链表

- 尾插法是正序

```
class Node:
    def __init__(self, val=0):
        self.val = val
        self.next = None


# 头插法
def create_linklist_head(nums):
    head = Node(nums[0])
    for element in nums[1:]:
        node = Node(element)
        node.next = head
        head = node
    return head


def create_linklist_tail(nums):
    head = Node(nums[0])
    tail = head
    for element in nums[1:]:
        node = Node(element)
        tail.next = node
        tail = node
    return head
```

二叉树

```python
class BiTreeNode:
    def __init__(self, data):
        self.data = data
        self.bf = None
        self.lchild = None
        self.rchild = None
        self.parent = None

class BST:
    def __init__(self, nums=None):
        self.root = None
        if nums:
            for num in nums:
                self.insert_no_rec(num)

    # 递归法
    def insert(self, node, val):
        if not node:
            node = BiTreeNode(val)
        elif val < node.data:
            node.lchild = self.insert(node.lchild, val)
            node.lchild.parent = node
        elif val > node.data:
            node.rchild = self.insert(node.rchild, val)
            node.rchild.parent = node
        return node

    # 非递归法
    def insert_no_rec(self, val):
        p = self.root
        if not p:
            self.root = BiTreeNode(val)
            return
        while True:
            if val < p.data:
                if p.lchild:
                    p = p.lchild
                else:
                    p.lchild = BiTreeNode(val)
                    p.lchild.parent = p
                    return
            elif val > p.data:
                if p.rchild:
                    p = p.rchild
                else:
                    p.rchild = BiTreeNode(val)
                    p.rchild.parent = p
                    return
            else:
                return
```

```
def query(self, node, val):
    if not node:
        return None
    if node.data < val:
        return self.query(node.rchild, val)
    elif node.data > val:
        return self.query(node.lchild, val)
    else:
        return node

def query_no_rec(self, val):
    p = self.root
    while p:
        if p.data < val:
            p = p.rchild
        elif p.data > val:
            p = p.lchild
        else:
            return p
    return None

# 前序遍历：根左右  EACBDGF
def pre_order(self, root):
    if root:
        print(root.data, end=', ')
        self.pre_order(root.lchild)
        self.pre_order(root.rchild)

# 中序遍历：左根右  ABCDEFG
def in_order(self, root):
    if root:
        self.in_order(root.lchild)
        print(root.data, end=', ')
        self.in_order(root.rchild)

# 后序遍历：左右根  BDCAFGE
def post_order(self, root):
    if root:
        self.post_order(root.lchild)
        self.post_order(root.rchild)
        print(root.data, end=', ')

def __remove_node_1(self, node):
    # 情况1：node是叶子节点
    if not node.parent:
        self.root = None
    # node是父亲的左孩子
    if node == node.parent.lchild:
        node.parent.lchild = None
        # node.parent = None
    else:  # 右孩子
```

```
            node.parent.rchild = None

    def __remove_node_21(self, node):
        # 情况2：node只有一个左孩子
        if not node.parent:
            self.root = node.lchild
            node.lchild.parent = None
        elif node == node.parent.lchild:
            node.parent.lchild = node.lchild
            node.lchild.parent = root.parent
        else:
            node.parent.rchild = node.lchild
            node.lchild.parent = node.parent

    def __remove_node_22(self, node):
        # 情况2：node只有一个右孩子
        if not node.parent:
            self.root = node.rchild
        elif node == node.parent.lchild:
            node.parent.lchild = node.rchild
            node.rchild.parent = root.parent
        else:
            node.parent.rchild = node.rchild
            node.rchild.parent = node.parent

    #
    def delete(self, val):
        if self.root:  # 非空树
            node = self.query_no_rec(val)
            if not node:
                return False
            if not node.lchild and not node.rchild:
                self.__remove_node_1(node)
            elif not node.rchild:  # 只有左孩子
                self.__remove_node_21(node)
            elif not node.lchild:  # 只有右孩子
                self.__remove_node_22(node)
            else:
                # 两个孩子都有
                min_node = node.rchild
                while min_node.lchild:
                    # 此时min_node为右子树里最小的值
                    min_node = min_node.lchild
                node.data = min_node.data
                if min_node.rchild:
                    self.__remove_node_22(node)
                else:  # 叶子节点
                    self.__remove_node_1(node)


tree = BST([1, 4, 2, 5, 3, 8, 6, 9, 7])
```

```
tree.in_order(tree.root)
print()
tree.delete(4)
tree.in_order(tree.root)
```