线性表的重要基本操作

1. 初始化
2. 取值
3. 查找
4. 插入
5. 删除

重要基本操作的算法实现

1. 初始化线性表L （参数用引用）

```
Status InitList_Sq(SqList &L){    //构造一个空的顺序表L
    L.elem=new ElemType[MAXSIZE];  //为顺序表分配空间
    if(!L.elem) exit(OVERFLOW);       //存储分配失败
    L.length=0;                                    //空表长度为0
    return OK;
}
```

1. 初始化线性表L （参数用指针）

```
Status InitList_Sq(SqList *L){    //构造一个空的顺序表L
    L-> elem=new ElemType[MAXSIZE];   //为顺序表分配空间
    if(! L-> elem) exit(OVERFLOW);       //存储分配失败
    L-> length=0;                                  //空表长度为0
    return OK;
}
```

补充：几个简单基本操作的算法实现

销毁线性表L

```
void DestroyList(SqList &L)
{
  if (L.elem) delete[]L.elem;    //释放存储空间
}
```

清空线性表L

```
void ClearList(SqList &L)
{
   L.length=0;                  //将线性表的长度置为0
}
```

补充：几个简单基本操作的算法实现

求线性表L的长度

```
int GetLength(SqList  L)
{
   return (L.length);
}
```

判断线性表L是否为空

```
int IsEmpty(SqList L)
{
  if (L.length==0) return 1;
   else return 0;
}
```

线性表的重要基本操作

1. 初始化
2. 取值
3. 查找
4. 插入
5. 删除

2. 取值（根据位置i获取相应位置数据元素的内容）

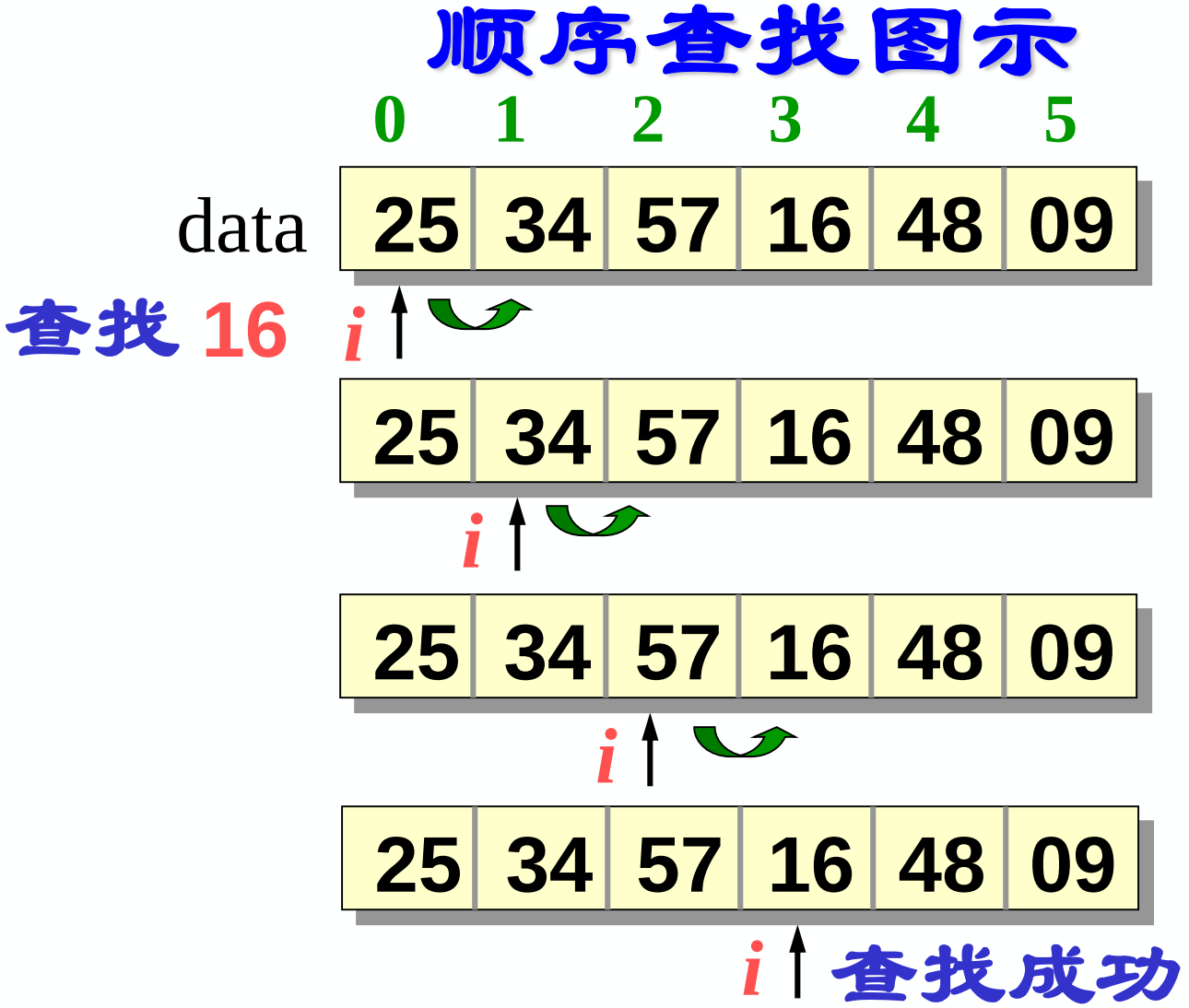
获取线性表L中的某个数据元素的内容

```
int GetElem(SqList L,int i,ElemType &e)
{
  if (i<1||i>L.length) return ERROR;
   //判断i值是否合理，若不合理，返回ERROR
  e=L.elem[i-1];   //第i-1的单元存储着第i个数据
  return OK;
}
```

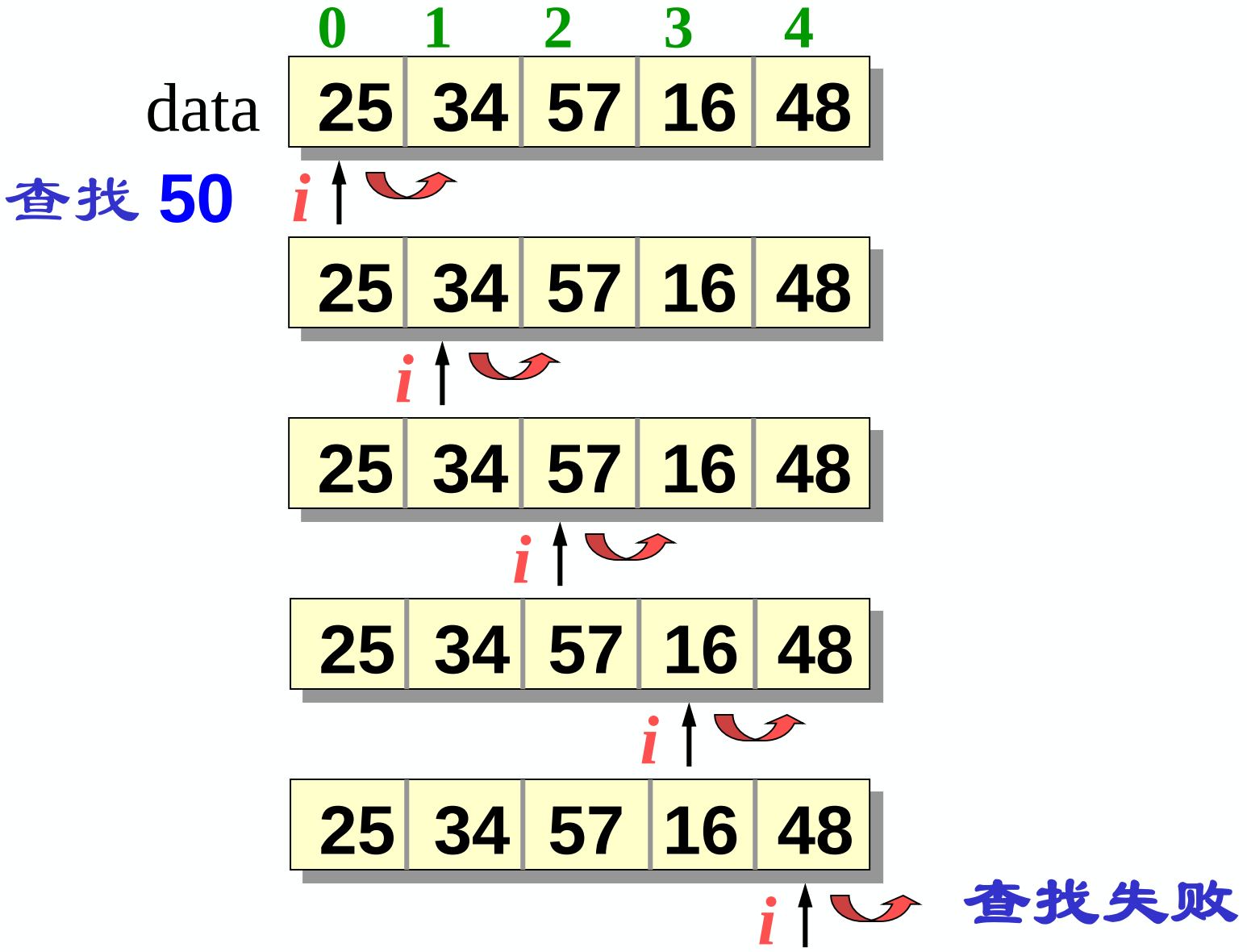
随机存取

Click Here

3.查找（根据指定数据获取数据所在的位置）



3. 查找（根据指定数据获取数据所在的位置）



3. 查找（根据指定数据获取数据所在的位置）

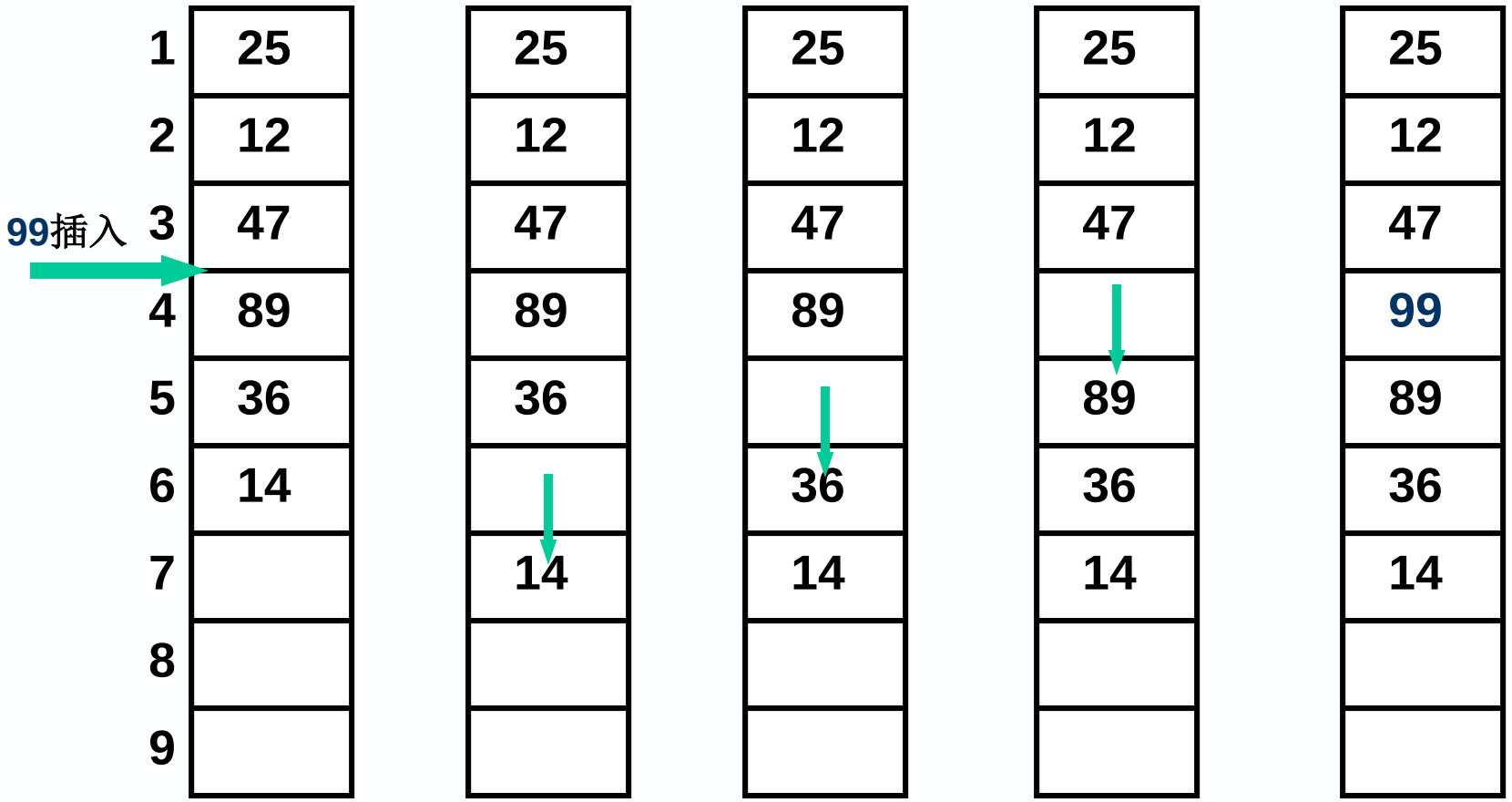
在线性表L中查找值为e的数据元素

```
int LocateELem(SqList L,ElemType e)
{
  for (i=0;i< L.length;i++)
      if (L.elem[i]==e) return i+1;
  return 0;
}
```

查找算法时间效率分析？？？

Click Here

4. 插入（插在第 i 个结点之前）



插第 4 个结点之前，移动 6 - 4+1 次

插在第 i 个结点之前，移动 n-i+1 次

【算法步骤】

(1) 判断插入位置i 是否合法。

(2) 判断顺序表的存储空间是否已满。

(3) 将第n至第i 位的元素依次向后移动一个位置，空出第i个位置。

(4) 将要插入的新元素e放入第i个位置。

(5) 表长加1，插入成功返回OK。

【算法描述】

4. 在线性表L中第i个数据元素之前插入数据元素e

```
Status ListInsert_Sq(SqList &L,int i ,ElemType e){
   if(i<1 || 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
   for(j=L.length-1;j>=i-1;j--)
       L.elem[j+1]=L.elem[j];    //插入位置及之后的元素后移
    L.elem[i-1]=e;                    //将新元素e放入第i个位置
  ++L.length;                        //表长增1
  return OK;
}
```

【算法分析】

算法时间主要耗费在移动元素的操作上

若插入在尾结点之后，则根本无需移动（特别快）；

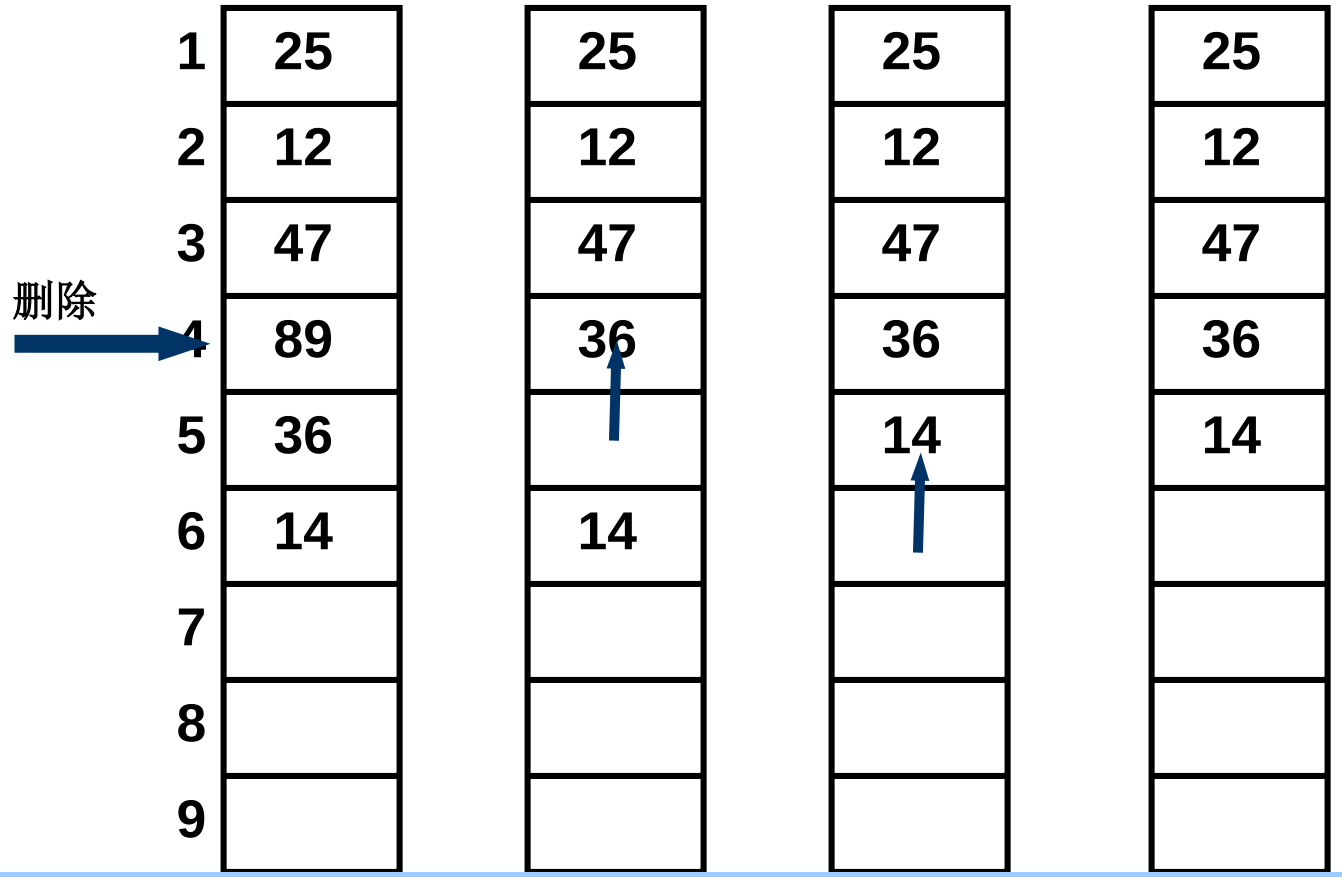
若插入在首结点之前，则表中元素全部后移（特别慢）；

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?

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

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

5. 删除（删除第 i 个结点）



删除第 4 个结点，移动 6 - 4 次

删除第 i 个结点，移动 n-i 次

【算法步骤】

(1) 判断删除位置i 是否合法（合法值为1≤i≤n）。
(2) 将欲删除的元素保留在e中。
(3) 将第i+1至第n 位的元素依次向前移动一个位置。
(4) 表长减1，删除成功返回OK。

【算法描述】

5. 将线性表L中第i个数据元素删除

```
Status ListDelete_Sq(SqList &L,int i){
   if((i<1)||(i>L.length)) return ERROR;      //i值不合法
   for (j=i;j<=L.length-1;j++)
         L.elem[j-1]=L.elem[j];       //被删除元素之后的元素前移
   --L.length;                                //表长减1
  return OK;
}
```

【算法分析】

算法时间主要耗费在移动元素的操作上

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?

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

查找、插入、删除算法的平均时间复杂度为

O(n)

显然，顺序表的空间复杂度S(n)=O(1)

（没有占用辅助空间）

16:15