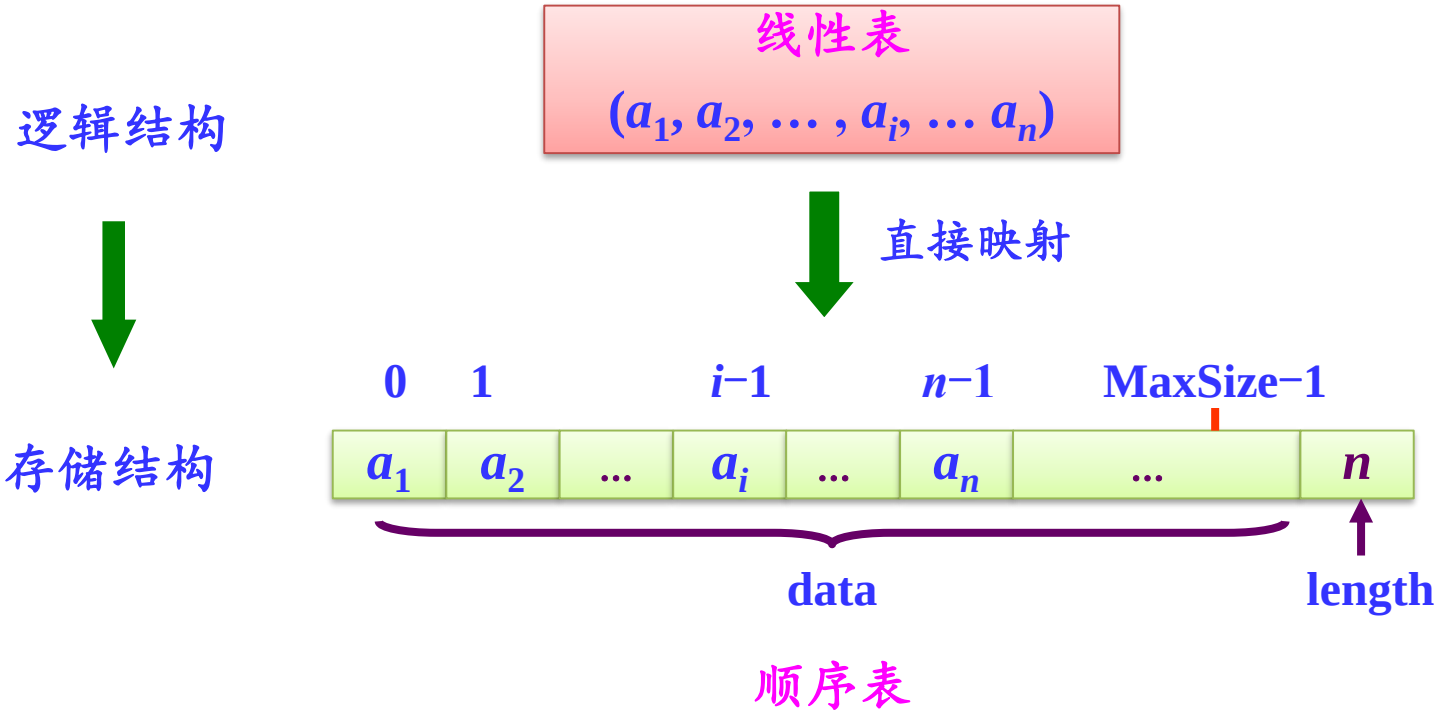
2.2 线性表的顺序存储结构

2.2.1 线性表的顺序存储—顺序表

线性表的顺序存储结构：把线性表中的所有元素按照顺序存储方法进行存储。

按逻辑顺序依次存储到存储器中一片连续的存储空间中。


线性表
$(a_1, a_2, \dots, a_i, \dots a_n)$
逻辑结构
直接映射
存储结构
0
1
$i-1$
$n-1$
MaxSize−1
a_1
a_2
...
a_i
...
a_n
...
n
data
length
顺序表

顺序表类型定义：

这里，假设ElemType为char类型

```
typedef struct
{    ElemType data[MaxSize];
     int length;
} SqList;        //顺序表类型
```

其中data成员存放元素，length成员存放线性表的实际长度。

说明：注意逻辑位序和物理位序相差1。

2.2.2 顺序表运算的实现

1、建立顺序表

$a[0..n-1]$ ⇨ 顺序表L — 整体创建顺序表。

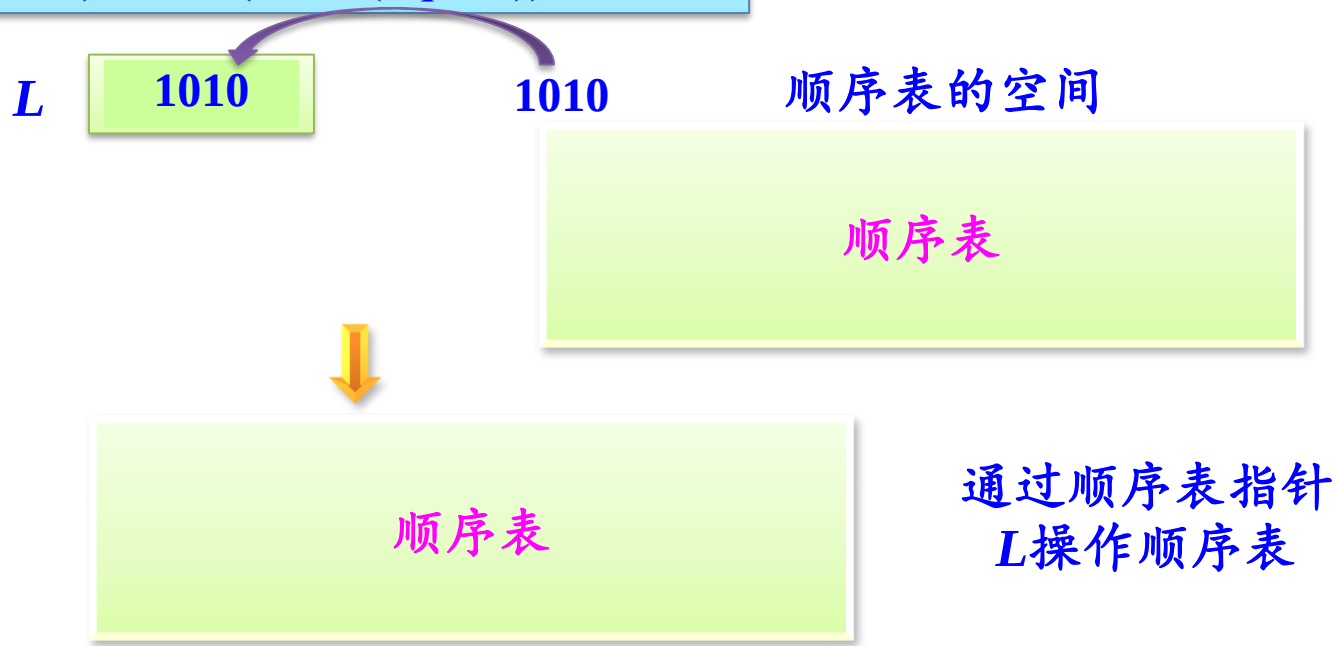
```
void CreateList(SqList *&L,ElemType a[],int n)
      //整体建立顺序表
{    int i;
     L=(SqList *)malloc(sizeof(SqList));
     for (i=0;i<n;i++)
           L->data[i]=a[i];
     L->length=n;
}
```

传递顺序表指针

算法参数说明

❶ 顺序表指针的含义

```
SqList  *L;
L=(SqList *)malloc(sizeof(SqList));
```



❷ 顺序表指针引用

```
void CreateList(SqList *&L,ElemType a[],int n)
```

引用参数：将执行结果回传给实参

- 引用符号“&”放在形参*L*的前面。
- 输出型参数均为使用“&”，不论参数值是否改变。

2、顺序表基本运算算法

（1）初始化线性表InitList(L)

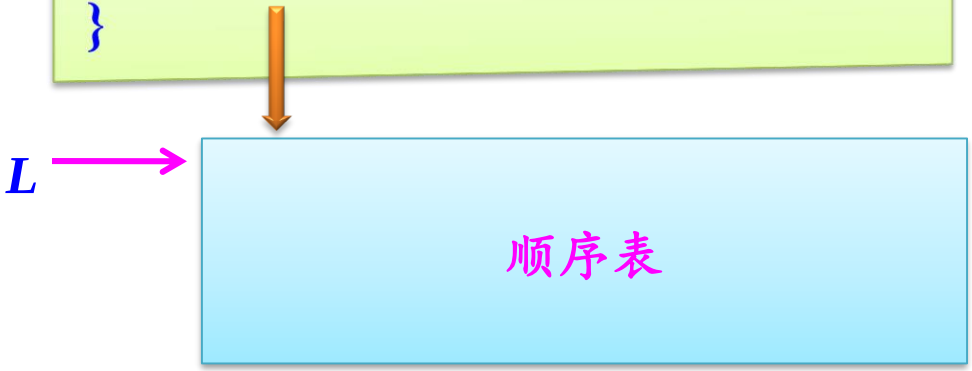
该运算的结果是构造一个空的线性表L。实际上只需将length成员设置为0即可。

```
void InitList(SqList *&L)
{     L=(SqList *)malloc(sizeof(SqList));
                  //分配存放线性表的顺序表空间
      L->length=0;
}
```

（2）销毁线性表DestroyList(L)

该运算的结果是释放线性表L占用的内存空间。

```
void DestroyList(SqList *&L)
{
    free(L);
}
```



free(*L*)释放*L*所指向的空间

（3）判定是否为空表ListEmpty(L)

该运算返回一个值表示L是否为空表。若L为空表，则返回true，否则返回false。

```
bool  ListEmpty(SqList *L)
{
    return(L->length==0);
}
```

（4）求线性表的长度ListLength(L)

该运算返回顺序表L的长度。实际上只需返回length成员的值即可。

```
int ListLength(SqList *L)
{
    return(L->length);
}
```

（5）输出线性表DispList(L)

该运算当线性表L不为空时,顺序显示L中各元素的值。

```
void DispList(SqList *L)
{   int i;
    if (ListEmpty(L)) return;
    for (i=0;i<L->length;i++)
        printf("%c",L->data[i]);
    printf("\n");
}
```

（6）求某个数据元素值GetElem(L,i,e)

该运算返回L中第 *i*（1≤*i*≤ListLength(L)）个元素的值，存放在*e*中。

```
bool GetElem(SqList *L,int i,ElemType &e)
{    if (i<1 || i>L->length)  return false;
     e=L->data[i-1];
     return true;
}
```

本算法的时间复杂度为O(1)。

体现顺序表的随机存取特性

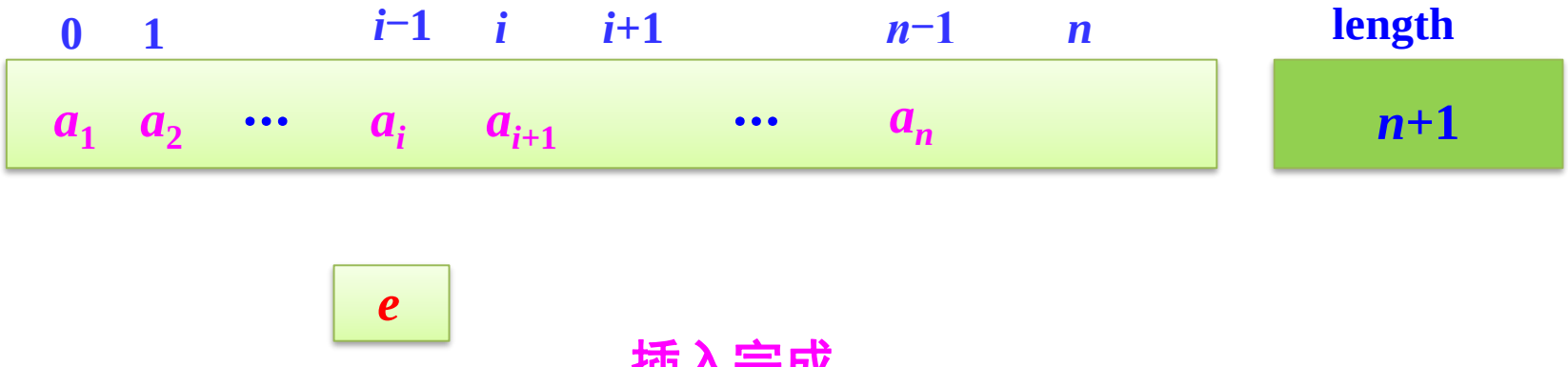
（7）按元素值查找LocateElem(L,e)

该运算顺序查找第1个值域与*e*相等的元素的逻辑位序。若这样的元素不存在，则返回值为0。

```
int LocateElem(SqList *L, ElemType e)
{    int i=0;
     while (i<L->length && L->data[i]!=e)
          i++;
     if (i>=L->length)  return 0;
     else  return i+1;
}
```

（8）插入数据元素ListInsert(L,i,e)

该运算在顺序表L的第i（$1 \leqslant i \leqslant$ListLength(L)+1）个位置上插入新的元素e。



插入算法如下：

```
bool  ListInsert(SqList *&L,int i,ElemType e)
{    int j;
     if (i<1 || i>L->length+1)
          return false;                    //参数错误时返回false
    i--;                                  //将顺序表逻辑序号转化为物理序号
    for (j=L->length;j>i;j--)             //将data[i..n-1]元素后移一个位置
         L->data[j]=L->data[j-1];
  L->data[i]=e;                           //插入元素e
  L->length++;                            //顺序表长度增1
  return true;                            //成功插入返回true
}
```

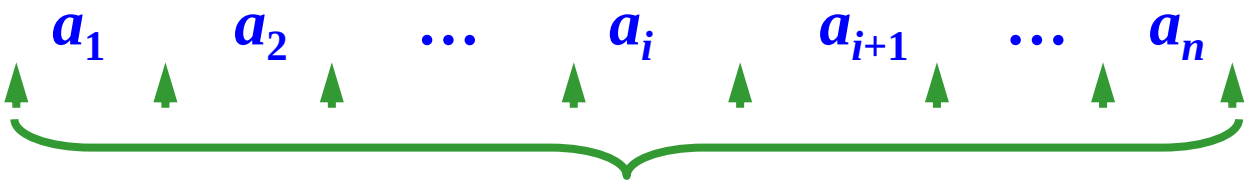
对于本算法来说，元素移动的次数不仅与表长L->length=n有关，而且与插入位置i有关：

- 当$i=n+1$时，移动次数为0；
- 当$i=1$时，移动次数为n，达到最大值。

算法最好时间复杂度为O(1)

算法最坏时间复杂度为O(n)

平均情况分析：



在线性表L中共有$n+1$个可以插入元素的地方

在插入元素a_i时，若为等概率情况，则$p_i=\frac{1}{n+1}$

此时需要将a_i～a_n的元素均后移一个位置，共移动$n-i+1$个元素。

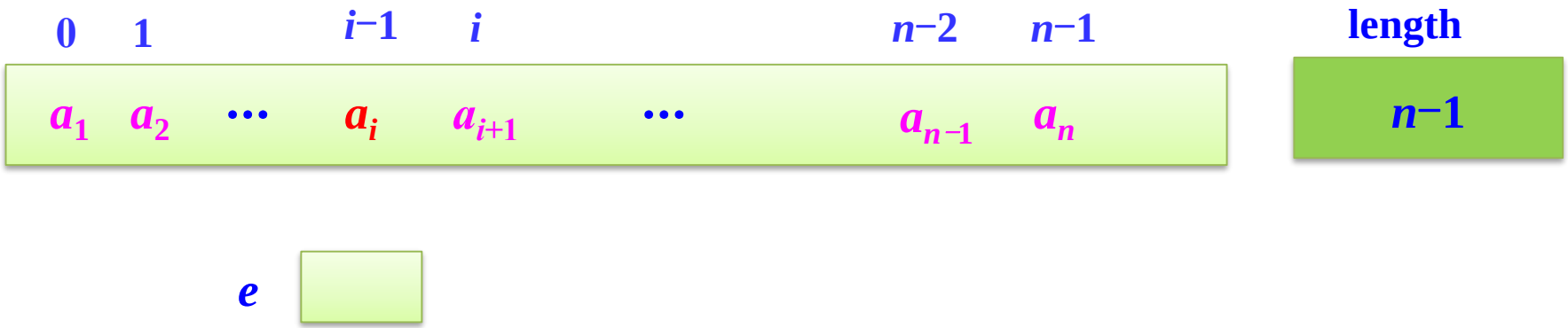
所以在长度为n的线性表中插入一个元素时所需移动元素的平均次数为：

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

因此插入算法的平均时间复杂度为O(n)。

（9）删除数据元素ListDelete(L,i,e)

该运算删除顺序表L的第i（$1 \leqslant i \leqslant$ListLength(L)）个元素。



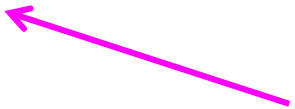
删除完成

删除算法如下：

```
bool ListDelete(SqList *&L,int i,ElemType &e)
{     int j;
      if (i<1 || i>L->length)            //参数错误时返回false
            return false;
     i--;                                //将顺序表逻辑序号转化为物理序号
     e=L->data[i];
    for (j=i;j<L->length-1;j++)    //将data[i..n-1]元素前移
          L->data[j]=L->data[j+1];
   L->length--;                          //顺序表长度减1
   return true;                          //成功删除返回true
}
```

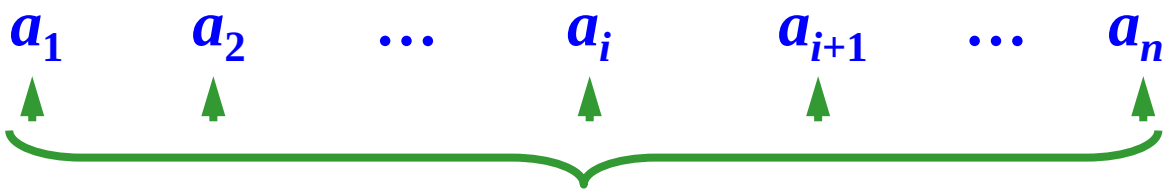
对于本算法来说，元素移动的次数也与表长n和删除元素的位置i有关：

- 当$i=n$时，移动次数为0；
- 当$i=1$时，移动次数为$n-1$。



删除算法最好时间复杂度为O(1)　删除算法最坏时间复杂度为O(n)

平均情况分析：



在线性表L中共有n个可以删除元素的地方

在删除元素a_i时，若为等概率情况，则$p_i=\frac{1}{n}$

此时需要将a_{i+1}～a_n的元素均前移一个位置，共移动$n-(i+1)+1=n-i$个元素。

所以在长度为n的线性表中删除一个元素时所需移动元素的平均次数为：

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

因此删除算法的平均时间复杂度为$O(n)$。

思考题

❶ 假如有一个学生表，每个学生包含学号、姓名和分数。你如何设计相应的学生顺序表？

❷ 如果需要对该学生表进行插入、修改和删除运算，你如何实现相关算法？

——本讲完——