



第4章 队列

- **ADT队列**
- 用指针实现队列
- 用循环数组实现队列
- 队列的应用

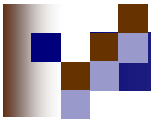


■ 4.1 ADT队列(Queue)

- 队列是另一种特殊的表，也是操作受限的线性表。
- 定义：限定只能在表的一端进行插入，在表的另一端进行删除的线性表。
 - 队尾(rear)——允许插入的一端
 - 队头(front)——允许删除的一端
- 特点：先进先出（FIFO）
- 例如：排队上车。

Bus Stop Queue





Bus Stop Queue



front

rear



Bus Stop Queue



front



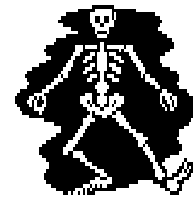
rear



Bus Stop Queue



front

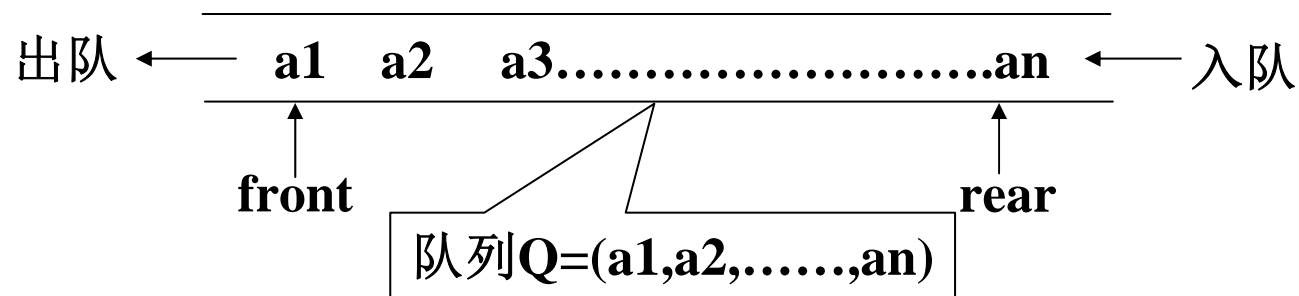


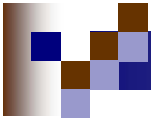
rear



■ 队列的操作示例

假设队列 $Q=(a_1, a_2, \dots, a_n)$ ，则 a_1 就是队首元素， a_n 为队尾元素。队列中的元素按 a_1, a_2, \dots, a_n 的次序进入，退出队列也只能按照这个次序依次退出。





- **ADT队列上定义的常用的基本运算**
 - **QueueEmpty(Q):** 测试队列**Q**是否为空
 - **QueueFull(Q):**测试队列**Q**是否已满
 - **QueueFirst(Q):** 返回队列**Q**的队首元素
 - **QueueLast(Q):** 返回队列**Q**的队尾元素
 - **EnterQueue(x, Q):**在队列**Q**的队尾插入元素**x**
 - **DeleteQueue(Q):**删除并返回队列**Q**的队首元素



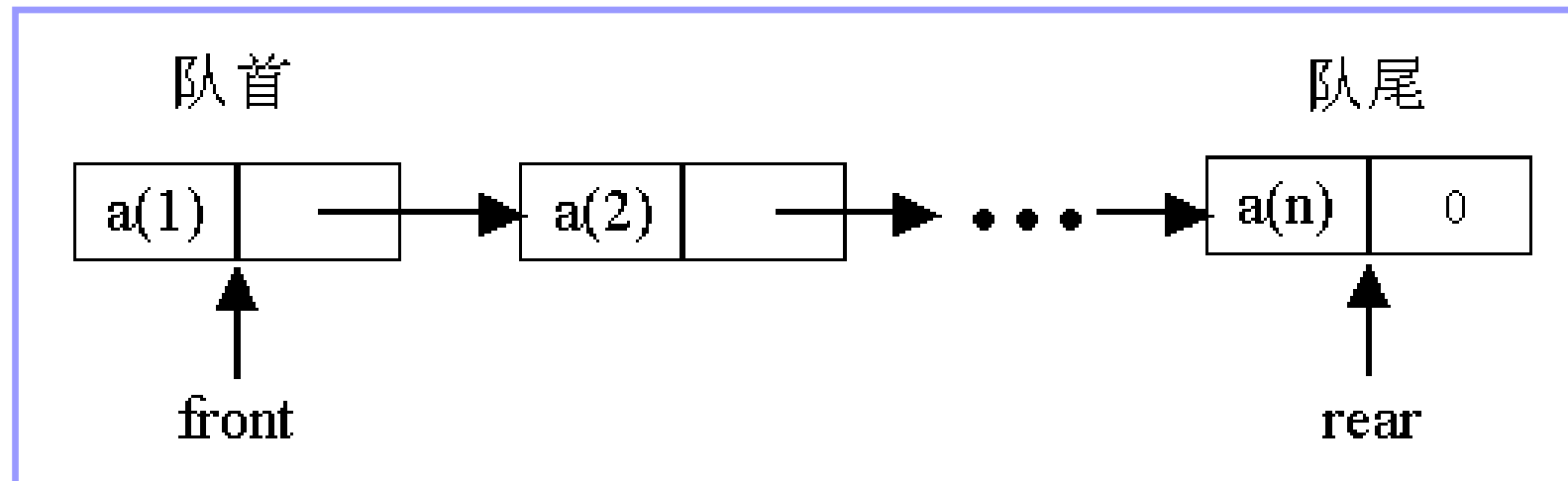
■ 4.2 用指针实现队列


- 与栈的情形相同，任何一种实现表的方法都可以用于实现队列。用指针实现队列实际上得到一个单链表。
- 队列结点的类型与单链表结点类型相同。

```
typedef struct qnode *qlink;  
typedef struct qnode {  
    QItem element;  
    qlink    next;  
} QNode;
```

- 用指针实现的队列**Queue**定义如下:

```
typedef struct lque *Queue;  
typedef struct lque  
{  
    qlink front;    //队首结点指针  
    qlink rear;     //队尾结点指针  
}Lqueue;
```





❧ 队列上的基本运算的实现

1、创建空队列

将队首指针**front**和队尾指针**rear**均置为空指针，创建一个空队列，实现方法如下：

```
Queue QueueInit ( )  
{  
    Queue Q=malloc(sizeof *Q);  
    Q->front=Q->rear=0;  
    return Q;  
}
```



2、判断队列是否为空

检测**front**是否为空指针，实现方法如下：

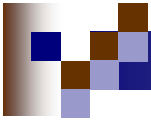
```
int QueueEmpty (Queue Q)
{
    return Q->front==0;
}
```



3、判断队列是否已满

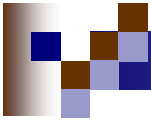
为队列**Q**试分配一个新结点，检测队列空间是否已满。

```
int QueueFull (Queue Q)
{
    return QMemFull( );
}
int QMemFull( )
{
    qlink p;
    if((p=malloc(sizeof(QNode)))==0    return 1;
    else { free(p); return 0; }
}
```



4、返回队首结点中的元素

```
QItem QueueFirst (Queue Q)
{
    if(QueueEmpty(Q))    Error("Queue is empty");
    else    return Q->front->element;
}
```



5、返回队尾结点中的元素

```
QItem QueueLast (Queue Q)
{
    if(QueueEmpty(Q))    Error("Queue is empty");
    else    return Q->rear->element;
}
```

6、在队列Q的队尾插入元素x

先为元素x创建一个新结点，然后修改队列Q的队尾结点指针，在队尾插入新结点，使新结点成为新队尾结点。

```
void EnterQueue (QItem x, Queue Q)
{
    qlink p;
    p=NewQNode( );
    p->element=x;
    p->next=0;
    if(Q->front)    Q->rear->next=p;
    else Q->front=p;
    Q->rear=p;
}
```


7、删除并返回队列Q的队首元素

先将队首元素存于x中，然后修改队列Q的队首结点指针，使其指向队首结点的下一个结点，从而删除队首结点，最后返回x。

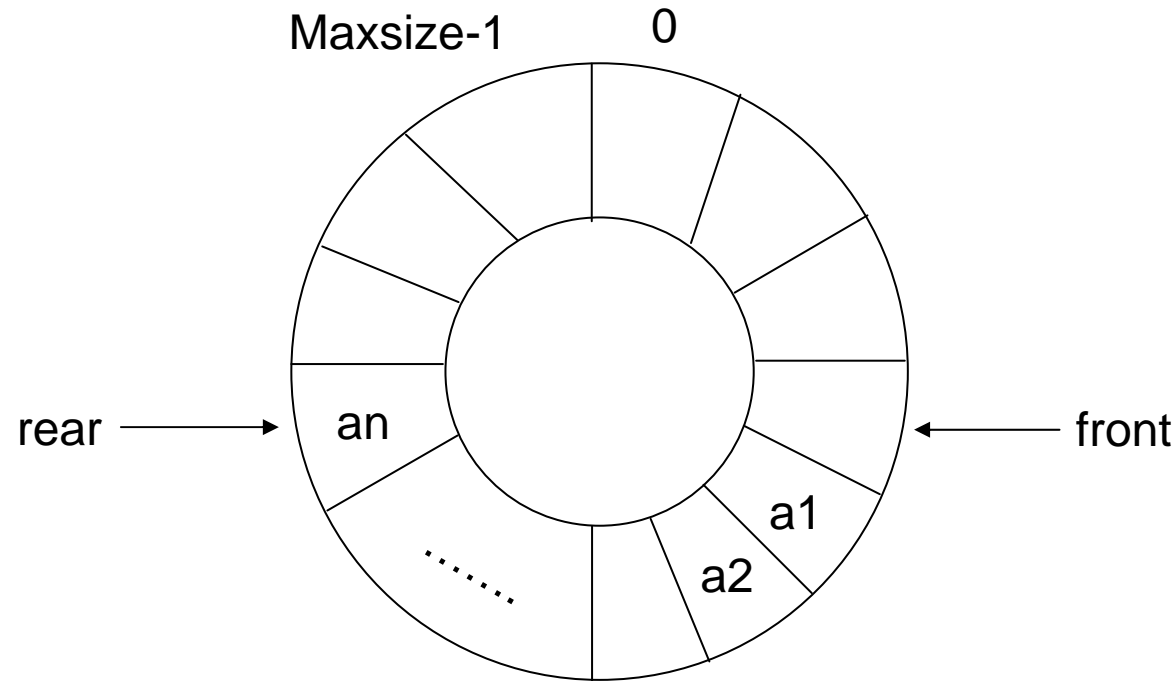
```
QItem DeleteQueue(Queue Q)
{
    qlink p;    QItem x;
    if(QueueEmpty(Q))    Error("Queue is empty");
    x=Q->front->element;
    p=Q->front;
    Q->front=Q->front->next;
    free(p);
    return x;
}
```



■ 4.3 用循环数组实现队列

- 用数组也可以实现队列，但效果不理想。尽管可以用一个游标来指示队尾，使得**EnterQueue**运算在 **$O(1)$** 时间内完成，但在执行**DeleteQueue**时，为了删除队首元素，必须将数组中其他所有元素都前移一个位置，很浪费时间。
- 为提高运算效率，可采用另一种观点来处理数组中各单元的位置关系。

- 设想数组`queue[0:maxsize-1]`中的单元不是排成一行，而是围成一个首尾相接的圆环，即`queue[0]`接在`queue[maxsize-1]`的后面。这种意义下的数组称为循环数组。





■ 用循环数组实现队列

- 将队列中从队首到队尾的元素按顺时针方向存放在循环数组的一段连续的单元中。

- 实现：利用“模”运算

- 入队：将队尾游标**rear**按顺时针方向移一位，并在此单元中存入新元素。

$\text{rear}=(\text{rear}+1)\% \text{Maxsize}; \quad \text{queue}[\text{rear}]=\text{x};$

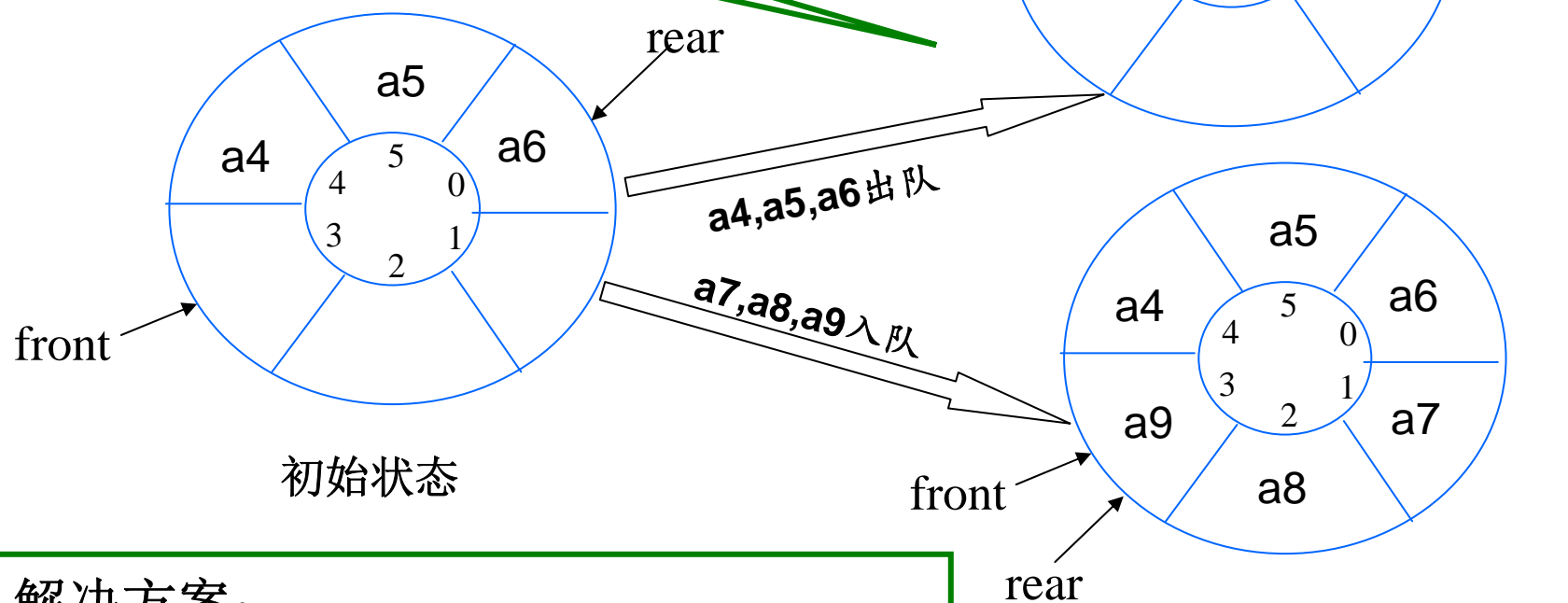
- 出队：将队首游标**front**按顺时针方向移一位即可。

$\text{front}=(\text{front}+1)\% \text{Maxsize}; \quad \text{x}=\text{queue}[\text{front}];$

- 队满、队空判定条件

队空: $\text{front} == \text{rear}$

队满: $\text{front} == \text{rear}$



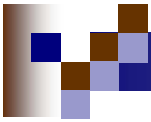
解决方案:

1. 另外设一个布尔量区别队空、队满

2. 少用一个元素空间:

队空: $\text{front} == \text{rear}$

队满: $(\text{rear} + 1) \% \text{Maxsize} == \text{front}$



- 用循环数组实现的队列**Queue**定义如下:

```
typedef struct aqueue *Queue;
typedef struct aqueue
{
    int maxsize;    //循环数组大小
    int front;      //队首游标
    int rear;       //队尾游标
    QItem *queue;   //循环数组
} Aqueue;
```



❧ 用循环数组实现队列上的基本运算

1、创建空队列

为队列分配一个容量为**size**的循环数组，并将队首游标**front**和队尾游标**rear**均置为**0**，创建一个空队列。

```
Queue QueueInit (int size)
{
    Queue Q=malloc(sizeof *Q);
    Q->queue=malloc (size*sizeof(QItem));
    Q->maxsize=size;
    Q->front=Q->rear=0;
    return Q;
}
```



2、判断队列是否为空

通过检测队列**Q**的队首游标**front**与队尾游标**rear**是否重合来判断队列**Q**是否为空队列。

```
int QueueEmpty (Queue Q)
{
    return Q->front==Q->rear;
}
```




3、判断队列是否已满

通过在队列**Q**的队尾插入一个元素后队首游标**front**与队尾游标**rear**是否重合，来判断队列**Q**是否为满队列。

```
int QueueFull (Queue Q)
{
    return (((Q->rear+1) % Q->maxsize == Q->front)?1:0);
}
```



4、返回队列Q的队首元素

由于队首游标**front**指向队首元素的前一位置，所以队首元素在循环数组**queue**中的下标是 **$(front+1)\%maxsize$** 。

```
QItem QueueFirst (Queue Q)
{
    if(QueueEmpty(Q))    Error("Queue is empty");
    else
        return Q->queue[(Q->front+1) % Q->maxsize];
}
```



5、返回队列Q的队尾元素

```
QItem QueueLast (Queue Q)
{
    if(QueueEmpty(Q))    Error("Queue is empty");
    else    return Q->queue[Q->rear];
}
```

6、在队列Q的队尾插入元素x

先计算出在循环的意义队列Q的队尾元素在循环数组 **queue** 中的下一个位置 **(rear+1)%maxsize**，然后在该位置插入元素x。

```
void EnterQueue (QItem x, Queue Q)
{
    if(QueueFull(Q))    Error("Queue is Full");
    Q->rear=(Q->rear+1) % Q->maxsize;
    Q->queue[Q->rear]=x;
}
```

7、删除并返回队列Q的队首元素

先将队首游标**front**修改为在循环意义下队首元素在循环数组**queue**中的下一个位置 **$(front+1)\%maxsize$** ，然后返回该位置处的元素。

```
QItem DeleteQueue(Queue Q)
{
    if(QueueEmpty(Q))    Error("Queue is empty");
    Q->front=(Q->front+1) % Q->maxsize;
    return Q->queue[Q->front];
}
```



■ 4.4 队列的应用举例

■ 例1 模拟打印机缓冲区

在主机将数据输出到打印机时，会出现主机速度与打印机的打印速度不匹配的问题。这时主机就要停下来等待打印机。显然，这样会降低主机的使用效率。为此人们设想了一种办法：为打印机设置一个打印数据缓冲区，当主机需要打印数据时，先将数据依次写入这个缓冲区，写满后主机转去做其他的事情，而打印机就从缓冲区中按照先进先出的原则依次读取数据并打印，这样做即保证了打印数据的正确性，又提高了主机的使用效率。由此可见，打印机缓冲区实际上就是一个队列结构。



■ 4.4 队列的应用举例

■ 例2 CPU分时系统

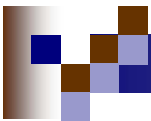
在一个带有多个终端的计算机系统中，同时有多个用户需要使用**CPU**运行各自的应用程序，它们分别通过各自的终端向操作系统提出使用**CPU**的请求，操作系统通常按照每个请求在时间上的先后顺序，将它们排成一个**队列**，每次把**CPU**分配给**当前队首**的请求用户，即将该用户的应用程序投入运行，当该程序运行完毕或用完规定的时间片后，操作系统再将**CPU**分配给新的队首请求用户，这样即可以满足每个用户的请求，又可以使**CPU**正常工作。



■ 例3 划分子集问题

问题描述：已知集合 $A=\{a_1, a_2, \dots, a_n\}$ ，及集合上的关系 $R=\{ (a_i, a_j) \mid a_i, a_j \in A, i \neq j \}$ ，其中 (a_i, a_j) 表示 a_i 与 a_j 间存在冲突关系。要求将 A 划分成互不相交的子集 $A_1, A_2, \dots, A_k, (k \leq n)$ ，使任何子集中的元素均无冲突关系，同时要求分子集个数尽可能少。

例 $A=\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
 $R=\{ (2, 8), (9, 4), (2, 9), (2, 1), (2, 5), (6, 2), (5, 9),$
 $(5, 6), (5, 4), (7, 5), (7, 6), (3, 7), (6, 3) \}$
可行的子集划分为：
 $A_1=\{ 1, 3, 4, 8 \}$
 $A_2=\{ 2, 7 \}$
 $A_3=\{ 5 \}$
 $A_4=\{ 6, 9 \}$



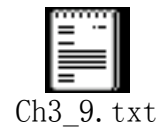
- 算法思想：利用**循环筛选**。从第一个元素开始，凡与第一个元素无冲突的元素划归一组；再将剩下的元素重新找出互不冲突的划归第二组；直到所有元素进组。
- 所用数据结构
 - 冲突关系矩阵
 - $r[i][j]=1$, i,j 有冲突
 - $r[i][j]=0$, i,j 无冲突
 - 循环队列**cq[n]**
 - 数组**result[n]**存放每个元素分组号
 - 工作数组**newr[n]**



■ 工作过程

- 初始状态: **A**中元素放于**cq**中, **result**和**newr**数组清零, 组号**group=1**
- 第一个元素出队, 将**r**矩阵中第一行“1”拷入**newr**中对应位置, 这样, 凡与第一个元素有冲突的元素在**newr**中对应位置处均为“1”, 下一个元素出队
 - 若其在**newr**中对应位置为“1”, 有冲突, 重新插入**cq**队尾, 参加下一次分组
 - 若其在**newr**中对应位置为“0”, 无冲突, 可划归本组; 再将**r**矩阵中该元素对应行中的“1”拷入**newr**中
- 如此反复, 直到9个元素依次出队, 由**newr**中为“0”的单元对应的元素构成第1组, 将组号**group**值“1”写入**result**对应单元中
- 令**group=2, newr**清零, 对**cq**中元素重复上述操作, 直到**cq**中**front==rear**, 即队空, 运算结束

■ 算法描述



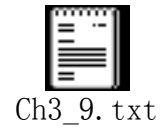
$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	1	2	3	4	5	6	7	8	9
初始	0	1	2	3	4	5	6	7	8
newr	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8
result	0	0	0	0	0	0	0	0	0



■ 算法描述

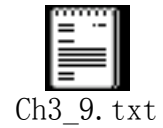


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq		2	3	4	5	6	7	8	9
	f								r
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8
result	1	0	0	0	0	0	0	0	0

■ 算法描述

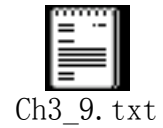


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	2		3	4	5	6	7	8	9
	\uparrow r	\uparrow f							
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8
result	1	0	0	0	0	0	0	0	0

■ 算法描述

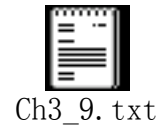


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	2			4	5	6	7	8	9
	\uparrow r		\uparrow f						
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	0	1	1	0	0
	0	1	2	3	4	5	6	7	8
result	1	0	1	0	0	0	0	0	0

■ 算法描述

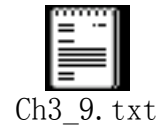


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	2				5	6	7	8	9
	\uparrow		\uparrow						
	r		f						
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	1	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	0	1	1	0	0	0	0	0

■ 算法描述

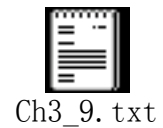


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	2	5				6	7	8	9
		\uparrow r			\uparrow f				
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	1	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	0	1	1	0	0	0	0	0

■ 算法描述

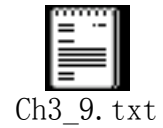


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	2	5	6				7	8	9
			\uparrow r			\uparrow f			
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	1	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	0	1	1	0	0	0	0	0

■ 算法描述

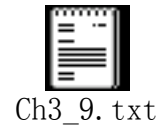


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	2	5	6	7				8	9
				↑ r			↑ f		
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	1	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	0	1	1	0	0	0	0	0

■ 算法描述

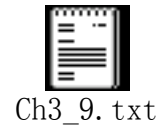


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	2	5	6	7					9
				↑ r				↑ f	
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	1	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	0	1	1	0	0	0	1	0

■ 算法描述

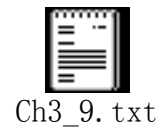


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	2	5	6	7	9				
				\uparrow r					\uparrow f
	0	1	2	3	4	5	6	7	8
newr	0	1	0	0	1	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	0	1	1	0	0	0	1	0

■ 算法描述



$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq		5	6	7	9				
	f				r				
	0	1	2	3	4	5	6	7	8
newr	1	0	0	0	1	1	0	1	1
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	0	0	0	1	0

■ 算法描述



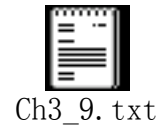
Ch3_9.txt

$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq			6	7	9	5			
		↑ f				↑ r			
	0	1	2	3	4	5	6	7	8
newr	1	0	0	0	1	1	0	1	1
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	0	0	0	1	0

■ 算法描述

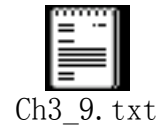


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq				7	9	5	6		
			↑ f				↑ r		
	0	1	2	3	4	5	6	7	8
newr	1	0	0	0	1	1	0	1	1
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	0	0	0	1	0

■ 算法描述

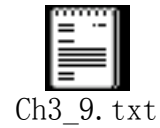


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq					9	5	6		
				f		r			
	0	1	2	3	4	5	6	7	8
newr	1	0	1	0	1	1	0	1	1
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	0	0	2	1	0

■ 算法描述

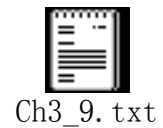


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq						5	6	9	
					f			r	
	0	1	2	3	4	5	6	7	8
newr	1	0	1	0	1	1	0	1	1
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	0	0	2	1	0

■ 算法描述

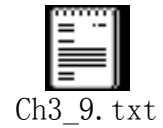


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq							6	9	
						f		r	
	0	1	2	3	4	5	6	7	8
newr	0	1	0	1	0	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	3	0	2	1	0

■ 算法描述

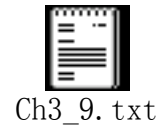


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq								9	6
							↑		↑
							f		r
	0	1	2	3	4	5	6	7	8
newr	0	1	0	1	0	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	3	0	2	1	0

■ 算法描述

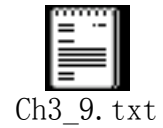


$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	9								6
	↑							↑	
	r							f	
	0	1	2	3	4	5	6	7	8
newr	0	1	0	1	0	1	1	0	1
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	3	0	2	1	0

■ 算法描述



$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq	9								
	↑							↑	
	r							f	
	0	1	2	3	4	5	6	7	8
newr	0	1	1	0	1	0	1	0	0
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	3	4	2	1	0

■ 算法描述

Ch3_9.txt

可行的子集划分为:

$A1 = \{ 1, 3, 4, 8 \}$

$A2 = \{ 2, 7 \}$

$A3 = \{ 5 \}$

$A4 = \{ 6, 9 \}$

$R = \{ (2,8), (9,4), (2,9), (2,1), (2,5), (6,2), (5,9), (5,6), (5,4), (7,5), (7,6), (3,7), (6,3) \}$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8
cq									
	0	1	2	3	4	5	6	7	8
newr	0	1	1	1	1	0	1	0	0
	0	1	2	3	4	5	6	7	8
result	1	2	1	1	3	4	2	1	4



本章小结

- 理解队列是满足**FIFO**存取原则的表。
- 熟悉定义在抽象数据类型队列上的基本运算。
- 掌握用指针实现队列的步骤和方法。
- 掌握用循环数组实现栈的步骤和方法。
- 理解用队列解决实际问题的方法。