2018年8月8日 星期三

数据结构第五次实验报告

5-1.py

```
class Node():
    '''树的节点'''

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

class BST():
    '''二叉搜索树的实现'''

    def __init__(self, node=None):
        self.node = node
        self._size = 0

    def insert(self, item):
        '''插入节点'''
        def recurse(node):
            # 要插入的节点将在node的左子树上
            if item.data < node.data:
                if node.left == None: # 如果左子树为空，直接插入
                    node.left = item
                else: # 递归插入左子树
                    recurse(node.left)
            # 要插入的节点将在node的右子树上
            elif node.right == None: # 如果右子树为空，直接插入
                node.right = item
            else: # 递归插入右子树
                recurse(node.right)
        # 二叉搜索树为空
        if self.node is None:
            self.node = item
        else: # 不为空
            recurse(self.node)
        # 增加节点个数
        self._size += 1

    def find(self, value):
        '''根据给定的元素查找树的节点，注意返回的是字符串'''
        def recurve(node):
            if node is None:
```

```
                return 'Not found'
            elif node.data == value:
                return 'Found'
            elif node.data < value:
                return recurve(node.right)
            else:
                return recurve(node.left)
        if self._size > 0:
            return recurve(self.node)
        else:
            return 'Not found'

    def deleteNode(self, value):
        '''根据给定的元素值删除树的节点'''
        def findMin(root): # 寻找root子树中最小的节点
            if root.left:
                return findMin(root.left)
            else:
                return root

        def delete(root, value):
            if root is None:
                return
            elif root.data > value: # 被删除的节点在左子树中
                root.left = delete(root.left, value)
            elif root.data < value: # 被删除的节点在右子树中
                root.right = delete(root.right, value)
            else: # 相等的情况：即找到要删除的节点
                if root.left and root.right:
                    # 左右节点均存在
                    temp = findMin(root.right) # 在右子树中查找最
小节点
                    root.data = temp.data
                    # 在右子树中删除该节点
                    root.right = delete(root.right, temp.data)
                elif root.left is None:
                    # 左子树为空
                    root = root.right
                else: # 右子树为空
                    root = root.left
            return root

        self.node = delete(self.node, value)

    def __str__(self):
        '''返回树的字符串表示，  按照先序遍历的顺序'''
```

```
        s = []
        def recurse(node):
            if node is None:
                return
            recurse(node.left)
            s.append(node.data)
            recurse(node.right)
        recurse(self.node)
        string = ''
        for i in s:
            string += str(i) + ' '
        return string


'''
该二叉搜索树为:
                    33
              22              88
        11                55    90
                    66              99
'''


if __name__ == '__main__':
    nlist = eval(input('输入序列: '))
    needfind = eval(input('需要查找的元素: '))
    needdel = eval(input('需要删除的元素: '))
    bst = BST()
    for i in nlist:
        bst.insert(Node(i))
    print()
    print('要查找的元素%d: ' % needfind, end='')
    print(bst.find(needfind))
    print('删除元素%d前的二叉排序树为: ' % needdel, end='')
    print(bst)
    bst.deleteNode(needdel)
    print('删除元素%d后二叉排序树为: ' % needdel, end='')
    print(bst)
```

```
In [41]: runfile('/Users/zhujun/Downloads/USTC/专业补课/DS/DS-experiment/5-1.py', wdir='/Users/
zhujun/Downloads/USTC/专业补课/DS/DS-experiment')

输入序列: 33,88,22,55,90,11,66,99

需要查找的元素: 78

需要删除的元素: 90

要查找的元素78: Not found
删除元素90前的二叉排序树为: 11 22 33 55 66 88 90 99
删除元素90后二叉排序树为: 11 22 33 55 66 88 99
```

2018年8月8日 星期三

5-2.py

```
nlist = eval(input())
def linkHash(nlist, p):
    '''链地址法解决哈希冲突'''
    hl = [ [] for i in range(p) ]
    for i in nlist:
        hl[i%p].append(i)
    return hl

def linearHash(nlist, p):
    '''线性探测解决哈希冲突'''
    ll = [ 0 for i in range(p) ]
    for i in nlist:
        k = 0
        if ll[i%p] == 0:
            ll[i%p] = i
        else:
            while ll[(i%p+k)%p] == 0:
                k += 1
            if k >= p:
                raise ValueError('哈希表已满')
            ll[(i%p+k)%p] = i%p
    return ll

# 测试链哈希表
it1 = linkHash(nlist, 11)
k = 0
print('linkHash:')
for i in it1:
    print(k, end=':')
    for j in i:
        print(j, end=' ')
    print()
    k += 1

# 测试线性再散列哈希表
it2 = linearHash(nlist, 11)
print('linearHash:', end='')
print(it2)
```

```
In [25]: runfile('/Users/zhujun/Downloads/USTC/专业补课/DS/DS-experiment/5-2.py', wdir='/Users/
zhujun/Downloads/USTC/专业补课/DS/DS-experiment')

19,14,23,1,68,20,84,27,56,11,10,79
linkHash:
0:11
1:23 1 56
2:68 79
3:14
4:
5:27
6:
7:84
8:19
9:20
10:10
linearHash:[11, 1, 2, 14, 0, 27, 0, 84, 19, 20, 10]
```