数据结构与算法

第二章 线性表（一）

王 昭
北京大学计算机学院

wangzhao@pku.edu.cn

温故而知新

- 对于数值计算问题，处理的对象为简单的数值，数学模型为数学方程；
- 对于非数值计算问题（如资料查询、交通管理等），处理的对象之间具有一定的逻辑关系，必须根据对象之间的逻辑关系建立描述问题的数据结构。
- 数据结构皆按照三个方面进行：
  - 逻辑定义（逻辑结构）
  - 存储结构
  - 及其各种运算的实现

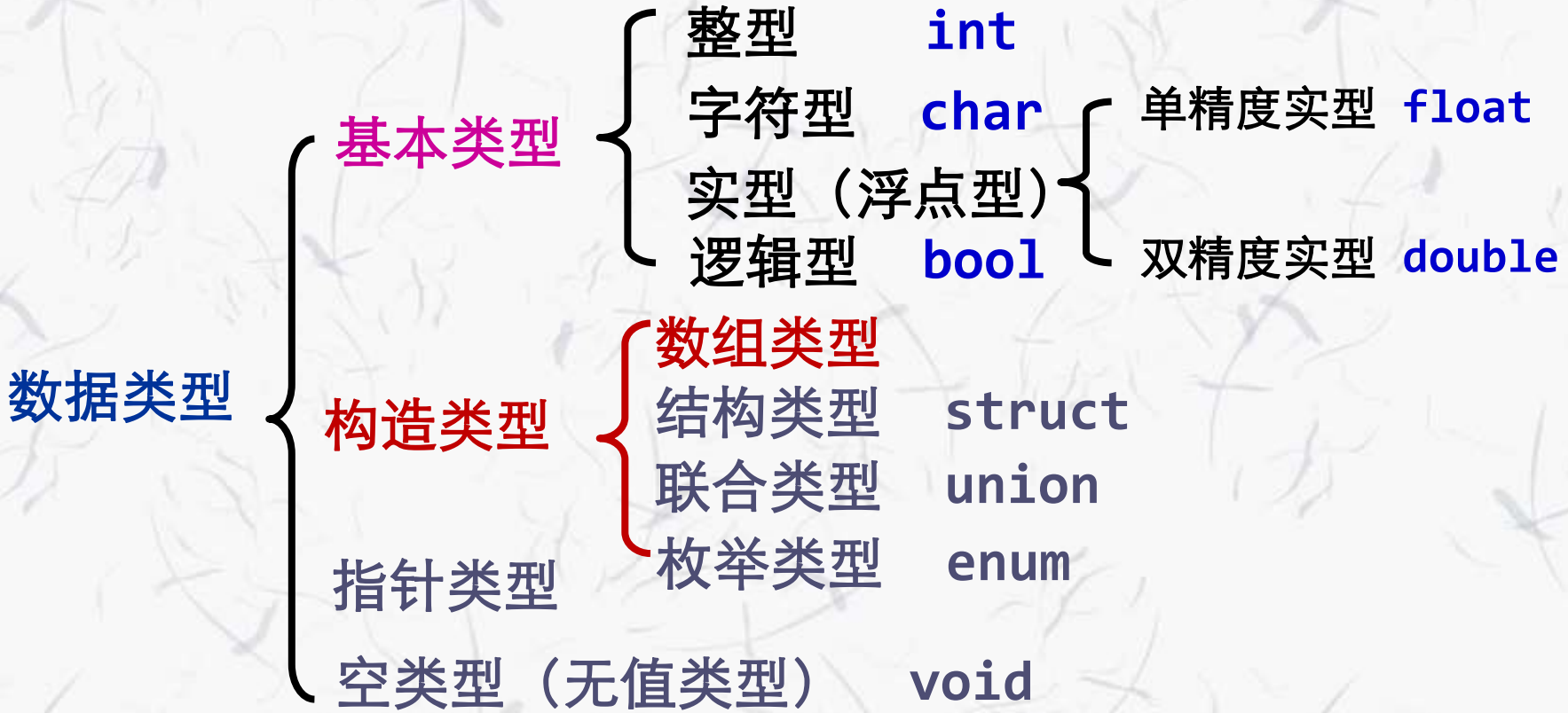
线性表

- *教学目的：*介绍线性表的逻辑结构和各种存储表示方法，以及定义在逻辑结构上的各种基本运算及其在存储结构上如何实现这些基本运算。要求在熟悉这些内容的基础上，能够针对具体应用问题的要求和性质，选择合适的存储结构设计出相应的有效算法，解决与线性表有关的实际问题。
- *教学重点：*熟练掌握顺序表和单链表上实现的各种基本算法及相关的时间性能分析
- *教学难点：*能够使用本章所学到的基本知识设计有效算法解决与线性表有关的应用问题。

内容提要

- 数据类型和抽象数据类型
- 线性表的概念（ADT）
- 线性表的顺序表示和实现（顺序表）
- 线性表的链式表示和实现（链表）
- 线性表的应用

C语言的基本数据类型


数据类型
基本类型
整型 int
字符型 char
实型（浮点型）
单精度实型 float
双精度实型 double
逻辑型 bool
构造类型
数组类型
结构类型 struct
联合类型 union
枚举类型 enum
指针类型
空类型（无值类型） void

数据类型

- 类型：是一组值（或者对象）的集合。
- 数据类型是高级语言用以刻划操作对象特性的一个概念。
- 数据类型：具有相同性质的计算机数据的集合及定义在这个数据集合上的一组运算的总称。
  - 隐含规定了程序执行期间变量的取值范围和允许的操作；
  - 如整型：[-minint, maxint]； +，-，*，/，%
  - 原子类型：int, char, float,….不可分解
  - 结构体类型：struct A { ……}， 可以分解

抽象数据类型
(Abstract Data Type, ADT)

- 定义：具有一定行为（操作）的抽象（数学）类型。
  - 例：线性表这样的抽象数据类型，其数学模型是：数据元素的集合，该集合内的元素有这样的关系：除第一个和最后一个外，每个元素有唯一的前趋和唯一的后继。可以有这样一些操作：插入一个元素、删除一个元素等。
- 在许多程序设计语言中预定义的类型，例如整型、实型、指针类型等，都可以看作是简单的抽象数据类型。
- 使用数据类型的人可以只关心它的逻辑特征，不需要了解类型中值的存储方式和类型中定义的运算的具体实现方法。

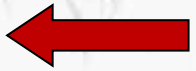
抽象数据类型的表示

- 三元组表示（D，R，P）
- D：数据对象，性质相同的数据元素的集合
- R：D上的有穷关系集合
- （D，R）指的是数据的逻辑结构，是对客观事物的抽象描述
- P：对D的基本运算集合

- ADT = 逻辑结构 + 基本运算集合

内容提要

- 数据类型和抽象数据类型
- 线性表的概念（ADT）
- 线性表的顺序表示和实现（顺序表）
- 线性表的链式表示和实现（链表）
- 线性表的应用



线性结构的特点

- 结构中的元素之间满足线性关系，按这个关系可以把所有元素排成一个线性序列。在数据元素的非空有限集中，
  - 存在唯一的一个被称为“第一个”的数据元素；
  - 存在唯一的一个被称为“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均只有一个“直接前驱”；
  - 除最后一个之外，集合中的每个数据元素均只有一个“直接后继”；

- 常用的线性结构：线性表、字符串、栈、队列等

线性表的基本概念

- 线性表简称表，是零个或多个（具有相同特性的）数据元素的有穷序列，通常可以表示成（k_0，k_1，…，k_{n-1}）。
  - 相同特性：所有元素属于同一数据类型
  - 有穷：数据元素个数是有限的
  - 序列：数据元素由逻辑序号唯一确定
- 索引（下标）：i称为元素k_i的“索引”或“下标”
- 表的长度：线性表中所含元素的个数n。
- 空表：长度为零的线性表(n=0)。

线性表的表示

- 线性表的逻辑结构可以用二元组$L=<K,R>$来表示，其中$K=\{k_0,k_1,\ \ldots,k_{n-1}\}$，$R=\{<k_i,k_{i+1}>|0\le i\le n-2\}$
- 第一个元素：当长度 n ≥1时，k_0称为第一个元素
- 最后一个元素：当长度 n ≥1时，k_{n-1}称为最后一个元素
- 前驱：称k_i（$0\le i\le n-2$）是k_{i+1}的“前驱”
- 后继：称k_{i+1} （$0\le i\le n-2$）是k_i的“后继”

- 除表中第一个元素k_0与最后一个元素k_{n-1}之外，其他每个元素k_i有且仅有一个前驱和一个后继

线性表的例子

- 线性表的均匀性：虽然不同线性表的数据元素可以是各种各样的，但对于同一线性表的各数据元素必定具有相同的数据类型和长度。

编号	姓名	性别	年龄	月收入
1	李泉	男	51	980
2	王怡	女	47	945
3	张三	男	35	870
4	马小丁	男	27	840
…	…	…	…	…

（A，B，C，D，…，Y，Z）

（1，2，3，4，...，9，10）

根据各城市的GDP值，给出全国GDP的总排名。

某年南方GDP排名前二十如下：

排名	城市	GDP 及增长
01	上海	10297+12.0%
02	广州	6068+14.7%
03	深圳	5684+15.0%
04	苏州	4820+15.5%
05	重庆	3486+12.2%
06	杭州	3441+14.3%
07	无锡	3360+15.0%
08	佛山	2927+19.3%
09	宁波	2864+13.4%
10	南京	2774+15.1%
11	成都	2750+13.8%
12	东莞	2624+19.1%
13	武汉	2590+14.8%
14	泉州	1901+15.0%
15	温州	1834+13.3%
16	长沙	1791+14.8%
17	南通	1758+15.7%
18	绍兴	1678+13.2%
19	福州	1660+12.2%
20	常州	1560+15.0%

同年北方GDP排名前二十如下：

排名	城市	GDP 及增长
01	北京	7720+12.0%
02	天津	4338+14.4%
03	青岛	3207+15.7%
04	大连	2568+16.4%
05	沈阳	2483+16.5%
06	烟台	2402+17.0%
07	唐山	2362+14.8%
08	济南	2185+15.7%
09	哈尔滨	2094+13.5%
10	石家庄	2064+13.2%
11	郑州	2002+15.7%
12	长春	1934+14.5%
13	潍坊	1721+16.5%
14	淄博	1645+15.8%
15	大庆	1618+11.4%
16	济宁	1456+16.5%
17	西安	1450+12.8%
18	东营	1450+17.0%
19	临沂	1405+16.3%
20	威海	1369+15.9%

中国南方主要河流径流量

排名	河流	流域面积(万平方千米)	长度(千米)	径流量(亿立方米)
1	长江	180.7	6300	9794
2	珠江	45.3	2197	3492
3	雅鲁藏布江	24.6	1940	1167
4	澜沧江	16.5	1612	743
5	怒江	14.3	1540	701
6	闽江	6.1	577	624
7	钱塘江	5.4	494	468
8	韩江	3.4	325	297
9	瓯江	1.8	338	194
10	元江	3.5	772	129

中国北方主要河流径流量

排名	河流	流域面积(万平方千米)	长度(千米)	径流量(亿立方米)
1	黑龙江	162.0	3420	2709
2	黄河	75.2	5464	575
3	淮河	18.6	1000	351
4	鸭绿江	6.3	773	328
5	海河	26.5	1090	226
6	伊犁河	5.7	375	118
7	额尔齐斯河	5.1	442	108
8	辽河	16.4	1430	95

1978-1987年世界主要区域电视机生产的变动情况

1978-1987年亚洲电视机生产的变动情况

国家	1978年产量(万台)	1987年产量(万台)
日本	1312	1478
韩国	483	1466
中国	52	1934
马来西亚	15	124
新加坡	73	212

1978-1987年欧洲及美国电视机生产的变动情况

国家	1978年产量(万台)	1987年产量(万台)
西德	439	354
英国	242	302
意大利	217	223
法国	210	218
美国	931	1131

线性表的基本运算

- 创建
- 插入
- 删除
- 查找
- 取值
- 判空

- 创建并返回一个空线性表；
- 插入一个元素；
  - 位置p之前或之后插入元素x，并返回插入成功与否的标志
- 根据下标或者值，删除某个元素；
  - 删除一个值为x的元素或删除位置为p的元素，并返回删除成功与否的标志
- 查找某个特定元素；
  - 返回一个值为x的元素的下标
  - 返回下标为某值的数据元素的值
- 判别一个线性表是否为空表。

线性表的抽象数据类型　　三元组表示（D, R, P）

ADT LIST {数据对象: D={ a| $a_i \in elemset, i = 1,2,\ldots,n; n \geq 0$ }

/*　定义线性表，具有n个数据元素的有限序列　*/

数据关系: R= $\{< a_{i-1}, a_i >, a_{i-1}, a_i \in D, i = 1,2,\ldots,n\}$

/* 说明线性表数据元素之间的相互关系　*/

基本运算:

SetNull(L) 初始条件: 线性表L存在

运算结果: 将线性表L置空;

Length(L) 初始条件: 线性表L存在

运算结果: 返回线性表L的数据元素个数（表长）;

Get (L, i)

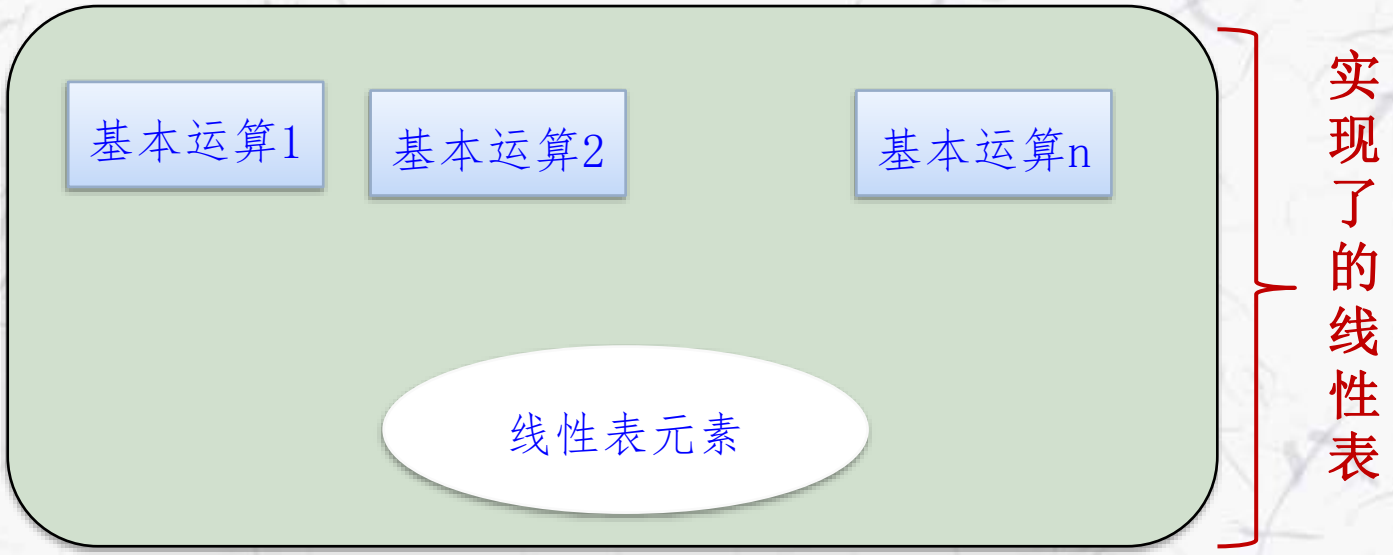
Locate(L, x)

Insert(L, x, i)

Delete(L, i)

} ADT　LIST

线性表的作用



- 程序员可以直接用它来存放数据——作为存放数据的容器
- 程序员可以直接使用它的基本运算——完成更复杂的功能

线性表的特点和存储结构

- 线性表特点：关系简单、操作灵活，其长度可以增长、缩短
- 存储结构：
  - 顺序存储
  - 链式存储

内容提要

- 数据类型和抽象数据类型
- 线性表的概念（ADT）
- 线性表的顺序表示和实现（顺序表）
- 线性表的链式表示和实现（链表）
- 线性表的应用

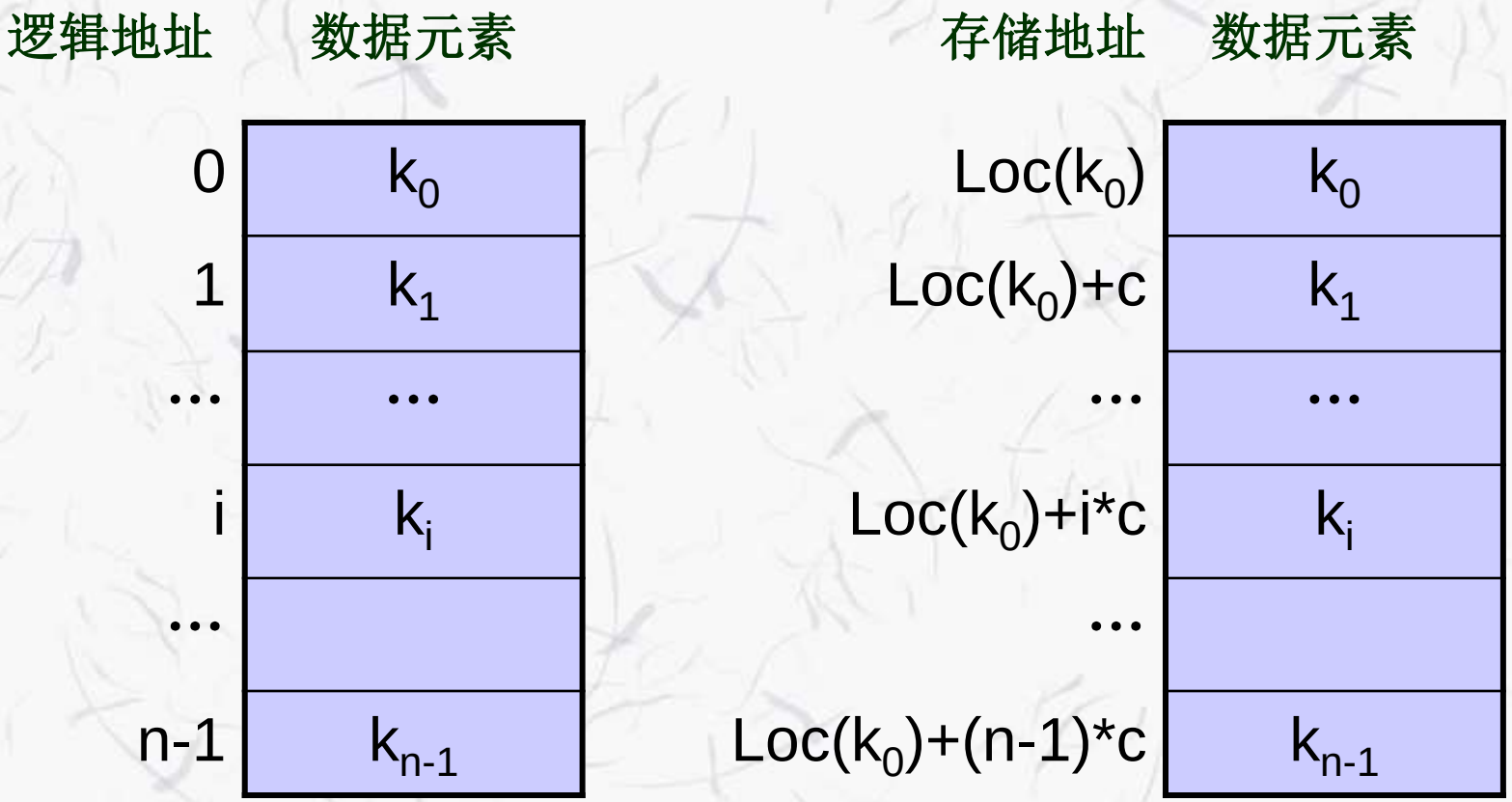
线性表的顺序表示和实现

- 线性表的顺序表示 
- 顺序表基本运算的实现
- 顺序表的应用

线性表的顺序表示

- 定义：将线性表中的元素一个接一个地存储在一片地址连续的存储单元中。
- 逻辑关系表达：以元素在计算机内存中的“物理位置相邻”来表示线性表中数据元素之间的逻辑关系，如下所示：
  - $Locate(a_{i+1}) = Locate(a_i) + sizeof(DataType)$
  - $Locate(a_i) = Locate(a_0) + sizeof(DataType)*i$
- 只要确定了首地址，线性表中任意数据元素都可以随机存取。因此，顺序表为“随机存取（random access，直接存取）的存储结构”。
- 顺序存取（sequential access）

线性表的顺序存储结构示意图



顺序存储（ Sequential Storage ）

用typedef定义类型

- 可以用typedef声明新的类型名来代替已有的类型名
- typedef int INTEGER;
  - int i,j; 与 INTERGER i,j; 等价
- typedef int DataType ;
  - DateType *element ; 与int *element ; 等价

- 习惯上把用typedef定义的类型名用大写字母表示

顺序表的定义

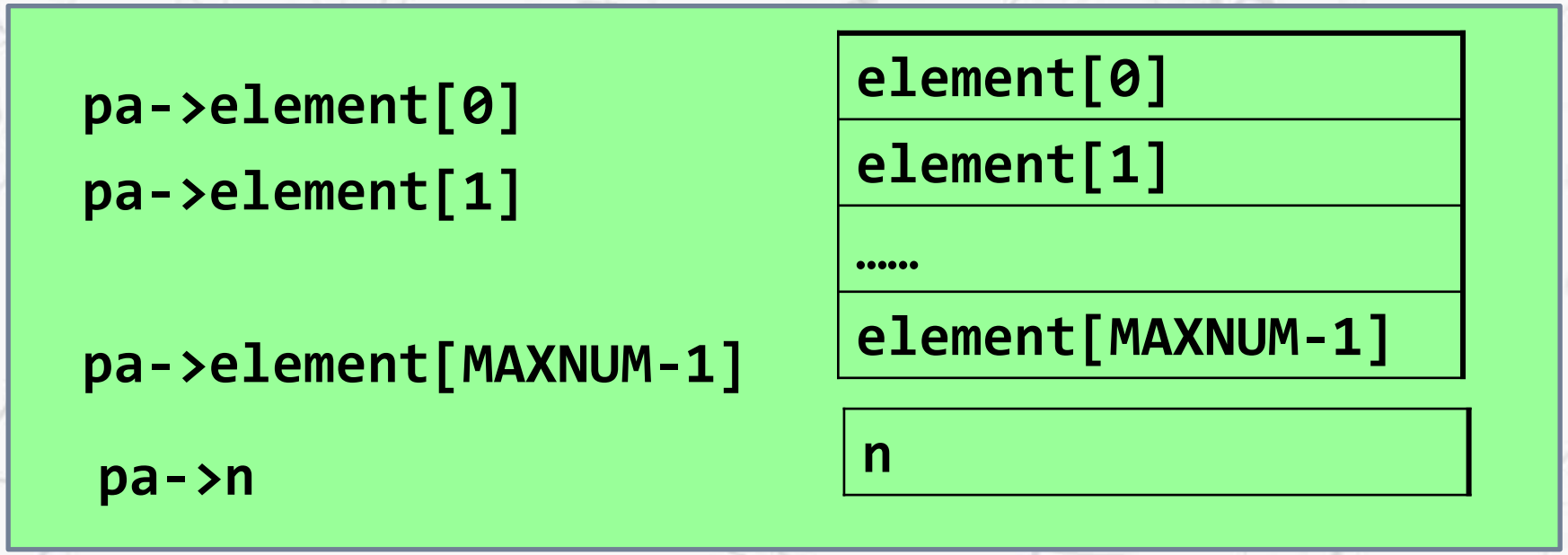
- 在C语言中可以用以下方式定义一个顺序表：

```
#define   MAXNUM   100/*最多允许的数据元素个数*/
int n;    /*顺序表中当前元素的个数，n<=MAXNUM*/
DataType  * element;
DataType  element[MAXNUM];
```

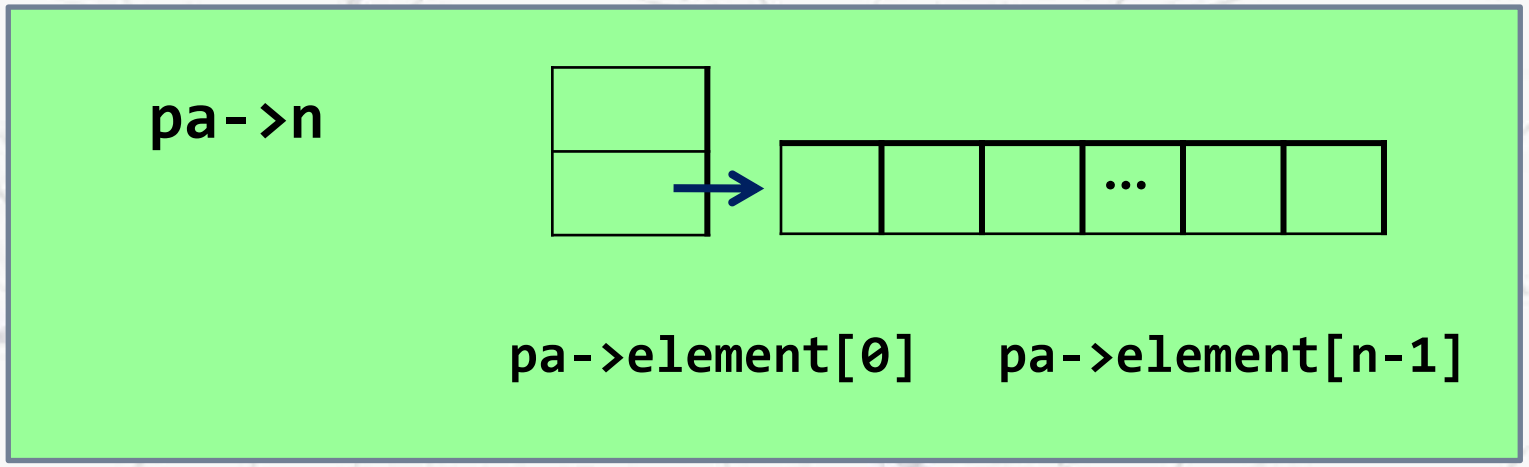
- 定义的缺陷：没有反映出element和n的内在联系，即没有指出n是顺序表element本身的属性。在这个定义中，n和element完全处于独立平等的地位。

- 为此，引进SeqList结构，它的定义为：

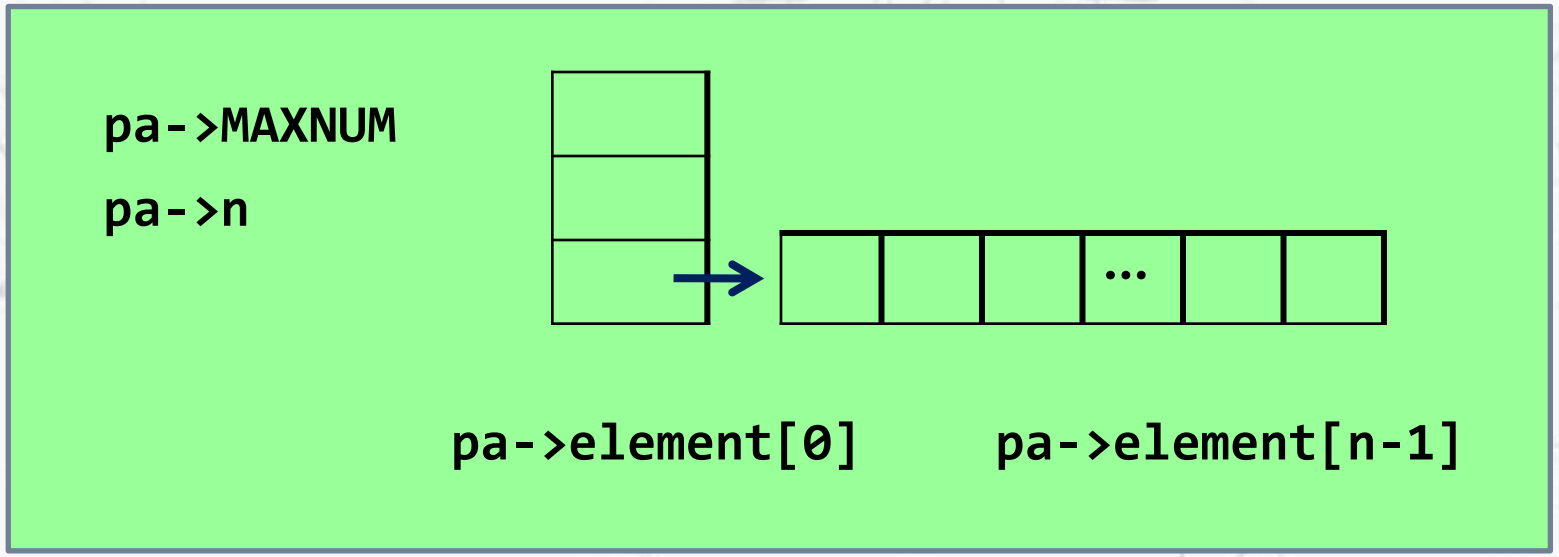
```
struct  SeqList
{  DataType element[MAXNUM]; /*存放线性表中的元素*/
   int n;  /* n < MAXNUM  , 顺序表中当前元素的个数*/
};                          剩余空间：n到MAXNUM-1
typedef struct SeqList *PSeqList;  /*实际应用中，为了使用方便，通常定义一个struct SeqList类型的指针类型*/
PSeqList pa;
```



```
struct  SeqList
{  int n;  /* n <= MAXNUM,顺序表中当前元素的个数 */
   DataType  *element;
    /* element[0],…,element[n-1]存放线性表中的元素 */
};
typedef struct SeqList *PSeqList;
PSeqList pa;
```



```
struct  SeqList
{   int MAXNUM;        /*顺序表中最大元素的个数*/
    int  n; /* n <= MAXNUM  , 顺序表中当前元素的个数*/
    DataType  *element;
          /* element[0],…,element[n-1]存放线性表中的元素 */
};
typedef struct SeqList *PSeqList;
PSeqList pa;
```



线性表的顺序表示和实现

- 线性表的顺序表示
- 顺序表基本运算的实现
- 顺序表的应用

顺序表的基本运算-1

- 创建
- 插入
- 删除
- 查找
- 取值
- 判空

- **PSeqlist createNullList_seq(int m)**
  - 创建空顺序表。
- **int isNullList_seq(PSeqList palist)**
  - 判别palist所指顺序表是否为空表。若为空则返回1，否则返回0
- **int locate_seq(PSeqList palist, DataType x)**
  - 在palist所指顺序表中寻找第一个值为x的元素的下标，若palist中不存在值为x的元素，则返回一个特殊的下标值-1
- **DataType retrieve_seq(PSeqList palist, int p)**
  - 在palist所指顺序表中，寻找下标为 p（0≤p＜ palist ->n）的元素，并将该元素的值作为函数值返回，若palist中无下标为p的元素，则返回一个特殊的值

顺序表的基本运算-2

- 创建
- 插入
- 删除
- 查找
- 取值
- 判空

- `int insertPre_seq( PSeqList palist, int p, DataType x)`
  - 在palist所指顺序表中下标为 p的位置上插入一个值为x的元素，并返回插入成功与否的标志。此运算在0≤p≤ palist->n时有意义

插入：（k_0, k_1, …, k_p, k_{p+1},…, k_{n-1}）　　n

（k_0, k_1, …, k_{p-1}, x, k_p,…, k_{n-1}）　　n+1

- `int insertPost_seq( PSeqList palist, int p, DataType x)`
  - 在palist所指顺序表中下标为p的位置之后插入一个值为x的元素，并返回插入成功与否的标志。此运算在0≤p ＜palist->n时有意义

顺序表的基本运算-3

- 创建
- 插入
- 删除
- 查找
- 取值
- 判空

- **int deleteP_seq(PSeqList palist, int p)**
  - 在palist所指顺序表中删除下标为p的元素，并返回删除成功与否的标志。此运算在0≤p＜ palist ->n时有意义

删除：（k_0, k_1, …, k_{p-1}, k_p, k_{p+1},…, k_{n-1}） n

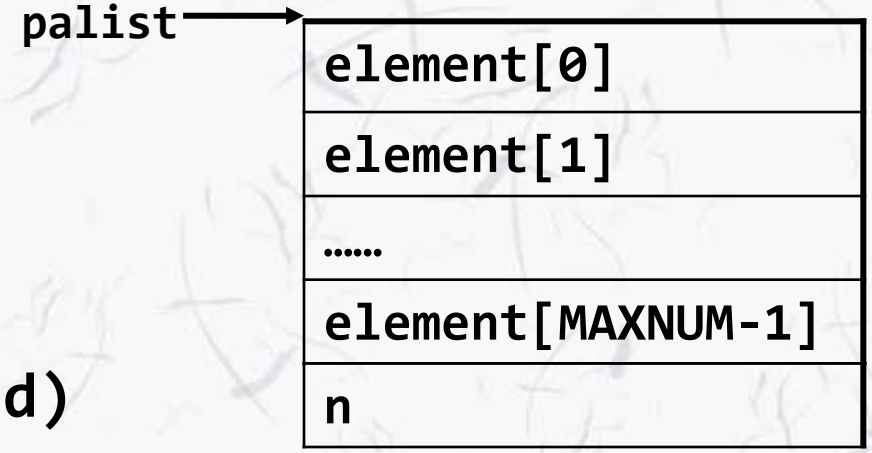
（k_0, k_1, …, k_{p-1}, k_{p+1},…, k_{n-1}） n-1

- **int deleteV_seq(PSeqList palist, DataType X)**
  - 在palist所指顺序表中删除值为x的元素，并返回删除成功与否的标志。

顺序表的基本运算

- 创建 
- 判空
- 插入
- 删除
- 取值
- 查找

建立空线性表-old



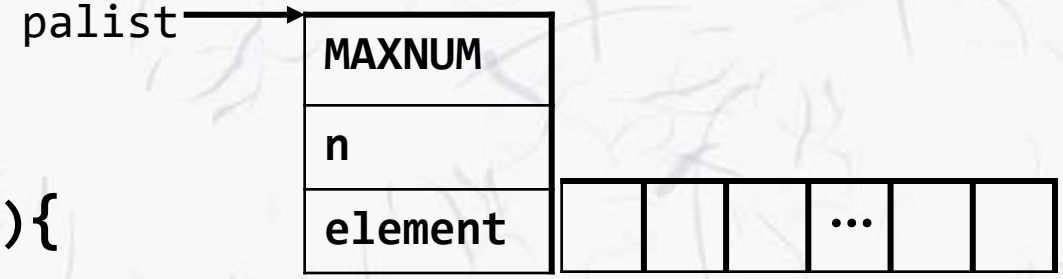
- 函数名： createNullList_seq
- 功能 ： 建立空线性表
- 程序：

```
PSeqList createNullList_seq (void)
{  PSeqList palist;
   palist =(PSeqList)malloc(sizeof(struct SeqList));
   if(palist ! = NULL) palist ->n = 0;
   else printf("out of space!\n");
   return (palist);
}
```

```
struct  SeqList
{  DataType  element[MAXNUM];
   int   n;
};
typedef struct SeqList *PSeqList;
```

建立空线性表-new

- 函数名：createNullList_seq
- 功能 ： 建立空线性表
- 程序




```
PSeqList createNullList_seq(int m){
PSeqList palist;
palist =(PSeqList ) malloc (sizeof(struct SeqList ) );
if(palist! = NULL){
  palist->element=(DataType *)malloc(sizeof(DataType)*m);
  if(palist->element){
    palist->MAXNUM=m;
    palist -> n = 0;  /*空表长度为0*/
    return (palist);}
    else free palist;
 }
 printf("out of space!\n");
              /*存储分配失败*/
 return NULL;
}
```

```
struct  SeqList
{  int MAXNUM;
   int n;
   DataType  *element;
};
typedef struct SeqList *PSeqList;
```

顺序表的基本运算

- 创建
- 判空
- 插入
- 删除
- 取值
- 查找

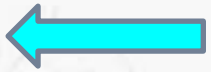
判断线性表是否为空

- 函数名:isNullList_seq
- 功能:判断线性表是否为空
- 程序:

```
int isNullList_seq(PSeqList palist){
    return(palist->n==0);
}
```

顺序表的基本运算

- 创建
- 判空
- 插入
- 删除
- 取值
- 查找



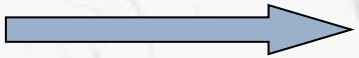
插入一个值为x的元素

- 函数名： insertPre_seq
- 功能 :在palist所指顺序表中下标为p的位置上插入一个值为x的元素，并返回插入成功与否的标志。此运算在0≤p≤palist->n时有意义
- 程序：

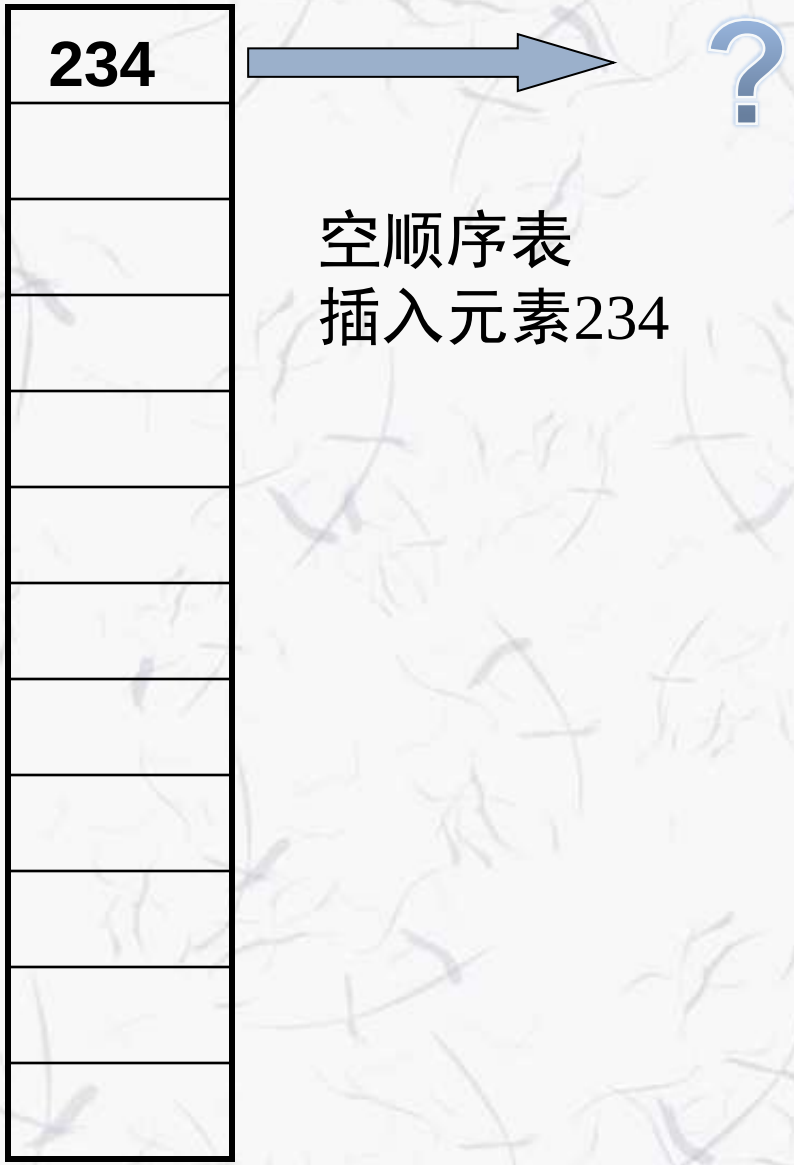
```
int insert_seq(PSeqList palist,int p, DataType x)
```

12
111
22
8888
333
123
56
99
23
46
78
456

下标7前
插入234

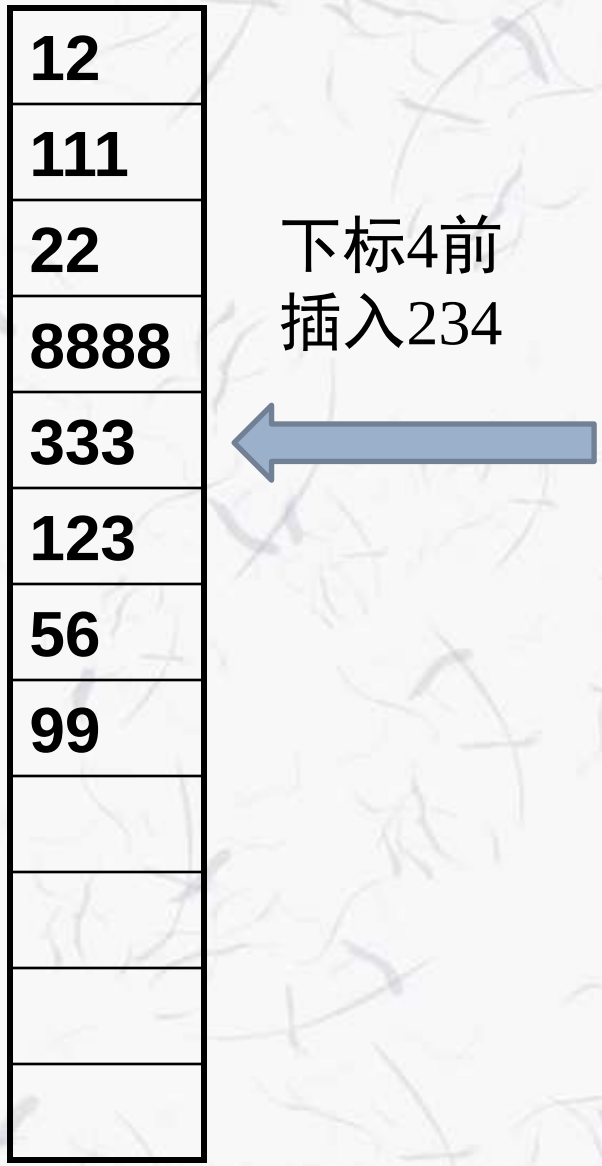


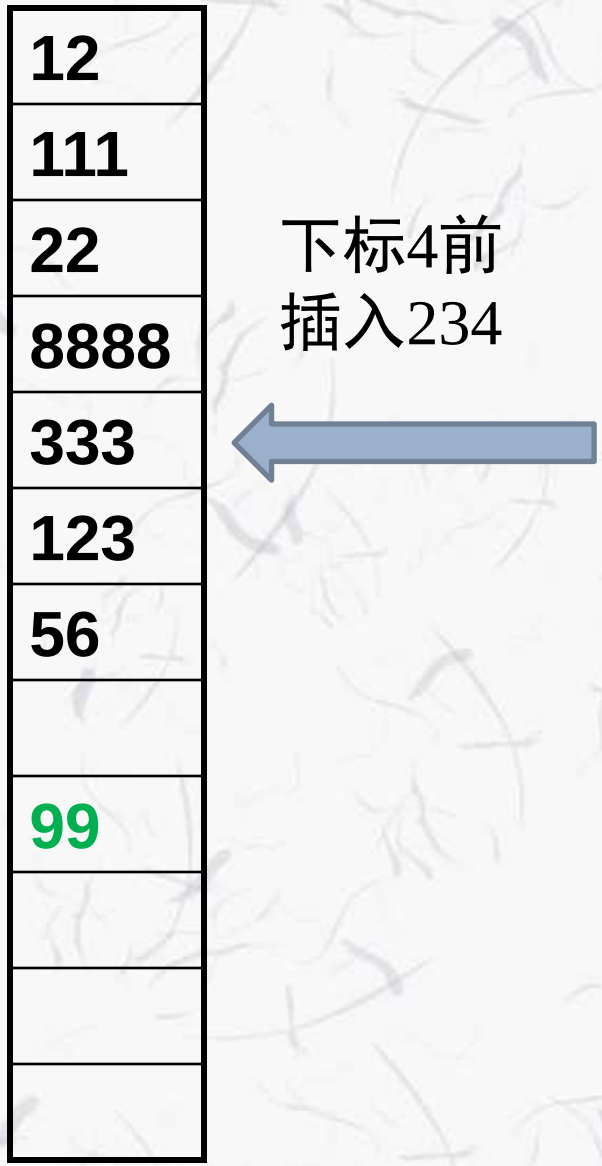
? No! Overflow!

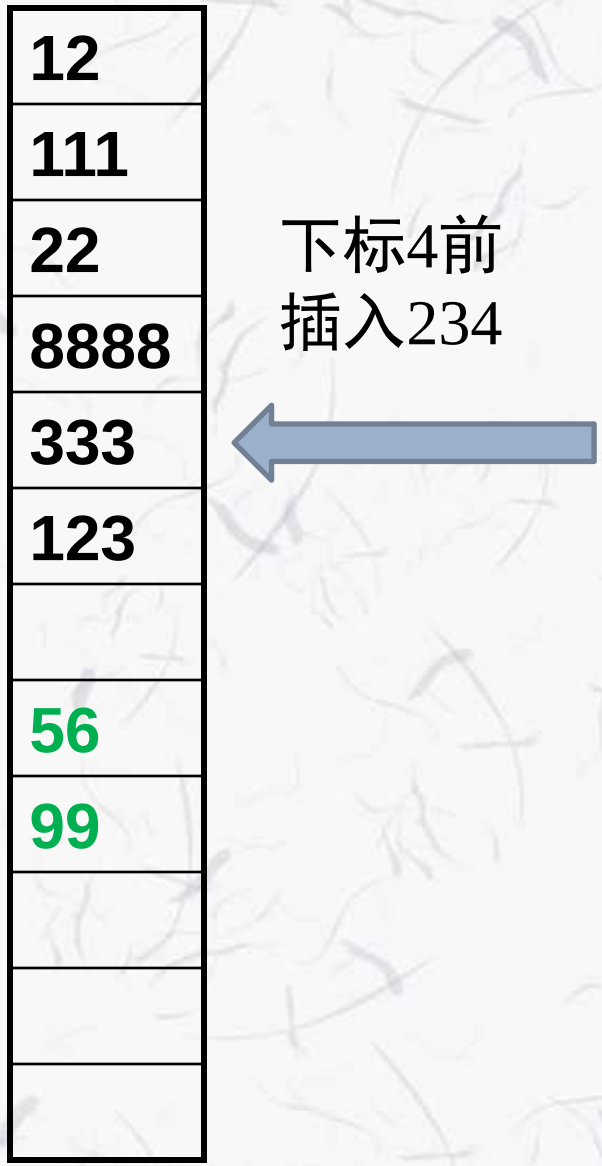


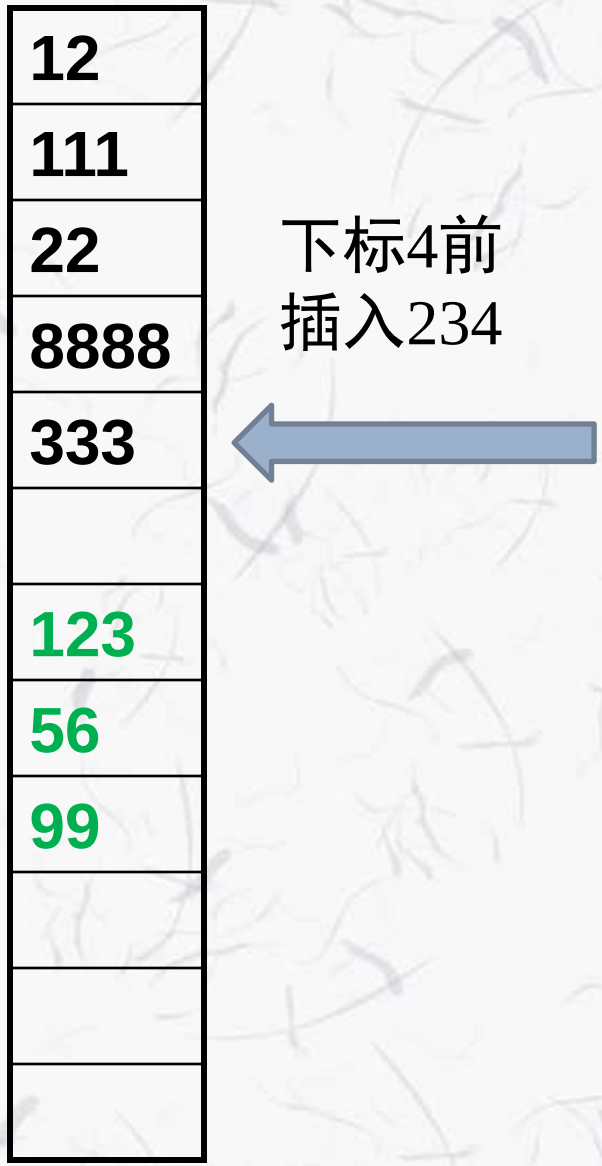
空顺序表
插入元素234


12
111
22
8888
333
123
56
99
下标4前
插入234
?
12
111
22
8888
234
333
123
56
99

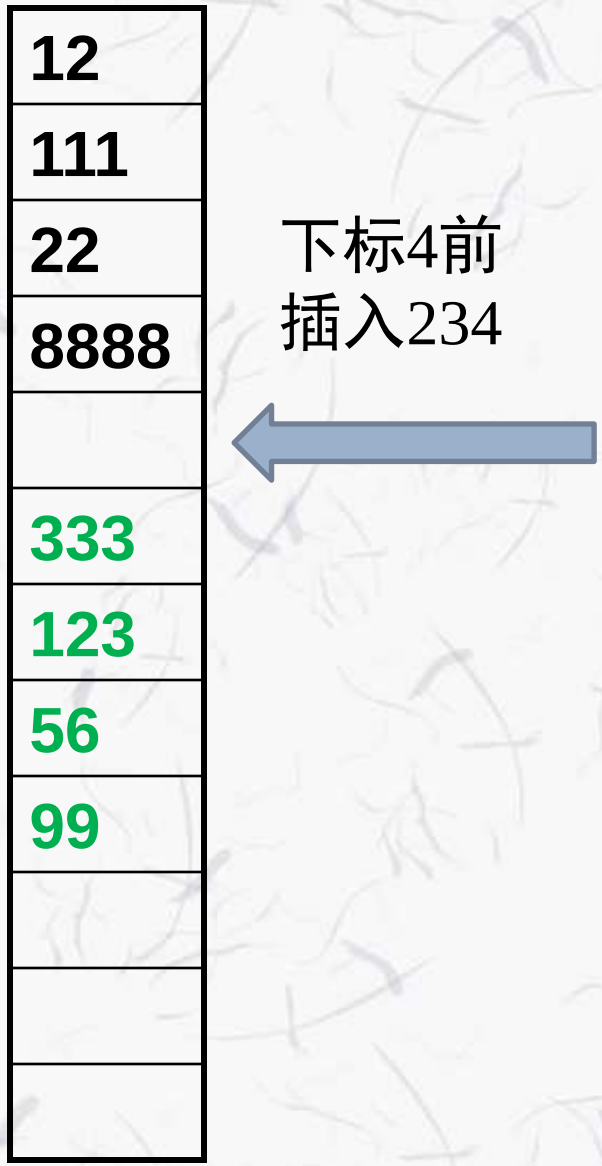

12
111
22
8888
333
123
56
99
下标4前
插入234

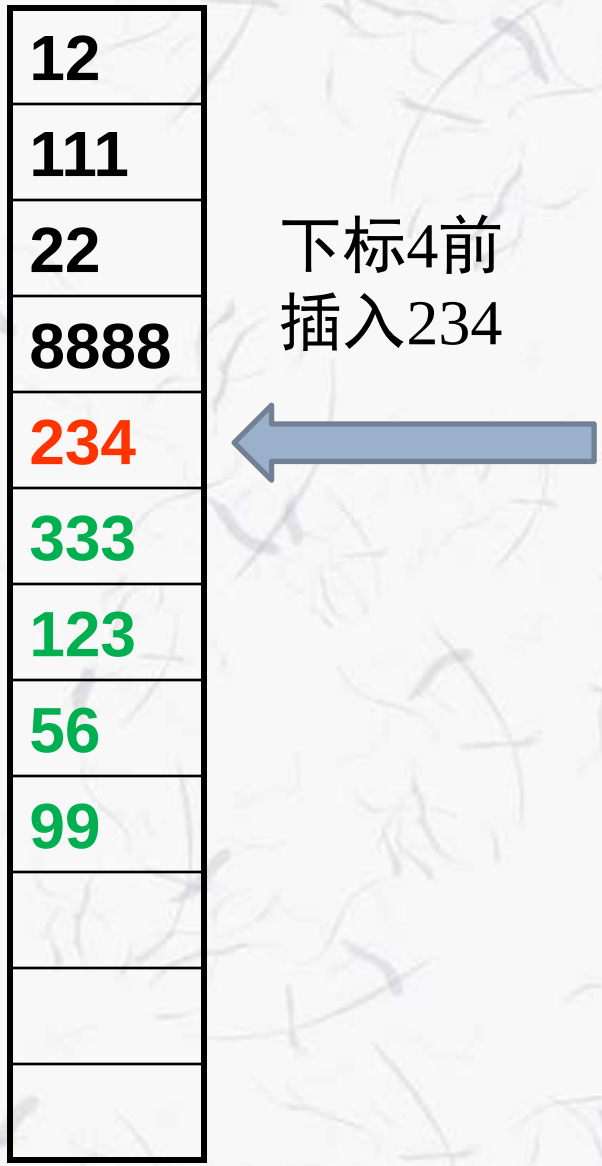

12
111
22
8888
333
123
56
99
下标4前
插入234


12
111
22
8888
333
123
56
99
下标4前
插入234

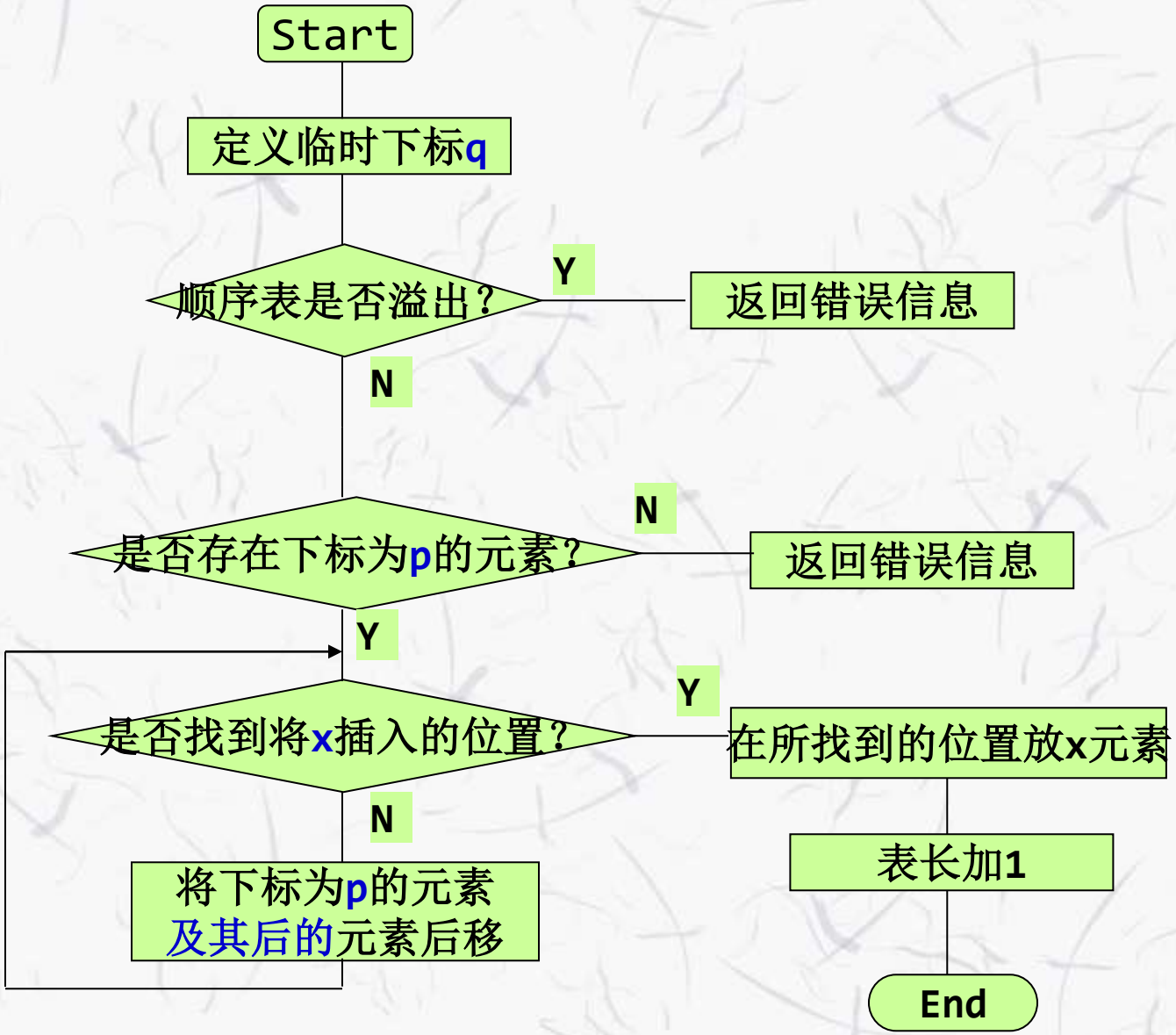


下标4前
插入234


12
111
22
8888
333
123
56
99
下标4前
插入234


12
111
22
8888
234
333
123
56
99
下标4前
插入234

在palist所指顺序表中下标为 p的位置上插入元素x



```
int insertPre_seq( PSeqList palist, int p, DataType x )
{  int q;
   if ( palist->n >=palist-> MAXNUM )   /* 溢出 */
   {   printf("Overflow!\n");return (0);
   }
   if(isNullList_seq(palist)){    /*空顺序表插入*/
      palist->element[0]=x;
      palist->n=1;return (1);
   }
   if ( p<0 || p >palist->n )  /*不存在下标为 p的元素 */
   {   printf("not exist! \n"); return (0); }
   /* 将 p 及以后的元素后移一个位置 */
   for(q=palist->n - 1; q>=p; q--)
      palist->element[q+1] = palist->element[q];
   palist->element[p] = x; /* 在p下标位置上放元素x */
   palist->n = palist->n + 1;    /* 元素个数加 1 */
   return (1);
}
```

插入算法的时间复杂性

- 设p_i是在下标为i的位置插入一个元素的概率，则在长度为n的线性结构中在下标为i的元素插入一个元素需移动元素的次数为(n-i),则插入时的平均移动次数为：

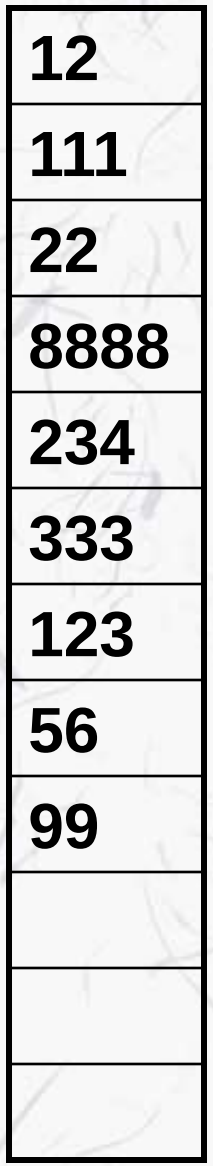
$$E_{is}=\sum_{i=0}^{n}p_i(n-i)$$

设 $p_i=\frac{1}{n+1}$ 则有

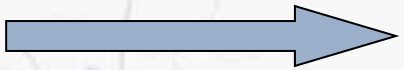
$$E_{is}=\frac{1}{n+1}\sum_{i=0}^{n}(n-i)=\frac{n}{2}$$

顺序表的基本运算

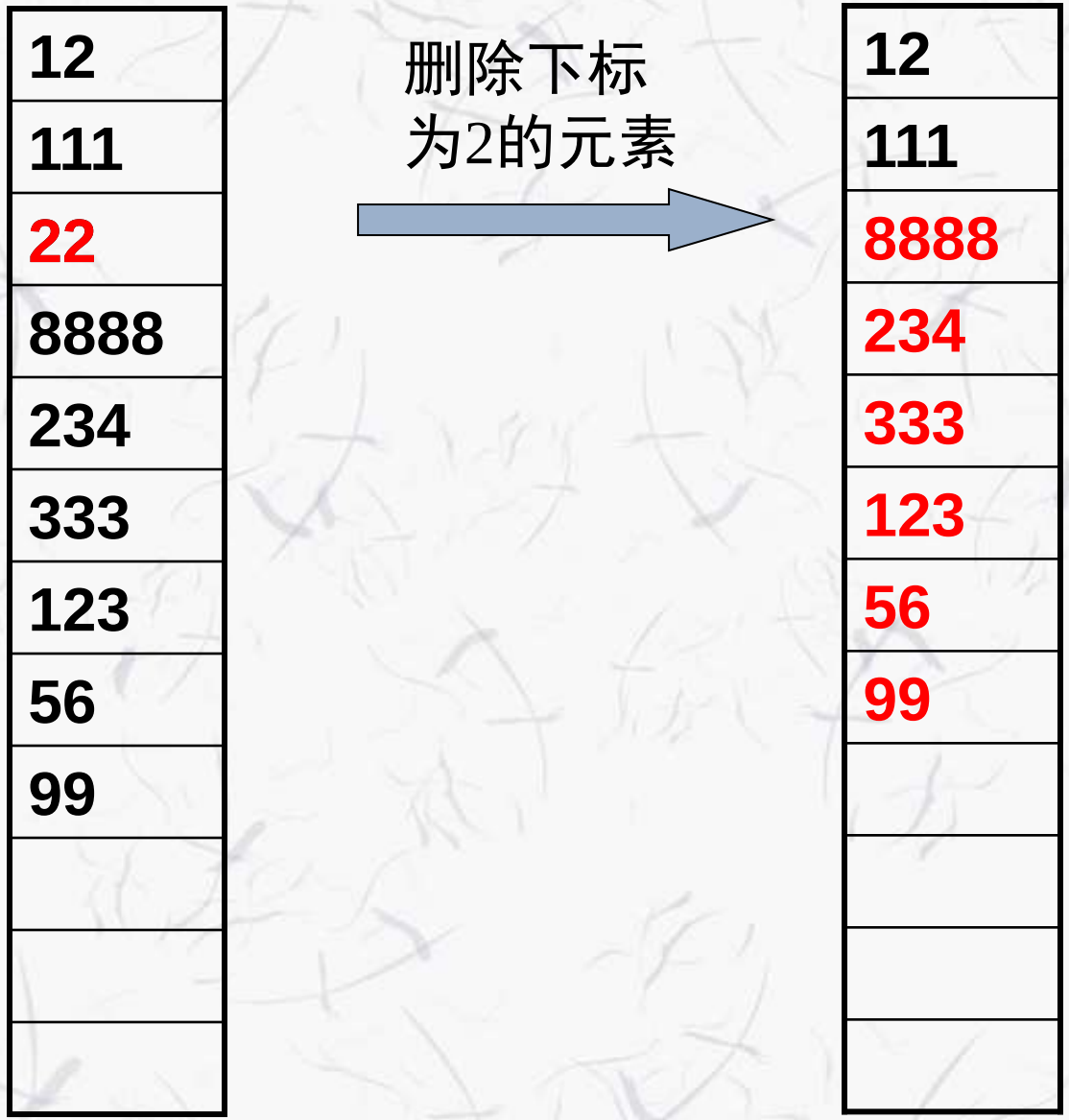
- 创建
- 判空
- 插入
- 删除
- 取值
- 查找

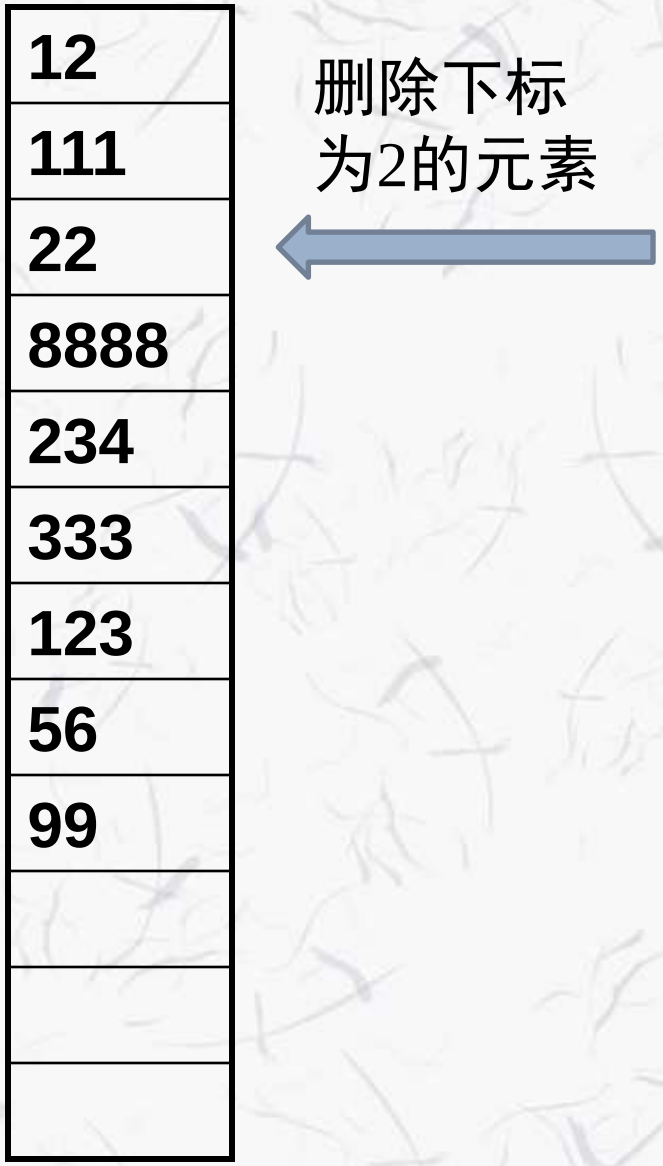


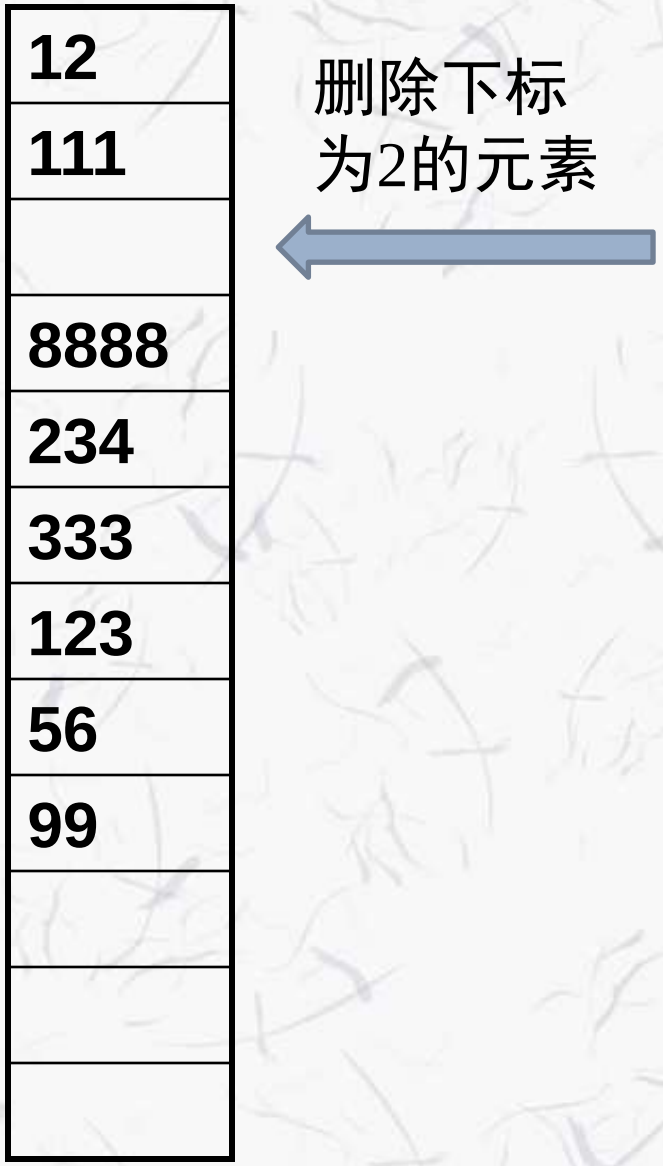
删除下标
为10的元素

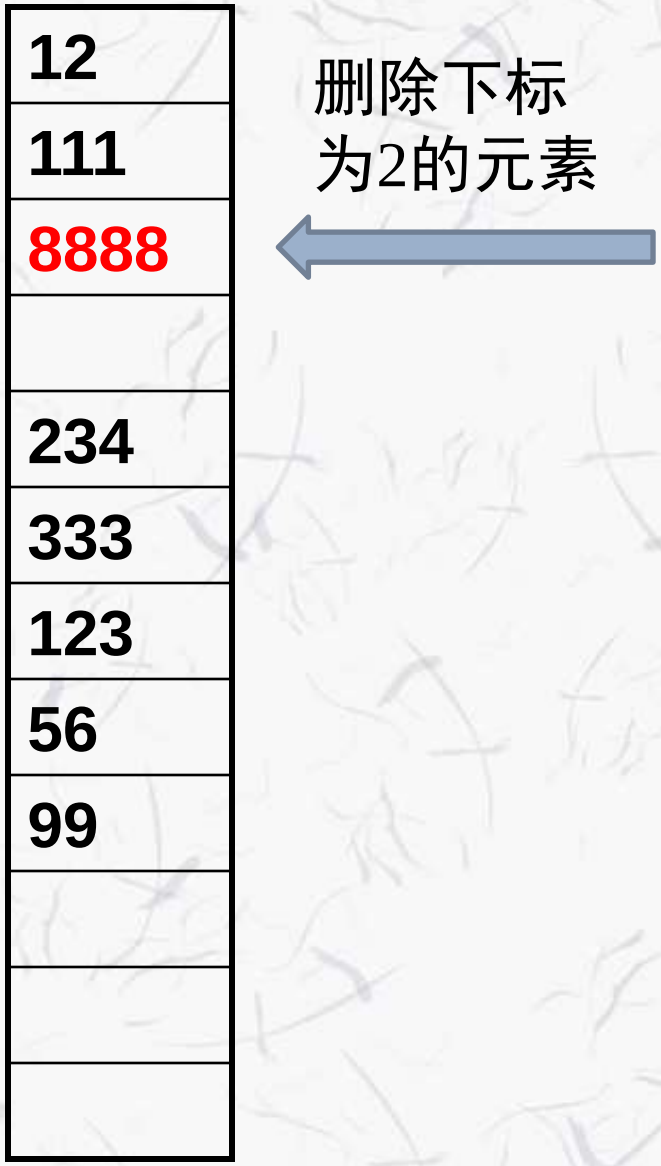


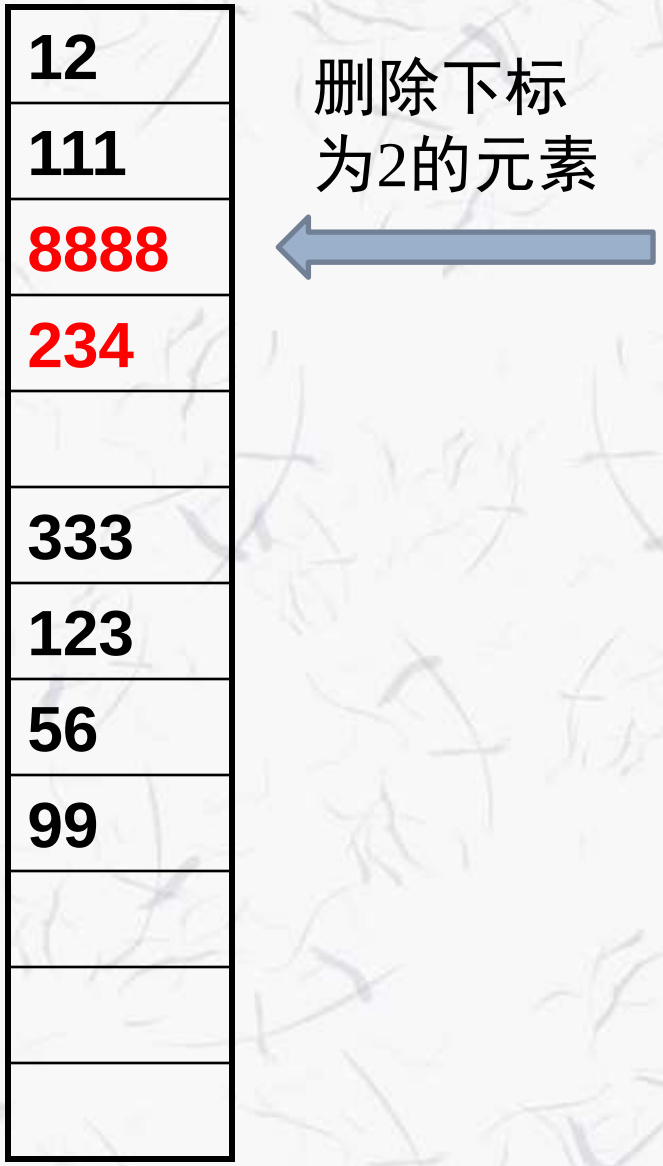
?No! not exist!

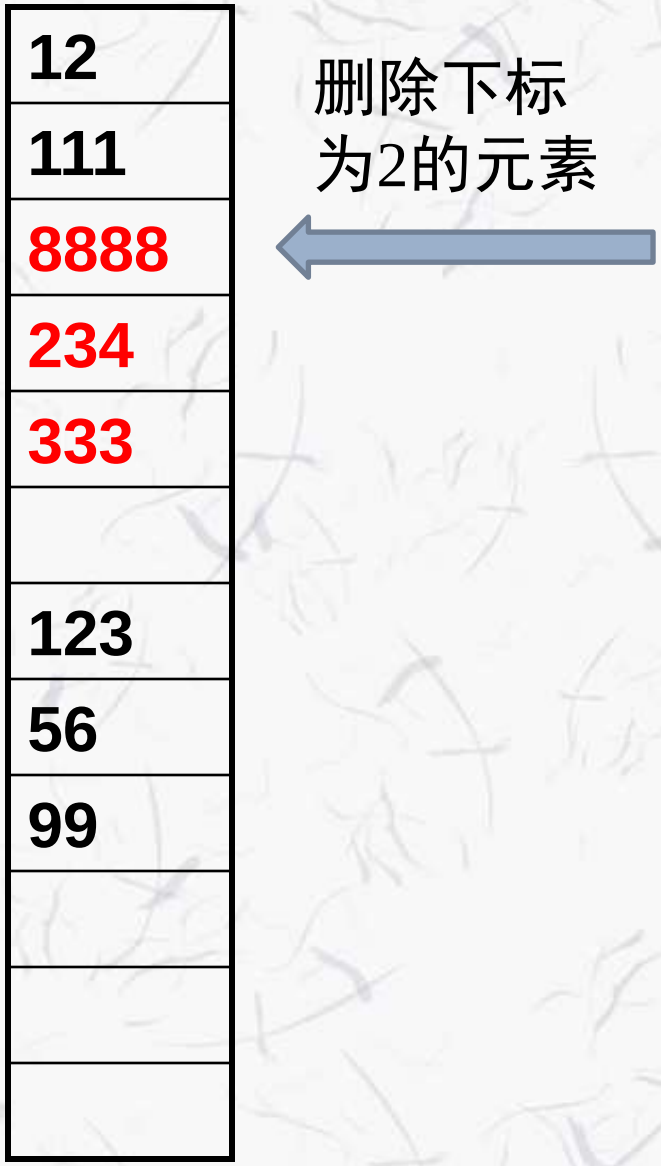

12
111
22
8888
234
333
123
56
99
删除下标
为2的元素
12
111
8888
234
333
123
56
99

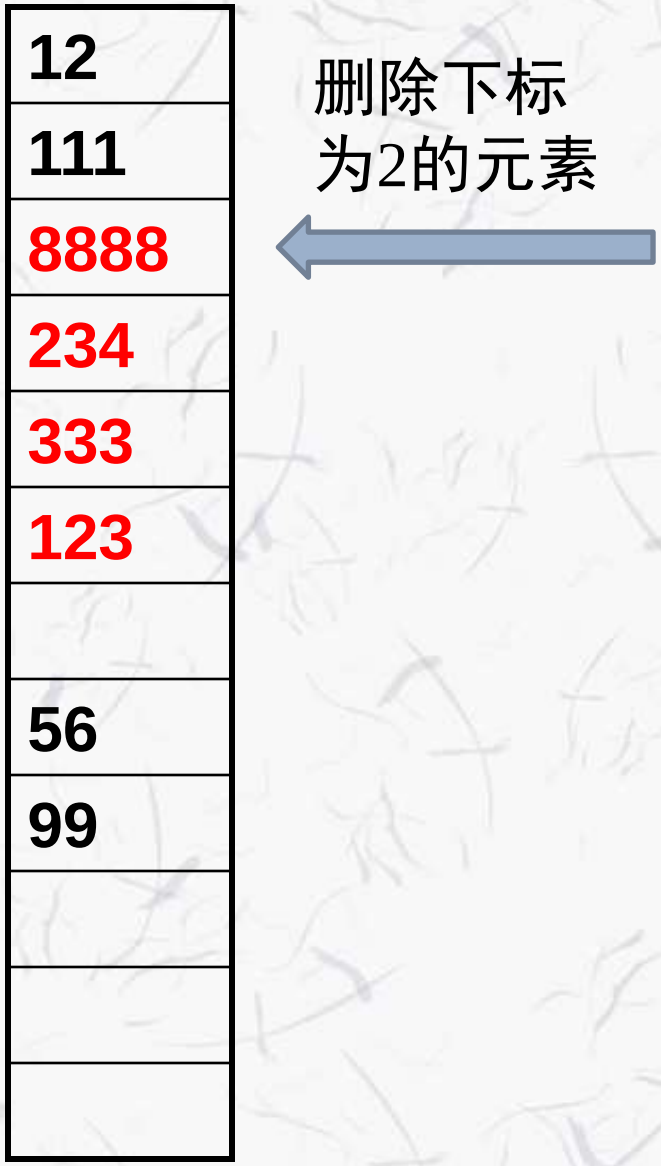

12
111
22
8888
234
333
123
56
99
删除下标
为2的元素

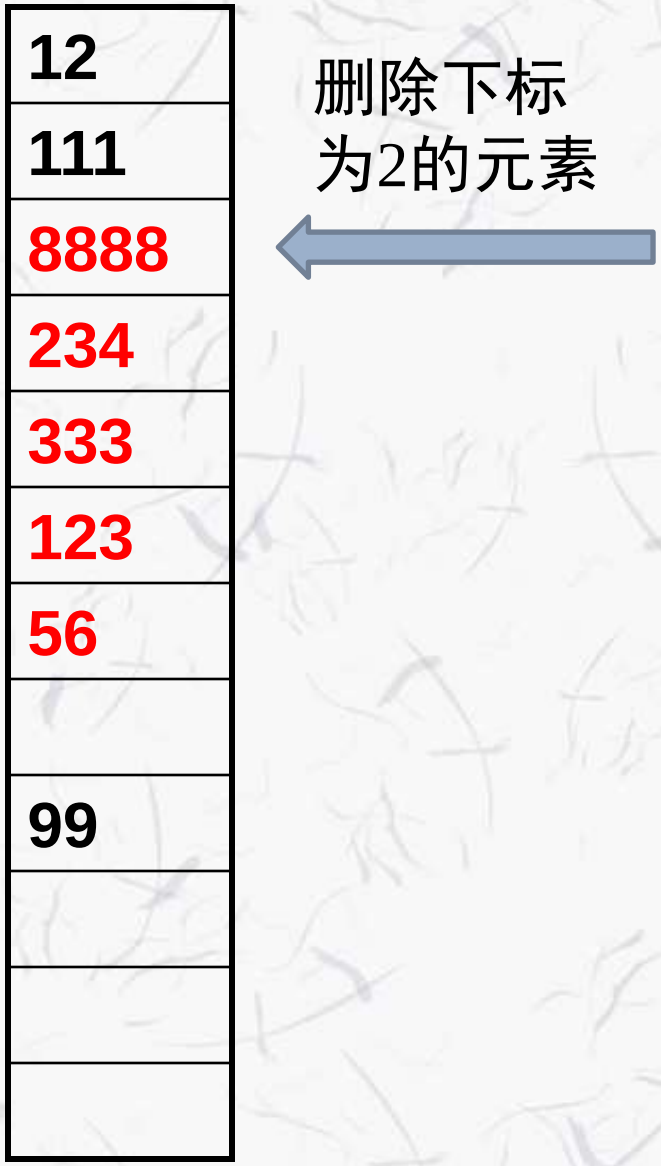

12
111
8888
234
333
123
56
99
删除下标
为2的元素

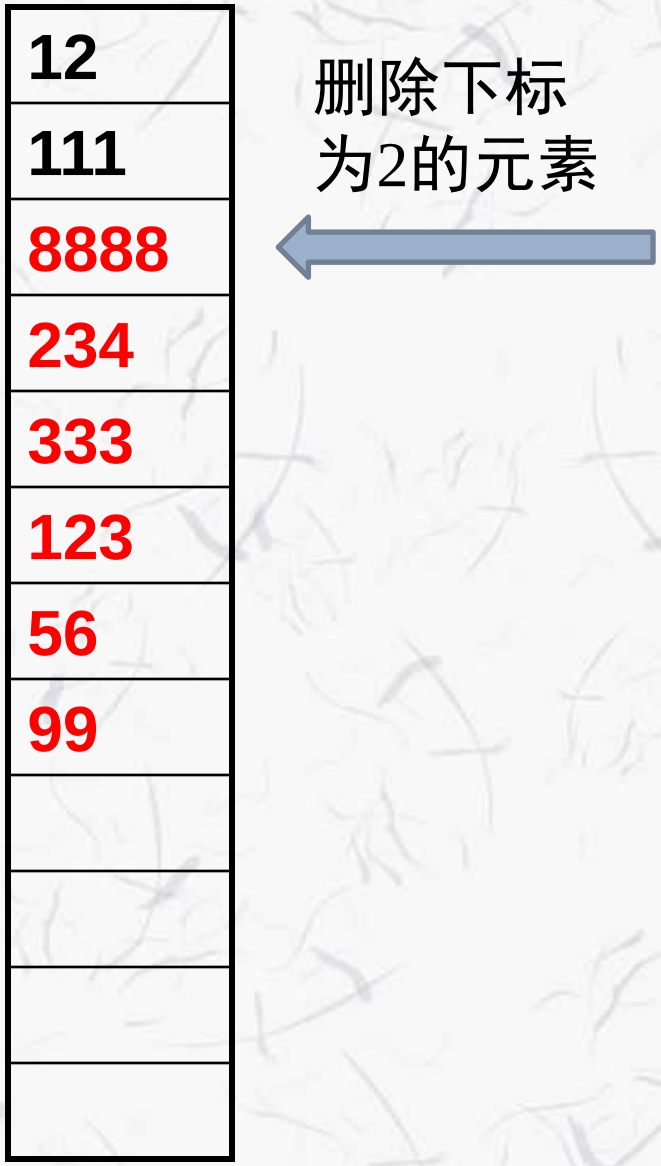

12
111
8888
234
333
123
56
99
删除下标
为2的元素


12
111
8888
234
333
123
56
99
删除下标
为2的元素


12
111
8888
234
333
123
56
99
删除下标
为2的元素


12
111
8888
234
333
123
56
99
删除下标
为2的元素


12
111
8888
234
333
123
56
99
删除下标
为2的元素


12
111
8888
234
333
123
56
99
删除下标
为2的元素

删除下标为 p 的元素

- 函数名：deleteP_seq
- 功能:在palist所指顺序表中删除下标为 p的元素，并返回删除成功与否的标志。

  此运算在0≤p＜ palist ->n时有意义。
- 程序：

```
int deleteP_seq( PSeqList palist, int p )
```

```
int  deleteP_seq( PSeqList palist, int p )
/* 在palist所指顺序表中删除下标为 p 的元素 */
{  int q;
   if (p<0||p>palist->n-1) /*不存在下标为 p 的元素 */
   {  printf("not exist!\n ");
      return(0);
   }

   /*将 p 以后的元素前移一个位置 */
   for(q=p; q<palist->n-1; q++)
       palist->element[q] = palist->element[q+1];

  palist->n = palist->n - 1;      /*元素个数减 1 */
  return (1);
};
```

删除算法的时间复杂性

设P_i'是在下标为i的位置上删除元素的概率，则在长度为n的线性结构中把下标为i（第i+1个）位置上的元素删除需要移动n-i-1个元素 ：

$$M_d = \sum_{i=0}^{n-1}(n-i-1)P_i'$$

设 $P_i' = \frac{1}{n}$ 则有

$$M_d = \frac{1}{n}\sum_{i=0}^{n-1}(n-i-1) = \frac{n-1}{2}$$

顺序表的基本运算

- 创建
- 判空
- 插入
- 删除
- 取值
- 查找

取下标为 p 的元素的值

- 函数名： retrieve_seq
- 功能：在palist所指顺序表中，寻找下标为 p 的元素，并将该元素的值作为函数值返回，若找不到则返回一个特殊的值。
- 程序：

```
DataType retrieve_seq(PSeqList palist,int p)
{  if(p>=0 && p<palist->n)
      return (palist->element[p]);
   printf("not exist.\n")
   return(SPECIAL);
}
```

顺序表的基本运算

- 创建
- 判空
- 插入
- 删除
- 取值
- 查找

查找值为x的元素

- 函数名： locate_seq
- 功能 :求x在palist所指顺序表中的下标，若不存在值为x的元素，返回特殊值-1
- 程序：

```
int locate_seq (PSeqList palist,Datatype x )
{  int q;
   for (q=0;q<palist->n;q++)
      if(palsit->element[q]==x)
         return (q);
   return (-1);
}
```

定位运算的时间复杂性

- 定位运算：通过比较完成，完成一次定位也平均需要n/2次比较，时间复杂度为O(n)；
- 对于有序表的改进：采用折半查找方法，$O(\log_2 n)$

删除一个值为x的元素

- 函数：

  deleteV_seq(PSeqList palist,DataType x)

- 功能：

在palist所指向的线性表中，删除一个值为x的元素。

- 算法：

  先调用locate_seq(palist,x)

  再调用deleteP_seq(palist,p)

线性表的顺序表示和实现

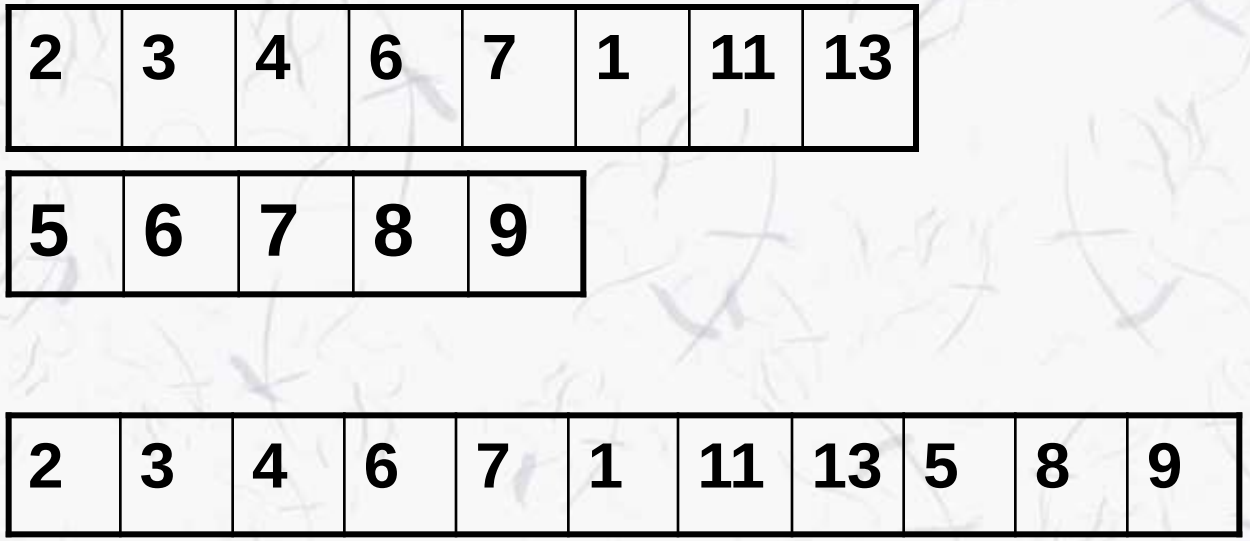
- 线性表的顺序表示
- 顺序表基本运算的实现
- 顺序表的应用

顺序表的应用

- 求集合的并集
- 纯集合构造
- 一个简单的完整程序
- 有序表的归并
- Josephus问题
- 一元多项式表示和运算

求集合的并集

- 利用线性表抽象数据类型提供的操作，可以完成其他更复杂的操作.
- 例①.假设有两个集合A和B分别用两个线性表来表示palist和pblist，（线性表中的数据元素即为集合的元素）。现在要求一个新集合A=AUB
- 上述问题可以演绎为:
  - 要求对线性表做如下操作:扩大线性表palist，将存在于线性表pblist中而不存在于线性表palist中的元素插入到线性表palist中去。


2 3 4 6 7 1 11 13
5 6 7 8 9
2 3 4 6 7 1 11 13 5 8 9

顺序表实现的操作步骤

- 从线性表B中依次取得每个元素
  - retrieve_seq(pblist, p) ⟶ x
- 在线性表A中查找x
  - locate_seq(palist, x)
- 若不存在，则插入，插入到表最后
  - insertPost_seq(palist,palist->n-1,x)
  - 完整的程序

```
void unionlist(PSeqList palist,PSeqList pblist)
{  int i;
   DataType x;
   for ( i=0;i<pblist->n;i++)
   {  x= retrieve_seq(pblist,i) ;
       //取pblist中序号为i的元素赋给x
      if ( locate_seq(palist,x)==SPECIAL0)
        insertPost_seq(palist, palist->n-1, x);
        //palist 中不存在和x相同的元素，插入之
   }
}// unionlist
```

- 算法的时间复杂度由控制结构和基本操作决定
- 在顺序表情况下时间复杂度为:n^2

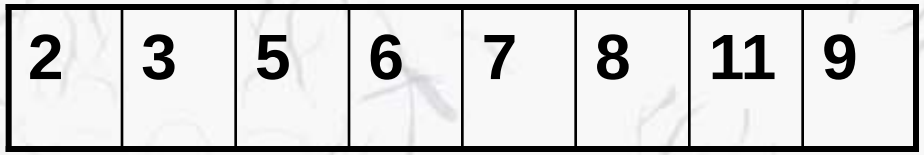
顺序表的应用

- 求集合的并集
- 纯集合构造
- 一个简单的完整程序
- 有序表的归并
- Josephus问题
- 一元多项式表示和运算

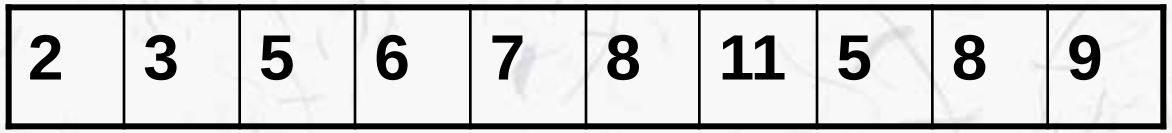
纯集合的构造

- 例②已知一个非纯集合B，试构造一个纯集合A，使A中包含B中各不相同的元素。

- 已知线性表pblist中包含集合B中所有元素，试构造线性表palist使palist中只包含pblist中不同的元素。



A



B

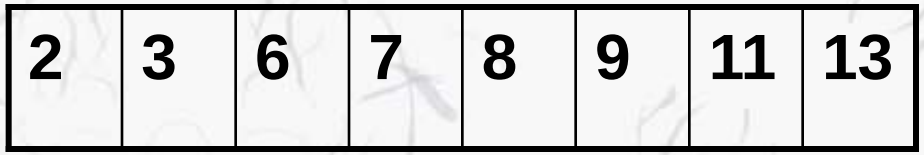
顺序表实现的操作步骤

- 从线性表B中依次取得每个元素
  - retrieve_seq(pblist, p) ⟶ x
- 在线性表A中查找x
  - locate_seq(palist, x)
- 若不存在，则插入，插入到表最后
  - insertPost_seq(palist, palist->n-1, x)

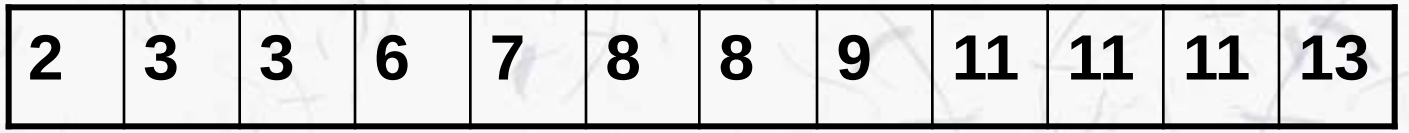
```
void purge(PSeqList palist,PSeqList pblist)
{
    int i=0;
    DataType x;
    for ( i=0;i<pblist->n;i++)
    {  x= retrieve_seq( pblist, i ) ;
        //取pblist中序号为i的元素赋给x
       if ( locate_seq( palist, x )== SPECIAL0)
           insertPost_seq(palist, palist->n-1, x);
           //palist 中不存在和x相同的元素，插入之
     }  //for
}  // purge
```

完整的程序

- 算法的时间复杂度由控制结构和基本操作决定
- 在顺序表情况下时间复杂度为:n^2



A



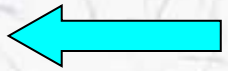
B

```
void purge(PSeqList palist,PSeqList pblist)
{  int i=0;
    DataType x,xn;
    xn= retrieve_seq(pblist,0);
    for ( i=0;i<pblist->n;i++)
    {  x= retrieve_seq(pblist,i) ;
                  //取pblist中序号为i的元素赋给x
       if ( isNullList_seq(palist)||x!=xn)
           insert_seq(palist, palist->n, x);
       xn=x; //palist 中不存在和x相同的元素，插入之
     }   // for
  }  // purge
```

完整的程序

- 算法的时间复杂度由控制结构和基本操作决定
- 在有序顺序表情况下时间复杂度：n

顺序表的应用

- 求集合的并集
- 纯集合构造
- 一个简单的完整程序 
- 有序表的归并
- Josephus问题
- 一元多项式表示和运算

求顺序表中第一个值为x的元素的前驱和后继的存储位置

数据结构

```
struct SeqList
{
    int MAXNUM;
    int n;                          /*元素个数n<MAXNUM*/
    DataType *element;
};
typedef struct SeqList *PSeqList;
int  SearchPN_seq(PSeqList palist,DataType x, int *pprev,int *pnext)
```

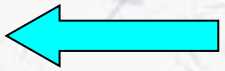
- 算法结束后,*pprev和*pnext中分别存放顺序表中第一个值为x的元素的前驱和后继的下标值(当不存在时用-1表示).当x在顺序表中出现时返回TRUE,否则返回FALSE

```
int SearchPN_seq(PSeqList palist,DataType x,int *pprev, int *pnext)
{  int index;
   for (index=0;index<palist->n;index++){
                        /*寻找值为x的元素*/
      if (palist->element[index]==x) /*找到值为x的元素*/
      {  *pprev=index-1;
         *pnext=index+1;
         if(index==palist->n-1) *pnext=-1;
                              /*最后一个元素无后继*/
          return TRUE;
      }
   }
   *pprev=-1;
   *pnext=-1;
   return FALSE;
}
```

完整程序

顺序表的应用

- 求集合的并集
- 纯集合构造
- 一个简单的完整程序
- 有序表的归并
- Josephus问题
- 一元多项式表示和运算

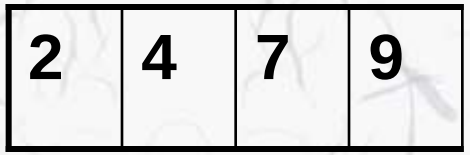


有序表的归并

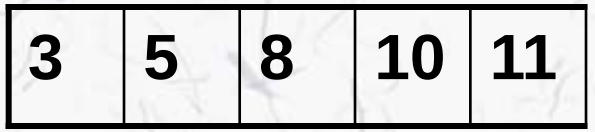
- 利用线性表抽象数据类型提供的操作,可以完成其他更复杂的操作.
- 例③.归并两个数据元素按非递减有序排列的线性表A和B,求得线性表C也具有同样的特性
- 上述问题可以演绎为:
  - 要求对线性表做如下操作:创建线性表pclist，将存在于线性表palist和线性表pblist中的元素按非递减的顺序插入到线性表pclist中去。

线性表实现的操作步骤

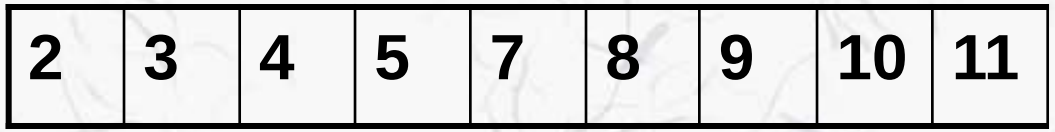
- 从线性表A、B中依次取得当前每个元素
  - retrieve_seq(pablist, i) ⟹ xa_i
  - retrieve_seq(pblist, j) ⟹ xb_j
- 若xa_i<=xb_j，则 把xa_i插入到表C中，否则把xb_j插入到表C中
  - insertPost_seq(pclist, pclist->n-1, x)
- 若A、B中还有没有处理的元素，把剩下的元素插入到C表中。



A



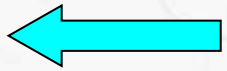
B



C

顺序表的应用

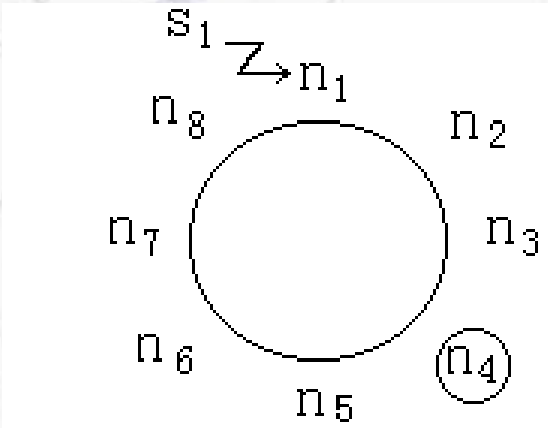
- 求集合的并集
- 纯集合构造
- 一个简单的完整程序
- 有序表的归并
- Josephus问题
- 一元多项式表示和运算



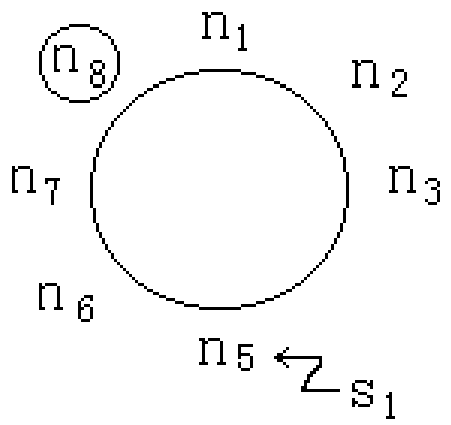
应用举例–Josephus问题

- n个人围坐在一圆桌周围，现从第s个人开始报数，数到第m的人出列，然后从出列的下一个人重新开始报数，数到第m的人又出列，如此反复直到所有的人全部出列为止。Josephus问题是：对于任意给定的n,s和m，求出按出列次序得到的n个人员的序列。

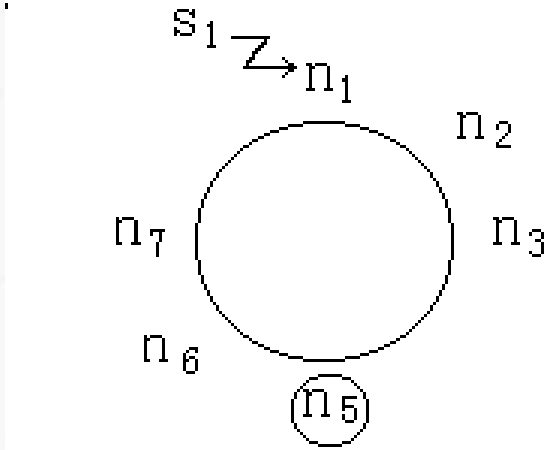
- 现以n=8，s=1，m=4为例，问题的求解过程如下列图所示。图中s_1指向开始报数位置，带圆圈的是本次应出列人员。若初始顺序为n1，n2，n3，n4，n5，n6，n7，n8，则问题的解为n4，n8，n5，n2，n1，n3，n7，n6。



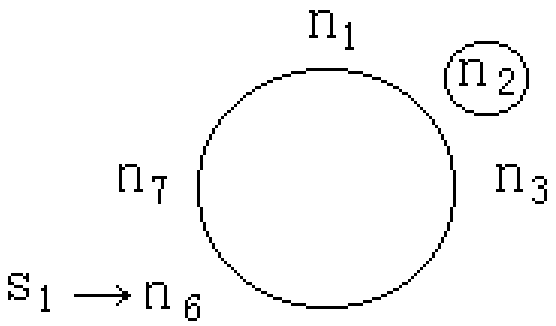
(a) n4



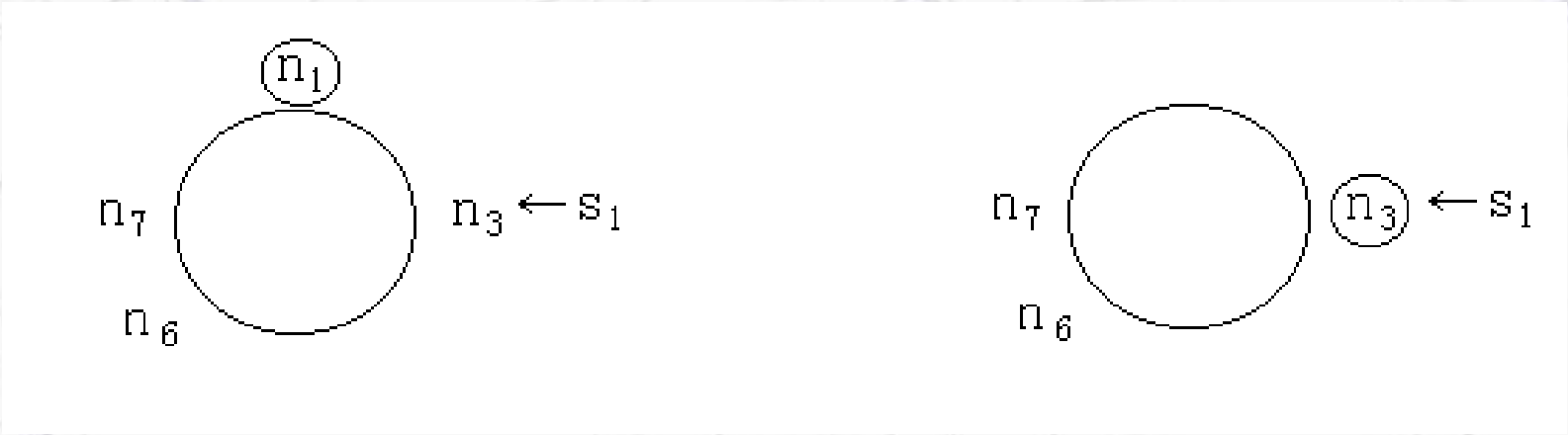
(b) n4 n8



(c) n4 n8 n5

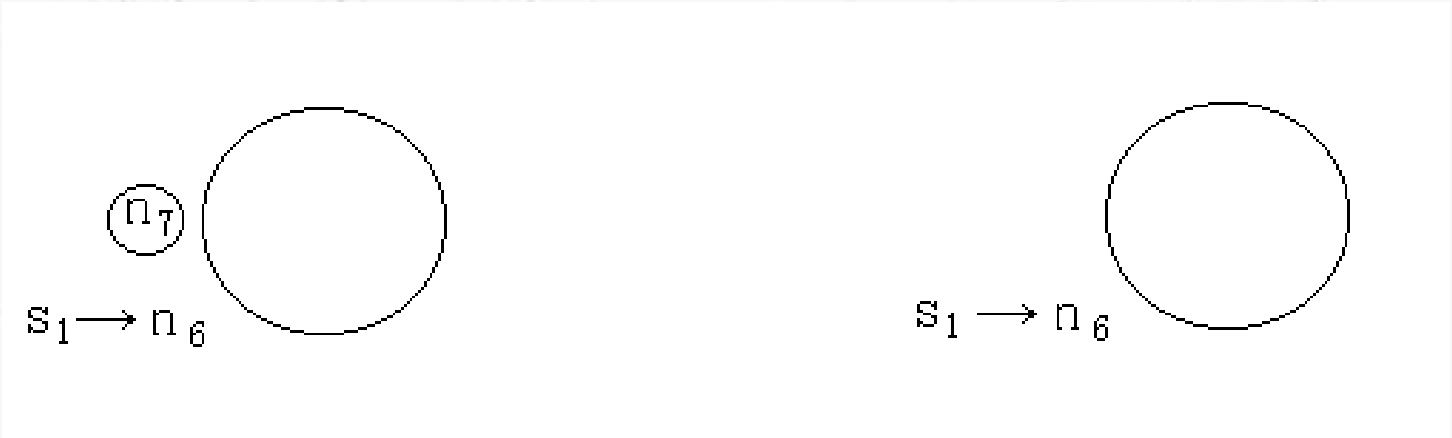


(d) n4 n8 n5 n2



(e) n4 n8 n5 n2 n1

(f) n4 n8 n5 n2 n1 n3



(g) n4 n8 n5 n2 n1 n3 n7

(h) n4 n8 n5 n2 n1 n3 n7 n6

顺序表方式实现

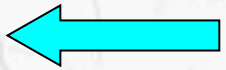
- 步骤：
  - 1：建立顺序表
  - 2：出列：
    - 定位：(s + m - 1) % n
    - 删除：打印 + 移动后面元素

- 时间复杂度分析：出列元素的删除（移动实现）为基本运算（每次最多i-1个元素移动，需要n-1次）

  (n-1)+(n-2)+……+1 = n(n-1)/2 => $O(n^2)$

顺序表的应用

- 求集合的并集
- 纯集合构造
- 一个简单的完整程序
- 有序表的归并
- Josephus问题
- 一元多项式表示和运算



一元多项式表示和运算-1

一元多项式：$P_n(x)=p_0 + p_1x + p_2x^2 +\dots +p_nx^n$

线性表表示：$P=(p_0, p_1, p_2, \dots, p_n)$

顺序表表示：只存系数（第i个元素存x^i的系数）

特殊问题：$p(x) = 1+2x^{10000}+4x^{40000}$ ⇨ 浪费空间

How to solve it ?

一元多项式表示和运算-2

- 数据：

  $f(x)=6x^{10}+12x^4+10x+11$

  $f(x)=3x^6+5x^2+9;$

- 运算：f(x)+g(x)　f(x)-g(x)　f(x)*g(x)　f(x)/g(x)

- 用线性表表示多项式；用结构体表示项；用结构体的成员（指数）表示项之间的降幂关系。

6	10
12	4
10	1
11	0

3	6
5	2
9	0

一元多项式表示和运算-3

- 数据定义：

```
struct item
 {  float c; //系数
    int e;   //指数
 } ;
```

- 操作：
  - 加法:相同指数对应结点的系数项相加，如和为0，删除元素，否则必定为和顺序表的一个结点。（实质上就是两个顺序表的合并问题）
  - 减法
  - 乘法
  - 除法

```
int ADD(List f,List g,List s)
{  item  a,b;
    int i=0,j=0;
    setNull(s);
    while(i<=length(f) && j<=length(g) )
    {  a=retrive(f,i); b=retrive(g,j);
       if(a.e = = b.e)
       {   a.c=a.c+b.c;
           if(a.c!=0){ insert(s,length(s)+1,a)); i++; j++;}
        }
       else
        {   if(a.e > b.e) { insert(s.length(s)+1,a); i++ }
            else{ insert(s,length(s)+1,b); j++; }
        }
   }
   for(;i<=length(f);i++)
   { a = retrive (f, i); insert(s,length(s)+1,a);}
   for(;j<=length(g); j++)
   { b = retrive (g,j); insert(s,length(s)+1,b);}
}
```

算法思路描述
（顺序表的方式）

两个一元多项式的乘法

- 可以利用两个一元多项式相加的算法来实现
- $M(x)=A(x)\times B(x)$

  $=A(x)\times[b_1x^{e1}+b_2x^{e2}+...+b_nx^{en}]$

  $=\sum_{i=1}^{n} b_i A(x)x^{e_i}$

顺序表示的优缺点

- 优点：逻辑关系上相邻的两个数据元素在物理位置上也是相邻的。因此，可以随机存取线性结构中的任一元素，并且数据元素的存储位置可用一个简单直观的公式来表示。
- 缺点：
  - 插入与删除运算要移动大量的元素，效率较低。
  - 要求占用连续的存储空间，存储分配只能预先进行，而且必须要按最大空间来分配，然而，估计最大空间需要是一件比较困难的事情。

➢ 改进：使用链接存储结构（不要求逻辑相邻物理也相邻，通过指针表达逻辑关系），克服顺序存储的不足，但失去了随机存取的优点。