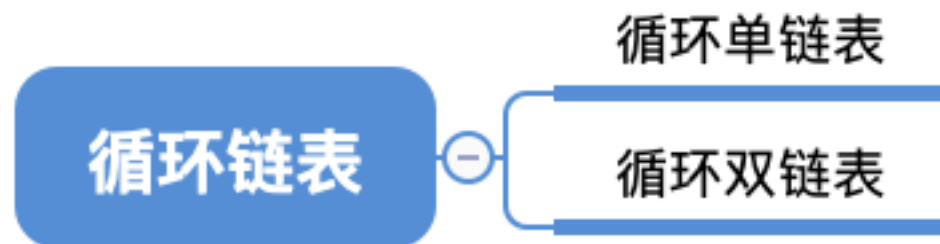


本节内容

循环链表

知识总览



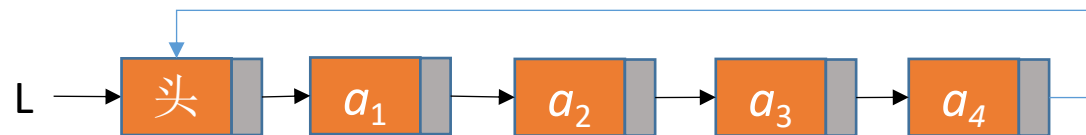
就那么简单



循环单链表



单链表：表尾结点的next指针指向 NULL



循环单链表：表尾结点的next指针指向头结点

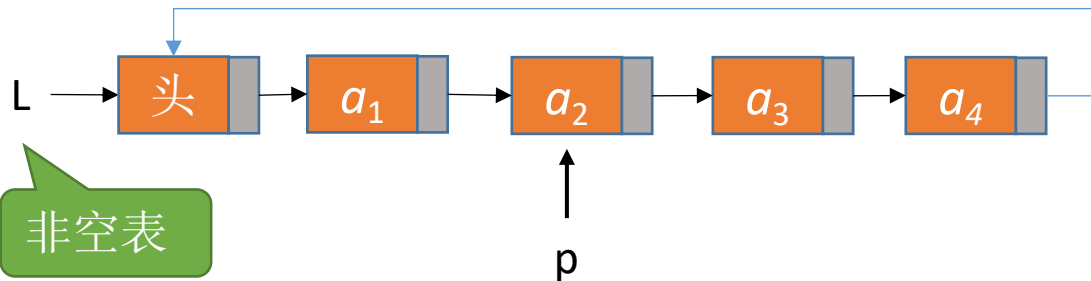
循环单链表

```
typedef struct LNode{  
    ElemType data;  
    struct LNode *next;  
}LNode, *LinkList;
```

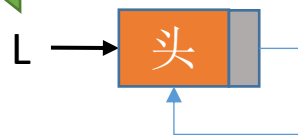
//定义单链表结点类型
//每个节点存放一个数据元素
//指针指向下一个节点

//初始化一个循环单链表

```
bool InitList(LinkList &L) {  
    L = (LNode *) malloc(sizeof(LNode));    //分配一个头结点  
    if (L==NULL)                            //内存不足, 分配失败  
        return false;  
    L->next = L;                            //头结点next指向头结点  
    return true;  
}
```



空表



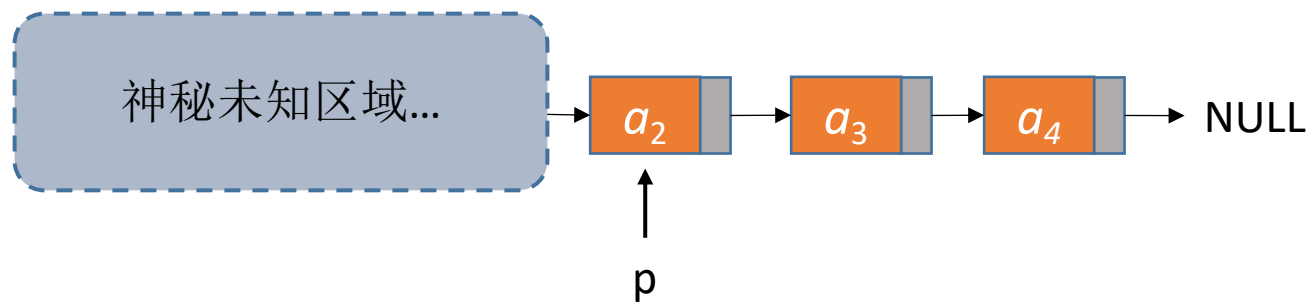
//判断循环单链表是否为空

```
bool Empty(LinkList L) {  
    if (L->next == L)  
        return true;  
    else  
        return false;  
}
```

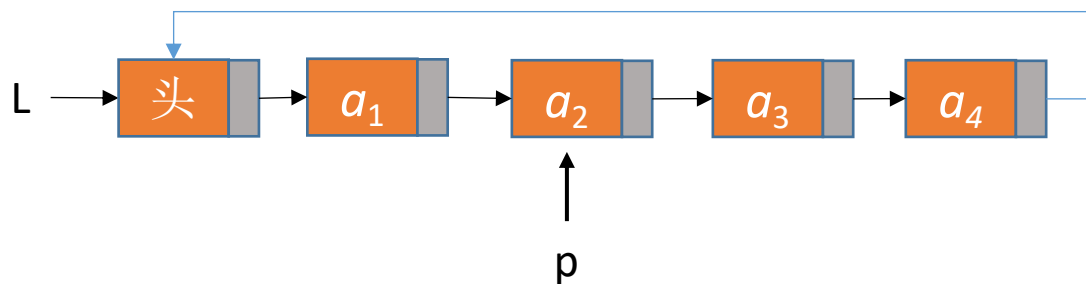
//判断结点p是否为循环单链表的表尾结点

```
bool isTail(LinkList L, LNode *p){  
    if (p->next==L)  
        return true;  
    else  
        return false;  
}
```

循环单链表

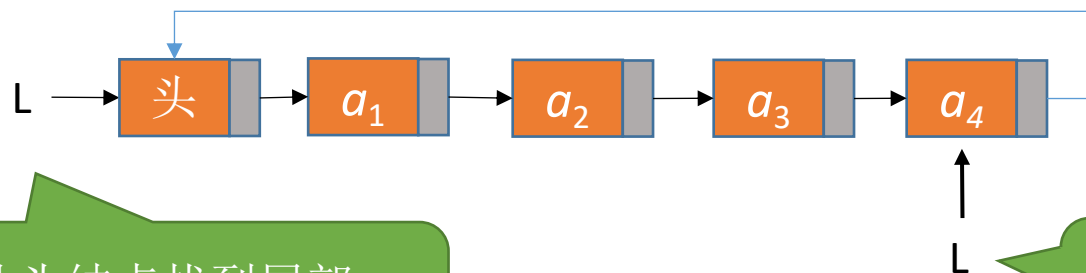


单链表：从一个结点出发只能找到后续的各个结点



循环单链表：从一个结点出发可以找到其他任何一个结点

循环单链表



从头结点找到尾部，
时间复杂度为 $O(n)$

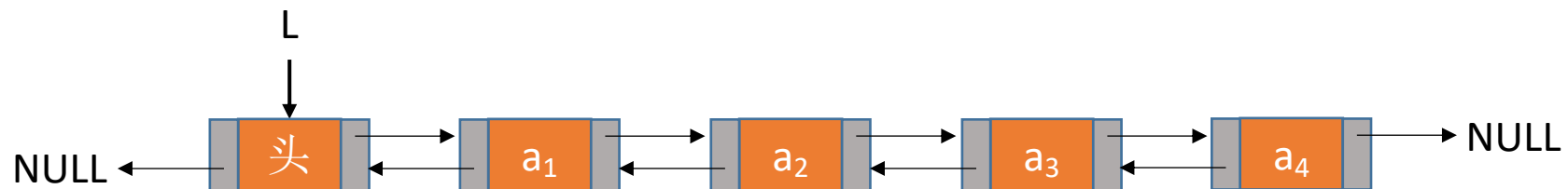
从尾部找到头部，
时间复杂度为 $O(1)$



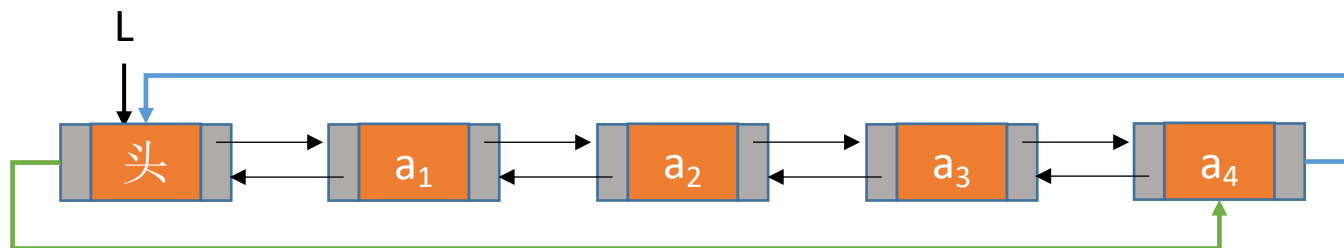
很多时候对链表的操作都是在头部或尾部

可以让L指向表尾元素
(插入、删除时可能需要修改L)

循环双链表



双链表：
表头结点的 prior 指向 NULL；
表尾结点的 next 指向 NULL



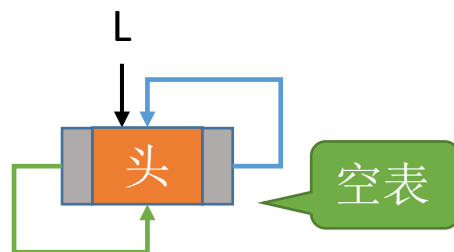
循环双链表：
表头结点的 prior 指向表尾结点；
表尾结点的 next 指向头结点

循环双链表的初始化

//初始化空的循环双链表

```
bool InitDLinkList(DLinklist &L){
    L = (DNode *) malloc(sizeof(DNode));    //分配一个头结点
    if (L==NULL)                            //内存不足, 分配失败
        return false;
    L->prior = L;                          //头结点的 prior 指向头结点
    L->next = L;                            //头结点的 next 指向头结点
    return true;
}
```

```
typedef struct DNode{
    ElemType data;
    struct DNode *prior,*next;
}DNode, *DLinklist;
```



```
void testDLinkList() {
    //初始化循环双链表
    DLinklist L;
    InitDLinkList(L);
    //...后续代码...
}
```

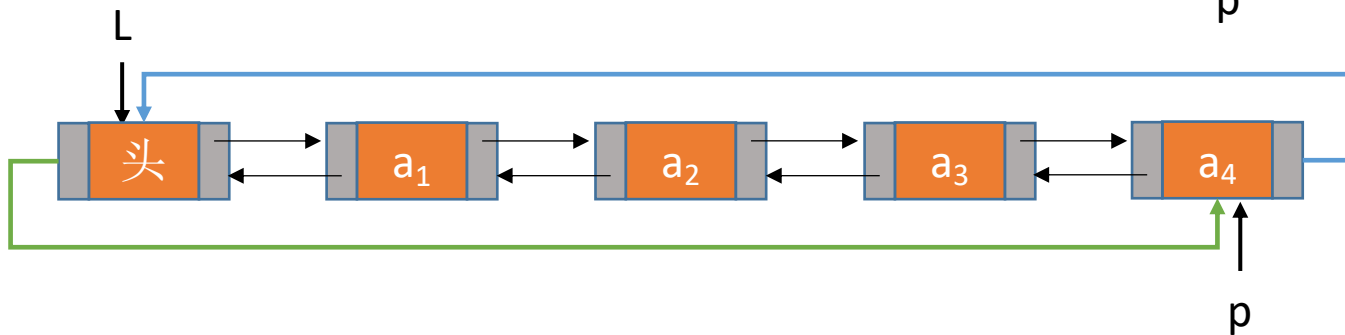
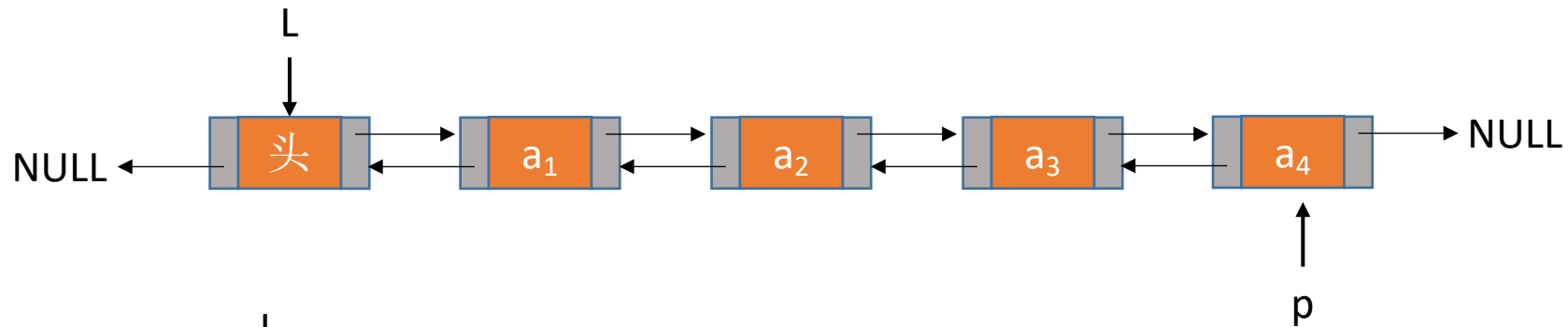
```
//判断循环双链表是否为空
bool Empty(DLinklist L) {
    if (L->next == L)
        return true;
    else
        return false;
}
```

```
//判断结点p是否为循环单链表的表尾结点
bool isTail(DLinklist L, DNode *p){
    if (p->next==L)
        return true;
    else
        return false;
}
```


双链表的插入

//在 p 结点之后插入 s 结点

```
bool InsertNextDNode(DNode *p, DNode *s){  
    s->next=p->next;    //将结点*s插入到结点*p之后  
    ➡ p->next->prior=s;  
    s->prior=p;  
    p->next=s;  
}
```



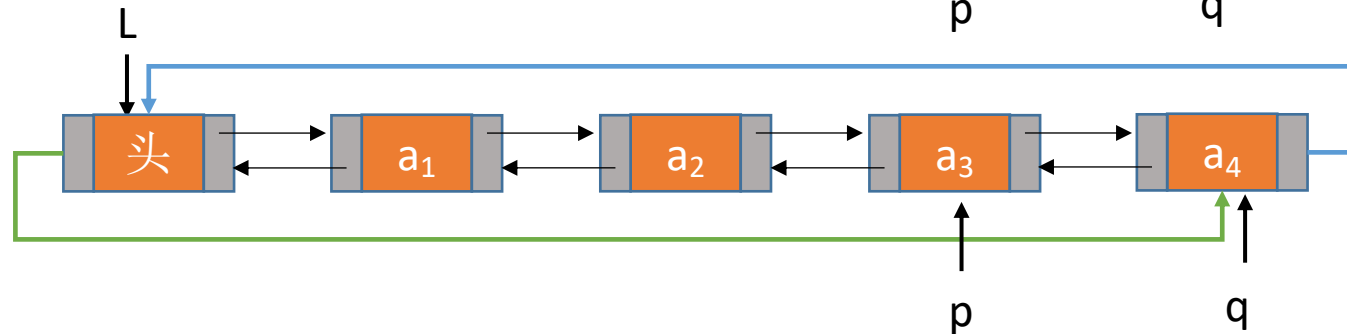
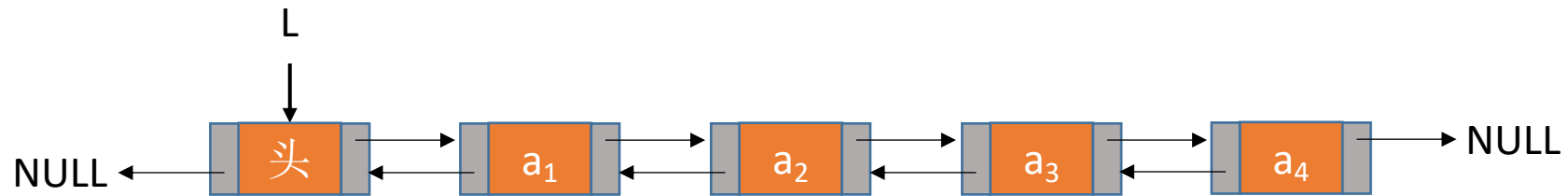
双链表的删除

//删除 p 的后继结点 q

$p \rightarrow \text{next} = q \rightarrow \text{next};$

→ $q \rightarrow \text{next} \rightarrow \text{prior} = p;$

$\text{free}(q);$

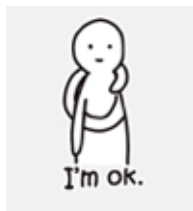


知识回顾与重要考点

循环链表

循环单链表

空表



单手抱住空虚的自己

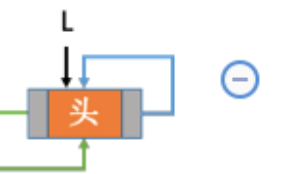
非空表



普通的空虚双链表

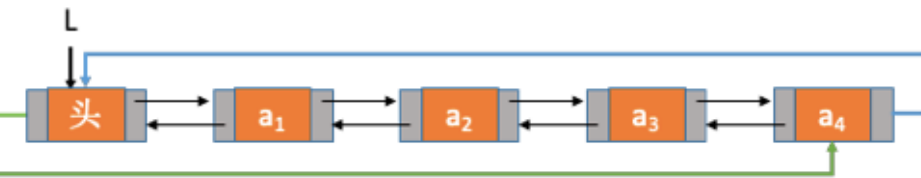
循环双链表

空表



双手抱住空虚的自己

非空表



如何判空

代码问题

如何判断结点p是否是表尾/表头结点

后向/前向遍历的实现核心

如何在表头、表中、表尾插入/删除一个结点

插入、删除操作的不易错思路

欢迎大家对本节视频进行评价~



学员评分：2.3.4 循环链表

扫一扫二维码打开或分享给好友



— 腾讯文档 —

可多人实时在线编辑，权限安全可控



公众号：王道在线



b站：王道计算机教育



抖音：王道计算机考研