



第三章 栈和队列

逻辑结构和线性表相同

运算受到了限制

根据所受限制的不同，分为栈和队列



栈----线性结构，插入和删除操作受限制

- 栈是限制在表的一端（表尾）进行插入和删除的线性表
- $(a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_{n-1}, a_n)$

栈顶

出栈

插入顺序

进栈

删除顺序

特点—先进后出



栈

■ 栈的主要操作:

1. 初始化空栈
2. 入栈
3. 出栈
4. 判断栈是否为空栈
5. 取栈顶数据元素

■ 栈的存储方式

1. 顺序存储结构----顺序栈
2. 链式存储结构----链栈

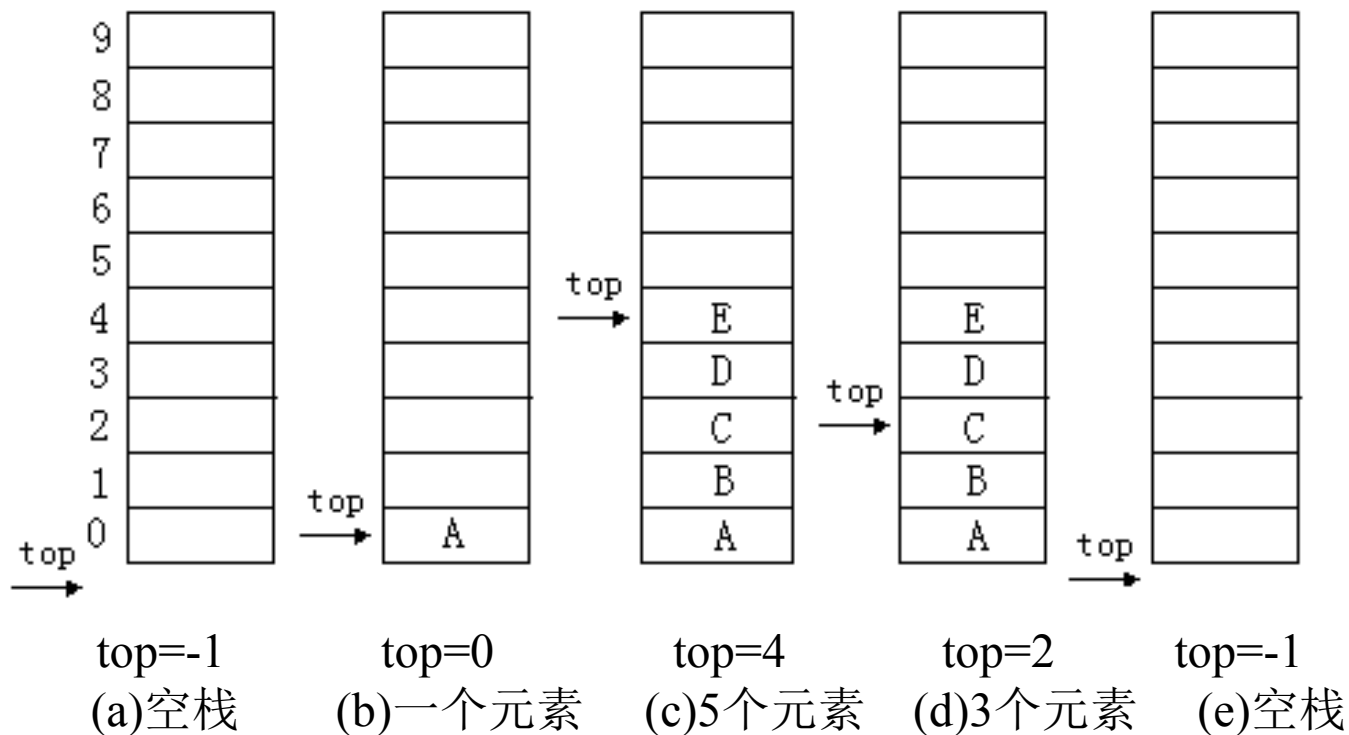


栈----顺序存储结构

- 定义：是用一组地址**连续**的存储单元**依次存储**栈中的每一个数据元素
- 实现：数组
- **说明：栈的主要操作均在栈顶进行，因此需要保存栈顶位置以方便操作进行**
- 栈顶位置的保存常见2种方法：
 1. 实际栈顶位置--栈顶元素在在数组中的位置，即数组下标
 2. 下一次入栈的位置--即下一次插入到“下标为多少的”数组元素
- 保存栈顶位置的变量称为**栈顶指针**

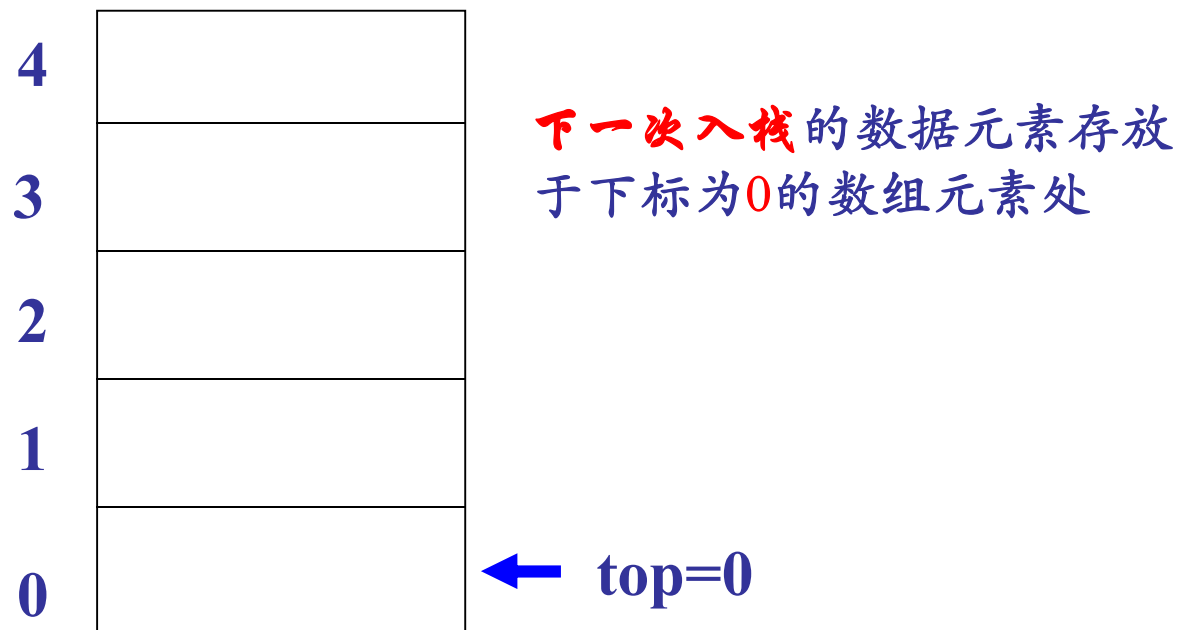
栈---顺序存储结构

常见方法1: **栈顶指针**指示栈顶的**实际位置**



栈---顺序存储结构

常见方法2: 栈顶指针指示下一次入栈的位置



空栈

栈---顺序存储结构

常见方法2: **栈顶指针**指示**下一次入栈**的位置



栈中有2个数据元素



栈----顺序存储结构

```
#define MAXSIZE 1024  
  
typedef struct  
{elemtype data[MAXSIZE];  
  int top;//栈顶指针  
}SeqStack;
```

说明： **elemtype**代表栈中数据元素的类型，具体应用时，栈中的数据元素是什么类型的，就将**elemtype**定义成相应类型



栈----顺序存储结构

```
#define MAXSIZE 5  
  
typedef struct  
{elemtype data[MAXSIZE];  
    int top; // 栈顶指针  
}SeqStack;  
  
SeqStack s ;
```

说明： elemtype代表栈中数据元素的类型，具体应用时，栈中的数据元素是什么类型的，就将elemtype定义成相应类型

说明： 为了方便演示栈操作，将MAXSIZE定义成5



栈----顺序存储结构

```
#define MAXSIZE 5
```

```
typedef struct
```

```
{int data[MAXSIZE];
```

```
int top; //栈顶指针
```

```
}SeqStack;
```

```
SeqStack s ;
```

说明：为了方便演示栈操作，假设栈中的数据元素为整型的

说明：elemtype代表栈中数据元素的类型，具体应用时，栈中的数据元素是什么类型的，就将elemtype定义成相应类型

说明：为了方便演示栈操作，将MAXSIZE定义成5

栈顶指针指示下一次入栈的位置

初始化: $s.top = -1$;

顺序栈

入栈: 只要有空间则

$s.data[++s.top] = x$;

$s.data[4]$

$s.data[3]$

$s.data[2]$

$s.data[1]$

$s.data[0]$

$s.top = -1$

栈顶指针指示下一次入栈的位置

初始化: $s.top = -1$;

顺序栈

入栈: 只要有空间则

$s.data[++s.top] = x$;

入栈 ($x = a_1$)

$s.data[4]$

$s.data[3]$

$s.data[2]$

$s.data[1]$

$s.data[0]$



← $s.top = 0$

栈顶指针指示下一次入栈的位置

初始化: $s.top = -1$;

顺序栈

入栈: 只要有空间则

$s.data[++s.top] = x$;

入栈 ($x = a_1$)

入栈 ($x = a_2$)

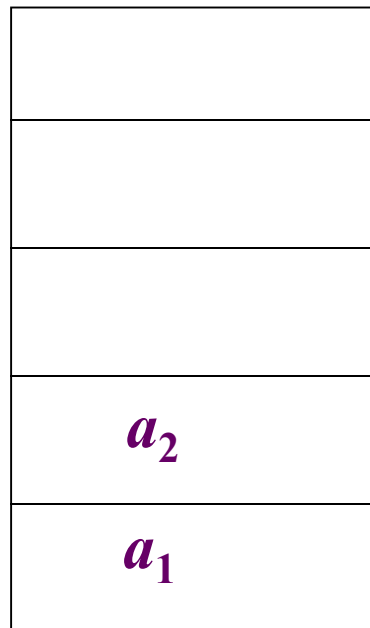
$s.data[4]$

$s.data[3]$

$s.data[2]$

$s.data[1]$

$s.data[0]$



← $s.top = 1$

栈顶指针指示下一次入栈的位置

初始化: $s.top = -1$;

顺序栈

入栈: 只要有空间则

$s.data[++s.top] = x$;

$s.data[4]$

$s.data[3]$

$s.data[2]$

$s.data[1]$

$s.data[0]$

a_3
a_2
a_1

← $s.top = 2$

入栈 ($x = a_1$)

入栈 ($x = a_2$)

入栈 ($x = a_3$)

栈顶指针指示下一次入栈的位置

初始化: $s.top = -1$;

顺序栈

入栈: 只要有空间则

$s.data[++s.top] = x$;

$s.data[4]$

$s.data[3]$

$s.data[2]$

$s.data[1]$

$s.data[0]$

a_4
a_3
a_2
a_1

← $s.top = 3$

入栈 ($x = a_1$)

入栈 ($x = a_2$)

入栈 ($x = a_3$)

入栈 ($x = a_4$)

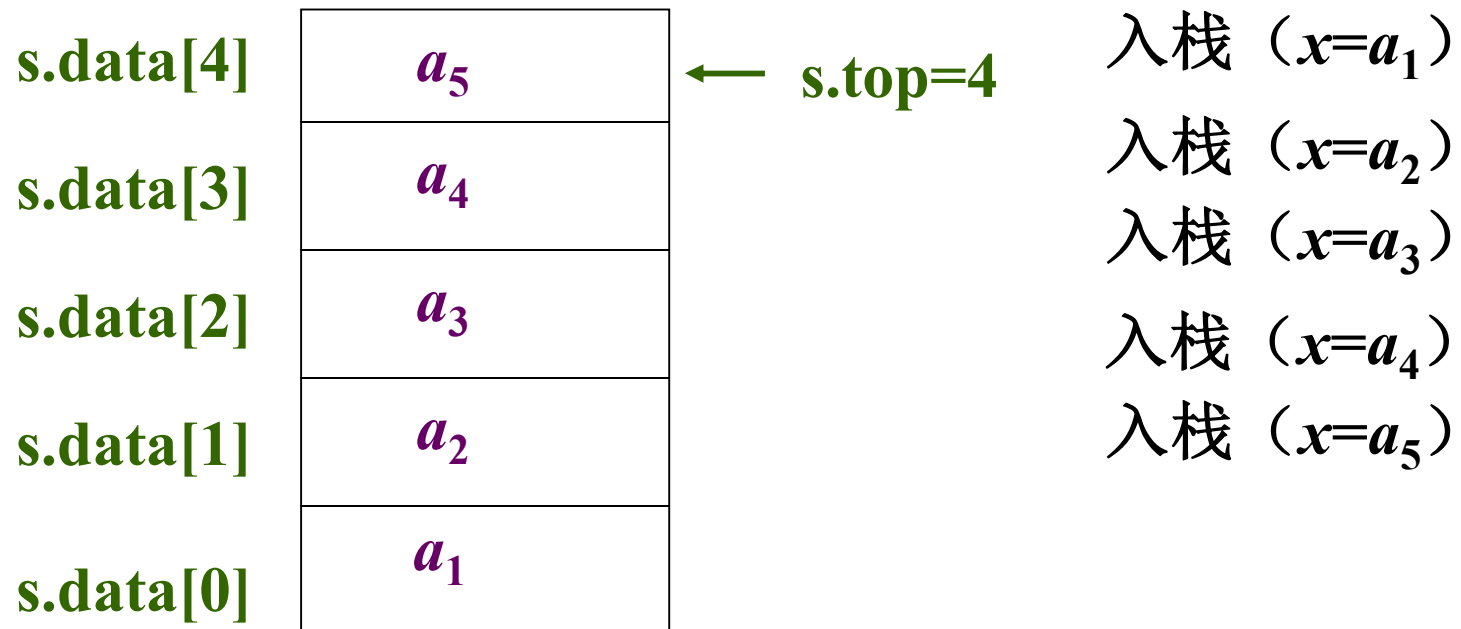
栈顶指针指示下一次入栈的位置

初始化: $s.top = -1$;

顺序栈

入栈: 只要有空间则

$s.data[++s.top] = x$;



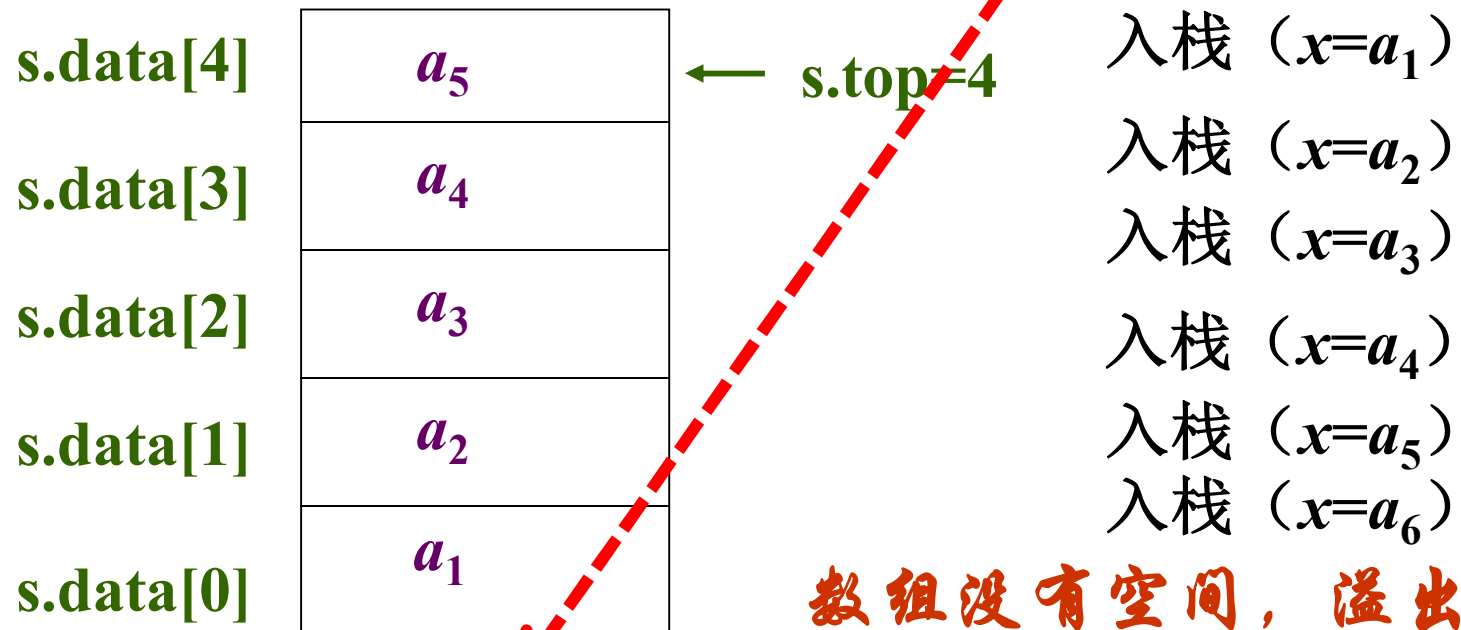
栈顶指针指示下一次入栈的位置

初始化: $s.top = -1$;

顺序栈

入栈: 只要有空间则

$s.data[++s.top] = x$;



$if(s.top < MAXSIZE - 1)$
 $s.data[++s.top] = x$;
else $printf("overflow");$

栈顶指针指示下一次入栈的位置

只要非空栈 $s.top--$;

顺序栈—出栈

$s.data[4]$	a_5	← $s.top=4$
$s.data[3]$	a_4	
$s.data[2]$	a_3	
$s.data[1]$	a_2	
$s.data[0]$	a_1	

栈顶指针指示下一次入栈的位置

只要非空栈 $s.top--$;

顺序栈—出栈

$s.data[4]$

a_5

$s.data[3]$

a_4

$s.data[2]$

a_3

$s.data[1]$

a_2

$s.data[0]$

a_1

← $s.top=3$

栈顶指针指示下一次入栈的位置

只要非空栈 $s.top--$;

顺序栈—出栈

$s.data[4]$	a_5	$\leftarrow s.top=2$
$s.data[3]$	a_4	
$s.data[2]$	a_3	
$s.data[1]$	a_2	
$s.data[0]$	a_1	

栈顶指针指示下一次入栈的位置

顺序栈—入栈 ($x=a_6$)

s.data[4]	a_5	← s.top=2
s.data[3]	a_4	
s.data[2]	a_3	
s.data[1]	a_2	
s.data[0]	a_1	

栈顶指针指示下一次入栈的位置

顺序栈—入栈 ($x=a_6$)

s.data[4]	a_5	← s.top=3
s.data[3]	a_6	
s.data[2]	a_3	
s.data[1]	a_2	
s.data[0]	a_1	

栈顶指针指示下一次入栈的位置

只要非空栈 $s.top--$;

顺序栈—出栈

$s.data[4]$	a_5	$\leftarrow s.top=2$
$s.data[3]$	a_6	
$s.data[2]$	a_3	
$s.data[1]$	a_2	
$s.data[0]$	a_1	

栈顶指针指示下一次入栈的位置

只要非空栈 $s.top--$;

顺序栈—出栈

$s.data[4]$

a_5

$s.data[3]$

a_6

$s.data[2]$

a_3

$s.data[1]$

a_2

$s.data[0]$

a_1

← $s.top=1$

栈顶指针指示下一次入栈的位置

只要非空栈 $s.top--$;

顺序栈—出栈

$s.data[4]$	a_5
$s.data[3]$	a_6
$s.data[2]$	a_3
$s.data[1]$	a_2
$s.data[0]$	a_1

← $s.top=0$

栈顶指针指示下一次入栈的位置

顺序栈—出栈

只要非空栈 $s.top--$;

$if(s.top \geq 0) \ s.top--$;

$s.data[4]$

a_5

$s.data[3]$

a_6

$s.data[2]$

a_3

$s.data[1]$

a_2

$s.data[0]$

a_1

对空栈进行删除称为下溢出

← $s.top = -1$



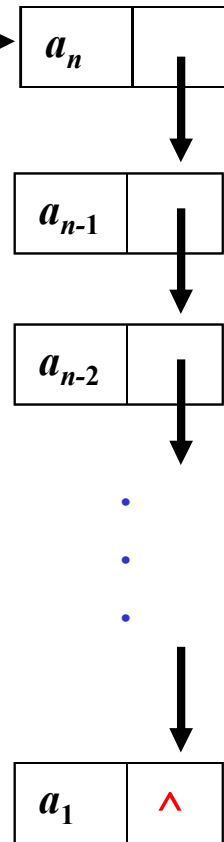
顺序栈

- 初始化: `s.top=-1;`
- 入栈: `if(s.top<MAXSIZE-1)`
`s.data[++s.top]=x;`
`else printf(“overflow”);`
- 出栈: `if(s.top>=0)`
`s.top--;else{...}`
- 判栈空: `s.top==-1`
- 判栈满: `s.top==MAXSIZE-1`
- 栈顶元素: `s.data[s.top]`

采用链式存储结构存放--链栈

top

data next

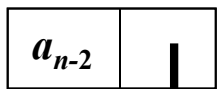
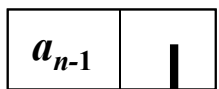
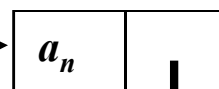


- 采用不带表头结点的单链表存放栈
- 根据栈的定义，为操作方便，每个数据元素对应结点空间的指针存放该数据元素的直接前驱
- 保留栈顶的位置即可----单链表的头指针对应为栈顶的位置
- 出栈在栈顶进行----删除单链表的第一个数据元素结点
- 入栈在栈顶进行----在单链表的第一个数据元素结点前插入一个新的数据元素

链栈

top

data next



.

.

.



```
■ typedef struct node{  
    int data ;  
    struct node *next; //直接前驱  
}Node, *LinkList;
```

```
■ LinkList top;
```

```
■ //出栈
```

```
➤ if (top)
```

```
{p=top; top=top->next; free(p);}
```

```
■ //入栈——将x入栈
```

```
➤ s=(LinkList)malloc(sizeof(Node));
```

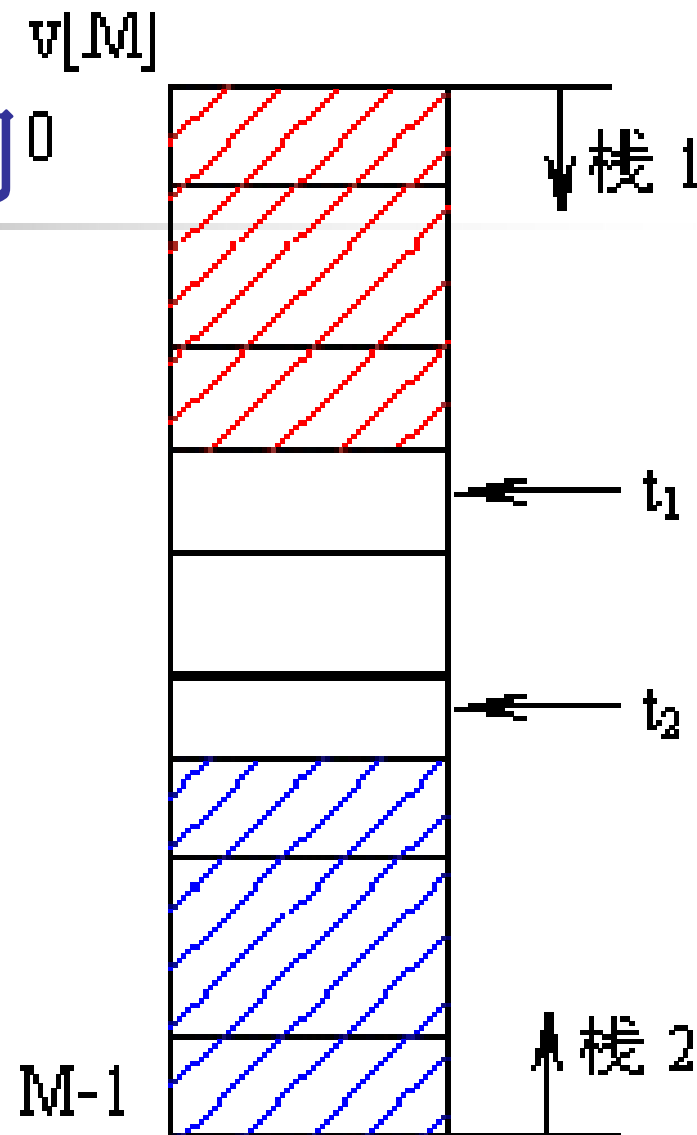
```
➤ s->data=x;
```

```
➤ s->next=top;
```

```
➤ top=s;
```

双栈共享存储空间

- 两个栈共用一个一维数组 $v[M]$ ，栈底分别设在数组的**两端**，各自向中间伸展，第一个栈自顶向下伸展，第二个栈自底向上伸展。两个栈共享存储空间，可互补空缺，使得某个栈实际可利用的空间大于 $M/2$



入队

队尾

队列

----线性结构，插入和删除操作受限制

- **限制** 插入在表一端（**表尾**）进行，而删除在表的另一端（**表头**）进行。

$(a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_{n-1}, a_n)$

队首

出队

插入顺序



删除顺序

特点—先进先出



队列

- 主要操作:

- 入队、出队、初始化空队列、求队长

- 存储方式:

- 1. 顺序存储结构-----顺序队列

- 2. 链式存储结构-----链队

- **说明:** 队列的主要操作在队首和队尾处完成, 要保留队头、队尾两个位置方便操作实现



队列的顺序存储结构

```
define MAXSIZE 5
```

```
typedef struct
```

```
{elemtype data[MAXSIZE];
```

说明1: **f**为队首指针, 指示队首位置; **r**为队尾指针, 指示队尾位置

```
int f,r; //队首和队尾指针
```

```
}SeQueue;
```

说明2: 所谓位置是指数组元素的下标

```
SeQueue q;
```

说明3: **elemtype**代表队列中数据元素的类型, 具体应用时, 队列中的数据元素是什么类型的, 就将**elemtype**定义成相应类型

队列的顺序存储结构

初始化（**建个空队列**）：

$q.f=0; q.r=0;$

入队：**只要有空间**

$q.data[q.r]=x; q.r++;$

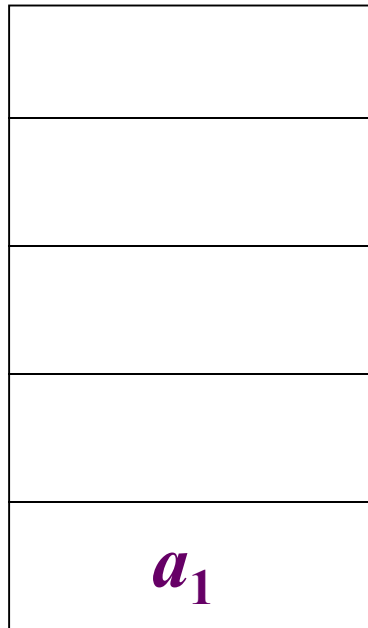
$q.data[4]$

$q.data[3]$

$q.data[2]$

$q.data[1]$

$q.data[0]$



← $q.r$
← $q.f$

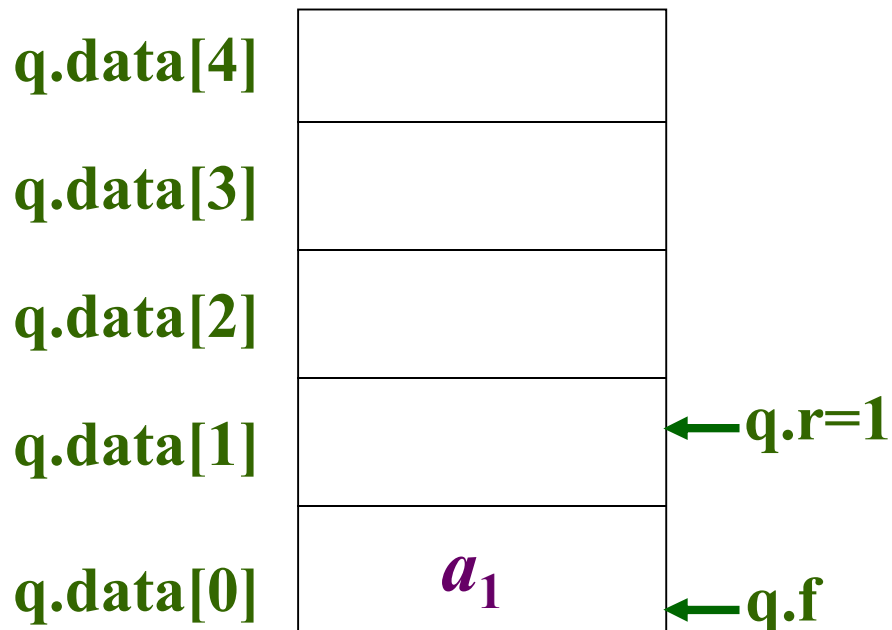
队列的顺序存储结构

初始化（建个空队列）：

$q.f=0; q.r=0;$

入队：只要有空间

$q.data[q.r]=x; q.r++;$



队列的顺序存储结构

初始化（建个空队列）：

$q.f=0; q.r=0;$

入队：只要有空间

$q.data[q.r]=x; q.r++;$

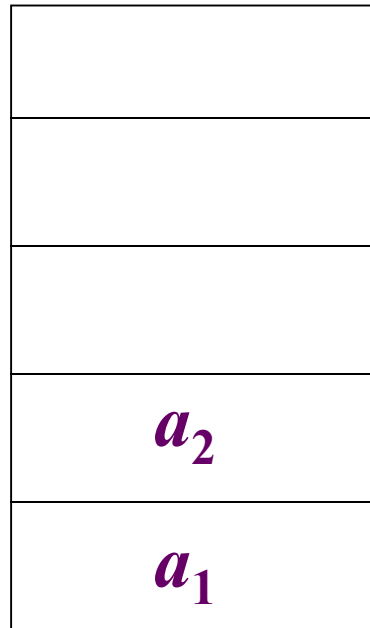
$q.data[4]$

$q.data[3]$

$q.data[2]$

$q.data[1]$

$q.data[0]$



← $q.r=2$

← $q.f$

队列的顺序存储结构

初始化（建个空队列）：

$q.f=0; q.r=0;$

入队：只要有空间

$q.data[q.r]=x; q.r++;$

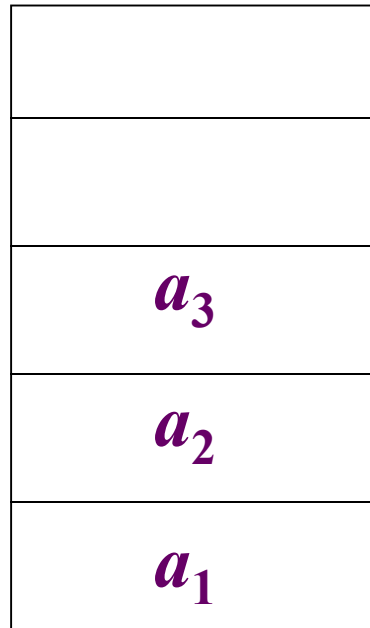
$q.data[4]$

$q.data[3]$

$q.data[2]$

$q.data[1]$

$q.data[0]$



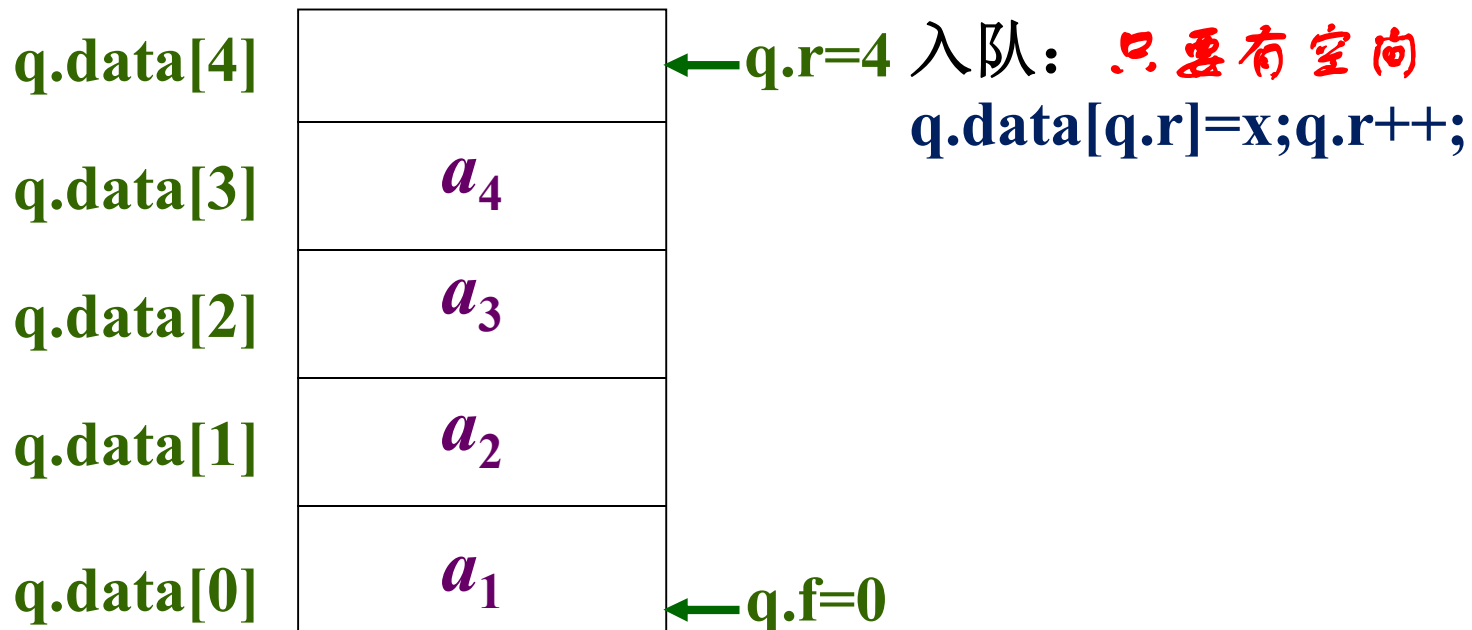
← $q.r=3$

← $q.f=0$

队列的顺序存储结构

初始化（建个空队列）：

$q.f=0; q.r=0;$



队列的顺序存储结构

初始化（建个空队列）：

← $q.r = 5$ $q.f = 0; q.r = 0;$

$q.data[4]$	a_5
$q.data[3]$	a_4
$q.data[2]$	a_3
$q.data[1]$	a_2
$q.data[0]$	a_1

入队：只要有空间

$q.data[q.r] = x; q.r++;$

再插入，无空间，溢出

有空间

$q.r < MAXSIZE$

← $q.f = 0$



队列的顺序存储结构

- 初始化: $q.f=0; q.r=0;$
- 入队: if ($q.r < \text{MAXSIZE}$)
 { $q.data[q.r]=x; q.r++;$ }
 else printf(“overflow”);

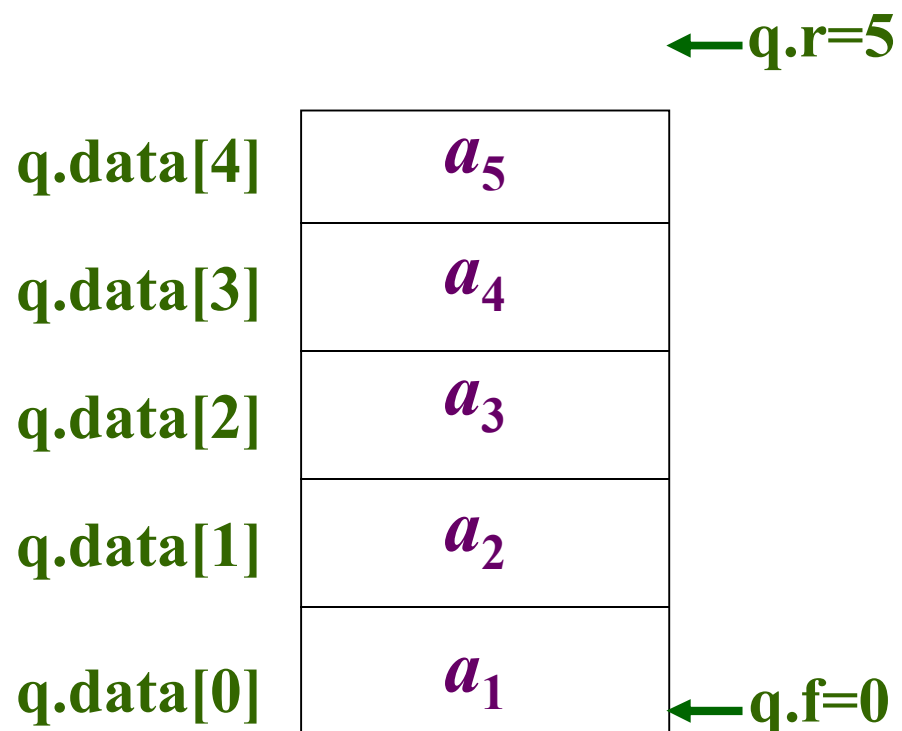


队列的顺序存储结构

- 初始化: `q.f=0;q.r=0;`
- 入队: **if (`q.r<MAXSIZE`)**
 {q.data[q.r]=x;q.r++;}
 else printf(“overflow”);
- 出队: **if (队不空)**
 q.f++;



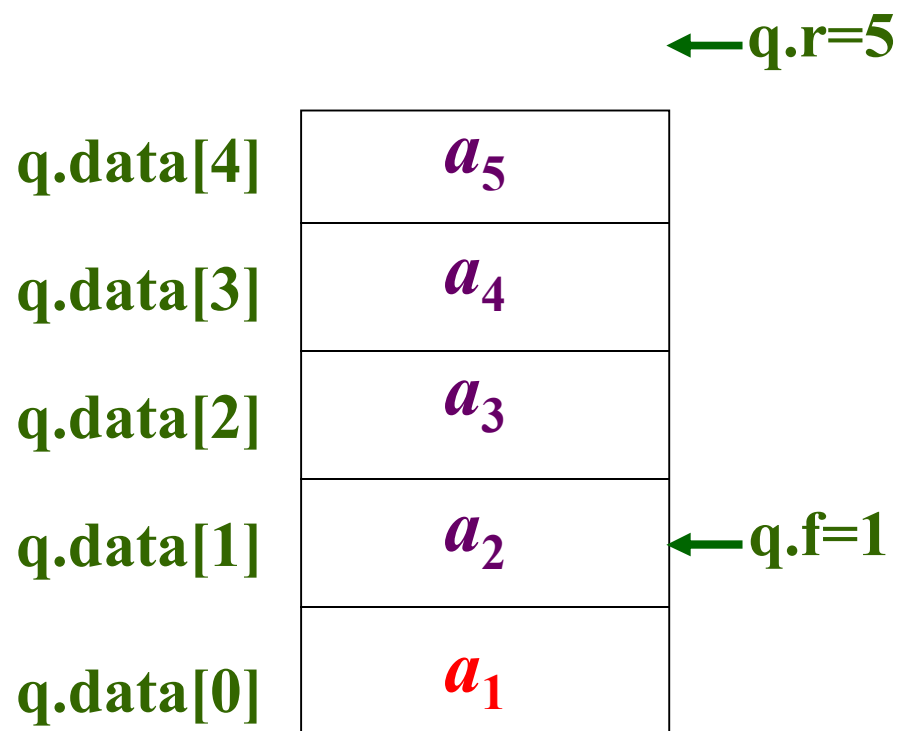
队列的顺序存储结构



出队



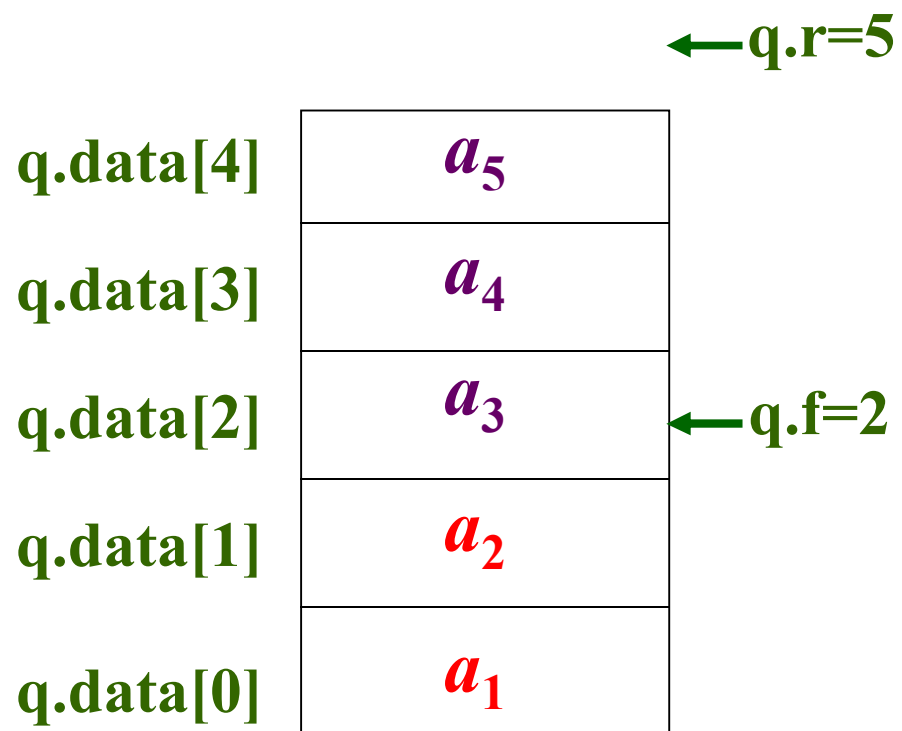
队列的顺序存储结构



出队



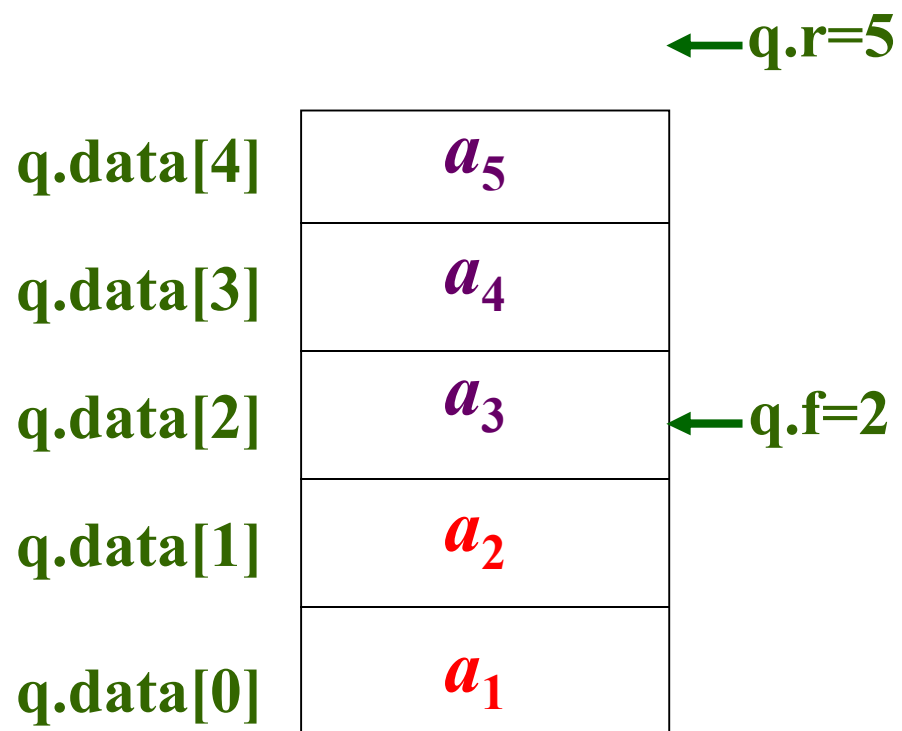
队列的顺序存储结构



出队



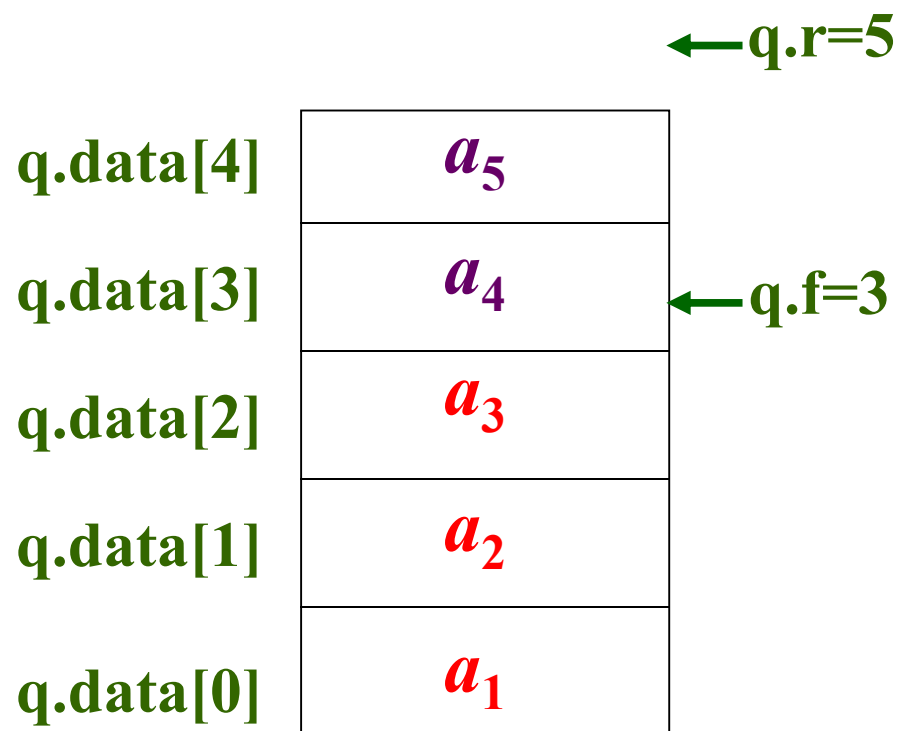
队列的顺序存储结构



出队

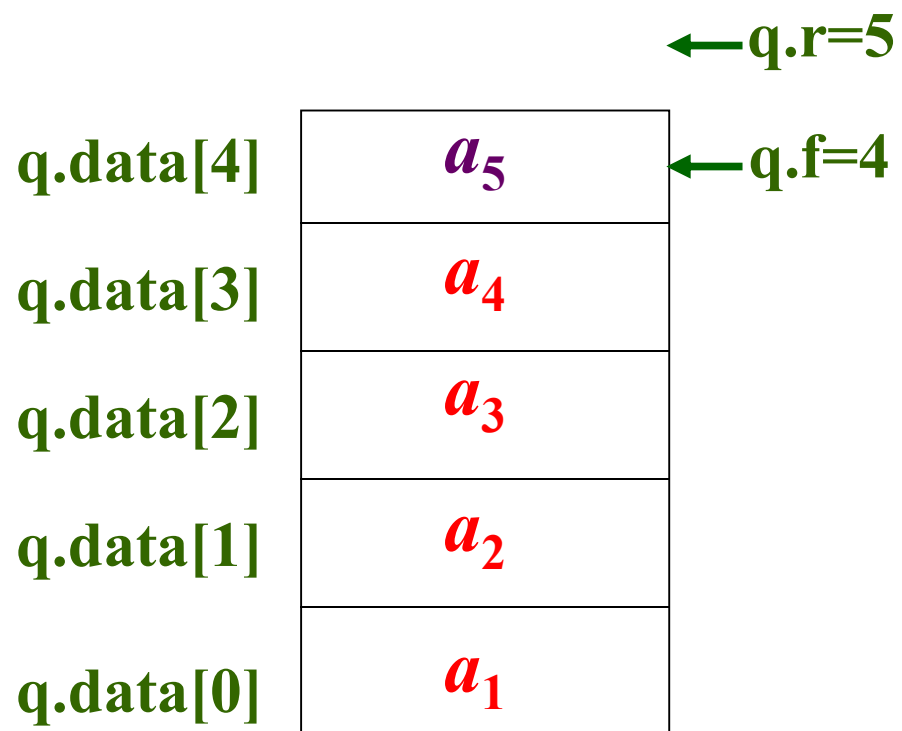


队列的顺序存储结构



出队

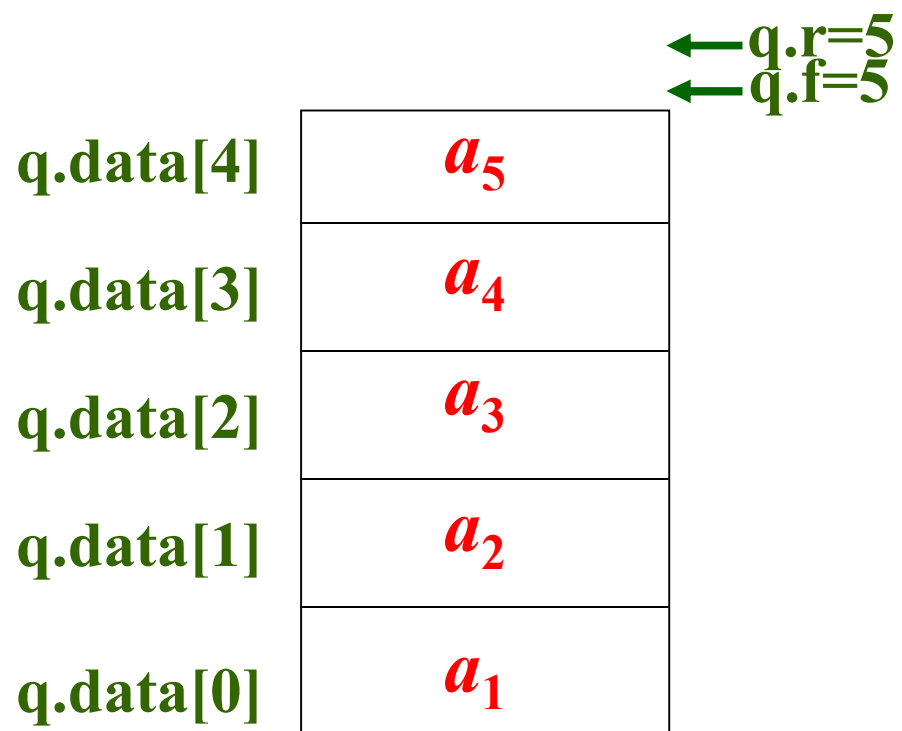
队列的顺序存储结构



出队

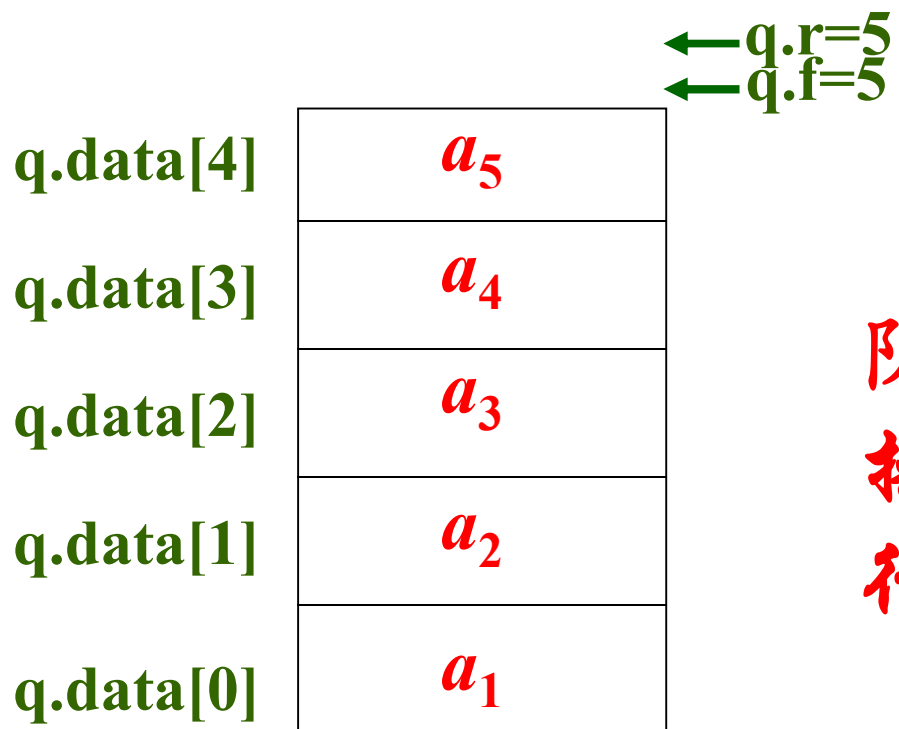


队列的顺序存储结构



出队

队列的顺序存储结构



队空——出队
操作不能进
行

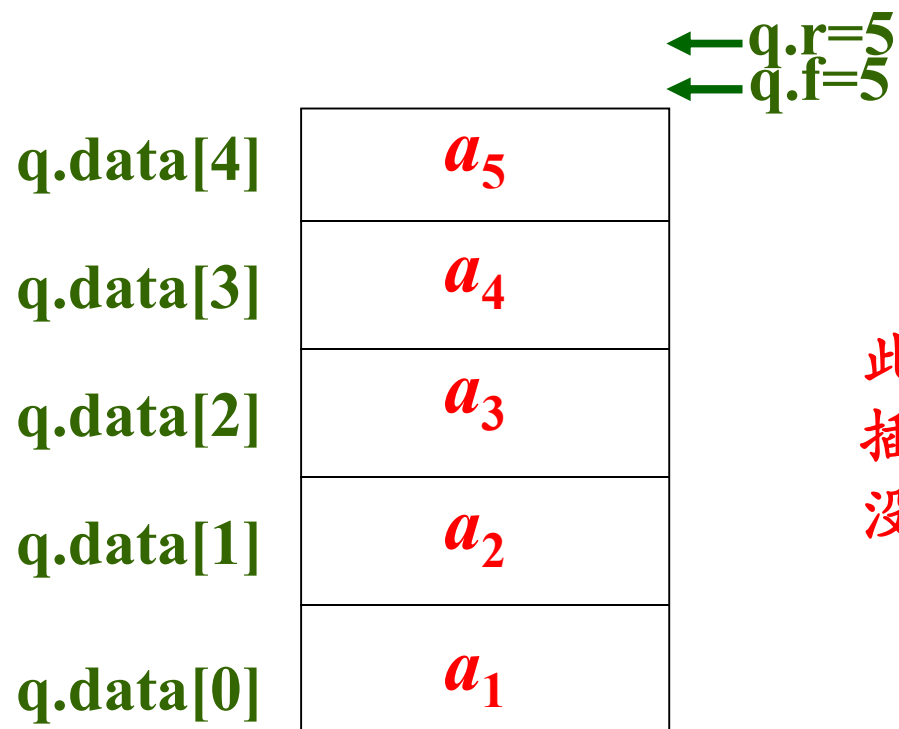
出队



队列的顺序存储结构

- 初始化: `q.f=0;q.r=0;`
- 入队: `if (q.r<MAXSIZE)`
 `{q.data[q.r]=x;q.r++;}`
 `else printf(“overflow”);`
- 出队: `if(q.f!=q.r)//if(q.f<q.r)`
 `q.f++;`

队列的顺序存储结构



此时队空，但是再次
插入数据元素却判断
没有空间——假溢出

入队 $x=a_6$

队列的顺序存储结构—假溢出

◆ 顺序队列存在假溢出

◆ 解决方法

➤ 方法1：每删除一个数据元素，余下的所有数据元素顺次移动一个位置——太累 😞

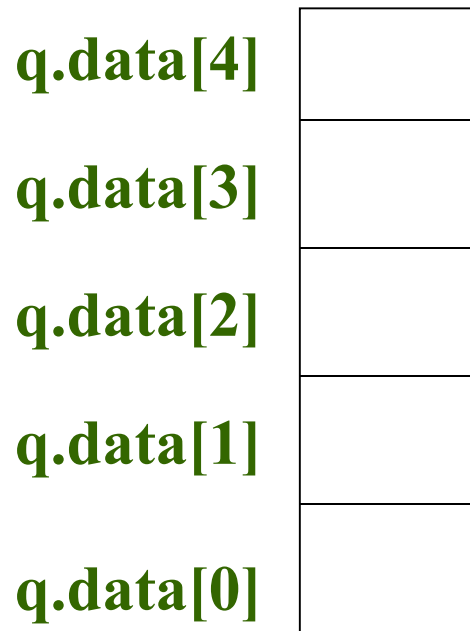
➤ 方法2：循环队列，将顺序队列data[0.. MAXSIZE - 1]看成头尾相接的循环结构



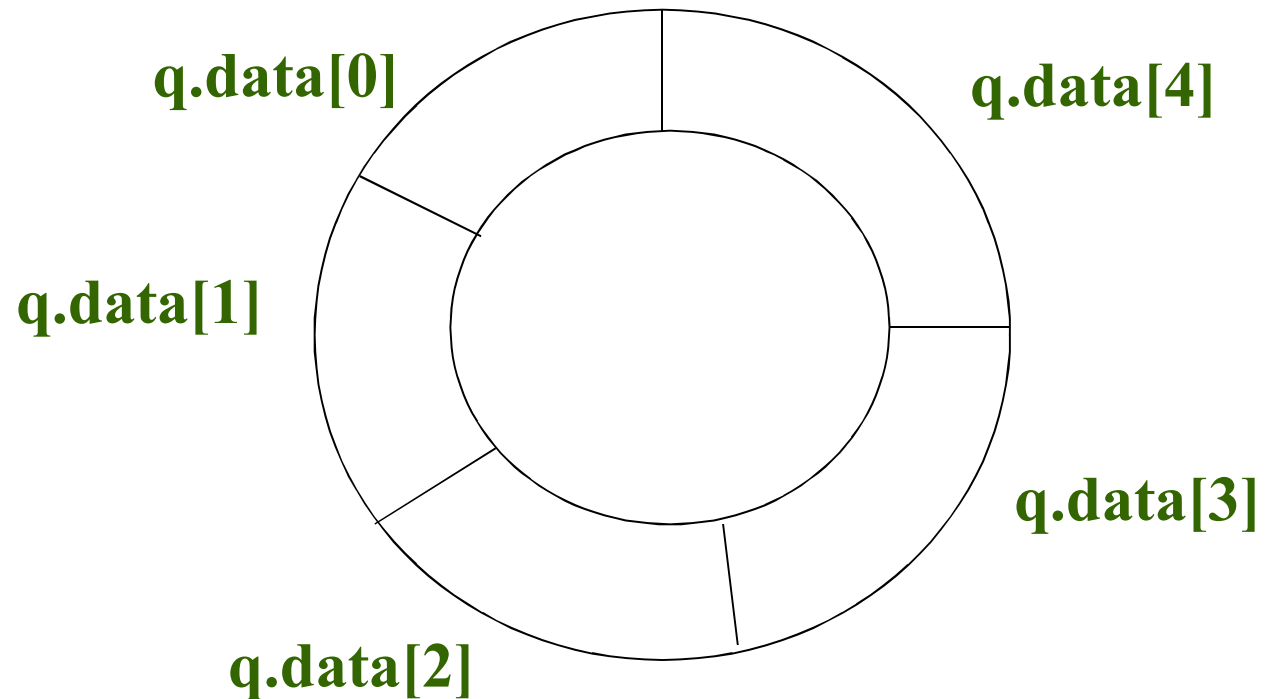
循环队列

初始化建个空队列: $q.f=0; q.r=0;$

将队列的数据区 $data[0..MAXSIZE-1]$ 看成头尾相接的循环结构



数组示意图



将正常的数组空间看成环状的

通过求余 ($\%$, MOD) 运算可实现数组的首尾相接



循环队列

- 初始化: $q.f=0; q.r=0;$
- 入队: if (有空间)
 $\{q.data[q.r]=x; q.r=(q.r+1)\%MAXSIZE;\}$
 else printf(“overflow”);

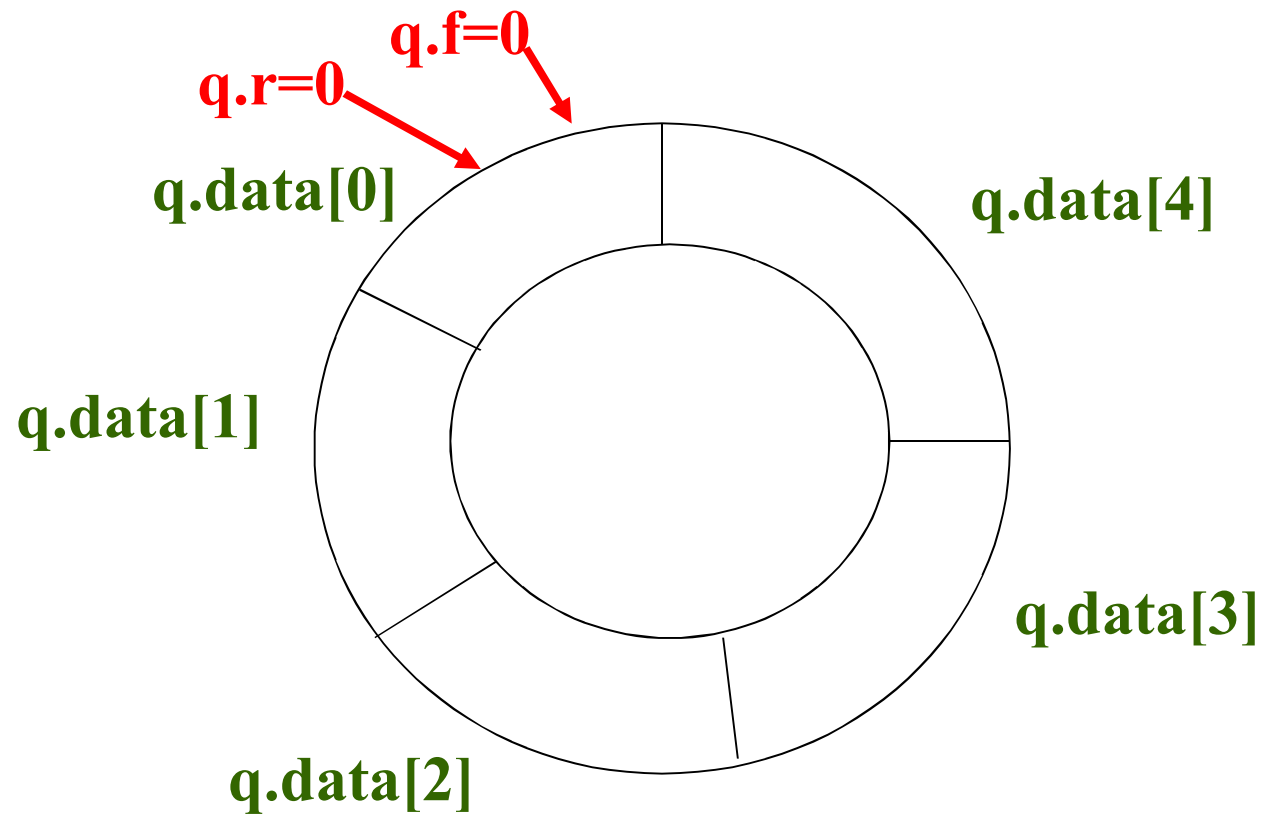


循环队列

- 初始化: $q.f=0; q.r=0;$
- 入队: if (有空间)
 $\{q.data[q.r]=x; q.r=(q.r+1)\%MAXSIZE;\}$
else printf(“overflow”);
- 出队: if (队不空)
 $q.f=(q.f+1)\%MAXSIZE;$

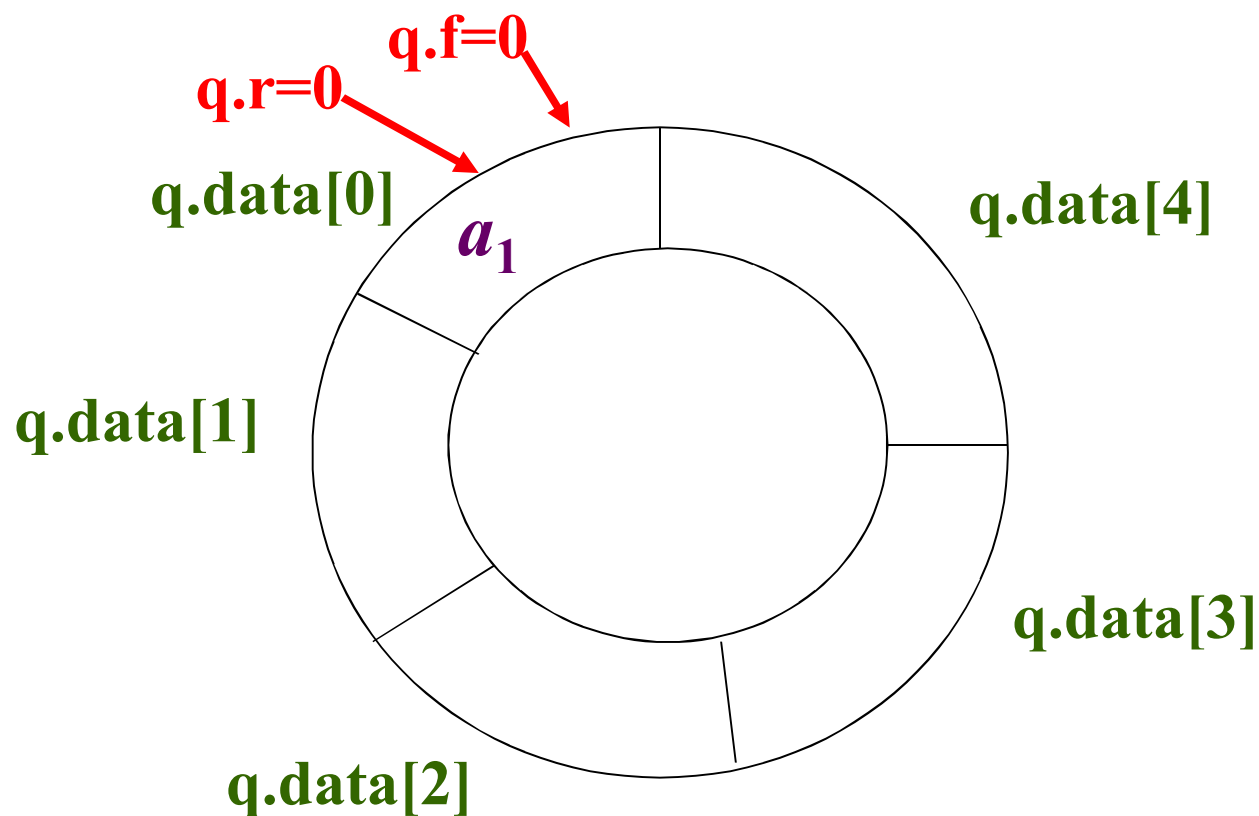
循环队列 --初始化建个空队列

$q.f=0; q.r=0;$



循环队列--入队 $x=a_1$

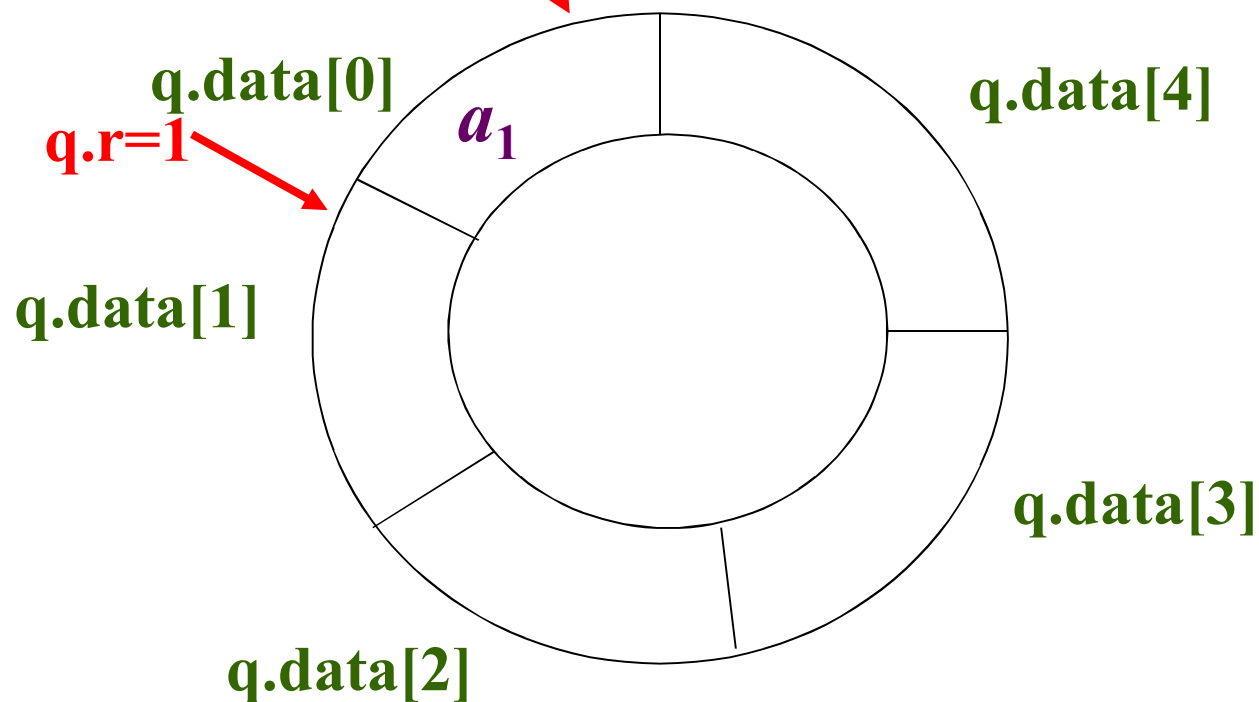
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;



循环队列--入队 $x=a_1$

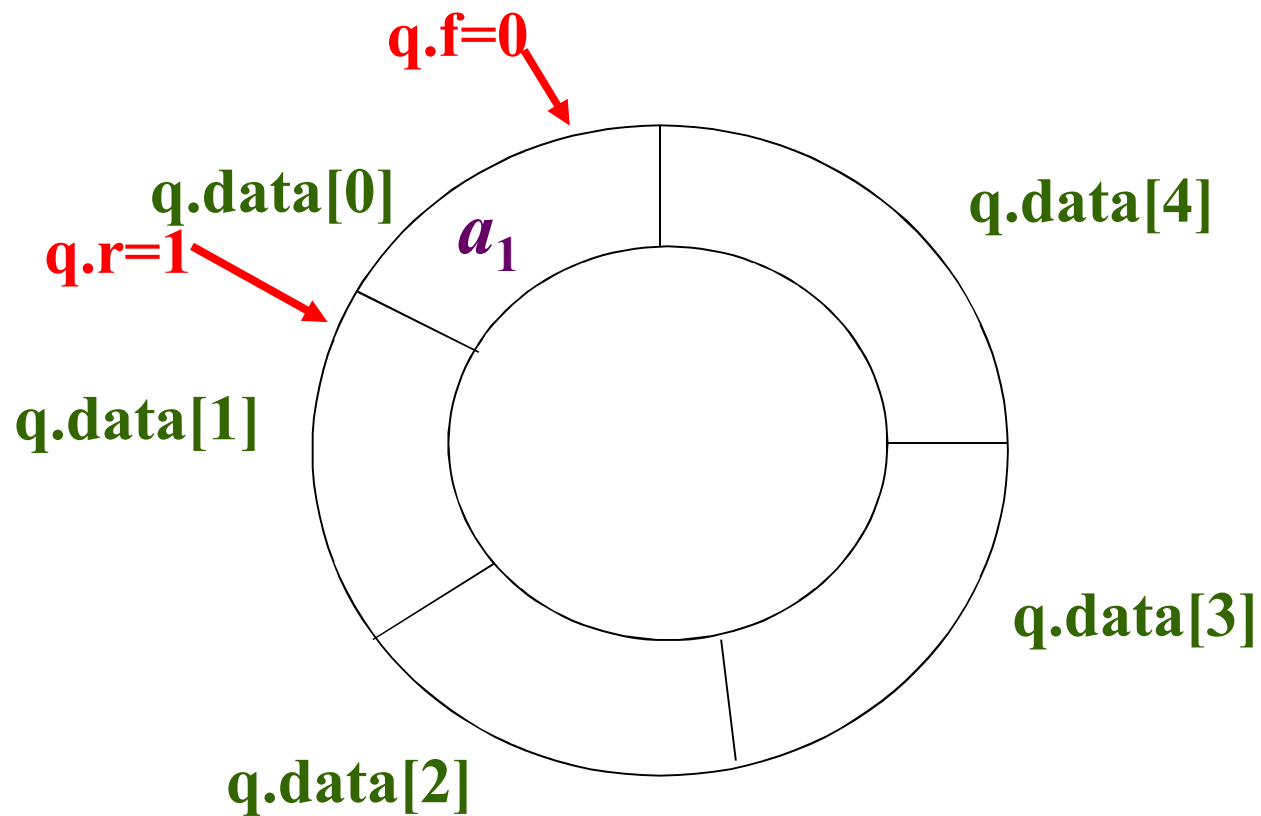
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;

$q.f=0$ 队尾指针下移-- $q.r=(q.r+1)\%MAXSIZE$;



循环队列--入队 $x=a_2$

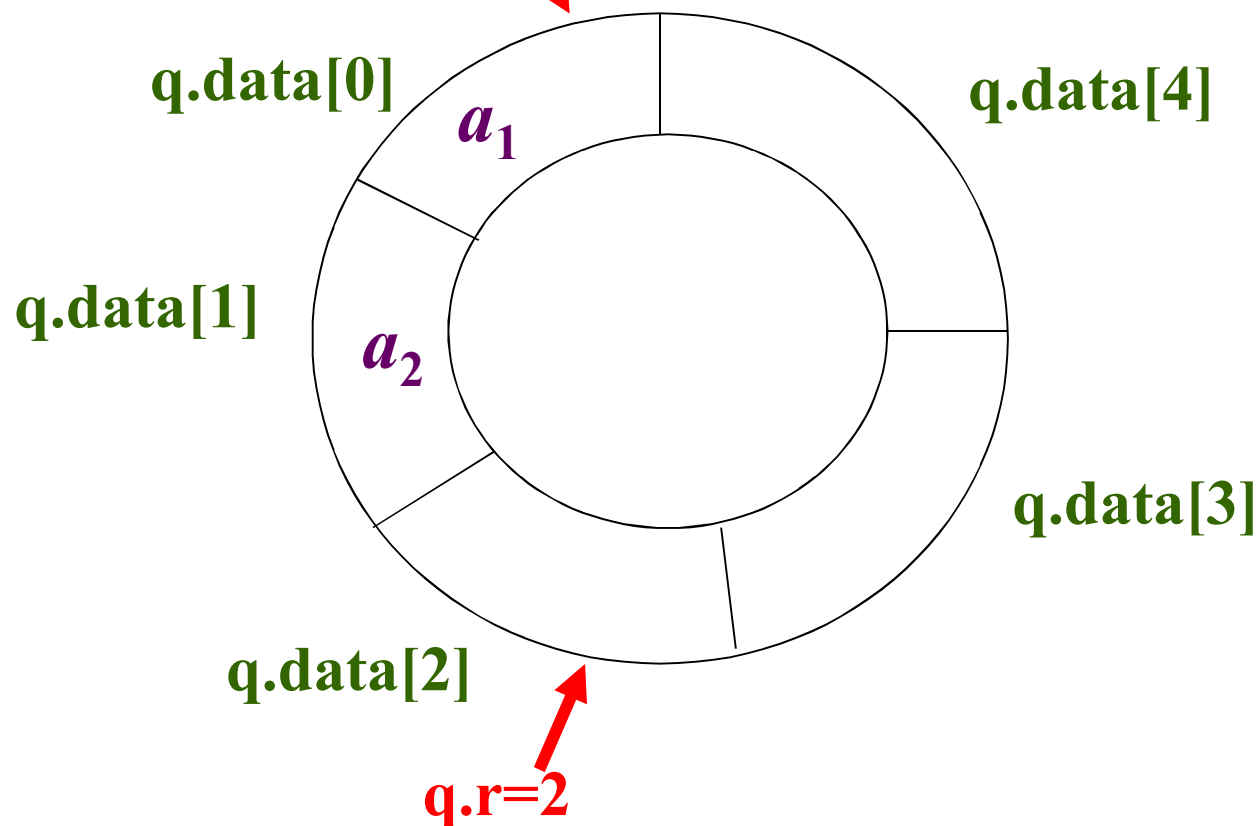
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;



循环队列--入队 $x=a_2$

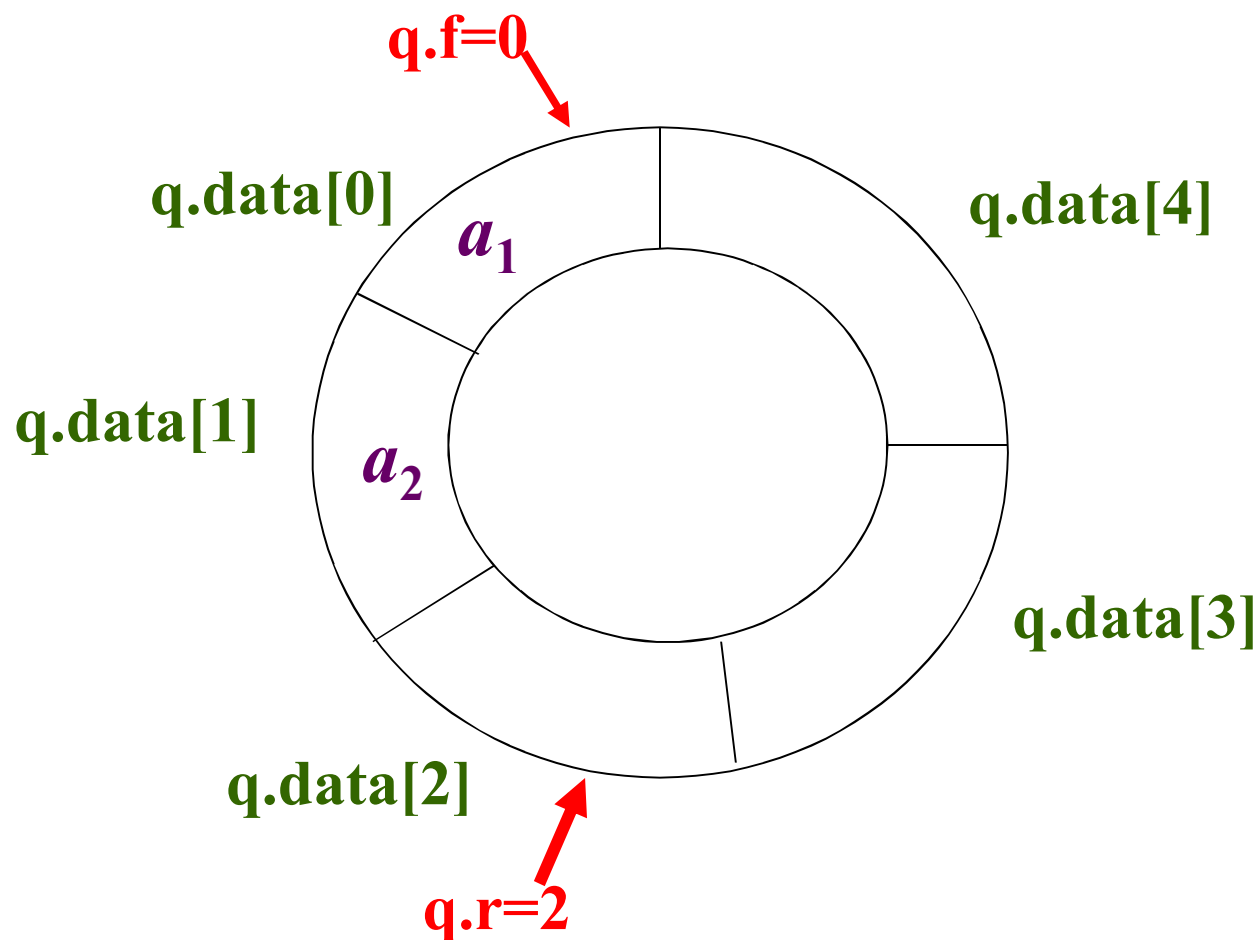
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;

$q.f=0$ 队尾指针下移-- $q.r=(q.r+1)\%MAXSIZE$;



循环队列--入队 $x=a_3$

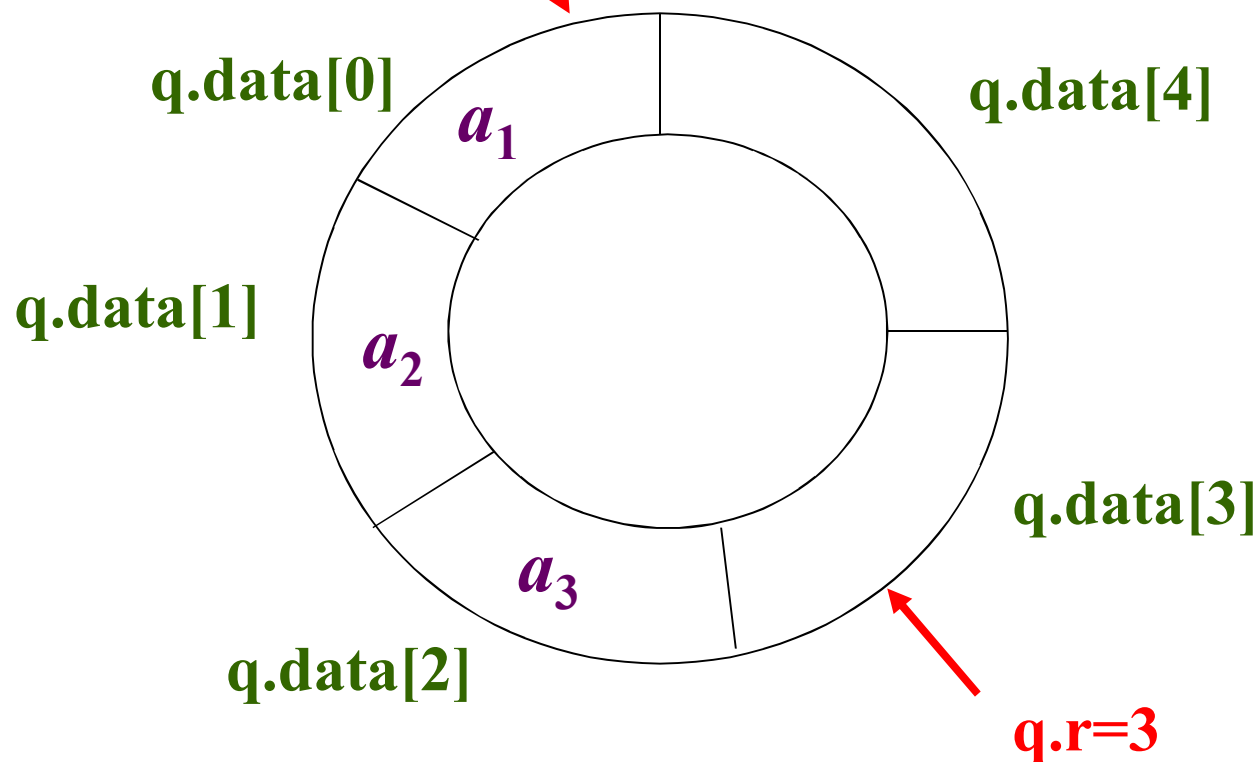
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;



循环队列--入队 $x=a_3$

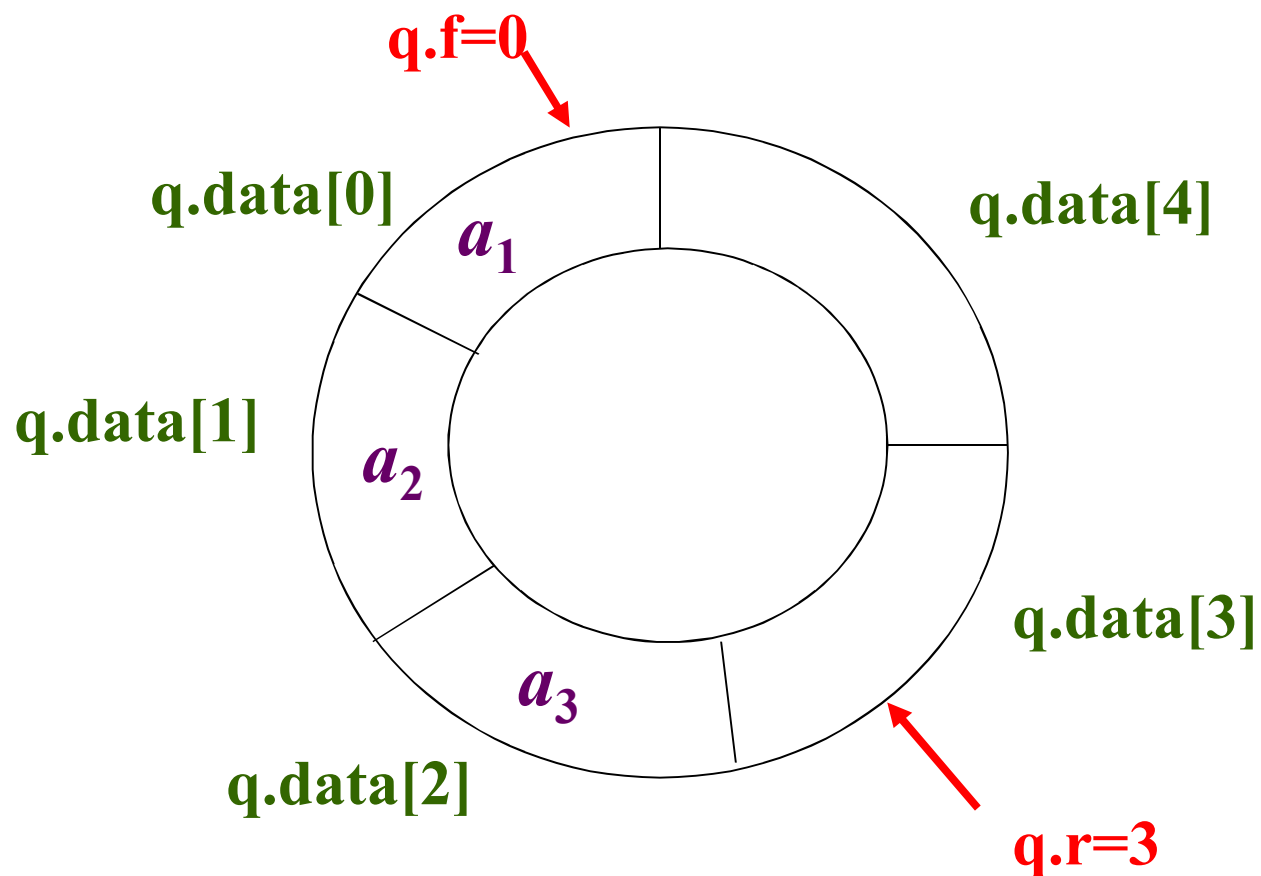
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;

$q.f=0$ 队尾指针下移-- $q.r=(q.r+1)\%MAXSIZE$;



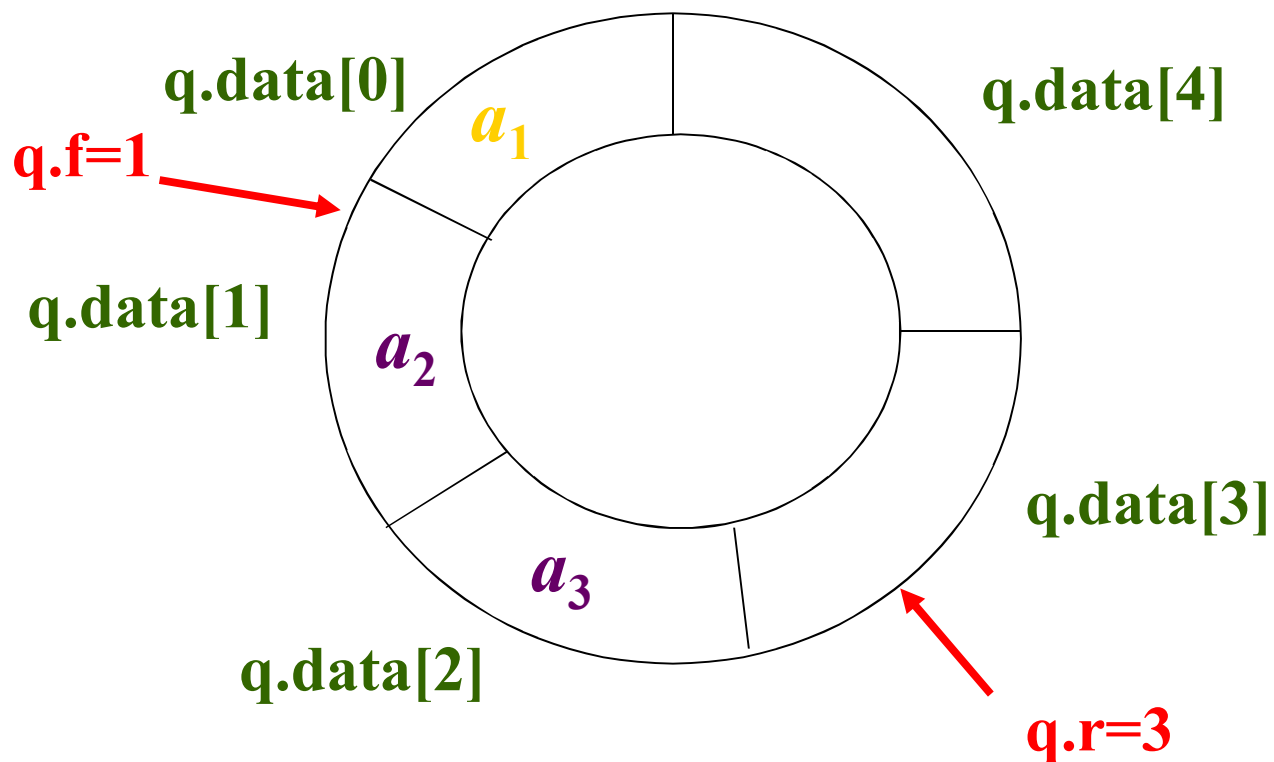
循环队列—出队

队列不空：将队首指针下移-- $q.f=(q.f+1)\%MAXSIZE$;



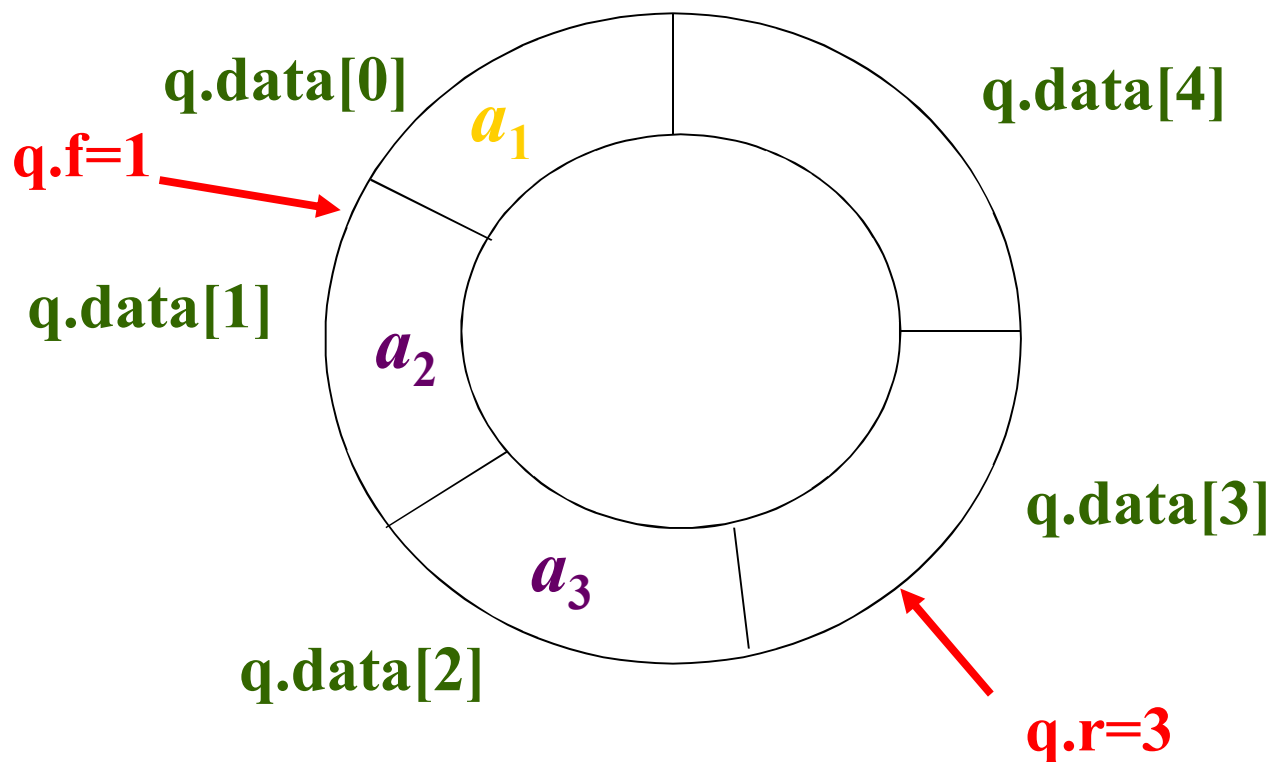
循环队列—出队

队列不空：将队首指针下移-- $q.f=(q.f+1)\%MAXSIZE$;



循环队列—入队 $x=a_4$

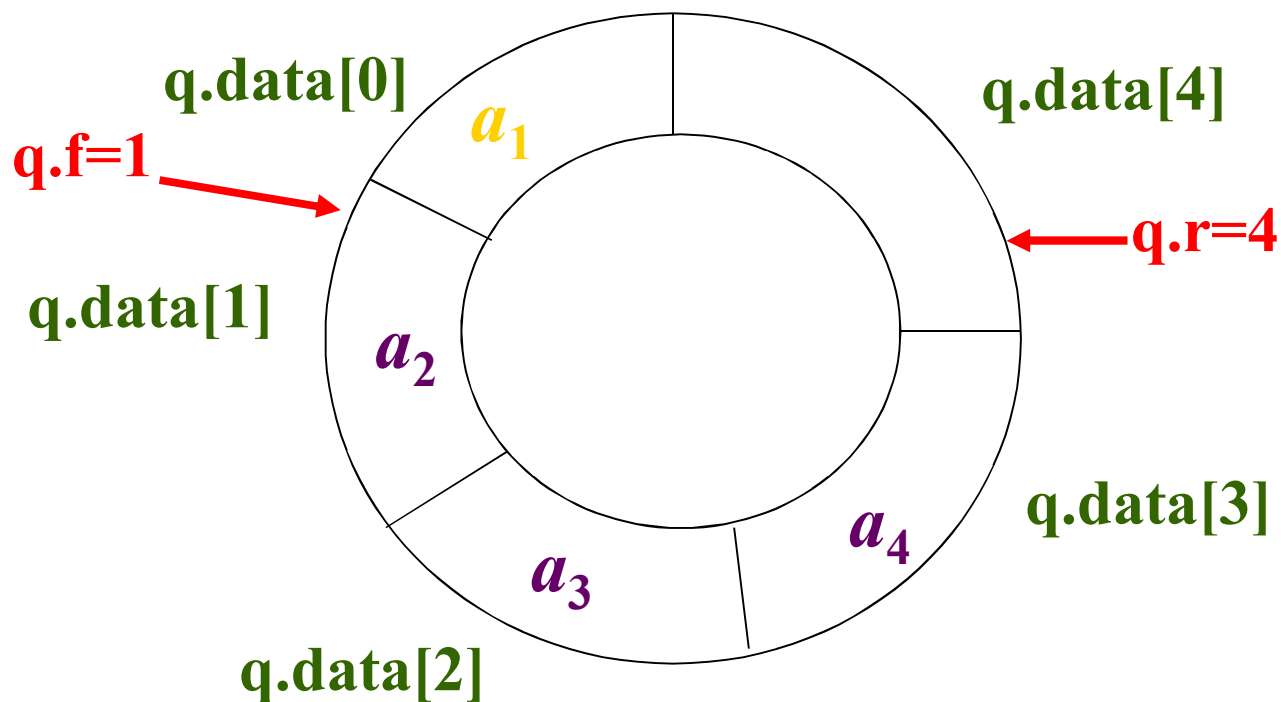
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;



循环队列—入队 $x=a_4$

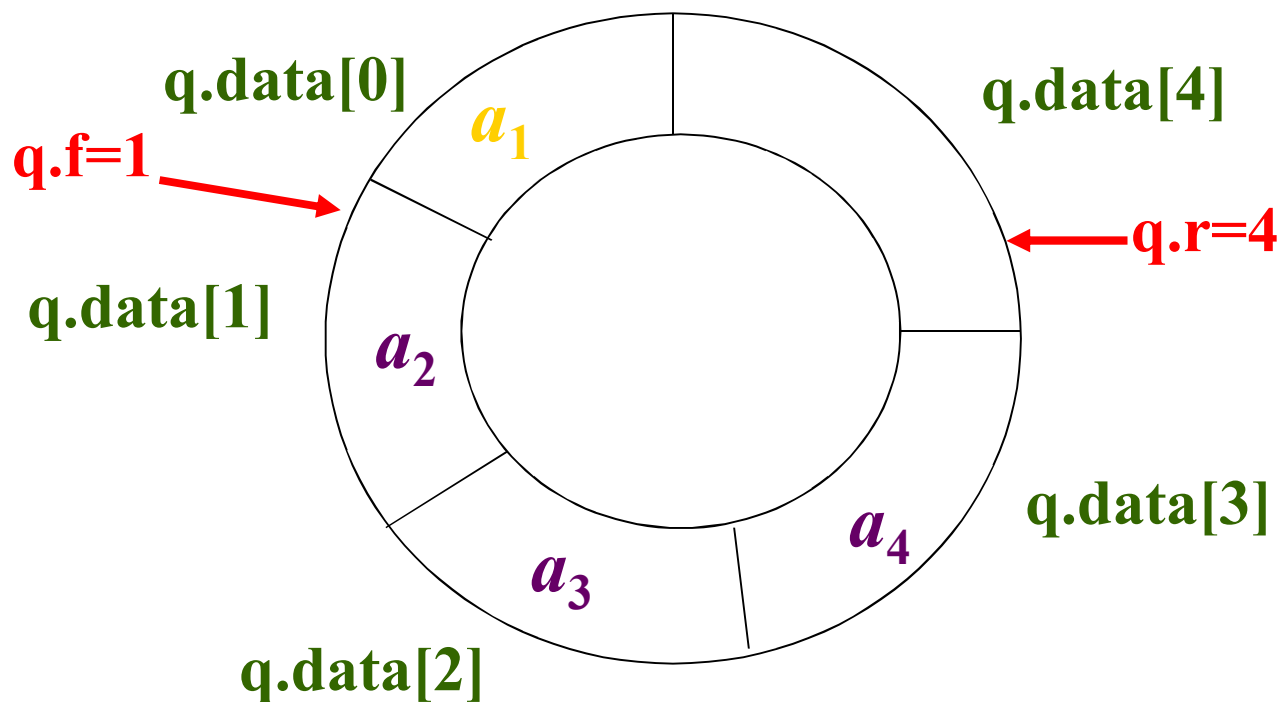
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;

队尾指针下移-- $q.r=(q.r+1)\%MAXSIZE$;



循环队列—入队 $x=a_5$

有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;

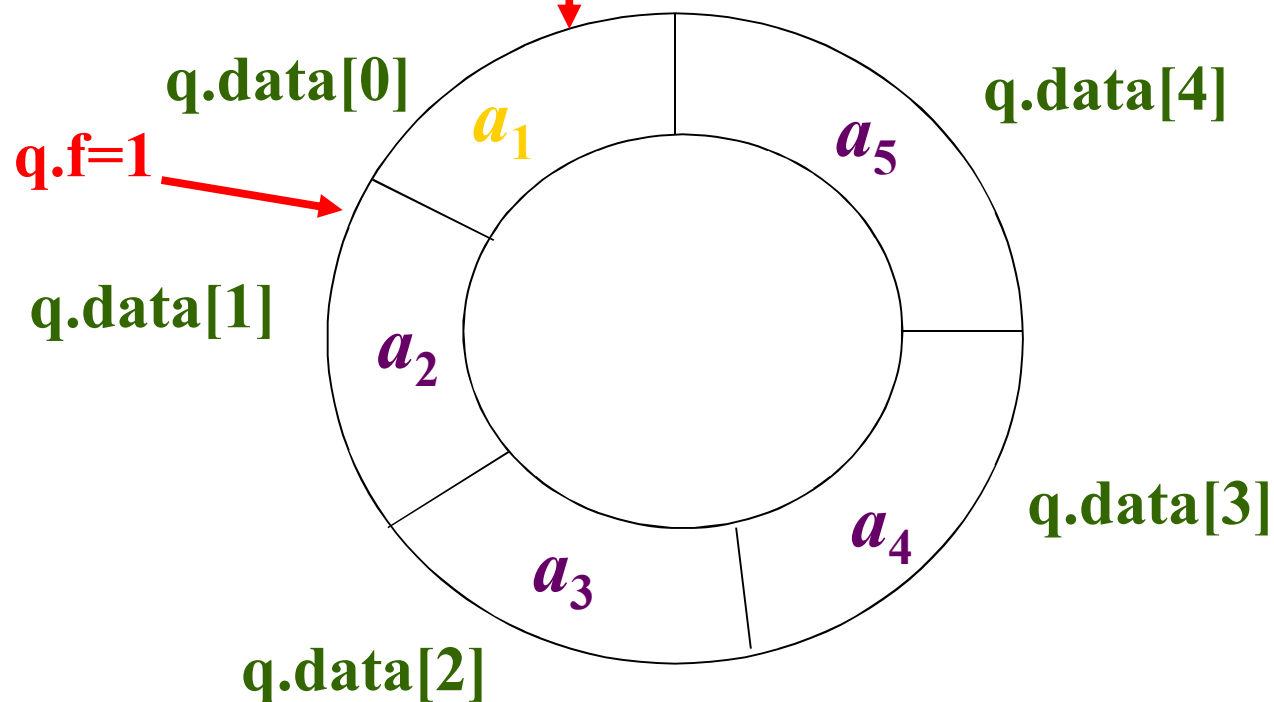


循环队列—入队 $x=a_5$

有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;

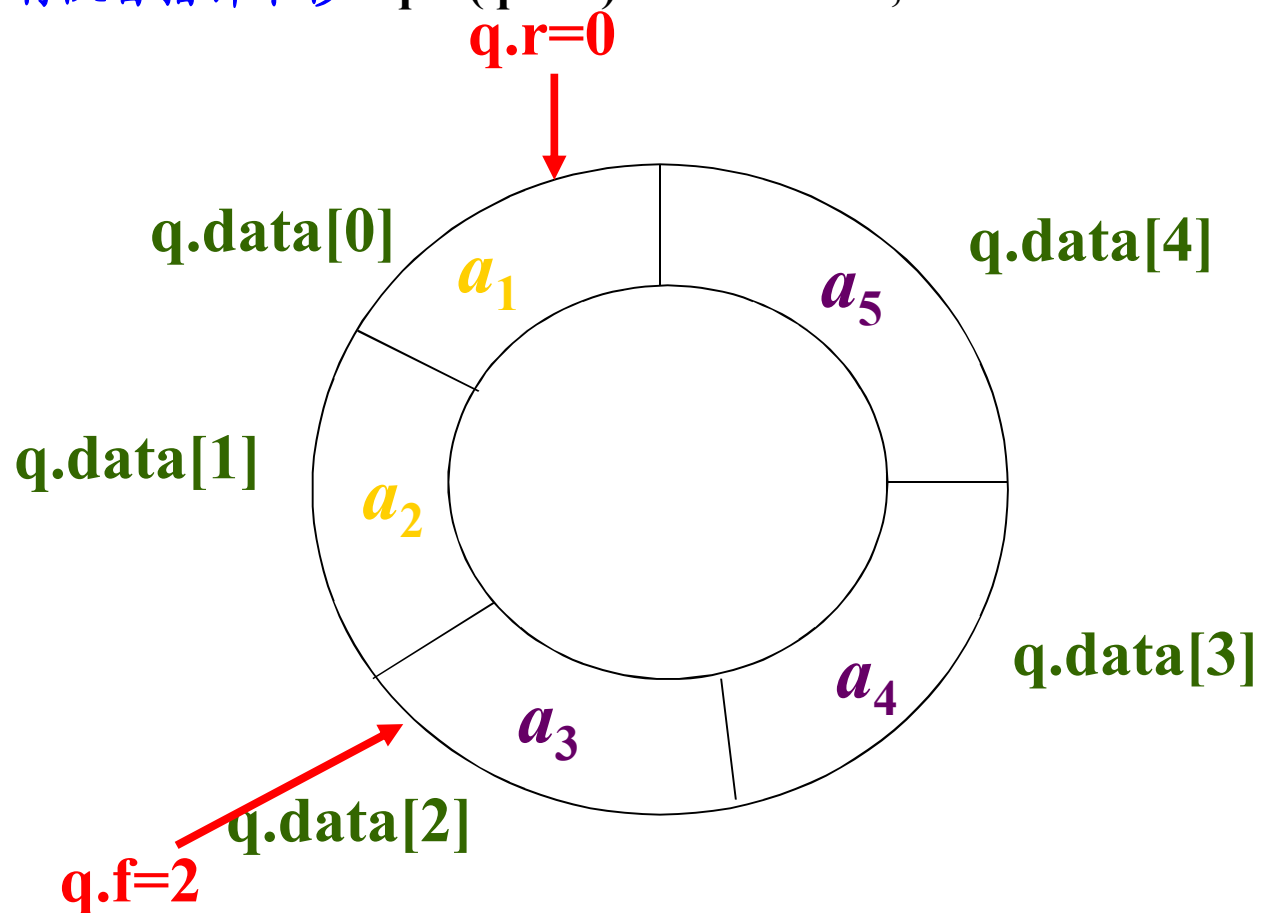
$q.r=0$

队尾指针下移-- $q.r=(q.r+1)\%MAXSIZE$;



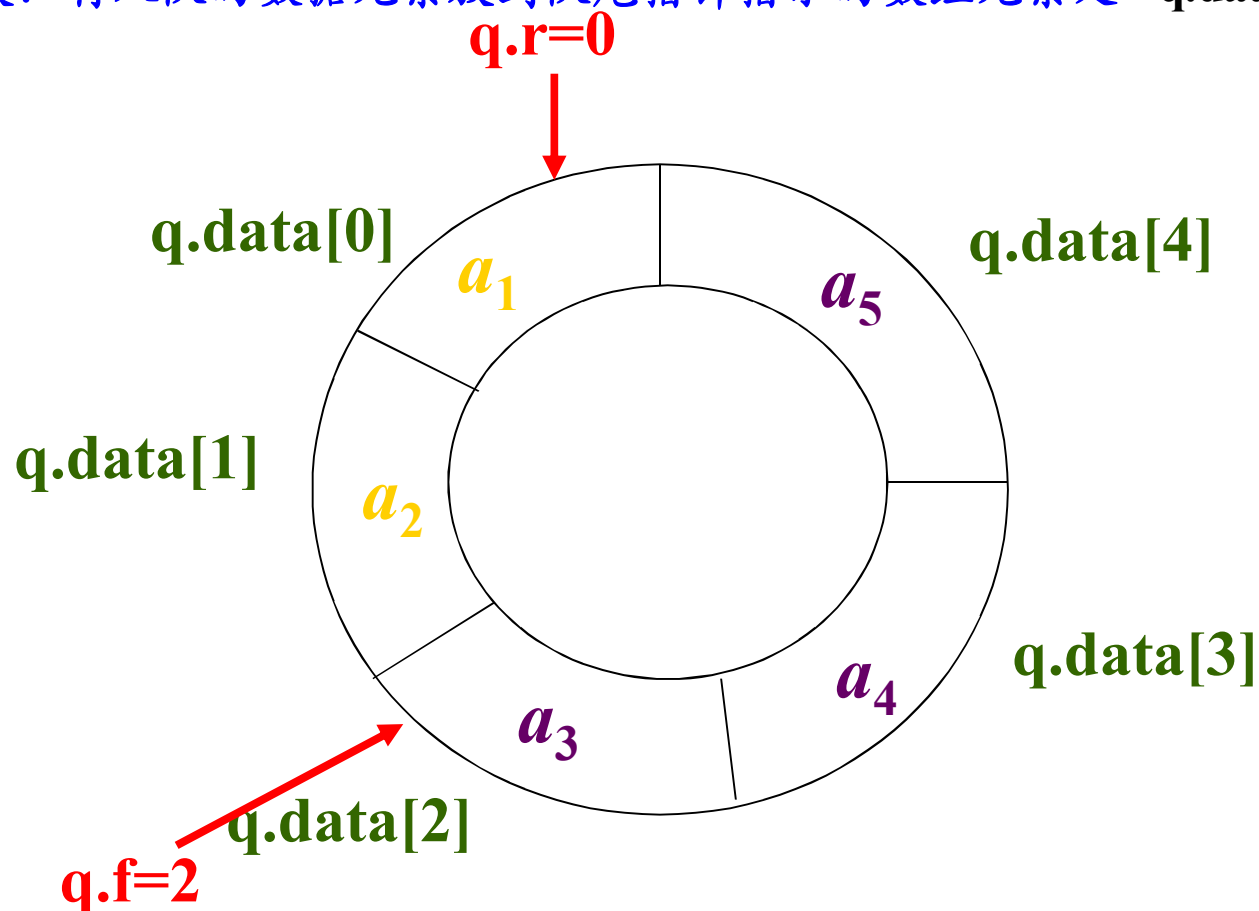
循环队列—出队

队列不空：将队首指针下移-- $q.f = (q.f + 1) \% \text{MAXSIZE}$;



循环队列—入队 $x=a_6$

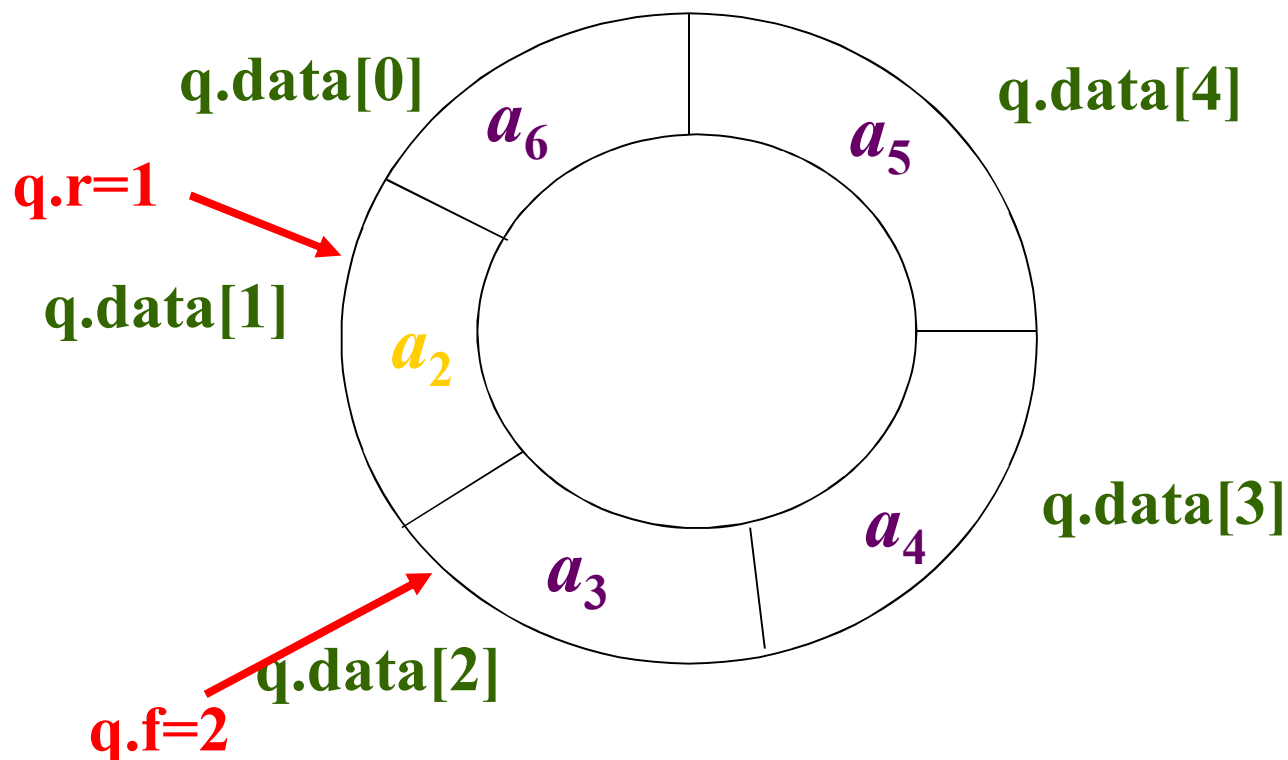
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;



循环队列—入队 $x=a_6$

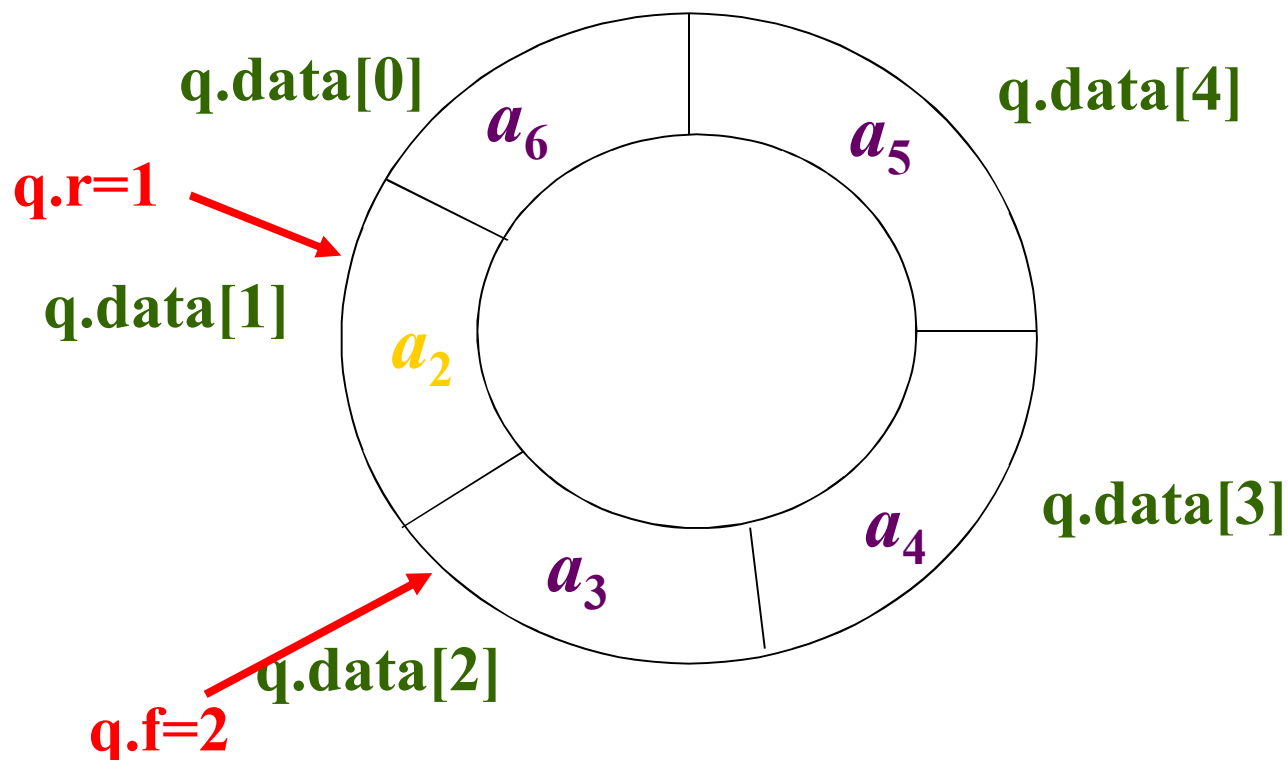
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;

队尾指针下移-- $q.r=(q.r+1)\%MAXSIZE$;



循环队列—入队 $x=a_7$

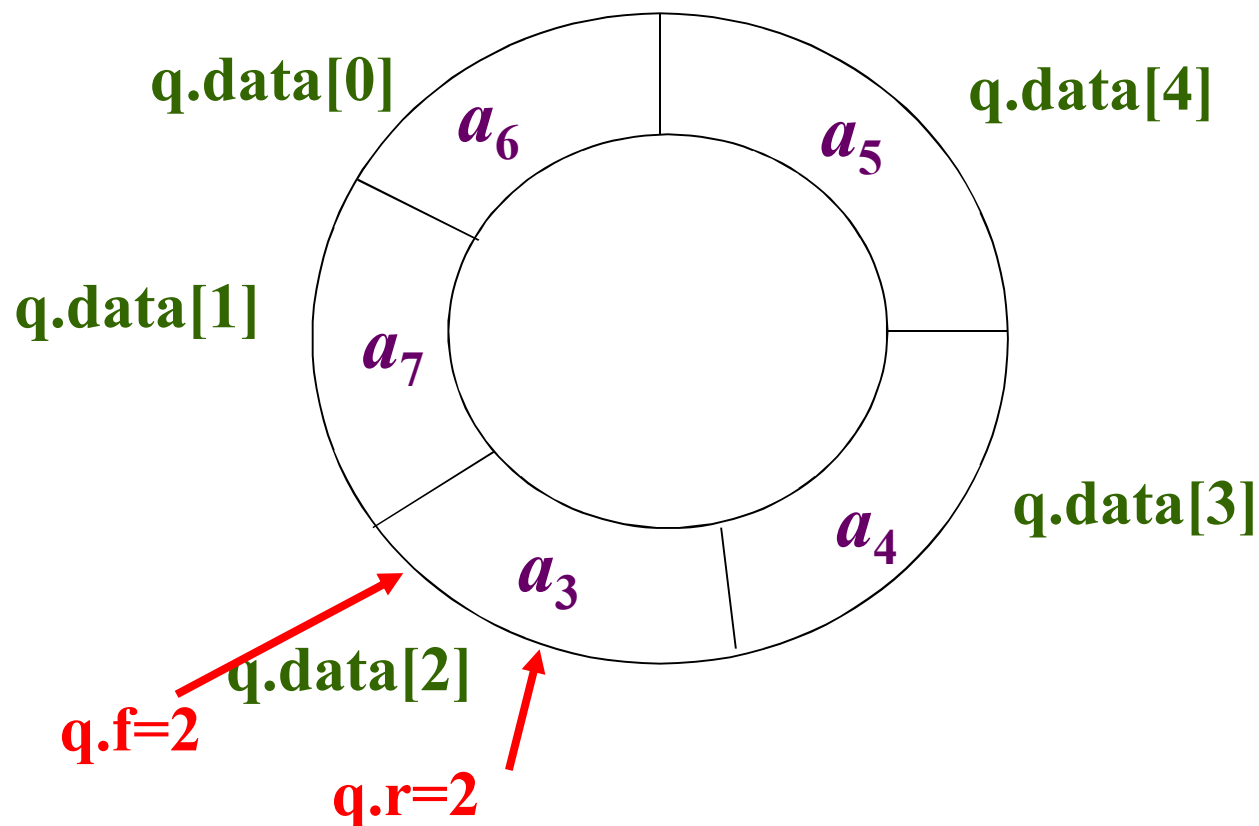
有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;



循环队列—入队 $x=a_7$

有空间存放：将入队的数据元素放到队尾指针指示的数组元素处-- $q.data[q.r]=x$;

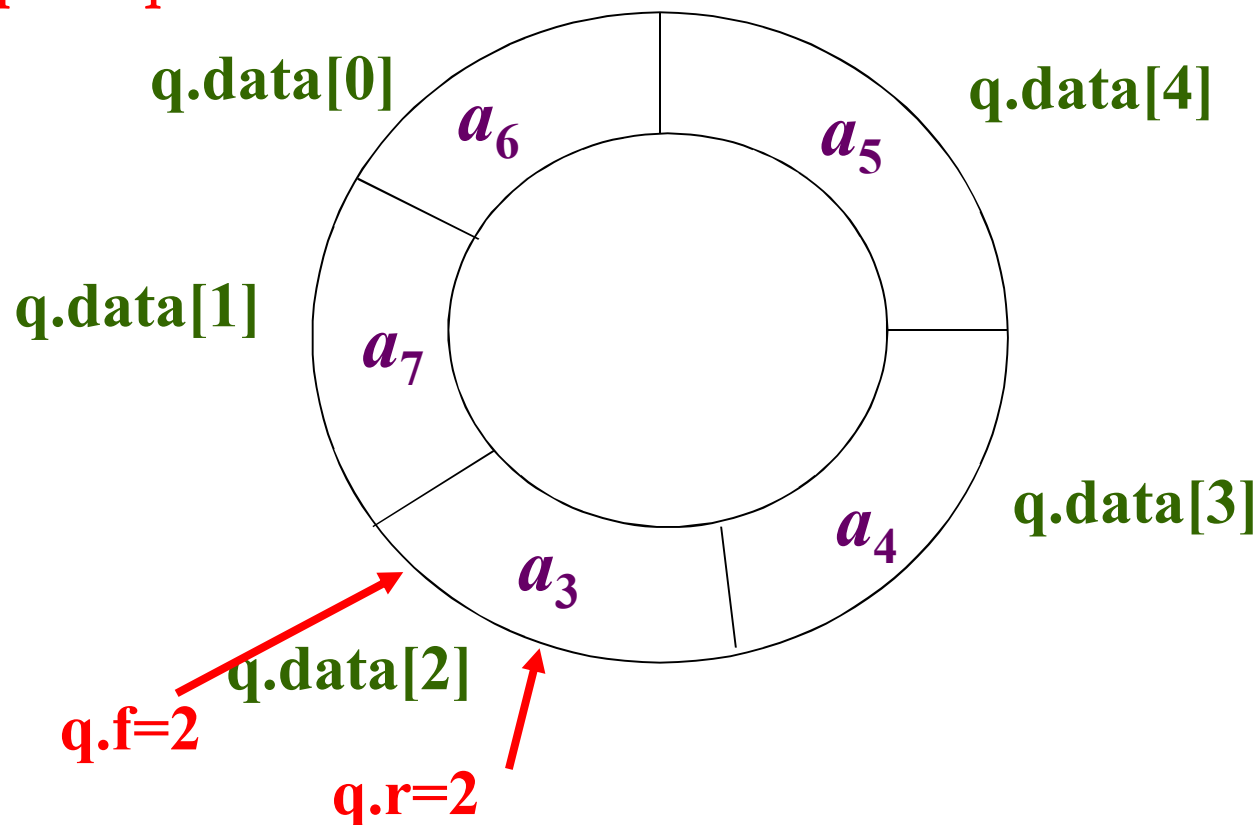
队尾指针下移-- $q.r=(q.r+1)\%MAXSIZE$;



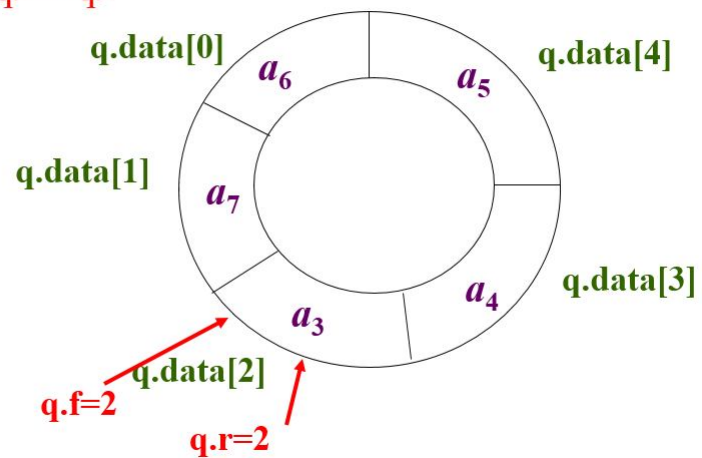
循环队列—入队 $x=a_8$

此时，所有的数组元素均已存放队列数据，**队满**，没有空间存放要入队的数据了—入队操作不能正常进行，给出出错信息，空间**溢出**。

队满： $q.f == q.r$

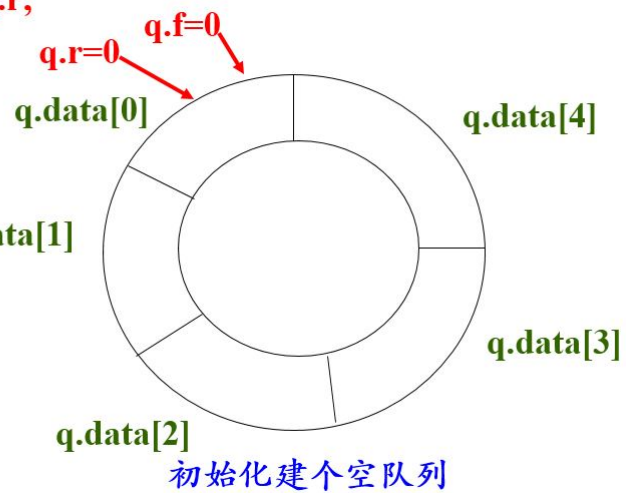


队满: $q.f == q.r$



队满示意图

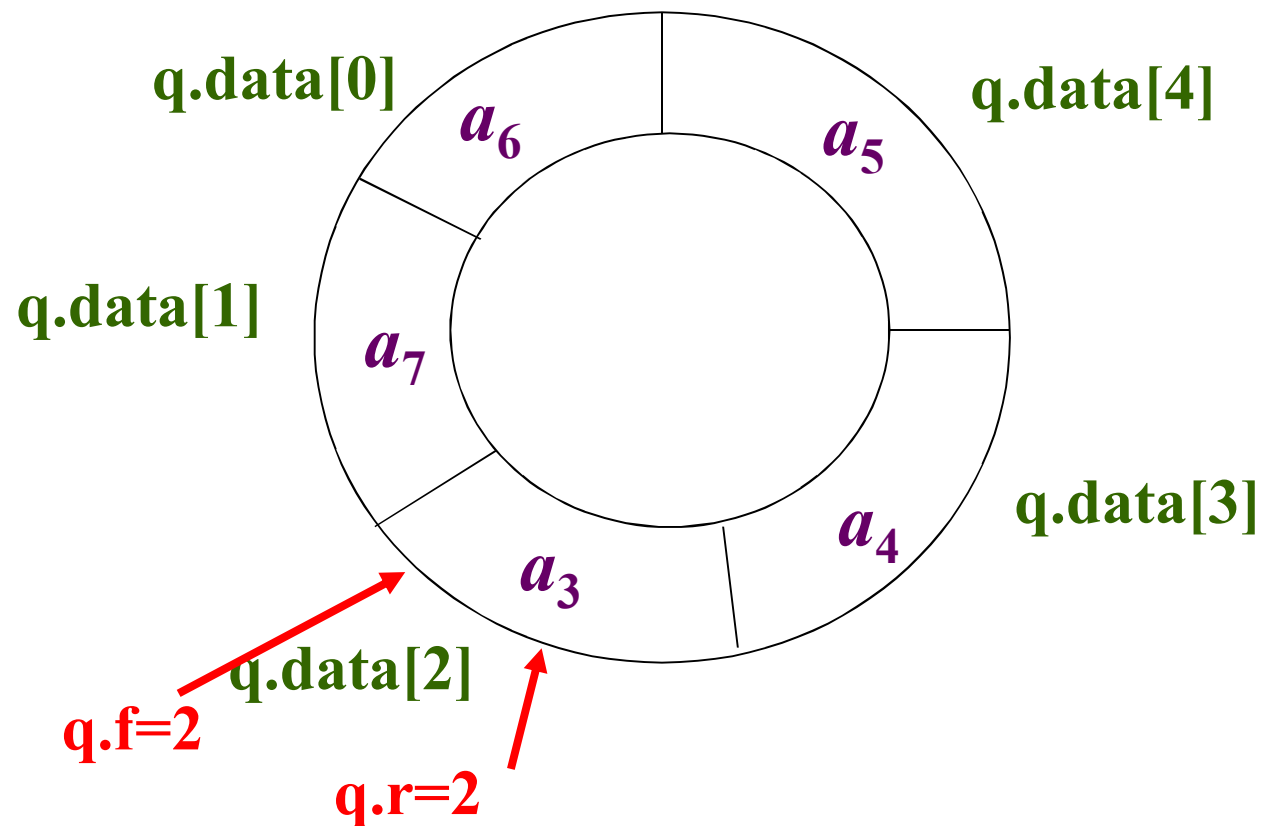
$q.f == q.r;$



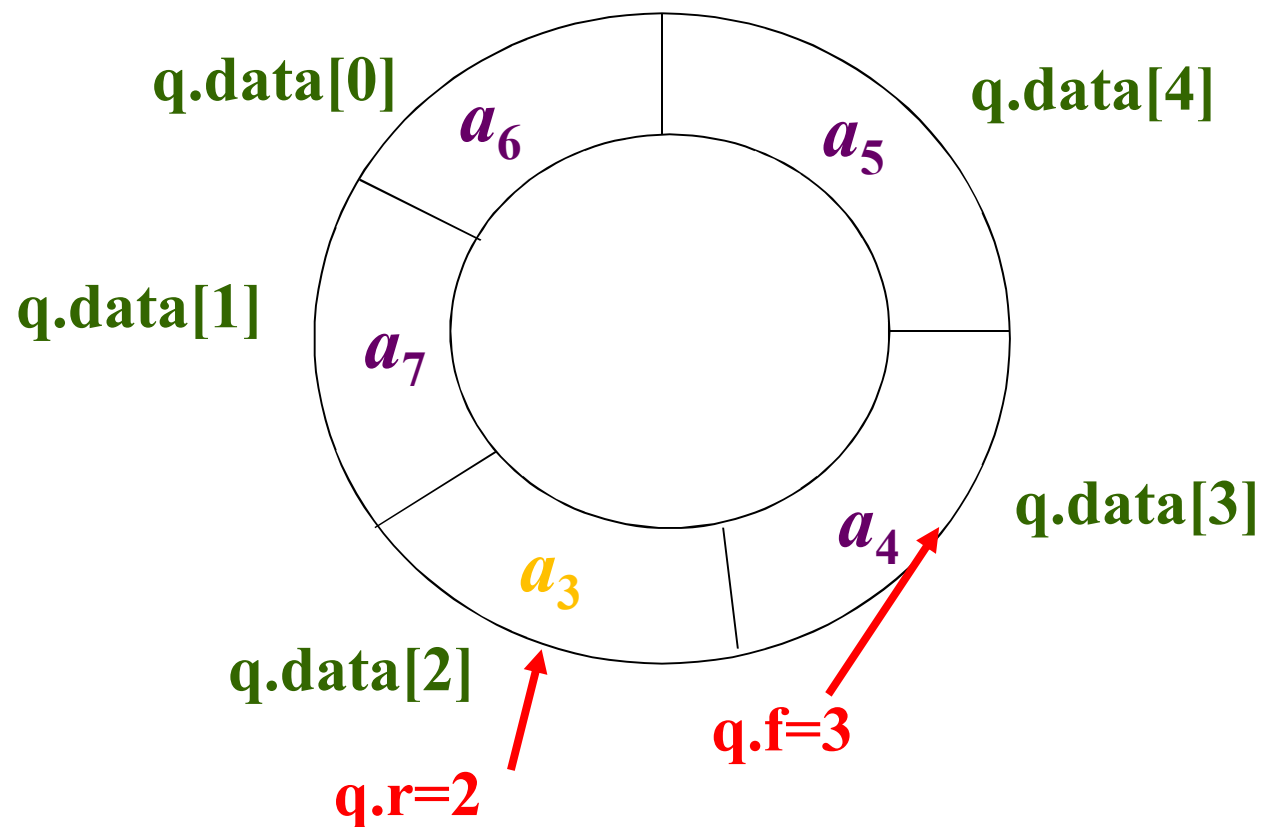
队空示意图1

循环队列—出队

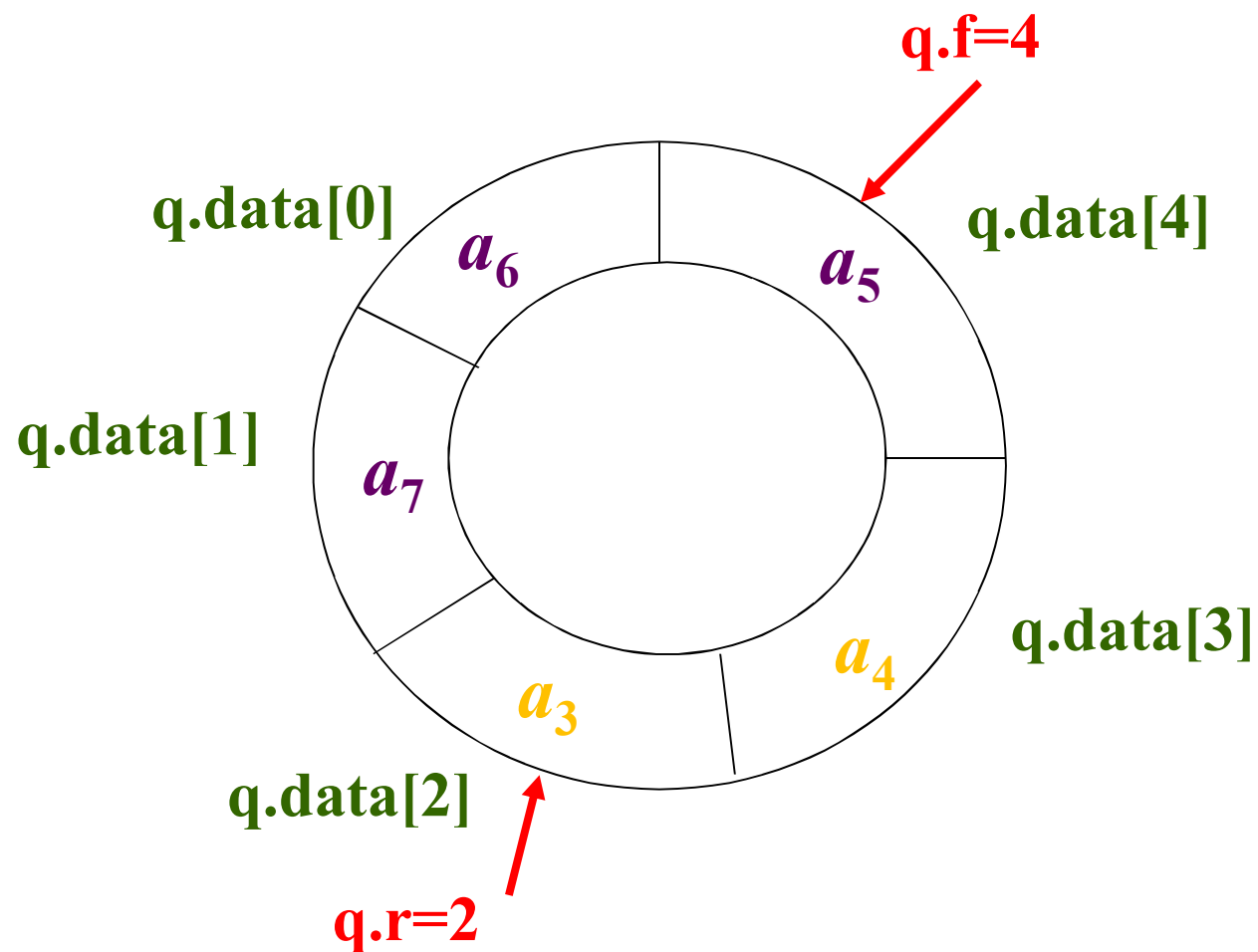
队满: $q.f == q.r$



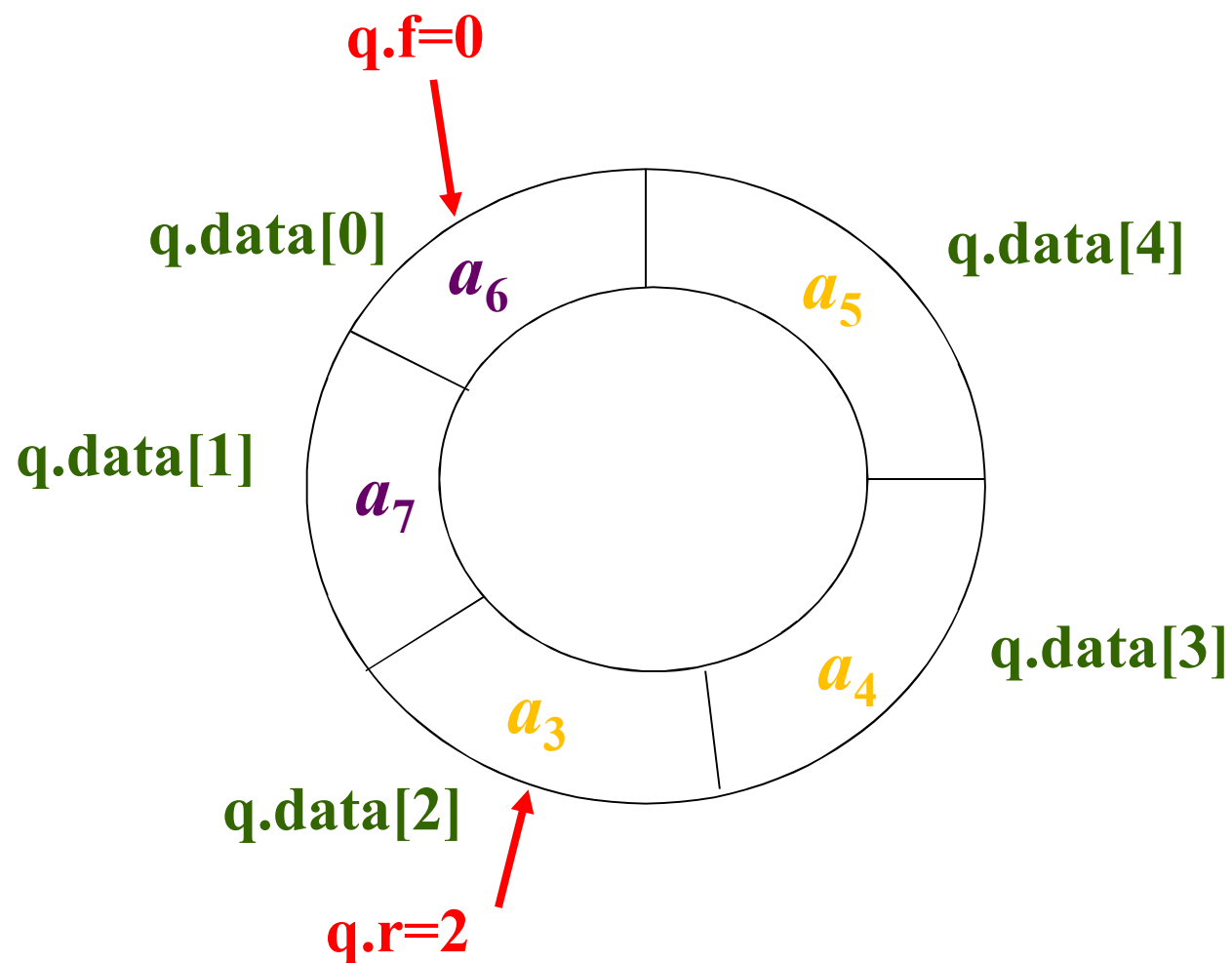
循环队列—出队



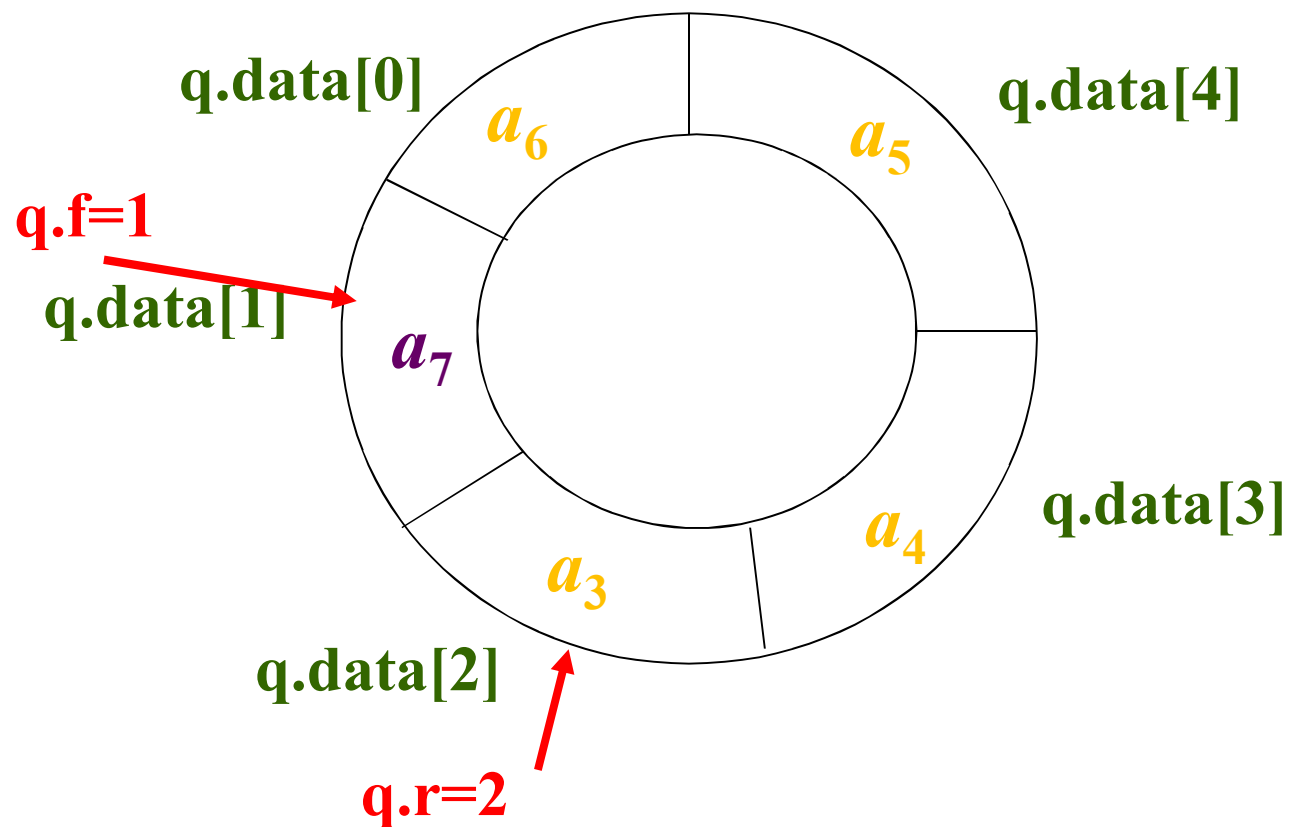
循环队列—出队



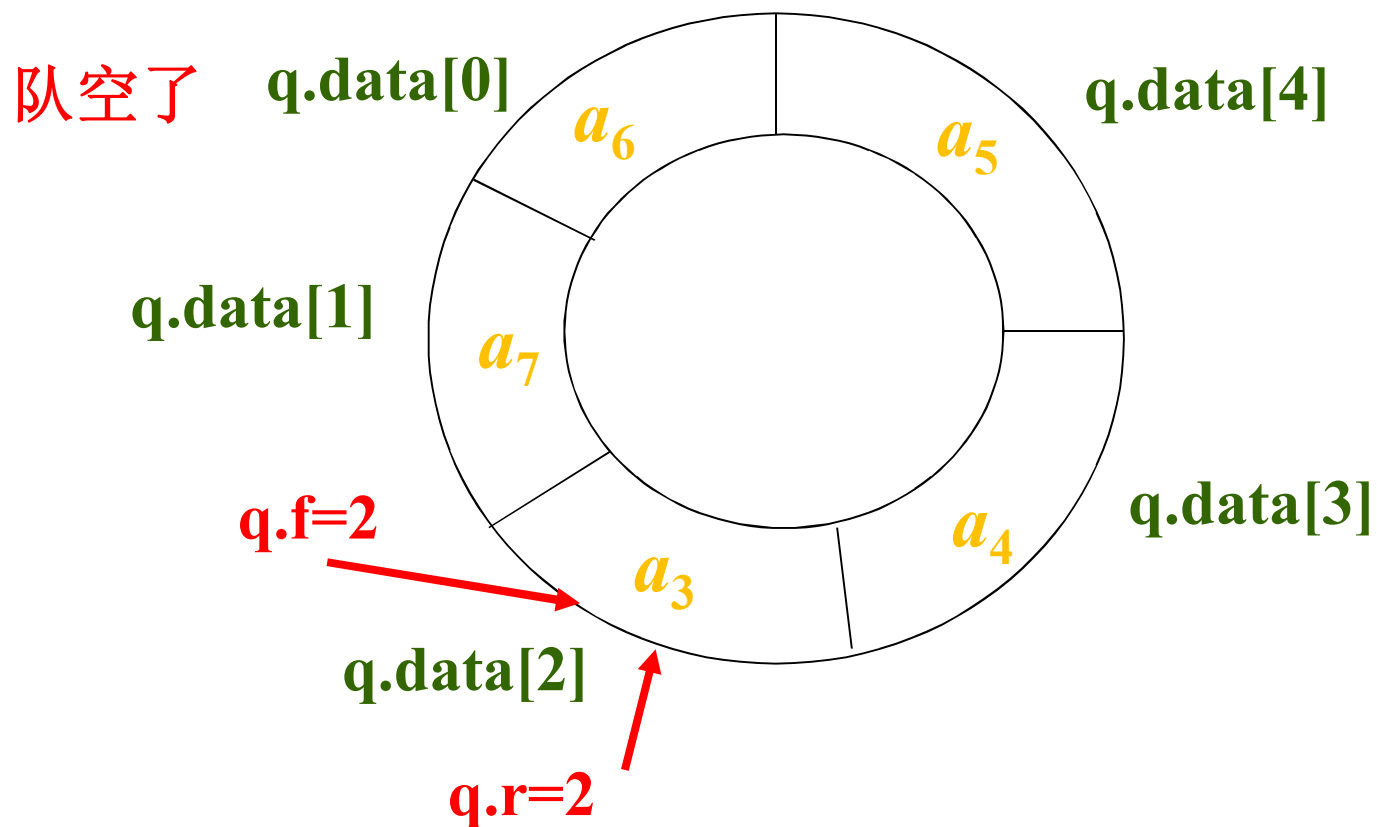
循环队列—出队



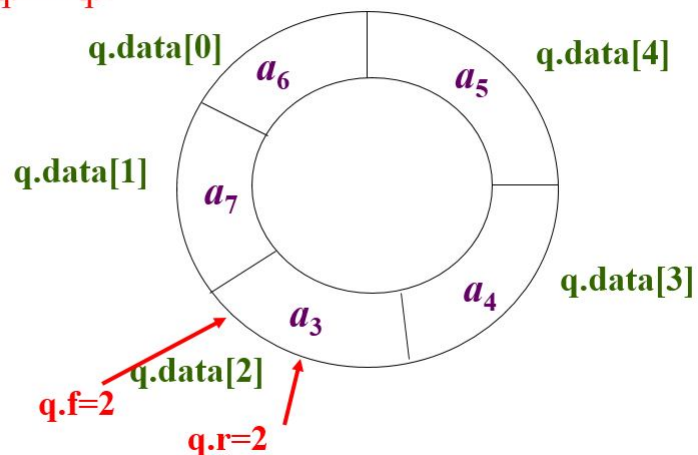
循环队列—出队



循环队列—出队

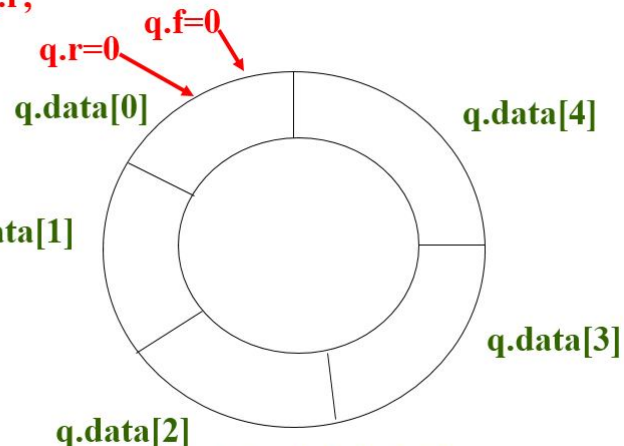


队满: $q.f == q.r$



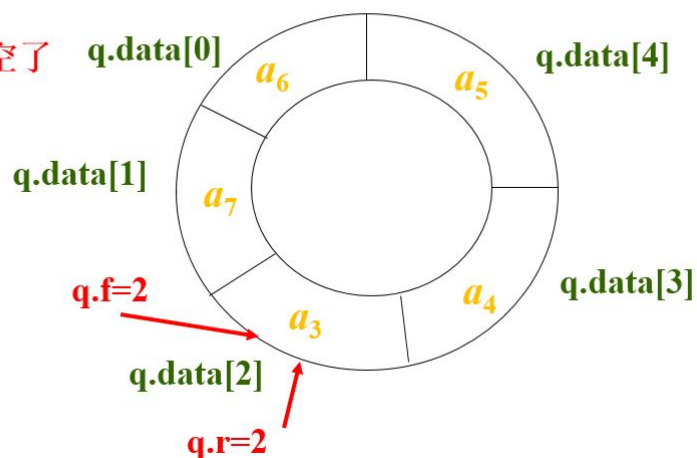
队满示意图

$q.f == q.r;$



队空示意图1

队空了



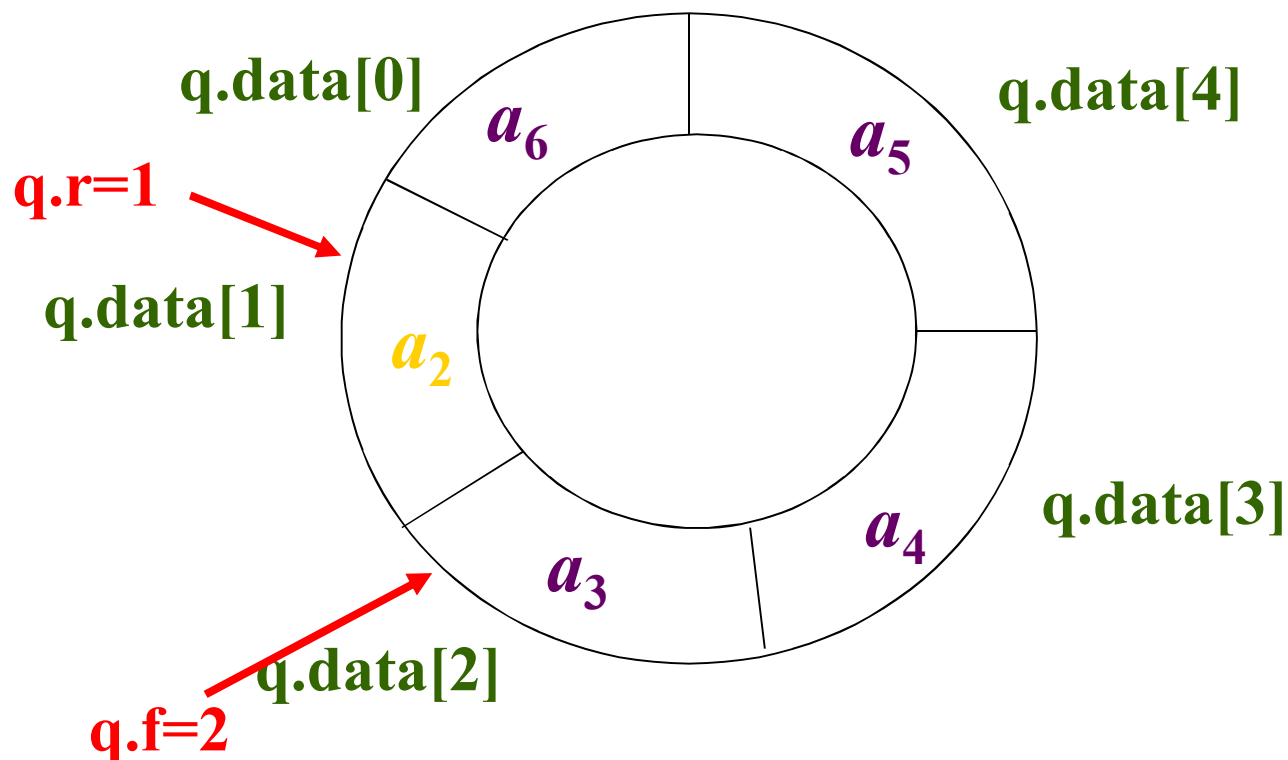
队空示意图2

问题: 队空和队满时 $q.f == q.r$, 无法根据队首和队尾指针的相对位置判断队列是处于“空”还是处于“满”的状态

队满, 队空时均有 $q.f == q.r$ 。如何区分何时队空? 何时队满?

循环队列—队空队满的区分

方法1: 为区分队空、队满, 牺牲一个存储位置,
当 $(q.r+1)\%MAXSIZE==q.f$ 时认为队满了, $q.r==q.f$ 为队空





循环队列—队空队满的区分

方法1:

队空: $q.r == q.f$

队满: $(q.r+1) \% \text{MAXSIZE} == q.f$

方法2: 设一计数器, 初始化时计数器清0, 入队时, 计数器+1, 出队时计数器-1

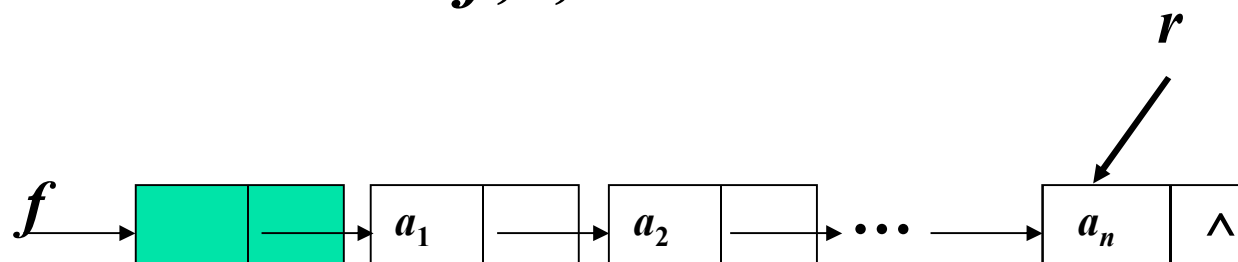
■ 练习

- 1 顺序队列如何解决假溢出?
- 2 循环队列如何判断队满和队空?

链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

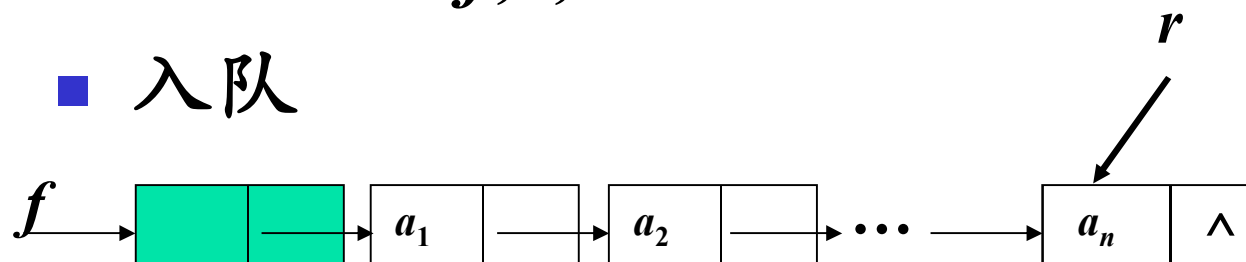
- 定义: 采用链式存储结构存放
- $\text{LinkList } f, r;$



链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

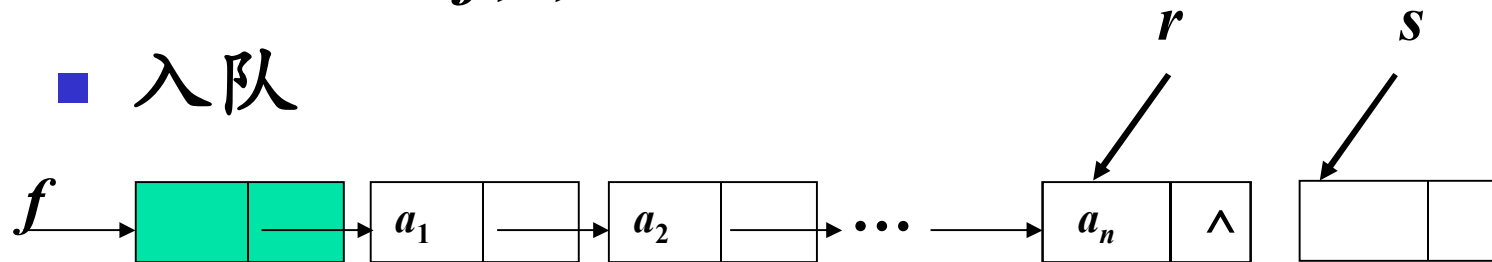
- 定义: 采用链式存储结构存放
- $\text{LinkList } f, r;$
- 入队



链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义: 采用链式存储结构存放
- $\text{LinkList } f, r;$
- 入队

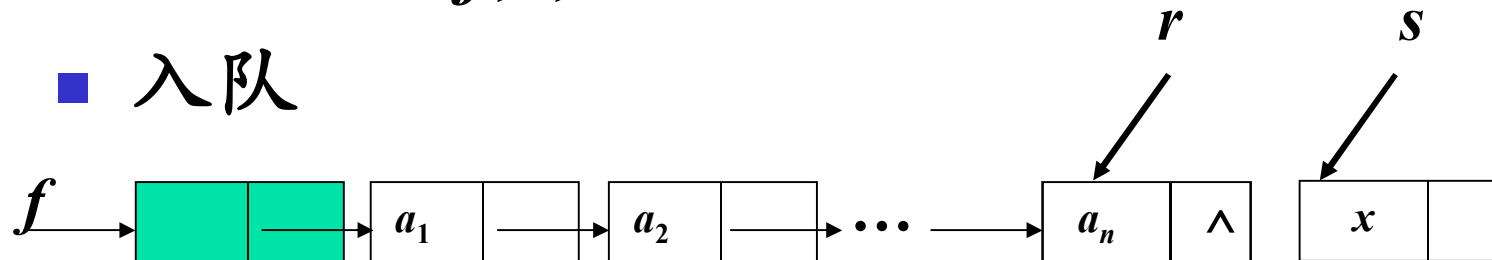


$s = (\text{LinkList})\text{malloc}(\text{sizeof}(\text{Node}));$

链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义: 采用链式存储结构存放
- $\text{LinkList } f, r;$
- 入队

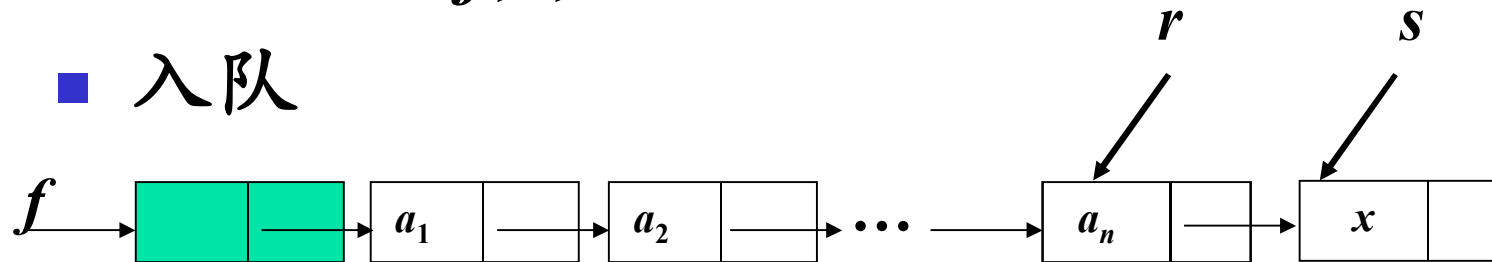


```
s=(LinkedList)malloc(sizeof(Node));  
s->data=x;
```


链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义: 采用链式存储结构存放
- $\text{LinkList } f, r;$
- 入队

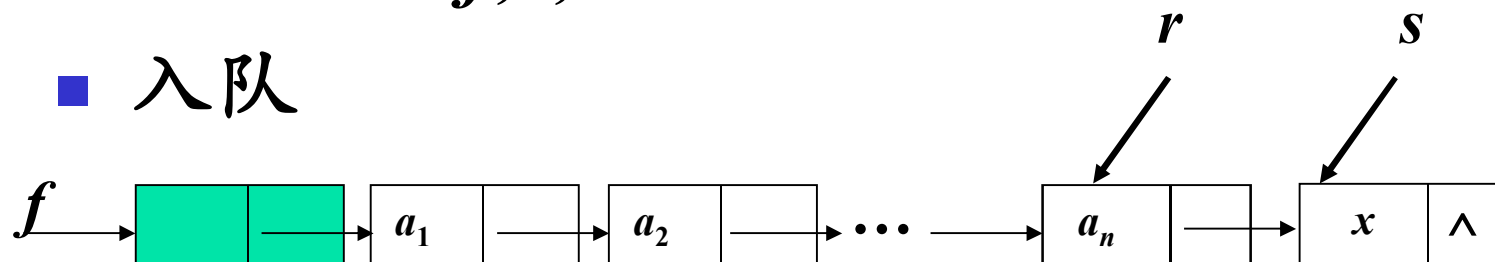


```
s=(LinkedList)malloc(sizeof(Node));  
s->data=x;  
r->next=s;
```

链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义: 采用链式存储结构存放
- $\text{LinkList } f, r;$
- 入队

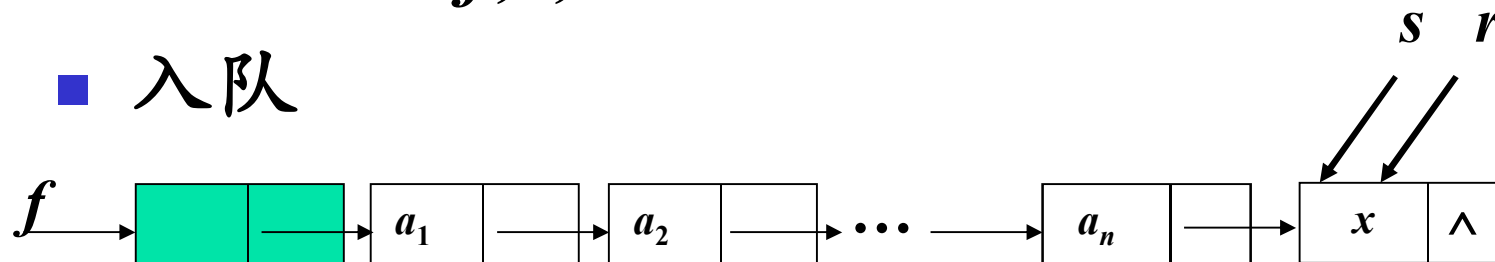


```
s=(LinkedList)malloc(sizeof(Node));  
s->data=x;  
r->next=s;s->next=NULL;
```

链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义: 采用链式存储结构存放
- $\text{LinkList } f, r;$
- 入队

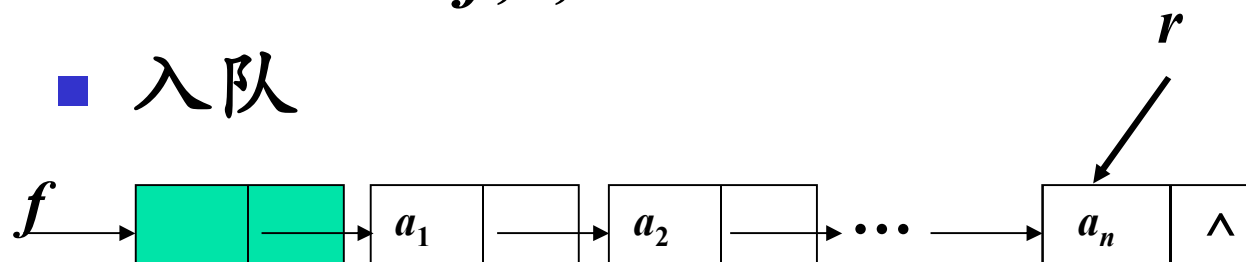


```
s=(LinkedList)malloc(sizeof(Node));  
s->data=x;  
r->next=s;s->next=NULL;  
r=s;
```

链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义: 采用链式存储结构存放
- $\text{LinkList } f, r;$
- 入队

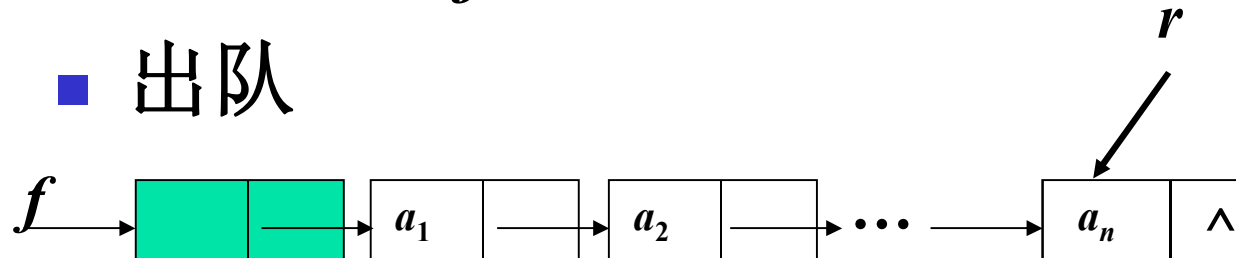


说明: f 为队首指针,
指示链队的队首位置;

r 为队尾指针, 指示链队
的队尾位置

链队 — $f \rightarrow \text{next} \neq \text{NULL}$

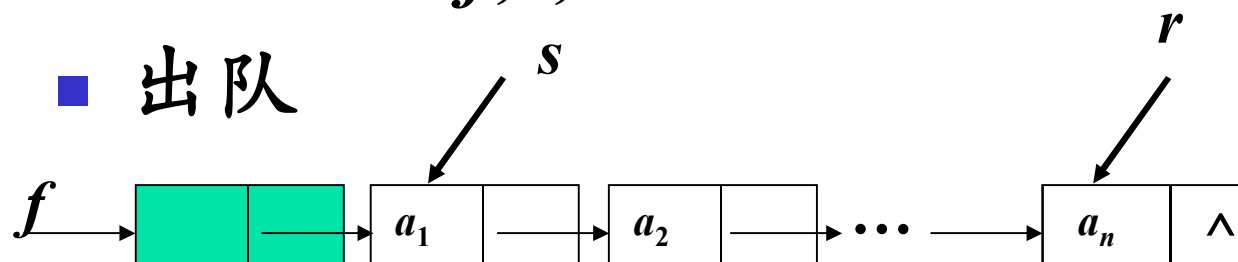
- 定义
- $\text{LinkList } f, r;$
- 出队



链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义: 采用链式存储结构存放
- `LinkedList f, r ;`
- 出队

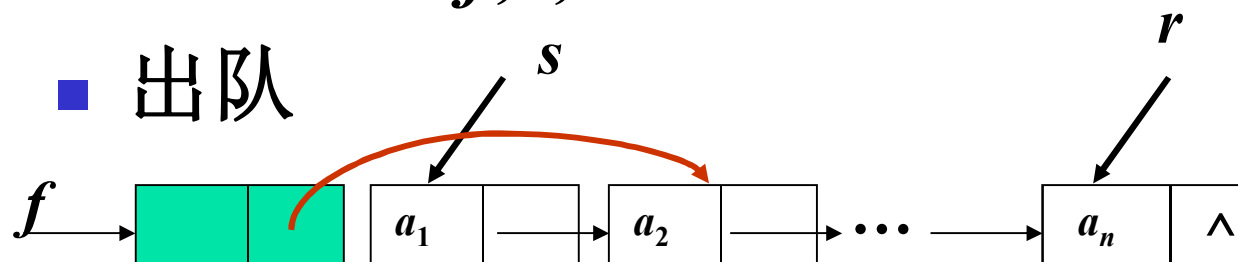


$s = f \rightarrow \text{next};$

链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义
- $\text{LinkList } f, r;$
- 出队

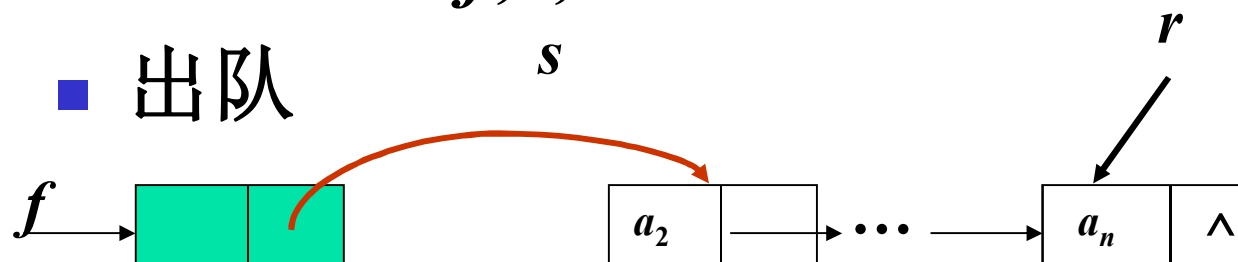


$s = f \rightarrow \text{next};$
 $f \rightarrow \text{next} = s \rightarrow \text{next};$

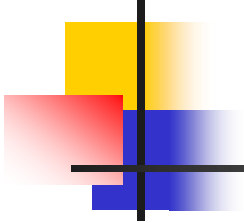
链队

说明: f 为队首指针,
指示链队的队首位置;
 r 为队尾指针, 指示链队
的队尾位置

- 定义
- $\text{LinkList } f, r;$
- 出队



$s = f \rightarrow \text{next};$
 $f \rightarrow \text{next} = s \rightarrow \text{next};$
 $\text{free}(s);$



双端队列：是限定插入和删除运算在表的两端进行的线性表，它好像一个特别书架，取、存书限定在两边进行。

超队列：是一种输入受限的双端队列，即删除仅可在一端进行，而插入仍允许在两端进行。它好像一种特殊的队列，允许有的刚插入的元素就可删除。