

本节内容

双链表

知识总览

双链表

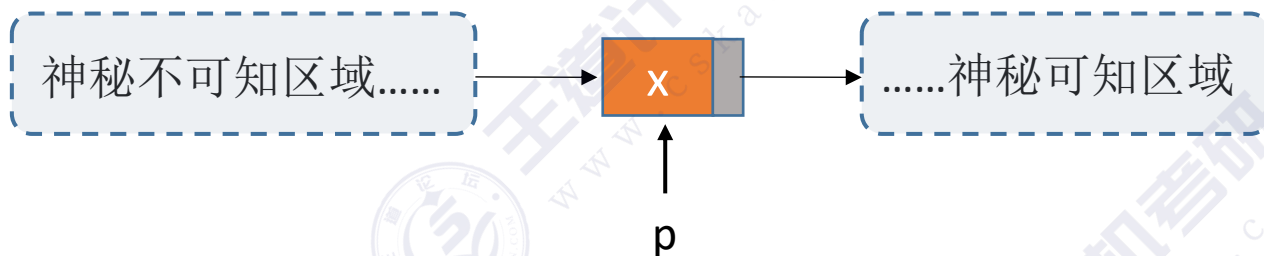
初始化

插入

删除

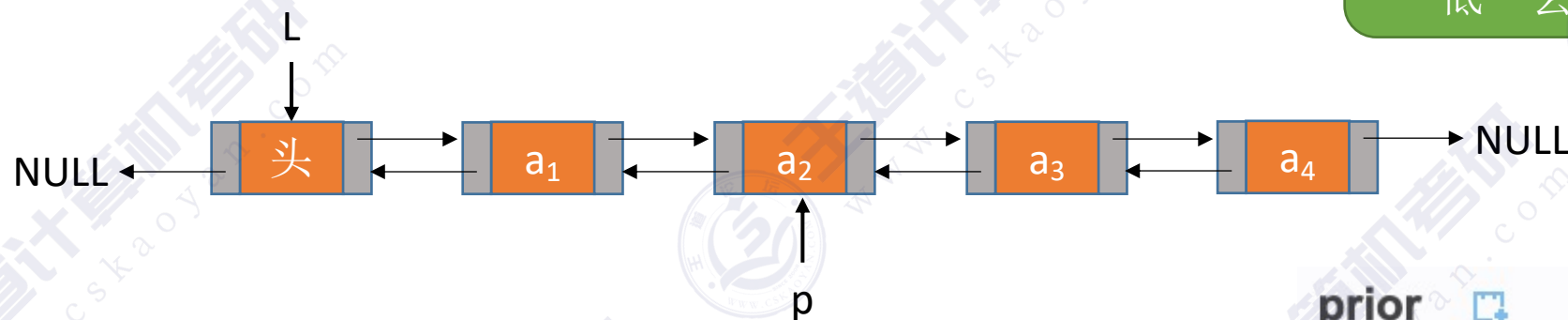
遍历

单链表 V.S. 双链表



单链表：无法逆向检索，有时候不太方便

双链表：可进可退，存储密度更低一丢丢



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

//定义双链表结点类型
//数据域
//前驱和后继指针

prior

英 ['praɪə(r)]

美 ['praɪər]

adj. (时间、顺序等) 先前的; 优先的

n. 男修道院副院长; 托钵会会长; (非正式) 犯罪前科; 先验;

n. (Prior) (美) 普廖尔 (人名)

[复数 priors]

双链表的初始化（带头结点）

//初始化双链表

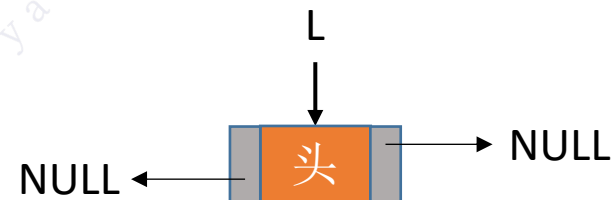
```
bool InitDLinkedList(DLinkedList &L){  
    L = (DNode *) malloc(sizeof(DNode));    //分配一个头结点  
    if (L==NULL)                            //内存不足，分配失败  
        return false;  
    L->prior = NULL;                        //头结点的 prior 永远指向 NULL  
    L->next = NULL;                        //头结点之后暂时还没有节点  
    return true;  
}
```

```
void testDLinkedList() {  
    //初始化双链表  
    DLinkedList L;  
    InitDLinkedList(L);  
    //后续代码...  
}
```

```
//判断双链表是否为空（带头结点）  
bool Empty(DLinkedList L) {  
    if (L->next == NULL)  
        return true;  
    else  
        return false;  
}
```

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

DLinkedList \longleftrightarrow 等价 \longleftrightarrow DNode *



双链表的插入

//在p结点之后插入s结点

```
bool InsertNextDNode(DNode *p, DNode *s){
```

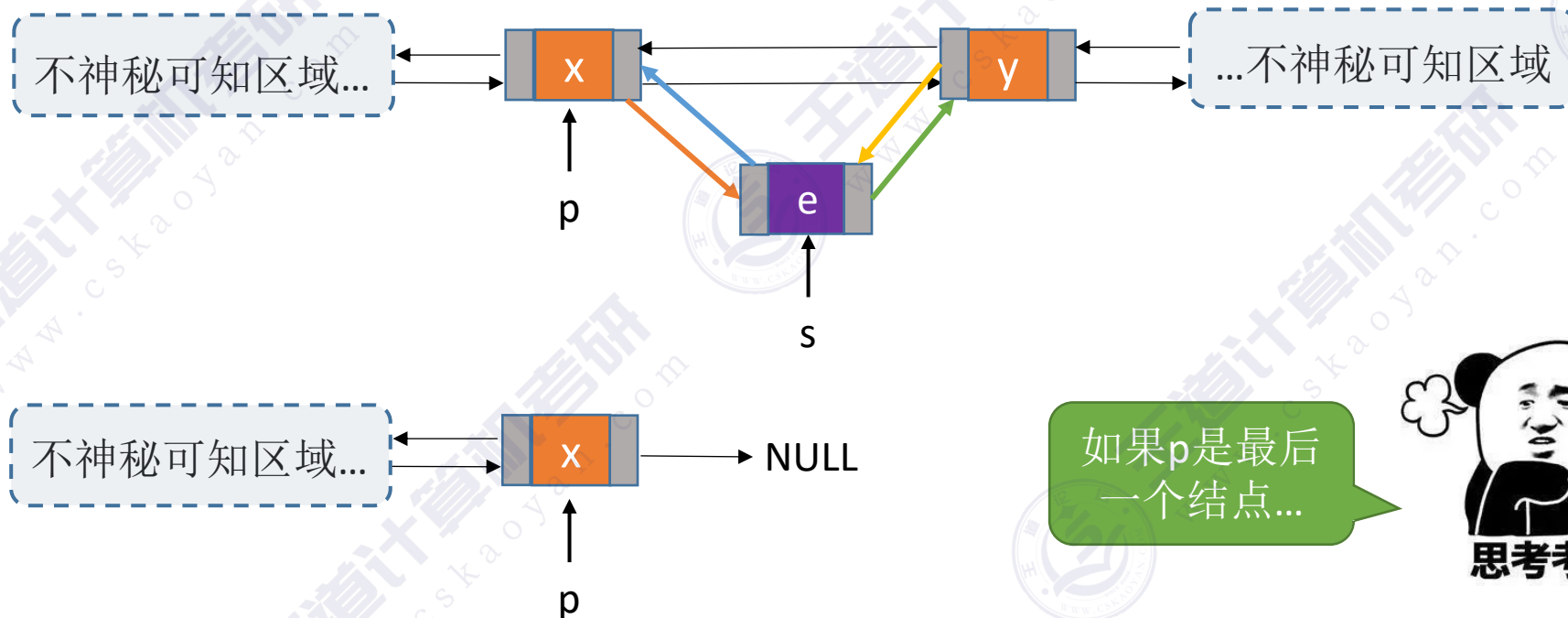
```
    s->next=p->next;    //将结点*s插入到结点*p之后
```

```
    p->next->prior=s;
```

```
    s->prior=p;
```

```
    p->next=s;
```

```
}
```



双链表的插入

//在p结点之后插入s结点

```
bool InsertNextDNode(DNode *p, DNode *s){  
    if (p==NULL || s==NULL) //非法参数  
        return false;  
    ① s->next=p->next;  
    ② if (p->next != NULL) //如果p结点有后继结点  
        p->next->prior=s;  
    ③ s->prior=p;  
    ④ p->next=s;  
    return true;  
}
```

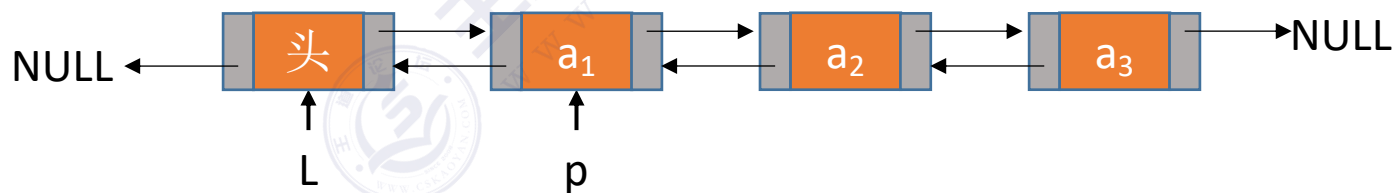
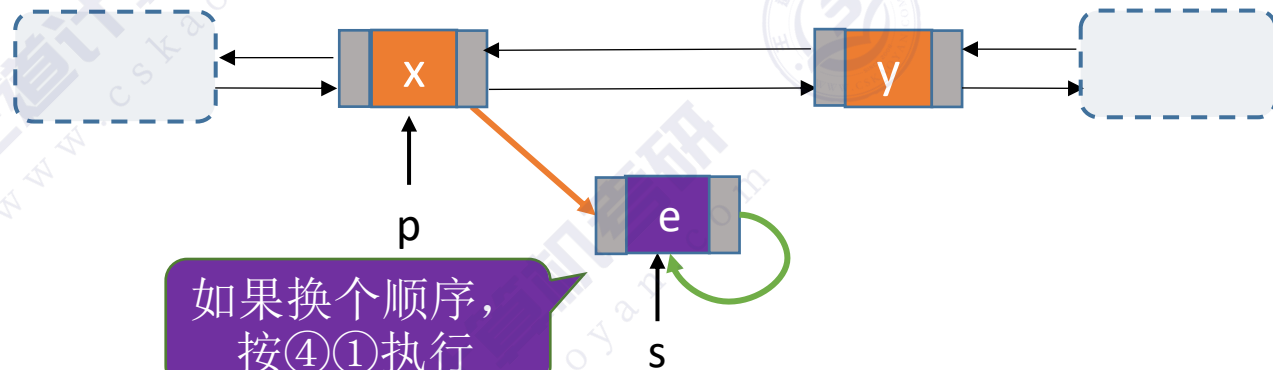
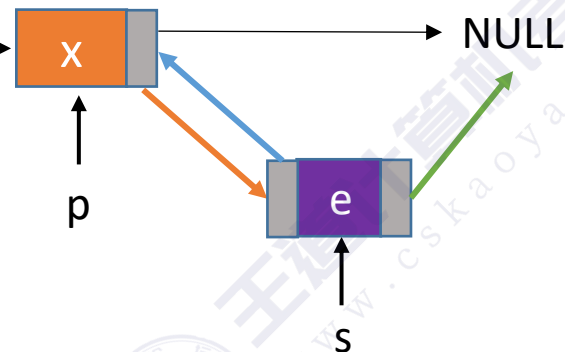
修改指针时要注意顺序

用后插操作实现结点的插入有什么好处?

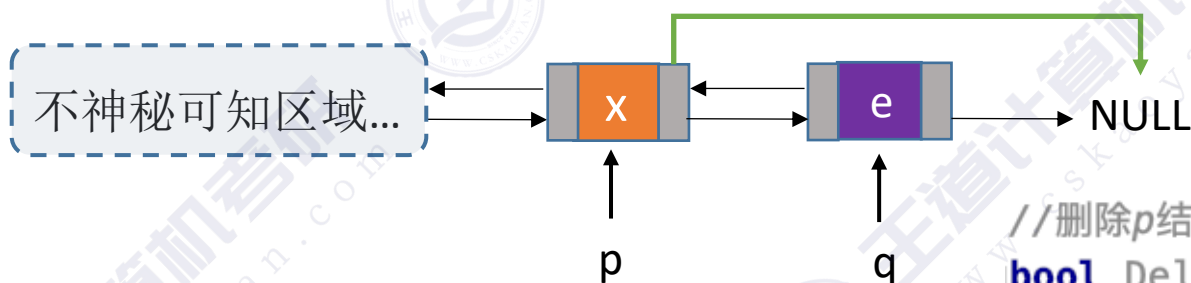
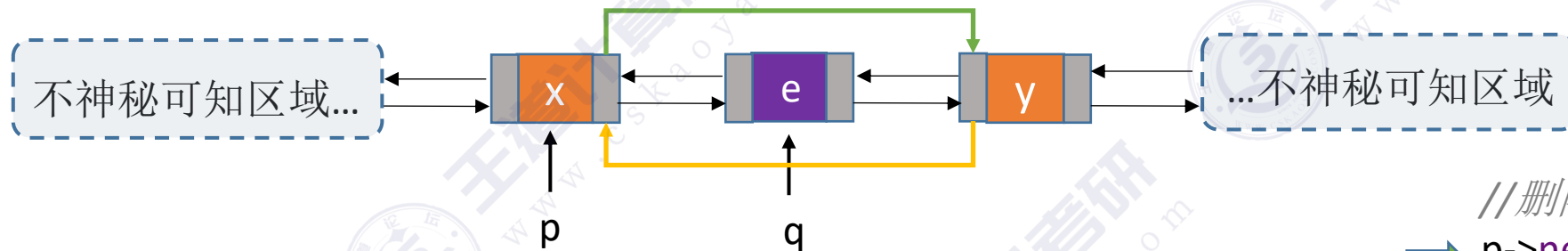
按位序插入
前插操作



神秘未知区域.....



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);

//删除p结点的后继结点

```
bool DeleteNextDNode(DNode *p){  
    if (p==NULL) return false;  
    DNode *q = p->next; //找到p的后继结点q  
    if (q==NULL) return false; //p没有后继  
    p->next=q->next;  
    if (q->next!=NULL) //q结点不是最后一个结点  
        q->next->prior=p;  
    free(q); //释放结点空间  
    return true;  
}
```

后删操作实现结点的删除有什么好处?

销毁表时才能删除头结点

```
void DestroyList(DLinklist &L){  
    //循环释放各个数据结点  
    while (L->next != NULL)  
        DeleteNextDNode(L);  
    free(L); //释放头结点  
    L=NULL; //头指针指向NULL  
}
```

双链表的遍历

后向遍历

```
while (p!=NULL){  
    //对结点p做相应处理，如打印  
    p = p->prior;  
}
```

前向遍历

```
while (p!=NULL){  
    //对结点p做相应处理  
    p = p->next;  
}
```

前向遍历（跳过头结点）

```
while (p->prior != NULL){  
    //对结点p做相应处理  
    p = p->prior;  
}
```

双链表不可随机存取，按位查找、按值查找操作都只能用遍历的方式实现。时间复杂度 $O(n)$

知识回顾与重要考点

双链表

如果不带头
结点呢？

初始化

头结点的 prior、next 都指向 NULL

插入 (后插)

注意新插入结点、前驱结点、后继结点的指针修改

边界情况：新插入结点在最后一个位置，需特殊处理

删除 (后删)

注意删除结点的前驱结点、后继结点的指针修改

边界情况：如果被删除结点是最后一个数据结点，需特殊处理

遍历

从一个给定结点开始，后向遍历、前向遍历的实现（循环的终止条件）

链表不具备随机存取特性，查找操作只能通过顺序遍历实现