2.3 线性链表及其运算

- 线性表的顺序存储结构容易实现，可以随机存取表中的任意元素。
- *顺序表*缺点是：
  - 难于插入、删除操作；
  - 需要预先分配空间，不管这些空间能否最大限度地利用。
- *链表存储结构*在这两个方面恰好是优点：
  - 容易插入、删除操作
  - 不需要预分空间。

链表基本概念

定义： 线性表的链式存储结构称为线性链表

结点（NODE）：表中元素的存储单元。由数据域和指针域组成。数据域存放元素数据，指针域存放指向下一结点位置的指针。

结点形式为：

数据域　　指针域

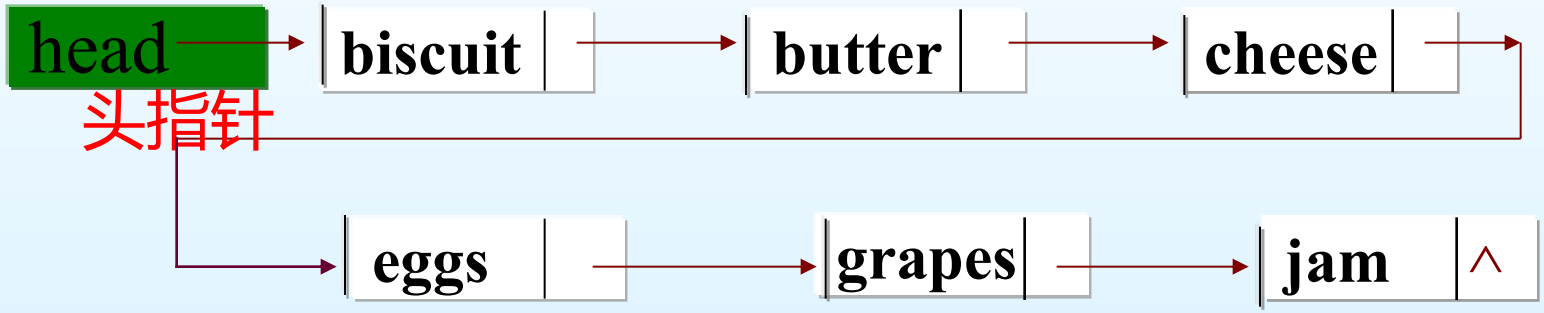
链表（Link）： 由结点组成的表。

头指针（head）： 指向链表中第1个结点的指针。

举例

- 由食品组成的单链表

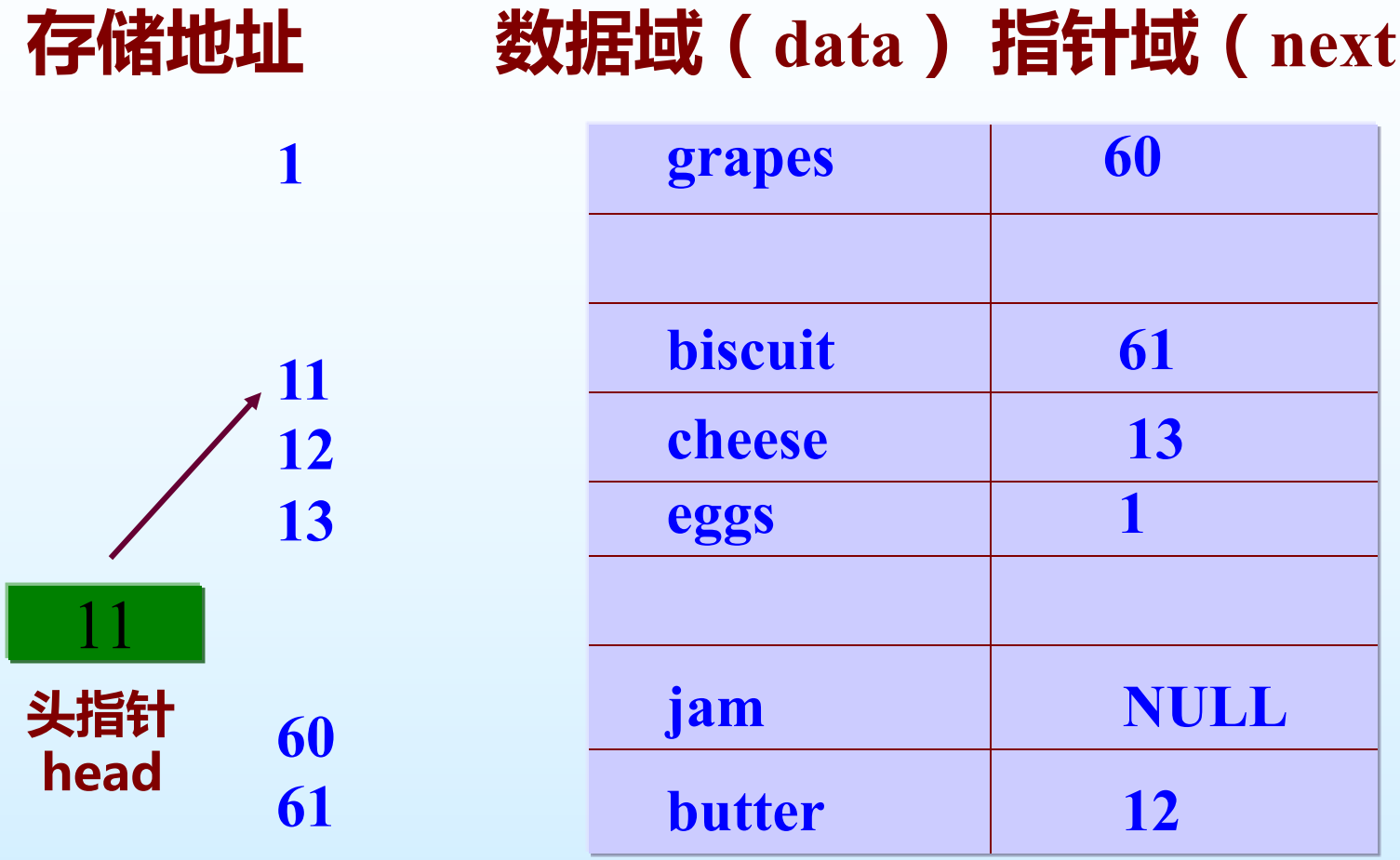
  （biscuit,butter,cheese,eggs,grapes,jam)



最后一个结点
指针域为空

单链表的物理存储

- (biscuit,butter,cheese,eggs,grapes,jam)



1.单链表的基本运算

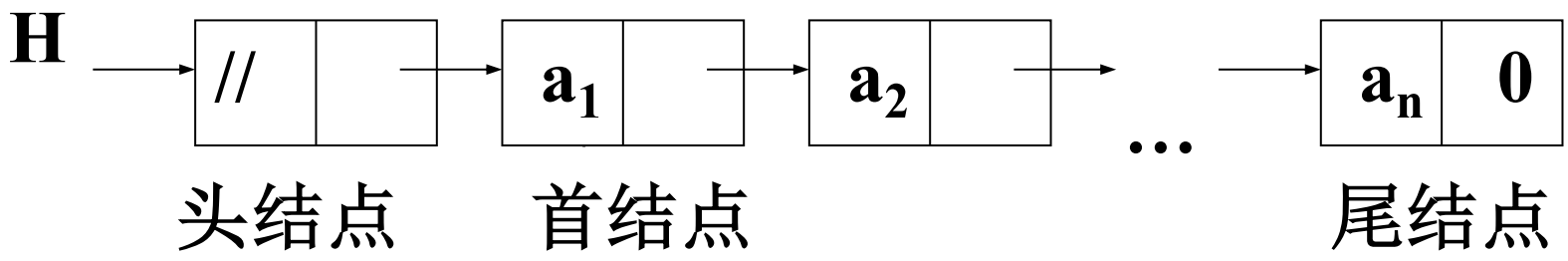
单链表结构类型定义如下[结点可按需定义]：

```
typedef struct node
{
ElemType data; /*数值域*/
struct node *next; /*指针域*/
}ListNode,*LinkList;
```

头结点：为了便于实现插入、删除结点的操作，通常在单链表的第一个结点之前增设一个表头结点。 该结点的结构与表中其他结点的结构相同，其数据域可以不存储任何信息，也可以存储如线性表的表名、长度等附加信息；若线性表为空表，则表头结点的指针域为空，设H为单链表的头指针，指向头结点；

首结点：存储单链表的第一个元素的结点称为开始结点（或称首结点），其指针域存放第二个结点的地址；

尾结点：称存储最后一个元素的结点称终端结点（或称尾结点），其指针域为空。



带表头结点和头指针的单链表示意图

依据新结点插入位置的不同，将生成的单链表分为先进先出、后进先出及有序三种。

① 先进先出单链表：在建立单链表时，将每次生成的新结点，总是插入到当前链表的表尾作为尾结点。

若用换行符‘\n’作为输入结束标志，用rear作为总是指向链表尾结点的尾指针，则建立带表头结点的先进先出单链表的算法如下：

```
LinkList CreateList_ff( )
{
   LinkList H,p,rear;
   char ch;
H=(LinkList)malloc(sizeof( L
istNode)); /*生成表头结点*/
```

```
 if(!H)
   {
     printf("\n 存储分配失败");
     exit(1);
   }
  H->next=NULL;   /*表初值为空*/
  rear=H;    /*尾指针指向表头结点*/
while((ch=getchar())!='\n')
{    p=(LinkList)malloc(sizeof(
ListNode));   /*生成新结点*/
```

```
if(!p)
   {
   printf("\n 存储分配失败");
   exit(1);
   }
      p->data=ch;
      rear->next=p;
      rear=p;   /*尾指针指向新结点*/
}
return(H);     /*返回头指针*/
}
```

② 后进先出表：在建立单连表时，将每次生成的新结点，总是插入到当前链表的表头结点之后作为当前链表的首结点。

```
LinkList CreateList_fr( )
```

③ 有序单链表：是指原表中结点的数据值，按从小到大（或从大到小）的顺序排列。为了建立一个有序单链表，每次生成的新结点，总是插入到当前链表的合适位置。在带表头结点的单链表中，所有位置上的插入操作都是一致的，不需要做特殊处理。

```
LinkList CreateList_or( )
```

以上3个算法的时间复杂度均为O(n)。

2）查找结点

在对单链表进行插入或删除运算时，总是首先要找到插入或删除的位置，这就需要对线性链表进行扫描查找。

通常，有查找第i个结点或查找具有给定元素值的结点，两种情况。

① 查找单链表中的第i个结点

```
LinkList GetElem(LinkList H,
 int i )
{
  int j=1;  /*j累计当前扫描的结点数*/
  LinkList p;
  p=H->next;     /*p指向第一个结点*/
  while(p&&j<i) /*沿指针向下搜索*/
 {
p=p->next;
j++;
}
```

```
if(j==i&&p)
return(p);/*找到返回指向该结点的指针*/
else
 return(NULL); /*第i个结点不存在，返回NULL*/
}
```

② 在单链表中查找给定值的结点

在带表头结点的单链表H中查找x的算法：

```
LinkList LocateElem(LinkList H,
ElemType x,int *i)
{
  LinkList p;   /*p为指针*/
  p=H->next; /*p指向第一个结点*/
  *i=1;
  while(p&&p->data!=x)/*向下搜索*/
 {
  p=p->next;
  (*i)++;
```

```
if(p->data==x)
return(p); /*找到返回结点位置*/
 else
return(NULL);/*没找到返回NULL*/
}
```

以上两个，查找算法的时间复杂度均为O(n)。

3）单链表的插入

单链表的插入，是指在单链表中插入一个新结点。有两种情况：在单链表的第i个位置插入一个新结点和在有序单链表中插入一个新结点(略)。

①在单链表的第i个位置插入一个新结点

假设单链表中有n个结点，插入位置i的允许取值范围为1至n+1。

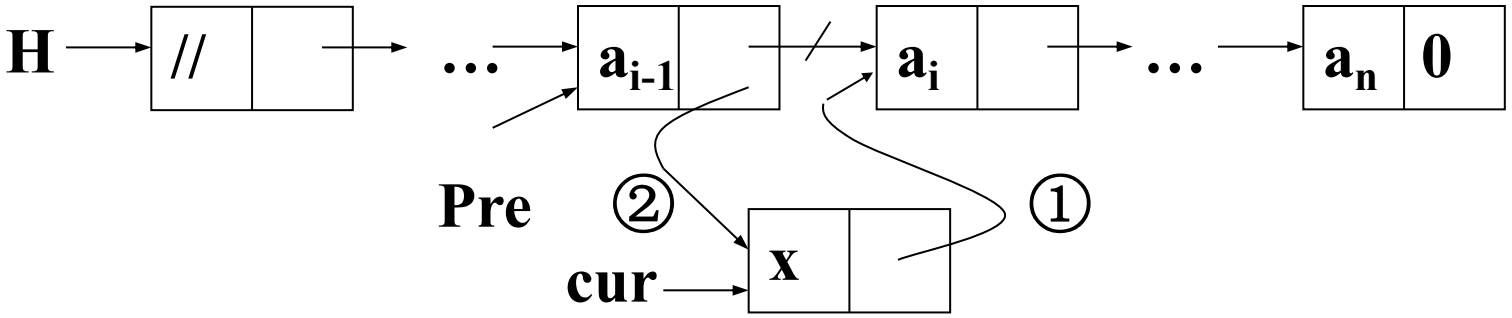
首先，要找到第i-1个结点，

指针pre指向第i-1个结点，

cur指向新结点，

cur->next=pre->next，

pre->next=cur。



在单链表的第i个位置插入结点示意图

在第i个位置，插入一个值为x的新结点算法：

```
InsertList(LinkList H,int i,ElemType x)
{
LinkList pre,cur;
pre=GetElem(H ,i-1);/*调用查找函数使pre
指向第i-1个结点*/
```

```
if(!pre)
{ printf("\n i值不合法");
exit(1);
}
cur=(LinkList)malloc
(sizeof(ListNode)); /*生成新结点*/
if(!cur)
{
printf("\n 存储分配失败");
exit(1);
}
```

```
cur->data=x;/*新结点的值为x*/
cur->next=pre->next;/*新结点的指针域值指向原来的第i个结点*/
pre-next=cur;/*第i-1个结点的指针域值指向新结点*/
}
```

在以上两个插入结点的算法中，时间复杂度的数量级均为O(n)。

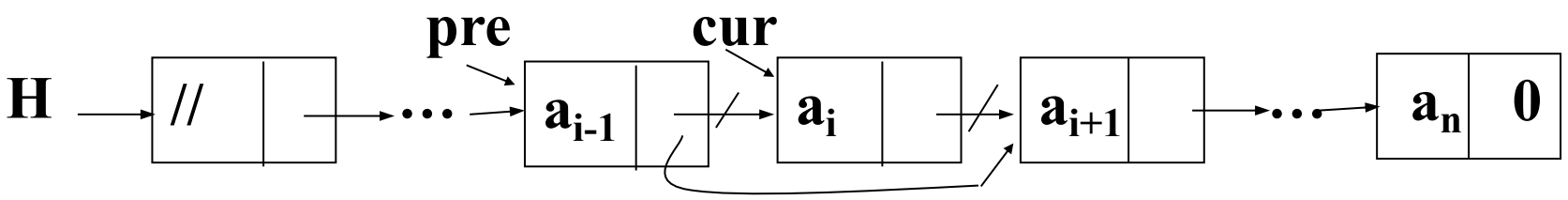
4）单链表的删除操作

单链表的删除操作，是指在单链表中，删除包括指定元素的结点。

一是删除单链表的第i个结点；

二是删除单链表中，具有给定值x的结点。

① 删除单链表的第i个结点



删除单链表的第i个结点示意图

```
DeleteList1(LinkList H,int i,
 ElemType *x)
{
LinkList pre,cur;
pre=GetElem(H,i-1);
/*用GetElem函数找第i-1个结点并使pre指
向它*/
```

```
if(pre==NULL||pre->next==NULL)
   {     /*判断i值是否合法*/
   printf("\n i值不合法!");
   exit(1);
   }
cur=pre->next;/*cur指第i个结点*/
pre->next=cur->next;/*使第i-1个结点的指针指向第i+1个结点*/
*x=cur->data;/*第i个结点的值由x返回*/
free(cur); /*释放第i个结点所占的空间*/
}
```

② 删除单链表中具有给定值的结点

在带表头结点的单链表H中，搜索具有给定值x的结点，若找到，就删除该结点。这种删除情况的算法如下：

```
DeleteList2(LinkList H ,
ElemType x)
{
LinkList p,pre;
p=H->next;/*p指向链表的第1个结点*/
pre=H; /*pre指向第1个结点的前件*/
```

```
while(p&&p->data!=x)
   {pre=p;        /*记住前一结点*/
p=p->next; /*指针p后移一个结点*/ }
if(p->data==x)
   {pre->next=p->next;/*删除值为x的
结点*/
 free(p);/*释放值为x的结点空间*/
 } else
printf("\n 链表中没有具有值为x的结点
");}
```

此算法的时间复杂度的数量级为O(n)。

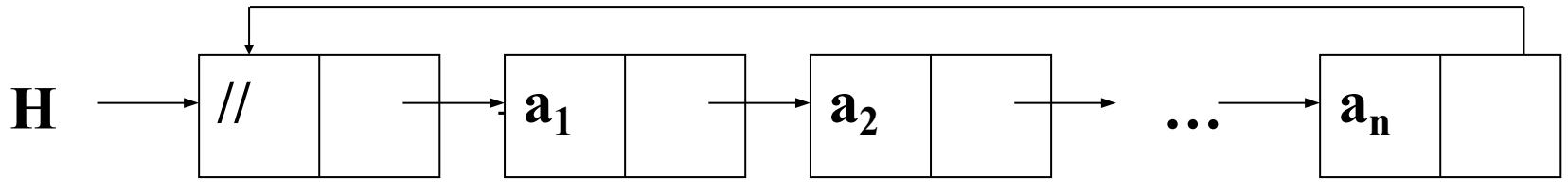
例子
课堂练习

1) 建立单链表（想想怎么做）
2) 依次在链表的尾部插入1--100
3) 删除链表中数据能被3整除的结点

2.循环链表及其基本运算

使带表头结点的单链表中的最后一个结点的指针指向头结点，使整个链表构成一个环形，这种链式存储结构被称为循环链表。

特别地，只有头结点的循环链表称为空循环链表。



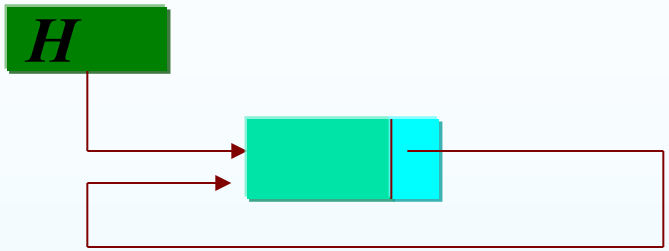
循环链表示意图

单循环链表特点：

- 从表中任一结点出发，均可以找到表中其它结点。
- 找其前件结点的时间复杂度是O（n）。

单循环链表为空的条件： ***H->next=H***

表示形式为:



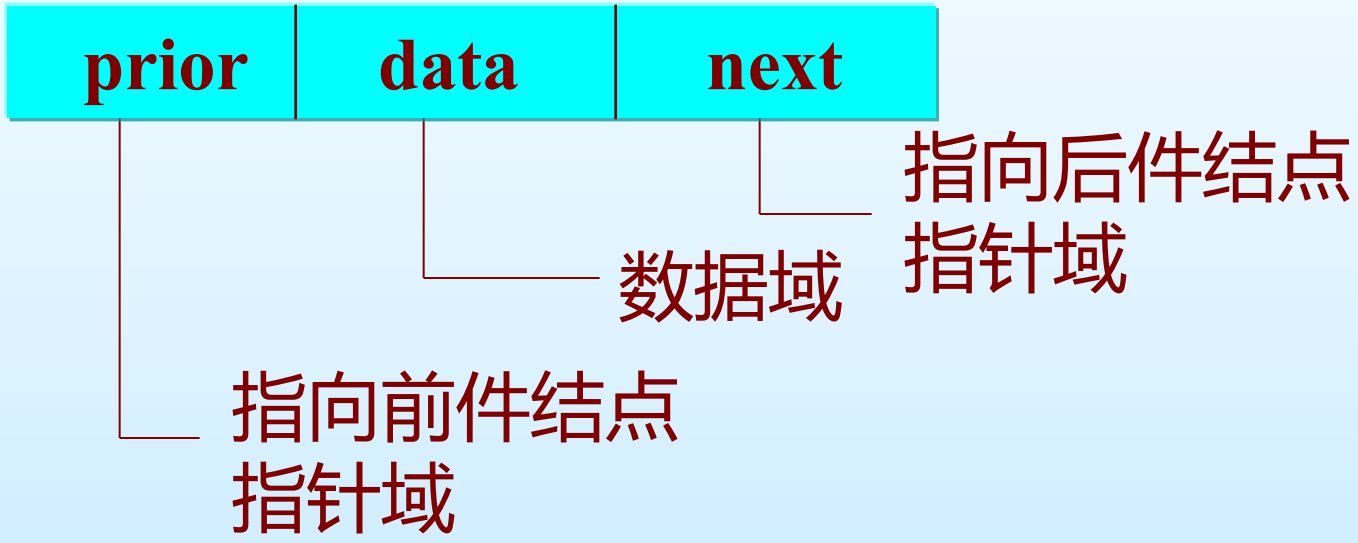
循环链表的插入和删除的方法与线性单链表基本相同，不再赘述。

只须注意一点：在单链表的结构中，空表的条件是**H->next==NULL**,而在循环链表结构中，空表的条件是**H->next==H**。

3. 双向链表

从循环链表的结构特征可看出，若要找一个结点的直接前件结点需要兜一个圈子才行，这当然是很不方便的。就像乘单向环行的公交车一样，若坐过了站，就要多坐许多站才能到达预定的地点。

双向链表的每个结点含有两个指针域：一个指向其直接前件结点，称为**prior**；一个指向其直接后件结点，称为**next**。

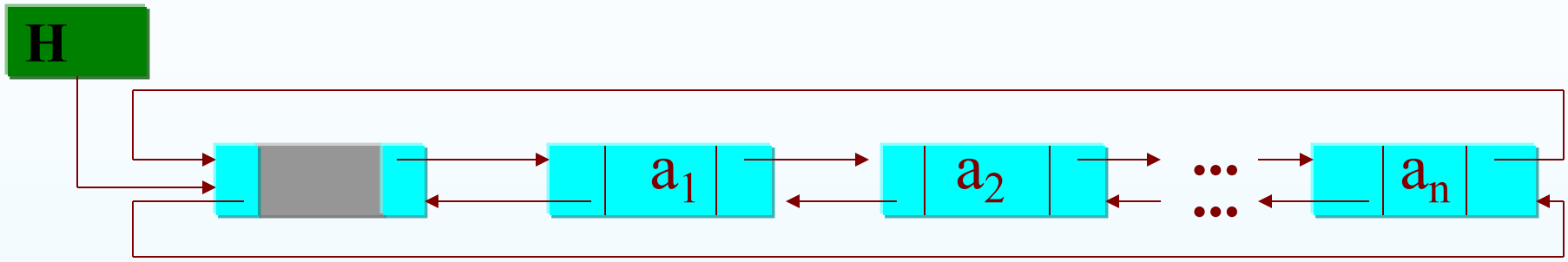


双向链表示意图

有了两个方向的链指针之后，在链表上进行访问非常方便，但是在双向链表上进行插入和删除运算，要涉及两个指针的链接，故比单链表的插入与删除运算复杂些。

双向循环链表表示形式

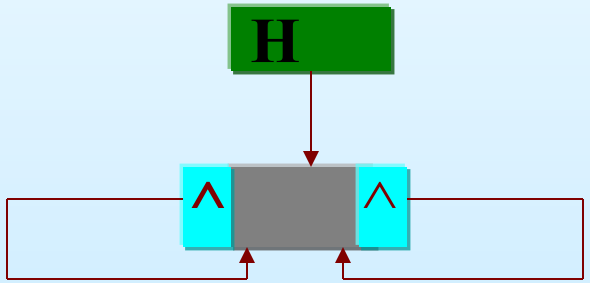
双向循环链表表示形式：



双向循环链表为空的条件：

H->prior = H->next = H

表示形式为：



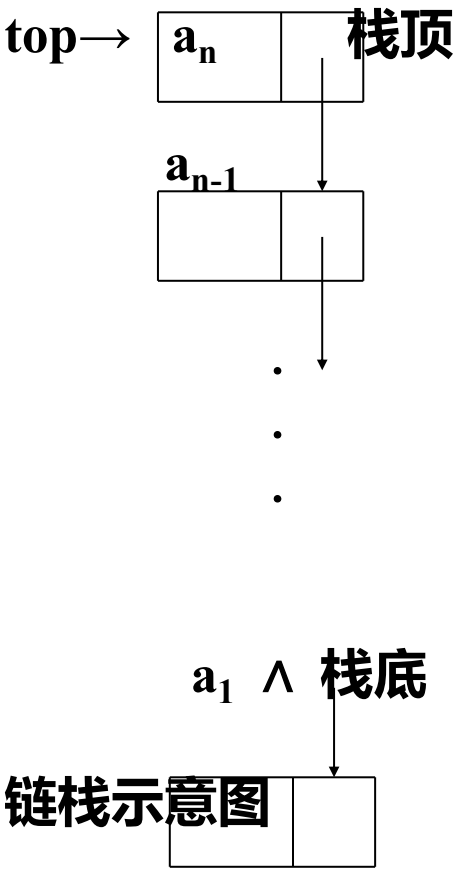
带链的栈

- 栈也是线性表，可以采用链式存储结构
- 顺序栈最多可用于2个栈的共享，对于更多的栈就难于表达了。
- 对于最大空间需要量事先不知的情况，就不能使用顺序栈了。这时，就需要采用链栈。

链栈：栈的链式存储结构

栈的链式存储结构及运算

栈的链式存储结构可通过单链表来实现



链栈示意图

```
typedef char ElemType;
        /* 新类型ElemType是字符型*/
typedef struct stacknode
{
 ElemType data;/*栈的值域是字符型*/
 struct stacknode *next;
}StackNode; /*StackNode为结构型*/
typedef struct
   {
     StackNode *top;/*栈顶指针*/
}LinkStack; /*LinkStack为结构类型*/
```

链栈的基本操作

①构造一个空链栈

```
void InitStack(LinkStack *s)
{
   s->top=NULL; /*栈顶指针置空*/
}
```

②判断链栈是否为空栈

```
int StackEmpty(LinkStack *s)
{     /*若栈空，则返回1否则返回0*/
   return s->top==NULL;
}
```

③进栈（入栈或压栈）

```
void push(LinkStack *s,ElemType x)
{ StackNode *p;
  p=(StackNode *)malloc(
sizeof(StackNode));
if(!p)
  { printf("\n 存储分配失败！");
   exit(1);
  }
  p->data=x;   /*新结点值域赋值x*/
  p->next=s->top;/*新结点*p插入到栈顶*/
  s->top=p;   /*修改栈顶指针*/
}
```

④退栈（出栈）

```
ElemType pop(LinkStack *s)
{
   ElemType x;
   StackNode *p;
   p=s->top;
   if(StackEmpty(s))
     {/*调用判断栈是否为空的函数，若栈空，则下溢*/
      printf("\n 栈已空，下溢！");
      exit(1);
     }
```

```
x=p->data;
      /*x□ □ □ □ □ □ □ □ □ □ □ □ */
 s->top=p->next;
      /*□ □ □ □ □ □ □ □ □ □ □ □ □ */
 free(p); /*□ □ □ □ □ □ □ □ */
 return x;/*□ □ □ □ □ □ □ □ */
}
```

⑤读链栈的栈顶元素

```
ElemType GetTop(LinkStack *s)
{
 if(StackEmpty(s))
   {
     printf("\n 栈已空！");
     exit(1);
   }
   return s->top->data; /*若链栈非空，则返回栈顶元素，但不删除*/
}
```

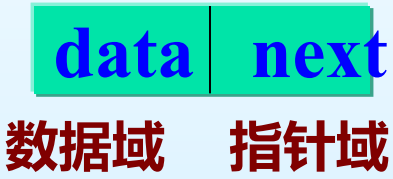
课堂练习

- 建立空链栈并初始化；
- 入栈 **10,20,30,40,50**；
- 退栈**2**次，然后输出栈中所有的元素；

4 带链的队列

1). 队列也是线性表，可以采用链式存储结构。

2). **存储结构的C 语言描述，**



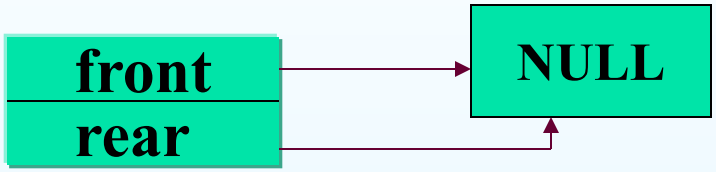
```
typedef char ElemType;
typedef struct queuenode
{              /*链队列中的结点类型 */
  ElemType data;
  struct queuenode *next;
}QueueNode;
typedef struct
{             /*将两个指针封装在一起*/
  QueueNode *front;/*队头指针*/
  QueueNode *rear;  /*队尾指针*/
}LinkQueue;
```

链队列为空的表示

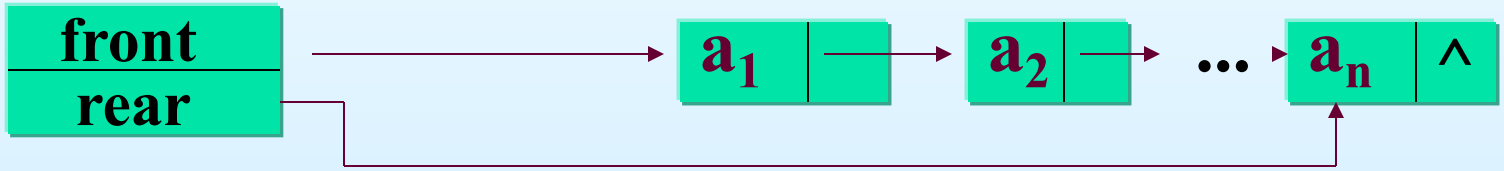
链队列为空，

Queue.front = Queue.rear

表示形式：



非空队列：



链队满的条件为:T = NULL。T 为新创建的结点，

链队列的入队操作

要求：在链队列中插入一个元素x（入队运算）。

算法操作步骤：

Step1：申请建立一个新结点p；

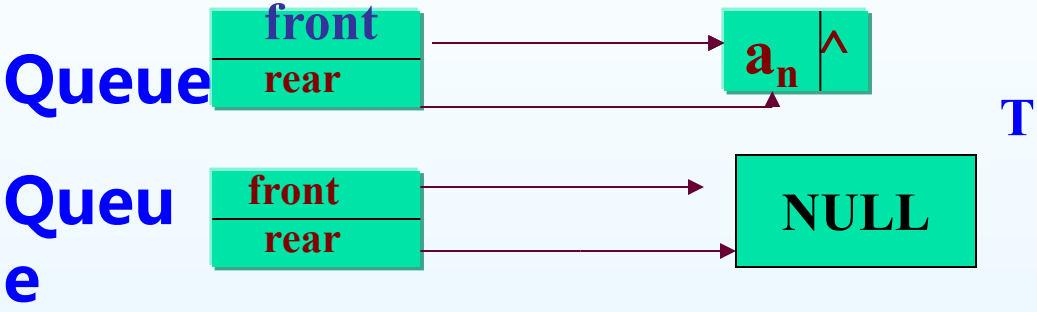
Step2：判别p是否为空；若空，表示 队列已满；

Step3：非空，将p插入链中，修改rear指针。

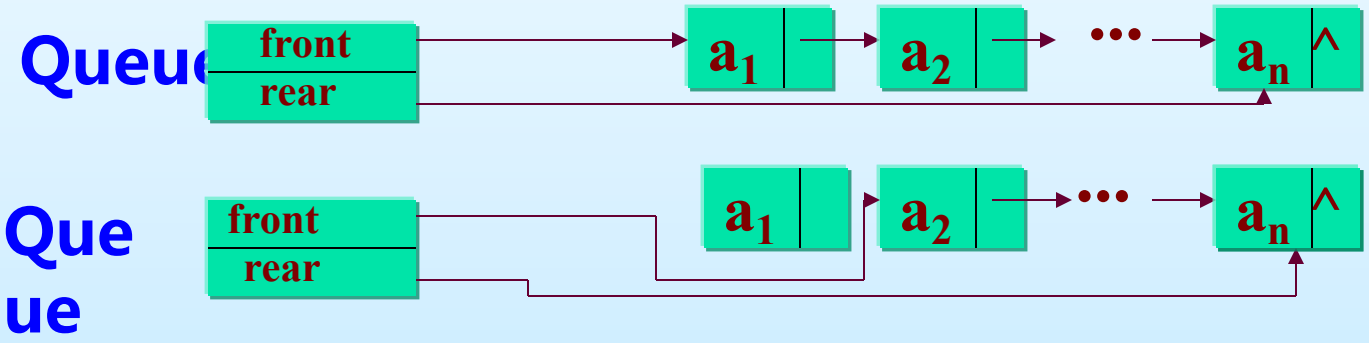
链队列的出队操作

出队操作要考虑两种情况：

- 当队列长度为1时，除了修改队头指针外，还要修改队尾指针。



- 当队列长度大于1时,只修改队头指针即可



链队列的出队操作

要求：在链队列中删除一个元素（退队运算）。

算法描述:

Step1：判别队列是否为空；若空，则显示队列‘下溢’；

Step2 ：非空，则判别队列长度是否为1；

Step2.1：不为1，修改头指针；

Step2.2：为1，则修改头、尾指针；

Step3：释放T。

链表存储结构的特点

- 插入、删除操作极为方便
- 数据非连续存放、顺序存取
- 逻辑上相邻，物理上不一定相邻
- 存储结构较复杂、需要额外的存储空间

结论：

链表存储结构适合于表中元素频繁变动的线性表。

作业

1）假设一单循环链表的长度大于1，且表中即无头结点也无头指针。已知S为指向链表中某结点的指针。试写出删除表中结点S 的算法。

2）假设以数组sequ[m]存放循环队列的元素，设变量rear和quelen分别为指示队尾元素位置和队中元素个数，试写出入队和出队算法。

2.4 数组

2.4.1 数组的顺序存储

- 数组是相同类型数据元素的有限集合；
- 数组中的各个分量称为数组元素；
- 每个数组元素值可以用数组名和一个下标值唯一地确定；

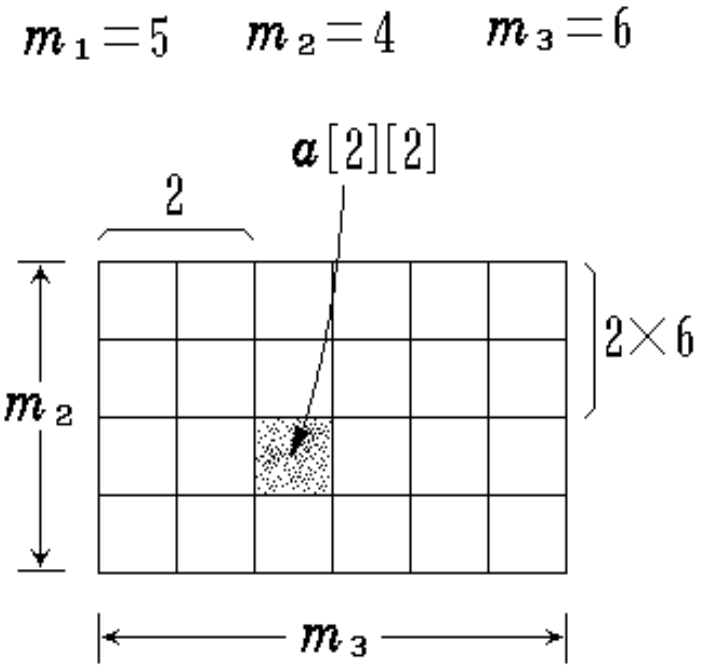
数组是有限个数组元素的集合。

数组中所有元素有相同的特性。

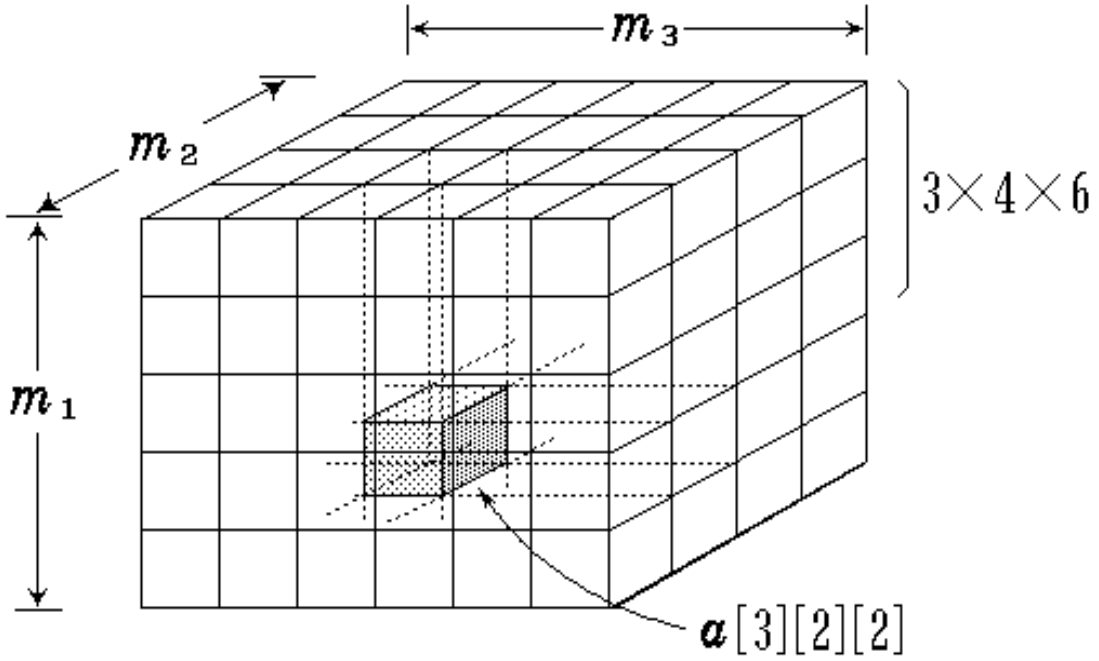
每个数组元素由数组名和下标组成。

每个具有下标值的数组元素有一个与该下标值对应的数组元素值。

二维数组



三维数组



- 行向量　下标 i　　页向量　下标 i
- 列向量　下标 j　　行向量　下标 j
  列向量　下标 k

数组元素之间的关系

二维数组m行n列可以看作是m个或n个一维数组:

$$\mathbf{A_{m\times n} = ((a_{11}a_{12}\ldots a_{1n}),(a_{21}a_{22}\ldots a_{2n}),\ldots (a_{m1}a_{m2}\ldots a_{mn}))}$$

$$或A_{mxn} = \left[\begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix}\begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix}\cdots\begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix}\right]$$

数组的操作

数组有两种基本的操作：

- 给定下标，读取相应的数组元素；
- 给定下标，修改相应数组元素的值。

数组的顺序存储结构

- 数组元素是连续存放的，因此采用顺序存储结构。
- 无论几维数组，在计算机中都是按一维数组来存放。数组存放通常采用两种方式：
  - **按行优先顺序(Pascal，C)**
  - **按列优先顺序(Fortran)**

1).按行优先顺序存储结构

例如：二维数组$A_{m\times n}$，可以看作m个行向量，每个行向量n个元素。数组中的每个元素由元素的两个下标表达式唯一确定。

地址计算公式：

$LOC(a_{ij})=LOC(a_{11})+((i-1)\times n+(j-1))\times L$

其中，L 是每个元素所占的存储单元。

二维数组按行优先存储举例

有二维数组如下：

$$A_{4\times4}=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$LOC(a_{23})= LOC(a_{11})+(2-1)\times4+(3-1)= 7$

$LOC(a_{34})= 1+(3-1)\times4+(4-1)= 12$

$LOC(a_{14})= 1+(1-1)\times4+(4-1)= 4$

2).按列优先顺序存储结构

按列优先顺序存放是将数组看作若干个列向量。

例如，二维数组Amxn，可以看作n个列向量，每个列向量m个元素。数组中的每个元素由元素的两个下标表达式唯一确定。

地址计算公式：

$$LOC(a_{ij}) = LOC(a_{11}) + ((j-1)*m+(i-1))*L$$

其中，L 是每个元素所占的存储单元。

二维数组按列优先存储举例

有二维数组如下：

$$A_{3\times4}=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

LOC（a_{23}）= LOC（a_{11}）+（3-1）* 4+（2-1）= 10

LOC（a_{34}）= 1 +（4-1）* 4 +（3-1）= 15

LOC（a_{14}）= 1 +（4-1）* 4 +（1-1）= 13

2.4.2 规则矩阵的压缩

实际工程问题中推导出的数组常常是高阶、含大量零元素的矩阵，或者是一些有规律排列的元素。为了节省存储空间，通常是对这类矩阵进行压缩存储。

压缩的含义是：

- 相同值的多个元素占用一个存储单元；
- 零元素不分配存储单元。

能够采用压缩存储的矩阵

- 对称矩阵：存储主对角线以上（下）的元素；
- 上（下）三角矩阵：只存储三角阵元素；
- 带状矩阵：只存储带状元素；
- 稀疏矩阵：只存储非零元素；

1 下三角矩阵的压缩存储

$$A=\begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & 0 & \\ \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

开辟一个长度为n（n+1）/2的一维数组B，然后一行接一行地依次存放A中下三角部分的元素。

以行为主压缩存储

$$a_{ij}=\begin{cases}B[i(i-1)/2+j] & (j\le i)\\ 0 & (j>i)\end{cases}$$

以列为主压缩存储

$$a_{ij}=\begin{cases}B[(2n-j+2)(j-1)/2+i-j+1] & (j\le i)\\ 0 & (j>i)\end{cases}$$

2 对称矩阵的压缩存储

对称矩阵的元素满足：$a_{ij} = a_{ji}$　$1 \le i，j \le n$

因此将n*n 个元素压缩存放到 n（n+1）/2 个单元的一维数组S（（n+1）*n/2）中。

（按行存放）a_{ij}的地址为：

$$LOC(a_{ij})=\begin{cases} i(i-1)/2+j & 当 \quad i>=j \\ j(j-1)/2+i & 当 \quad i<j \end{cases}$$

对称矩阵的压缩存储举例

设有A_{3x3}矩阵: $A=\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$

存于一维数组S[6]

S[6]=(a_{11}, a_{21}, a_{22}, a_{31}, a_{32}, a_{33})

1 2 3 4 5 6

$LOC(a_{31})=3(3-1)/2+1=4$

$LOC(a_{22})=2(2-1)/2+2=3$

$LOC(a_{21})=2(2-1)/2+1=2$

3 三对角矩阵的压缩存储

$$A=\begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & 0 & \\ & a_{32} & a_{33} & a_{34} & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ & & & & a_{n,n-1} & a_{nn} \end{bmatrix}$$

在三对角矩阵中，三条对角线以外的元素均为零，并且，除了第一行与最后一行外，其他每一行均只有三个元素为非零，因此，**n**阶三对角矩阵共有**3n-2**个非零元素。

对角矩阵的压缩存储

以行为主存放：

$$a_{ij}=\begin{cases} B[2(i-1)+j] & (i-1\le j\le i+1) \\ 0 & (j<i-1 \text{ 或 } j>i+1) \end{cases}$$

以列为主存放：

$$a_{ij}=\begin{cases} B[2(\mathrm{j}-1)+\mathrm{i}] & (i-1\le j\le i+1) \\ 0 & (j<i-1 \text{ 或 } j>i+1) \end{cases}$$

2.4.3 一般稀疏矩阵的表示

- 如果一个矩阵中绝大多数的元素值为零，只有很少的元素值非零，则称该矩阵为稀疏矩阵 。

$$A=\begin{bmatrix} 0 & 0 & 3 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 2 & 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

1 三列二维数组表示

（1）非零元素所在的行号i；

（2）非零元素所在的列号j；

（3）非零元素的值V。

即每一个非零元素可以用下列三元组表示：

（i，j，V）

- 例如，上述稀疏矩阵A中的8个非零元素可以用以下8个二元组表示（以行为主的顺序排列）：

  （1，3，3） （1，8，1）（3，1，9）（4，5，7）

  （5，7，6）（6，4，2）（6，6，3） (7，3，5）

- 为了表示的唯一性，除了每一个非零元素用一个三元组表示外，在所有表示非零元素的三元组之前再添加一个三元组：（I，J，t）

- 其中I表示稀疏矩阵的总行数，J表示稀疏矩阵的总列数，t表示稀疏矩阵中非零元素的个数。

- 上述稀疏矩阵A可以用以下9个三元组表示：

  （7，8，8）（3，1，9）（5，7，6）

  （6，6，3）（1，3，3）（4，5，7）

  （6，4，2）（7，3，5）（1，8，1）
- 其中第一个三元组表示了稀疏矩阵的总体信息（总行数，总列数、非零元素个数），
- 其后的8个三元组依次（以行为主排列）表示稀疏矩阵中每一个非零元素的信息（所在的行号、列号以及非零元素值）。

为了使各三元组的结构更紧凑，通常将这些三元组组织成三列二维表格的形式，一般又表示成三列二维数组的形式，并简称为三列二维数组。

$$B=\begin{bmatrix}7&8&8\\1&3&3\\1&8&1\\3&1&9\\4&5&7\\5&7&6\\6&4&2\\6&6&3\\7&3&5\end{bmatrix}$$

- 为了便于在三列二维数组B中访问稀疏矩阵A中的各元素，通常还附设两个长度与稀疏矩阵A的行数相同的向量POS与NUM，
- POS（k）表示稀疏矩阵A中第k行的第一个非零元素（如果有的话）在三列二维数组B中的行号，
- NUM（k）表示稀疏矩阵A中第k行中非零元素的个数，
- 这两个向量之间存在以下关系：

  $POS(1)=2$

  $POS(k)=POS(k-1)+NUM(k-1)，2 \le k \le m$

课堂练习

$$A=\begin{bmatrix} 9 & 0 & 4 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(1) 请用三列二维数组表示；
(2) 写出其POS和NUM向量；

2 线性链表表示

- 稀疏矩阵运算后非零元素个数变化时，采用三列二维数组表示不方便；
- 可采用链表的方式来表示
- 结点定义：

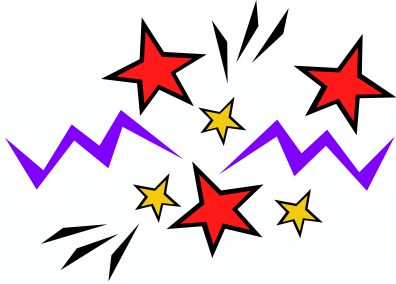
行域	列域	值域	指针域

3、十字链表

row	col	val
行域	**列域**	**值域**
向下域	**向右域**	
down	right	

每个结点有五个域：行域、列域、值域、向下域与向右域。

十字链表表示稀疏矩阵



（1）稀疏矩阵的每一行与每一列均用带表头结点的循环链表表示。

（2）表头结点中的行域与列域的值均置为-1（即row＝-1，col＝-1）。

（3）行、列链表的表头结点合用，且这些表头结点通过值域（即val）相链接，并再增加一个结点作为它们的表头结点H，其行、列域值分别存放稀疏矩阵的行数与列数。

例如，稀疏矩阵

$$A=\begin{bmatrix}3 & 0 & 0 & 7\\ 0 & 0 & 1 & 0\\ 2 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 9\end{bmatrix}$$

注意：其十字链表表示的稀疏矩阵如图所示。**P110**

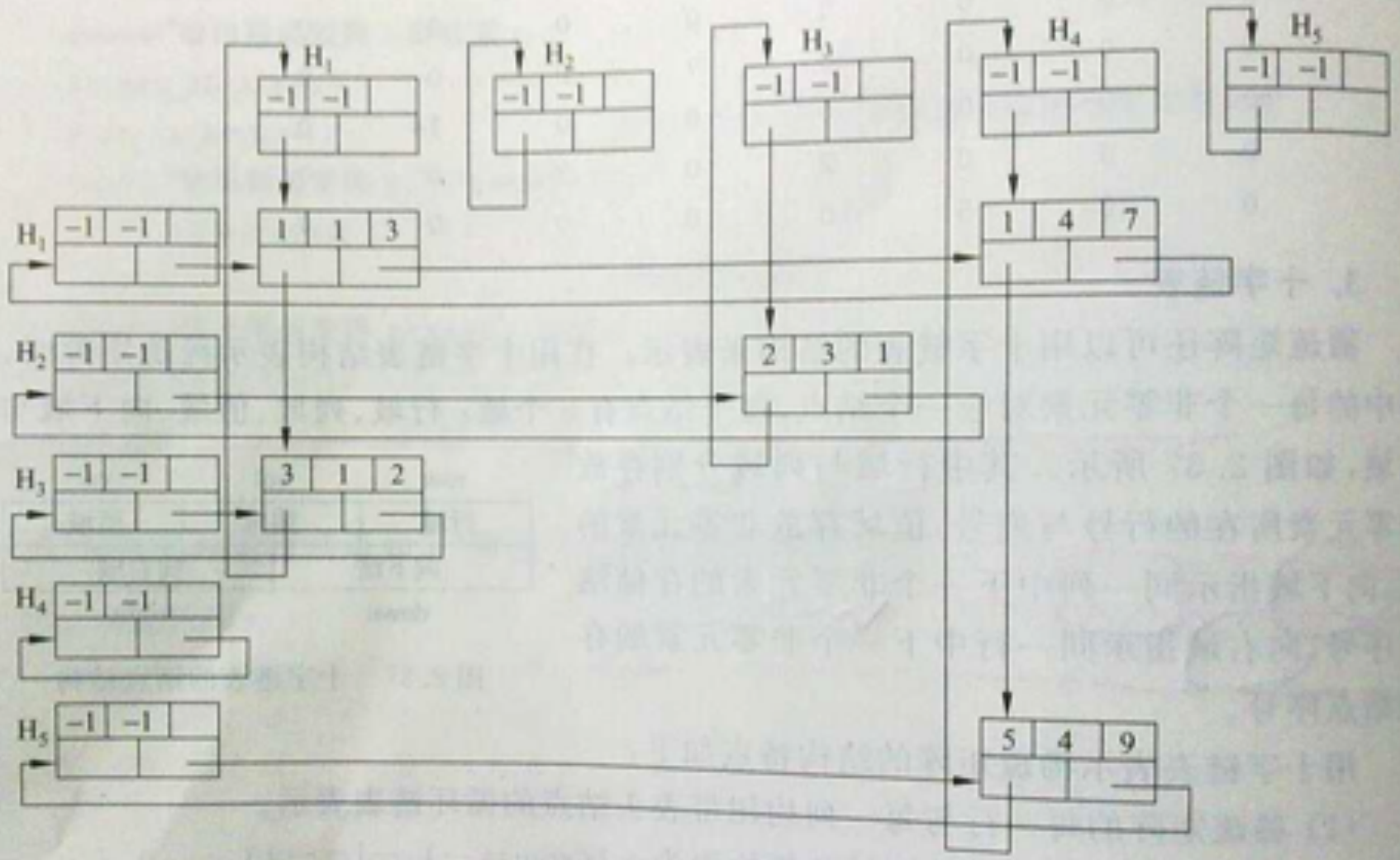


图 2.38　十字链表例