寻找自我的博客

python写数据结构

分类： Python 2012-09-01 01:25 68人阅读 评论(0) 收藏 举报

栈

stack.py

```
class Stack:
    def __init__(self,size=20):
        self.stack= []
        self.size= size;
        self.top= -1

    def setSize(self,size):
        self.size=size;

    def push(self,element):
        if self.isFull():
            raise "StackOverflow"
        else:
            self.stack.append(element)
            self.top = self.top + 1

    def pop(self):
        if self.isEmpty():
            raise "StackUnderflow"
        else:
            element=self.stack[-1]
            self.top=self.top-1;
            del self.stack[-1]
            return element

    def Top(self):
        return self.top

    def empty(self):
        self.stack=[]
        self.top=-1

    def isEmpty(self):
        if self.top == -1:
            return True
        else:
            return False

    def isFull(self):
        if self.top == self.size-1:
            return True
        else:
            return False

if  __name__ == "__main__":

    stack=Stack()

    for i in range(10):
        stack.push(i)
    print stack.Top()
    for i in range(10):
       print stack.pop()

    stack.empty()
    print stack.Top()
```

队列

queue.py

```
class Queue:
    def __init__(self,size=20):
        self.queue=[]
        self.size=size
        self.end=-1

    def setSize(self,size):
        self.size=size

    def In(self,element):
        if self.end < self.size -1:
            self.queue.append(element)
            self.end = self.end + 1
        else:
            raise "QueueFull"

    def Out(self):
        if self.end != -1:
            element = self.queue[0]
            self.queue=self.queue[1:]
            self.end = self.end-1
            return element
        else:
            raise "QueueEmpty"

    def End(self):
        return self.end

    def empty(self):
        self.queue=[]
        self.end=-1

if __name__ == "__main__":

    queue=Queue()
    for i in range(10):
        queue.In(i)
    print queue.End()

    for i in range(10):
        print queue.Out()
```

二叉树

btree.py

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

    def insertLeft(self,value):
        self.left=BTree(value)
        return self.left

    def insertRight(self,value):
        self.right=BTree(value)
        return self.right

    def show(self):
        print self.data

def preorder(node):
    if node.data:
        node.show()
        if node.left:
            preorder(node.left)
        if node.right:
            preorder(node.right)

def inorder(node):
    if node.data:
        if node.left:
            inorder(node.left)
        node.show()
        if node.right:
            inorder(node.right)

def postorder(node):
    if node.data:
        if node.left:
             postorder(node.left)
        if node.right:
            postorder(node.right)
        node.show()

if __name__ == "__main__":

    Root=BTree("root")
    A=Root.insertLeft("A")
    C=A.insertLeft("C")
    D=C.insertRight("D")
    F=D.insertLeft("F")
    G=D.insertRight("G")
    B=Root.insertRight("B")
    E=B.insertRight("E")

    print "pre-traversal"
    preorder(Root)

    print "in-traversal"
    inorder(Root)

    print "post-traversal"
    postorder(Root)
```

图

graph.py

```
def searchGraph(graph,start,end):
    results=[]
    generatePath(graph,[start],end,results)
    results.sort(lambda x,y:cmp(len(x),len(y)))
    return results

def generatePath(graph,path,end,results):
    state=path[-1]
    if state == end:
        results.append(path)
    else:
        for arc in graph[state]:
            if arc not in path:
                generatePath(graph,path+[arc],end,results)


if __name__ == "__main__":
        Graph={
              'A':['B','C','D'],
              'B':['E'],
              'C':['D','F'],
              'D':['B','E','G'],
              'E':[],
              'F':['D','G'],
              'G':['E']
              }
        r = searchGraph(Graph,'A','D')
        print "A to D"
        for i in r:
            print i

        r=searchGraph(Graph,'A','E')
        print "A to E"
        for i in r:
            print i
```

二分查找

binarysearch.py

```
def BinarySearch(l,key):
    low=0
    high=len(l)-1
    i=0
    while(low <= high):
        i = i+1
        mid = low + ((high-low)>>1)
        if(l[mid] < key):
            low = mid + 1
        elif (l[mid] > key):
            high = mid -1
        else:
            print "use %d times" % i
            return mid
    return -1

if __name__ == "__main__":
    l=[1,4,5,6,7,8,9,44,333,2233]
    print l
    print BinarySearch(l,4)
    print BinarySearch(l,44)
    print BinarySearch(l,8)
    print BinarySearch(l,2233)
    print BinarySearch(l,77)
```

堆排序

sort.py

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

    def insertLeft(self,value):
        self.left=BTree(value)
        return self.left

    def insertRight(self,value):
        self.right=BTree(value)
        return self.right

    def show(self):
        print self.data

def inorder(node):
    if node.data:
        if node.left:
            inorder(node.left)
        node.show()
        if node.right:
            inorder(node.right)


def rinorder(node):
    if node.data:
        if node.right:
            rinorder(node.right)
        node.show()
        if node.left:
            rinorder(node.left)

def insert(node,value):
    if value > node.data:
        if node.right:
            insert(node.right,value)
        else:
            node.insertRight(value)
    else:
        if node.left:
            insert(node.left,value)
        else:
            node.insertLeft(value)


if __name__ == "__main__":
```

```
l=[88,11,2,33,22,4,55,33,221,34]
Root=BTree(l[0])
node=Root
for i in range(1,len(l)):
    insert(Root,l[i])

print "1---->10"
inorder(Root)
print "10--->1"
rinorder(Root)
```