```
Search(self, key):
    if ______ :  # 找不到的情況
        return None
    if ______ :  # 找到 ''  ← root 就是
                             root 不是
        return key
    if ______ :
        return root
```

⇩

```
Search ( self , key ):
    if self.key < self.root :
        跟 Left child 比較
    if self.key > self.root :
        跟 right 比較
    if self.key = self.root:
        return ~~key~~ root
    else:
        return None
```

Loop ⇒
=return Search (i)
# i 是 key 的話就沒意義
∴ 需另一變數 root 來
比較。

⇩

```
Search (self, root, key)
  if self.key < self.root:
      return Search(leftchild, key).
  if self.key > self.root:
      return Search(rightchild . key)

  if self.key = self.root:
      return self.root
  else:
      return None.
```

```
Class Solution:
    def oo ():
    def xx ():
```

a.search(

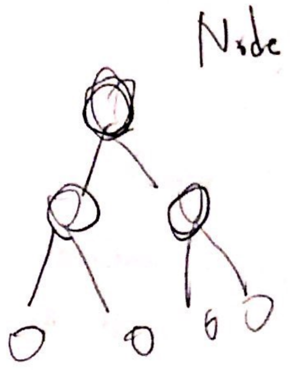
此頁空白

```
def insert ( key ):
    if ____ :                # root 為空
        insert root
    if key>root:             # root 不空, 插
        insert (右,key)        左/右邊

    if key < root:
        insert (左,key)
```

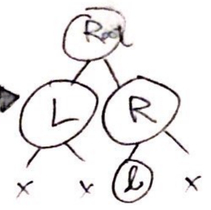
⇩

```
                     BTS
def insert(self, root, key)
    if self.root is None:
root.val ~~root~~ root <- self.root = node(key)
    else:
        if  key > ~~roots~~.root.val:
            { 判斷是否 None.
                  是 => root.right = node(key)
                  否 => insert(root.rights, key)
        else  key < root.val:

            {     "
```

```
class Node :
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key
```



```
delete (self, root, key):
    if No left/right:
    elif has one left/right child:
    else has both left/right child:
```



delete Ⓛ: Root.left = None
delete Ⓡ: Root.right = l.val

```
delete (self, root, key):
    if root is None: #沒找到
        return False
    elif key > root.val:
        root.right = $\overset{}{\text{delete}}$(root.right, key)
                     self.
    elif key < root.val:
        root.left = self.delete(root.left, key)
```

跟insert一樣, 先找目標位置

```
else: (#找到, 開的刪除) # root.val == key
    if root.left is None:
        rep = root.right
        root = None
    elif root.right is None:
        ~~root.left = root~~ rep = root.lelf
                            (root)
        root = None
```

有1子情況

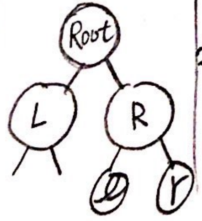
把子節拿起來

把節刪掉

一樣意思, 只是replace不同

```
else: #有2子情況
```

1. 刪除Root (代替)
2. 補Root:
   從Root的右子集找min or
   " 左 " max 補



```
else: #2子情況
    root.key = rep.key → 從左邊找max:
    def max( , root): if root.left is not None
                          root = self.max(root.left)
                      else:
                          return root
```

Result = None
∴ 改while 迴圈