

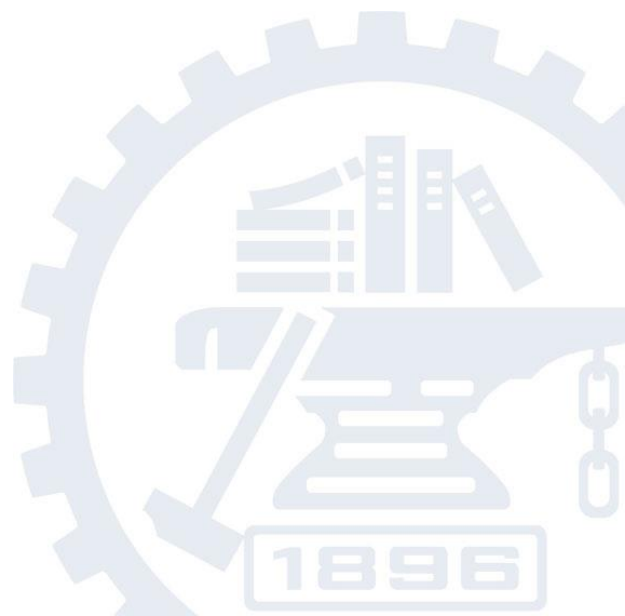


北京交通大学
BEIJING JIAOTONG UNIVERSITY



高级程序设计训练

SPT-02 线性表





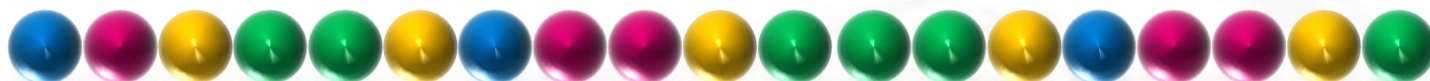
北京交通大学
BEIJING JIAOTONG UNIVERSITY





问题简化

- 假设：小球队列是静止的



- 要求：设计一个算法实现计算机自己玩祖玛游戏
 - 1. 计算机初始化一个长度为 N 的小球队列，并逐一给所有小球随机分配颜色
 - 2. 计算机生成一个随机颜色（设为 Y ）的小球 A
 - 3. 计算机查找小球队列中 Y 色小球连续两个以上排列的位置，返回连续排列中第一个小球的位置；否则，返回第一个颜色为 Y 的小球的位置；
 - 4. 将球 A 插入到该位置
 - 5. 删除该位置起始的所有颜色为 Y 的小球



操作对象

- 小球



操作对象的静态特征

- 各自不同的颜色
- 各自不同的位置
- 唯一的“头球”
- 唯一的“尾球”
- 固定的前驱
- 固定的后继
- 个数有限

线性表



结构特征
线性结构



2.1 线性表的类型定义

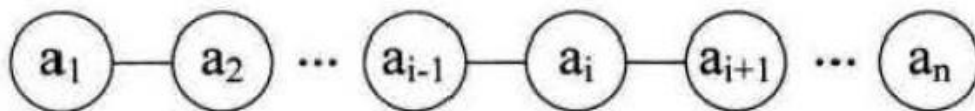


线性表的特点

- 线性表是一个有限序列，是线性结构的一种。
 - 存在唯一的一个被称做“**第一个**”的数据元素；
 - 存在唯一的一个被称做“**最后一个**”的数据元素；
 - 除第一个之外，集合的每个数据元素均只有一个“**前驱**”；
 - 除最后一个之外，集合每个数据元素均只有一个“**后继**”；
- 线性结构包括：线性表，栈，队列，串，数组

如果用数学语言来进行定义。可如下：

若将线性表记为 $(a_1, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n)$ ，则表中 a_{i-1} 领先于 a_i ， a_i 领先于 a_{i+1} ，称 a_{i-1} 是 a_i 的直接前驱元素， a_{i+1} 是 a_i 的直接后继元素。当 $i=1, 2, \dots, n-1$ 时， a_i 有且仅有一个直接后继，当 $i=2, 3, \dots, n$ 时， a_i 有且仅有一个直接前驱。如图 3-2-1 所示。





操作对象

- 小球



操作对象的静态特征

- 各自不同的颜色
- 各自不同的位置
- 小球线性排列

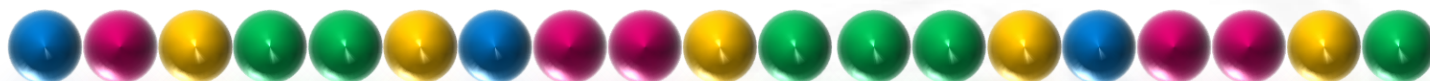


问题使用的操作？



问题简化

- 假设：小球队列是静止的



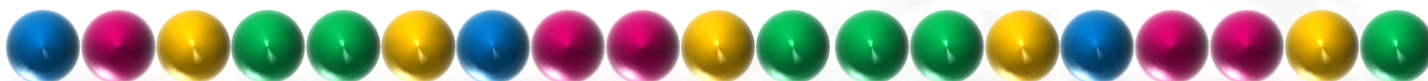
- 要求：设计一个算法实现计算机自己玩祖玛游戏
 - 1. 计算机初始化一个长度为 N 的小球队列，并逐一给所有小球随机分配颜色
 - 2. 计算机生成一个随机颜色（设为 Y ）的小球 A
 - 3. 计算机查找小球队列中 Y 色小球连续两个以上排列的位置，返回连续排列中第一个小球的位置；否则，返回第一个颜色为 Y 的小球的位置；
 - 4. 将球 A 插入到该位置
 - 5. 删除该位置起始的所有颜色为 Y 的小球



问题简化

极端情况.

- 假设：小球队列是静止的



- 要求：设计一个算法实现计算机自己玩祖玛游戏
 - 1. 计算机**初始化**一个长度为N的小球队列，并**逐一给所有**小球随机分配颜色
 - 2. 计算机生成一个随机颜色（设为Y）的小球A
 - 3. 计算机**查找**小球队列中Y色小球**连续两个**以上排列的位置，返回连续排列中第一个小球的位置；否则，返回第一个颜色为Y的小球的位置；
 - 4. 将球A**插入**到该位置
 - 5. **删除**该位置起始的所有颜色为Y的小球



问题使用的基本操作

1. 计算机**初始化**一个长度为N的小球队列，并**逐一给所有**小球随机分配颜色
2. 计算机生成一个随机颜色（设为Y）的小球A
3. 计算机**查找**小球队列中Y色小球**连续两个**以上排列的位置，返回连续排列中第一个小球的位置；否则，返回第一个颜色为Y的小球的位置；
4. 将球A**插入**到该位置
5. **删除**该位置起始的所有颜色为Y的小球

④ 初始化 `initList(*L);`

④ 遍历

`ListTraverse(L, * visit());`

④ 查找 `LocateElem(L,e);`

④ 求后继

`NextElem(L,cur_e, *next_e);`

④ 插入

`ListInsert(*L, i, e);`

④ 删除 `ListDelete(*L,i,*e);`

④ 销毁 `DestroyList(*L);`



线性表的物理（存储）结构

- ① 存储结构有两种：② 顺序存储结构和链式存储结构。
- 顺序存储结构的线性表称为顺序表。
- 链式存储结构的线性表称为链表。
- 不同的存储结构会对应不同的结构体定义和算法。





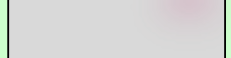



2.2 线性表的顺序表示和实现

用一组^①地址连续的存储单元^②依次存储线性表的数据元素。

$$LOC(a_{i+1}) = LOC(a_i) + \lambda$$

$$LOC(a_i) = LOC(a_1) + (i - 1) * \lambda$$

存储地址	内存状态	位序
b	a_1 	1
$b + \lambda$	a_2 	2
\vdots	\vdots	\vdots
$b + (i - 1)\lambda$	a_i 	i
\vdots	\vdots	\vdots
$b + (n - 1)\lambda$	a_n 	n
\vdots		空闲
		



2.2 线性表的顺序表示和实现

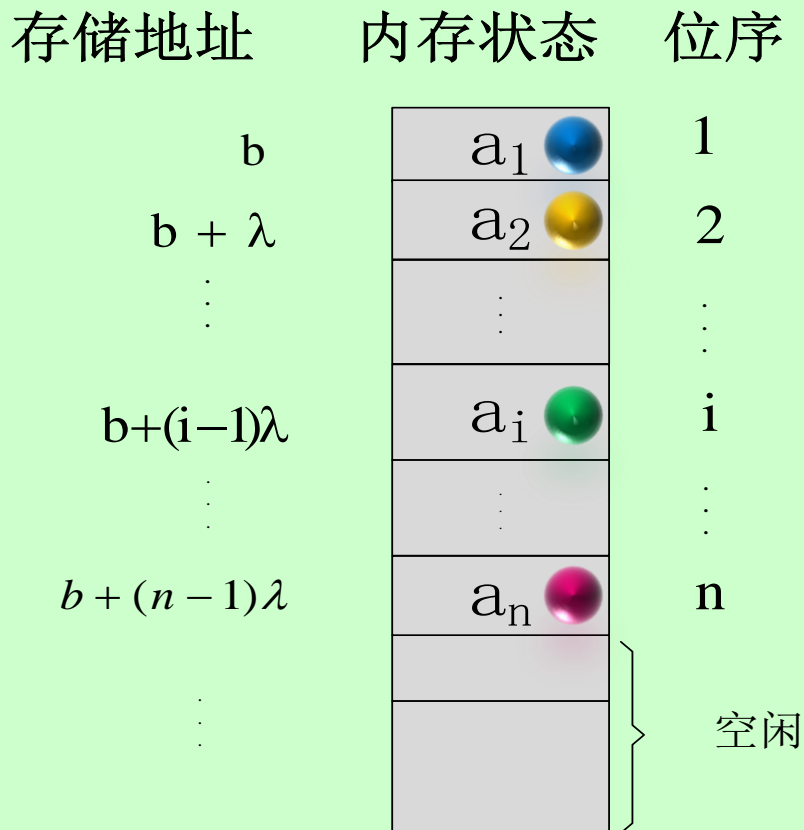


特点

- 逻辑上相邻的元素在物理位置上也相邻
- 能对任何元素进行随机存取
- 存储位置可用简单、直观的公式来表示

$$LOC(a_{i+1}) = LOC(a_i) + \lambda$$

$$LOC(a_i) = LOC(a_1) + (i - 1) * \lambda$$





- 2.1 线性表的概念和特点
- 2.2 顺序表的结构体定义及基本操作算法
- 2.3 链表的结构体定义及基本操作算法



顺序表的存储结构定义

```
#define LIST_SIZE 100 //线性表存储空间的分配量
```

```
ElemType SqList[LIST_SIZE]; //存放线性表元素的一维数组
```

ElemType
↓
Element

↓
SqList
↓
Sequence List



问题使用的基本操作

1. 计算机**初始化**一个长度为N的小球队列，并**逐一给所有**小球随机分配颜色
2. 计算机生成一个随机颜色（设为Y）的小球A
3. 计算机**查找**小球队列中Y色小球**连续两个**以上排列的位置，返回连续排列中第一个小球的位置；否则，返回第一个颜色为Y的小球的位置；
4. 将球A**插入**到该位置
5. **删除**该位置起始的所有颜色为Y的小球

④ 初始化 `initList(*L);`

④ 遍历

`ListTraverse(L, * visit());`

④ 查找 `LocateElem(L,e);`

④ 求后继

`NextElem(L,cur_e, *next_e);`

④ 插入

`ListInsert(*L, i, e);`

④ 删除 `ListDelete(*L,i,*e);`

④ 销毁 `DestroyList(*L);`



问题使用的基本操作

④ 初始化 `initList(*L);`

④ 遍历

`ListTraverse(L, * visit());`

需要已知线性表中的元素个数

④ 查找 `LocateElem(L,e);`

需要已知线性表中的元素个数

④ 求后继

`NextElem(L,cur_e, *next_e);`

需要已知线性表中的元素个数

④ 插入

`ListInsert(*L, i, e);`

需要已知线性表中的元素个数

④ 删除 `ListDelete(*L,i,*e);`

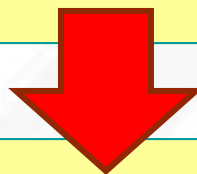
需要已知线性表中的元素个数

④ 销毁 `DestroyList(*L);`



顺序表的存储结构定义

```
#define LIST_SIZE 100 //线性表存储空间的分配量  
ElemType SqList[LIST_SIZE]; //存放线性表元素的一维数组  
int length; //线性表长度
```



```
#define LIST_SIZE 100 //线性表存储空间的分配量  
typedef struct{  
    ElemType elem[LIST_SIZE]; //存放线性表元素的一维数组  
    int length; //线性表长度  
}SqList;
```



- 如果采用数组形式的结构体定义，初始化顺序表函数应该如何写？

```
#define MAXSIZE 20 /* 存储空间初始分配量 */
```

```
typedef struct
```



```
{
```

```
    ElemType data[MAXSIZE];
```

```
/* 数组，存储数据元素 */
```

```
    int length;
```

```
/* 线性表当前长度 */
```

```
}SqList;
```



头文件

```
#include "stdio.h"
```

全局变量

```
typedef int ElemType;          /* ElemType类型根据实际情况而定
```

```
#define MAXSIZE 20 /* 存储空间初始分配量 */
```

```
typedef struct
```

```
{  
    ElemType data[MAXSIZE];      /* 数组，存储数据元素 */  
    int length;                  /* 线性表当前长度 */  
}SqList;
```

主函数

函数定义



头文件

```
#include "stdio.h"
```

全局变量

```
typedef int ElemType;          /* ElemType类型根据实际情况而定
```

```
#define MAXSIZE 20 /* 存储空间初始分配量 */
```

```
typedef struct
```

```
{  
    ElemType data[MAXSIZE];      /* 数组，存储数据元素 */  
    int length;                  /* 线性表当前长度 */  
}SqList;
```

函数定义

主函数



函数定义

```
[返回值类型] initList (参数)
{
    初始化结构体变量SqList;
}
```

主函数

```
int main()
{
    //检查结构体初始化是否成功;
    调用函数initList(实参);
    方法一: 检查函数返回值是否表示成功;
    方法二: 检查结构体成员是否被初始化;
}
```



(2) 找元素位置

```
int LocateElem_Sq ( ) {
```

```
    i=0;
```

```
    /*防御语句*/
```

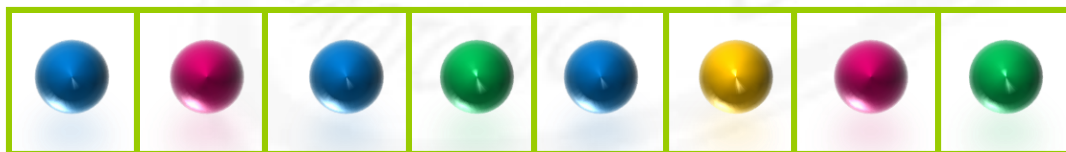
```
    /*查找语句*/
```

```
    if(i<=L.length) return i;
```

```
    else return 0;
```

```
} //LocateElem_Sq;
```

i=6

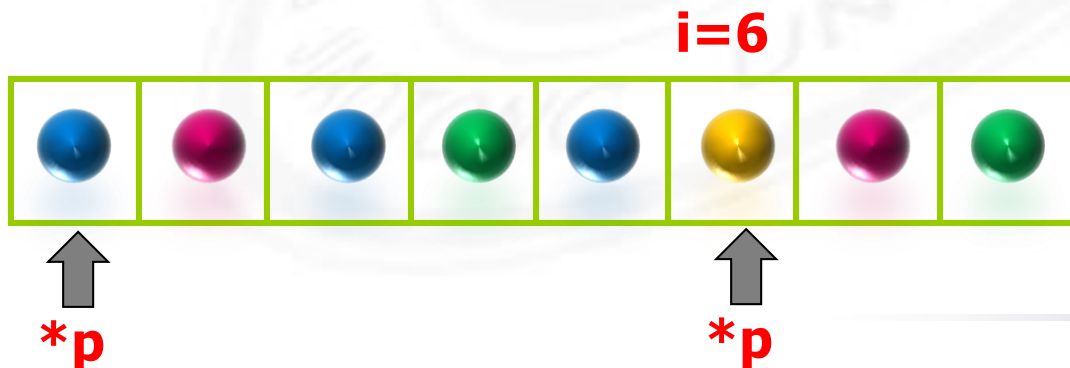


$$T(n)=O(n)$$



(2) 找元素位置

```
int LocateElem_Sq (SqList L, ElemType e) {  
    i=1;  
    p=L.elem;  
    while((i<L.length) && (*p != e)) {i++; p++;}  
    if(i<=L.length) return i;  
    else return 0;  
} //LocateElem_Sq;
```





(3) 找元素的后继

```
ElemType NextElem_Sq (SqList L, ElemType e){
```

```
    int i=0;
```

```
    i=LocateElem_Sq(L, e);
```

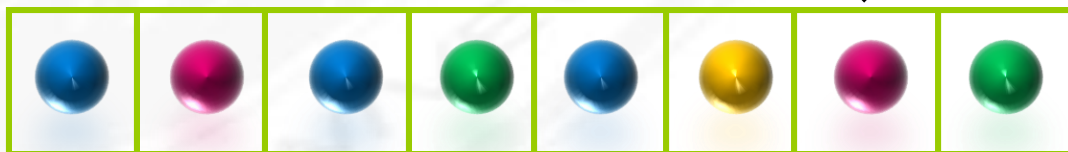
```
    if (i<L.length)
```

```
        return L.elem[i];
```

```
    else
```

```
        return ERROR;
```

```
} // NextElem_Sq
```



$L.elem[i-1]$

$T(n)=O(n)$



(3) 找元素的后继

```
Status NextElem_Sq (SqList L, ElemType e, *Next_e){
```

```
    int i=0;
```

```
    i=LocateElem_Sq(L, e);
```

```
    if (i<L.length)
```

```
    { *Next_e=L.elem[i];
```

```
        return OK;}
```

```
    else
```

```
        return ERROR;
```

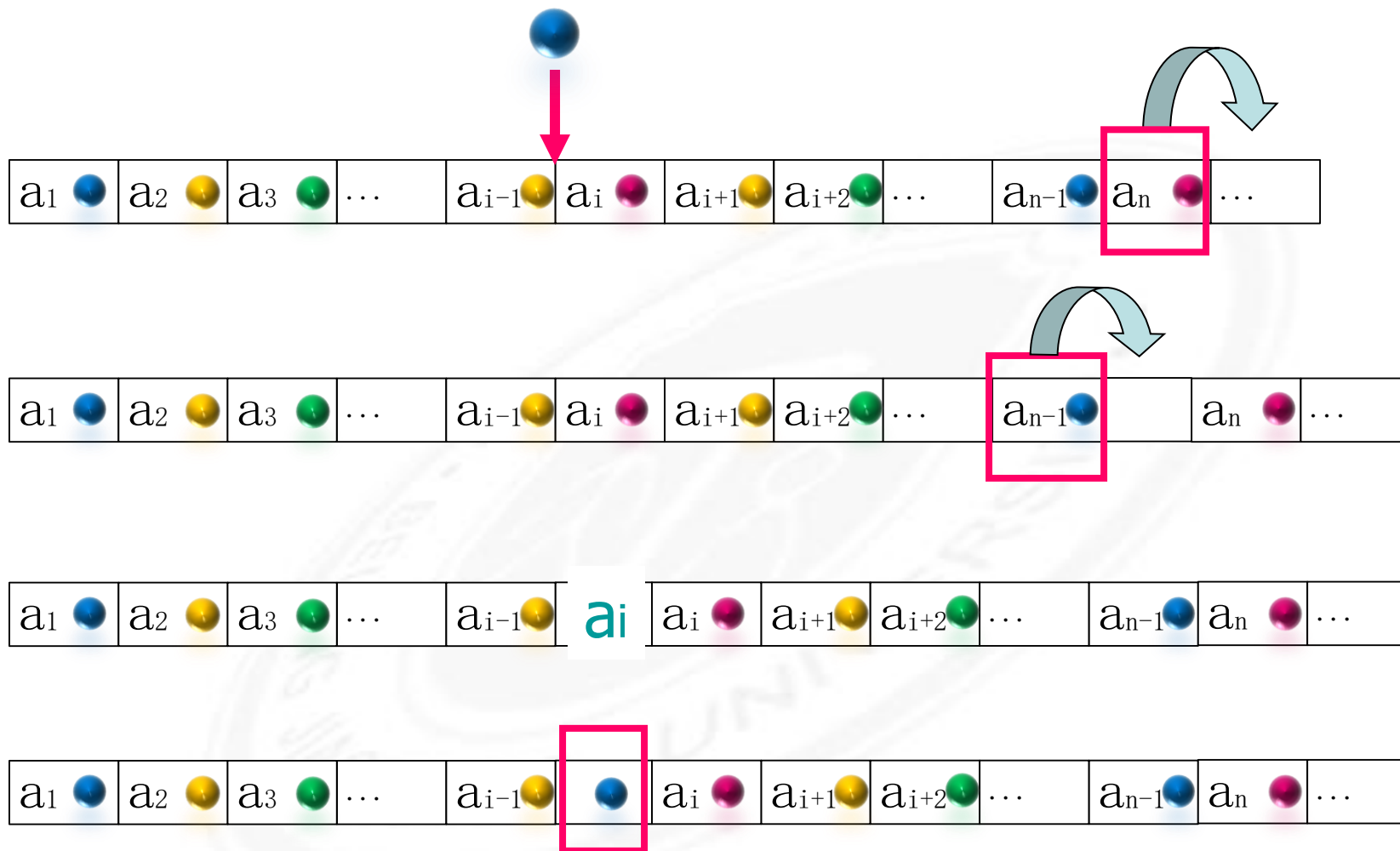
```
} // NextElem_Sq
```



$T(n)=O(n)$



(4) 顺序表的插入





(5) 顺序表的插入

Status ListInsert_Sq() {

if($i < 1 \parallel i > L.length + 1$) return ERROR; //i值不合法

if($L.length == MAXSIZE$) return ERROR; //当前存储空间已
满不允许插入

```
if (i <= L->length)
{
    for(k= ; k >= ; k--)
        L->data[k+1] = L->data[k];
    L->data[ ] = e;
    L->length++;
    return OK;
}
```



(5) 顺序表的插入

```
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后移

```
*q=e;
```

//插入蓝色球

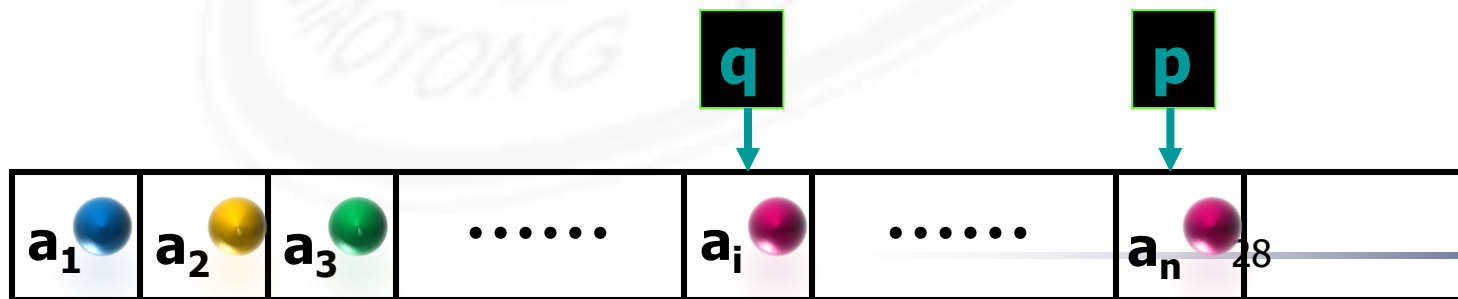
```
++L.length;
```

//表长增1

```
return OK;
```

注意!

```
} //ListInsert_Sq
```





插入操作的时间复杂性分析

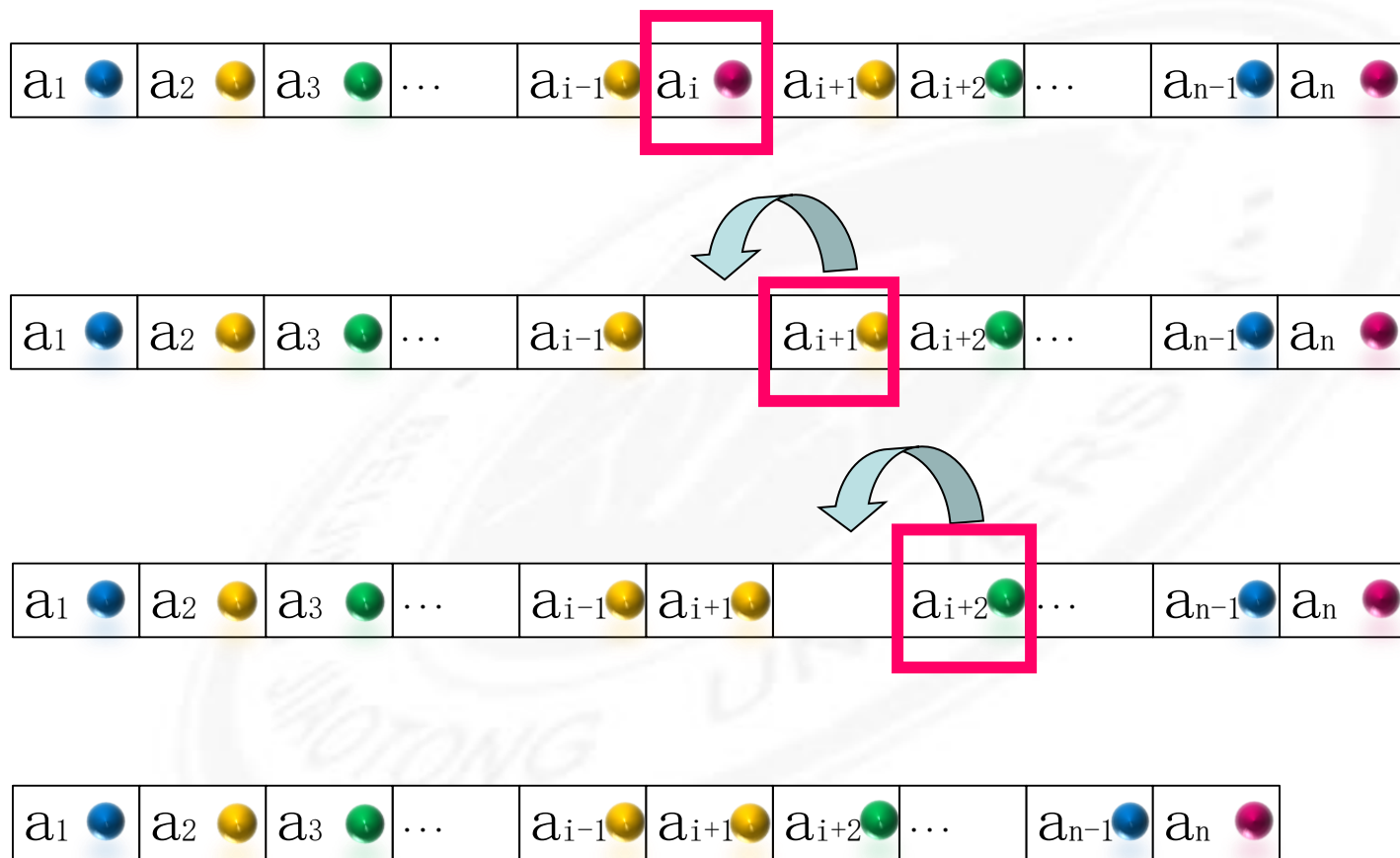
$$E_{is} = \sum_{i=1}^{n+1} p_i (n - i + 1) \quad p_i = \frac{1}{n+1}$$

$$E_{is} = \frac{1}{n+1} \sum_{i=1}^{n+1} (n - i + 1) = \frac{n}{2} \quad T(n) = O(n)$$

在顺序表中插入元素，平均约移动一半元素



(6) 顺序表的删除





(6) 顺序表的删除

Status ListDelete_Sq(SqList &L, int i, ElemType &e){

if(i<1 || (i>L.length)) return ERROR; //i值不合法

p=&(L.elem[i-1]); //p为被删除元素的位置

e=*p; //被删除元素的值赋给e

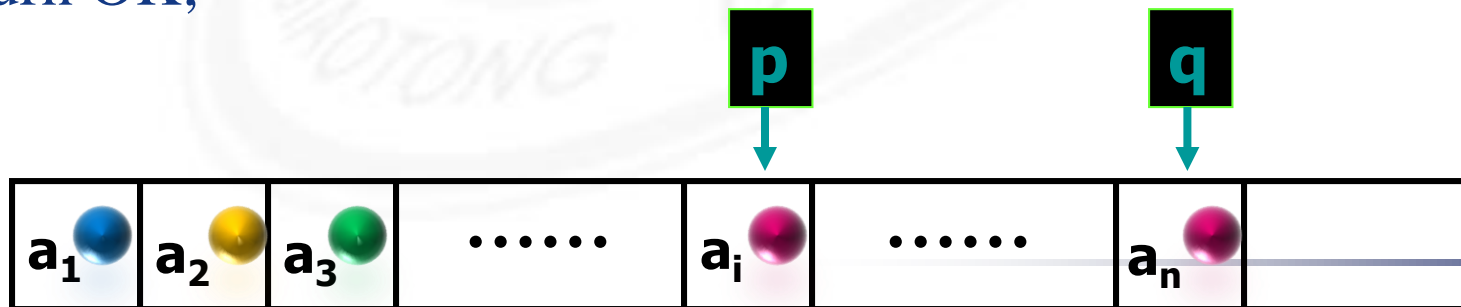
q=L.elem+L.length-1; //表尾元素的位置

for(++p; p<=q; ++p) *(p-1)=*p; //被删除之后的元素前移

--L.length; //表长减1

return OK;

}





(5) 顺序表的删除

```
00109: Status ListDelete(SqList *L,int i,ElemType *e)
00110: {
00111:     int k;
00112:     if (L->length==0)
00113:         return ERROR;
00114:     if (i<1 || i>L->length)
00115:         return ERROR;
00116:     *e=L->data[ ];
00117:     if (i<L->length)
00118:     {
00119:         for(k= ;k< ;k++)
00120:             L->data[k-1]=L->data[k];
00121:     }
00122:     L->length--;
00123:     return OK;
00124: }
```




删除操作的时间复杂性分析

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

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

$$T(n) = O(n)$$

在顺序表中删除元素，平均约移动一半元素



(7) 顺序表的遍历

```
Status ListTraverse (SqList L) {
```

```
    for(i=1;i_____;i++)
```

```
        visit(_____);
```

```
    printf("\n");
```

```
    return OK;
```

```
}// ListDelete_Sq
```

```
Status visit(ElemType c)
```

```
{
```

```
    printf("%3d ", c);
```

```
    return OK;
```

```
}
```

$T(n)=O(n)$



(7) 顺序表的遍历

```
Status ListTraverse (SqList L, Status (*visit)(ElemType)) {  
    for(i=1;i<=L.length;i++)  
        if(!visit(L.elem[i-1])) return ERROR;  
} // ListDelete_Sq
```

```
Status visit(ElemType e){  
    printf("The value of the element is %d\n", (int) e);  
}
```

$T(n)=O(n)$



《实验一：顺序表的实现及基本操作》

- ④ 创建有若干个元素（可以是整型数值）的顺序表，实现对顺序表的初始化，对已建立的顺序表插入操作、删除操作、遍历输出顺序表。
- ④ 要求各个操作均以函数的形式实现，在主函数中调用各个函数实现以下操作：
 - ① 创建顺序表21、18、30、75、42、56，并输出顺序表中的各元素值。
 - ② 在顺序表的第3个位置插入67，并输出此时顺序表中的各元素值。
 - ③ 删除顺序表中的第6个数据元素，并输出此时顺序表中的各元素值。
 - ④ 查找顺序表中是否有75这个元素，如果有返回该元素在顺序表中的位序。