第二章 线性表（一）

马媛

2021/09/17

多项式的表示

[例] 一元多项式及其运算

一元多项式 $f(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n$

主要运算：多项式相加、相减、相乘等

[分析] 如何表示多项式?

多项式的关键数据：

- 多项式项数 n
- 各项系数a_i及指数 i

方法1：顺序存储结构直接表示

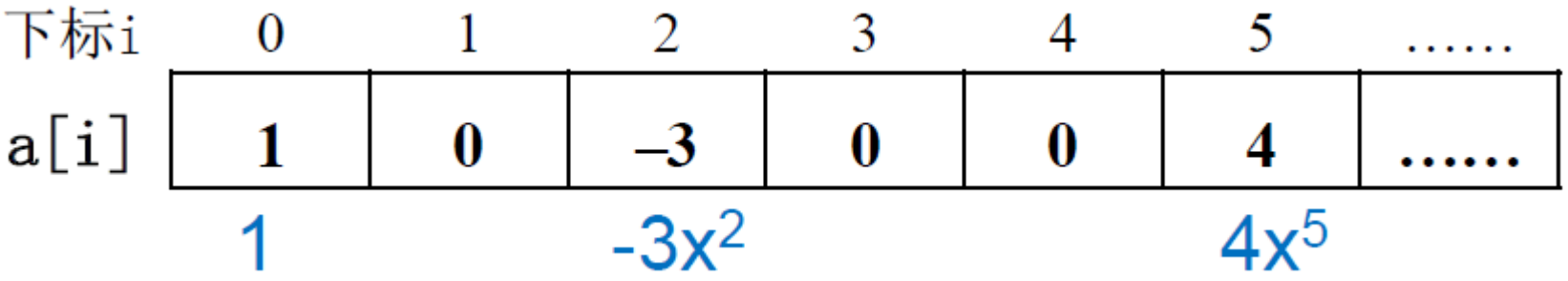
- 数组各分量对应多项式各项：

  $a[i]$：项x^i的系数a_i

例如：　$f(x) = 4x^5 - 3x^2 + 1$

表示成：

问题：如何表示多项式$x + 3x^{2000}$？



两个多项式相加：　两个数组对应分量相加

方法2：顺序存储结构表示非零项

- 每个非零项$a_i x^i$涉及两个信息：系数a_i和指数 i

  可以将一个多项式看成是一个(a_i, i)二元组的集合

- 用结构数组表示：数组分量是由系数a_i, 指数i组成的结构，对应一个非零项

例如： $P_1(x) = 9x^{12} + 15x^8 + 3x^2$ 和 $P_2(x) = 26x^{19} - 4x^8 - 13x^6$+82

下标i	0	1	2	……
系数a^i	9	15	3	–
指数i	12	8	2	–

(a) $P_1(x)$

下标i	0	1	2	3	……
系数a^i	26	–4	–13	82	–
指数i	19	8	6	0	–

(b) $P_2(x)$

按指数大小有序存储！

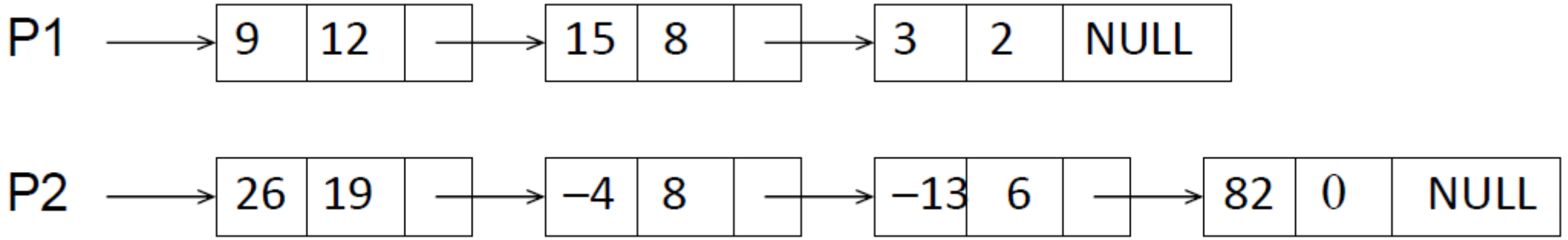
方法3：链表结构存储非零项

- 链表中每个结点存储多项式中的一个非零项，包括系数和指数两个数据域以及一个指针域

系数a_i	指数 i	指针link

例如： $P_1(x) = 9x^{12} + 15x^8 + 3x^2$ 和 $P_2(x) = 26x^{19} - 4x^8 - 13x^6$+82

链表存储形式为：



什么是线性表？

多项式表示问题的启示：

I. 同一个问题可以有不同的表示（存储）方法

II. 有一类共性问题：有序线性序列的组织和管理

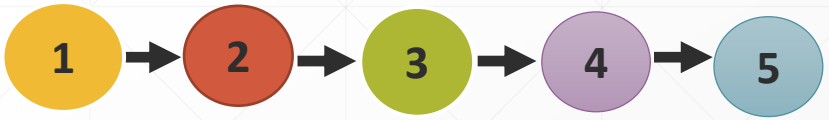
线性表**(Linear List)**：由同类型数据元素构成有序序列的线性结构

$$(a_1, a_2, \ldots, a_i, \ldots, a_n)$$

其中，a_i是表中元素，i表示元素a_i的位置，n是表的长度

线性表

- 表中元素个数称为线性表的长度
- 线性表没有元素时，称为空表
- 表起始位置称表头，表结束位置称表尾
- 除第一个元素外，每个数据元素均只有一个前驱
- 除最后一个元素外，每个数据元素均只有一个后驱



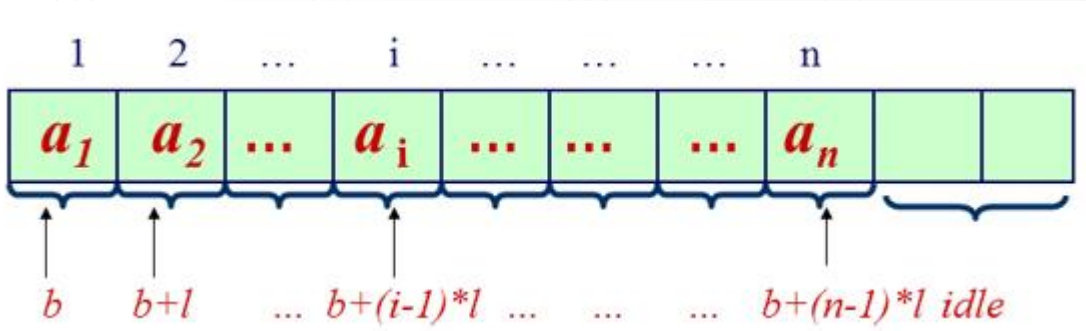
顺序表

- 顺序表是线性表的顺序存储表示
  - 顺序表采用一组地址连续的存储单元依次存储线性表的数据元素
  - 顺序表元素的位置：

$LOC(a_i)=LOC(a_{i-1})+l$

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

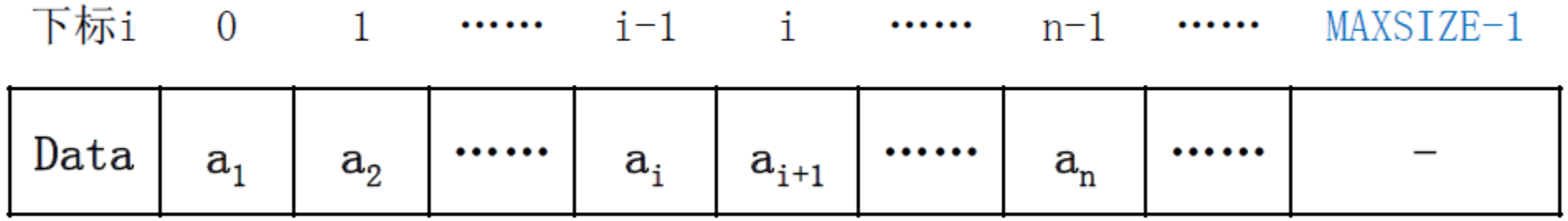
l-元素占用的内存单元数



可以在已知第一个元素的存储起始地址，以及每个元素所占用的存储单元个数基础上，直接计算表中任意指定元素的存储地址

顺序表的定义和创建

- 采用C语言中的数组表示（定义）顺序表

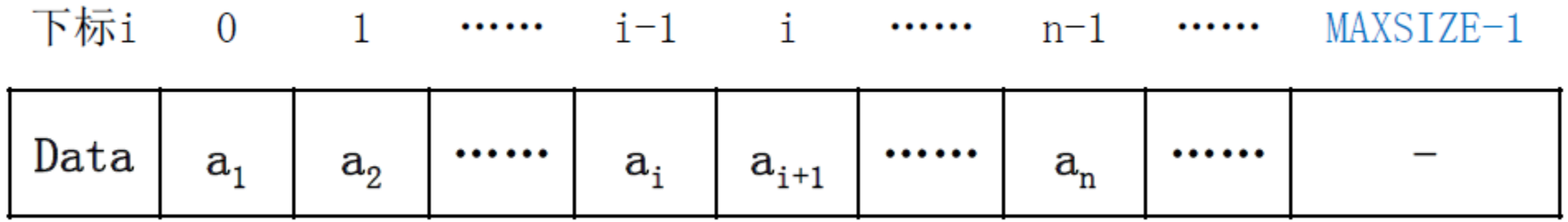


插入、删除 → 表长受影响 → 表长需要保存 → length

表中元素需要保存 → 用一维数组存储 → *elem

顺序表的定义和创建

- 采用C语言中的数组表示（定义）顺序表




```
typedef struct{
    ElemType *elem;  //存储的是数组第一个元素的地址
    int length;  //顺序表的当前长度
}SqList;  //定义了结构体数据类型SqList, 用于表示顺序表
SqList L;
```

抽象表示

数据类型

线性表的长度：L.length;

访问第i个元素：L.elem[i-1]

顺序表的查找

- 根据给定元素的序号进行查找 （通过数组下标定位）
- 根据给定的元素值进行查找

基本思想：将给定的元素e和顺序表中的每个元素依次进行比较

✓ 若找到与e相等的元素，则查找成功，返回其在表中的位序值；

✓ 若找遍整个顺序表，没有找到与e相等的元素，则查找失败，返回-1

顺序表的查找

```
int Locate_Sq(SqList L, ElemType e)
{ int i=0;
  while(i<L.length && e!=L.elem[i])
        i++;
  if (i>=L.length) return -1;  //如果没找到，返回-1
  else return i;     //找到后返回的是存储位置
}
```

查找成功的平均比较次数为 (n +1)/2，平均时间性能为 O(n)。

顺序表的插入

- 第i $(1 \leq i \leq n+1)$个位置上插入一个值为X的新元素

下标 i	0	1	……	i-1	i	……	n-1	……	MAXSIZE-1
Data	a_1	a_2	……	a_i	a_{i+1}	……	a_n	……	–

先移动，再插入

下标 i	0	1	……	i-1	i	i+1	……	n	……	SIZE -1
Data	a_1	a_2	……	X	a_i	a_{i+1}	……	a_n	……	–

顺序表的插入

- 在顺序表中，第i个位置上插入一个元素，需要向后移动元素的个数为：$n-i+1$
- 平均移动元素数为：

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

当插入位置等概论时，$p_i=1/(n+1)$，因此

$$E_{is}=\sum_{i=1}^{n+1}[1/(n+1)]\times(n-i+1)=n/2$$

- 顺序表插入操作的时间复杂度为$\boldsymbol{O}(n)$

顺序表的插入

```
Status Insert_Sq( SqList &L, int i, ElementType x)
{ if ( i<1 || i>L.length+1)    /* 位置不合法*/
        return ERROR;
  if ( L.length>=MAXSIZE)  /* 表空间已满，不能插入*/
        return ERROR;
  for ( k=L.length-1; k >= i-1; k--)
        L.elem[k+1]=L.elem[k];    /*将 a_i ～ a_n倒序向后移动*/
  L.elem[i-1]=x;    /*新元素插入*/
  L.length++;      /*表长+1*/
  return OK;
}
```

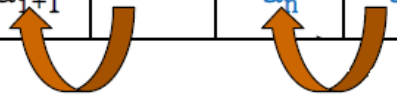
顺序表的删除

- 删除表的第$i(1 \le i \le n)$个位置上的元素

下标 i	0	1	……	i-1	i	……	n-1	……	MAXSIZE-1
Data	a_1	a_2	……	a_i	a_{i+1}	……	a_n	……	–

后面的元素依次前移

下标 i	0	1	……	i-1	……	n-2	n-1	……	MAXSIZE-1
Data	a_1	a_2	……	a_{i+1}	……	a_n	a_n	……	–



顺序表的删除

- 在顺序表中，删除第i个位置上的元素，需要向前移动元素的个数为：$n-i$
- 平均移动元素数为：

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

当插入位置等概论时，$q_i=1/(n)$，因此

$$E_{is}=\sum_{i=1}^{n}[1/n] \times(n-i)= (n-1)/2$$

- 顺序表删除操作的时间复杂度为$\boldsymbol{O}(n)$