4、单链表的算法设计方法

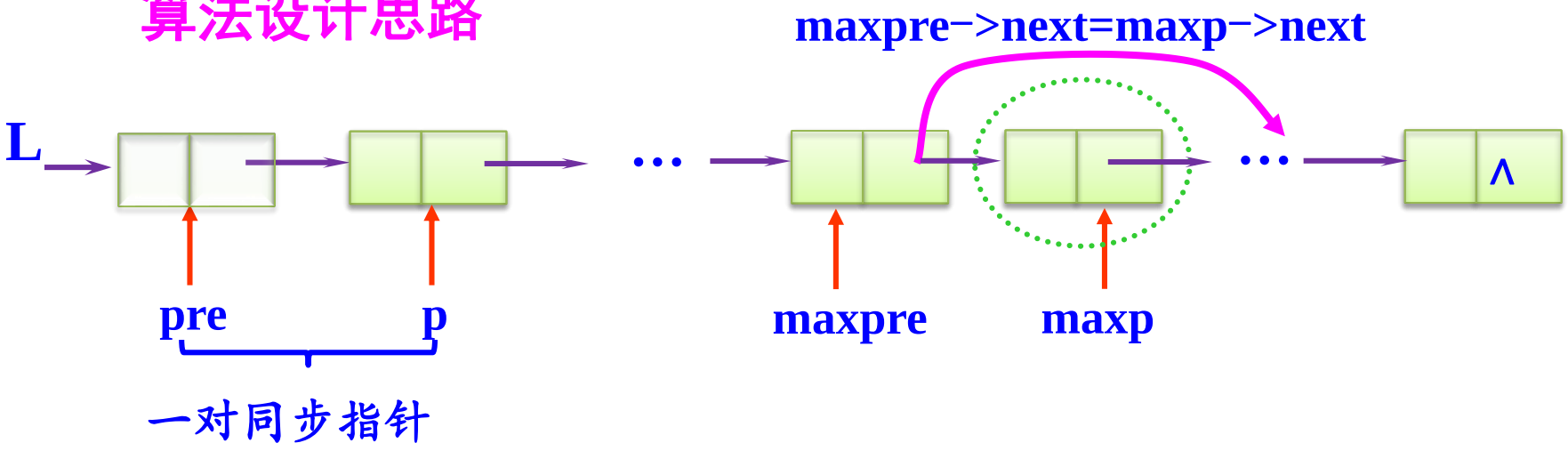
单链表的算法设计是线性表链式存储结构算法设计的基础，是需要重点掌握的内容。这里总结一般的算法设计方法。

❶ 以查找为基础的算法设计

- 按照条件进行节点查找；
- 进行插入或者删除操作。

【例2-3】设计一个算法，删除一个单链表L中元素值最大的节点（假设最大值节点是唯一的）。

算法设计思路



```
void delmaxnode(LinkList *&L)
{    LinkList *p=L->next,*pre=L,*maxp=p,*maxpre=pre;
     while (p!=NULL)
     {    if (maxp->data<p->data) //若找到一个更大的节点
          {    maxp=p;             //更改maxp
               maxpre=pre;         //更改maxpre
          }
          pre=p;                   //p、pre同步后移一个节点
          p=p->next;
     }
     maxpre->next=maxp->next;  //删除*maxp节点
     free(maxp);               //释放*maxp节点
}
```

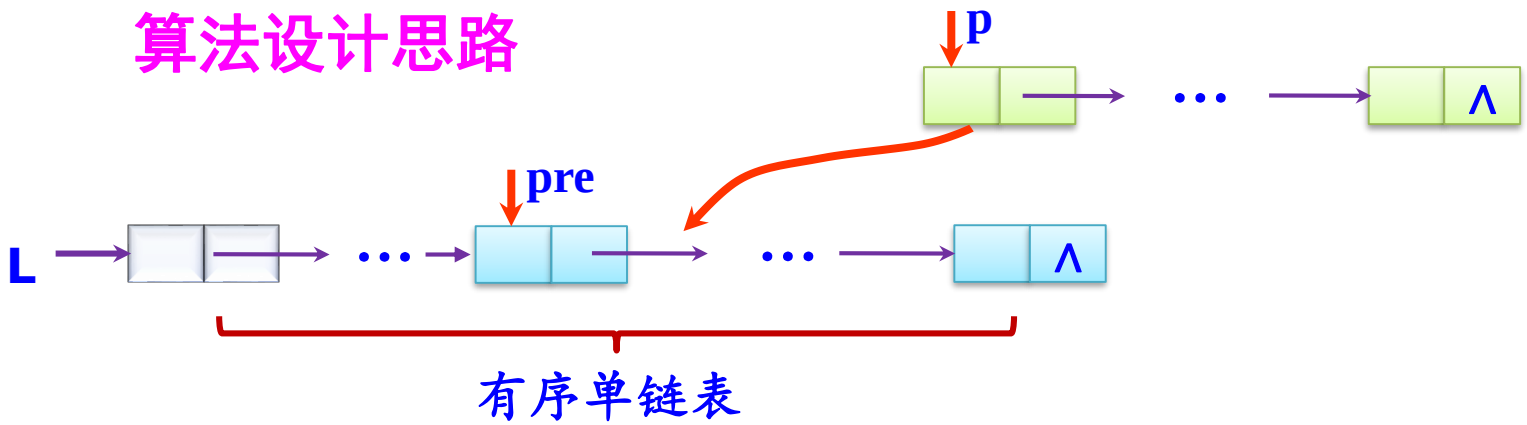
查找最大值节点的前趋节点*maxpre

删除最大值节点并释放空间

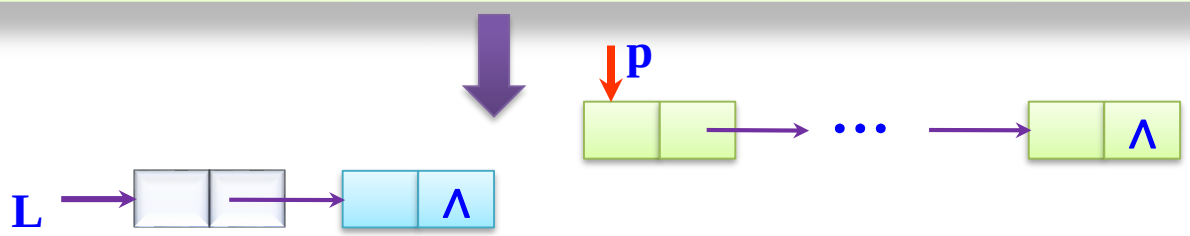
该算法的时间复杂度为O(*n*)。

【例2-4】有一个带头节点的单链表L（至少有一个数据节点），设计一个算法使其元素递增有序排列。

算法设计思路



```
void sort(LinkList *&L)
{
    LinkList *p,*pre,*q;
    p=L->next->next;              //p指向L的第2个数据节点
    L->next->next=NULL;           //构造只含一个数据节点的有序表
```



含一个数据节点的单链表是有序单链表

将L拆分为两个部分

```
    while (p!=NULL)
    {    q=p->next;          //q保存*p节点后继节点的指针
         pre=L;              //从有序表开头进行比较,pre指向插入*p的前趋节点
         while (pre->next!=NULL && pre->next->data<p->data)
              pre=pre->next;          //在有序表中找插入*p的前趋节点*pre
         p->next=pre->next;
         pre->next=p;
         p=q;                        //扫描原单链表余下的节点
    }
}
```

在有序单链表中查找插入节点的前趋节点*pre

在*pre之后插入*p

该算法的时间复杂度为$O(n^2)$。

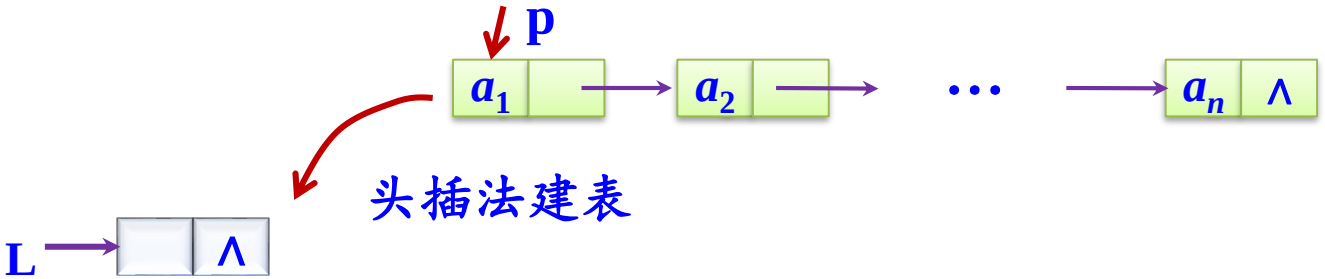
❷ 以建表算法为基础的算法设计

- 单链表有尾插法和头插法两种建表算法。
- 很多算法是以这两个建表算法为基础进行设计的。

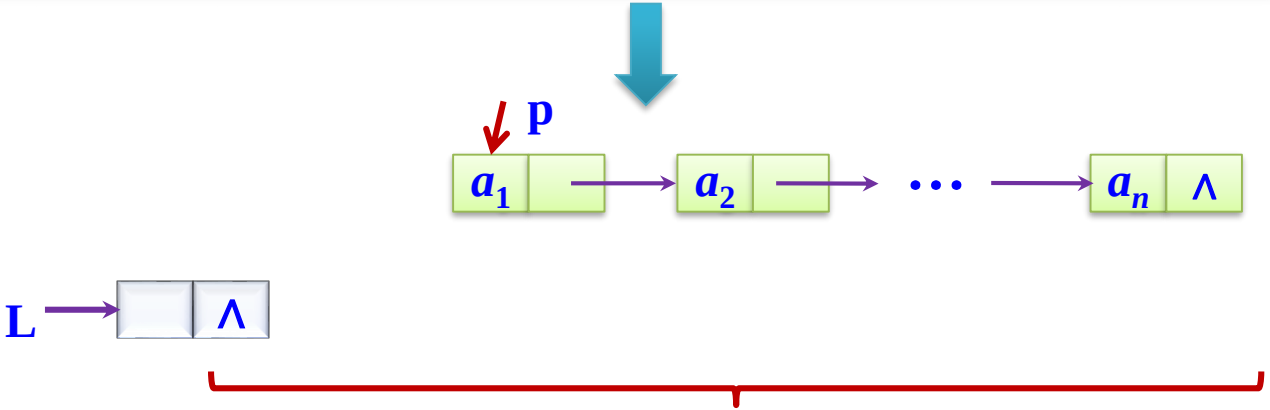
【例2-5】 假设有一个带头节点的单链表L={a_1,a_2,…,a_n}。设计一个算法将所有节点逆置，即：

L={a_n，a_{n-1}，…，a_1}

算法设计思路

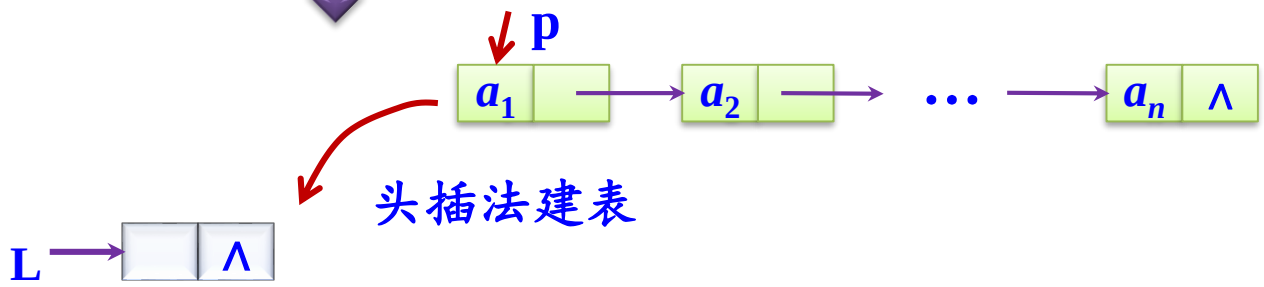


```
void  Reverse(LinkList *&L)
{
      LinkList *p=L->next,*q;
      L->next=NULL;
```

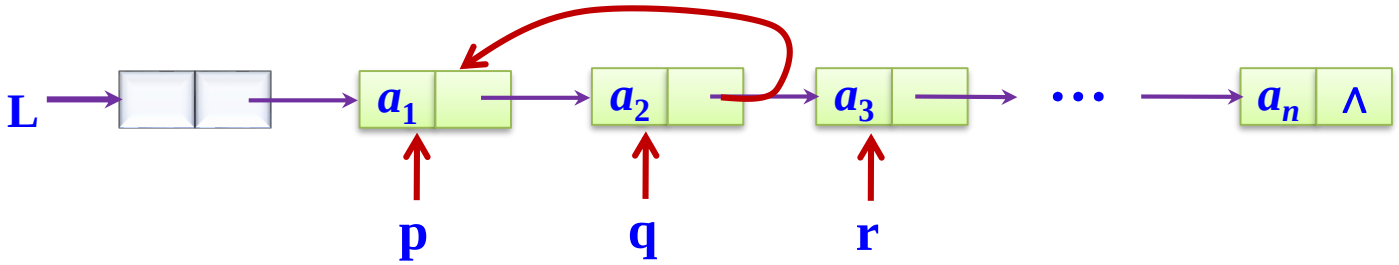


将L拆分为两个部分

```
    while  (p!=NULL)
    {     q=p->next;
          p->next=L->next;
          L->next=p;
          p=q;
    }
}
```



另一种解法

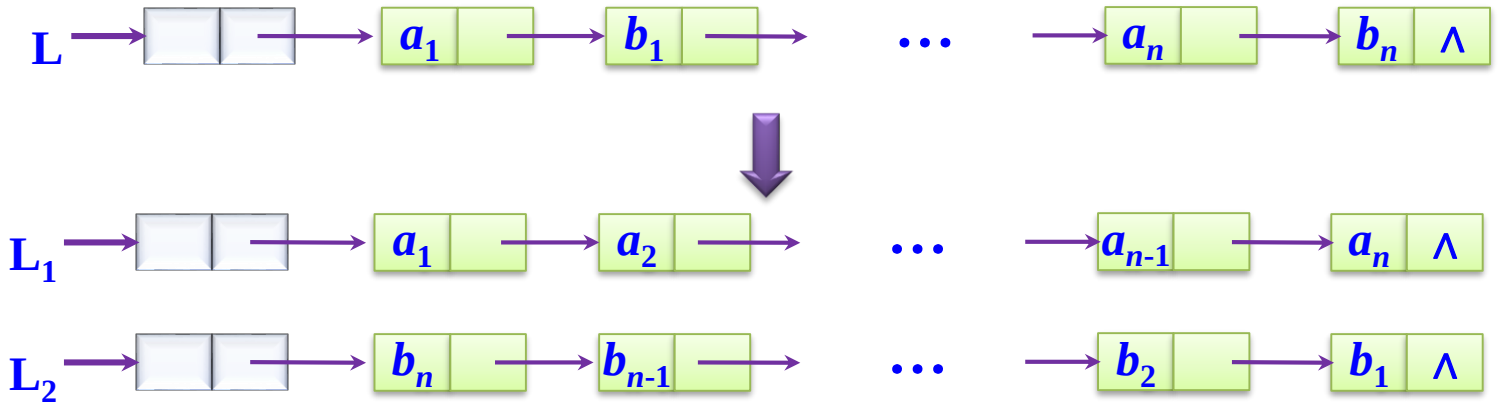


这种解法远不如前面解法清晰！

【例2-6】 假设有一个带头节点的单链表L={a_1，b_1，a_2，b_2，…，a_n，b_n}。设计一个算法将其拆分成两个带头节点的单链表L1和L2：

L1={a_1，a_2，…，a_n}，L2={b_n，b_{n-1}，…，b_1}

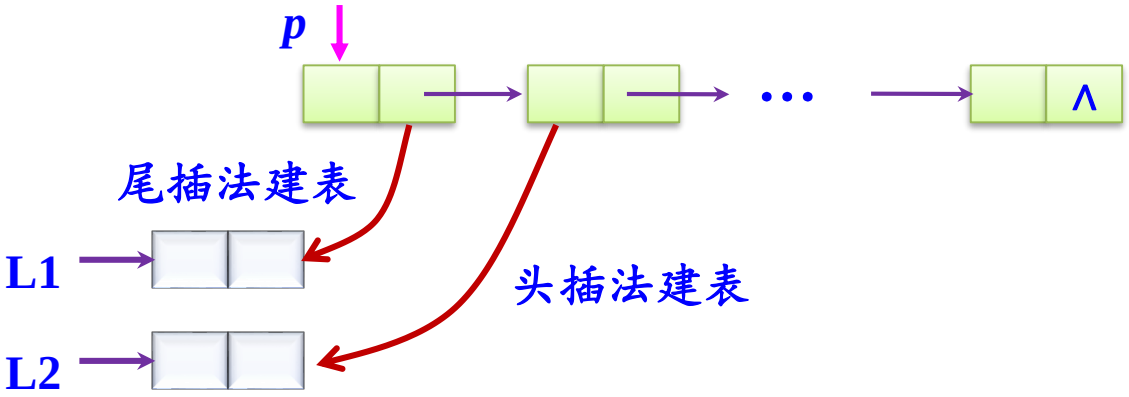
要求L1使用L的头节点。



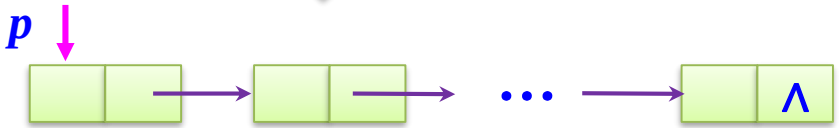
解：利用原单链表L中的所有节点通过改变指针域重组成单链表L1和L2。

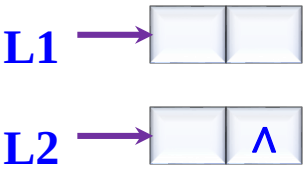
由于L1中节点的相对顺序与L中的相同，所以采用尾插法建立单链表L1；

由于L2中节点的相对顺序与L中的相反，所以采用头插法建立单链表L2。

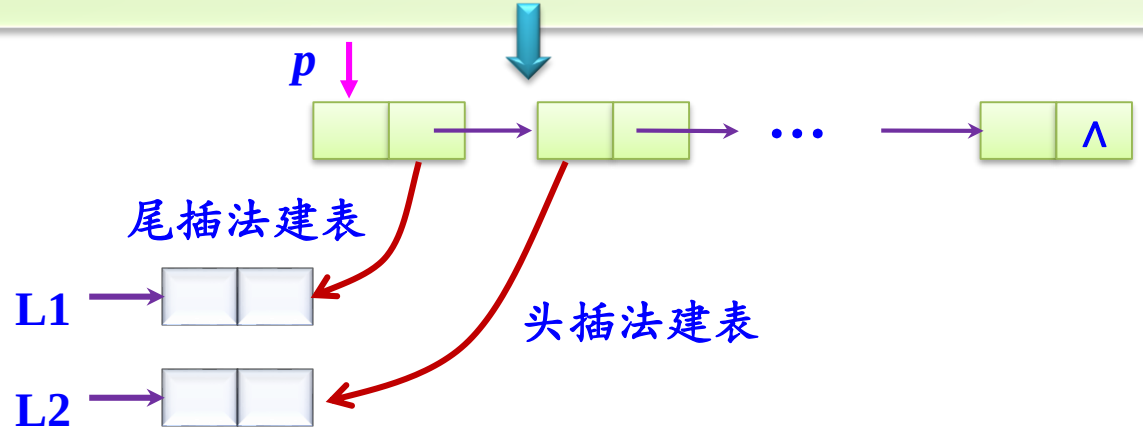


```
void split(LinkList *&L,LinkList *&L1,LinkList *&L2)
{    LinkList *p=L->next,*q,*r1;        //p指向第1个数据节点
     L1=L;                              //L1利用原来L的头节点
     r1=L1;                             //r1始终指向L1的尾节点
     L2=(LinkList *)malloc(sizeof(LinkList));   //创建L2的头节点
     L2->next=NULL;                     //置L2的指针域为NULL
```





```
    while (p!=NULL)
    {       r1->next=p;                 //采用尾插法将*p(data值为ai)插入L1中
            r1=p;
            p=p->next;                  //p移向下一个节点(data值为bi)
            q=p->next;                  //用q保存*p的后继节点
            p->next=L2->next;           //采用头插法将*p插入L2中
            L2->next=p;
            p=q;                        //p重新指向a(i+1)的节点
    }
    r1->next=NULL;                      //尾节点next置空
}
```



思考题

假设有一个带头节点的单链表L，每个节点值由单个数字、小写字母和大写字母构成。设计一个算法将其拆分成3个带头节点的单链表L1、L2和L3，L1包含L中的所有数字节点，L2包含L中的所有小写字母节点，L3包含L中的所有大写字母节点。

该算法如何设计？

——本讲完——