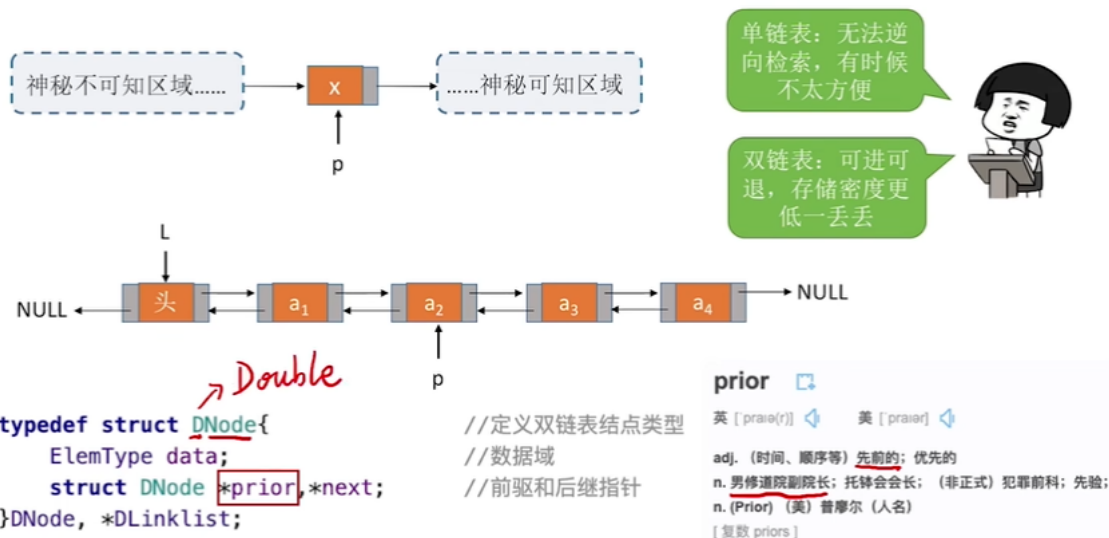


双链表

哔哩哔哩

单链表 V.S. 双链表



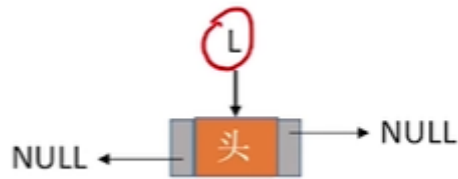
• 双链表的初始化 (带头结点)

```
1 typedef struct DNode
2 {
3     ElemType data;
4     struct DNode *prior, *next;
5 }DNode, *DLinklist;
6
7 //初始化双链表
8 bool InitDLinklist(DLinklist &L)
9 {
10     L = (DNode *)malloc(sizeof(DNode)); //分配一个头结点
11
12     if (L == NULL) //内存不足，分配失败
13         return false;
14
15     L->prior = NULL; //头结点的prior永远指向NULL
16     L->next = NULL; //头结点之后暂时还没有节点
17     return true;
18 }
19
20 void testDLinklist()
21 {
22     //初始化双链表
23     DLinklist L;
24     InitDLinklist(L);
25     //后续代码...
26 }
27
28 //判断双链表是否为空 (带头结点)
29 bool Empty(DLinklist L)
30 {
31     if(L->next == NULL)
```

```

32     return true;
33     else
34         return false;
35 }

```



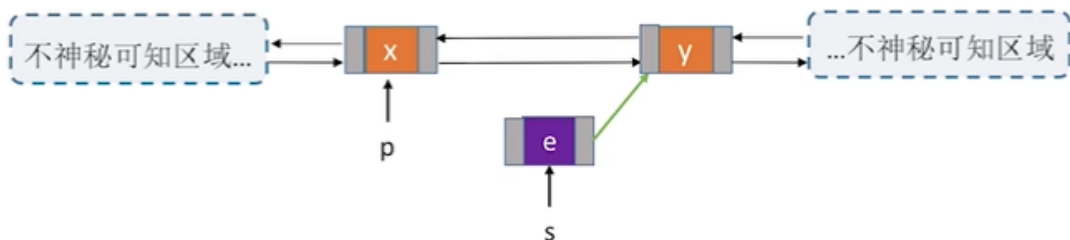
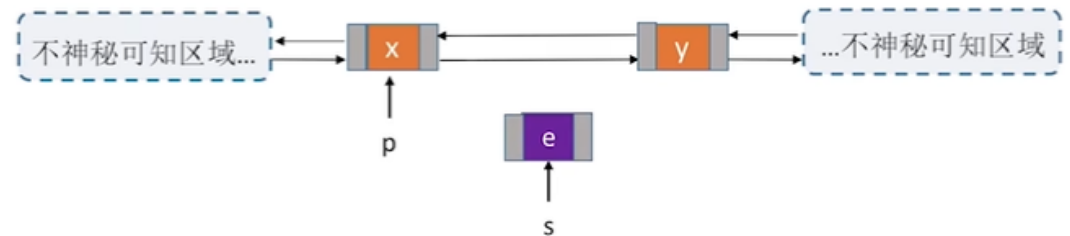
DLinklist \longleftrightarrow 等价 \longleftrightarrow DNode *

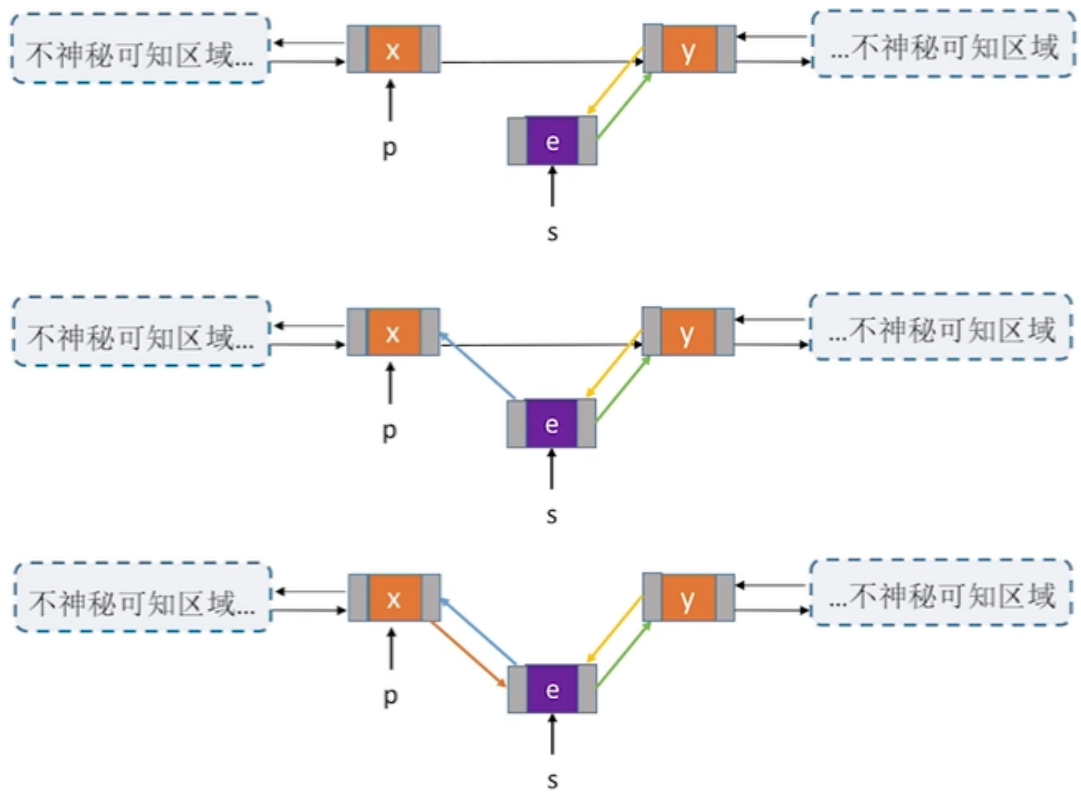
• 双链表的插入

```

1 //在p结点之后插入s结点
2 bool InsertNextDNode(DNode *p,DNode *s)
3 {
4     if(p == NULL || s == NULL) //非法参数
5         return false;
6     s->next = p->next; //将结点*s插入到结点*p之后
7
8     if(p->next != NULL)
9         p->next->prior = s; //如果p结点有后继结点
10    s->prior = p;
11    p->next = s;
12    return true;
13 }

```

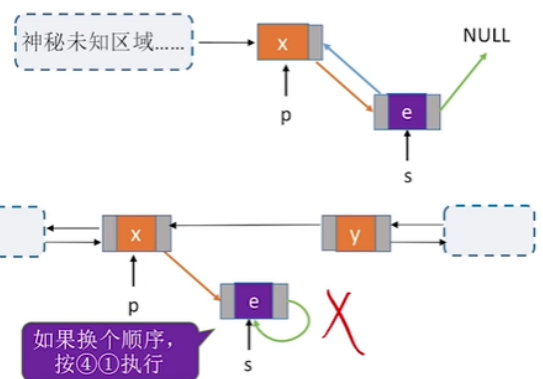




p无后继结点时:

```
//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

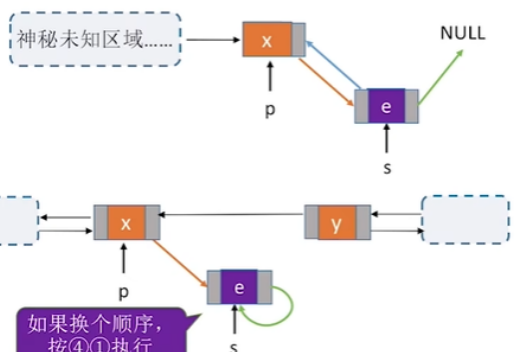


如果换个顺序, 按④①执行

按位序前插操作可以找到前结点后在前结点执行后插操作

```
//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;
}
```

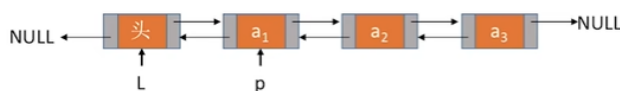
修改指针时要注意顺序



如果换个顺序, 按④①执行

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✓



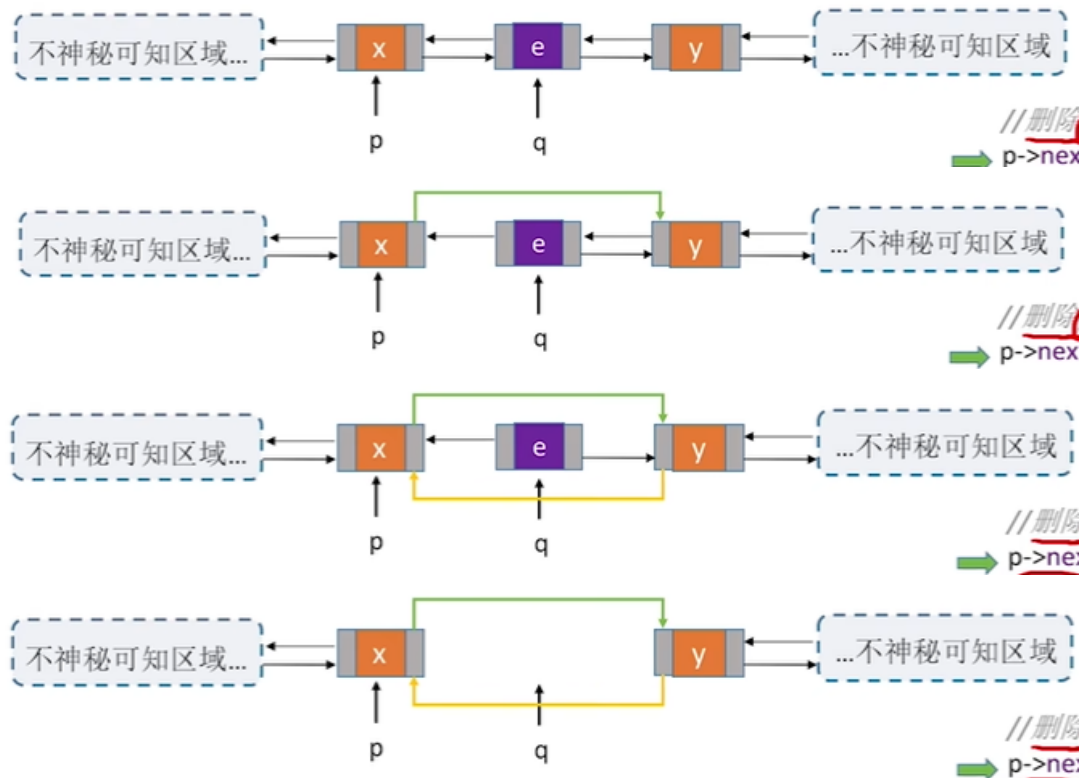
• 双链表的删除

```

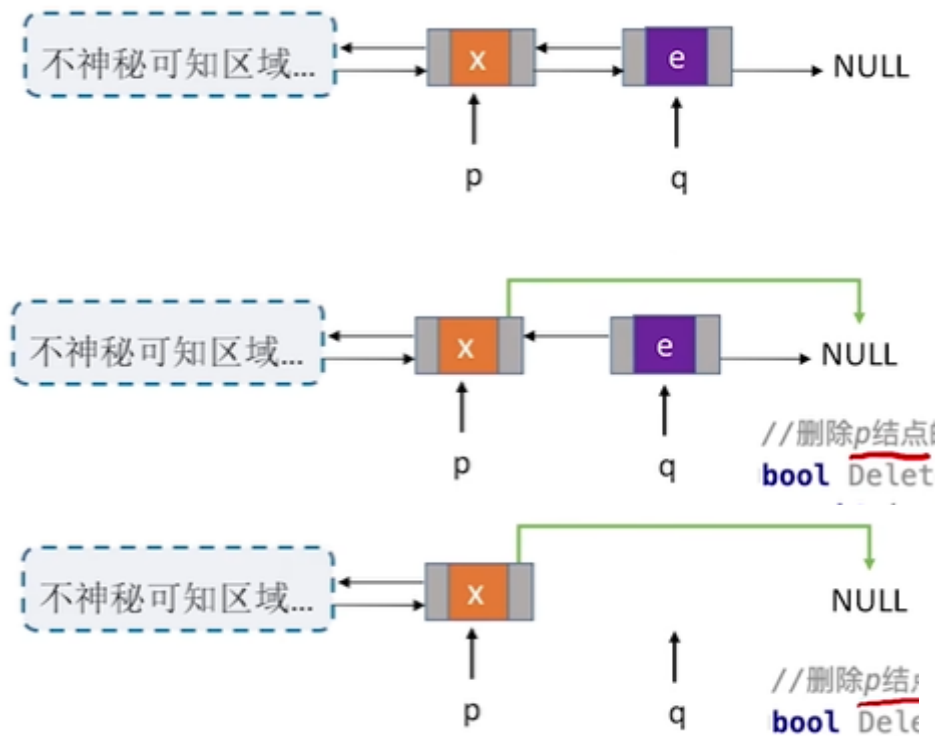
1 //删除p结点的后继结点
2 bool DeleteNextDNode(DNode *p)
3 {
4     if(p == NULL)
5         return false;
6     DNode *q = p->next;    //找到p的后继结点q
7
8     if(q == NULL)          //p没有后继
9         return false;
10    p->next = q->next;
11
12    if(q->next != NULL)     //q结点不是最后一个结点
13        q->next->prior = p;
14    free(q);                //释放节点空间
15    return true;
16 }
17
18 //销毁双链表
19 void DestroyList(DLinkList &L)
20 {
21     //循环释放各个数据结点
22     while(L->next != NULL)
23         DeleteNextDNode(L);
24     free(L);                //释放头结点
25     L = NULL;              //头指针指向NULL
26 }

```

要删除的结点有后继结点：



要删除的结点无后继结点：

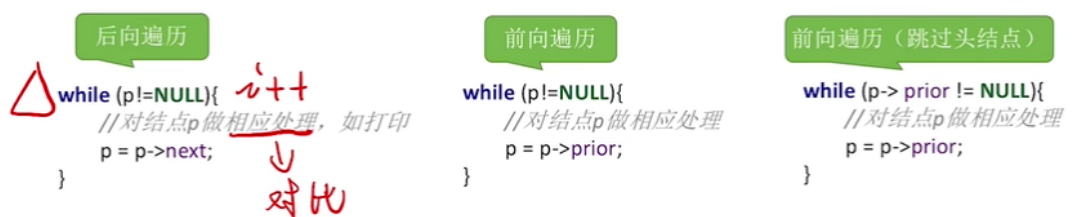


• 双链表的遍历

```

1  //后向遍历
2  while(p != NULL)
3  {
4      //对结点做相应处理，如打印
5      p = p->next;
6  }
7
8  //前向遍历
9  while(p != NULL)
10 {
11     //对结点做相应处理
12     p = p->prior;
13 }
14
15 //前向遍历（跳过头结点）
16 while(p->prior != NULL)
17 {
18     //对结点做相应处理
19     p = p->prior;
20 }

```



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 $O(n)$

• 知识总结

