顺序表的基本运算

1.顺序表的初始化即构造一个空表，将L设为指针参数，动态分配存储空间，将线性表的当前长度lengh设为0。算法如下：

```
void InitList_Sq(SqList &L) {  //构造一个空的顺序表
  L.elem= （ElemType*）
        malloc(LIST_INIT_SIZE*sizeof(ElemType);
        if(!L.elem) Error(“Overflow!”);   //存储分配失败
  L.length=0;                          //空表长度为
  L.listsize=LIST_INIT_SIZE;//初始存储容量
}// InitList_Sq
```

顺序表的基本运算

2. 顺序表的插入运算：

线性表的插入是指在表的第i个位置上插入一个值为 x 的新元素，插入后使原表长为 n的表，成为表长为 n+1 的表。

32

顺序表的基本运算

例如，在第5个位置上插入结点91的步骤如下：

0	1	2	3	4	5	6	7	8		
16	9	8	4	21	26	41	67			

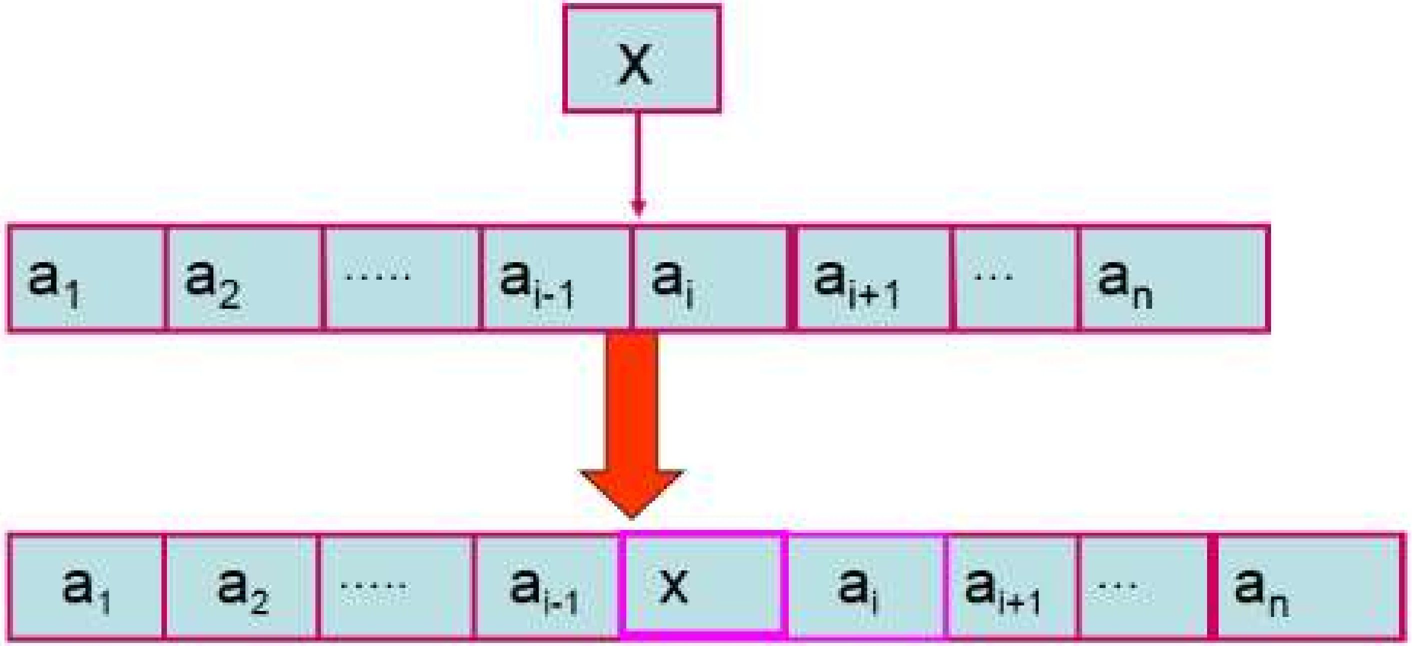
0	1	2	3	4	5	6	7	8		
16	9	8	4		21	26	41	67		

0	1	2	3	4	5	6	7	8		
16	9	8	4	91	21	26	41	67		

33

顺序表的基本运算

将插入情况推广到一般情况的步骤如下：



34

顺序表的基本运算

顺序表插入结点运算的步骤如下：

（1）将a_n～a_i之间的所有结点依次后移，为新元素让出第i个位置；

（2）将新结点x插入到第i个位置；

（3）修改 lengh，表长加1。

顺序表的基本运算

```
void ListInsert_Sq(SqList &L, int i, ElemType e) {
 //在顺序线性表L的第i个元素之前插入新的元素e,
 // i的合法值为1≤i≤Length+1。
   if (i < 1 || i > L.length+1)  Error("Position Error!");
   if (L.length >=LIST_SIZE)  Error("Overflow!");
    q = &(L.elem[i-1]);            //q为插入位置
   for (p = &(L.elem[L.length-1]); p>=q; --p)
    *(p+1) = *p;    //插入位置及之后的元素右移
   *q = e;       //插入元素e
   ++L.length;     //表长增1
}//ListInsert_Sq
```

36

顺序表的基本运算

要注意的问题是：

（1） 顺序表中数据区域有MAXLEN个存储单元，所以在插入时先检查顺序表是否已满，在表满的情况下不能再做插入，否则产生溢出错误。

（2）检验插入位置的有效性，这里 i 的有效范围是：$1<=i<=n+1$，其中 n 为原表长。

（3） 注意数据的移动方向，必须从原线性表最后一个结点(a_n)起往后移动。

37

顺序表的基本运算

3. 顺序表的删除运算：

线性表的删除运算是指将表中第 i 个元素从线性表中去掉，删除后使原表长为 n 的线性表：

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

变为表长为 n－1 的线性表：

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

i 的取值范围为：1<=i<=n 。

38

顺序表的基本运算

例如，删除第5个位置上的元素的步骤如下：

0	1	2	3	4	5	6	7	8		
16	9	8	4	21	26	41	67			

0	1	2	3	4	5	6	7	8		
16	9	8	4	26	41	67				

39

顺序表的基本运算

顺序表删除结点运算的步骤如下：

（1）将a_{i+1}～a_n之间的结点依次顺序向上移动。

（2）修改lengh，表长减1。

40

顺序表的基本运算

```
ListDelete_Sq(SqList &L, int i, ElemType &e) {
  //删除顺序线性表L的第i个元素，并用元素e返回其值
  // i的合法值为1≤i≤Length。
   if (i < 1 || i > L.length)  Error("Position Error!");
  e=L.elem[i-1];                   //取出被删除元素
  p=&(L.elem[i-1]);                //指向L中待删除元素的位置
  q = L.elem+L.lenghth-1;          //指向L中最后一个元素的位置
  for (++p;p<=q;++p)   *(p-1) = *p;
   --L.length;
}//ListDelete_Sq
```

41

顺序表的基本运算

顺序表的删除运算：

要注意的问题是：

（1）首先要检查删除位置的有效性，删除第i个元素，i的取值为：1<=i<=n。

（2）当表空时不能做删除，因表空时L.lengh的值为0，条件（i<1 || i>L.lengh）也包括了对表空的检查。

（3）删除 a_i 之后，该数据则已不存在，如果需要，必须先取出 a_i 后，再将其删除。

42

顺序表的时间复杂度分析

（1）插入运算的时间性能分析：

若插入在尾结点之后，则根本无需移动（特别快）；

若插入在首结点之前，则表中元素全部后移（特别慢）；

若要考虑在各种位置插入（共n+1种可能）的平均移动次数，该如何计算？

$$\text{AMN} = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{1}{n+1}(n+\cdots+1+0)$$

$$= \frac{1}{(n+1)}\frac{n(n+1)}{2} = \frac{n}{2}$$

其时间主要消耗在了移动表中元素上，（分析需要移动表中元素的规模），该算法的时间复杂度为O(n)。

43

顺序表的时间复杂度分析

（2）删除运算的时间性能分析：

若删除尾结点，则根本无需移动（特别快）；

若删除首结点，则表中n-1个元素全部前移（特别慢）；

若要考虑在各种位置删除（共n种可能）的平均移动次数，该如何计算？

$$AMN=\frac{1}{n}\sum_{i=1}^{n}(n-i)=\frac{1}{n}\frac{(n-1)n}{2}=\frac{n-1}{2}$$

与插入运算相同，其时间主要消耗在了移动表中元素上，（大约需要移动表中一半的元素），显然该算法的时间复杂度也为O(n)。

44