2.5 线性表的应用实例

1、一元多项式的表示

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{i=0}^{n} a_i x^i$$

- n阶多项式$P_n(x)$有n+1项。
  - 系数 $a_0, a_1, a_2, \ldots, a_n$
  - 指数 0, 1, 2, ..., n。按升幂排列

第一种表示方法

建立多项式系数的表 $P_n=(a_0, a_1, a_2, \ldots, a_n)$，用数组表示多项式的系数，数组下标表示指数。

	0	1	2		degree	maxDegree
coef	a_0	a_1	a_2	·····	a_n	·····
					↑ n	

适用于指数连续排列、“0”系数较少的情况。但对于指数不全的多项式，如$P_{20000}(x) = 3 + 5x^{50} + 14x^{20000}$，会造成系统空间的巨大浪费。

	0	1	2	3	……50	…… ….20000								
coef	3	0	0	0		5	0	0		14				

第二种表示方法

一般情况下，一元多项式可写成：

$P_n(x)=p_1x^{e1}+p_2x^{e2}+\ldots+p_mx^{em}$

其中：p_i是指数为ei的项的非零系数，

$0 \le e1 \le e2 \le \ldots \le em \le n$

抽象成二元组表示的表：$((p_1,e1),(p_2,e2), \ldots ,(p_m,em))$

例：$P_{999}(x) = 7x^3 - 2x^{12} - 8x^{999}$

表示成: ((7,3),(-2,12),(-8,999))

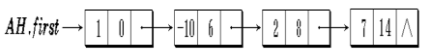
利用数组进行存储如下：

	0	1	2		i		m
coef	a_0	a_1	a_2	·····	a_i	·····	a_m
exp	e_0	e_1	e_2	·····	e_i	·····	e_m

利用链表进行存储如下：

$AH = 1 - 10x^6 + 2x^8 + 7x^{14}$

（1,0），(-10,6)，(2,8)， (7,14)

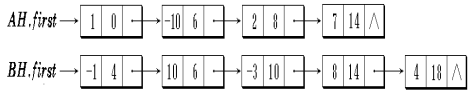


$AH = 1 - 10x^6 + 2x^8 + 7x^{14}$

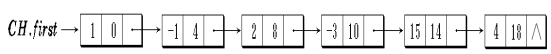
（1,0），(-10,6)，(2,8)， (7,14)

$BH = -x^4 + 10x^6 - 3x^{10} + 8x^{14} + 4x^{18}$

（-1,4）， (10,6)， (-3,10)， (8,14)， (4,18)



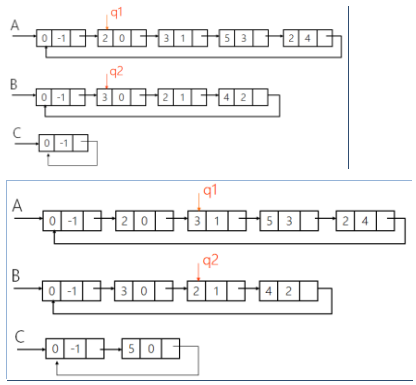
(a) 两个相加的多项式

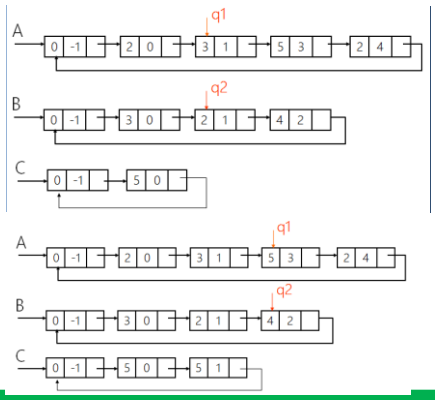


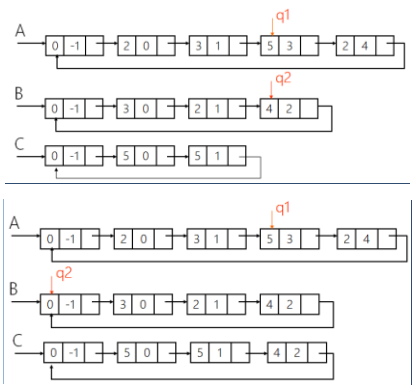
(b) 相加结果的多项式

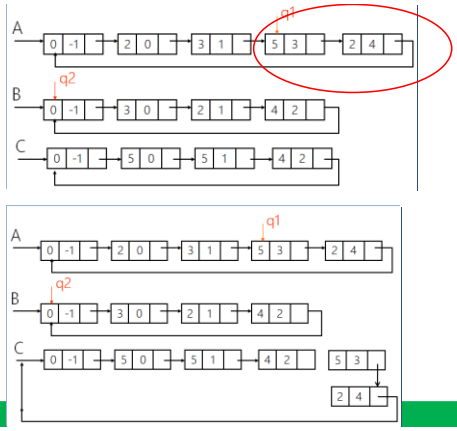
算法思想：

初始化；
While (两个链都没处理完)
{ if (指针指向当前节点的指数项相同)
{系数相加，在C链中填加新的节点；
A、B链的指针均前移；}
else
{以指数小的项的系数添入C链中的新节点；
指数小的相应链指针前移；}
}
While(A链处理完)
{ 顺序处理B链；}
While(B链处理完)
{ 顺序处理A链；}

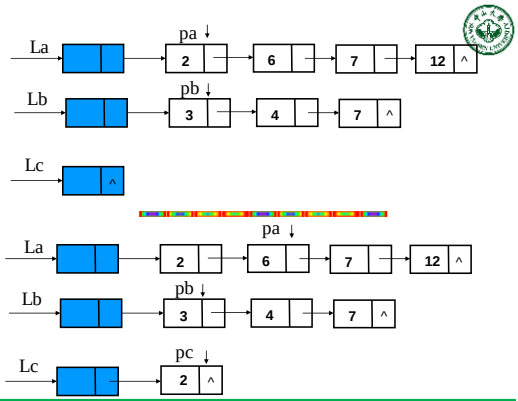


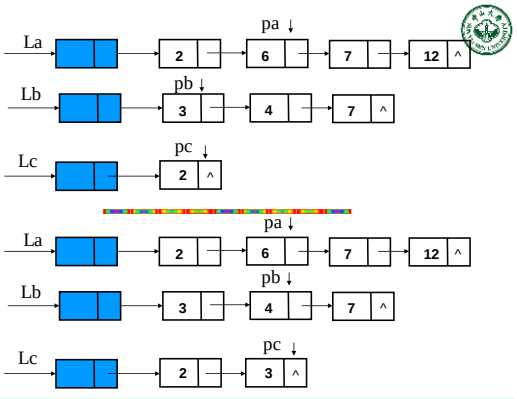






- 设有两个递增有序的单链表，它们的头指针分别是La和Lb，将它们合并为以Lc为头指针的递增有序的单链表。
- 设有两个递增有序的单链表，它们的头指针分别是La和Lb，将它们合并为以Lc为头指针的递减有序的单链表。

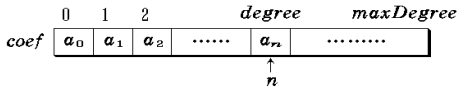




2. 数组下标与线性表的应用

- **规律：当被处理对象之间符合表的特征时建立表，用数组存储表中元素时，对数据元素的加工处理算法，可以充分利用数据元素的下标变化规律。**

针对建立的多项式系数的表 $P_n=(a_0, a_1, a_2, \ldots, a_n)$，用数组表示多项式的系数，数组下标表示指数。



每一个问题中的信息往往是多方面的，在算法中一般有输入信息、输出信息和信息加工处理过程中的中间信息。如何确定用数组进行信息存储，数组元素下标与信息如何对应等问题，很大程度上影响着算法的编写效率和运行效率。

下面的例子恰当地选择了用数组存储的信息，并把题目中的有关信息作为下标使用，使算法的实现过程大大简化。

【例1】某次选举，要从五个候选人（编号分别为1、2、3、4、5）中选一名厂长。请编程完成统计选票的工作。

算法设计：

1）虽然选票发放的数量一般是已知的，但收回的数量通常是无法预知的，所以算法采用随机循环，设计停止标志为“-1”。

2）统计过程的一般方法为：先为五个候选人各自设置五个“计数器”S1，S2，S3，S4，S5，然后根据录入数据通过多分支语句或嵌套条件语句决定为某个“计数器”累加1，这样做效率太低。

建立候选人的一个线性表并用数组进行存储，让选票中候选人的编号xp做下标，执行A[xp]=A[xp]+1就可方便地将选票内容累加到相应的“计数器”中。也就是说数组结构是存储统计结果的，而其下标正是原始信息。

3）考虑到算法的健壮性，要排除对1-5之外的数据进行统计。

算法描述：

```
vote( )
{ int i,xp,a[6];
  print(“input data until input -1”);
  input(xp );
  while(xp!=-1)
  { if (xp>=1 and xp<=5 )
       a[xp]=a[xp]+1;
   else
      print(xp, "input error!");
    input(xp );
   }
   for (i=1;i<=5;i++)
     print(i,"number get", a[i], "votes");
}
```

【例2】编程统计身高（单位为厘米）。统计分150-154；155-159；160-164；165-169；170-174；175-179及低于150、高于179共八档次进行。

算法设计：

输入的身高可能在50-250之间，若用输入的身高数据直接作为数组下标进行统计，要开辟200多个空间，计数之后还要再次求和。

而统计是分八个档次进行的，统计区间的大小是都固定为5，则可以建立八个档次的线性表，线性表的元素对应档次。

这样用“身高/5-29”做下标，则只需开辟8个元素的数组，对应八个统计档次，即可完成统计工作。

0 1 2 3 4 5 6 7

150, 150-154, 155-159, 160-164, 165-169, 170-174, 175-179, 179

152/5-29=1 161/5-29=3 170/5-29=5

算法如下：

```
main( )
{ int i,sg,a[8];
print("input height data until input -1");
input(sg );
while ( sg<>-1 )
   {if (sg>179)   a[7]=a[7]+1 ;
    else
      if (sg<150)   a[0]=a[0]+1 ;
      else    a[sg/5-29]=a[sg/5-29]+1 ;
       input( sg );
   }
 for (i=0;i<=7;i=i+1)
  print(i+1 ,"field the number of people : ", a[i]);
}
```


【例3】开灯问题

有从1到*n*依次编号的*n*个同学和*n* 盏灯。

1号同学将所有的灯都关掉；

2号同学将编号为2的倍数的灯都打开；

3号同学则将编号为3的倍数的灯作相反处理（该号灯如打开的，则关掉；如关闭的，则打开）；

以后的同学都将自己编号的倍数的灯，作相反处理。

问：经*n*个同学操作后，哪些灯是打开的？

- 问题分析：
- 1）用数组表示某种灯的状态，这里定义有*n*个元素的a数组，它的每个下标变量a[*i*]视为一灯，*i*表示其编号。

  a[*i*]=1表示灯处于打开状态，a[*i*]=0表示灯处于关闭状态。
- 2）实现将第*i* 灯作相反处理的操作：
  - 条件语句if表示：

    if a[i] == 1 a[i] = 0；

    if a[i] == 0 a[i] = 1；
  - 通过算术运算更简练、逼真：

    a[i]=1-a[i]。

此用法也称为“乒乓开关”。简化逻辑表达式、减少条件判断

- int a[1000];

输入n的数值；

关闭所有灯，即a[1]~a[n]置为0；

第2个学生->第n个学生(学生i)进行操作：

操作对象：数组a，灯编号含因数i，即a[i*k]；

操作： a[i*k] = 1 - a[i*k]；

输出灯的开关状态。

【例4】某一次考试共考了语文、代数和外语三科。一个小组共有九人，考后各科及格名单如下表，请编写算法找出三科全及格的学生的名单（学号）。

科目	及格学生学号
语文	1，9，6，8，4，3，7
代数	5，2，9，1，3，7
外语	8，1，6，7，3，5，4，9

方法一：

- 从语文及格名单中逐一抽出各及格学生学号，先在代数及格名单核对，若有该学号（说明代数也及格了），再在外语及格名单中继续查找，看该学号是否也在外语及格名单中。若仍在，说明该号属全及格学生的学号，否则就是至少有一科不及格的。若语文及格名单中就没有某生的号，显然该生根本不在比较之列，自然不属全及格学生。
- 方法采用了枚举尝试的方法
- A，B，C三组分别存放语文、代数、外语及格名单，尝试范围为三重循环：
- I循环， 初值0， 终值6， 步长1
- J循环， 初值0， 终值5， 步长1
- K循环， 初值0， 终值7， 步长1
- 定解条件： A[I]=B[J]=C[K]
- 共尝试7*6*8=336次。


```
main( )
 {int   a[7], b[6], c[8],i,j,k,v,flag;
  for( i =0;i<=6;i=i+1)    input(a[i]);
  for( i =0;i<=5;i=i+1)    input(b[i]);
  for( i =0;i<=7;i=i+1)    input(c[i]);
  for( i =0;i<=6;i=i+1)
  {v=a[i];
   for( j =0;j<=5;j=j+1)
      if ( b[j]=v )
         for(k =0;k<=7;k=k+1)
            if(c[k]=v)
          {print(v); break;}
   }
}
```


方法二 ：

- 建立九个学生的线性表并用数组A存储，数组A的下标分量作为各学号考生及格科目的计数器。当扫描完毕总及格名单后，凡下标计数器的值为3的就是全及格的，其余则至少有一科不及格的。
- 该方法同样也采用了枚举尝试的方法。


```
main( )
{int  a[10],i,xh;
for(i =1;i<=21;i=i+1)
  {input(xh);
   a[xh]=a[xh]+1;}
for(xh =1;xh<=9;xh=xh+1)
  if(a[xh] =3 )
     print(xh);
}
```

共尝试7+6+8=21次。

【例5】编程将编号“翻译”成英文。例35706“翻译”成three-five-seven-zero-six。

- 算法设计：
- 1）编号一般位数较多，可按长整型输入和存储。
- 2）通过求余、取整运算，可以取到编号的各个位的数字。用这个数字通过**if 语句**就可以**找到**对应的英文数字。

- 改进的算法设计：
- 将英文的“zero-nine”存储在数组中，对应下标为0-9。这样无数值规律可循的单词，通过下标就可以方便存取、访问了。


```
main( )
 {int i,a[10], ind;    long num1,num2;
  char eng[10][6]={“zero”,”one”,”two”,”three ”,” four”,
 ” five”,”six”,”seven”,“eight”,”nine”};
  print(“Input a num”);
  input(num1);    num2=num1;  ind =0;
 while (num2<>0)
{a[ind]=num2 mod 10;  ind= ind +1;  num2=num2/10; }
 print(num1, “English_exp:”, eng[a[ind-1]]);
for( i=ind-2;i>=0;i=i-1)
   print(“-”，eng[a[i]]);
}
```

小　结

- 表的逻辑结构及其特点
- 表的物理结构及其特点
- 表的基本操作算法及性能分析
- 表的应用实例