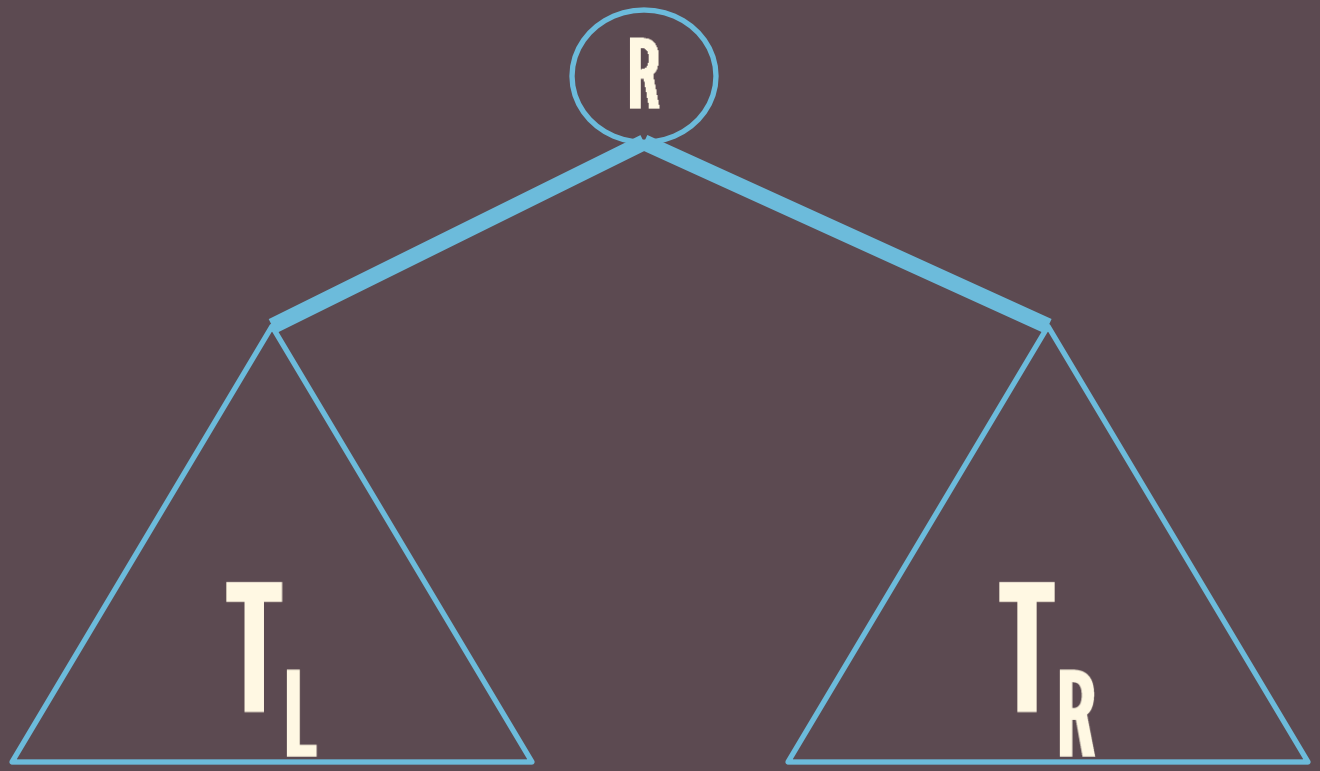
BINARY TREE

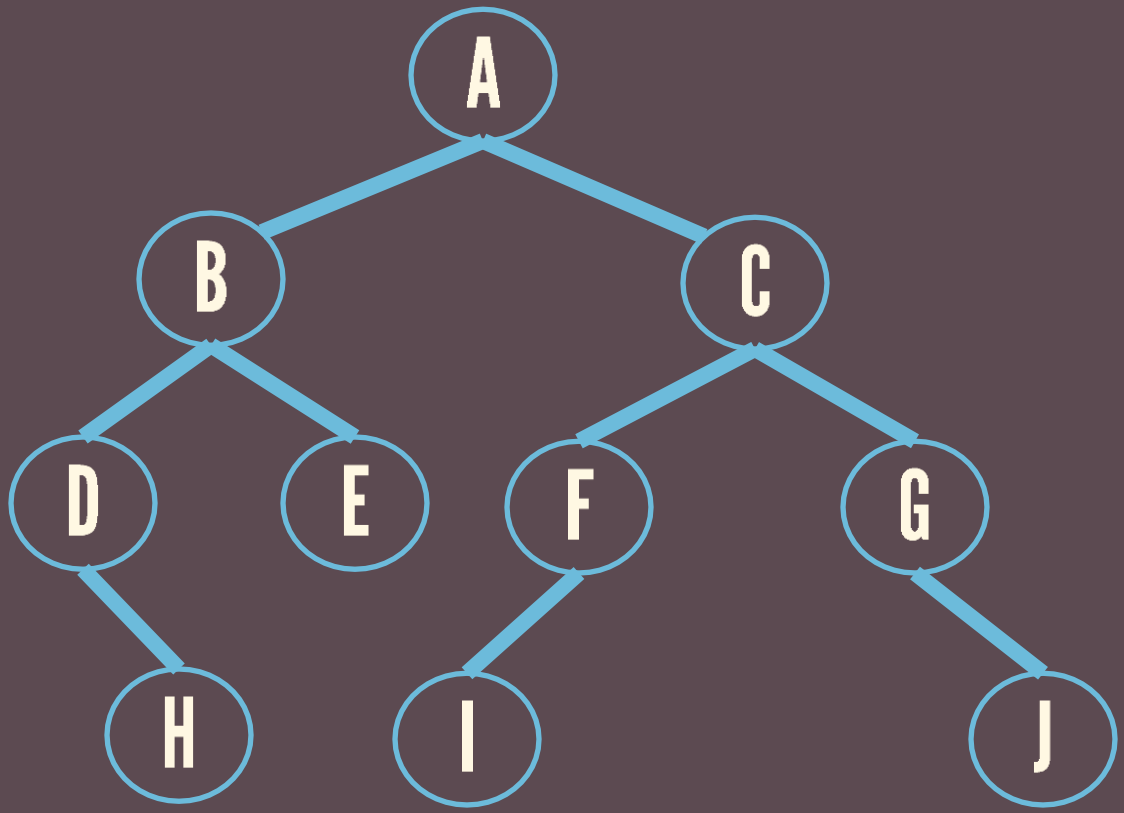
A tree in which no node can have more than two children.

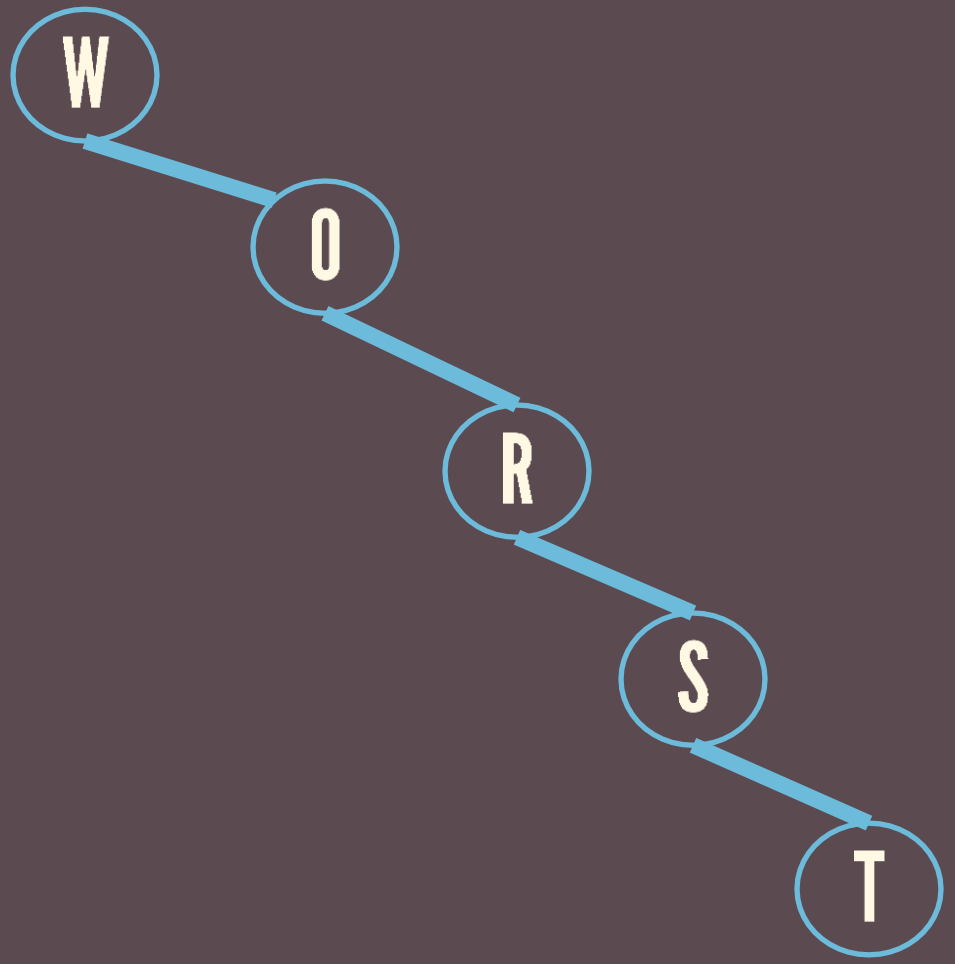
BINARY TREE

A tree where each node has either

- no children
- a left child
- a right child
- both left and right child


R
T_L
T_R

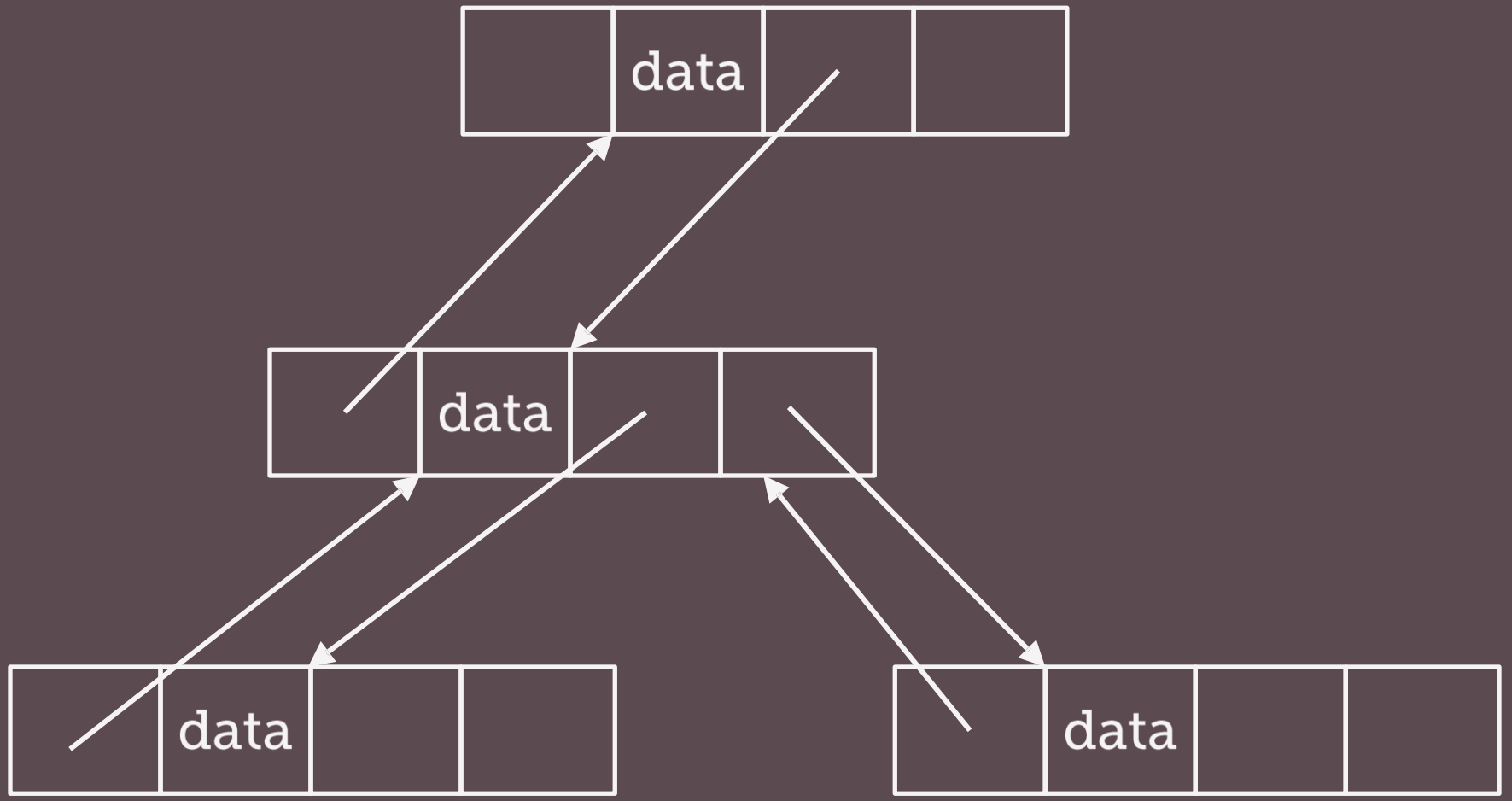

A
B
C
D
E
F
G
H
I
J


W
O
R
S
T

IMPLEMENTATION

LINKED REPRESENTATION

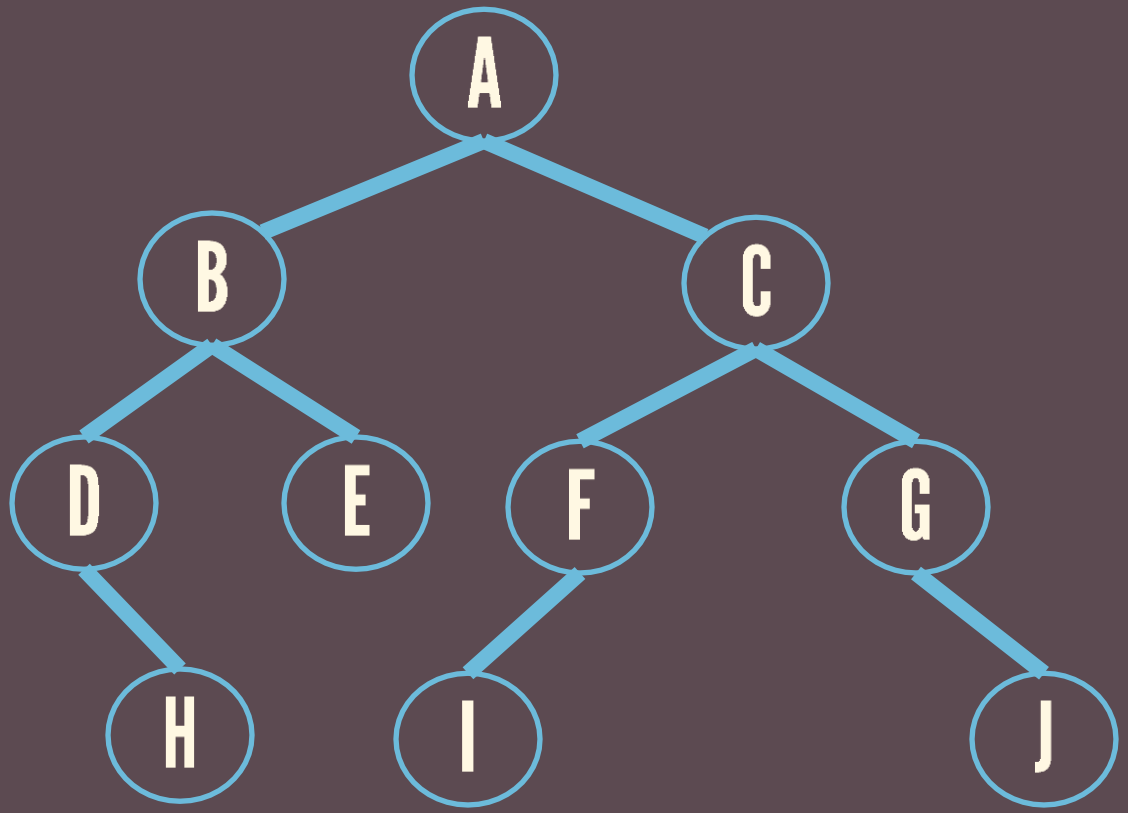
```
class TreeNode:
  def __init__(self, data):
    self.data = data
    self.parent = None
    self.left = None
    self.right = None
```


data
data
data
data

full

LEVEL

Level i is full if there are exactly 2^i nodes at this level.


A
B
C
D
E
F
G
H
I
J

SEARCH TREE ADT

BST

AVL

BST

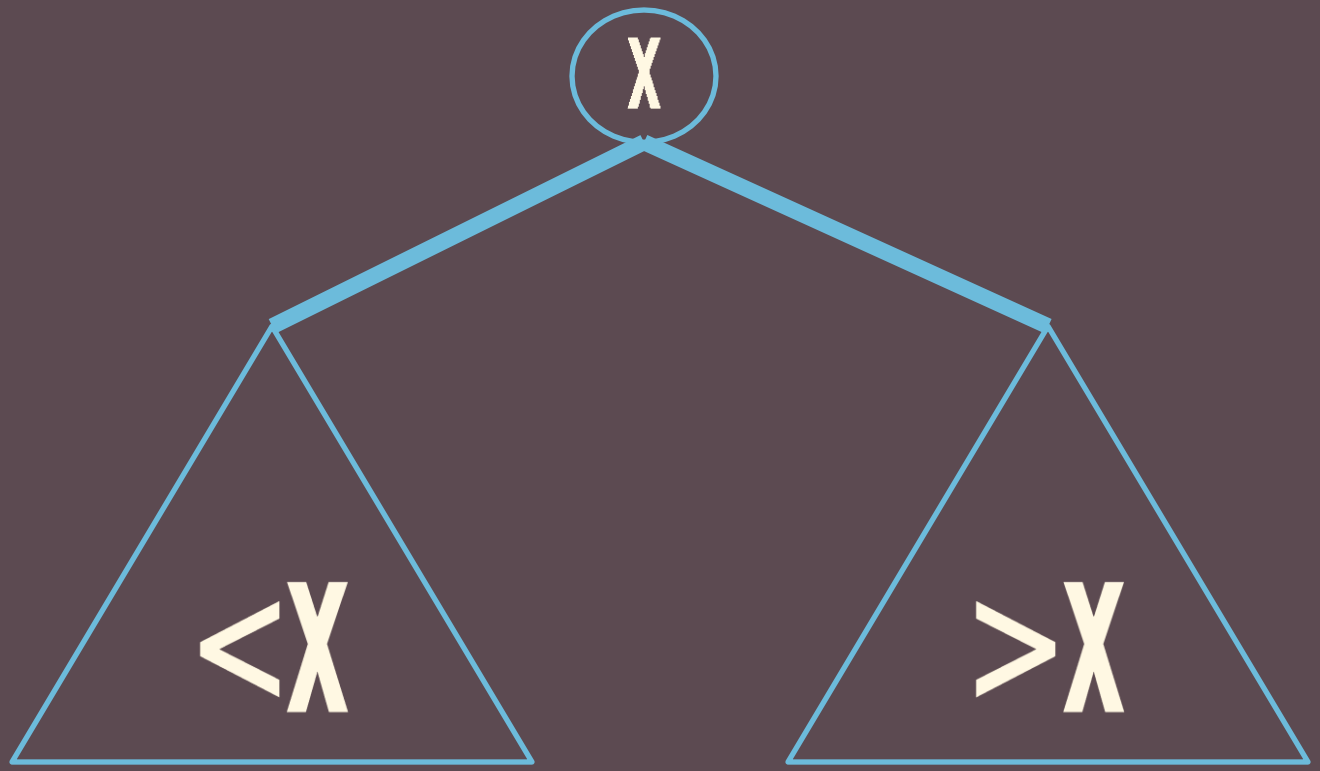
BINARY SEARCH TREE

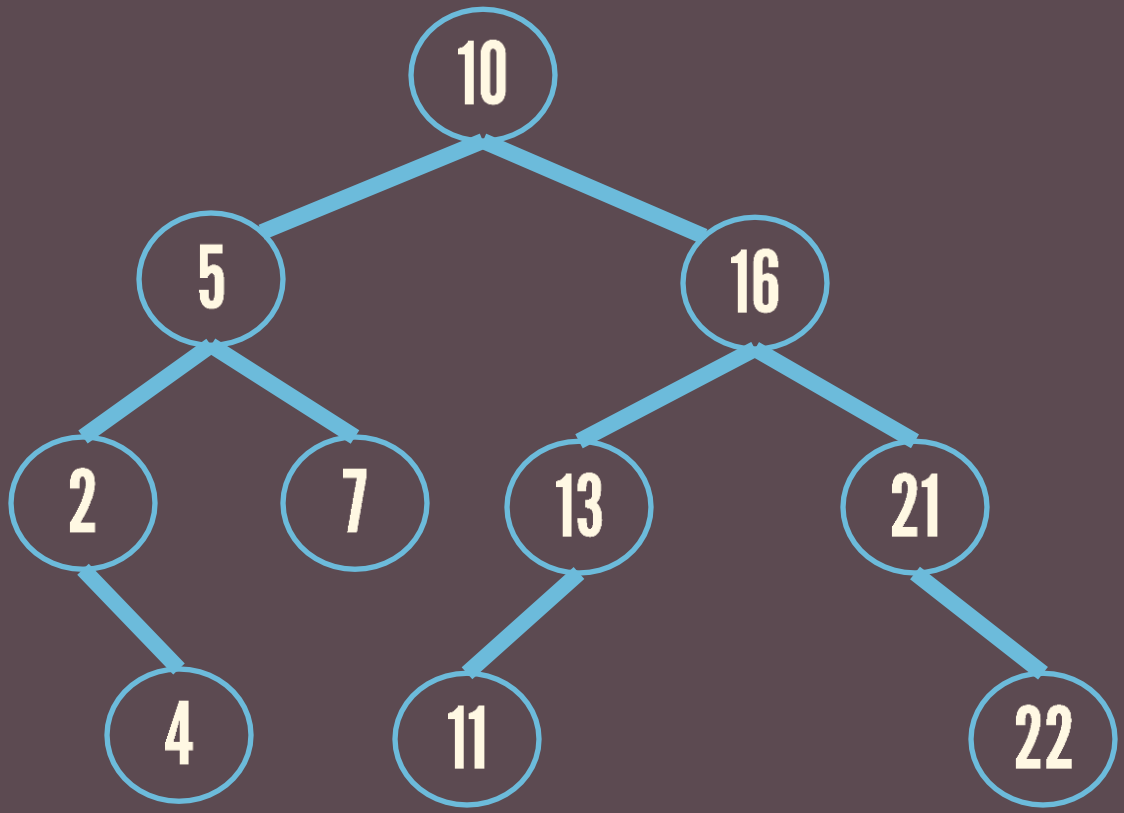
BST

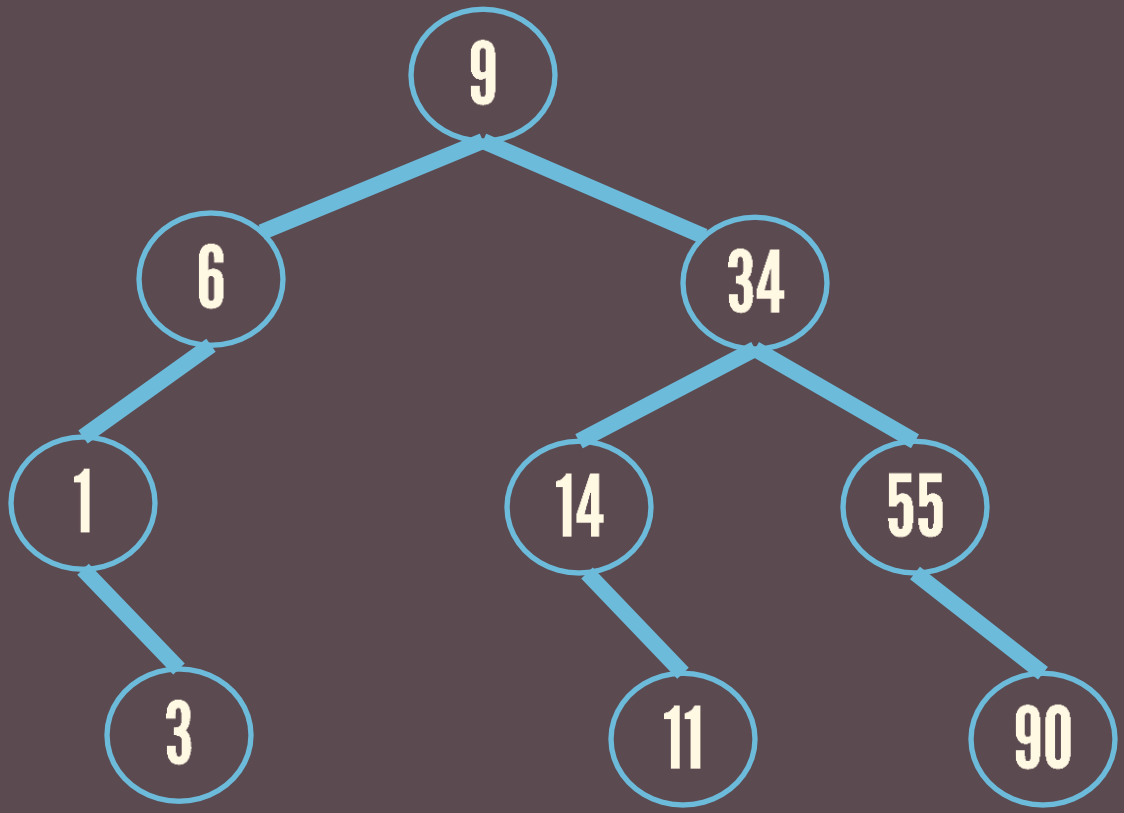
For every node, X in the tree,

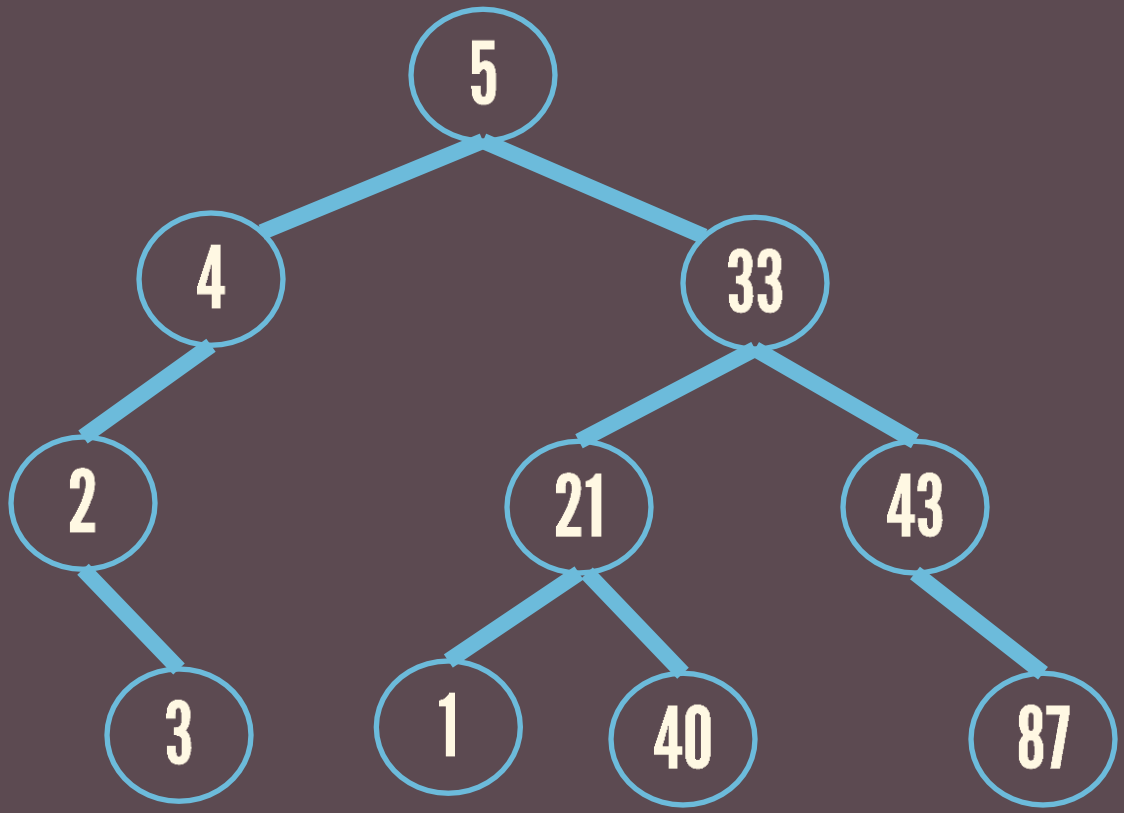
the values of all the keys in the left subtree are less than the key value in X; and

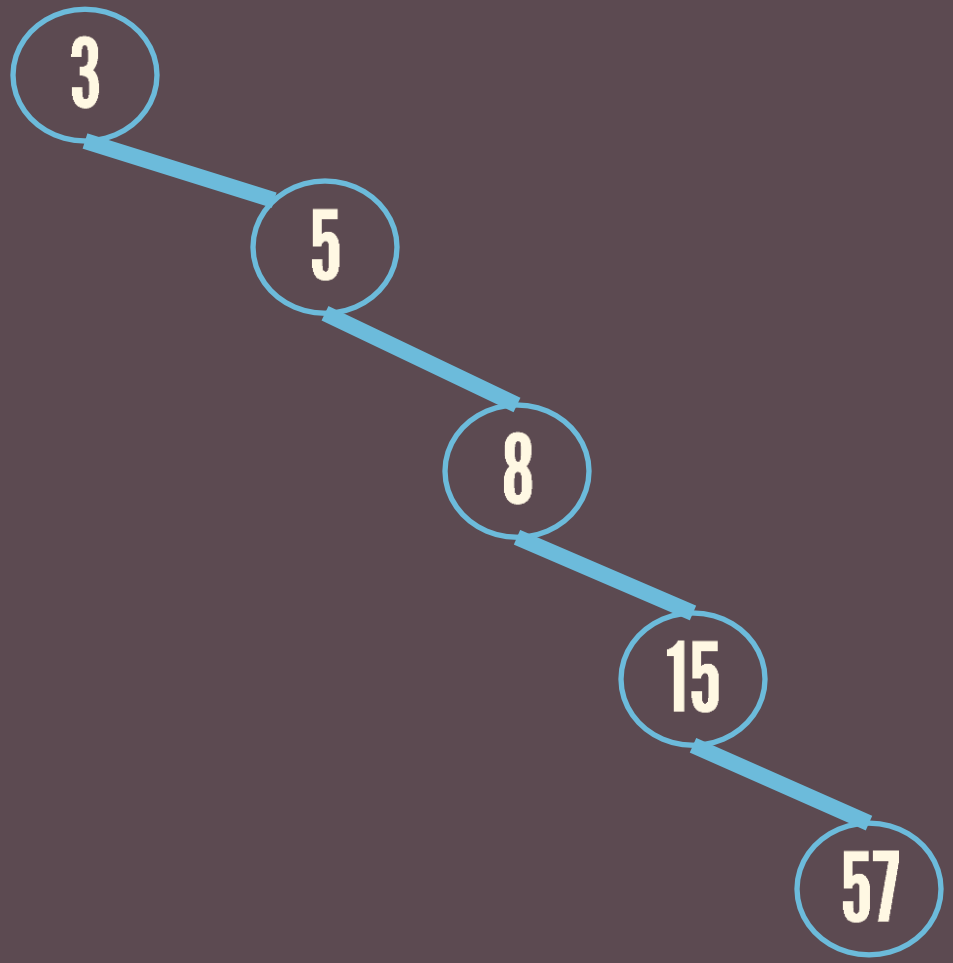
the values of all the keys in the right subtree are larger than the key value in X.


X
<X
>X


10
5
16
2
7
13
21
4
11
22


9
6
34
1
14
55
3
11
90


5
4
33
2
21
43
3
1
40
87


3
5
8
15
57

BST

OPERATIONS

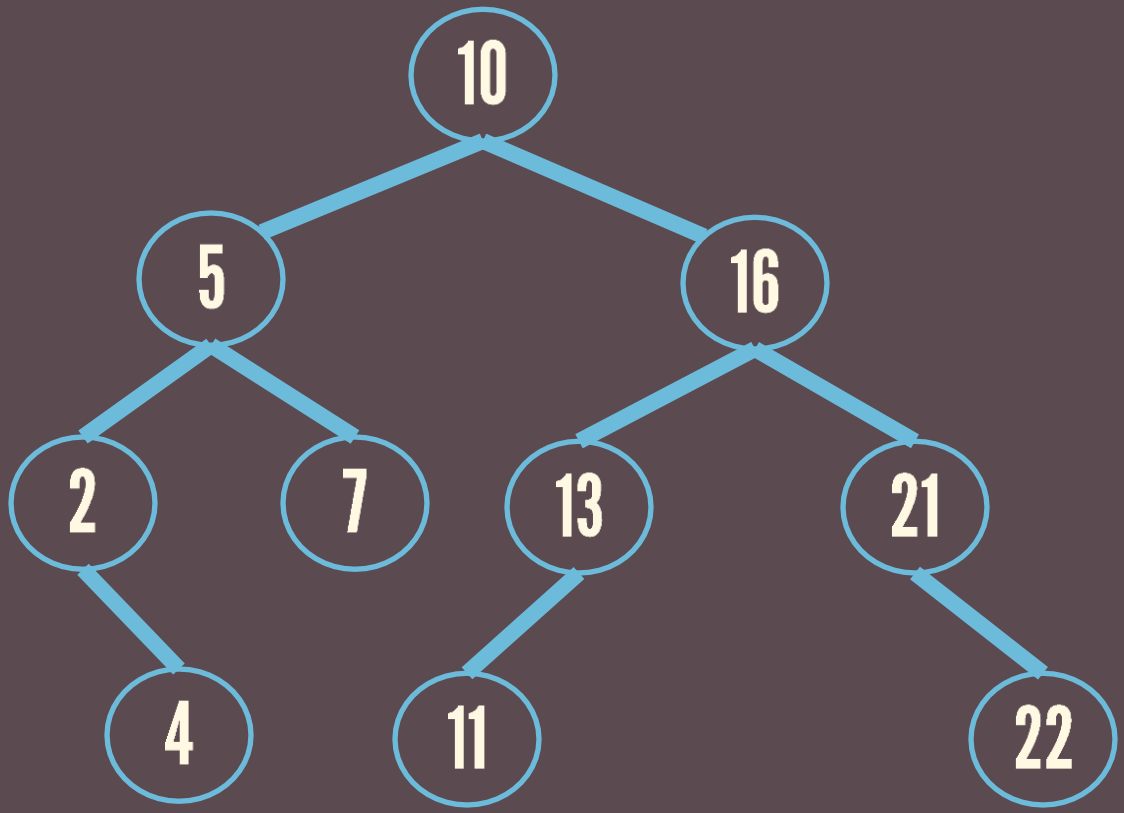
find
insert
delete
minimum
maximum
successor
predecessor

IMPLEMENTATIONS

recursive
non-recursive

IMPLEMENTATIONS

find()


10
5
16
2
7
13
21
4
11
22

```
class BSTNode:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data
```

```
def findval(self, lkpval):
   if lkpval < self.data:
      if __________ is None:
        return str(lkpval)+" Not Found"
      return ____________________
   elif lkpval > self.data:
      if __________ is None:
        return str(lkpval)+" Not Found"
      return ________________________
   else:
      print(str(self.data) + ' is found')
```

```
def findval(self, lkpval):
   if lkpval < self.data:
      if self.left is None:
        return str(lkpval)+" Not Found"
      return self.left.findval(lkpval)
   elif lkpval > self.data:
      if self.right is None:
        return str(lkpval)+" Not Found"
      return self.right.findval(lkpval)
   else:
      print(str(self.data) + ' is found')
```

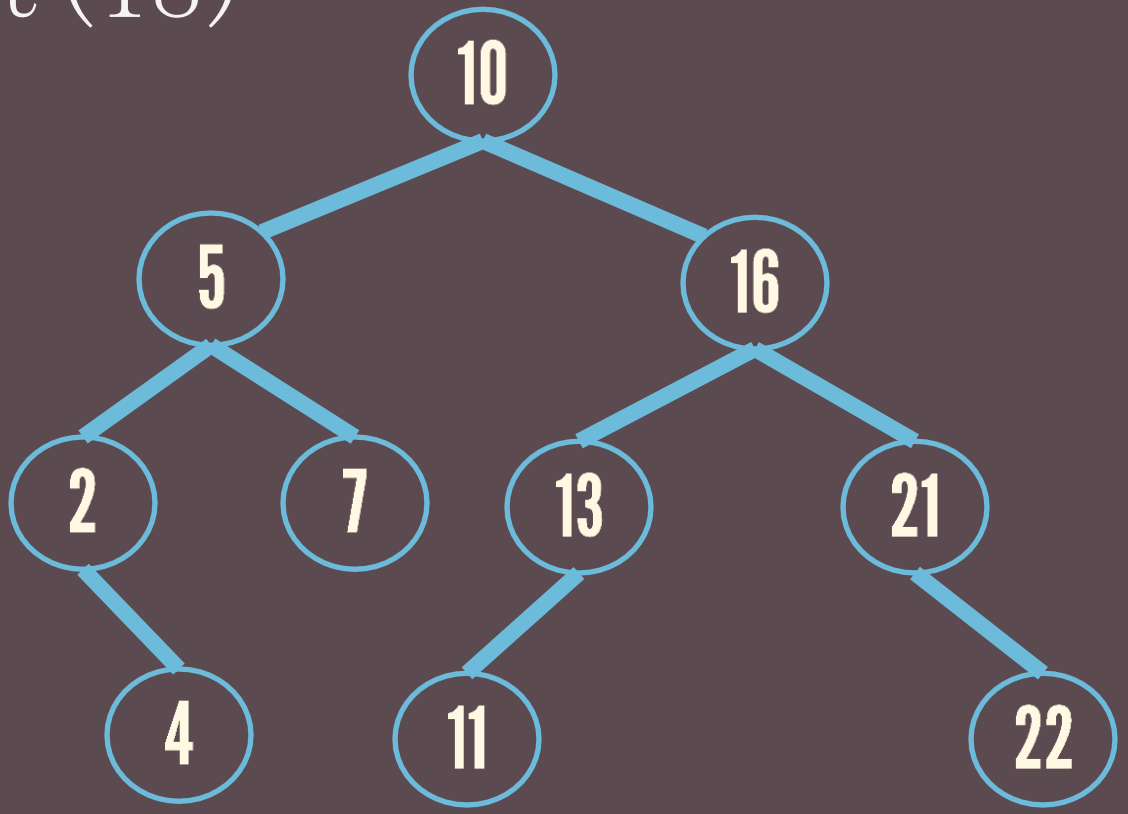
IMPLEMENTATIONS

find()

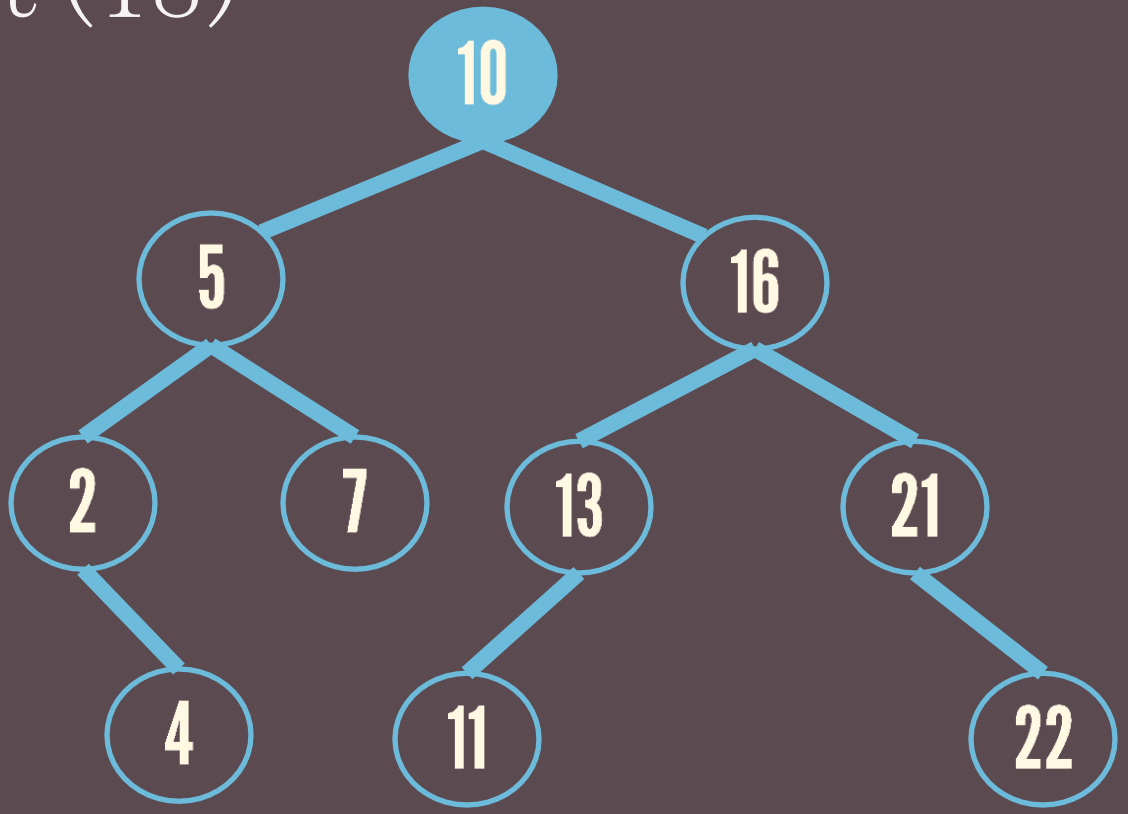
IMPLEMENTATIONS

insert()

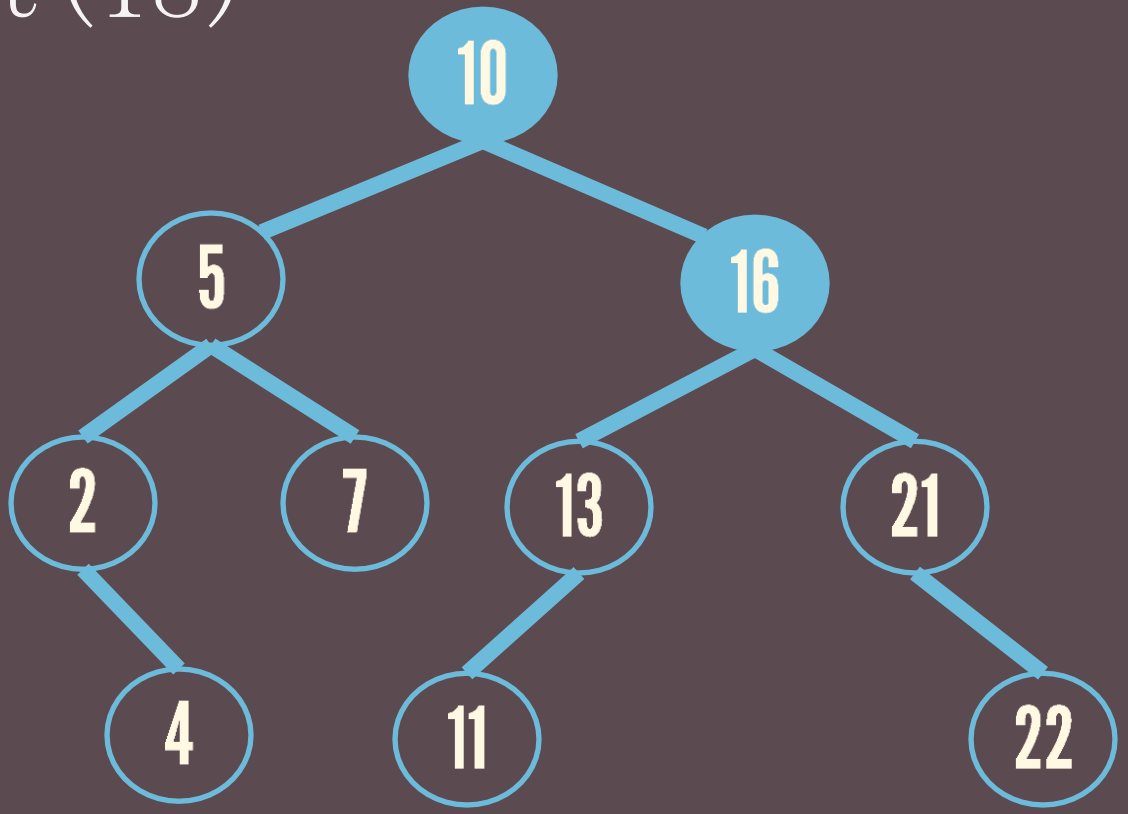
insert(18)



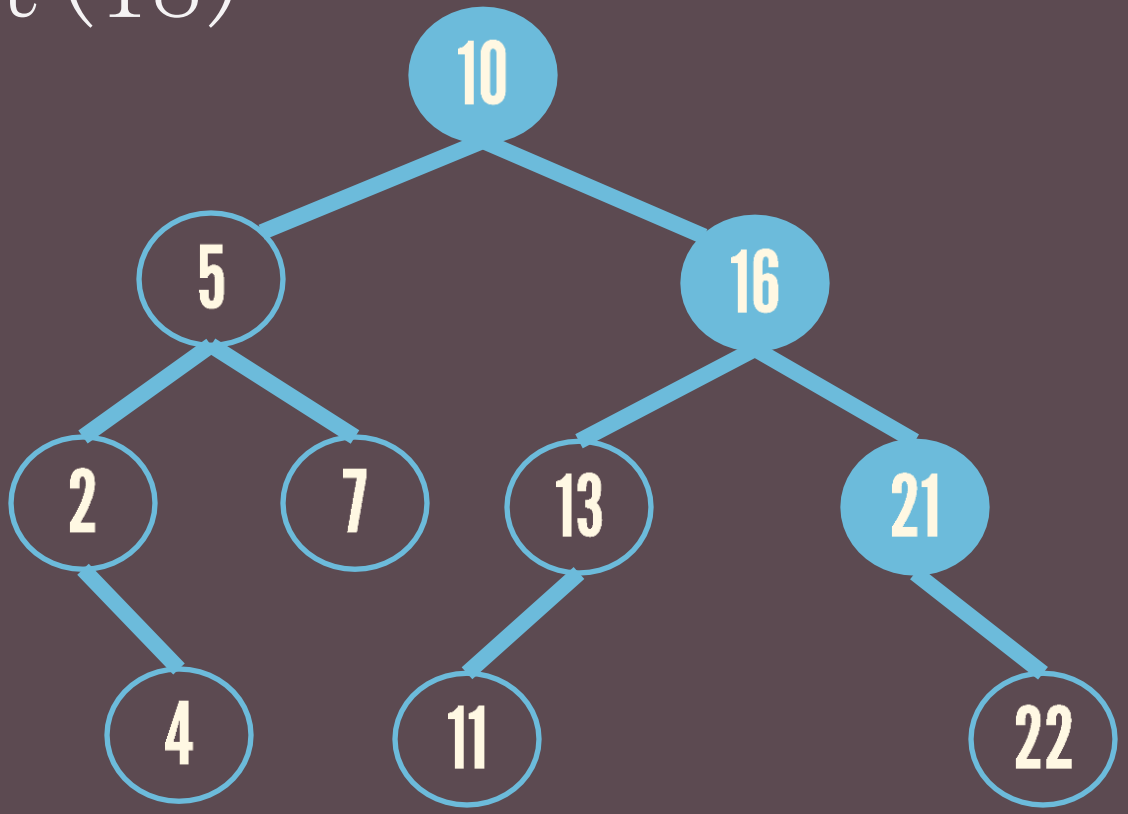
insert(18)



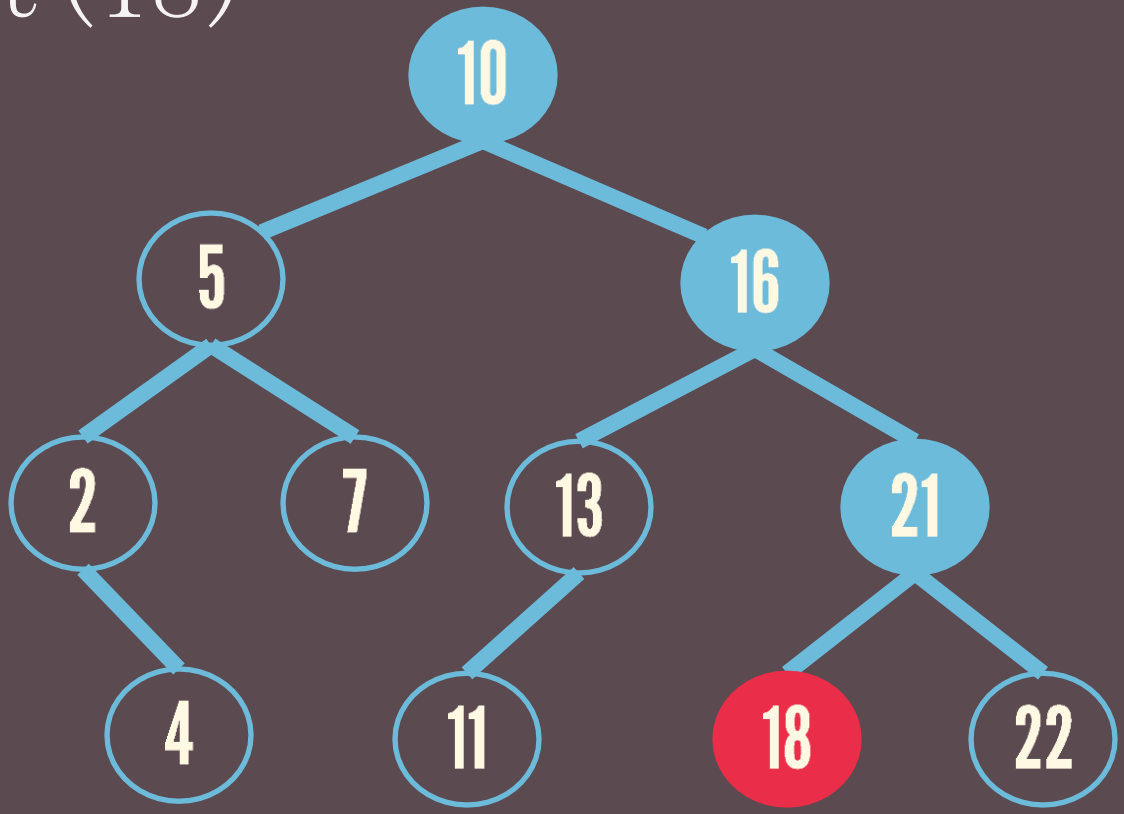
insert(18)



insert(18)



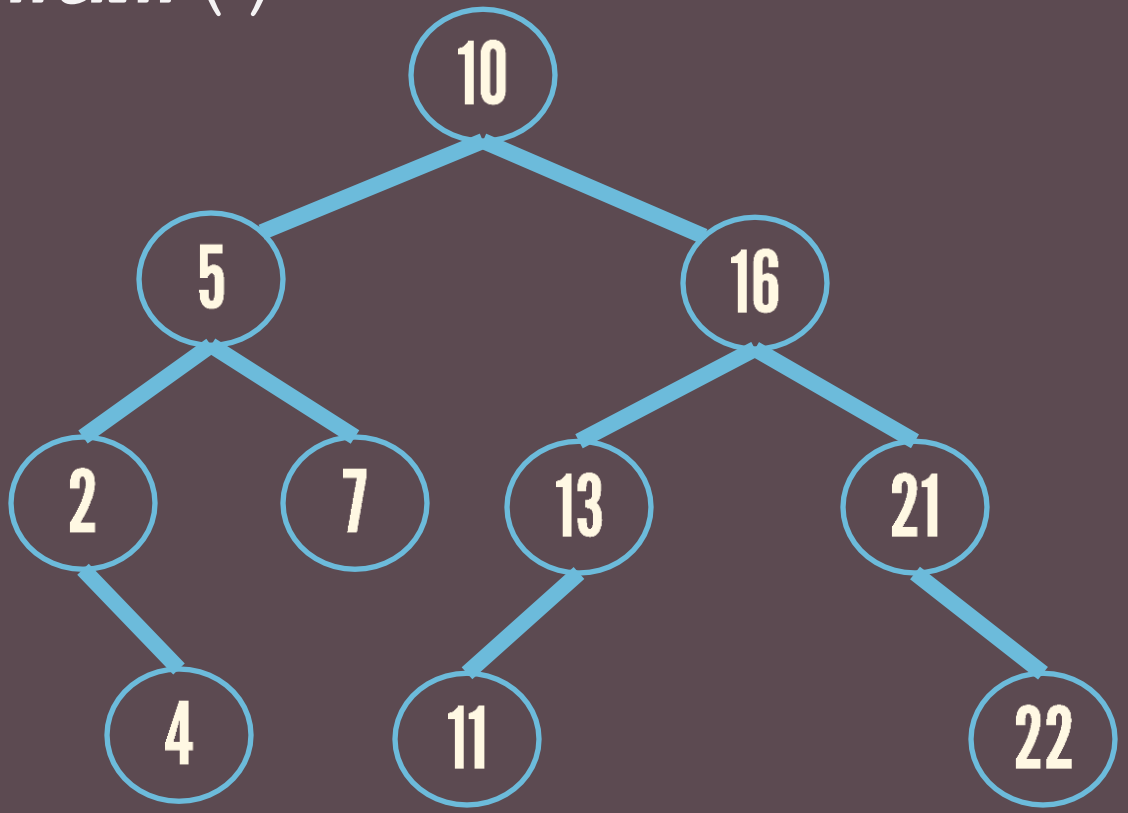
insert(18)



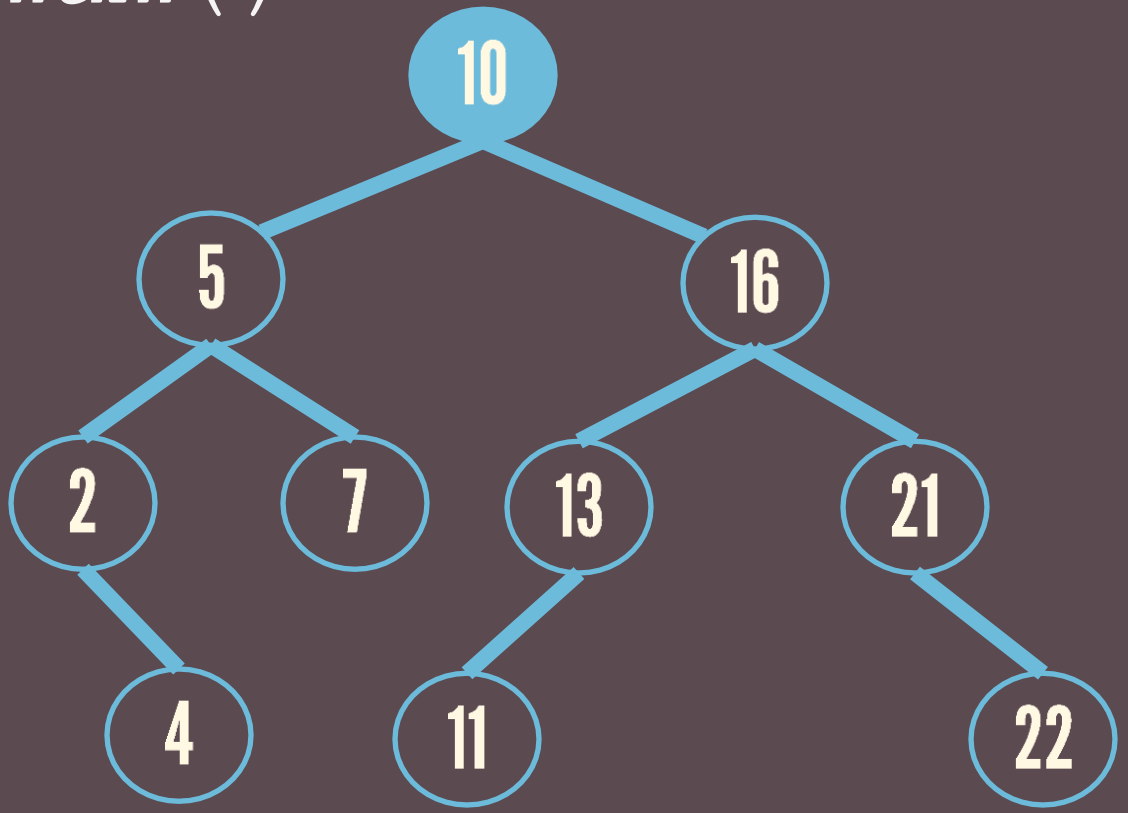
IMPLEMENTATIONS

minimum()

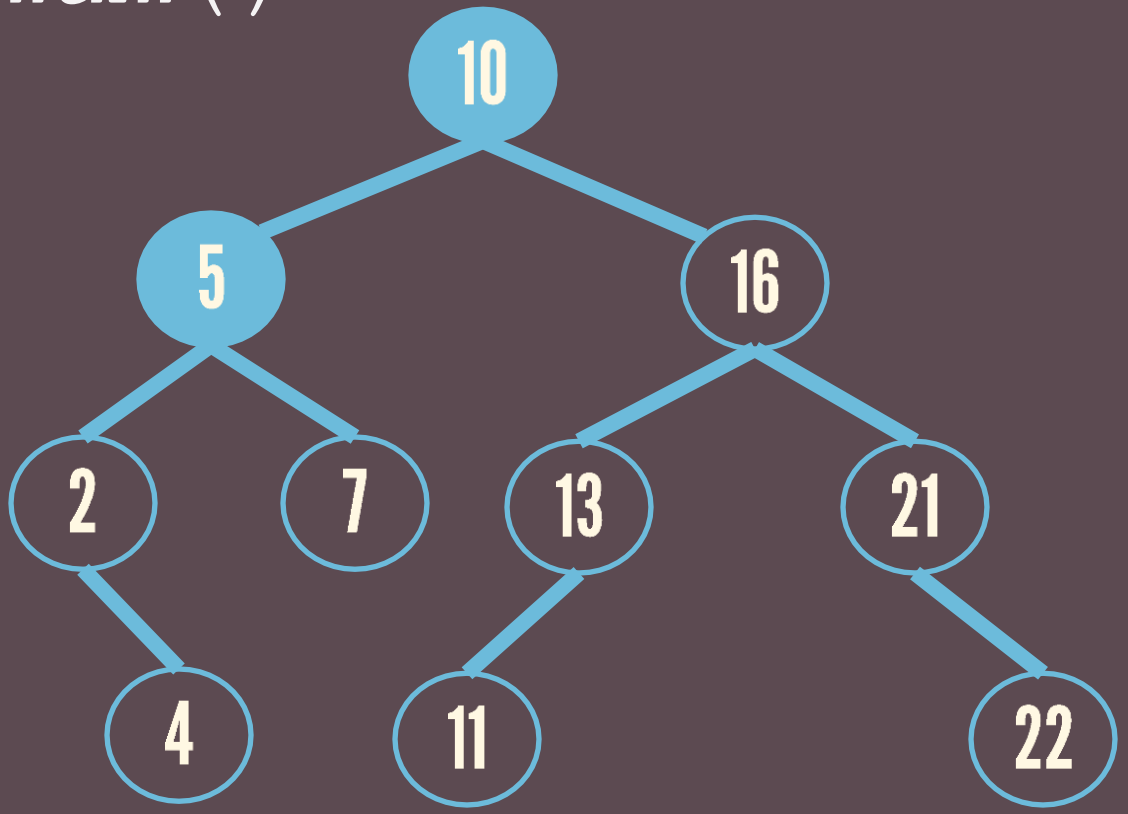
`minimum()`



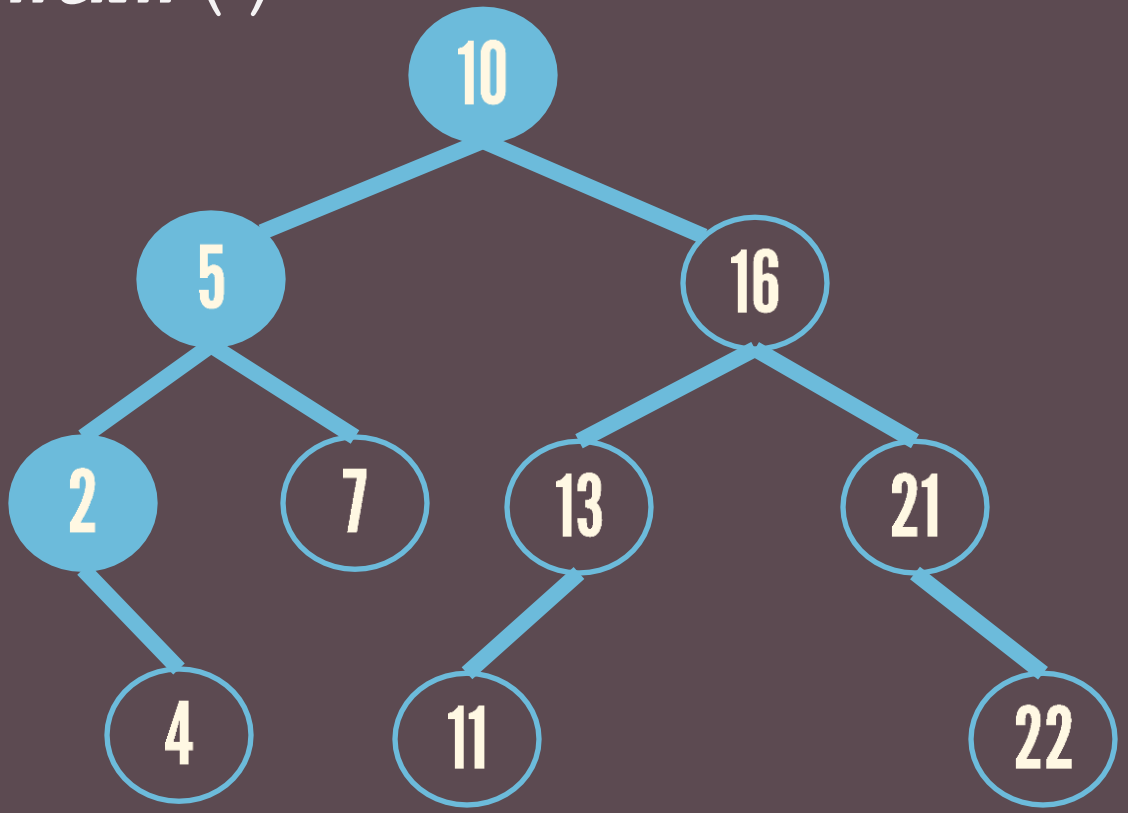
minimum()



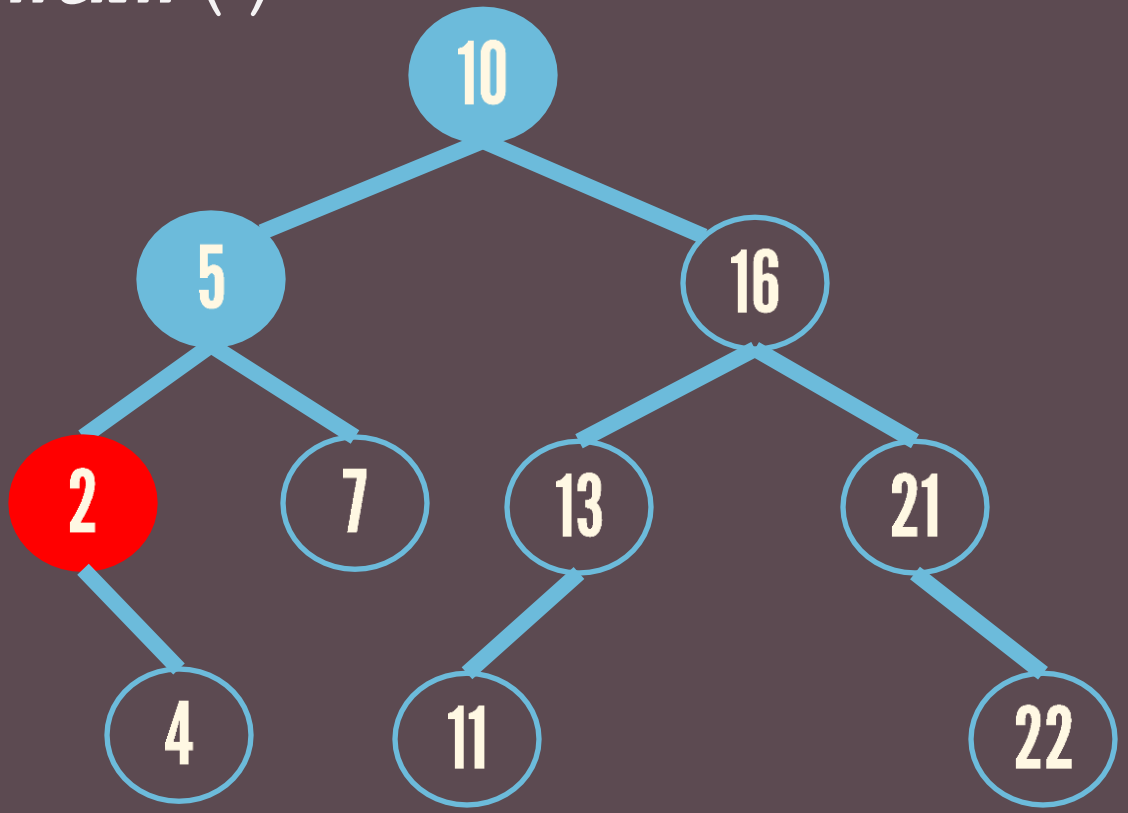
minimum()



minimum()



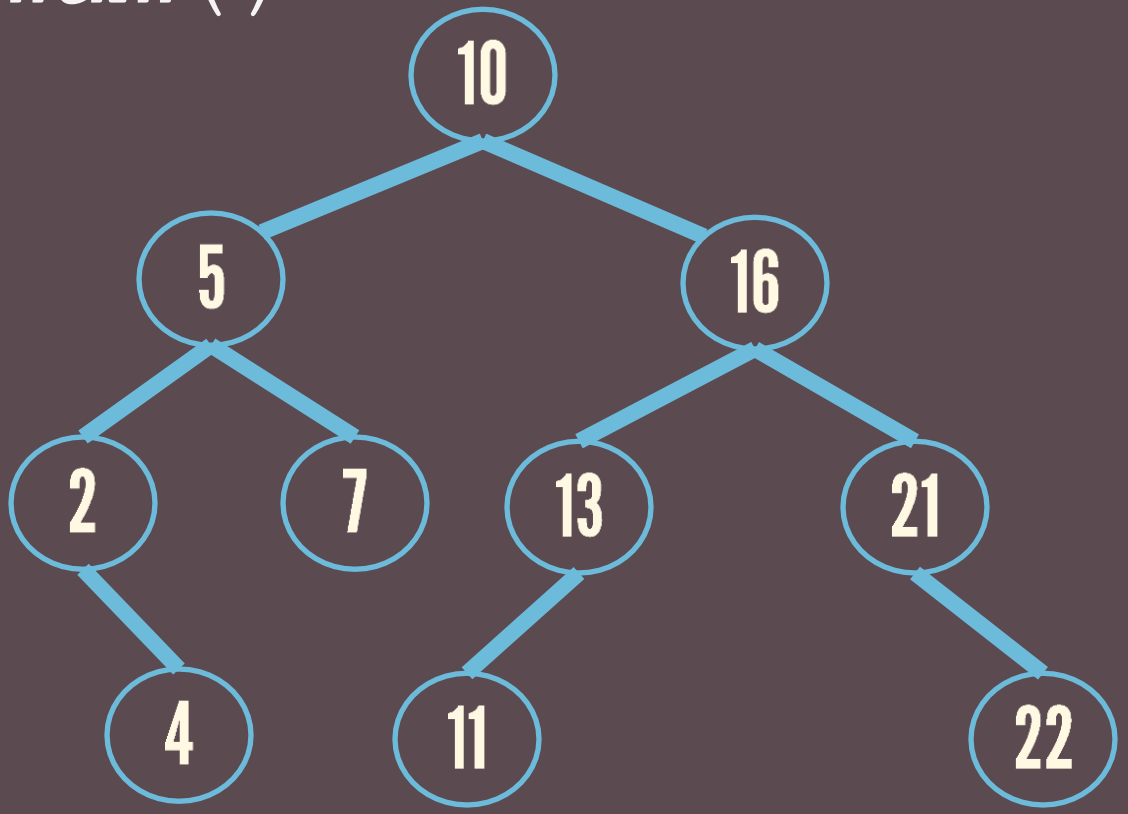
minimum()



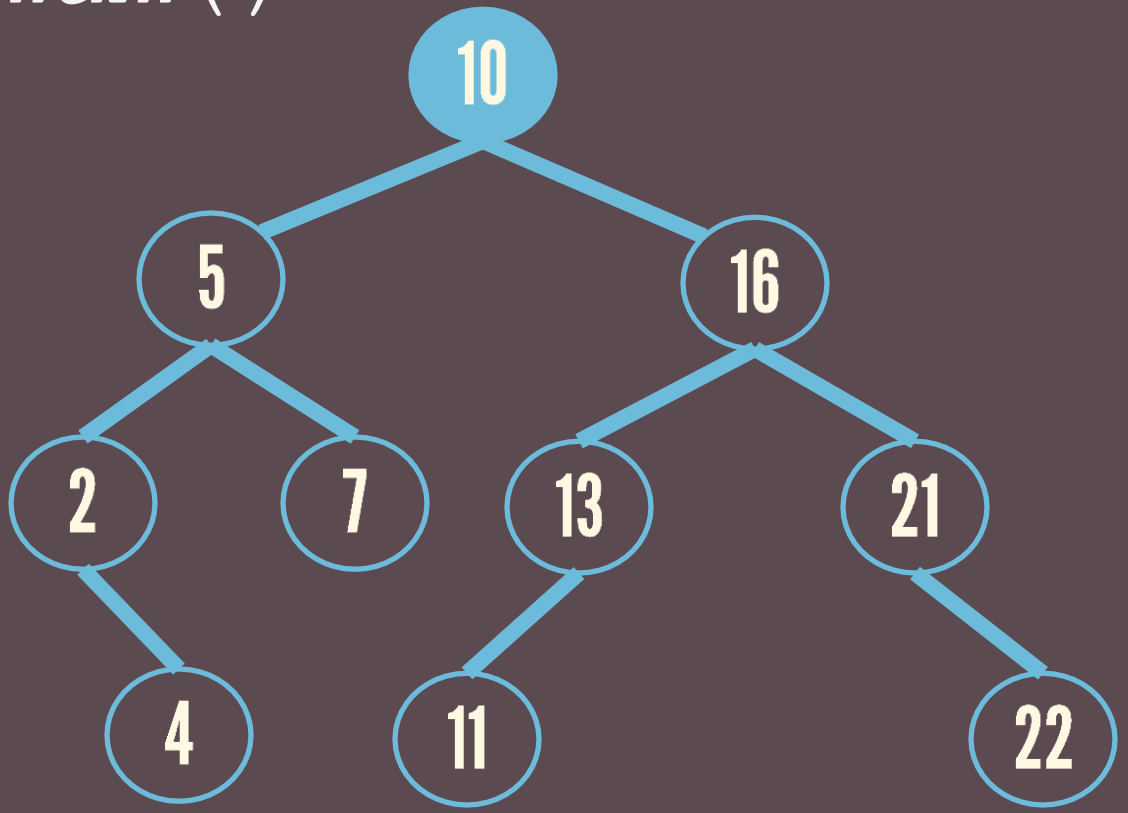
IMPLEMENTATIONS

maximum()

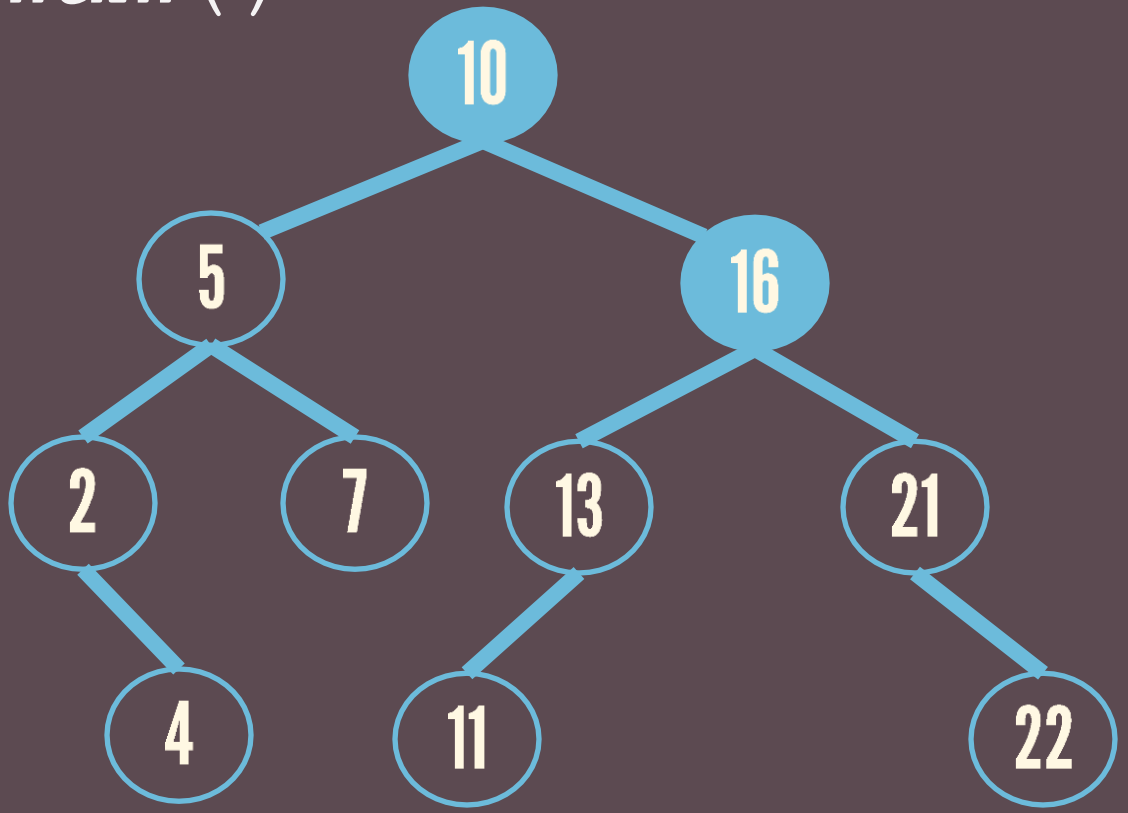
maximum()



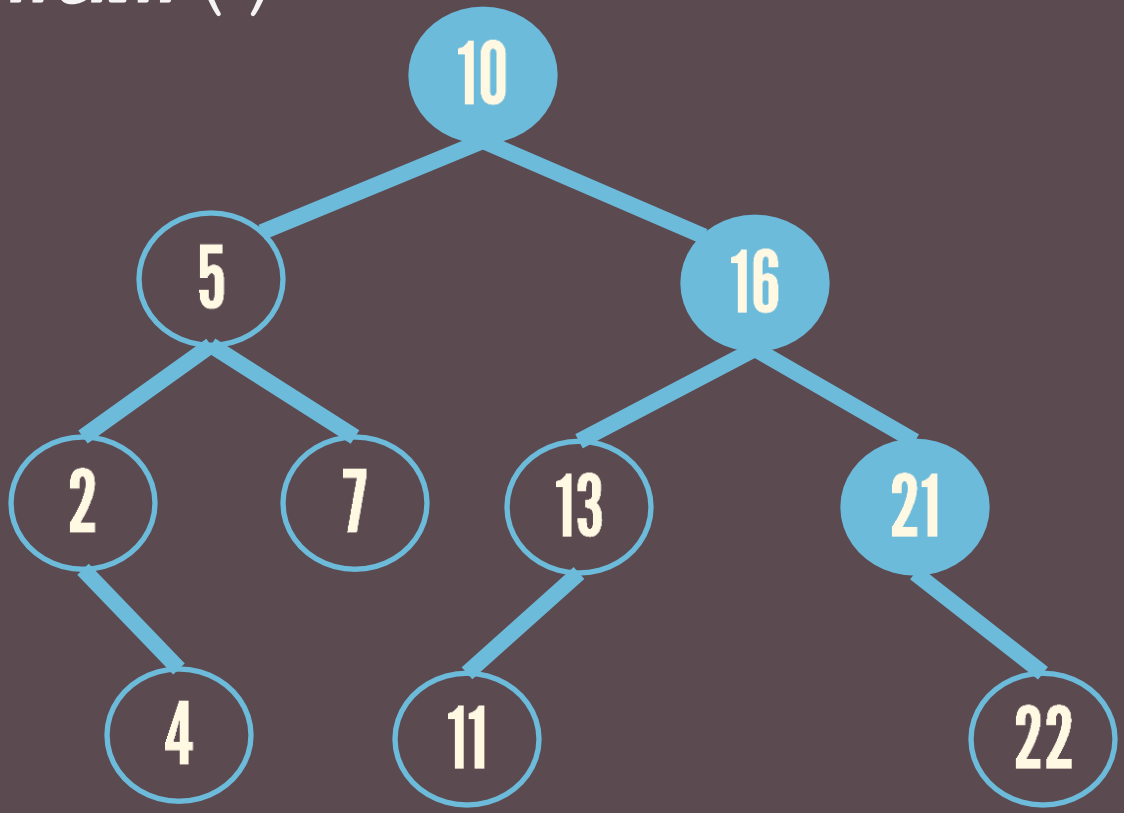
maximum()



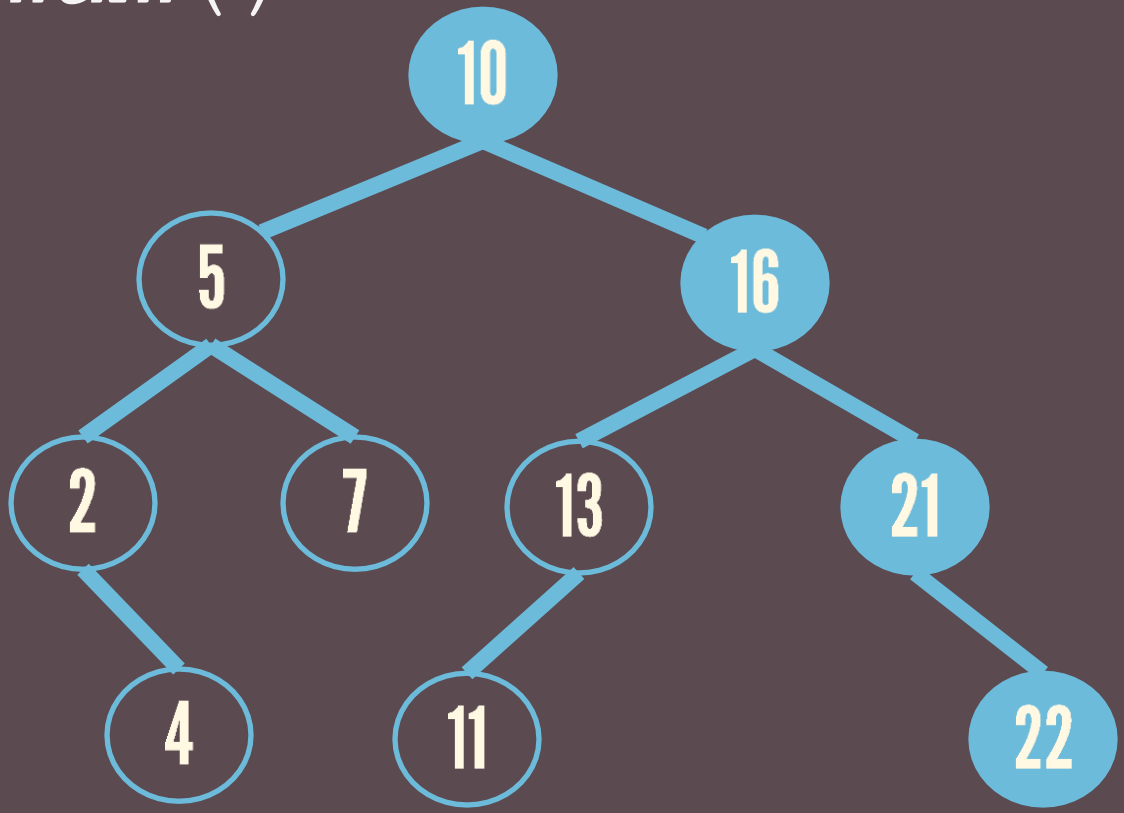
maximum()



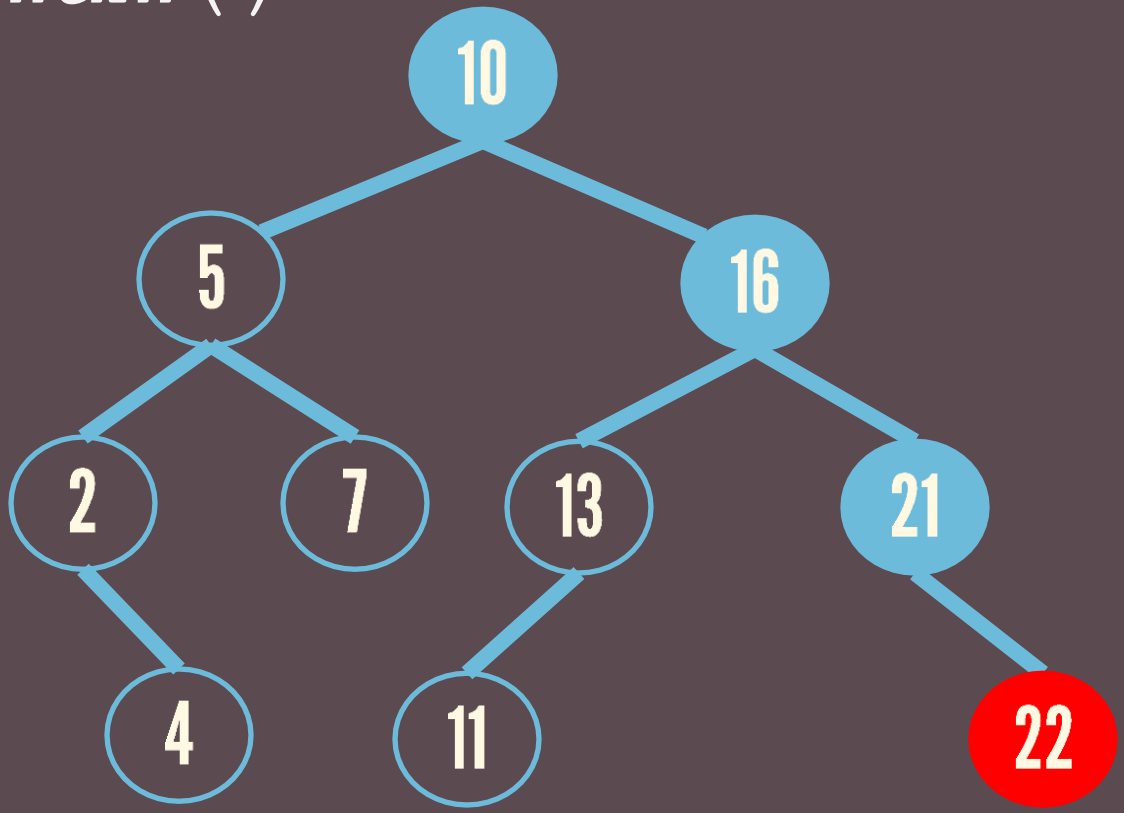
maximum()



maximum()



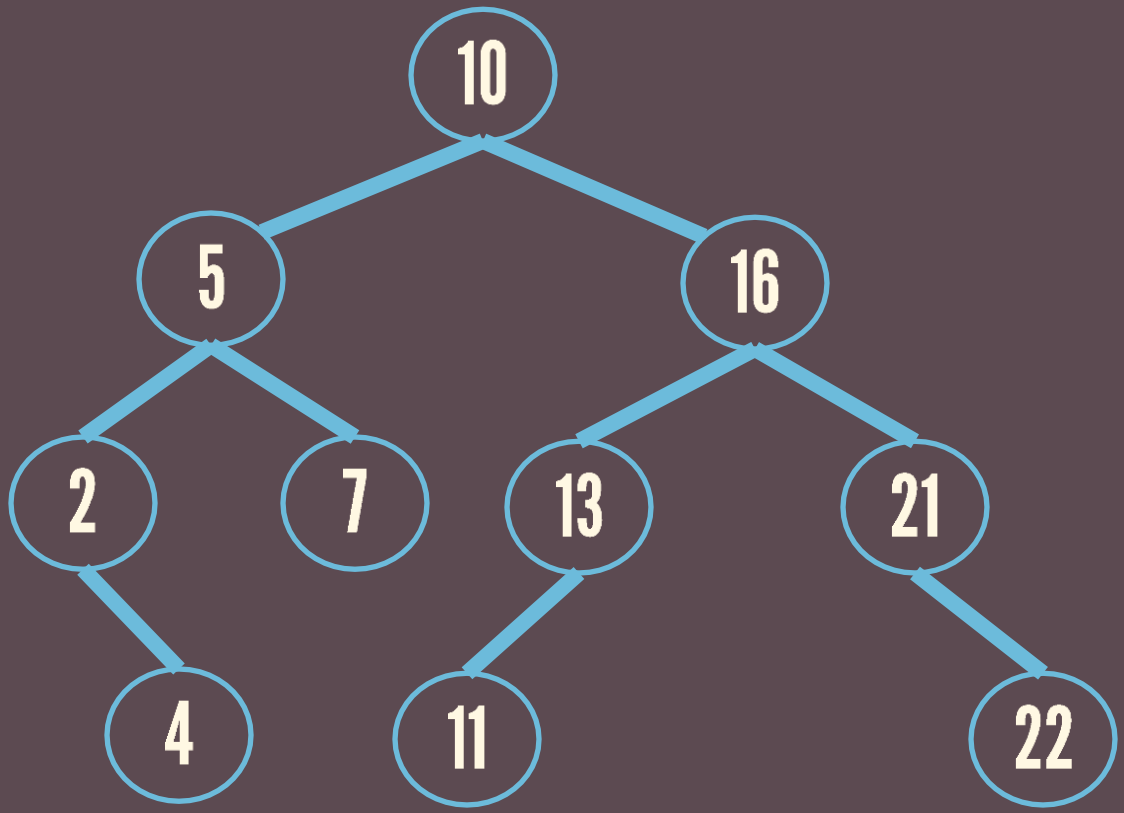
maximum()



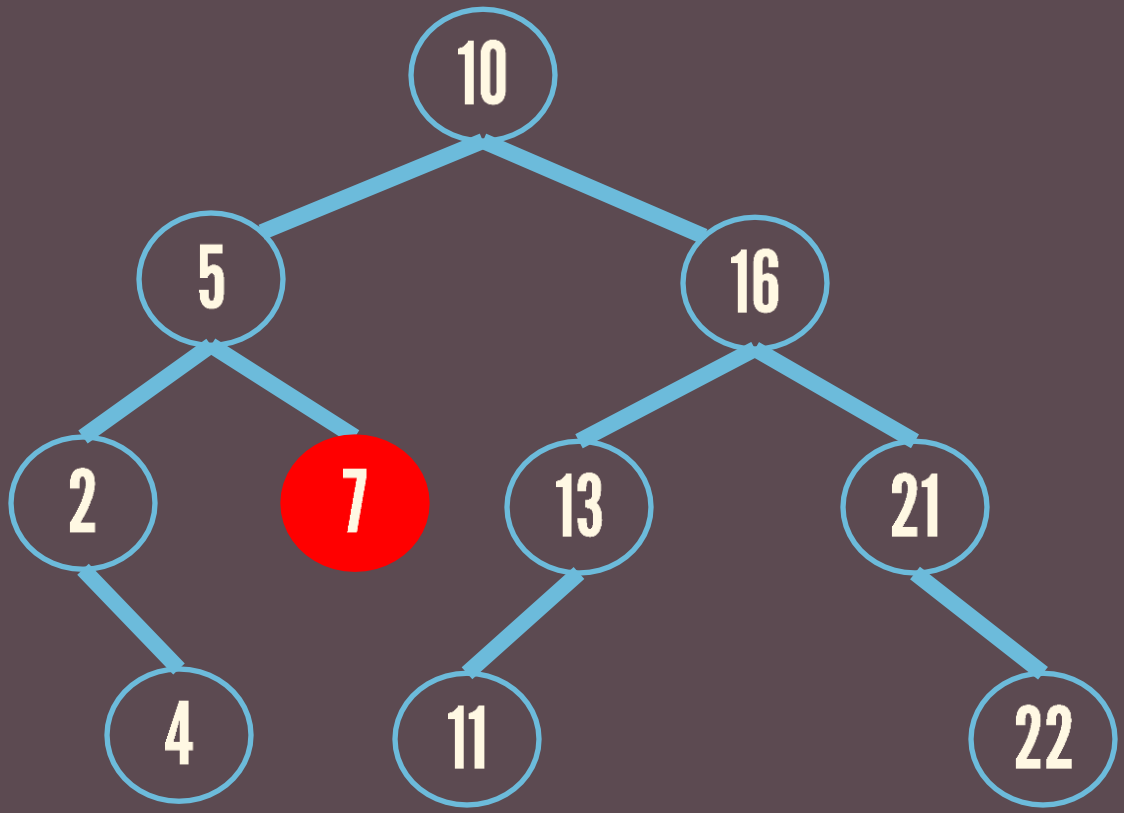
IMPLEMENTATIONS

predecessor()

`predecessor(8)`



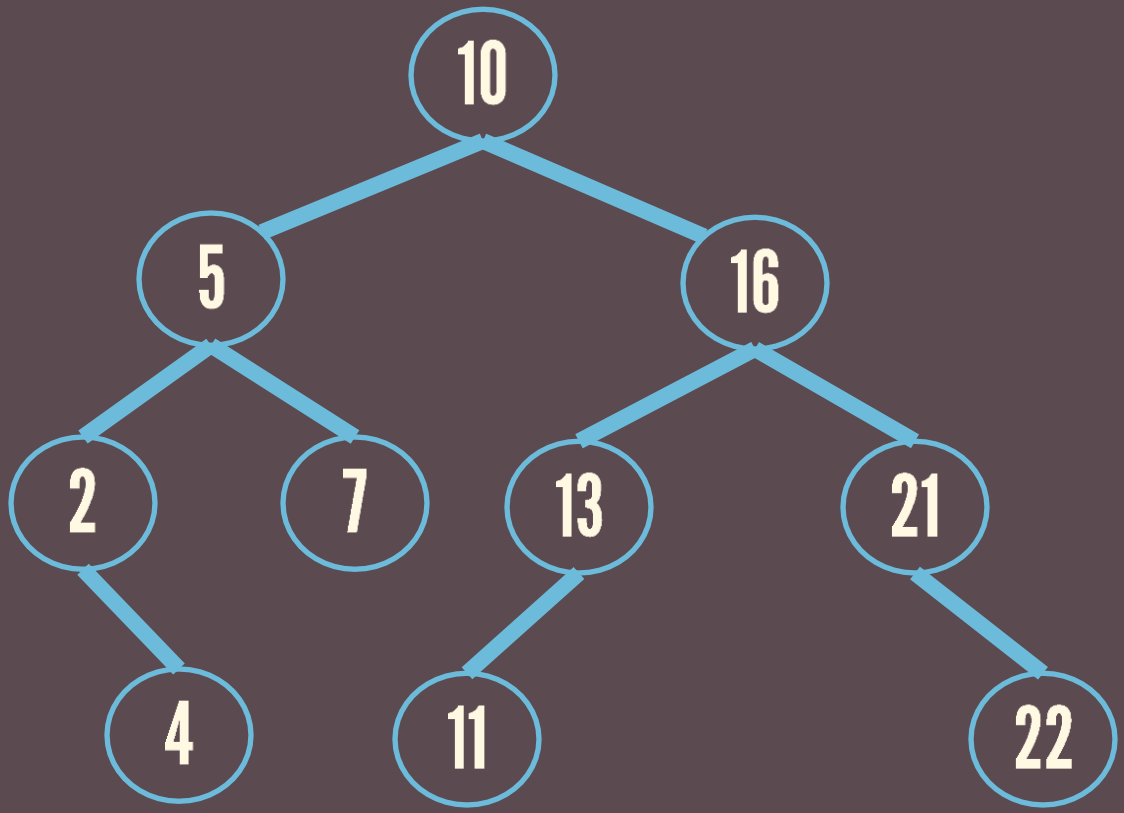
`predecessor(8)`



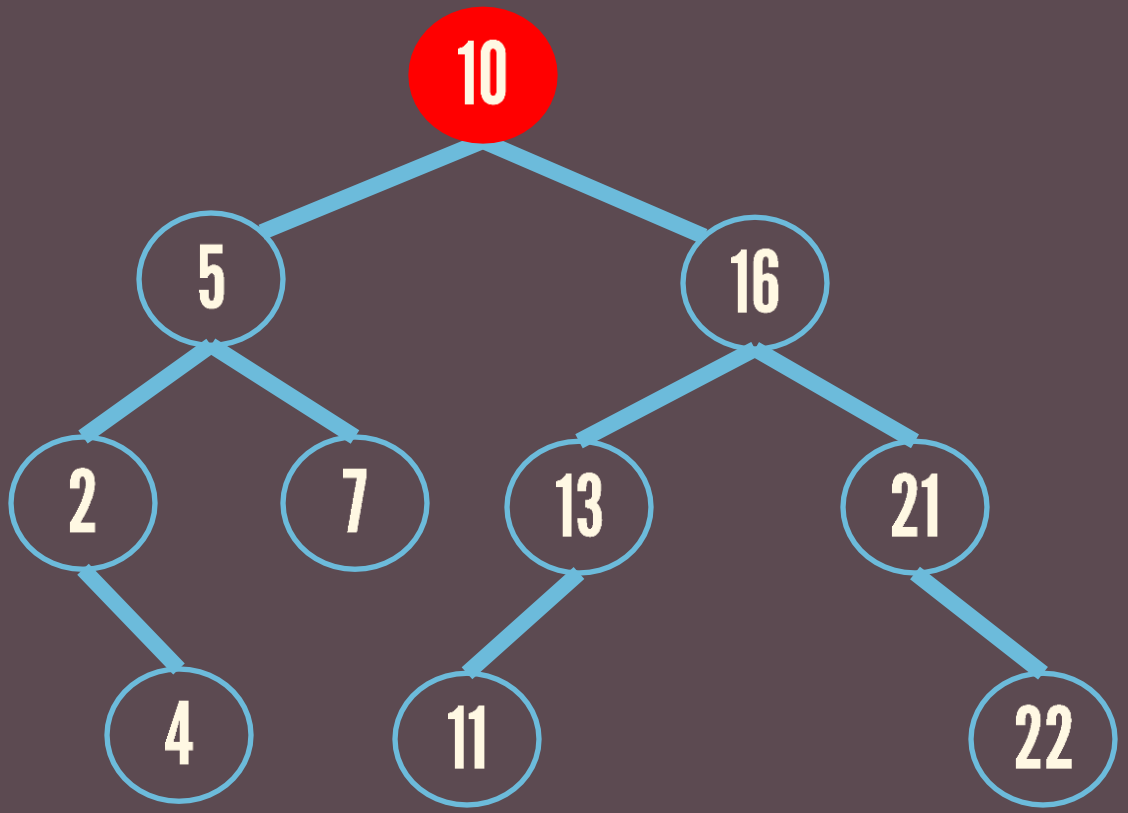
IMPLEMENTATIONS

successor()

successor(8)



successor(8)



IMPLEMENTATIONS

printBST()

IMPLEMENTATIONS

delete()

delete()

3 cases

Node is a leaf

Node has one child

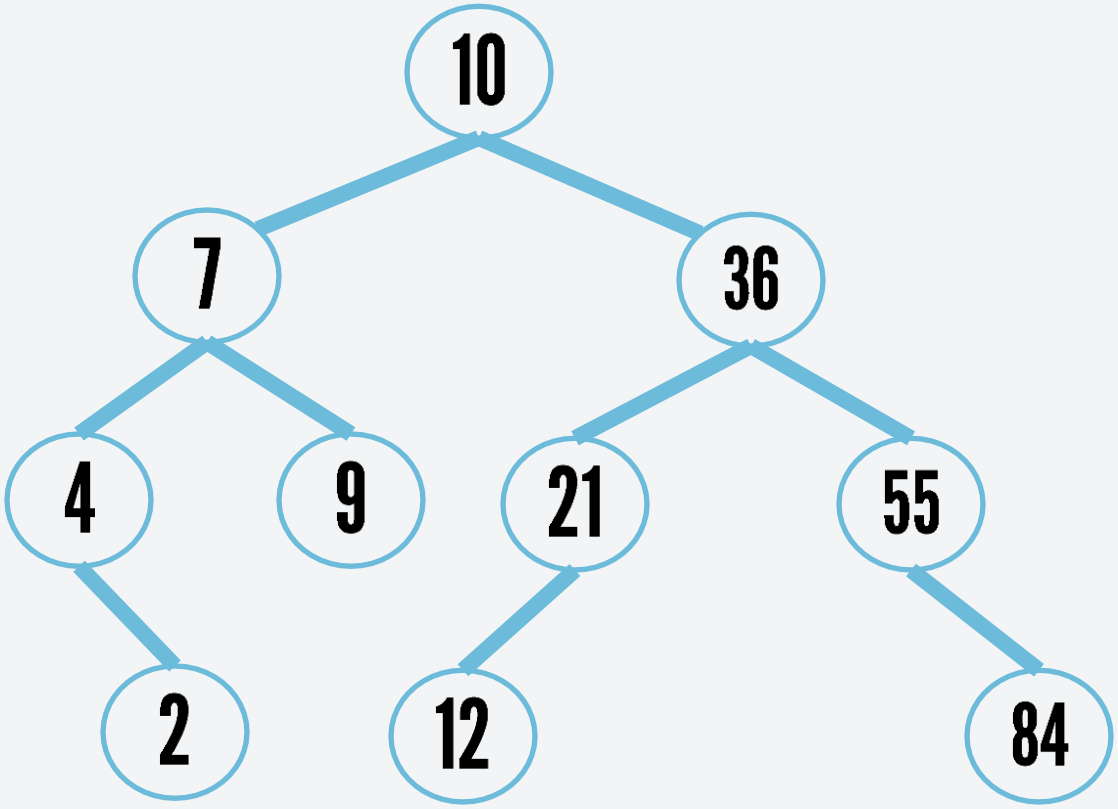
Node has two children

Node is a leaf

The node can be deleted immediately.

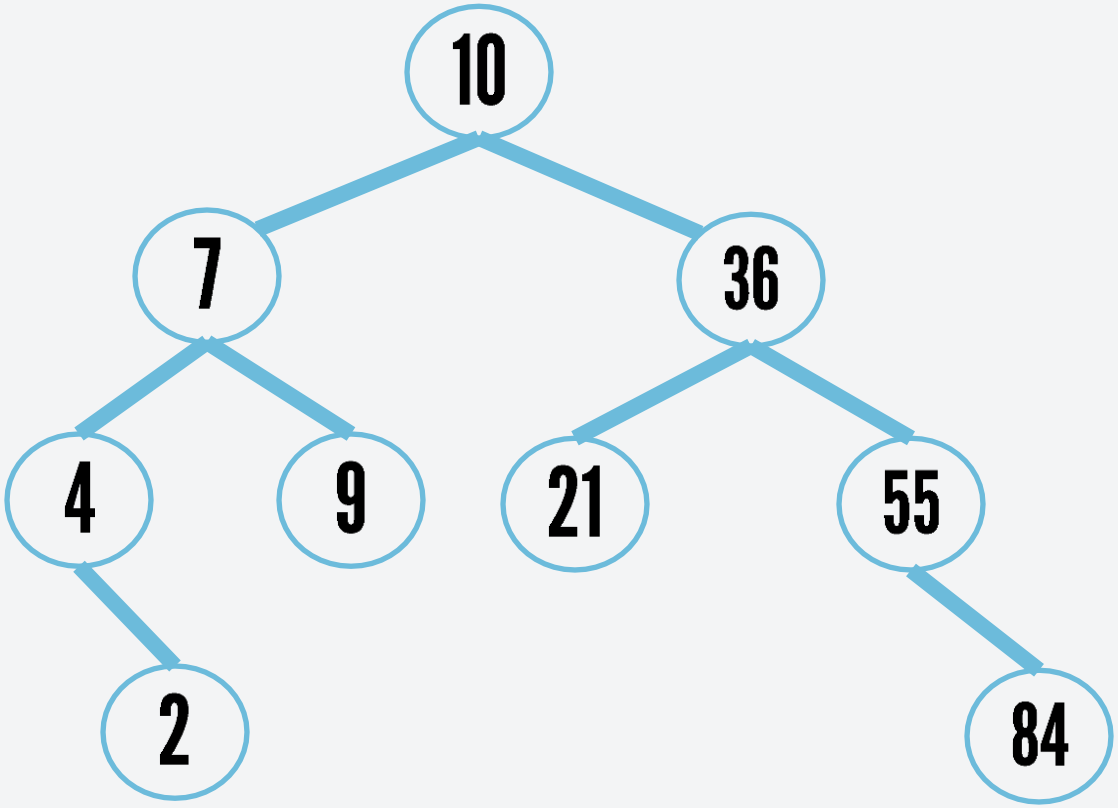
Node is a leaf

delete(12)



Node is a leaf

delete(12)

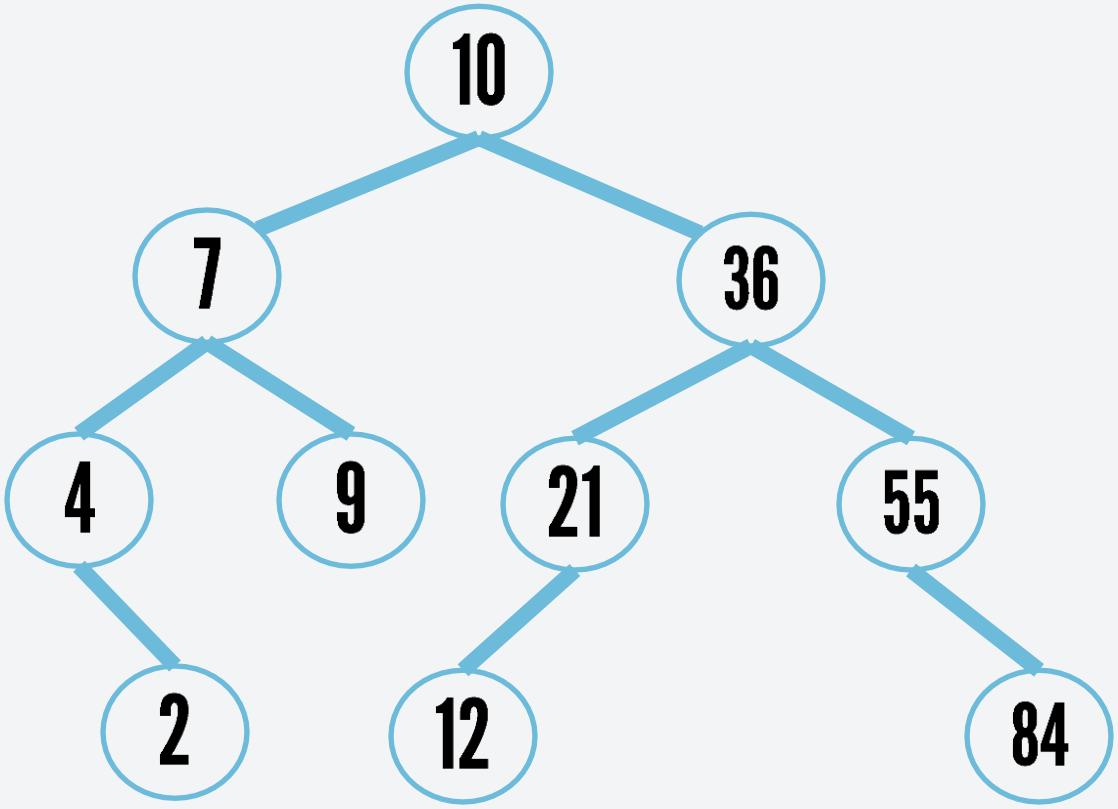


Node has one child

Its parent adjusts a pointer to bypass the node.

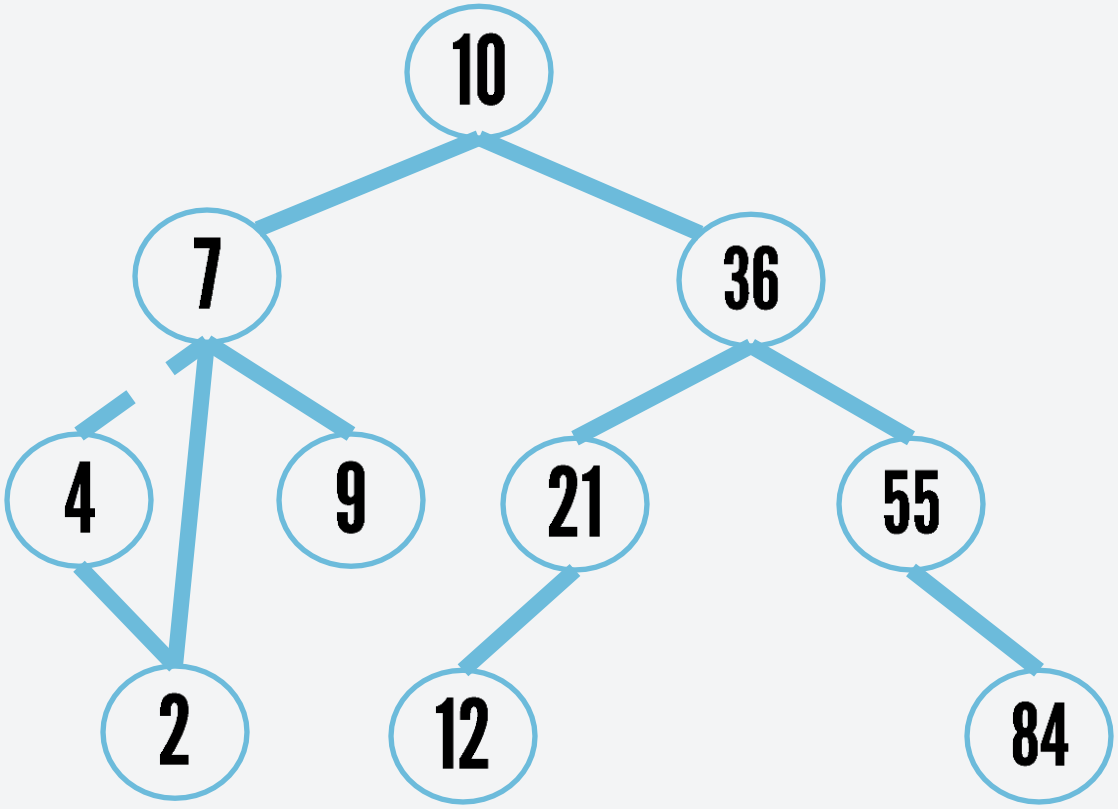
delete(4)

Node has one child



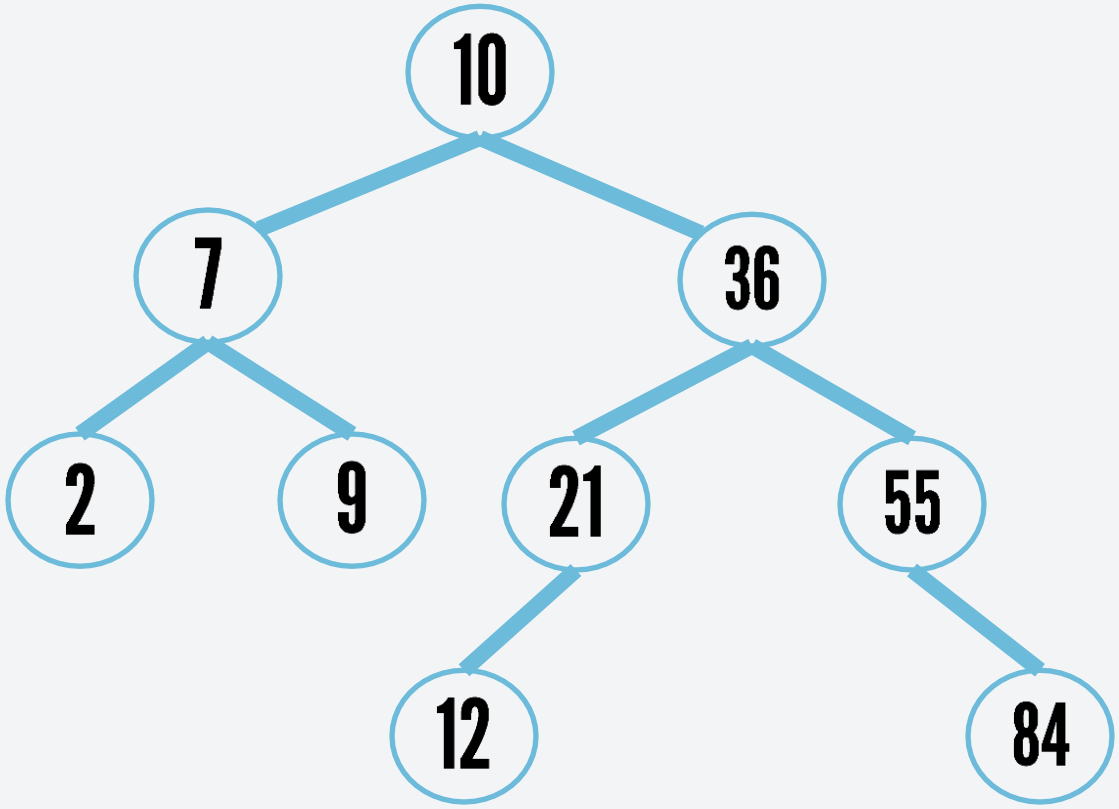
Node has one child

delete(4)



delete(4)

Node has one child



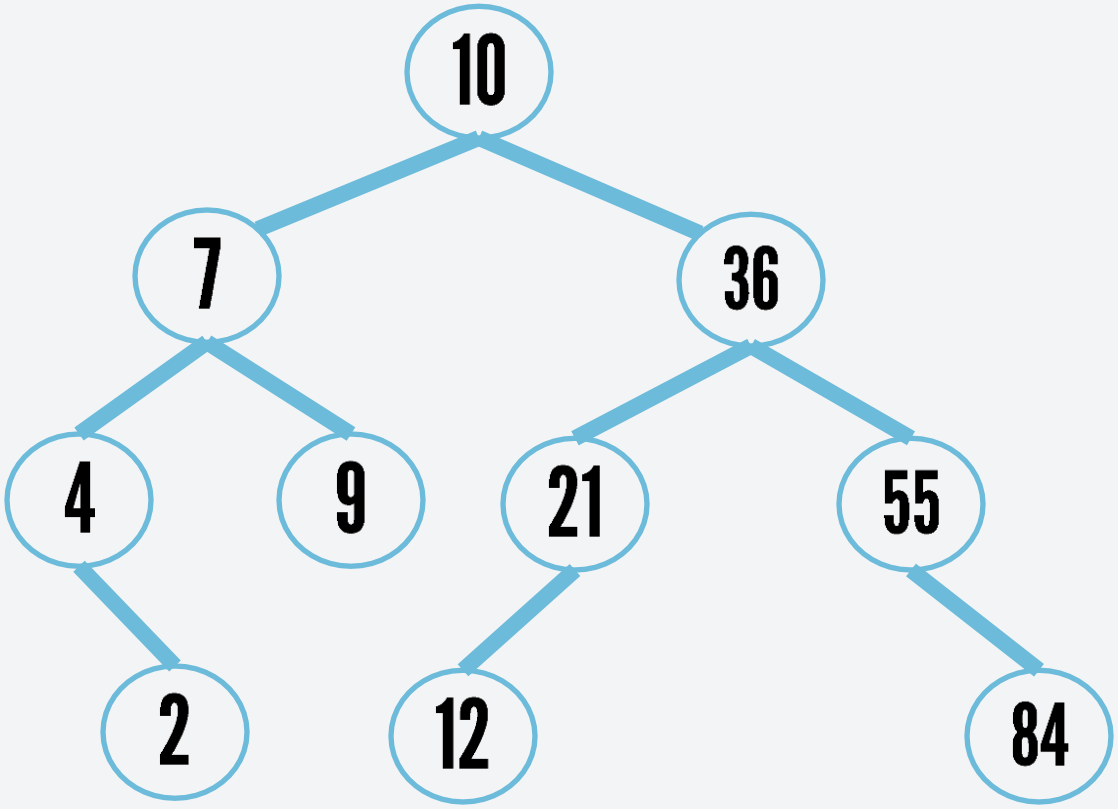
Node has two children

Replace this node with the smallest key of the right subtree.

Recursively delete this node.

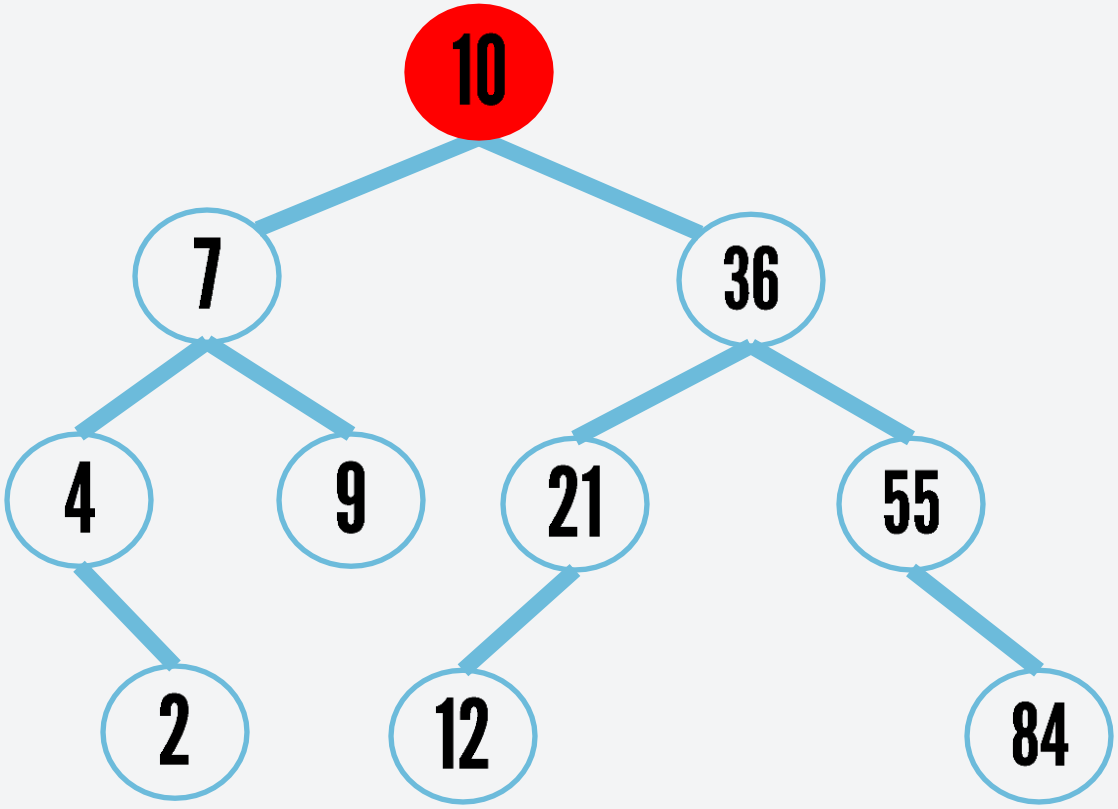
Node has two children

delete(10)



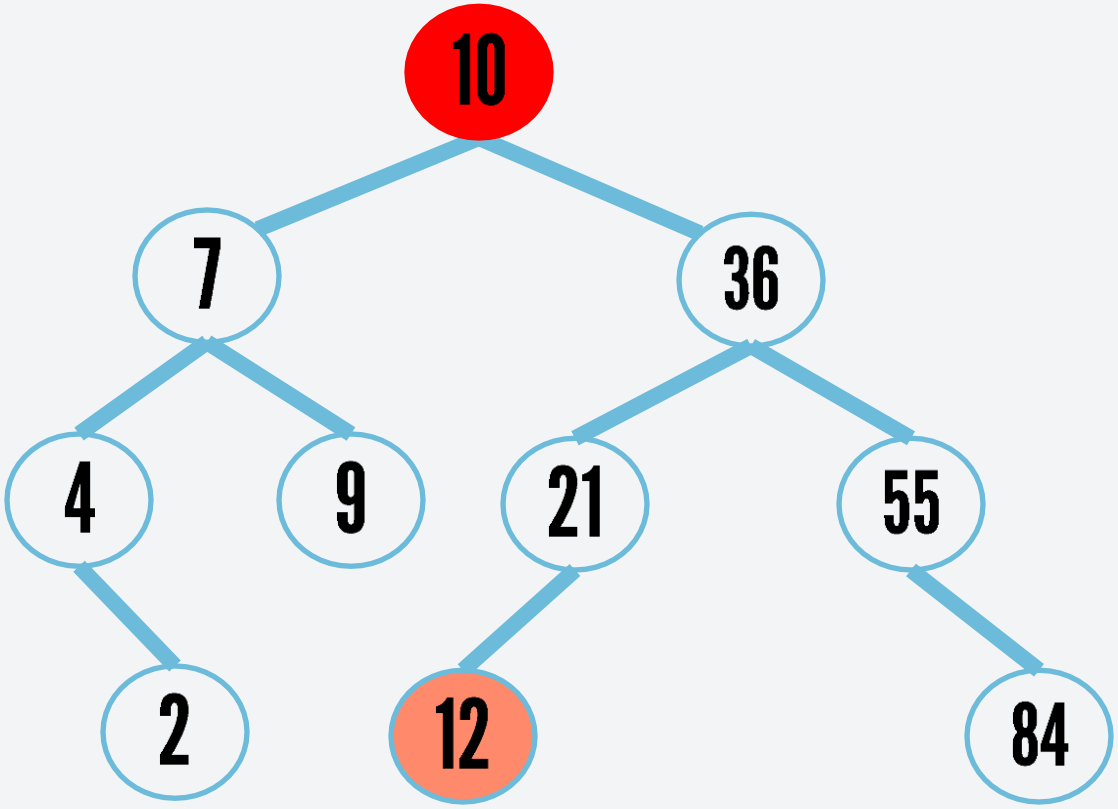
Node has two children

delete(10)



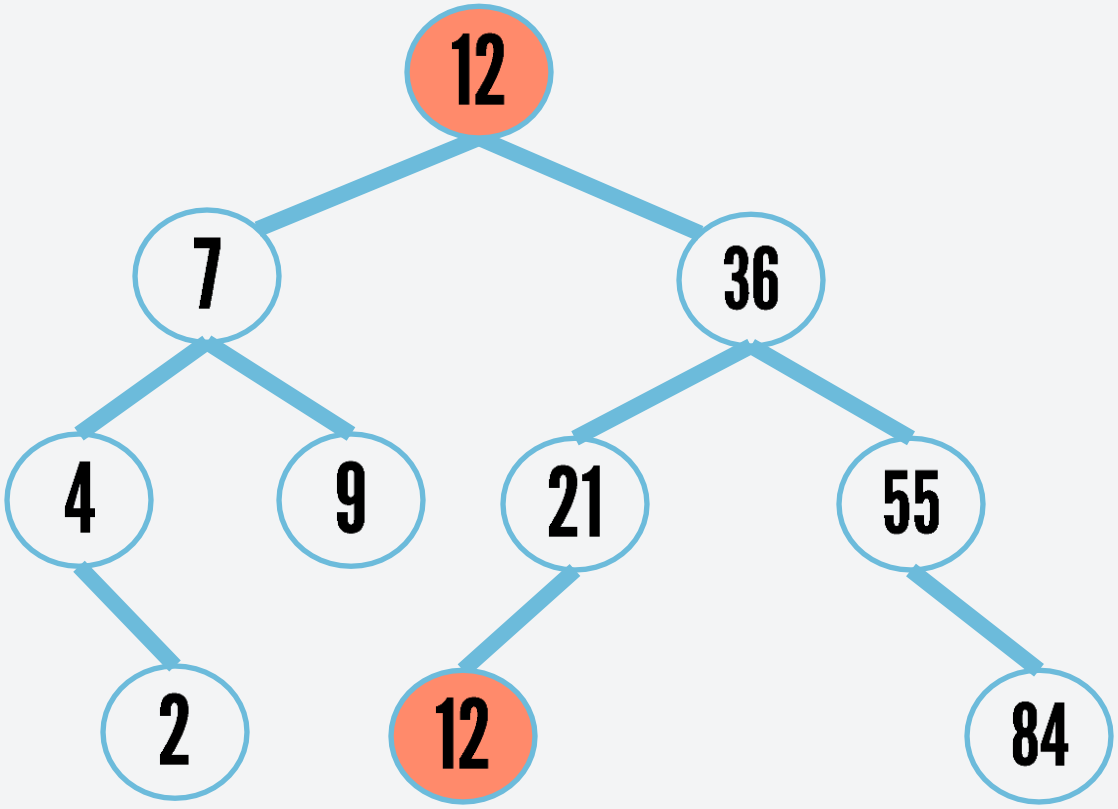
Node has two children

delete(10)



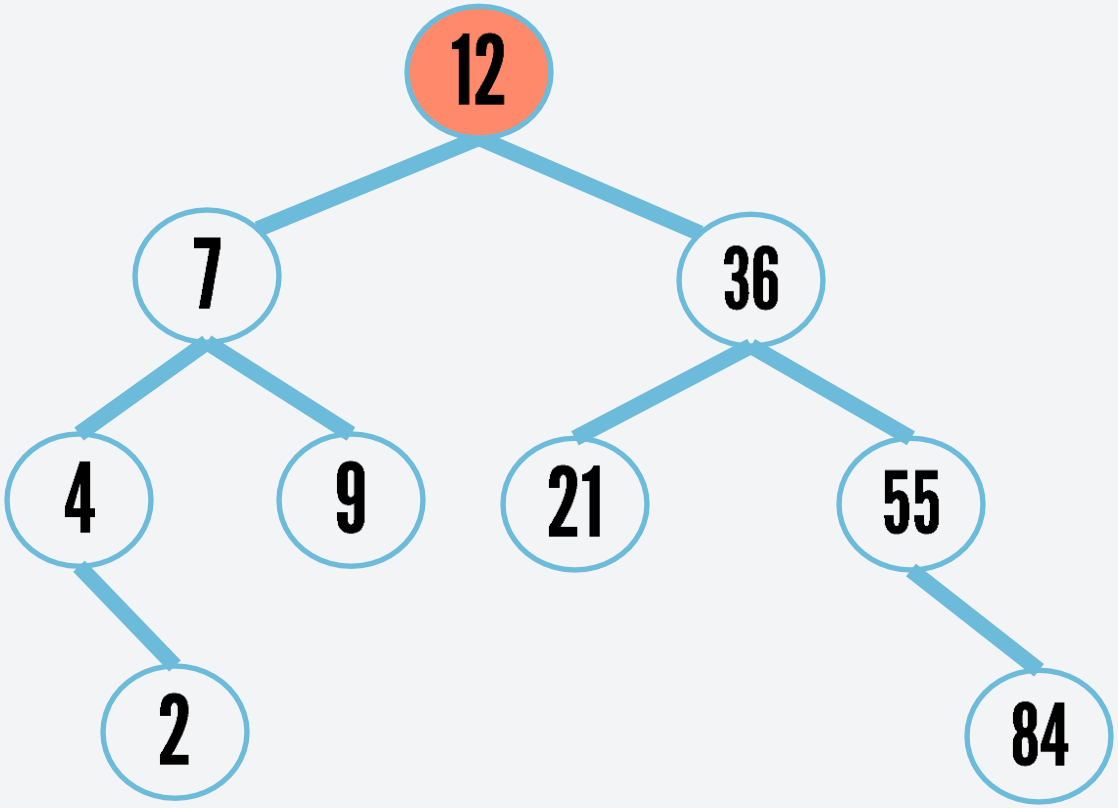
Node has two children

delete(10)



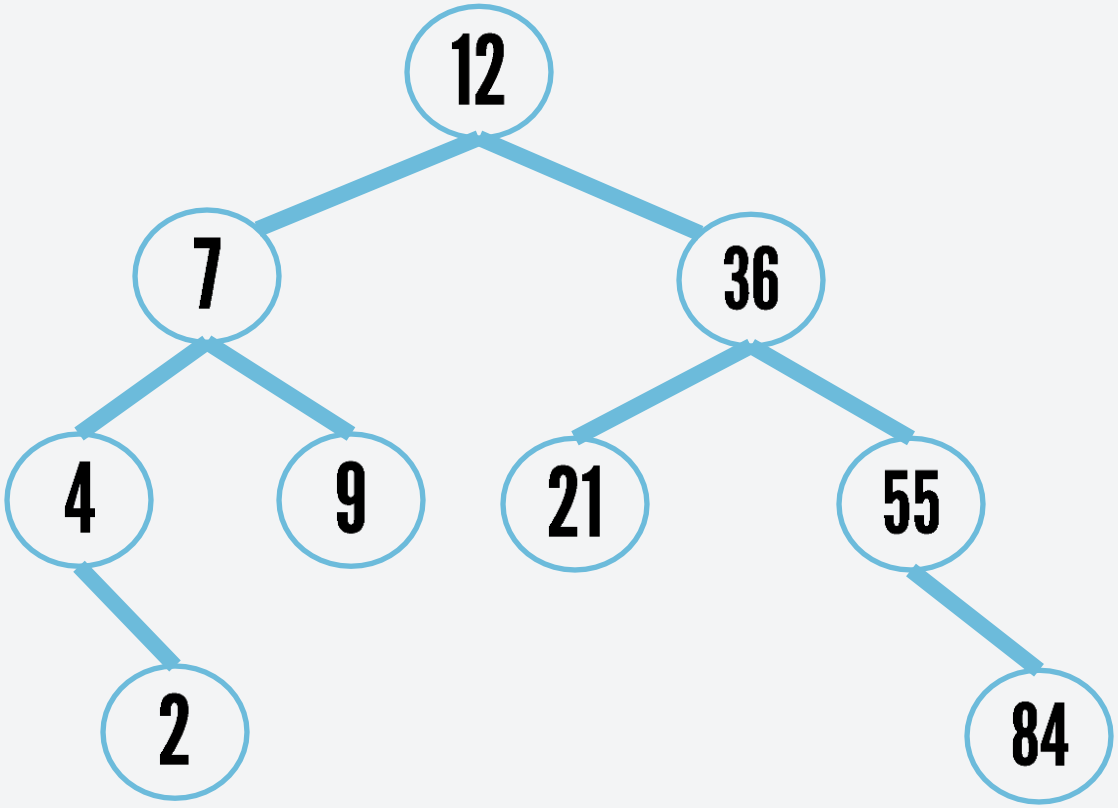
Node has two children

delete(10)



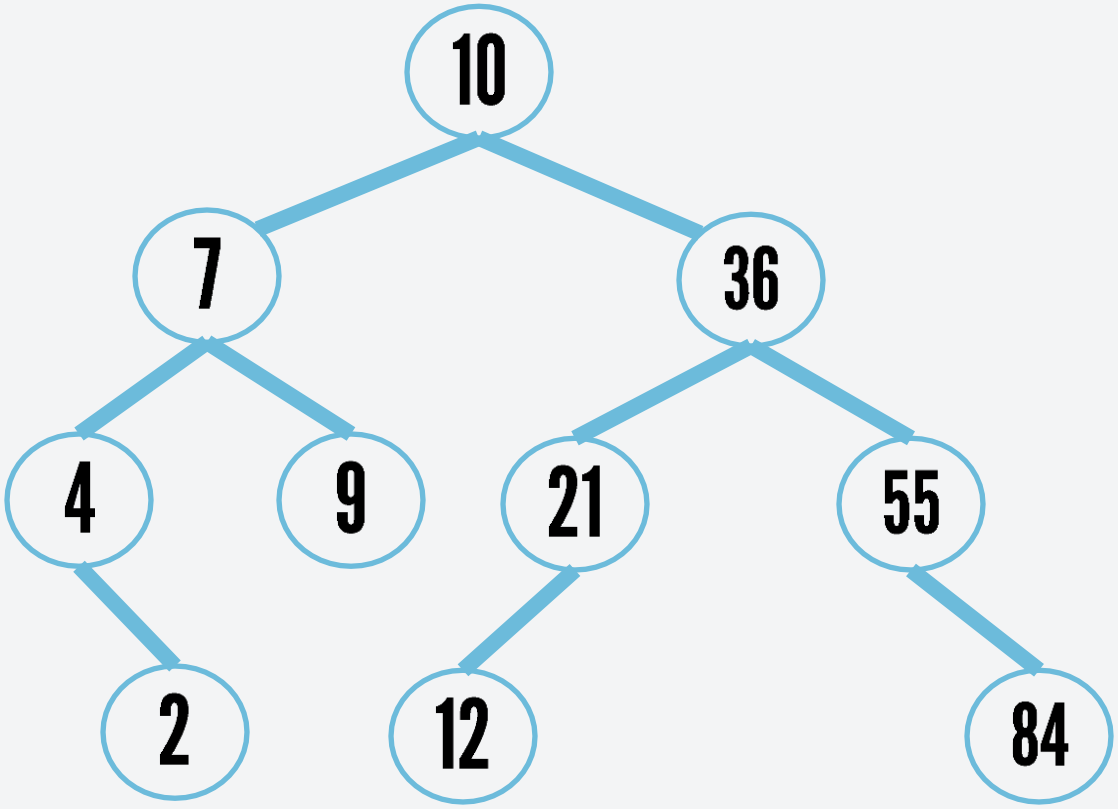
Node has two children

delete(10)



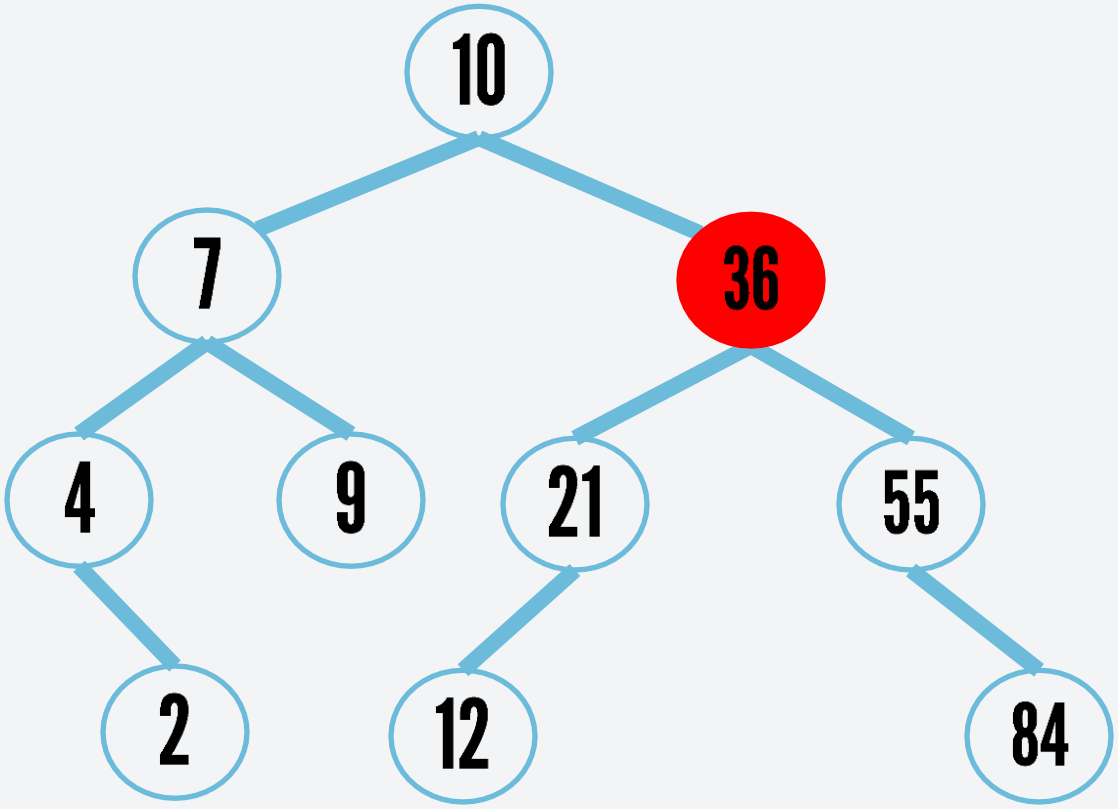
Node has two children

delete(36)



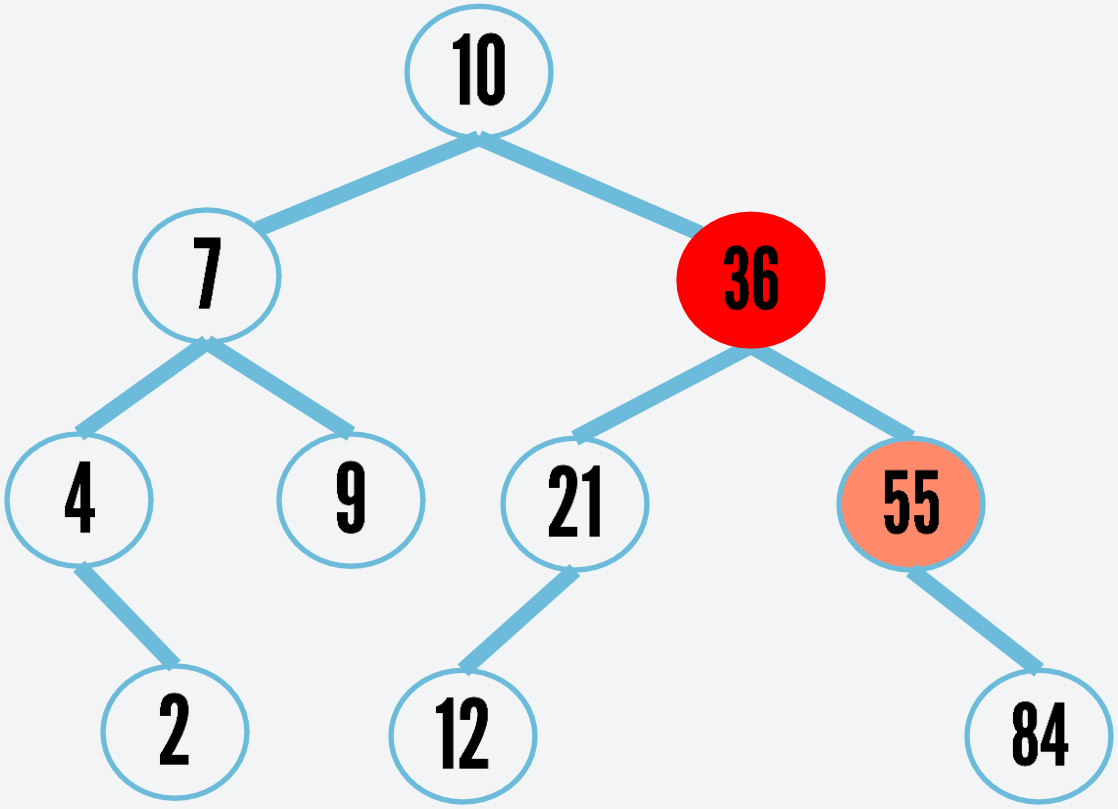
Node has two children

delete(36)



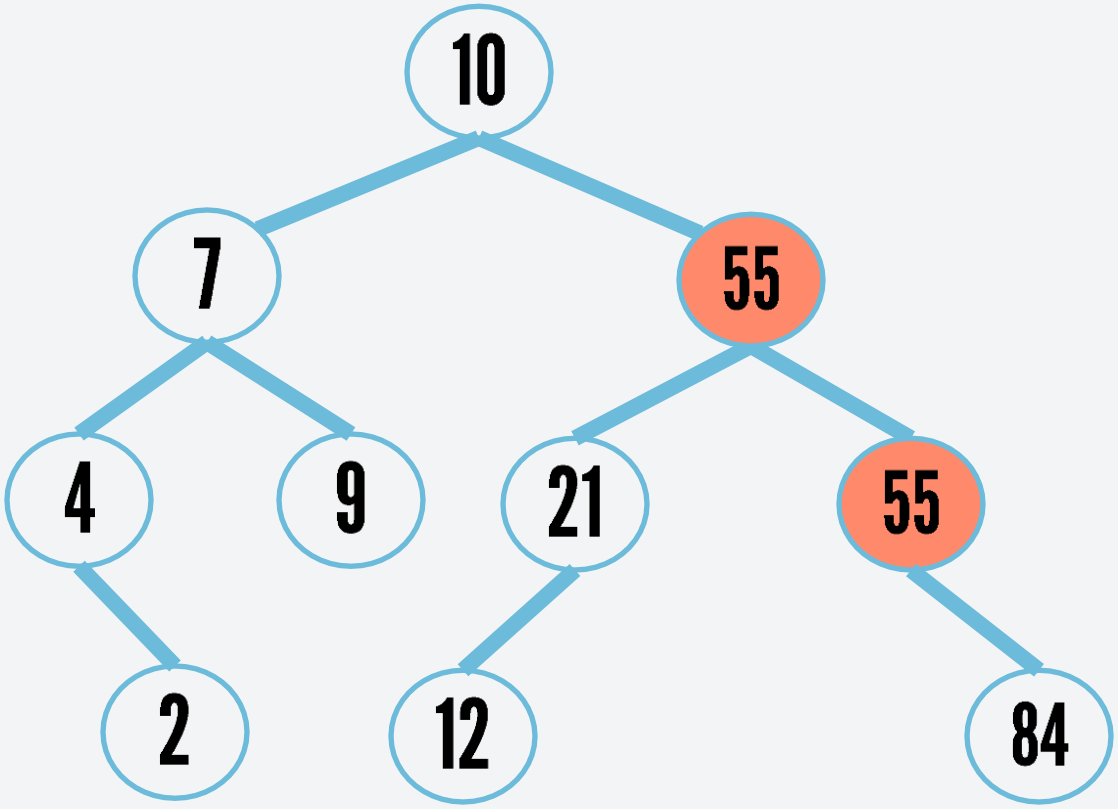
Node has two children

delete(36)



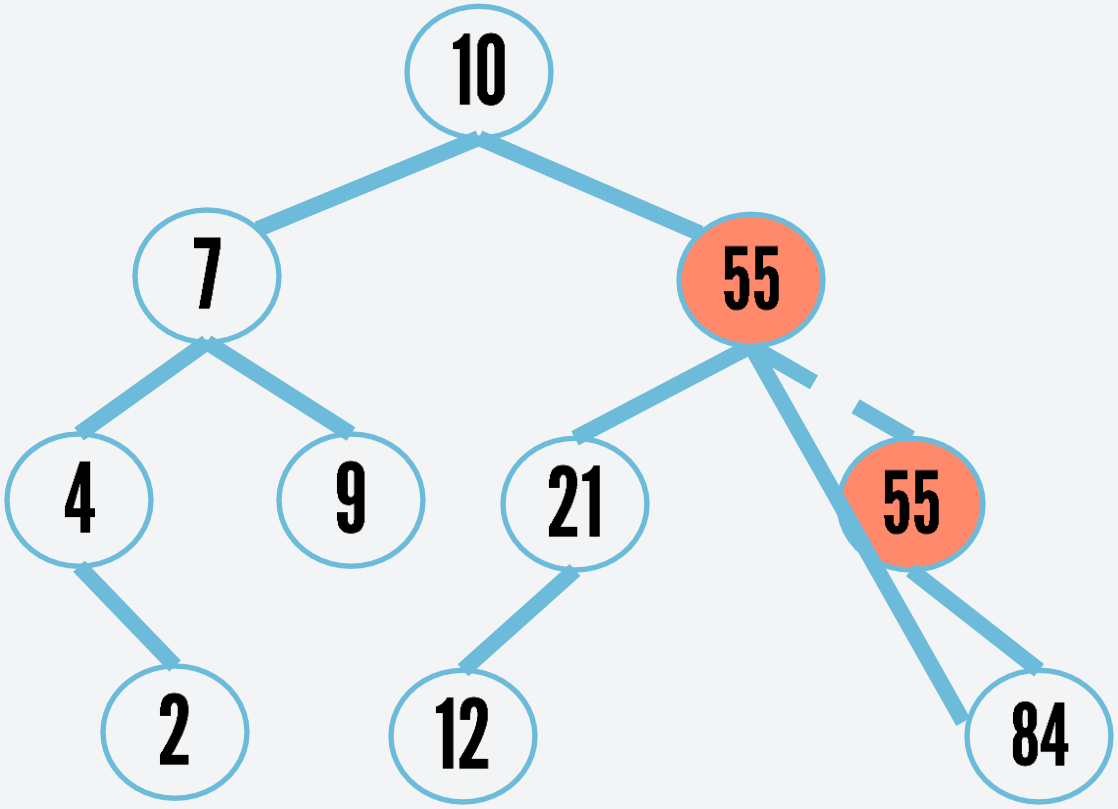
Node has two children

delete(36)



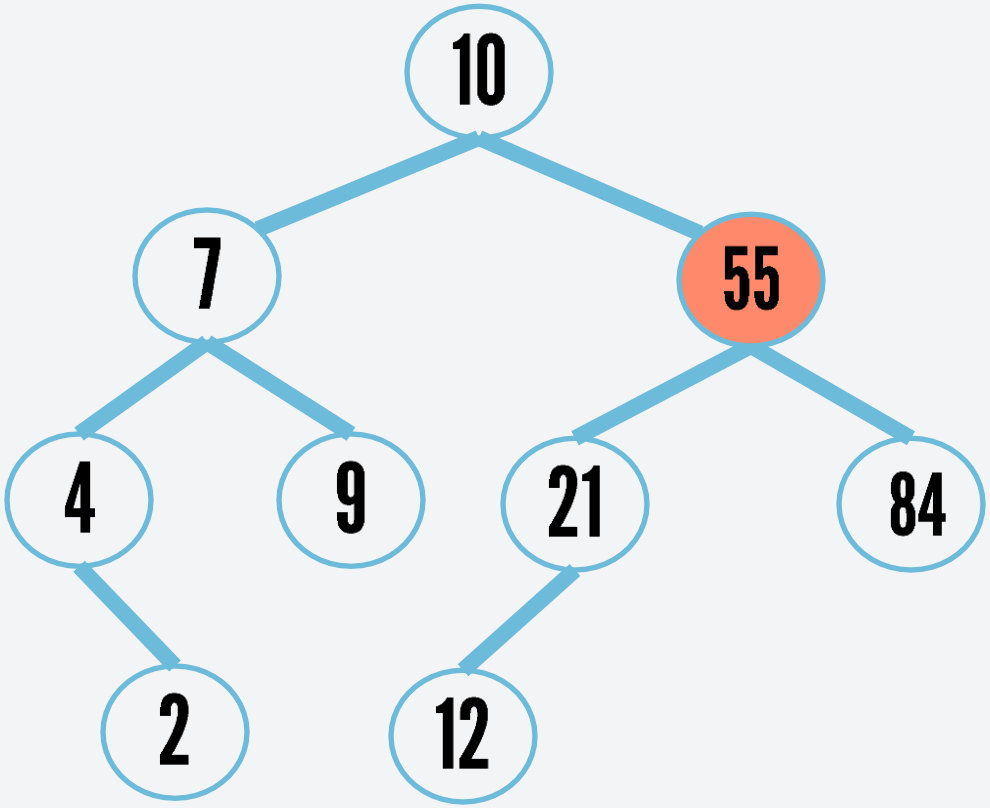
Node has two children

delete(36)



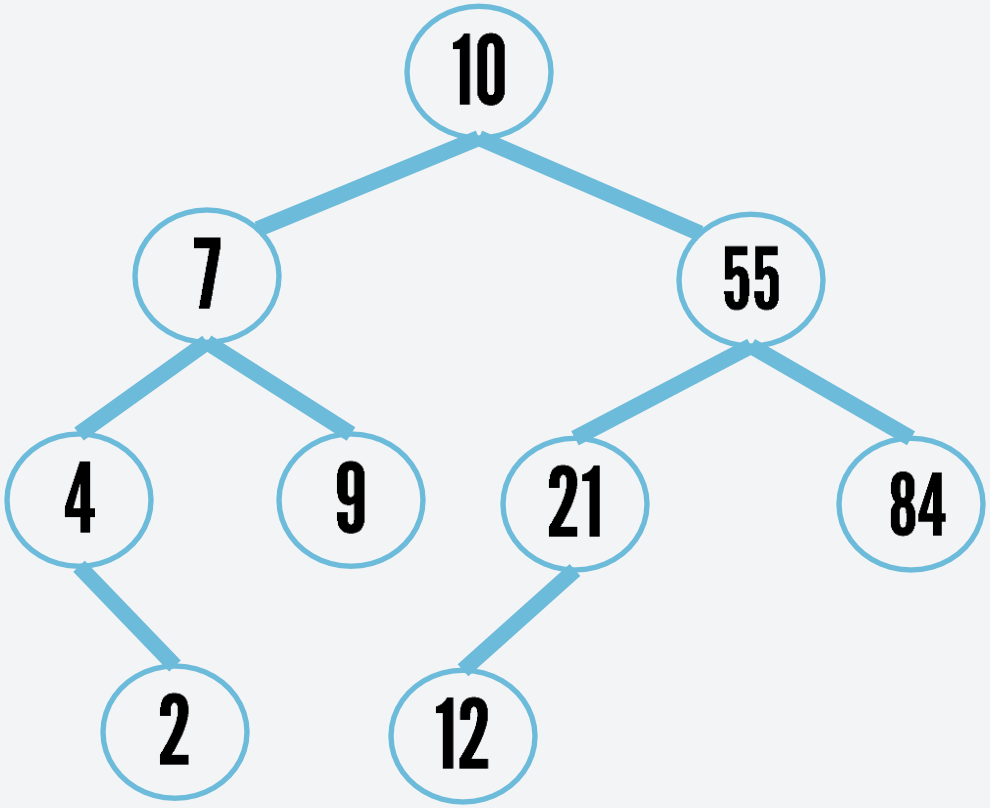
Node has two children

delete(36)



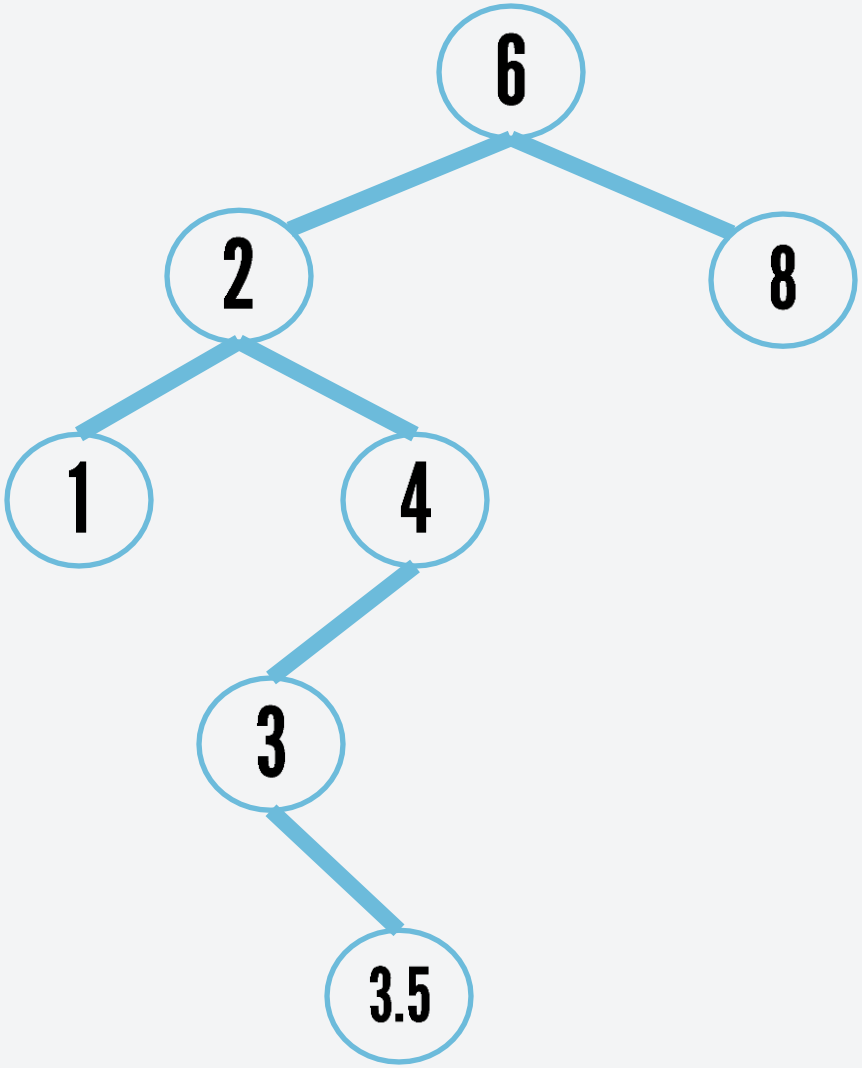
Node has two children

delete(36)



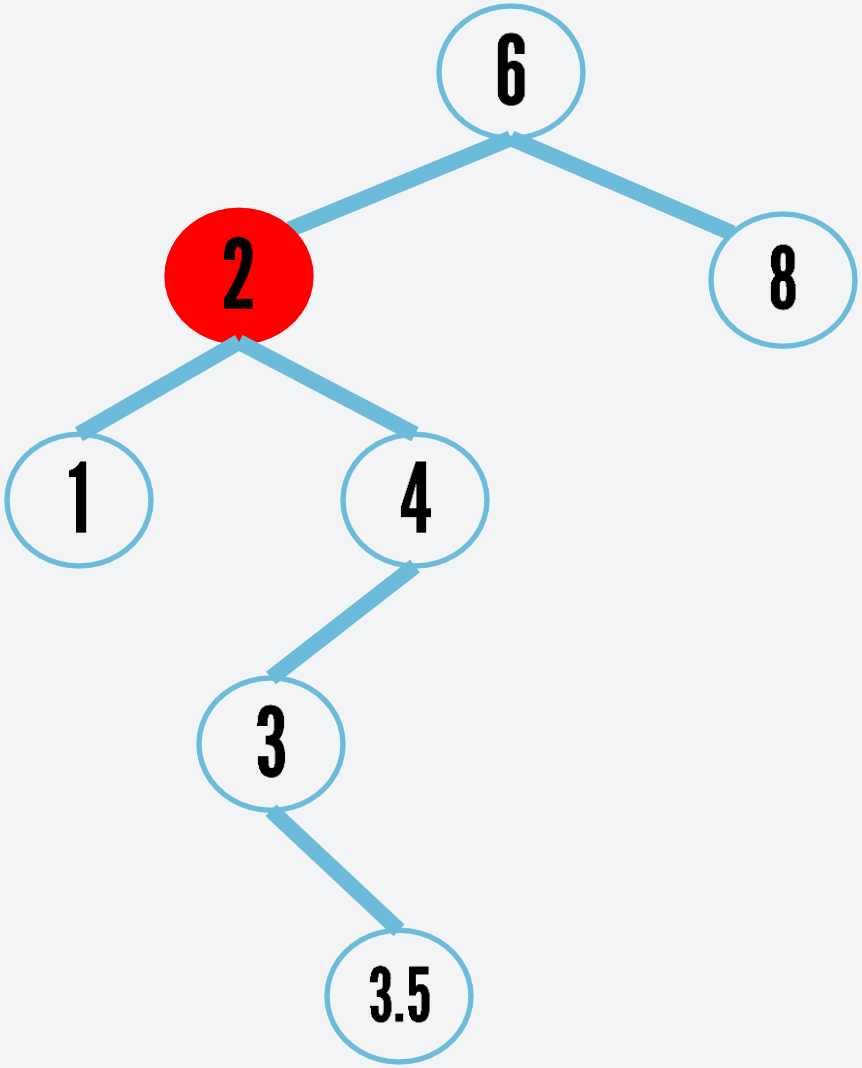
Node has two children

delete(2)



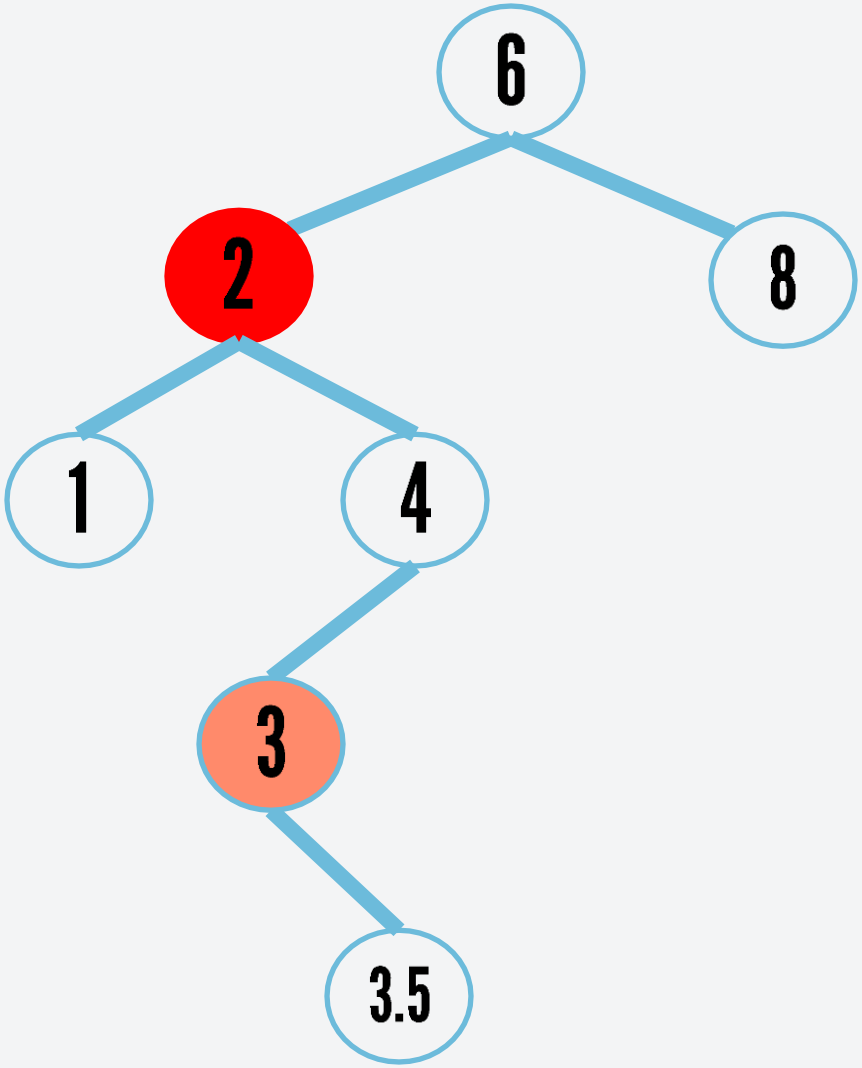
Node has two children

delete(2)



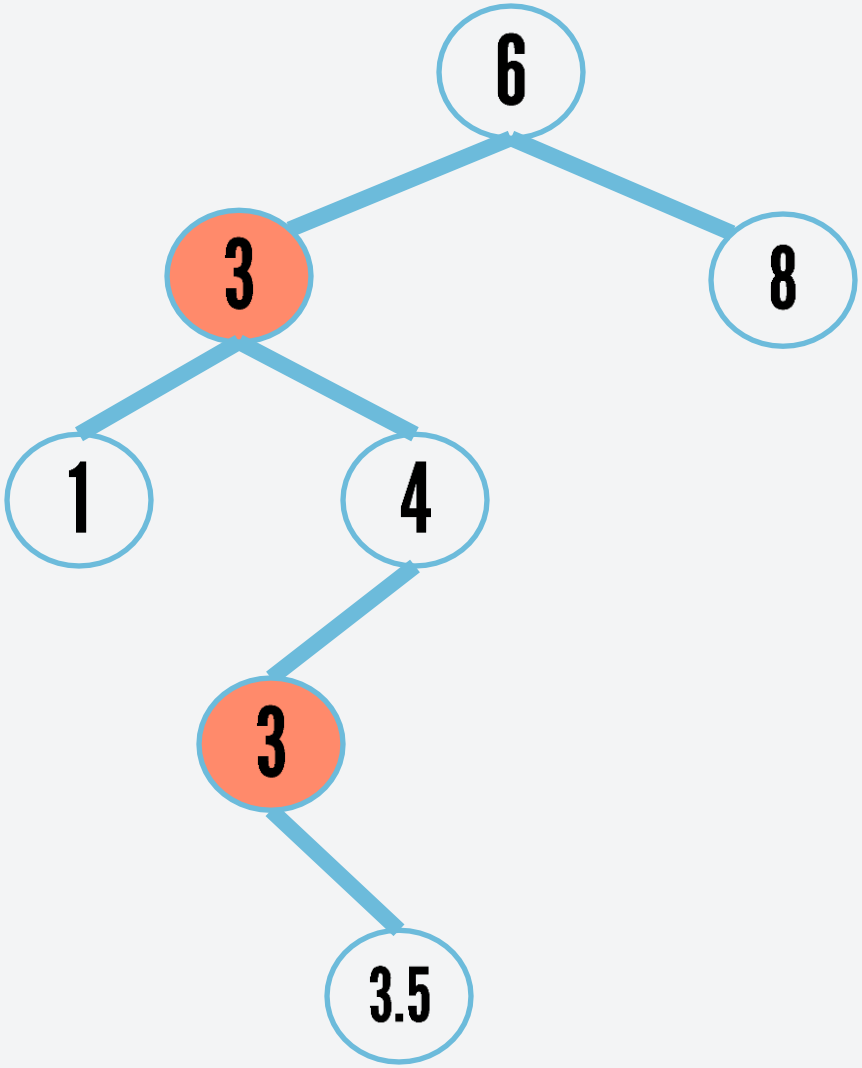
Node has two children

delete(2)



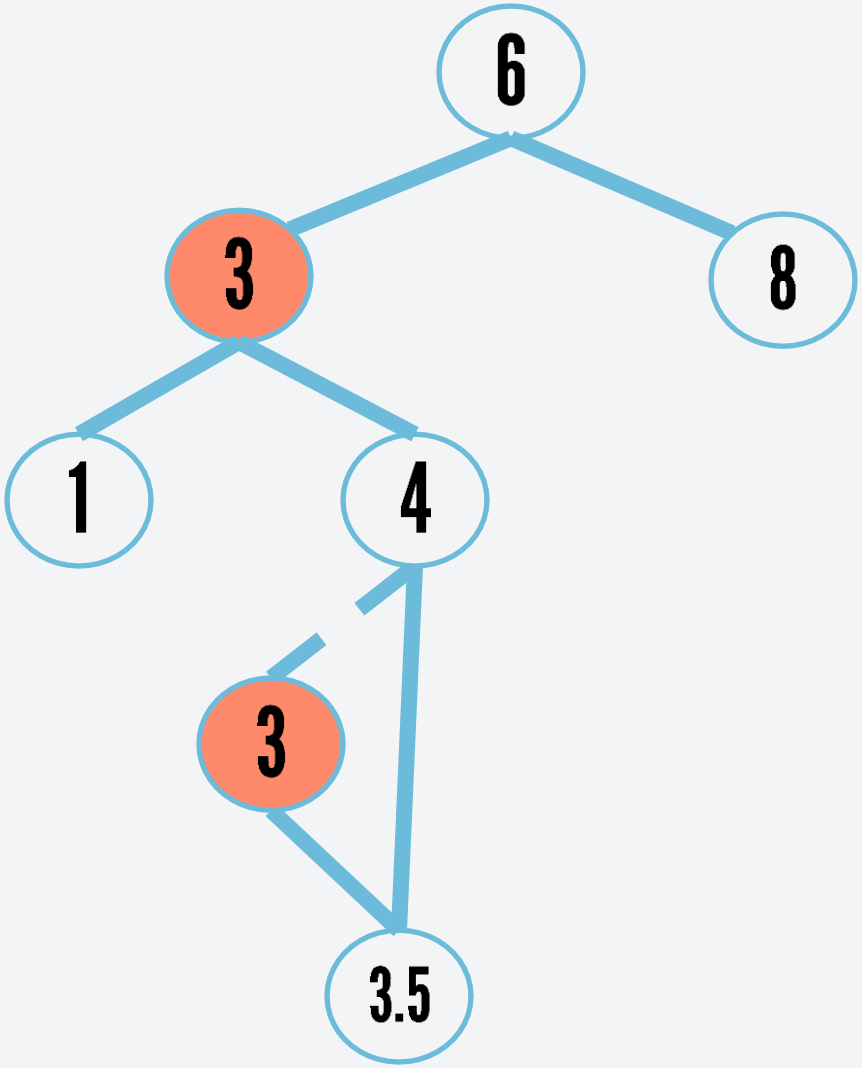
Node has two children

delete(2)



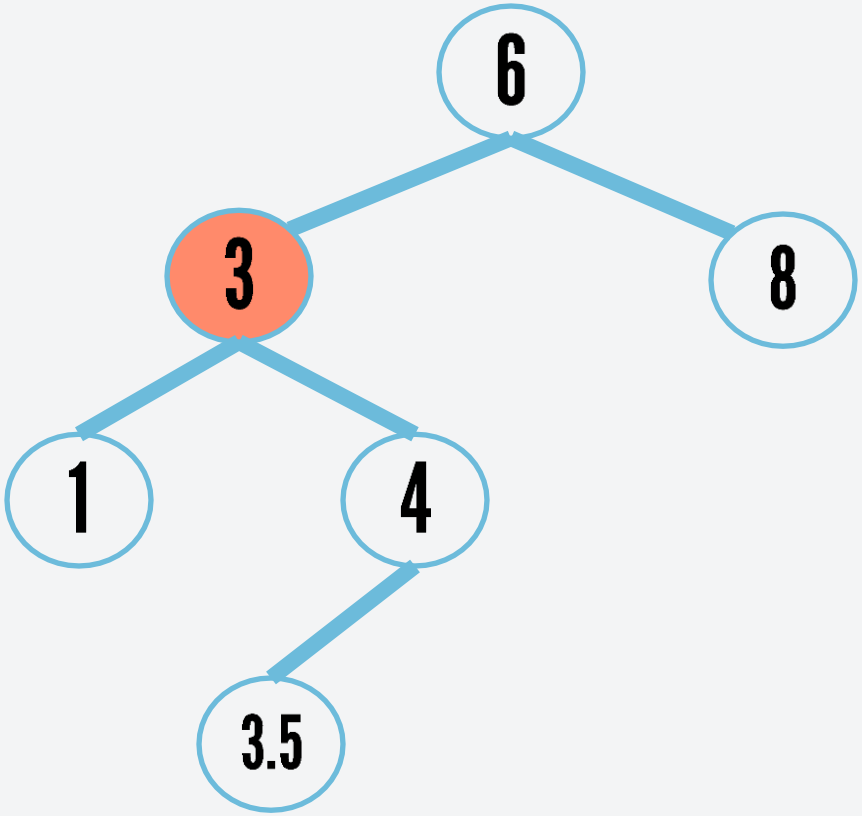
Node has two children

delete(2)



Node has two children

delete(2)



delete(2)

Node has two children