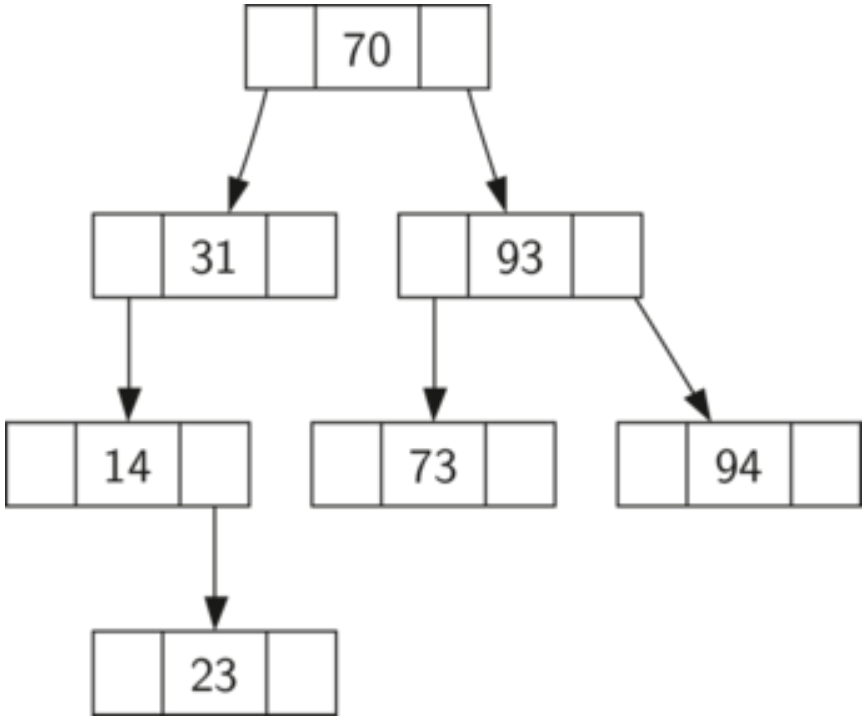
ต้นไม้ค้นหาแบบทวิภาค
(binary search tree)

คุณสมบัติ

- ทุก key ของ left subtree จะต้องมีค่าน้อยกว่า key ของ root เสมอ
- ทุก key ของ right subtree จะต้องมีค่ามากกว่า key ของ root เสมอ
- เราจะสร้าง search tree สำหรับ map

70,31,93,94,14,23,73



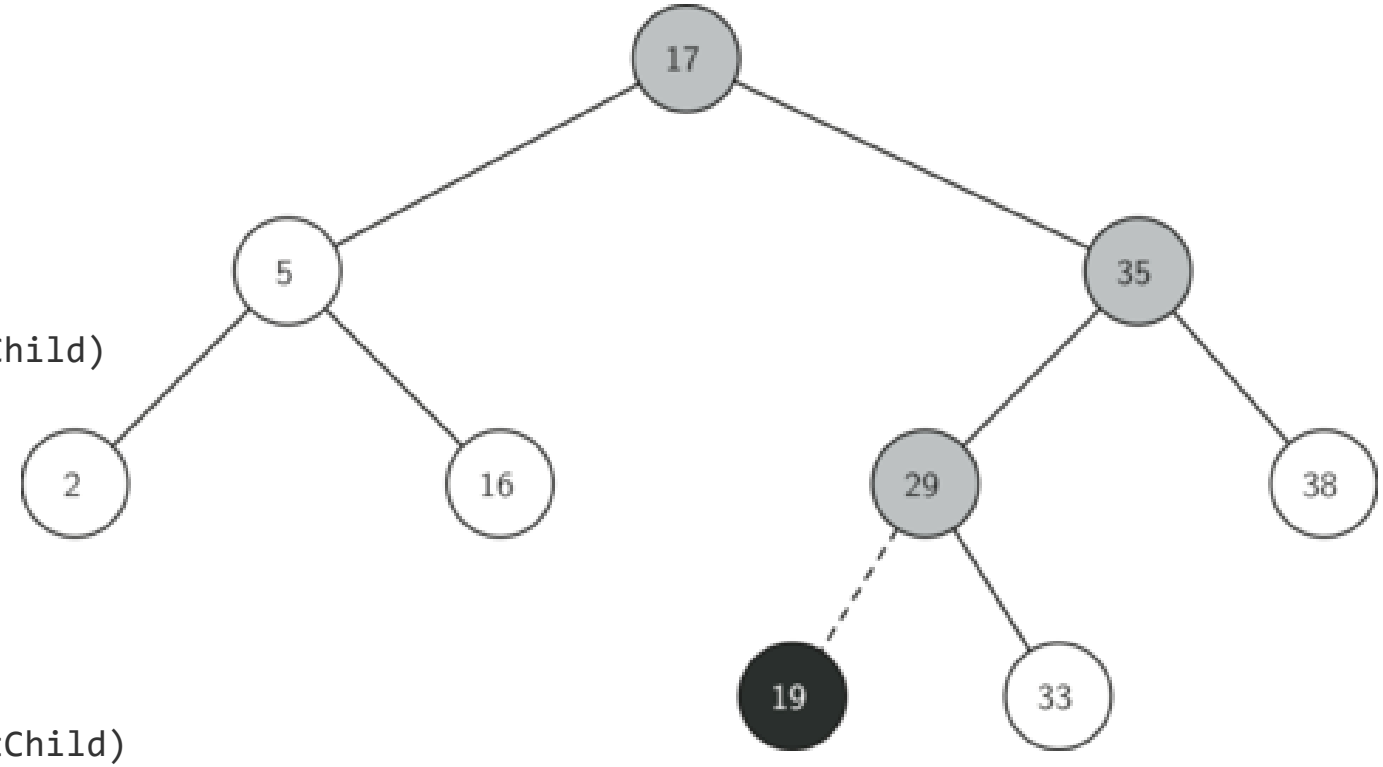
```
class BinarySearchTree:

    def __init__(self):
        self.root = None
        self.size = 0

    def length(self):
        return self.size

    def __len__(self):
        return self.size
```

```
class TreeNode:
   def __init__(self,key,val,left=None,right=None,
                                        parent=None):
        self.key = key
        self.payload = val
        self.leftChild = left
        self.rightChild = right
        self.parent = parent

    def hasLeftChild(self):
        return self.leftChild

    def hasRightChild(self):
        return self.rightChild

    def isLeftChild(self):
        return self.parent and self.parent.leftChild == self

    def isRightChild(self):
        return self.parent and self.parent.rightChild == self
```

```
    def isRoot(self):
        return not self.parent

    def isLeaf(self):
        return not (self.rightChild or self.leftChild)

    def hasAnyChildren(self):
        return self.rightChild or self.leftChild

    def hasBothChildren(self):
        return self.rightChild and self.leftChild

    def replaceNodeData(self,key,value,lc,rc):
        self.key = key
        self.payload = value
        self.leftChild = lc
        self.rightChild = rc
        if self.hasLeftChild():
            self.leftChild.parent = self
        if self.hasRightChild():
            self.rightChild.parent = self
```

```
def put(self,key,val):
    if self.root:
        self._put(key,val,self.root)
    else:
        self.root = TreeNode(key,val)
    self.size = self.size + 1


def _put(self,key,val,currentNode):
    if key < currentNode.key:
        if currentNode.hasLeftChild():
                self._put(key,val,currentNode.leftChild)
        else:
                currentNode.leftChild =
TreeNode(key,val,parent=currentNode)
    else:
        if currentNode.hasRightChild():
                self._put(key,val,currentNode.rightChild)
        else:
                currentNode.rightChild =
TreeNode(key,val,parent=currentNode)
```

```
def __setitem__(self,k,v):
    self.put(k,v)
```



```
def get(self,key):
    if self.root:
        res = self._get(key,self.root)
        if res:
                return res.payload
        else:
                return None
    else:
        return None

def _get(self,key,currentNode):
    if not currentNode:
        return None
    elif currentNode.key == key:
        return currentNode
    elif key < currentNode.key:
        return self._get(key,currentNode.leftChild)
    else:
        return self._get(key,currentNode.rightChild)
```

```
def __getitem__(self,key):
   return self.get(key)

def __contains__(self,key):
    if self._get(key,self.root):
        return True
    else:
        return False
```

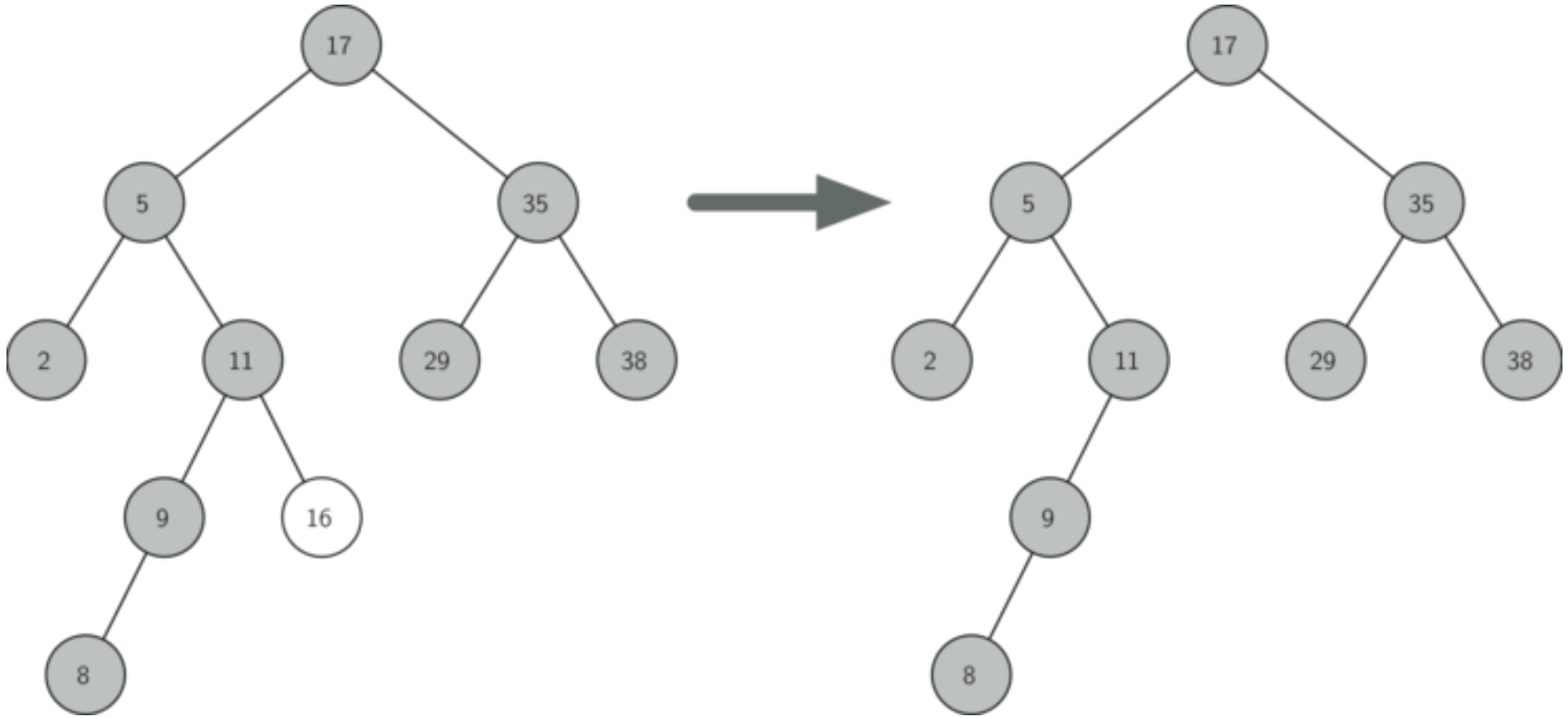
```
if 'Northfield' in myZipTree:
    print("oom ya ya")
```

```
def delete(self,key):
    if self.size > 1:
        nodeToRemove = self._get(key,self.root)
        if nodeToRemove:
            self.remove(nodeToRemove)
            self.size = self.size-1
        else:
            raise KeyError('Error, key not in tree')
    elif self.size == 1 and self.root.key == key:
        self.root = None
        self.size = self.size - 1
    else:
        raise KeyError('Error, key not in tree')

def __delitem__(self,key):
    self.delete(key)
```

- remove มี 3 กรณี
  - node ที่ต้องการ remove ไม่มีลูก
  - node ที่ต้องการ remove มีลูก 1 node
  - node ที่ต้องการ remove มีลูก 2 nodes

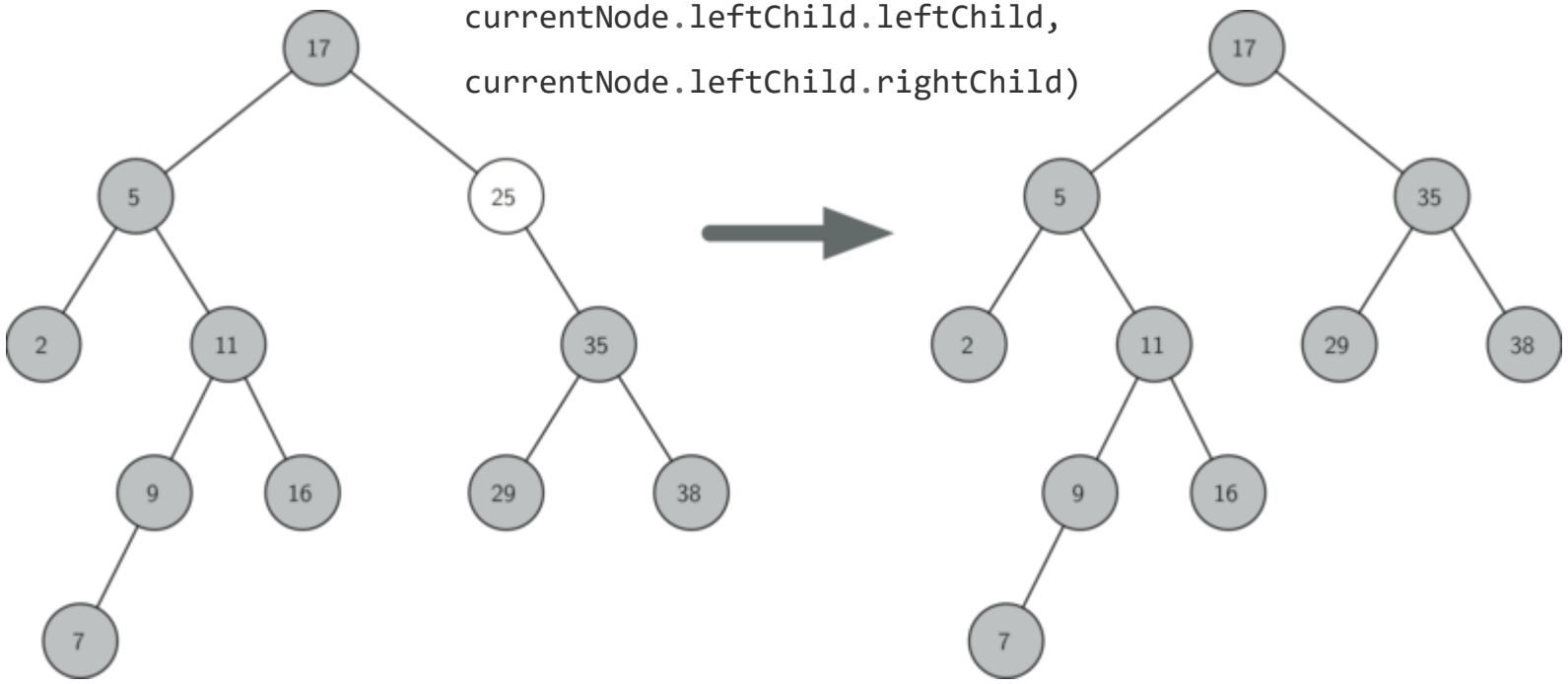
กรณีเป็น leaf node

```
if currentNode.isLeaf():
    if currentNode == currentNode.parent.leftChild:
        currentNode.parent.leftChild = None
    else:
        currentNode.parent.rightChild = None
```



```
else: # this node has one child
   if currentNode.hasLeftChild():
      if currentNode.isLeftChild():
          currentNode.leftChild.parent = currentNode.parent
          currentNode.parent.leftChild = currentNode.leftChild
      elif currentNode.isRightChild():
          currentNode.leftChild.parent = currentNode.parent
          currentNode.parent.rightChild = currentNode.leftChild
      else:
          currentNode.replaceNodeData(currentNode.leftChild.key,
                                      currentNode.leftChild.payload,
                                      currentNode.leftChild.leftChild,
                                      currentNode.leftChild.rightChild)
```

```
else:
      if currentNode.isLeftChild():
          currentNode.rightChild.parent = currentNode.parent
          currentNode.parent.leftChild = currentNode.rightChild
      elif currentNode.isRightChild():
          currentNode.rightChild.parent = currentNode.parent
          currentNode.parent.rightChild = currentNode.rightChild
      else:
          currentNode.replaceNodeData(currentNode.rightChild.key,
                                      currentNode.rightChild.payload,
                                      currentNode.rightChild.leftChild,
                                      currentNode.rightChild.rightChild)
```

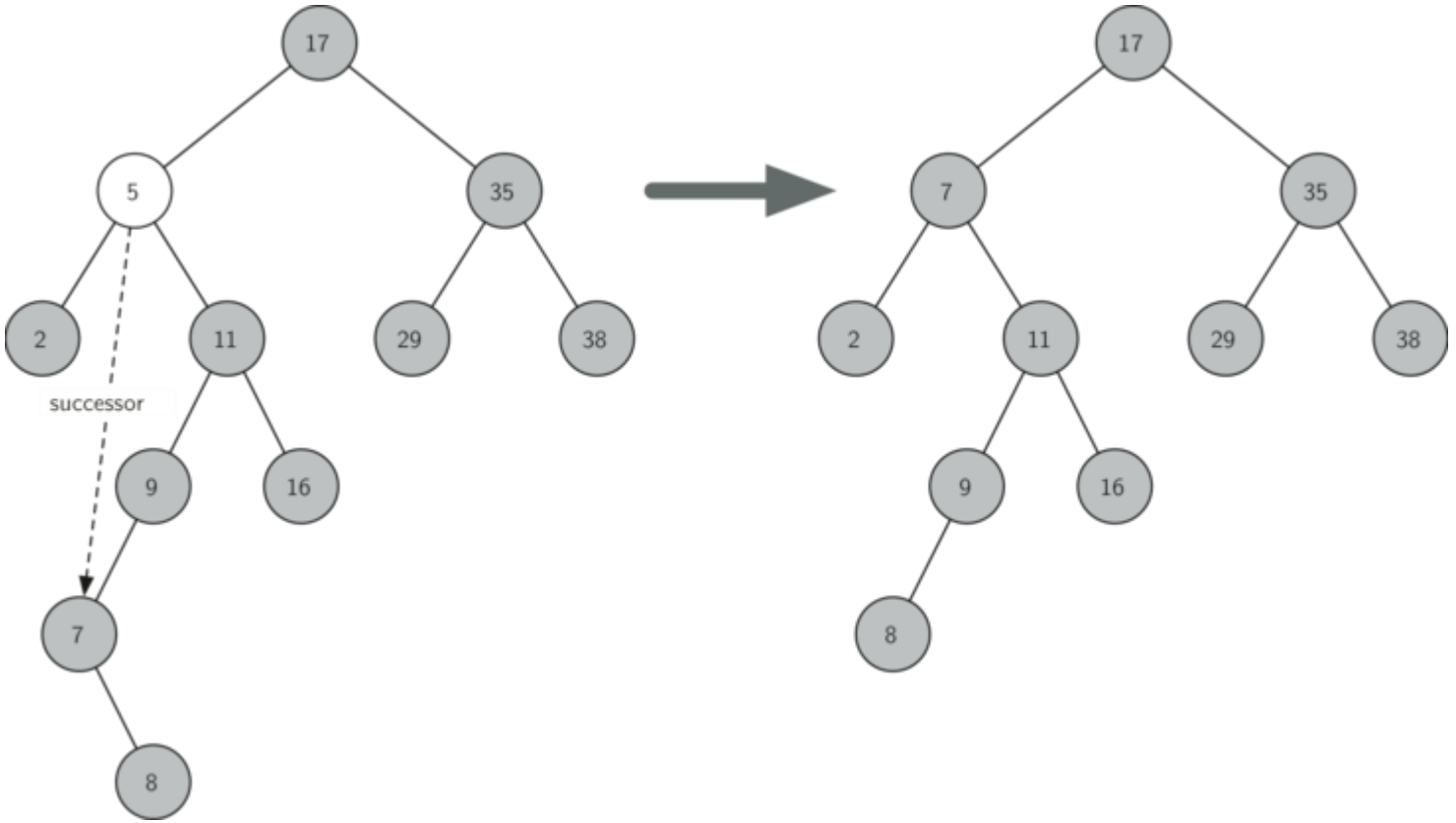


กรณีมีลูก 1 ให้เอาลูกมาแทนตำแหน่งตัวเอง

กรณีมีลูก 2

- หาตัวแทนที่ยังรักษาคุณสมบัติของ search tree
  - ทุก key ของ left subtree จะต้องมีค่าน้อยกว่า key ของ root เสมอ
  - ทุก key ของ right subtree จะต้องมีค่ามากกว่า key ของ root เสมอ
- ตัวแทนคือ node ที่มี key มากเป็นลำดับถัดไปจาก key ของ node ที่ต้องการ remove (successor)
- ตัวแทนจะมีลูกไม่เกิน 1 ตัว
  - เก็บตัวแทนไว้
  - remove ตัวแทนที่ตำแหน่งเก่า
  - ใส่ตัวแทนกลับเข้าไปที่ตำแหน่ง node ที่ต้องการ remove

```
elif currentNode.hasBothChildren(): #interior
        succ = currentNode.findSuccessor()
        succ.spliceOut()
        currentNode.key = succ.key
        currentNode.payload = succ.payload
```



findSuccessor()

- พิจารณาในกรณีทั่วไป
- ถ้ามี right child แล้ว successor คือ node ที่มีค่า key น้อยสุดของ right subtree
- ถ้าไม่มี right child และตัวมันเองเป็น left child แล้ว successor คือ parent
- ถ้าไม่มี right child และตัวมันเองเป็น right child แล้ว successor คือ successor ของ parent ที่ไม่รวมตัวมันเอง

```
def findSuccessor(self):
    succ = None
    if self.hasRightChild():
        succ = self.rightChild.findMin()
    else:
        if self.parent:
            if self.isLeftChild():
                succ = self.parent
            else:
                self.parent.rightChild = None
                succ = self.parent.findSuccessor()
                self.parent.rightChild = self
    return succ

def findMin(self):
    current = self
    while current.hasLeftChild():
        current = current.leftChild
    return current
```

```
def spliceOut(self):
  if self.isLeaf():
    if self.isLeftChild():
      self.parent.leftChild = None
    else:
      self.parent.rightChild = None
  elif self.hasAnyChildren():
    if self.hasLeftChild():
      if self.isLeftChild():
        self.parent.leftChild = self.leftChild
      else:
        self.parent.rightChild = self.leftChild
      self.leftChild.parent = self.parent
    else:
      if self.isLeftChild():
        self.parent.leftChild = self.rightChild
      else:
        self.parent.rightChild = self.rightChild
      self.rightChild.parent = self.parent
```