数据结构

深圳技术大学
大数据与互联网学院

练习

一. 写出以下代码的时间复杂度

a.
```
for (i=1; i< POW(2, n); i++)
   //POW(x, y)函数表示x的y次幂
    a = a+100;
```

b.
```
for  (i=1; i<=n; i++)
    for (j=1; j<=i; j++)
        a = a + 1;
```

c.
```
i = s = k = 1;
while ( k <= s ) {
        if (i<=n) { i++;  s*=i; }
        k ++;
}
```

第二章 线性表

2.1 线性表的类型定义
2.2 线性表的顺序表示和实现
2.3 线性表的链式表示和实现
2.4 一元多项式的表示和实现

2.1 线性表的类型定义

一. 线性表概念

- 线性表是n个数据元素的有限序列
- 线性数据结构的特点
  - 数据同一性，同一个线性表的数据属同一类数据对象
  - 顺序性，数据之间存在序偶关系

$$(a_1,\ a_2,\ \ldots a_{i-1},\ a_i,\ a_{i+1},\ \ldots a_{n-1},\ a_n)$$

其中，a_i是表中元素，i表示元素a_i的位置，n是表的长度

2.2 线性表的类型定义

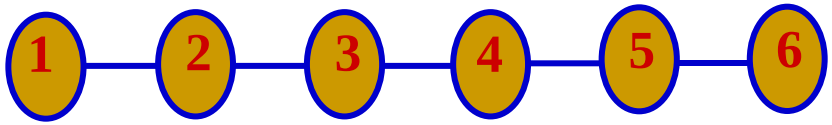
一. 线性表概念

- 数据同一性
  - 线性表中的元素具有相同的特性，属于同一数据对象，如：
  - 26个字母的字母表：(A, B, C, D, …, Z)
  - 近期每天的平均温度：(30℃，28℃，29℃, …)

2.1 线性表的类型定义

一. 线性表概念

- 数据的顺序性
  - 存在惟一的一个被称作“第一个”的数据元素(如“1”)
  - 存在惟一的一个被称作“最后一个”的数据元素(如“6”)
  - 除第一个元素外，每个数据元素均只有一个前驱
  - 除最后一个元素外，每个数据元素均只有一个后继(next)(如“1”的next是“2”，“2”的next是“3”)



2.1 线性表的类型定义

二. 线性表的ADT定义

ADT List{

数据对象：数据元素同属一个集合

数据关系：序偶关系

基本操作：

Init创建、Destroy销毁、

Clear清空、Empty是否为空、

Length取表长度、Get取表元素、Locate查找元素

Prior取元素前驱、Next取元素后继

Insert插入元素、Delete删除元素

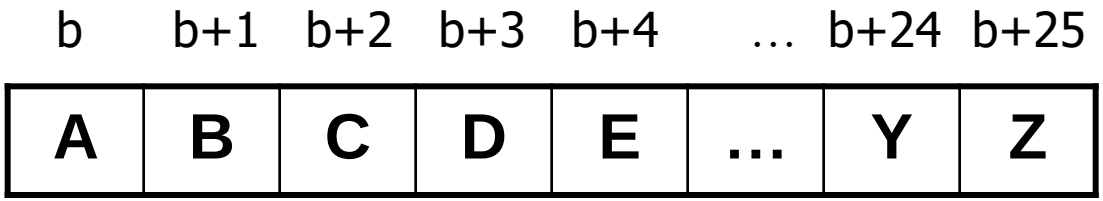
Traverse遍历表

}

2.2 顺序表

一. 顺序表概念

- 顺序表是线性表的顺序表示，用一组地址连续的存储单元依次存储线性表的数据元素



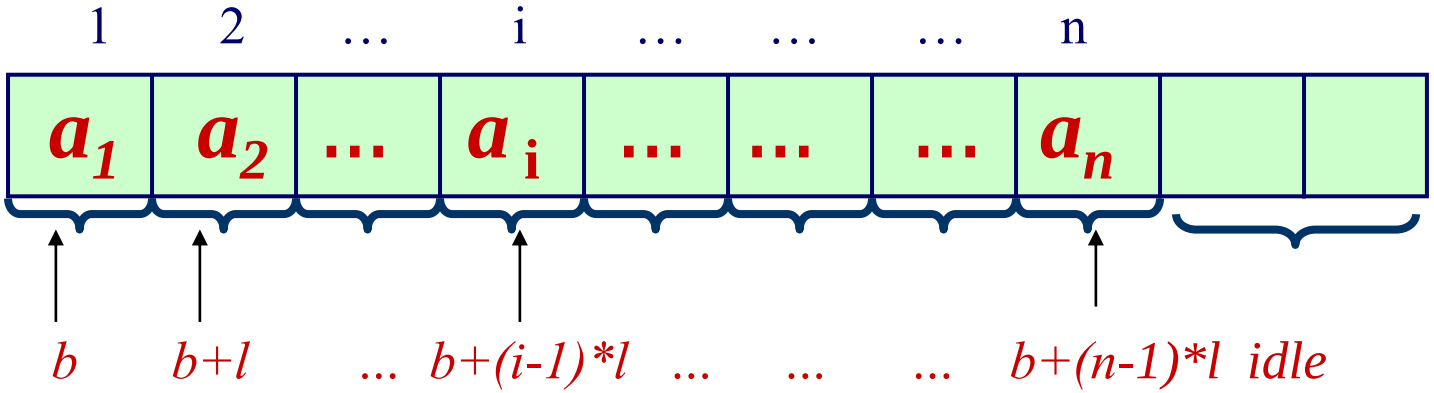
2.2 顺序表

二. 顺序表的数据位置

- 顺序表数据元素的位置：

$LOC(a_i) = LOC(a_{i-1}) + l$

$LOC(a_i) = LOC(a_1)+(i-1)*l$　　*l表示元素占用的内存单元数*



2.2 顺序表

三. 顺序表的定义

- 采用C++语言中动态分配的一维数组表示顺序表

```
#define LIST_INIT_SIZE  100        // 线性表存储空间的初始分配量
#define LISTINCREMENT  10  // 线性表存储空间的分配增量

template<class ElemType>
class Sqlist {
    public:
    ElemType  *elem;  // 存储空间基址
    int  length;                  // 顺序表当前实际保存元素数
    int  listsize;           //顺序表当前分配的存储容量数
    ... ... // 各种构造函数与析构函数
    Status InitList_Sq(int l=LIST_INIT_SIZE);   // 创建顺序表
    Status ListInsert_Sq(int i, ElemType &e);   // 插入元素
    Status ListDelete_Sq(int i, ElemType &e);   // 删除元素
    ... ... // 其他成员函数
}
```

2.2 顺序表

三. 顺序表的创建

```
template<class ElemType>
Status Sqlist<ElemType>::InitList_Sq(int l) {
    listsize = l + LISTINCREMENT;  // 初始存储容量为参数值加上递增值
    elem = new ElemType[listsize]; // 动态分配
    if (!elem) exit(OVERFLOW);    // 存储分配失败
    length = 0;                   // 空表长度为0
    return OK;
};
```

Status可以是枚举类型：

```
enum Status {OK, ERROR, OVERFLOW, ...};
```

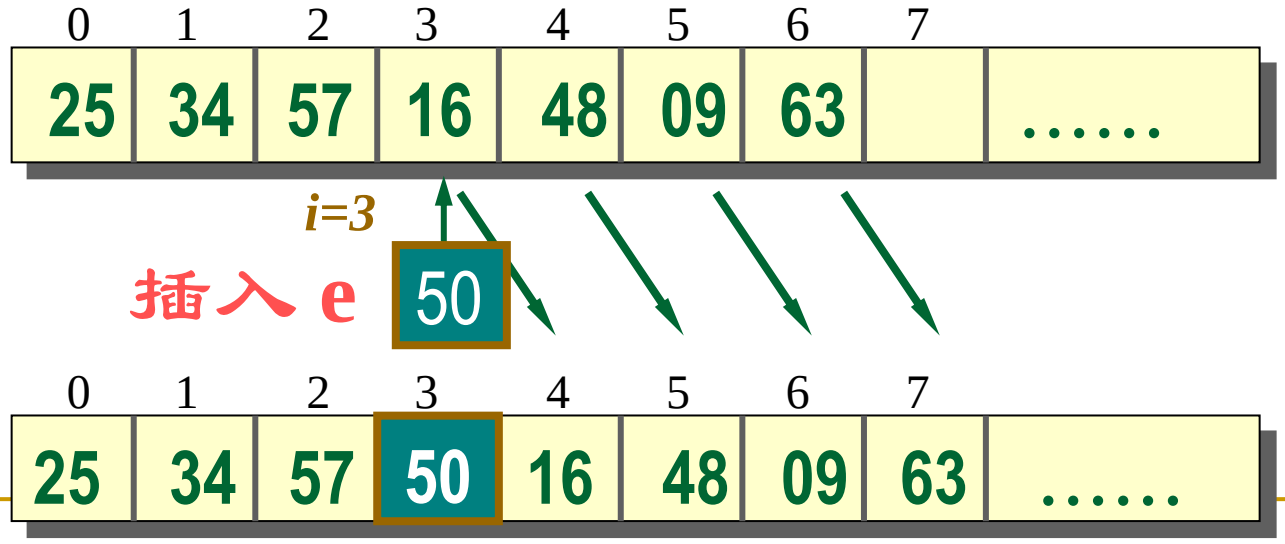
即相当于定义int常量OK=0, ERROR=1, OVERFLOW=2, …

这样可以增加代码的可读性

2.2 顺序表

四. 顺序表的插入

- 给出插入的队列对象、位置、数据
  - 在顺序表的第i-1个数据元素和第i个数据元素之间插入一个新的数据元素
  - 操作包括后移、插入、长度+1
  - 例如在第3个元素与第4个元素之间插入新元素e，需要将最后元素n至第4元素(共7-4+1)都向后移一位置，长度加1



2.2 顺序表

四. 顺序表的插入

```
template<class ElemType>
Status SqList<ElemType>::ListInsert_Sq(int i, ElemType &e)
{
    if (i<0 || i>length) return OUT_OF_INDEX;
    if (length >= listsize) {  // 当前已经使用了已分配的全部空间
        listsize += LISTINCRMENT;  // 增加分配的空间
        ElemType* newbase = new ElemType[listsize];
        if (!newbase) exit(OVERFLOW);
        memcpy(newbase, elem, length*sizeof(ElemType));  //
复制原来的数据
        elem = newbase;
    }     // 以上皆为准备阶段
    for (int p=length-1; p>=i; --p)
        elem[p+1] = elem[p];     // 循环右移，腾出空间
    elem[i] = e;
    ++length;
    return OK;
} // ListInsert_Sq
```

2.2 顺序表

四. 顺序表插入的时间复杂度

- 在顺序表中插入一个元素，需要向后移动元素个数为：n-i+1
- 平均移动元素数为：

$$E_{is} = \sum_{i=1}^{n+1} p_i \times (n-i+1)$$

- 当插入位置等概率时，$p_i=1/(n+1)$，因此：

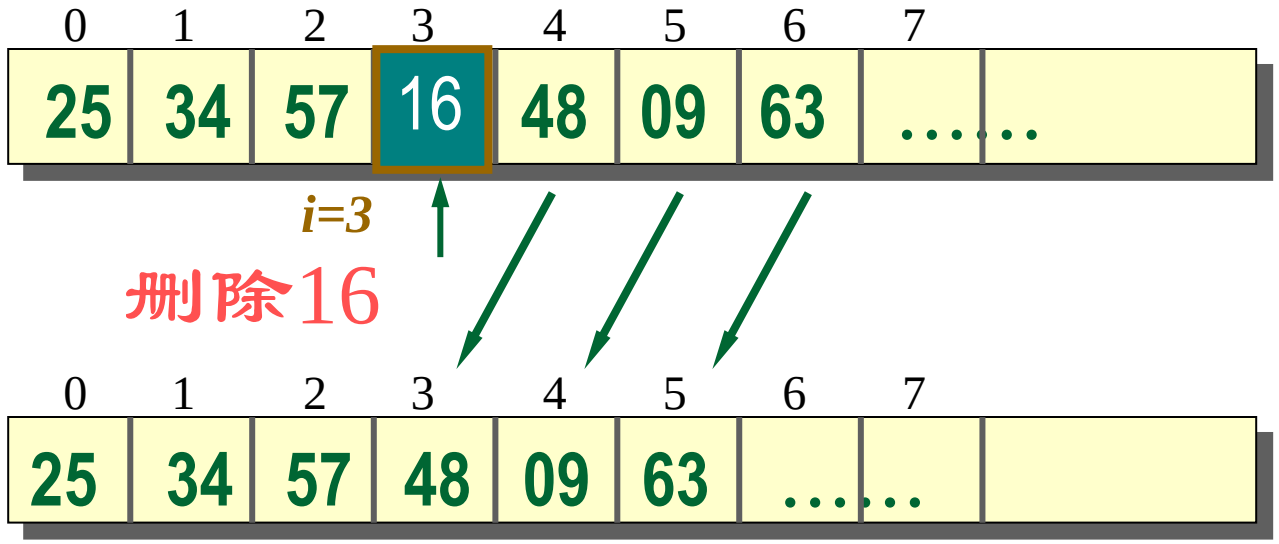
$$E_{is} = \sum_{i=1}^{n+1} [1/(n+1)] \times (n-i+1) = n/2$$

- 顺序表插入操作的时间复杂度为O(n)

2.2 顺序表

五. 顺序表的删除

- 给出删除的队列对象、位置
  - 指将顺序表的第i个数据元素删除
  - 操作包括删除、前移、长度-1
  - 例如将第4个元素删除，需要将最后元素n至第5元素(共7-4)都向前移一位置，长度减1



2.2 顺序表

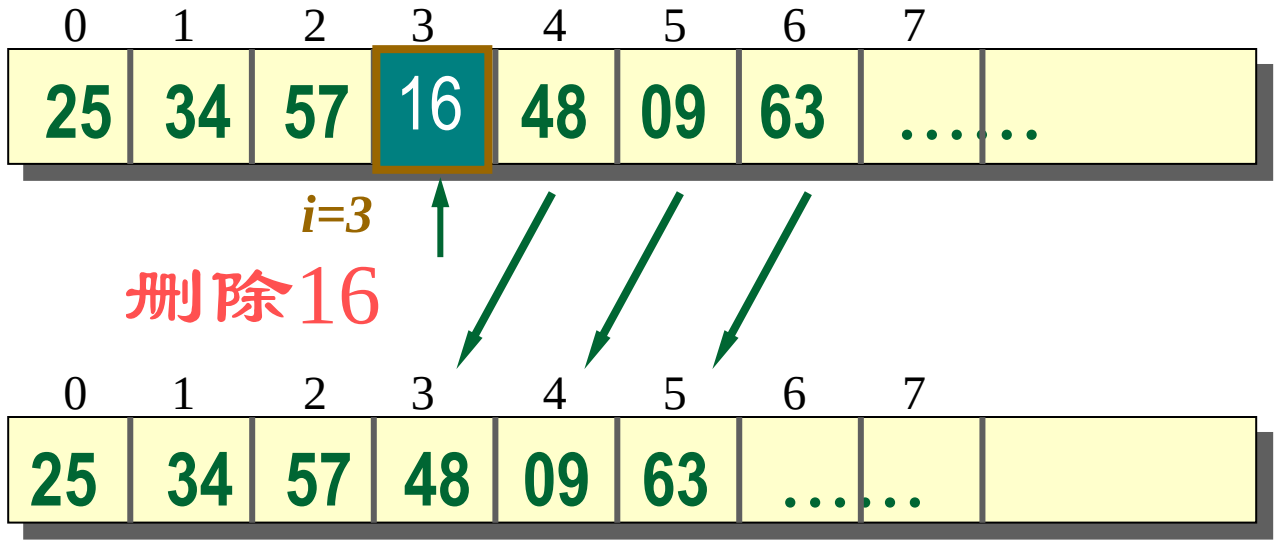
五. 顺序表的删除

```
template<class ElemType>
Status SqList<ElemType>::ListDelete_Sq(int i, ElemType &e)
{
    if (i<0 || i>=length) return OUT_OF_INDEX;
    e = elem[i];
    for (int p=i; p<length; ++p)
        elem[p] = elem[p+1];      // 左移
    --length;                     // 表长减1
    return OK;
} // ListDelete_Sq
```

2.2 顺序表

五. 顺序表的删除

- 给出删除的队列和数据
  - 将顺序表的数值为x的第一个元素进行删除
  - 操作包括比较、前移（删除）、长度-1
  - 删除数值为16的首个元素



2.2 顺序表

五. 顺序表的删除

- 给出删除的队列和数据

```
Status Locate_Delete_SqList(Sqlist *L, ElemType x){
    //  删除线性表L中值为x的第一个结点
    int  i=0 , k ;
    while  (i<L->length) {      // 查找值为x的第一个结点
        if  (L->elem[i]!=x )  i++ ;
        else {
            for (k=i+1; k< L->length; k++)
                L->elem[k-1]=L->elem[k];
            L->length--;
            break ;
        }
    }
    if  (i>=L->length) {  return NOT_FOUND ;  }
    return  OK;
}
```

2.2 顺序表

五. 顺序表删除的时间复杂度

- 在顺序表中删除一个元素，需要向前移动元素个数为：n-i，平均移动元素数为：

$$E_{dl} = \sum_{i=1}^{n} q_i \times (n-i)$$

- 当删除位置等概率时，$q_i=1/n$，因此：

$$E_{dl} = \sum_{i=1}^{n} [1/n] \times (n-i) = (n-1)/2$$

- 顺序表删除操作的时间复杂度为O(n)

2.2 顺序表

五. 顺序表的合并

- 合并两个递增有序的队列，生成一个新的有序队列
  - 已知两个顺序表a和b，合并成c，三者都是递增有序

1. 设定指针pa、pb、pc分别指向顺序表a、b、c的起始位置
2. 循环，条件是pa和pb都没到末尾
   比较pa和pb指向表a和表b的元素
   。。。。。。。。
3. 把表a或表b的剩余元素复制到表c

结束

2.2 顺序表

六. 顺序表的优缺点

- 优点：
  - 元素可以随机存取
  - 元素位置可用一个简单、直观的公式表示并求取
- 缺点：
  - 在作插入或删除操作时，需要移动大量元素
  - 因此引入链表，减少移动操作

2.3 链表

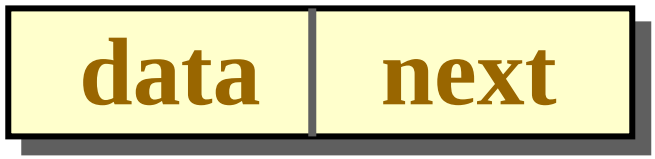
一. 链表的概念

- 链表是线性表的链式存储表示
- 链表中逻辑关系相邻的元素不一定在存储位置上相连，用一个链(指针)表示元素之间的邻接关系
- 线性表的链式存储表示主要有三种形式：
  - 线性链表
  - 循环链表
  - 双向链表

2.3 链表

二. 线性链表

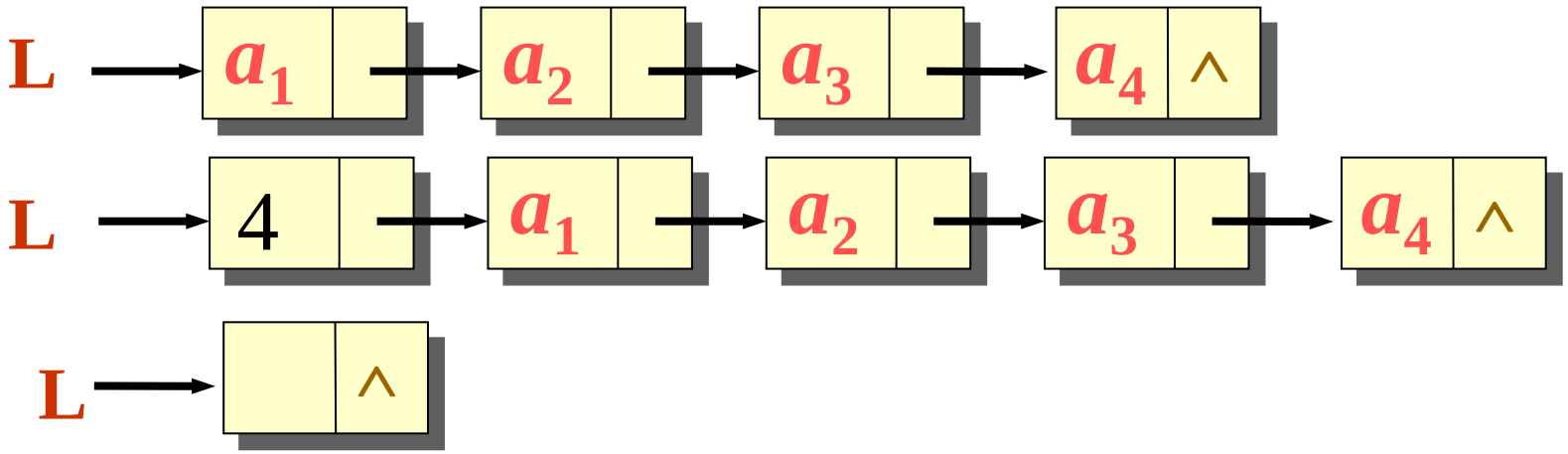
- 线性链表的元素称为结点(node)
- 结点除包含数据元素信息的数据域外，还包含指示直接后继的指针域
- 每个结点，在需要时动态生成，在删除时释放



2.3 链表

三. 单链表

- 每个结点只包含一个指针域的链表称为单链表
- 单链表可由头指针惟一确定
- 单链表最后一个元素的指针域为空（NULL）
- 为了操作方便，有时在线性链表的第一个结点之前附设一个头结点，其数据域可以为空，也可以为线性链表的长度信息。



2.3 链表

三. 单链表

- 单链表的定义

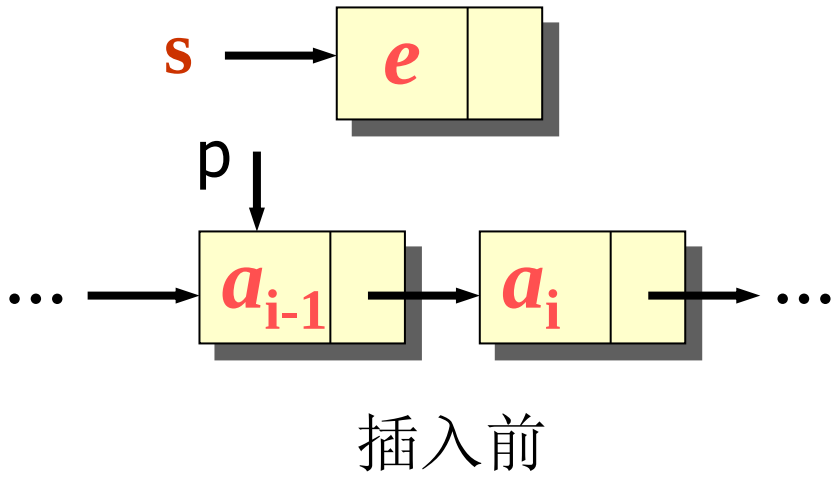
```
//定义一个链表的结点
template<class ElemType>
class LNode {
    ElemType data;                  // 数据域
    LNode *next = nullptr;          // 后继指针
};
```


```
// 定义一个链表
template<class ElemType>
class LinkList {
    LNode<ElemType> head;   // 头结点
    ... ... // 其他数据成员
    ... ... // 成员函数
}
```

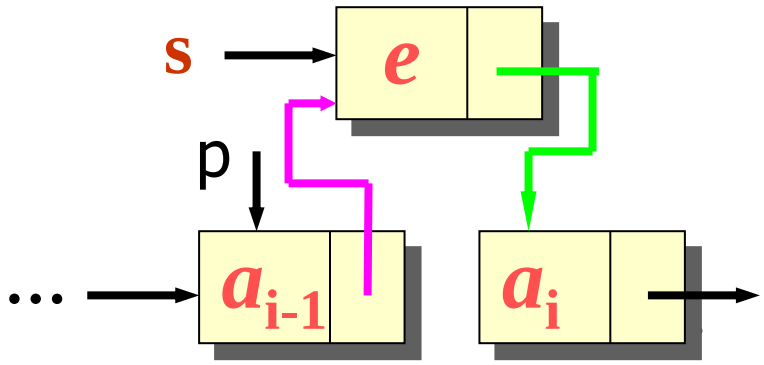
2.3 链表

三. 单链表

- 单链表的插入是在链表的第i-1元素与第i元素之间插入一个新元素



插入前



插入后

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

2.3 链表

三. 单链表

- 单链表插入的代码

```
template<class ElemType>
Status LinkList<ElemType>::Insert_L(int i, ElemType &e) {
    // 在带头结点的单链表L中第i个位置插入元素e
    LNode<ElemType> *p = &head;
    int j = 0;
    while (p && j<i) { p = p->next; j++; }  // 寻找i-1结点
    if (!p && j<i) return ERROR;
    LNode *s = new LNode<ElemType>;         // 生成新结点
    s->data = e; s->next = p->next; p->next = s;       //
将指定数据插入到i-1的next位置
    return OK;
} // ListInsert_L
```

2.3 链表

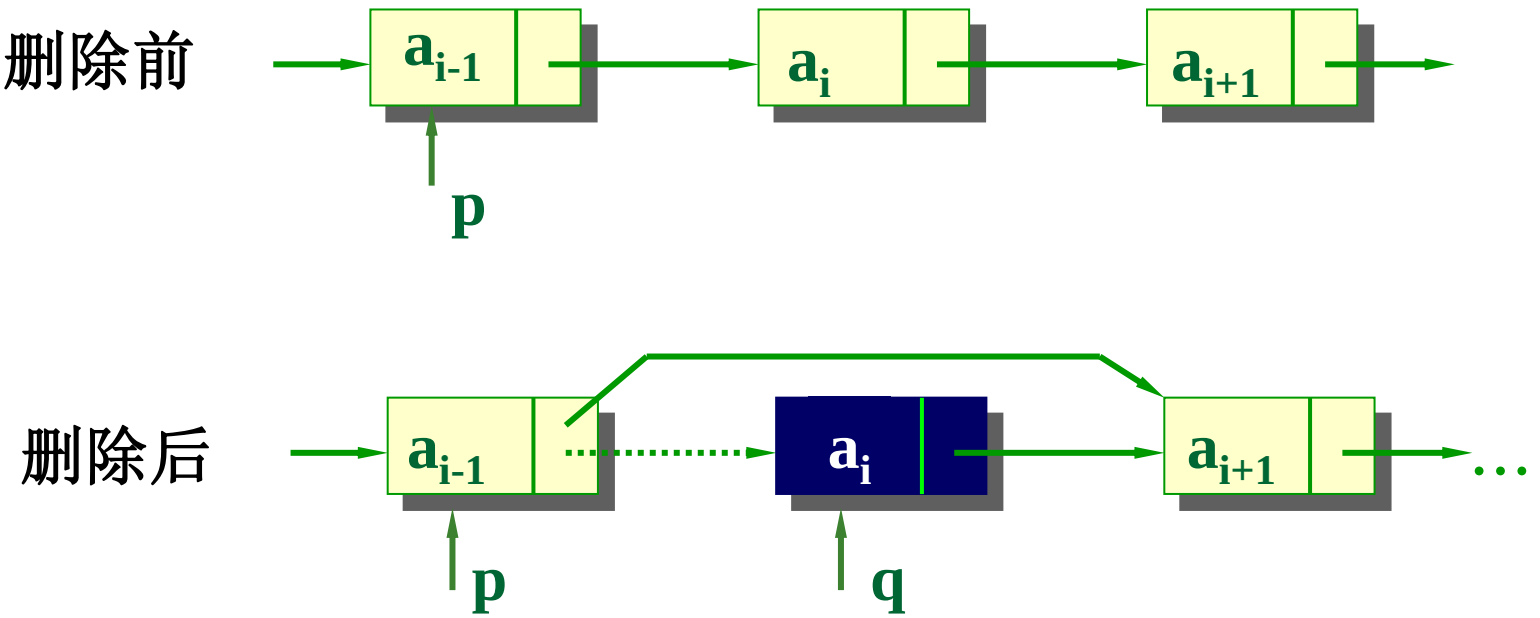
三. 单链表

- 单链表插入的算法时间复杂度主要取决于while循环中的语句频度
- 频度与在线性链表中的元素插入位置有关，因此线性链表插入的时间复杂度为O(n)

2.3 链表

三. 单链表

- 单链表的删除是将第i元素删除



- p->next = p->next ->next

2.3 链表

三. 单链表

- 单链表删除的代码

```
template<class ElemType>
Status LinkList<ElemType>::ListDelete_L(int i, ElemType
&e) {
    // 在带头结点的单链表L中，删除第i个位置的元素
    LNode<ElemType> *p = &head; int j = 0;
    while (p && j<i) { p = p->next; j++; }  // 寻找i-1结
点
    if (!p && j<i) return ERROR;
    q = p->next; p->next = q->next;    // 删除i结点
    e = q->data;                       // 取值
    delete q;                          // 释放结点
    return OK;
} // ListDelete_L
```

2.3 链表

三. 单链表

- 单链表删除的算法时间复杂度主要取决于while循环中的语句频度
- 频度与在线性链表中的删除位置有关，因此线性链表删除的时间复杂度为O(n)

2.3 链表

三. 单链表

- 单链表的查找
- 按序号查找，取单链表中的第i个元素
  - 不能象顺序表中那样直接按序号i访问结点，而只能从链表的头结点出发，沿链域next逐个结点往下搜索，直到搜索到第i个结点为止。因此，链表不是随机存取结构
- 按值查找，在链表中查找是否有结点值等于给定值key的结点序号查找

2.3 链表

三. 单链表

- 单链表按位置查找的代码

```
template<class ElemType>
ElemType LinkList<ElemType>::Get_Elem(int i) {
    int j=0;
    LNode<ElemType> *p = head.next;
    while (p && j<i) { p=p->next; j++; }
    if(j!=i)  return ERROR;
    else  return(p->data);
}
```

2.3 链表

三. 单链表

- 单链表按值查找的代码

```
template<class ElemType>
LNode *Locate_Node(LNode *L, ElemType &e){
    LNode<ElemType> *p = head.next;
    while ( p && p->data!=key ) p=p->next;
    if (p->data==key)    return p;
    else {
          cerr << "所要查找的结点不存在!!\n" << endl;
        retutn NULL;
    }
}
```

2.3 链表

三. 单链表

- 遍历整个单链表的代码框架

```
template<class ElemType>
ElemType Traverse(LNode *L){
    LNode<ElemType> *p = head.next; //使p指向第一个结点
    while  (p!=NULL)  {    x
        //.....你要执行的代码写这里
        cout << p->data << endl;
        p=p->next;
      }
    // ..... 后续操作代码写这里
    return  0;
}
```

2.3 链表

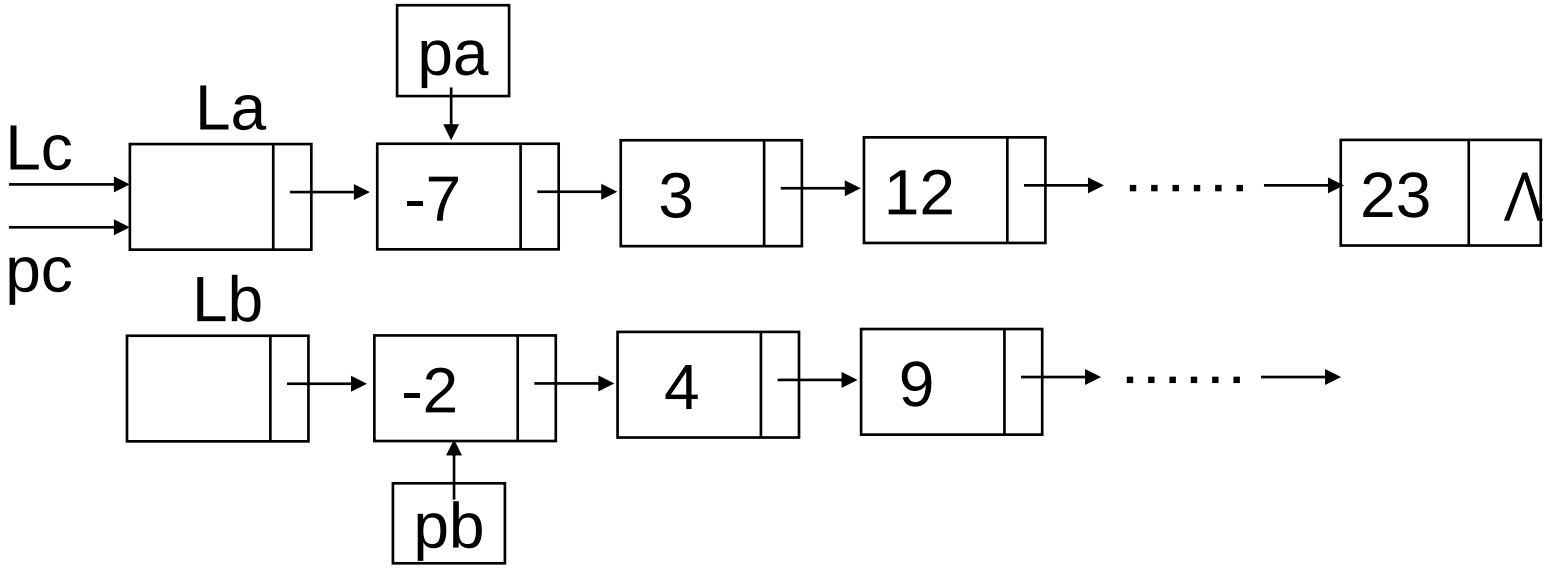
三. 单链表

- 单链表查找的算法时间复杂度主要取决于while循环中的语句频度
- 频度与被查找元素在单链表中的位置有关，若1≤i≤n，则频度为i-1，否则为n，因此时间复杂度为O(n)

2.3 链表

三. 单链表

- 单链表的合并
  - 把两个递增有序的单链表La和Lb合并成Lc，保持递增有序

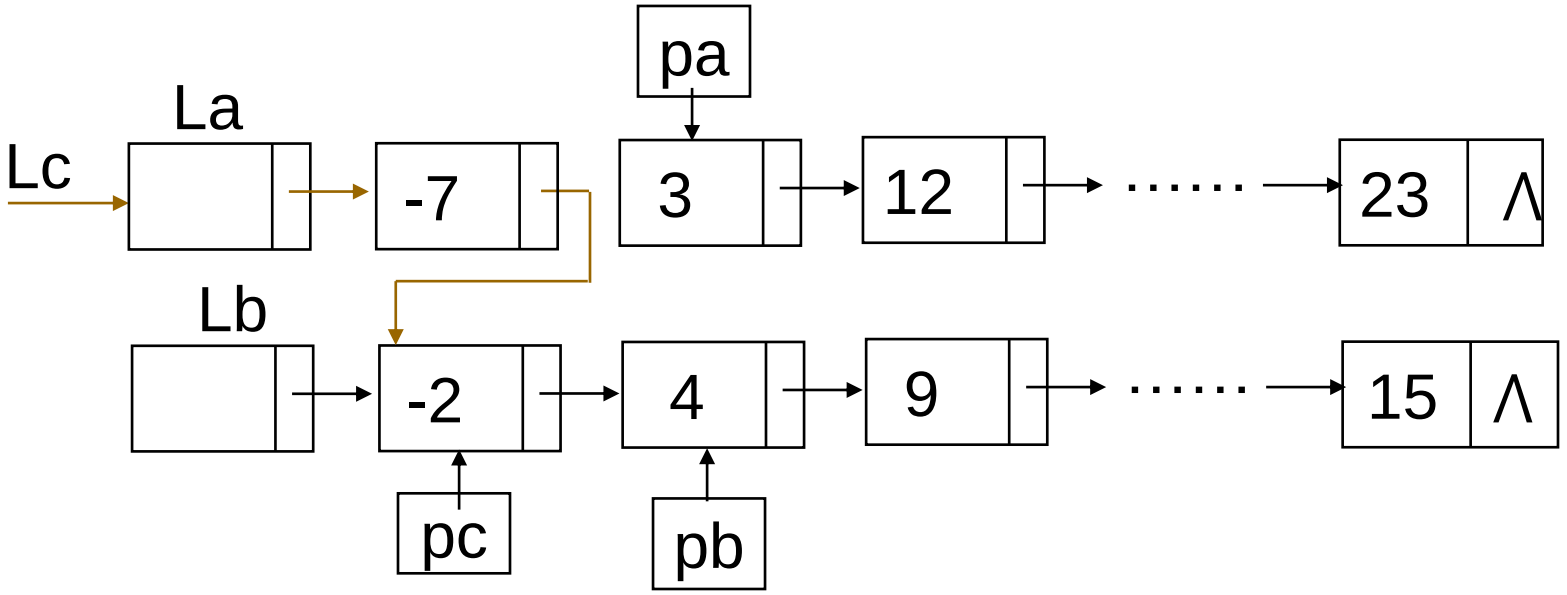


两个有序的单链表La，Lb的初始状态

2.3 链表

三. 单链表

- 单链表的合并
  - 把两个递增有序的单链表La和Lb合并成Lc，保持递增有序



合并了值为-7，-2的结点后的状态

2.3 链表

三. 单链表

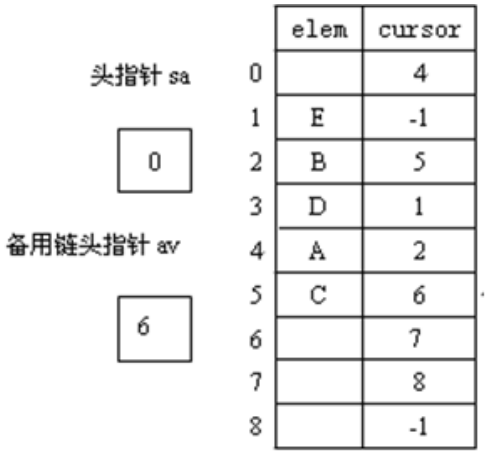
- 单链表合并代码

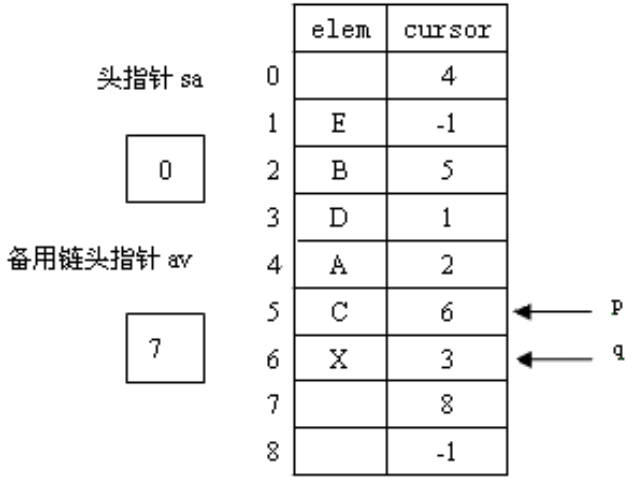
```
void MergeList_L(LinkList &La, LinkList &Lb, LinkList
&Lc) {
    LNode<ElemType> *pa = La.head.next;
    LNode<ElemType> *pb = Lb.head.next;
    LNode<ElemType> *pc = &(Lc.head);
    while (pa && pb) {
        if (pa->data <= pb->data) {
            pc->next = pa;   pc = pa;   pa = pa->next;
        }
        else {
            pc->next = pb;   pc = pb;   pb = pb->next;
        }
  }
  pc->next = pa ? pa : pb;  // 插入剩余段
  La.clear(); Lb.clear()  // 释放La和Lb
} // MergeList_L
```

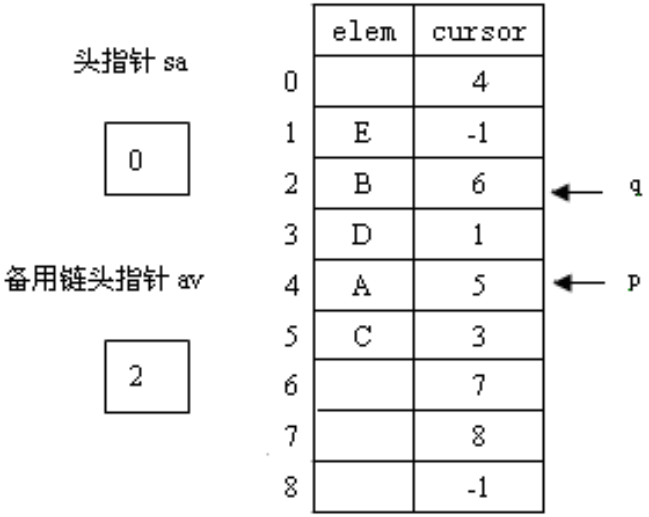
2.3 链表

三. 单链表

- 静态链表，用数组方式表示链表，为了与一般链表区分，称为静态链表
- 插入X和删除B，使用p\q指针修改cursor







Take Home Message

- 线性表： n个数据元素的有限序列

- 线性表顺序表示：用一组地址连续的存储单元依次存储线性表的数据元素
  - 随机存取
  - 增删时需要移动大量元素

- 线性表链式表示：存储单元并不连续，通过指针等手段来表示数据之间的邻接关系
  - 不可随机存取
  - 增删时只需要修改前后节点