数据结构和算法
（Python描述）

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

线性表

顺序表

河北草原天路

顺序表

- 即Python的列表，以及其它语言中的数组
- 元素在内存中连续存放
- 根据下标访问元素时间O(1)
- 在头部或中间插入删除元素时间O(n)
- 在尾部添加、删除元素时间O(1) (通过预先多分配已有元素固定倍数的空间来实现)
- 几乎不需要花费额外存储空间

链表概述

宁夏中卫沙坡头

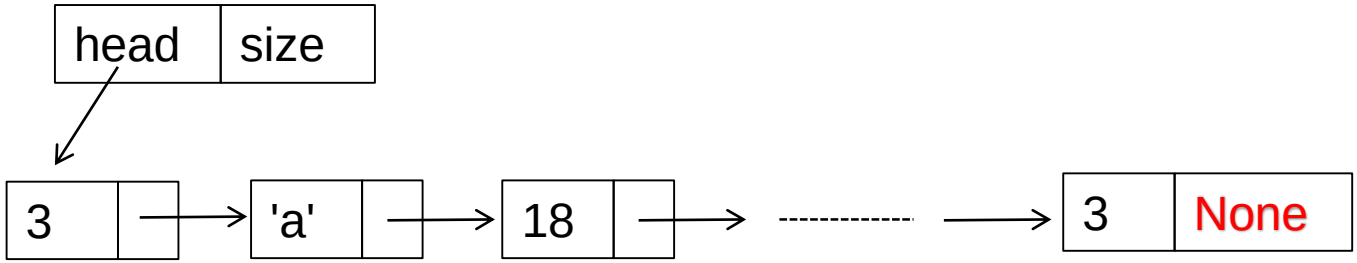
链表

- 元素在内存中并非连续存放
- 访问第i个元素，复杂度为O(n)
- 已经找到插入或删除位置的情况下，插入和删除元素的复杂度O(1)
- 有多种形式：
  - 单链表
  - 循环单链表
  - 双向链表
  - 循环双向链表

单链表

张掖冰沟丹霞

单链表

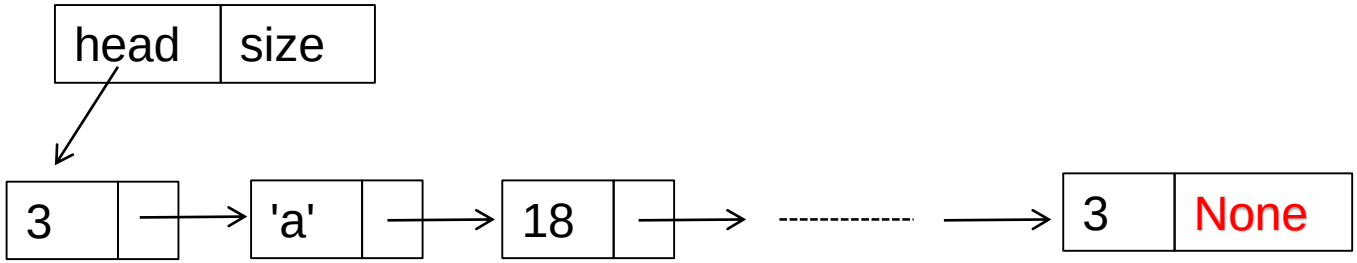


链表结构形式:

```
class LinkList:
    def __init__(self, head = None, size = 0):
        self.head, self.size = head,size
```

head: 表头元素指针
size: 链表元素个数

单链表

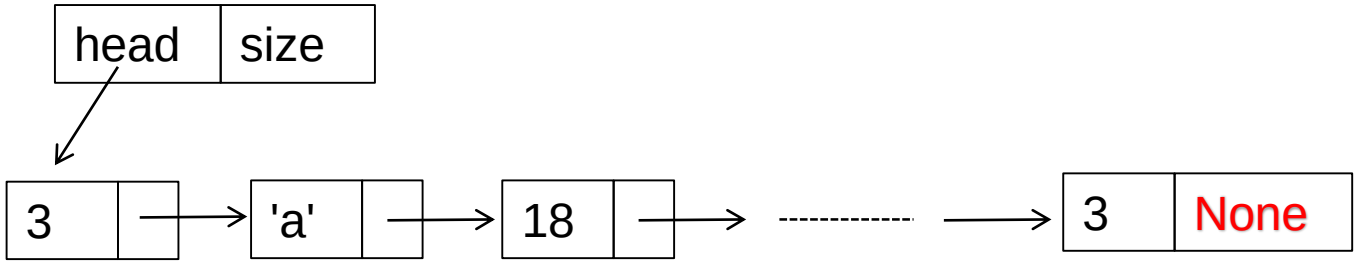


表结点形式：

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

data: 数据
next: 指向下一个节点的指针，即下一个节点。链表最后一个结点该值为None

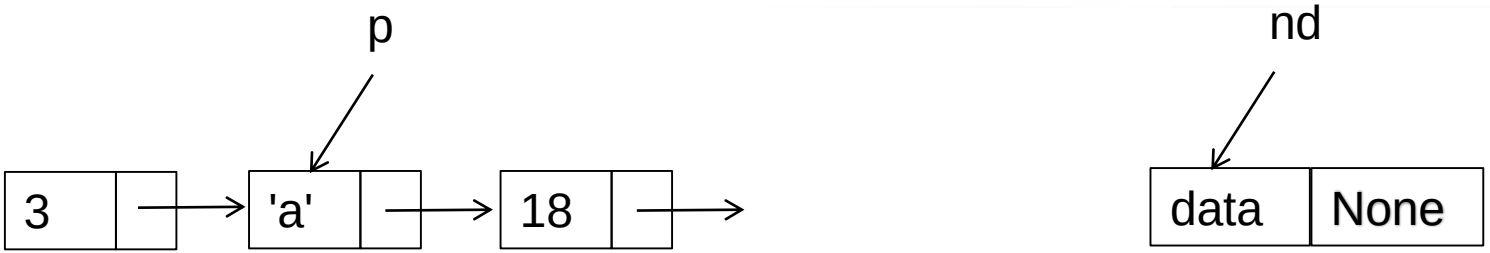
单链表



操作复杂度：

表头插入删除：O(1)
表尾添加、删除：O(n)　　要先从头开始找到表尾
在指定位置p进行插入删除：O(1)

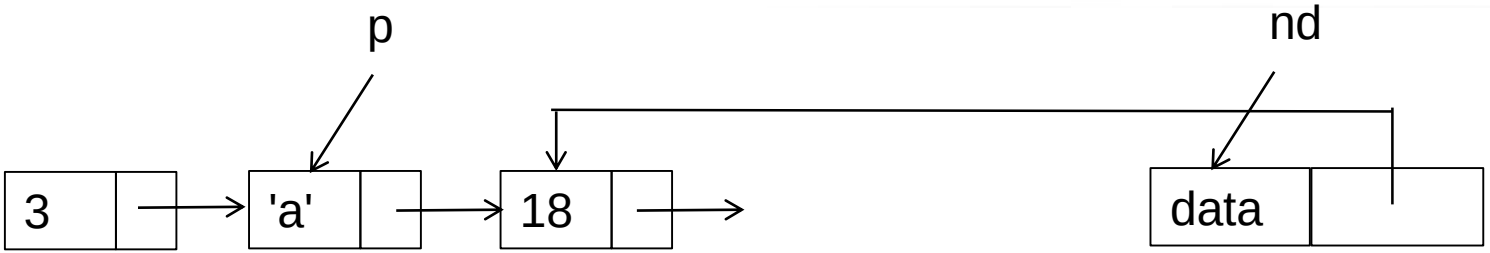
单链表指定位置插入元素



在p结点后面插入data:

```
nd = Node(data)  #新建节点包含数据data
```

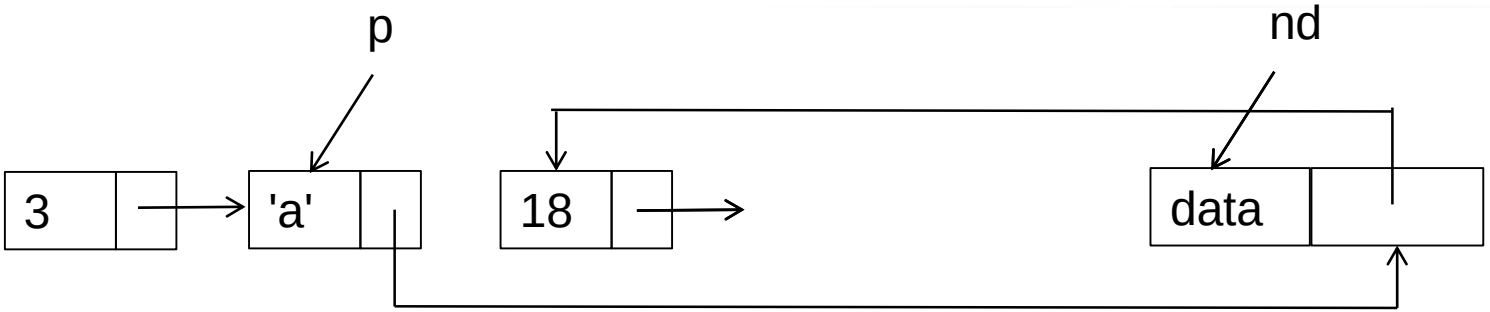
单链表指定位置插入元素



在p结点后面插入data:

```
nd = Node(data)  #新建节点包含数据data
nd.next = p.next
```

单链表指定位置插入元素

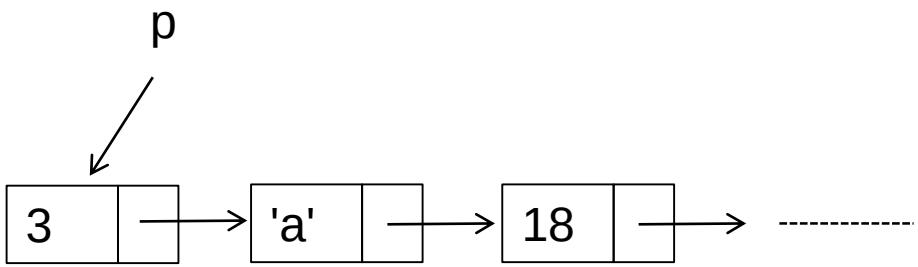


在p结点后面插入data:

```
nd = Node(data)  #新建节点包含数据data
nd.next = p.next
p.next = nd
```

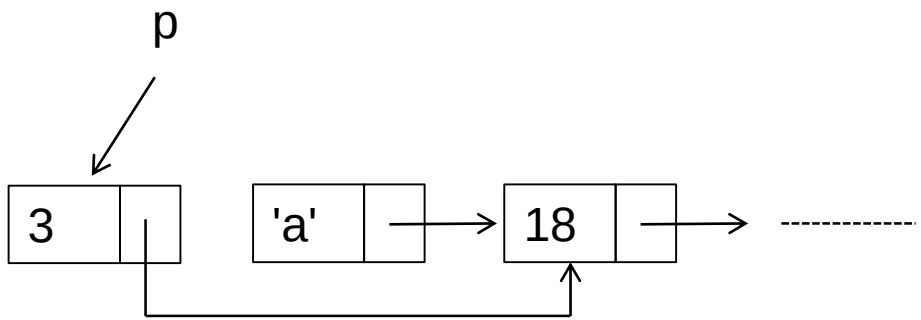
还要修改链表的size。
空链表插入第一个元素的情况单独处理。

单链表指定位置删除元素



删除p结点后面的结点：

单链表指定位置删除元素



删除p结点后面的结点：

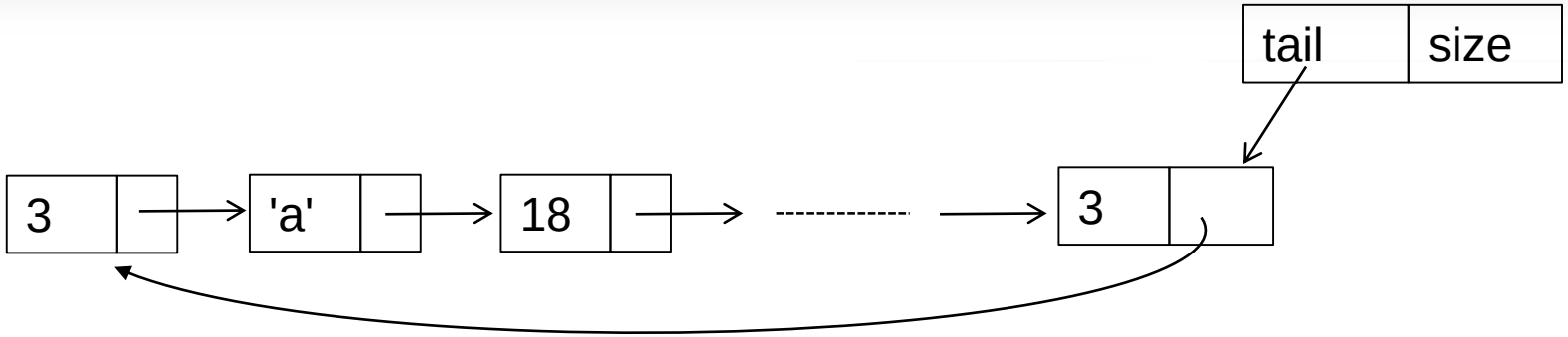
```
p.next = p.next.next
```

被删除的结点会被Python解释器自动回收

单循环链表

张掖平山湖大峡谷

单循环链表

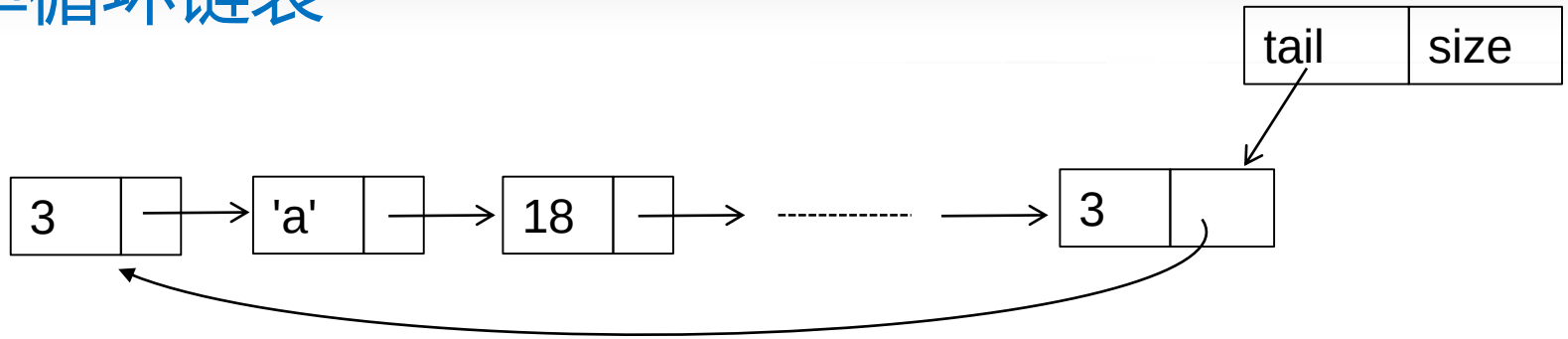


链表结构形式:

```
class LinkList:
    def __init__(self, tail = None, size = 0):
        self.tail, self.size = tail,size
```

tail: 表尾元素指针, tail.next算表头
size: 链表元素个数

单循环链表



操作复杂度

表头插入、删除：O(1)
表尾添加：O(1)
表尾删除：O(n)　要从表头开始找到表尾前面那个结点

单循环链表实现

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


class LinkList:  #单循环链表
    def __init__(self):
        self.tail = None
        self.size = 0
    def isEmpty(self):
        return self.size == 0
```

单循环链表实现

```
def pushFront(self,data):
    nd = Node(data)
    if self.tail == None:
        self.tail = nd
        nd.next = self.tail
    else:
        nd.next = self.tail.next
        self.tail.next = nd
    self.size += 1

def pushBack(self,data):
    self.pushFront(data)  #循环链表在尾部插入和在头部插入是一样的
    self.tail = self.tail.next
```

单循环链表实现

```
def popFront(self):
    if self.size == 0:
        return None
    else:
        nd = self.tail.next
        self.size -= 1
        if self.size == 0:
            self.tail = None
        else:
            self.tail.next = nd.next
    return nd.data
```

单循环链表实现

```
def printList(self):
    if self.size > 0:
        ptr = self.tail.next #ptr是表头
        while True:
            print(ptr.data,end = " ")
            if ptr == self.tail:
                break
            ptr = ptr.next
        print("")
```

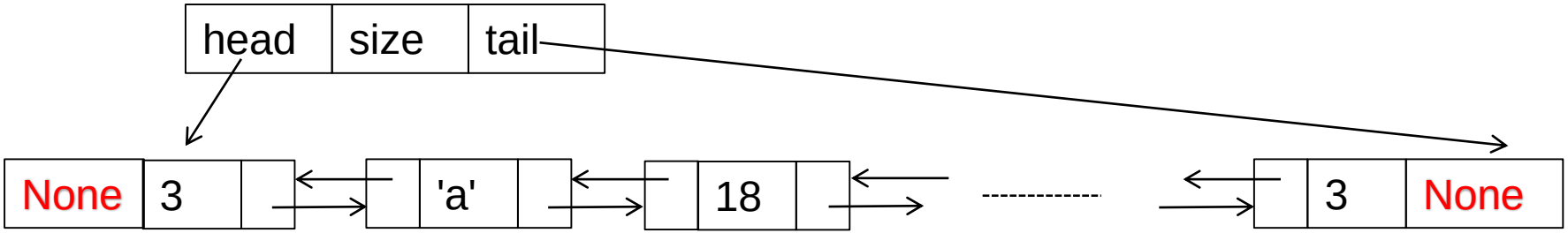
单循环链表实现

```
def reverse(self):  #前后颠倒整个链表
#实现方法：新建一个链表，把老链表元素依次删除并插入到新链表前端
    p = None  #这个是新链表头元素
    head = self.tail.next #老链表头元素
    self.tail.next = None
    tmpHead = head  #这个是新链表最后一个元素
    while head != None:
        q = head
        head = head.next
        q.next = p #插在新链表前面
        p = q
        self.tail = tmpHead
        tmpHead.next = p
```

双向链表

张掖平山湖大峡谷

双向链表

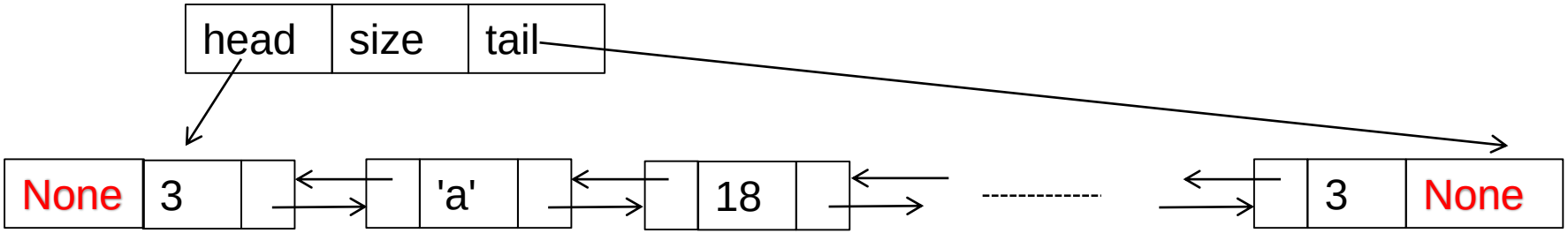


链表结构形式:

```
class LinkList:
    def __init__(self, head = None, tail = None size = 0):
        self.head, self.tail,self.size = head,tail,size
```

head: 表头元素指针
size: 链表元素个数
tail: 表尾元素指针

双向链表

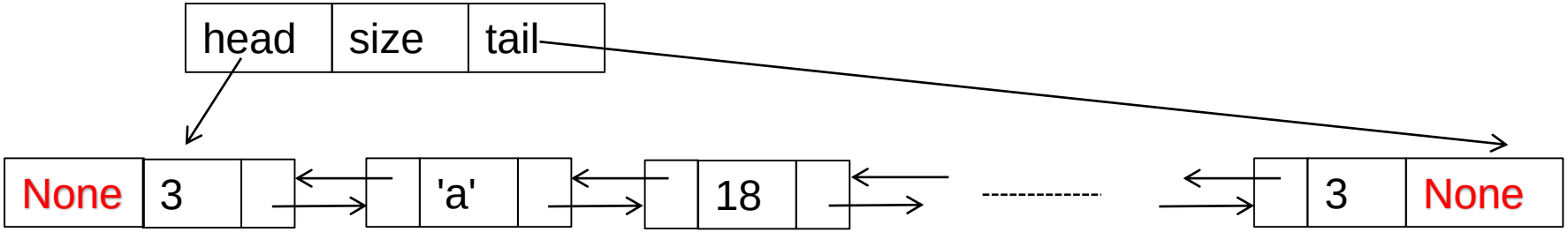


表结点结构形式:

```
class Node:
    def __init__(self, data, prev=None,next=None):
        self.data, self.prev, self.next = data,prev,next
```

data: 数据
prev: 指向上一个节点的指针，即上一个节点。链表头一个结点该值为None
next: 指向下一个节点的指针，即下一个节点。链表最后一个结点该值为None

双向链表



操作复杂度:

两端增删元素：O(1)

双向链表实现

```
class Node:
    def __init__(self, data, prev = None, next=None):
        self.data, self.prev, self.next = data, prev,next
class BiLinkList:
    def __init__(self):
        self.head = self.tail = None
        self.size = 0
    def isEmpty(self):
        return self.size == 0
    def pushFront(self,data):
        nd = Node(data)
        if self.head == None:
            self.head = self.tail = nd
        else:
            nd.next = self.head
            self.head.prev = nd
            self.head = nd
        self.size += 1
```

双向链表实现

```
def pushBack(self,data):
    nd = Node(data)
    if self.head == None:
        self.head = self.tail = nd
    else:
        nd.prev = self.tail
        self.tail.next = nd
        self.tail = nd
    self.size += 1
```

双向链表实现

```
def popFront(self):
    if self.size == 0:
        return None
    else:
        nd = self.head
        self.size -= 1
        if self.size == 0:
            self.head = self.tail = None
        else:
            self.head = self.head.next
            self.head.prev = None
    return nd.data
```

双向链表实现

```
def popBack(self):
    if self.size == 0:
        return None
    else:
        nd = self.tail
        self.size -= 1
        if self.size == 0:
            self.head = self.tail = None
        else:
            self.tail = self.tail.prev
            self.tail.next = None
    return nd.data
```

双向链表实现

```
def printList(self):
    if self.size > 0:
        ptr = self.head
        while ptr != None:
            print(ptr.data,end = " ")
            ptr = ptr.next
        print("")
```

链表结合顺序表

祁连牛心山

链表结合顺序表

- 顺序表

  中间插入太慢

- 链表：

  访问第i个元素太慢
  顺序访问也慢(现代计算机有cache，访问连续内存域比跳着访问内存区域快很多)

链表结合顺序表

- collections.deque：

结合链表和顺序表的特点。

是一张双向链表，每个结点是一个64个元素的顺序表。

```
class Node:
    def __init__(self ,prev=None,next = None):
        self.data = [0 for i in range(64)]
        self.data[0],self.data[-1] = prev,next
```