本节内容

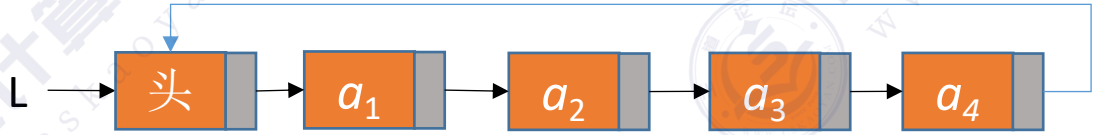
循环链表

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

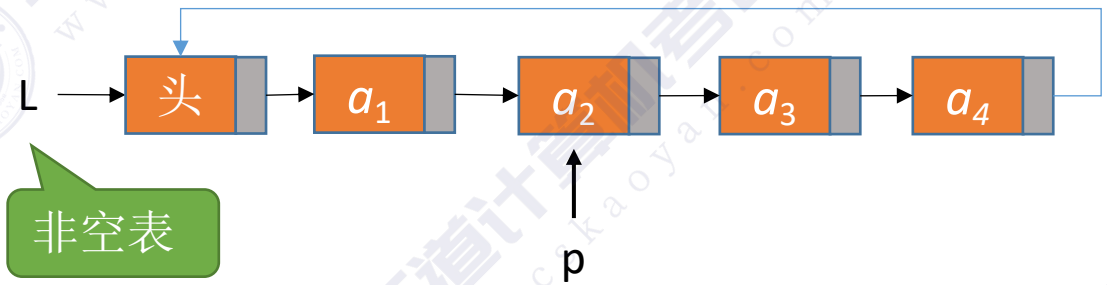


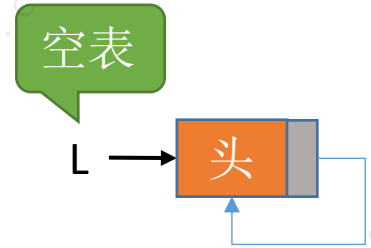
循环单链表：表尾结点的next指针指向头结点

循环单链表

```
typedef struct LNode{          //定义单链表结点类型
    ElemType data;             //每个节点存放一个数据元素
    struct LNode *next;        //指针指向下一个节点
}LNode, *LinkList;

//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;
}
```

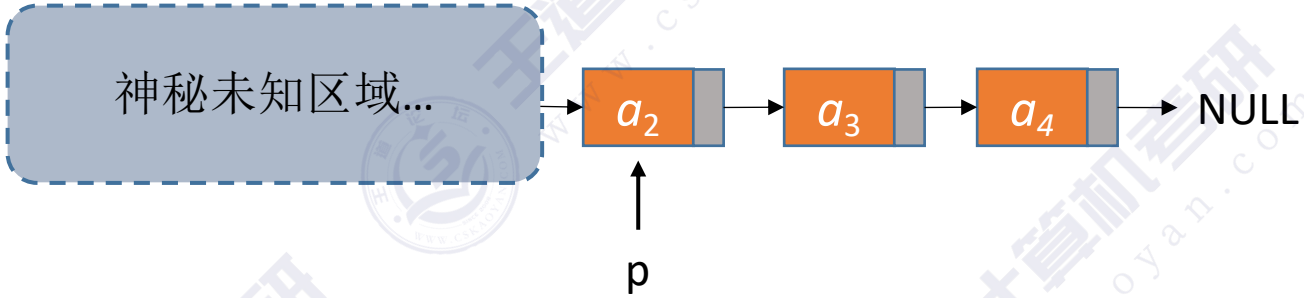




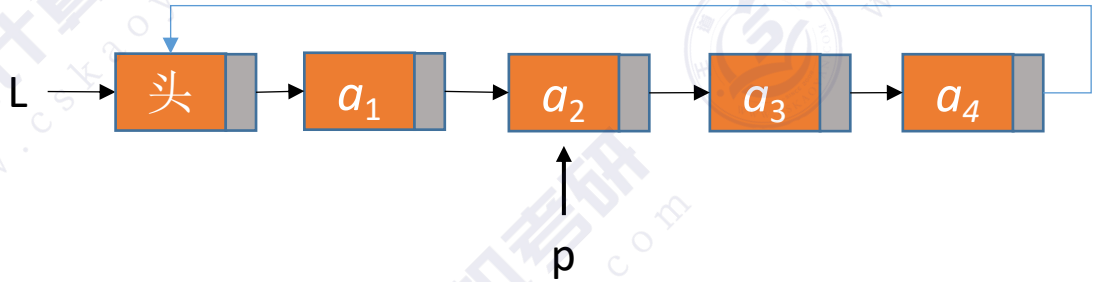

```
//判断循环单链表是否为空
bool Empty(LinkList L) {
    if (L->next == L)
        return true;
    else
        return false;
}
```

```
//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

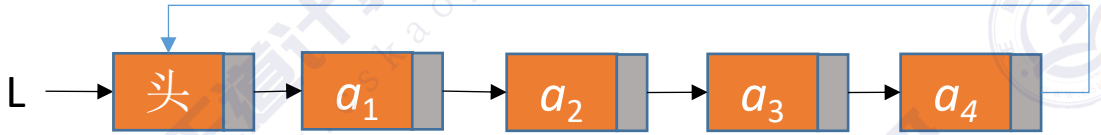


单链表：从一个结点出发只能找到后续的各个结点



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)

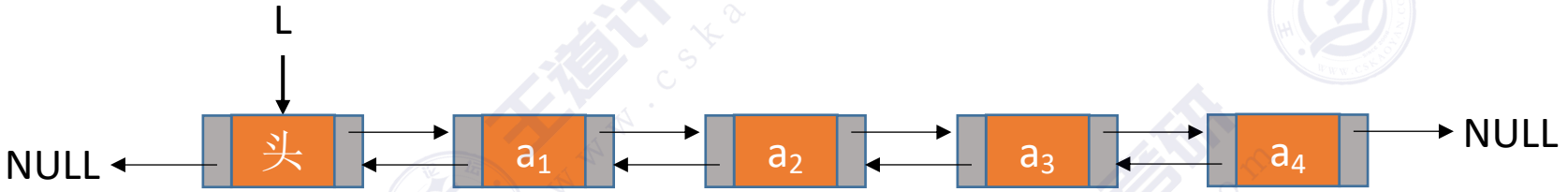
L

从尾部找到头部，
时间复杂度为O(1)

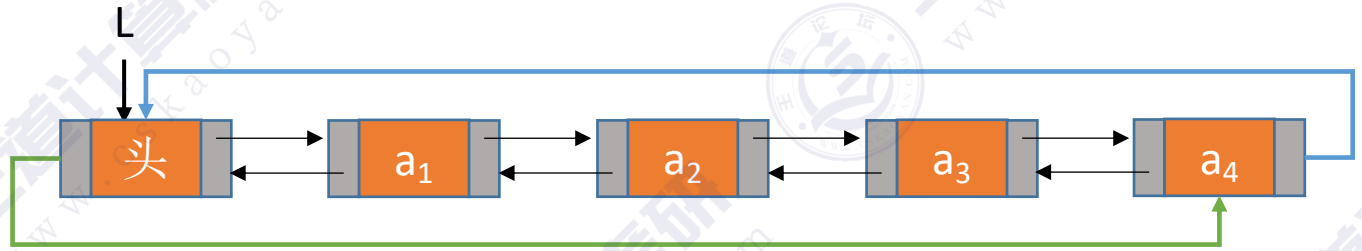
很多时候对链表的操作都是在头部或尾部

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

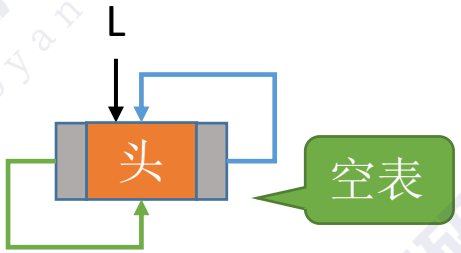


循环双链表：
表头结点的 prior 指向表尾结点；
表尾结点的 next 指向头结点

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;
```



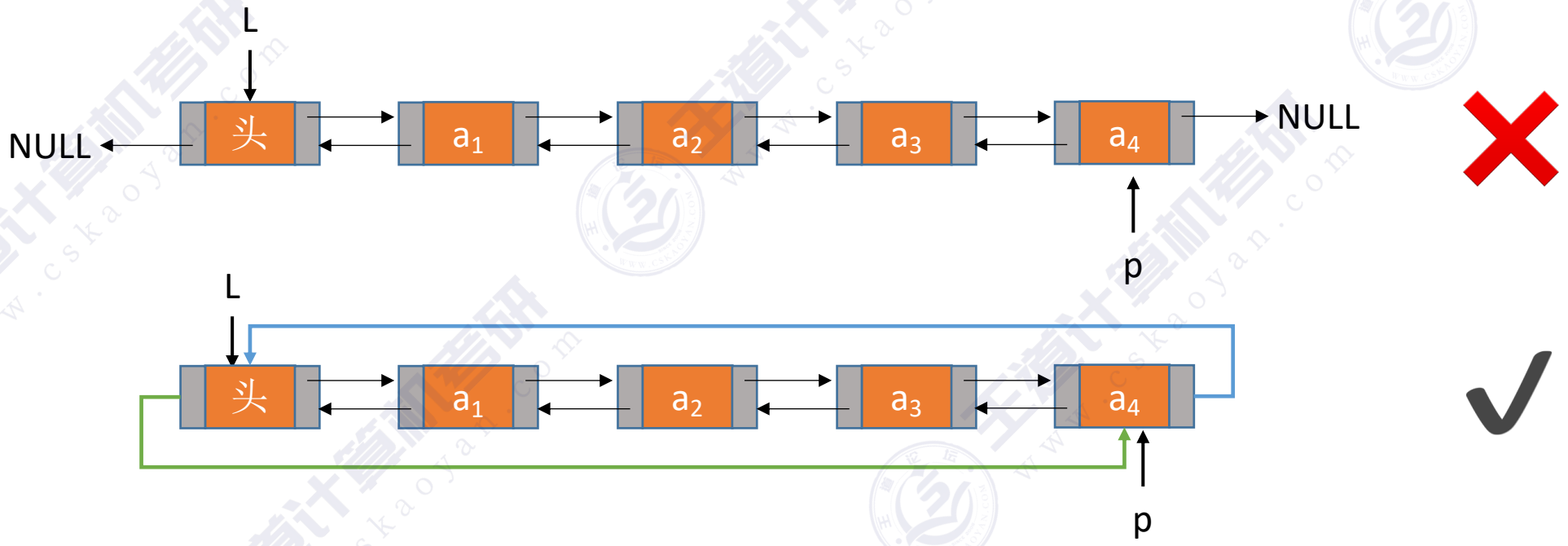

```
void testDLinkList() {
    //初始化循环双链表
    DLinklist L;
    InitDLinkList(L);
    //...后续代码...
}
```

```
//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;
}
```

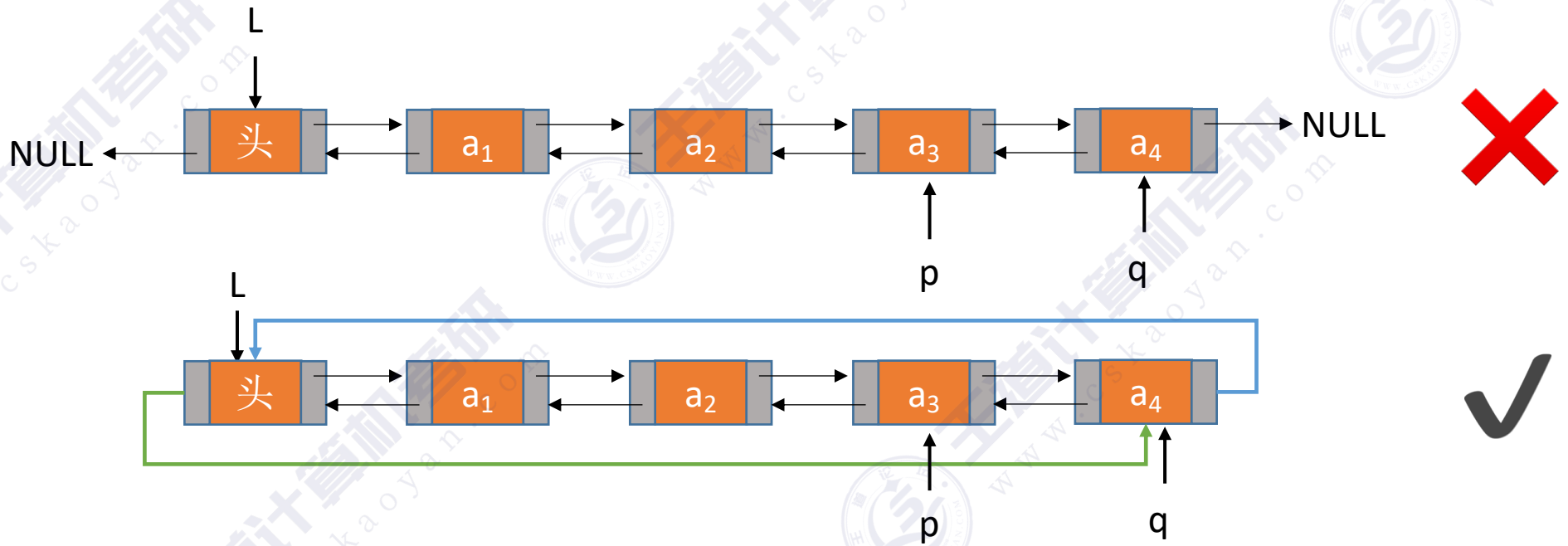
双链表的插入

```
//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;
}
```

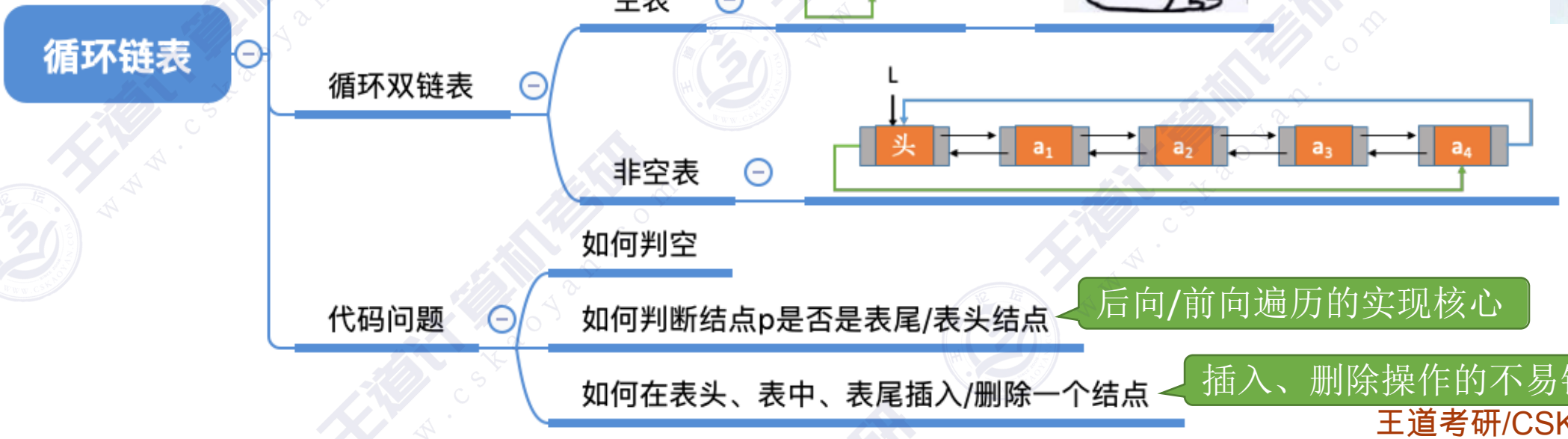


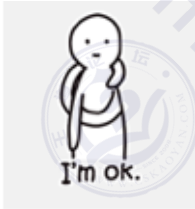
双链表的删除

```
//删除p的后继结点q
p->next=q->next;
q->next->prior=p;
free(q);
```



知识回顾与重要考点





单手抱住空虚的自己

普通的空虚双链表

双手抱住空虚的自己

后向/前向遍历的实现核心

插入、删除操作的不易错思路