第二章 线性表

主要内容：线性结构的定义、存储、操作实现

线性结构的特点——一对一

- 数学模型为线性结构，其中的数据元素存在**一对一**的关系：
  - 存在唯一的被称为“**第一个**”的数据元素；
  - 存在唯一的被称为“**最后一个**”的数据元素；
  - 除第一个之外，集合中的每个数据元素有**唯一的直接前驱**；
  - 除最后一个之外，集合中的每个数据元素有**唯一的直接后继**。
- 线性表、栈、队列等是线性结构

2.1 线性表的定义

- 线性表：是$n(n\geq 0)$个数据元素的有限序列，记作List=$(a_1,a_2,\ldots\ldots,a_n)$, n为表长，n=0时为**空表**； n>0时为非空表，满足以下条件：

1. $a_i(1\leq i\leq n\text{-}1)$有唯一的直接后继$a_{i+1}$

   $a_i(2\leq i\leq n)$有唯一的直接前驱a_{i-1}

2. 有唯一的被称为“**第一个**”的数据元素a_1，和唯一被称为“**最后一个**”的数据元素a_n

3. a_1到a_n的性质完全相同，属同一**数据对象**

- **数据元素在线性表中的位置取决于它自身的序号**

线性表举例

	姓　名	学　号	性　别	年　龄	班　级	健康状况
a_1	王小林	790631	男	18	计 91	健康
a_2	陈　红	790632	女	20	计 91	一般
a_3	刘建平	790633	男	21	计 91	健康
a_4	张立立	790634	男	17	计 91	神经衰弱
	: :	: :	: :	: :	: :	: :

$a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow \ldots\ldots$　　一对一

线性表举例

- LIST2=（A, B, ...， Z)
  - “A”是第一个数据元素 ，
  - “Z”是最后一个数据元素；
  - “B”的直接前驱为“A”，直接后继为“C”
  - 表长n=26

线性表操作

- **插入、删除、创建、查找**等等
- 操作的实现要根据线性表的存储结构（物理结构）设计

线性表的抽象数据类型定义

ADT List{ // List是为线性表抽象数据类型起的名字

数据对象: $D=\{a_i: a_i \in ElemSet; 1 \le i \le n; n \ge 0\}$

数据关系: $R=\{<a_i, a_{i+1}>: a_i, a_{i+1} \in D, 1 \le i \le n-1\}$

基本操作:

InitList(&L)

初始条件：表L不存在

操作结果：构造一个空的线性表

DestroyList(&L)

初始条件：线性表L存在

操作结果：销毁线性表L

ClearList(&L)

初始条件：线性表L存在

操作结果：将线性表L置为空表

ListEmpty(L)

初始条件：线性表L存在。

操作结果：若L为空，则返回TRUE;否则返回FALSE。

GetElem(L,*i*, &*e*)

初始条件：表L存在且1≤*i*≤List Length(L)

操作结果：返回线性表L中的第*i*个元素的值

List Length (L)

初始条件：表L存在

操作结果：返回线性表中的所含元素的个数

……

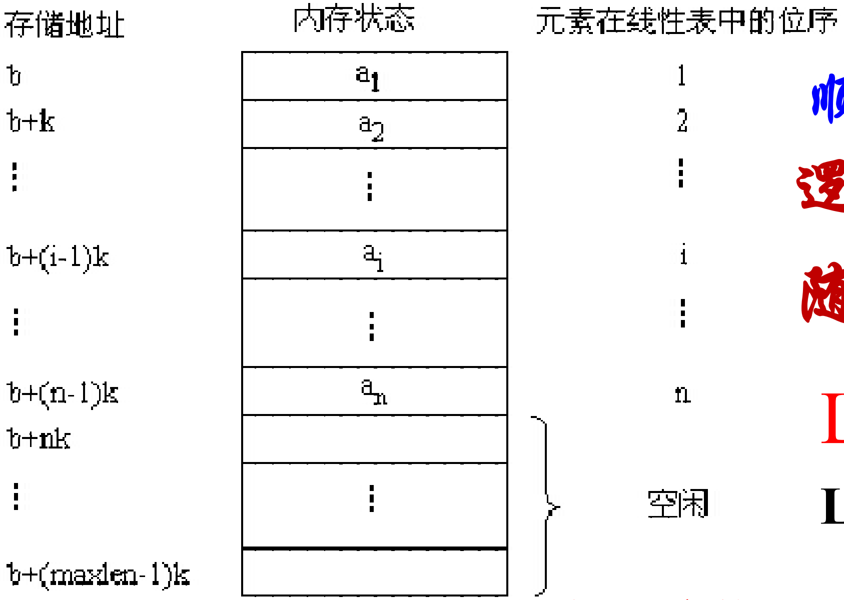
}ADT List

线性表的抽象数据类型定义

- 若实现了线性表的抽象数据类型定义，那么可以定义该类型的变量，并调用其包含的基本操作，例如：
  - List L;
  - GetElem(L,5, &*e*); printf(“%f”,e);
- 实现了线性表的抽象数据类型定义（1）解决存储，（2）实现操作

线性表的顺序表示和实现

- 性表**顺序存储结构**是用一组地址<u>**连续**</u>的存储单元<u>**依次**</u>存储线性表的数据元素
- 以顺序存储结构存放的线性表称为***顺序表***



List=(a_1,a_2,……,a_n)的顺序存储结构图示

顺序存储结构特点1:

逻辑相邻 → 物理相邻

随机存取

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

LOC(a_i)为a_i的存放地址

顺序表的实现——方法1静态**数组**

- **define *maxlen* 100**

```
int elem[maxlen];
int length;
```

线性表中第一个数据元素存放于数组中下标为0的数组元素处

顺序表的实现--方法1静态数组

- define *maxlen* 100
- typedef struct{

      int elem[*maxlen*];
      int length;
  } SeqList;
- SeqList L;
- L.length=*n*

L.elem[0]	a_1
L.elem[1]	a_2
	⋮
L.elem[*i*-1]	a_i
	⋮
L.elem[*n*-1]	a_n
	⋮
L.elem[*maxlen*-1]	

设置一个足够大的数组（可存放maxlen个数组元素）存放线性表，L.length存放任意时刻表中实际含有的数据元素个数n

顺序表的实现--方法2 指针数组

- define LIST_INIT_SIZE 100 //指针数组初始申请空间的大小
- define LISTINCREMENT 10 //指针数组每次扩大空间的大小
- typedef struct

```
      {   int *elem;
          int length;
          int listsize;
      } SqList;
```

- SqList L;

表示设置一个足够大的数组存放线性表，
L.listsize任何时刻数组的容量，
L.length存放任意时刻表中实际含有的数据元素个数n

InitList——指针数组的初始化操作：建空的线性表

- int InitList(SqList &L)//指针数组

```
{
    L.elem=(int *)malloc(LIST_INIT_SIZE*sizeof(int));
    if(L.elem==0) exit(OVERFLOW);
    L.length=0; //建的是空的线性表，含0个数据元素
    L.listsize=INIT_LIST_SIZE; //当前用于存放线性表的数组的容量
    return OK;//OK为预先定义的常量1
}
```

静态数组的初始化操作：建空线性表?

线性表的插入InserList

- **问题**：在线性表的第i个数据元素前插入一个值为x的新元素。
- **分析：**
- 插入前：$(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$

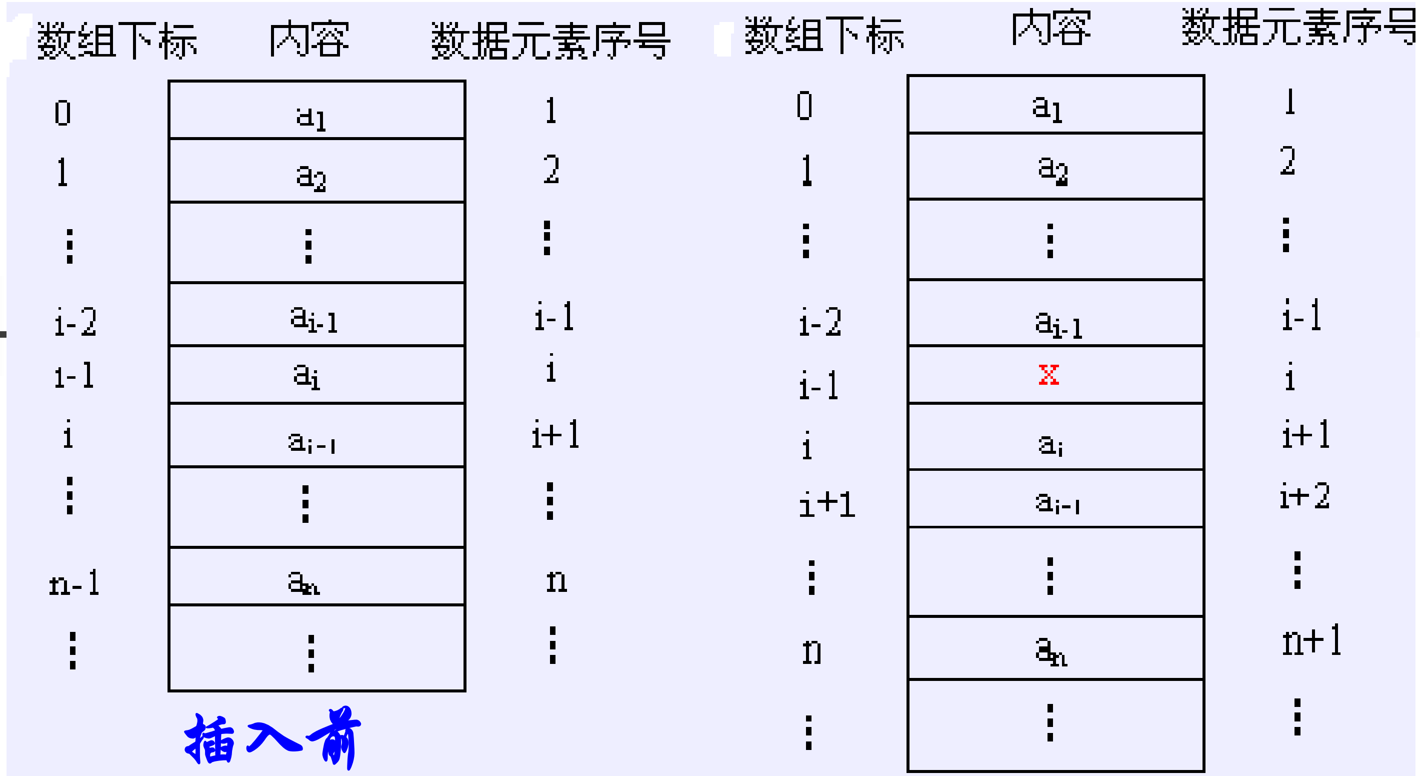
  插入后：$(a_1, a_2, \ldots, a_{i-1}, x, a_i, a_{i+1}, \ldots, a_n)$

  插入后表长为n+1,插入操作进行前要检查：

> 插入位置i的合理性：$1 \le i \le n+1$
> 空间是否够用— L.length <=L.listsize?（指针数组）或
> L.length <=*maxlen*?（静态数组）

L.Length代表线性表中目前有多少个数据元素，也即

$(a_1, a_2, \ldots, a_n)$中的n



插入后

- **插入算法步骤：**
  - 判断插入位置是否合法
  - 判断存储空间是否溢出
  - 将$a_n \sim a_i$顺序向下移动，为新元素让出位置；
  - 将 x 置入空出的第i个位置；
  - **修改表长**

```
int ListInsert(SqList &L, int i, int x)
{  int *j,*newbase;
   if ( i <1|| i>L.length+1) return -1;//检查插入位置i的合理性
   if (L.length==L.listsize) //检查空间是否够用
   {//检查空间不够用时，增大空间
        newbase=(int*)realloc(L.elem,(L.listsize+
                  LISTINCREMENT)* sizeof(int));
        if(newbase==0) exit(OVERFLOW);
        L.elem=newbase;
        L.listsize= L.listsize+LISTINCREMENT;
   }
   //将an～ai顺序向下移动，为新元素让出位置
   for(j=L.length;j>=i;j--) L.elem[j]=L.elem[j-1];
   L.elem[i-1]=x;//插入新元素
   ++L.length;//修改表长
   return 1;
}
```

```
int ListInsert(SqList &L, int i, int x)
{  int *p,*q;
   if ( i <1|| i>L.length+1) return -1;
   if (L.length==L.listsize)
   {

    newbase=(int*)realloc(L.elem,(L.listsize+LISTINCREMENT)*size
    of(int));
       if(newbase==0) exit(OVERFLOW);
       L.elem=newbase;
       L.listsize= L.listsize+LISTINCREMENT;
   }
   q=&(L.elem[i-1]);
   for(p=&(L.elem[L.length-1]);p>=q;--p) *(p+1)=*p;
   *q=x;
   ++L.length;
   return OK;
}
```

另一种表达方式

插入算法分析

- 算法的基本操作是***数据移动***
- $(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$每个位置进行插入的概率相等条件下，插入算法的平均时间复杂度
- 做一次插入平均要移动数据多少次?

插入算法分析

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

  i=2　n-1 次

  .　　.

  .　　.

  i　　n-i+1 次

  .　　.

  n　　1 次

  n+1　0 次

插入算法分析

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

  i=2　n-1 次

  .　　.

  .　　.

  i　　n-i+1 次

  .　　.

  n　　1 次

  n+1　0 次

插入一次平均移动数据：

(n+n-1+…+n-i+1+…+1+0)/(n+1)

=n/2

插入算法分析

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

  i=2 n-1 次

  . .

  . .

  i n-i+1 次

  . .

  n 1 次

  n+1 0 次

插入一次平均移动数据：

(n+n-1+…+n-i+1+…+1+0)/(n+1)

=n/2 插入操作数据移动量大

算法的时间复杂度 $O(n)$

删除ListDelete

- **问题**：删除线性表的第 i 个数据元素
- **分析：**

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

➢ 删除后：$(a_1,\ a_2,\ \ldots,\ a_{i-1},\ a_{i+1},\ \ldots,\ a_n)$

删除后表长为 n-1

➢ **删除操作进行前要检查**

i 的取值范围为：$1 \le i \le n$

- 步骤：

(1) 将a_{i+1}～a_n顺序向上移动。

(2) 修改表长

数组下标	内容	数据元素序号
0	a_1	1
1	a_2	2
⋮	⋮	⋮
i-2	a_{i-1}	i-1
i-1	a_i	i
i	a_{i+1}	i+1
⋮	⋮	⋮
n-1	a_n	n
⋮	⋮	⋮

删除前

数组下标	内容	数据元素序号
0	a_1	1
1	a_2	2
⋮	⋮	⋮
i-2	a_{i-1}	i-1
i-1	a_{i+1}	i
⋮	⋮	⋮
n-2	a_n	n-1
⋮	⋮	⋮

删除后

算法ListDelete

```
int ListDelete(SqList &L; int i)
{ int  j;
     if(i<1 || i>L.length)
         { printf ( " 不存在第i个元素 " ); return 0; }
     for(j=i+1;j<=L.length;j++)
         L.elem[j-2]=L.elem[j-1];
     L.length--;
     return 1;
 }
```

算法ListDelete

```
int ListDelete(SqList &L; int I, int& e)
{ int  j;
      if(i<1 || i>L.length)
           { printf ( " 不存在第i个元素 " ); return 0; }
      e=L.elem[i-1];
      for(j=i+1;j<=L.length;j++)
           L.elem[j-2]=L.elem[j-1];
      L.length--;
      return 1;
 }
```

删除算法分析

- 算法的基本操作是数据移动
- $(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$每个位置进行删除的概率相等条件下，删除算法的平均时间复杂度
- 做一次删除平均要移动数据多少次?

删除算法分析

- (a_1, a_2, … , a_{i-1}, a_i, a_{i+1}, … , a_n)
- i=1　n-1 次

i=2　n-2 次

.　　.

.　　.

i　　n-i 次

.　　.

n　　0 次

删除算法分析

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

  i=2　n-2 次

  .　.

  .　.

  i　n-i 次

  .　.

  n　0 次

删除一次平均移动数据

(n-1+n-2+…+n-i+…+1)/n

=(n-1)/2

删除算法分析

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

  i=2 n-2 次

  . .

  . .

  i n-i 次

  . .

  n 0 次

删除一次平均移动数据

(n-1+n-2+…+n-i+…+1)/n

=(n-1)/2 删除操作数据移动量大

算法的时间复杂度 $O(n)$

查找ListSearch

- 问题：在线性表中的查找与给定值x相等的数据元素。若存在返回其位置序号。
- 分析：
- 从第一个元素 a_1 起依次和x比较，直到找到一个与x相等的数据元素，则返回它在顺序表中的序号；或者查遍整个表都没有找到与 x 相等的元素，返回-1。

查找算法

- ```
  int ListSearch(SqList L, int x)
  {  int i;
     for(i=0;i<L.length;i++)
     if ( L.elem[i]== x) return i+1;
     return -1;
  }
  ```
- $O(n)$
  ```

小结

- 逻辑相邻一定物理相邻
- 可以随机存取
- 空间问题:插入操作要判断空间是否够用
- 插入和删除的数据移动量大，等概率情况下，平均一次插入（或删除）要移动n/2(或(n-1)/2)的数据
- 按数据元素的值查找为O(n)
- 按数据元素的位置序号查找O(1)