

王欢 学号: 220181499 github: [https://github.com/njustwh2014/data\\_structure\\_example](https://github.com/njustwh2014/data_structure_example)

## 线性结构

令 $A$ 是一个长度为 $n$ 的正整数序列。试设计一个时间和空间复杂度分别为 $O(n)$ 和 $O(1)$ 的算法，判断 $A$ 中是否存在这样的元素 $x$ ， $x$ 在序列中出现次数超过 $n/3$ 。若存在这样的 $x$ ，则将其输出。

```
def findA(A):
    #最多同时出现两个元素超过三分之一
    reg1=0;
    counter1=0;
    reg2=0;
    counter2=0;
    ret=[];
    for item in A:
        if(counter1==0 or item==reg1):
            counter1=counter1+1;
            reg1=item;
        elif(counter2==0 or item==reg2):
            counter2=counter2+1;
            reg2=item;
        else:
            counter1=counter1-1;
            counter2=counter2-1;
    counter1=0;
    counter2=0;
    for item in A:
        if(item==reg1):
            counter1=counter1+1;
        elif(item==reg2):
            counter2=counter2+1;
    if(counter1>len(A)/3):
        ret.append(reg1);
    if(counter2>len(A)/3):
        ret.append(reg2);
    return ret;
```

在长度为 $n$ 的一维数组 $A$ 中，数组元素为互不相同的整型数。若存在这样的数 $x$ ，它大于它左侧所有数，小于右侧所有数，则称 $x$ 为 $A$ 中的一个中间数。例如：若数组 $A=\{3, 1, 6, 4, 5, 7, 9, 8, 10, 14, 12\}$ ，则 $A$ 中有中间数7和10。试设计一个线性时间复杂度的算法，找出给定数组 $A$ 中的所有中间数。

```
import sys
def finMid(A):
    lenA=len(A);
    Amin=[sys.maxsize]*lenA;
    Amax=[0]*lenA;
    Amin[lenA-1]=A[lenA-1];
```

```

Amax[0]=A[0];
ret=[];
for i in range(lenA-2,-1,-1):
    if(A[i]<Amin[i+1]):
        Amin[i]=A[i];
    else:
        Amin[i]=Amin[i+1];

for i in range(1,lenA):
    if(A[i]>Amax[i-1]):
        Amax[i]=A[i];
    else:
        Amax[i]=Amax[i-1];
for i in range(lenA):
    if(Amax[i]==A[i] and Amin[i]==A[i]):
        ret.append(A[i])
return ret;

```

**S**是一个正整数序列，试设计一个算法，判断**S**能否被划分成**m**份，使得每份的和相等。若可以，给出划分出的**m**个序列。

例如，若**S**为[6, 1, 3, 7, 4, 4, 5, 4, 1, 1]，当**m**为3时，划分：[1, 1, 4, 6]，[5, 4, 3]，[4, 7, 1] 的每个部分的和相等（为 12）；当**m**为4时，划分：[1, 1, 7]，[4, 5]，[4, 4, 1]，[3, 6] 的每个部分的和相等（为 9）。

给定一个单链表**L**: **A0**→**A1**→...→**An-1**→**An**, 将它重排为: **A0**→**An**→**A1**→**An-1**→**A2**→**An-2**→...。要求原地（**in-place**）操作且不改变结点中的内容。例如：给定**1**→**2**→**3**→**4**，重排为**1**→**4**→**2**→**3**

对于一个单链表**L**，设计算法（原地）判断**L**中结点的值是否是对称的。例如：**1**→**2**→**3**→**4**→**5**→**4**→**3**→**2**→**1** 就是对称的（可以对**L**进行重构，判定原**L**是否是对称的）。

```

class Node():
    def __init__(self,data=0,next=0):
        self.data=data;
        self.next=next;

class LinkedList():
    def __init__(self):
        self.head=0;
        self.length=0;
    def is_empty(self):
        if(self.head==0):
            return True;
        else:
            return False;

    def get_item(self,data):

```

```
    if(self.is_empty()==True):
        print("The LinkedList is empty!");
        return -1;
    else:
        j=0;
        p=self.head;
        while(p.next!=0):
            if(data==p.data):
                return j;
            else:
                p=p.next;
                j=j+1;
        if (data == p.data):
            return j;
        print("Objects that do not exist in the linked list!");
        return -1;

def append(self,data):
    if(self.is_empty()==True):
        newNode=Node(data);
        self.head=newNode;
        self.length=self.length+1;
    else:
        newNode=Node(data);
        p=self.head;
        while(p.next!=0):
            p=p.next;
        p.next=newNode;
        self.length=self.length+1;

def insert(self,data,index):
    if(index<0 and index>self.length):
        print("the index is wrong!");
        return False;
    j=0;
    p=self.head;
    while(j<index):
        p=p.next;
        j=j+1;
    newNode=Node(data);
    pnext=p.next;
    p.next=newNode;
    newNode.next=pnext;
    self.length=self.length+1;
    return True;

def get_length(self):
    return self.length;

def delete(self,data):
    if(self.get_item(data)==-1):
        print("Objects that do not exist in the linked list!");
        return False;
    p=self.head;
```

```

    pfront=0;
    if(self.head.data==data):
        self.head=0;
        self.length=0;
        return True;
    pfront=p;
    p=p.next;
    while(p.next!=0):
        if(p.data==data):
            pfront.next=p.next;
            self.length=self.length-1;
            return True;
        else:
            pfront=p;
            p=p.next;
    if(p.data==data):
        pfront.next = p.next;
        self.length = self.length - 1;
        return True;
    return False;
def printAll(self):
    if(self.length==0):
        print("the linklist is empty!");
        return ;
    p=self.head;
    print("there are {} nodes:".format(self.length));
    while(p.next!=0):
        print(p.data,end=" ");
        p=p.next;
    print(p.data);
    return ;

def reorderList(self):
    if self.head == 0 or self.head.next == 0:
        return
    pre = self.head
    lat = self.head.next
    while lat != 0 and lat.next != 0:
        pre = pre.next
        lat = lat.next.next
    # self.printAll();
    p = pre.next
    pre.next = 0
    # reverse

    cur = 0
    while p != 0:
        q = p.next
        p.next = cur
        cur = p
        p = q
    # self.printAll();
    pre = self.head
    while pre != 0 and cur != 0:

```

```

        tmp = cur.next
        cur.next = pre.next
        pre.next = cur
        pre = pre.next.next
        cur = tmp
    # self.printAll();

def reverseLinkedList_self(self):
    curNode = self.head;
    prevNode = 0;
    nextNode = 0;
    reversedHead = 0;
    while (curNode != 0):
        nextNode = curNode.next;
        if (nextNode == 0):
            reversedHead = curNode;
        curNode.next = prevNode;
        prevNode = curNode;
        curNode = nextNode;
    self.head=reversedHead;

def reverseLinkedList(self,head):
    curNode=head;
    prevNode=0;
    nextNode=0;
    reversedHead=0;
    while(curNode!=0):
        nextNode=curNode.next;
        if(nextNode==0):
            reversedHead=curNode;
        curNode.next=prevNode;
        prevNode=curNode;
        curNode=nextNode;
    return reversedHead;

def reorderLinkedList(self):
    #1->2->3->4->5->None reorder 1->5->2->4->3->None
    #find mid Node
    #split LinkedList to two parts
    #reverse later LinkedList part
    #insert one by one
    if(self.length<2):
        return;
    slowNode=self.head;
    fastNode=self.head.next;
    while(fastNode!=0 and fastNode.next!=0):
        fastNode=fastNode.next.next;
        slowNode=slowNode.next;
    # find mid node
    midNode=slowNode.next;
    slowNode.next=0;
    midNode=self.reverseLinkedList(midNode);
    slowNode=self.head;
    tempSlow=0;

```

```

tempMid=0;
while(slowNode!=0 and midNode!=0):
    tempSlow=slowNode.next;
    slowNode.next=midNode;
    tempMid=midNode.next;
    midNode.next=tempSlow;
    slowNode=tempSlow;
    midNode=tempMid;
def SymmetryLinkedList(self):
    if(self.length==0):
        return True;
    j=0;
    stack_prev=stack();
    p=self.head;
    jIndex=(int)(self.length/2);
    while(j<jIndex):
        stack_prev.push(p.data);
        j=j+1;
        p=p.next;
    if(self.length%2==0):
        if(p.data!=stack_prev.pop()):
            return False;
    p=p.next;
    while(p!=0):
        if(p.data!=stack_prev.pop()):
            return False;
        p=p.next;
    return True;

```

若入栈元素属于任意符号集合 $S$ ，入栈序列是 $S$ 中集合元素的一个排列 $C$ 。试设计算法，判定 $S$ 的另一个不同于 $C$ 的排列，是否可能是一个对应于 $S$ 的出栈序列。

试设计算法，将栈 $S$ 中的元素排序。要求不用辅助数据结构，仅通过对 $S$ 自身的操作完成 $S$ 中元素的排序。

现有栈 $S$ ，试设计算法，将 $S$ 中的元素逆置。要求不用辅助数据结构，仅通过对 $S$ 自身的操作完成 $S$ 中元素的逆置。

以上三题代码实现

```

class Node():
    def __init__(self,data=0,next=0):
        self.data=data;
        self.next=next;

class stack():
    def __init__(self):
        self.top=0;
    def is_empty(self):
        if(self.top==0):
            return True;

```

```
        else:
            return False;
def TopItem(self):
    if(self.is_empty()):
        return ;
    else:
        return self.top.data;
def push(self,x):
    newNode=Node(x,self.top);
    self.top=newNode;
def pop(self):
    if(self.is_empty()):
        return ;
    else:
        ret=self.top.data;
        self.top=self.top.next;
        return ret;

def printAll(self):
    # just used for debug
    if(self.is_empty()):
        print("the stack is empty!");
        return ;
    tempStack=stack();
    while(self.top!=0):
        print(self.top.data,end=" ");
        tempStack.push(self.pop());
    while(tempStack.top!=0):
        self.push(tempStack.pop());
    print("");
    return ;

def sort(self):
    if (self.is_empty()):
        print("the stack is empty!");
        return;
    tempStack=stack();
    tempStack.push(self.pop());
    while(not self.is_empty()):
        x=self.pop();
        j=0;
        while(x<tempStack.TopItem()):
            j=j+1;
            self.push(tempStack.pop());
            if(tempStack.is_empty()):
                break;
        tempStack.push(x);
        while(j!=0):
            tempStack.push(self.pop());
            j=j-1;
    while(not tempStack.is_empty()):
        self.push(tempStack.pop());

def getStackBottomAndRemove(self):
```

```
#use recursion
#get stack bottom Item and remove it.
x=self.TopItem();
self.pop();
if(self.is_empty()):
    return x;
last=self.getStackBottomAndRemove();
self.push(x);
return last;
def reverseStack(self):
    # reverse Stack by recursion
    if(self.is_empty()):
        return ;
    i=self.getStackBottomAndRemove();
    self.reverseStack();
    self.push(i);
    return ;
```

设计算法，将指定的广义表的内容原地逆置。例如：若广义表GL为[1, [2, 3], 4, [5, [6, 7], 8], 9]，逆置后GL为 [9, [8, [7, 6], 5], 4, [3, 2], 1] 。