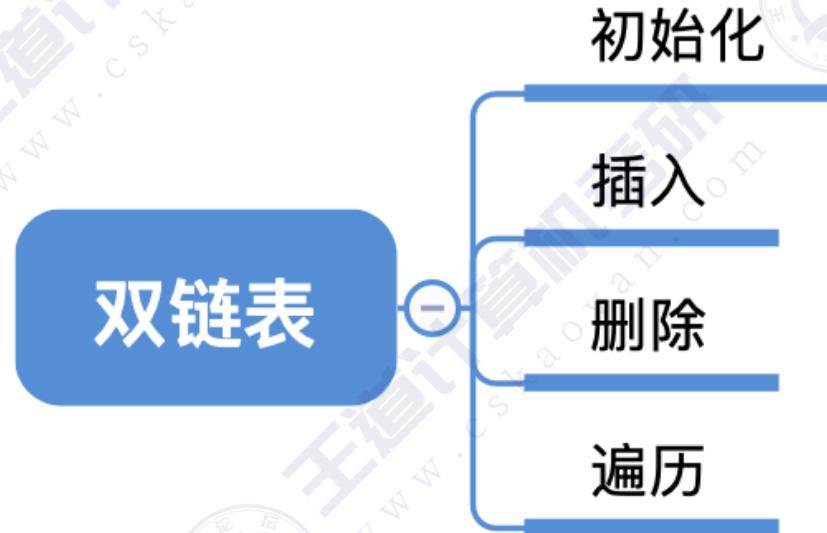


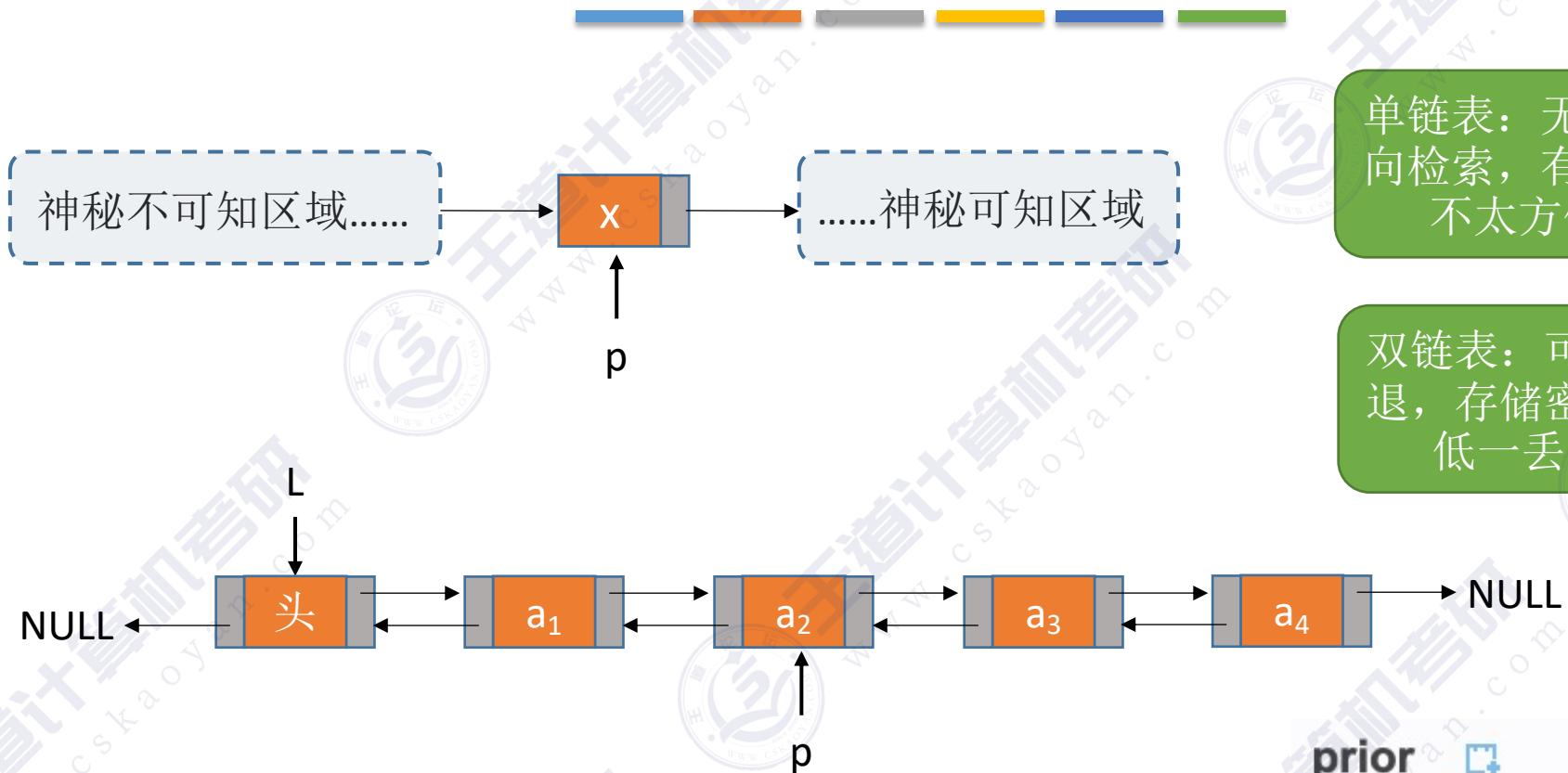
本节内容

双链表

知识总览



单链表 v.s. 双链表



```
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 [ˈpreɪə(r)] 美 [ˈpreɪər]

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) 普摩尔 (人名)

[复数 priors]

双链表的初始化（带头结点）

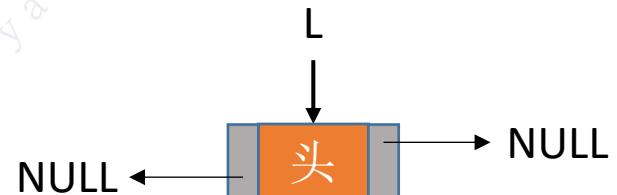
```
//初始化双链表
bool InitDLinkList(DLinklist &L){
    L = (DNode *) malloc(sizeof(DNode)); //分配一个头结点
    if (L==NULL) //内存不足，分配失败
        return false;
    L->prior = NULL; //头结点的 prior 永远指向 NULL
    L->next = NULL; //头结点之后暂时还没有节点
    return true;
}
```

```
void testDLinkList() {
    //初始化双链表
    DLinklist L;
    InitDLinkList(L);
    //后续代码。。
}
```

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

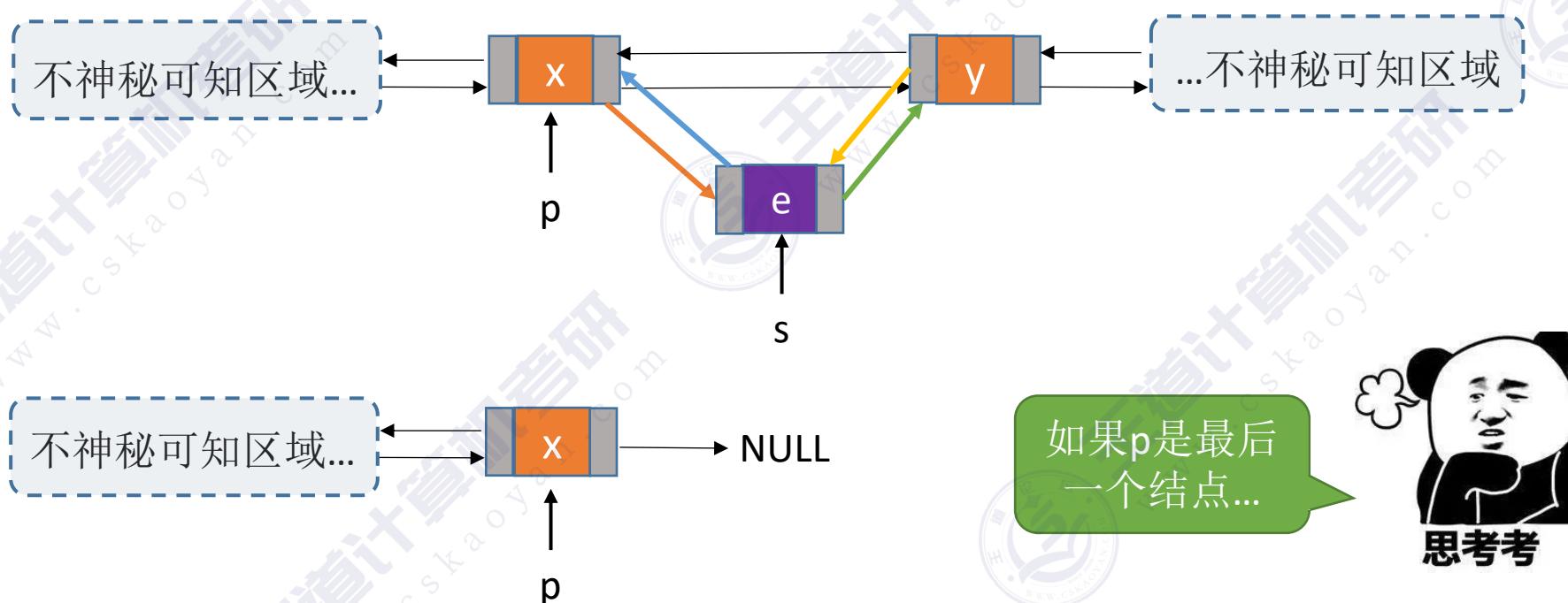
```
typedef struct DNode{
    ElemType data;
    struct DNode *prior,*next;
}DNode, *DLinklist;
```

DLinklist  等价 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)
        p->next->prior=s;
    ③ s->prior=p;
    ④ p->next=s;
    return true;
}
```

修改指针时要注意顺序

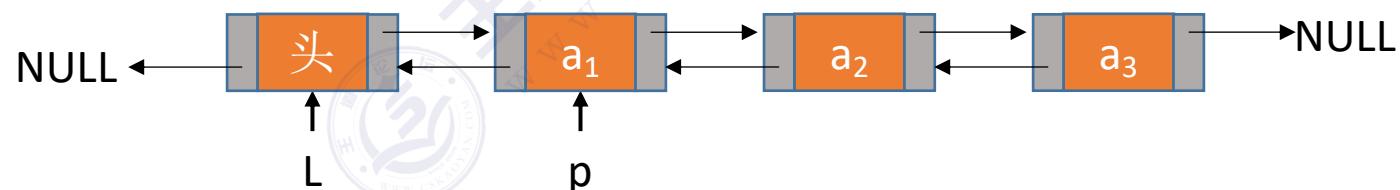
用后插操作实现结点的插入有什么好处？

按位序插入
前插操作

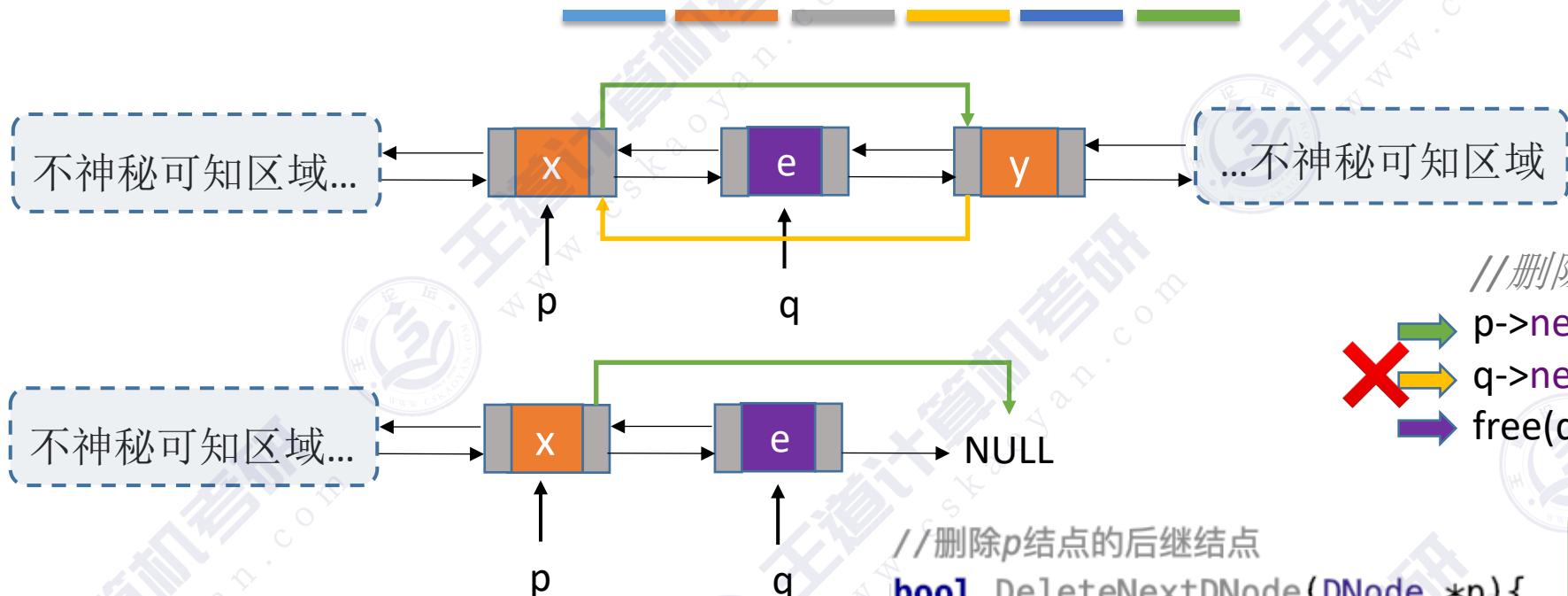


神秘未知区域.....

如果换个顺序，
按④①执行



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);

后删操作实现结点的
删除有什么好处?

```
void DestoryList(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  
}
```

销毁表时
才能删除
头结点

```
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->next->prior=p;  
    free(q);  
    return true;  
}
```

//q结点不是最后一个结点

//释放结点空间

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->next;  
}
```

前向遍历

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

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

