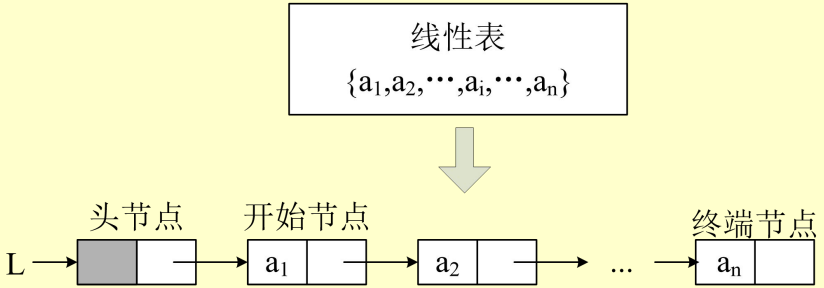
本节主题：

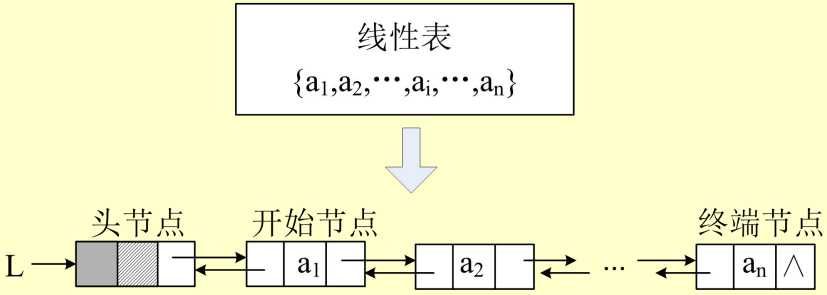
双链表

单链表和双链表

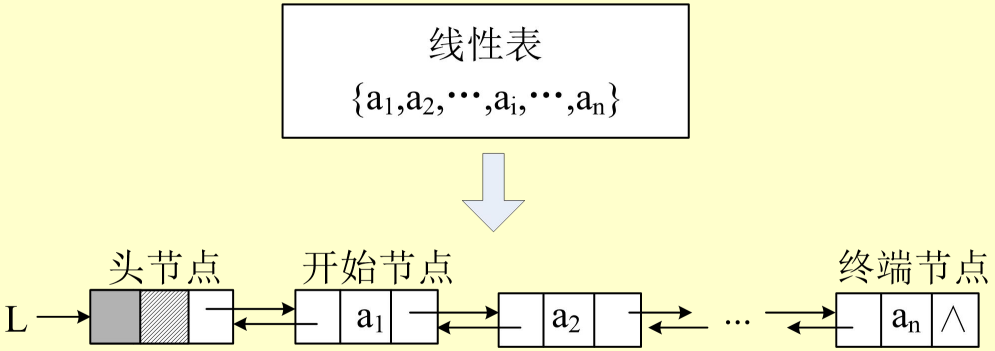
- 单链表
  - 在每个节点中除包含有数据域外，只设置一个指针域，用以指向其后继节点。
  - 别称：线性单向链接表。



- 双链表
  - 在每个节点中除包含有数值域外，设置有两个指针域，分别用以指向其前驱节点和后继节点。
  - 别称：线性双向链接表。
  - 既可以依次向后访问每一个节点，也可以依次向前访问每一个节点。

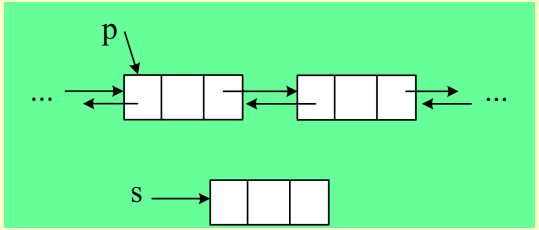


双链表的存储结构

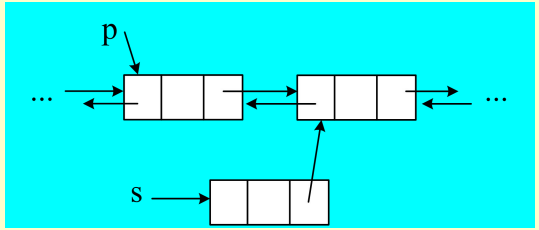



```
typedef struct DNode
{
    ElemType data;
    struct DNode *prior;
    struct DNode *next;
} DLinkList;
```

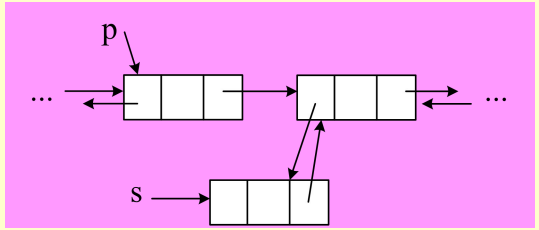
在双链表中插入结点(在*p节点之后插入节点*s)



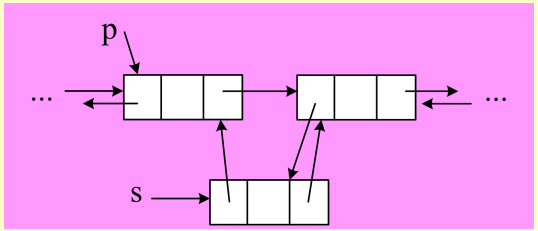
(1)插入前



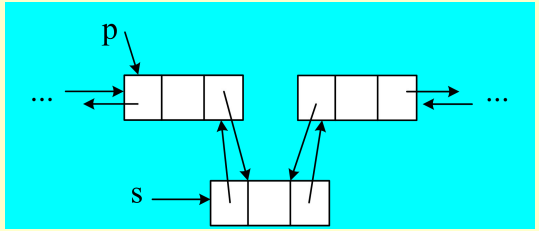
(2) s->next=p->next



(3) p->next->prior=s

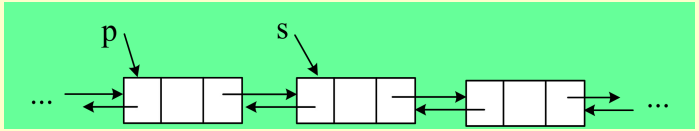


(4) s->prior=p



(5) p->next=s

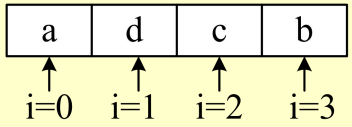
```
s->next=p->next;
p->next->prior=s;
s->prior=p;
p->next=s;
```

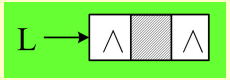


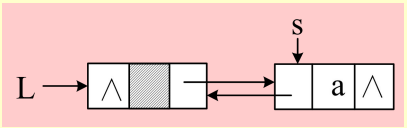
(6)插入后

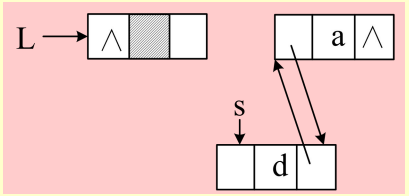
头插法建立双链表

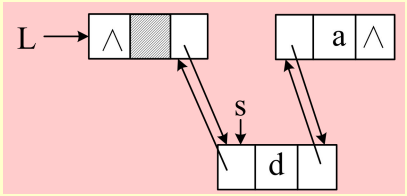
```
void CreateListF(DLinkList *&L,ElemType a[],int n)
{
   DLinkList *s;
   int i;
   L=(DLinkList *)malloc(sizeof(DLinkList));
   L->prior=L->next=NULL;
   for (i=0; i<n; i++)
   {
      s=(DLinkList *)malloc(sizeof(DLinkList));
      s->data=a[i];
      s->next=L->next;
      if (L->next!=NULL)
         L->next->prior=s;
      L->next=s;
      s->prior=L;
   }
}
```

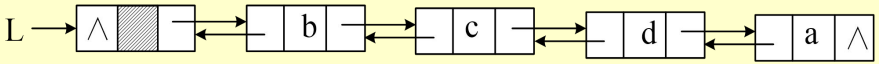






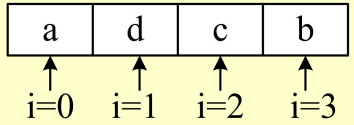


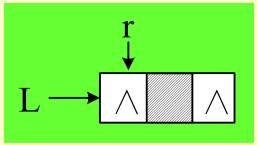


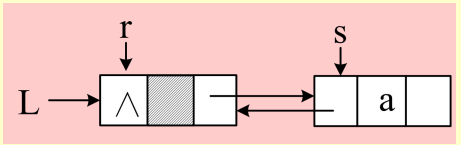


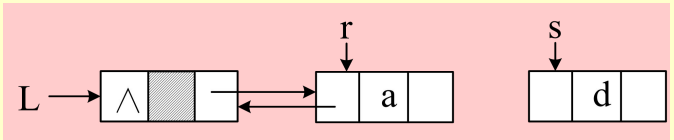
尾插法建立双链表

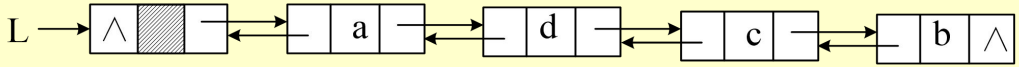
```
void CreateListR(DLinkList *&L,ElemType a[],int n)
{
    DLinkList *s,*r;
    int i;
    L=(DLinkList *)malloc(sizeof(DLinkList));
    r=L;
    for (i=0; i<n; i++)
    {
        s=(DLinkList *)malloc(sizeof(DLinkList));
        s->data=a[i];
        r->next=s;
        s->prior=r;
        r=s;
    }
    r->next=NULL;
}
```





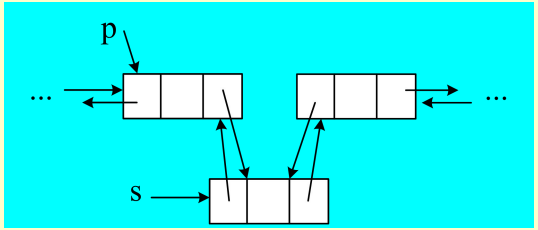






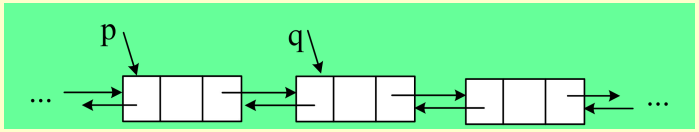
基本运算：插入数据元素ListInsert(&L,i,e)

```
bool ListInsert(DLinkList *&L,int i,ElemType e)
{
    int j=0;
    DLinkList *p=L,*s;
    while (j<i-1 && p!=NULL)
    {
        j++;
        p=p->next;
    }
    if (p==NULL)
        return false;
    else
    {
        s=(DLinkList *)malloc(sizeof(DLinkList));
        s->data=e;
        s->next=p->next;
        if (p->next!=NULL)
            p->next->prior=s;
        s->prior=p;
        p->next=s;
        return true;
    }
}
```

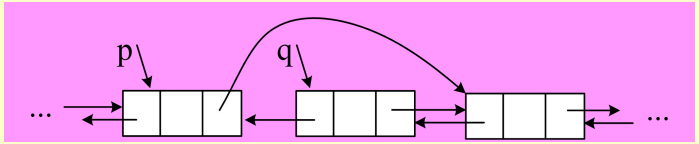



```
s->next=p->next;
p->next->prior=s;
s->prior=p;
p->next=s;
```

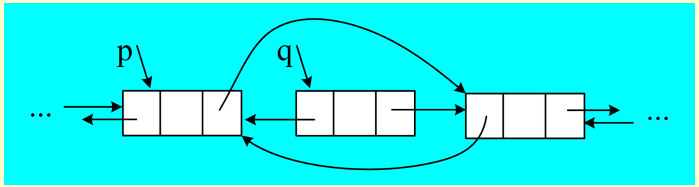
在双链表中删除结点(删除*p节点之后的一个节点)



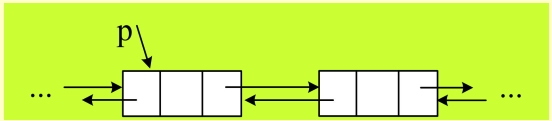
(1)删除前



(2) p->next=p->next->next
或 p->next=q->next



(3) p->next->prior=p

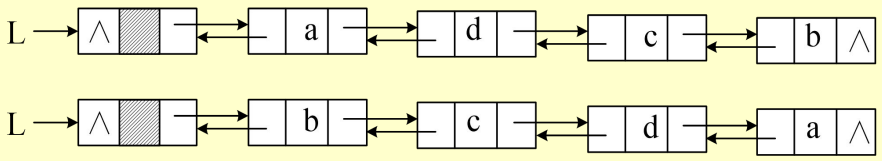


(4)删除后

```
bool ListDelete(DLinkList *&L,int i,ElemType &e)
{
    int j=0;
    DLinkList *p=L,*q;
    while (j<i-1 && p!=NULL)
    {
        j++;
        p=p->next;
    }
    if (p==NULL)
        return false;
    else
    {
        q=p->next;
        if (q==NULL)
            return false;
        e=q->data;
        p->next=q->next;
        if (p->next!=NULL)
            p->next->prior=p;
        free(q);
        return true;
    }
}
```

例：逆置双链表

- 问题
  - 有一个带头节点的双链表L，设计一个算法将其所有元素逆置，即第1个元素变为最后一个元素，第2个元素变为倒数第2个元素，…，最后一个元素变为第1个元素。
- 策略
  - 采用头插法建表
- 算法




```
void reverse(DLinkList *&L)
{
    DLinkList *p=L->next,*q;
    L->next=NULL;
    while (p!=NULL)
    {
        q=p->next;
        p->next=L->next;
        if (L->next!=NULL)
            L->next->prior=p;
        L->next=p;
        p->prior=L;
        p=q;
    }
}
```