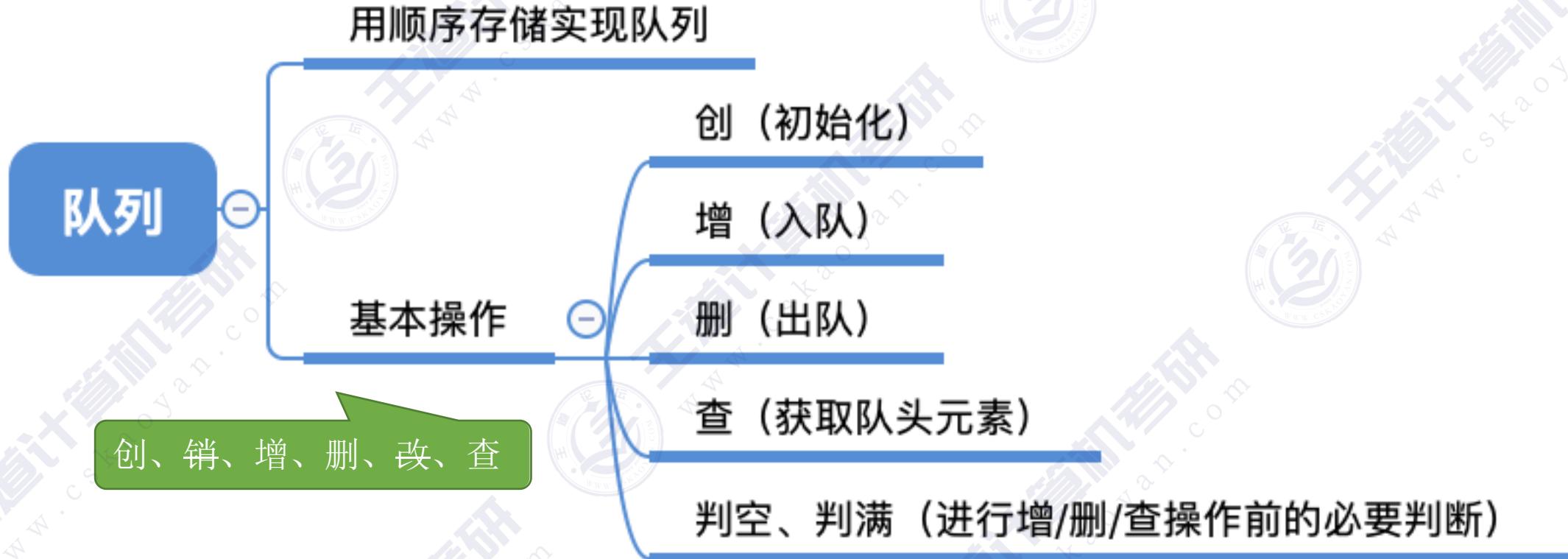


本节内容

队列

顺序实现

知识总览



队列的顺序实现

```
#define MaxSize 10
typedef struct{
    ElemType data[MaxSize];
    int front, rear;
} SqQueue;
```

Sq: sequence —— 顺序

rear

英 [riə(r)] 美 [rɪr]

n. 后面; 后方部队; 屁股

adj. 后方的, 后面的; 背面的

```
void testQueue(){
    SqQueue Q; // 声明一个队列 (顺序存储)
    // ...后续操作...
}
```

// 定义队列中元素的最大个数

// 用静态数组存放队列元素
// 队头指针和队尾指针

连续的存储空间, 大小
 $MaxSize * \text{sizeof}(ElemType)$

front

英 [frənt] 美 [frənt]

n. 前面; 正面; 前线

vt. 面对; 朝向; 对付

vi. 朝向

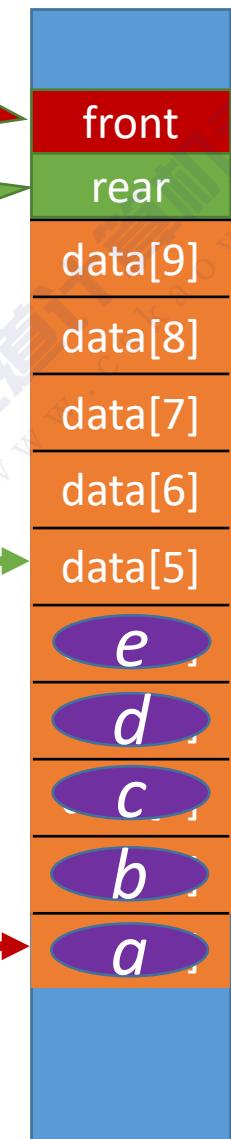
adj. 前面的; 正面的

指向队尾元素的后一个位置
(下一个应该插入的位置)

rear

指向队头元素

front



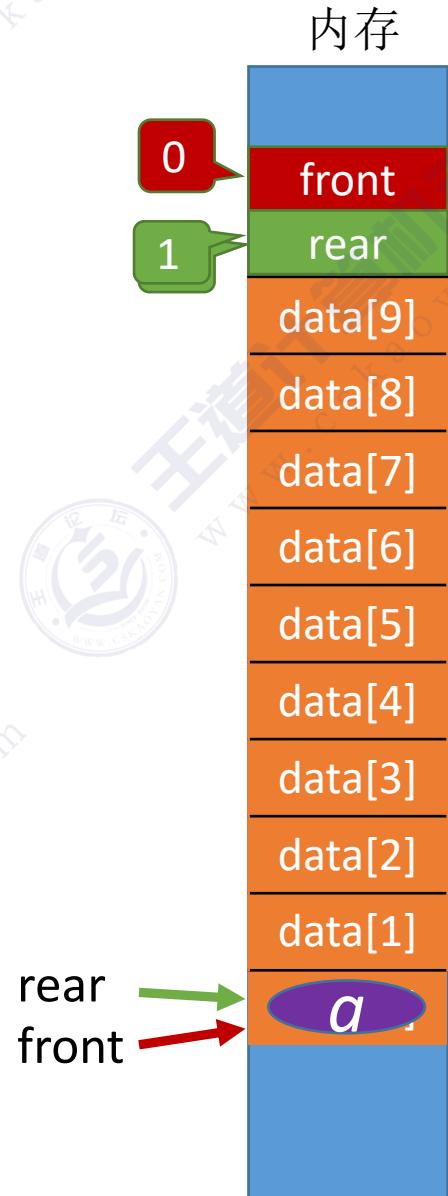
初始化操作

```
#define MaxSize 10           // 定义队列中元素的最大个数
typedef struct{
    ELEMTYPE data[MaxSize]; // 用静态数组存放队列元素
    int front, rear;        // 队头指针和队尾指针
} SqQueue;

// 初始化队列
void InitQueue(SqQueue &Q){
    // 初始时 队头、队尾指针指向0
    Q.rear=Q.front=0;
}

void testQueue(){
    // 声明一个队列 (顺序存储)
    SqQueue Q;
    InitQueue(Q);
    // ... 后续操作...
}
```

增删改查



入队操作

只能从队尾入队（插入）

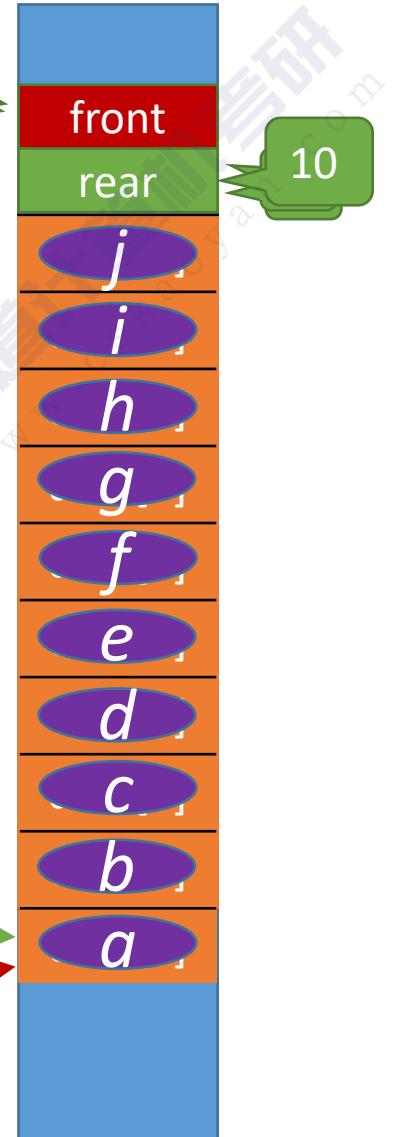
内存

```
#define MaxSize 10
typedef struct{
    ElecType data[MaxSize];
    int front, rear;
} SqQueue;
```

//定义队列中元素的最大个数

//用静态数组存放队列
//队头指针和队尾指针

队列已满的条件:
rear==MaxSize ???



//入队

```
bool EnQueue(SqQueue &Q, ElecType x){
    if(队列已满)
        return false; //队满则报错
    Q.data[Q.rear]=x; //将x插入队尾
    Q.rear=Q.rear+1; //队尾指针后移
    return true;
}
```

rear
front

入队操作

只能从队尾入队（插入）

内存

```
#define MaxSize 10          // 定义队列中元素的最大个数
typedef struct{
    ElemType data[MaxSize]; // 用静态数组存放队列元素
    int front, rear;        // 队头指针和队尾指针
} SqQueue;

// 入队
bool EnQueue(SqQueue &Q, ElemType x){
    if(队列已满)
        return false;           // 队满则报错
    Q.data[Q.rear]=x;           // 新元素插入队尾
    Q.rear=(Q.rear+1)%MaxSize; // 队尾指针加1取模
    return true;
}
```

$\{0, 1, 2, \dots, \text{MaxSize}-1\}$
将存储空间在逻辑上
变成了“环状”

跨考Tips: 取模运算, 即取余运算。两个整数 a, b , $a \% b = a$ 除以 b 的余数
在《数论》中, 通常表示为 $a \text{ MOD } b$

rear

front

3

0

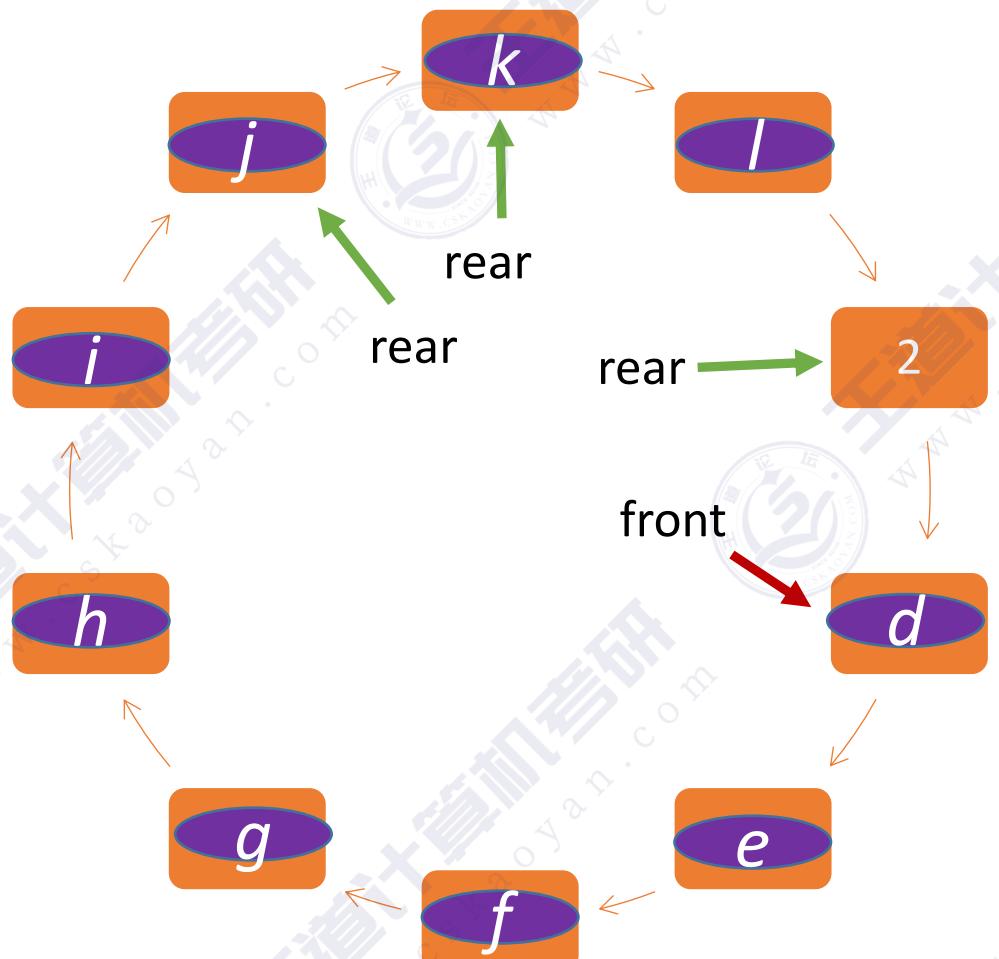
模运算将无限的整数
域映射到有限的整数
集合 $\{0, 1, 2, \dots, b-1\}$ 上

循环队列

用模运算将存储空间在逻辑上变成了“环状”

→ `Q.data[Q.rear]=x;`
→ `Q.rear=(Q.rear+1)%MaxSize;`

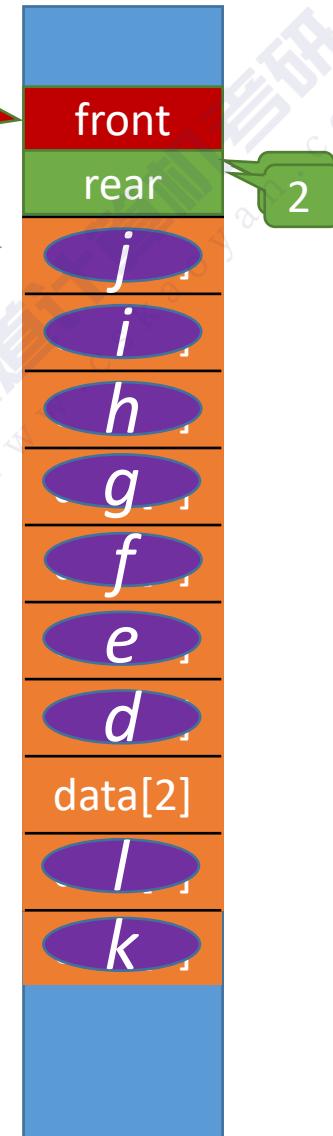
//新元素插入队尾
//队尾指针加1取模



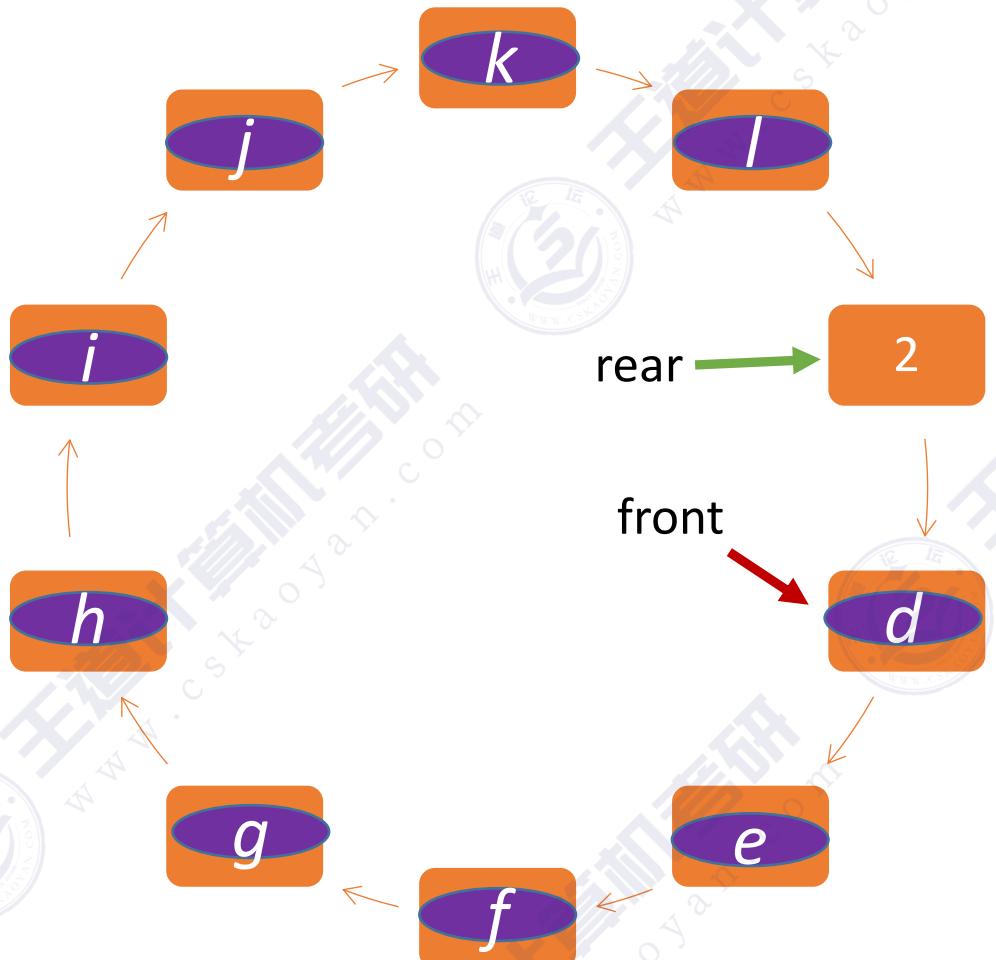
队列已满的条件：队尾指针的再下一个位置是队头，即
 $(Q.rear+1)\%MaxSize==Q.front$

代价：牺牲一个存储单元

rear →
front →



循环队列——入队操作



//判断队列是否为空

bool QueueEmpty(SqQueue Q){

if(Q.rear==Q.front) //队空条件

return true;

else

return false;

判断队空

//入队

bool EnQueue(SqQueue &Q, **ElemType** x){

if((Q.rear+1)%**MaxSize**==Q.front)

return false;

//队满则报错

Q.data[Q.rear]=x;

//新元素插入队尾

Q.rear=(Q.rear+1)%MaxSize;

//队尾指针加1取模

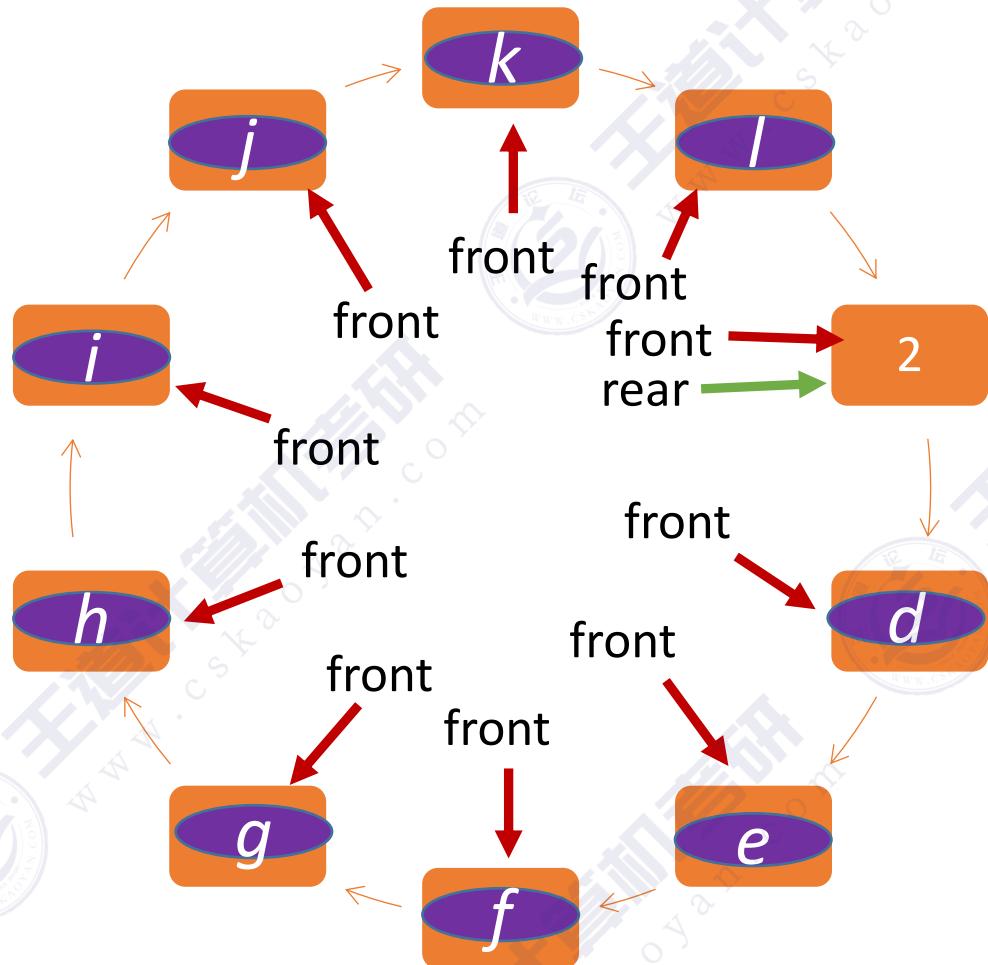
return true;

判断队满

用模运算将存储空间在逻辑上变成了“环状”

循环队列——出队操作

只能让队头元素出队



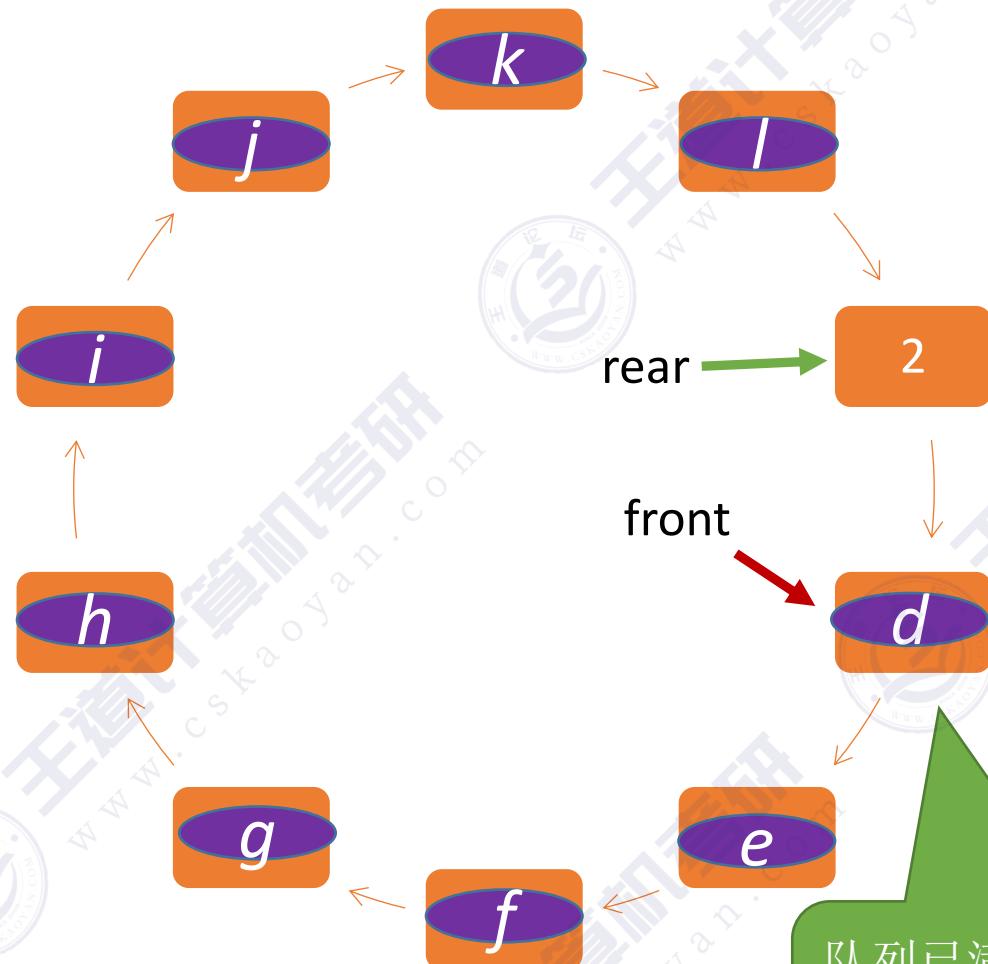
```
//出队 (删除一个队头元素, 并用x返回)
bool DeQueue(SqQueue &Q, ElemType &x){
    if(Q.rear==Q.front) //判断队空
        return false; //队空则报错
    x=Q.data[Q.front];
    Q.front=(Q.front+1)%MaxSize;
    return true;
}
```

```
//获得队头元素的值, 用x返回
bool GetHead(SqQueue Q, ElemType &x){
    if(Q.rear==Q.front)
        return false; //队空则报错
    x=Q.data[Q.front];
    return true;
}
```

队列元素个数:

$$(\text{rear}+\text{MaxSize}-\text{front})\% \text{MaxSize}$$

方案一：判断队列已满/已空



队列已满的条件: 队尾指针的再下一个位置是队头, 即
 $(\text{Q}.\text{rear}+1)\% \text{MaxSize}==\text{Q}.\text{front}$

```
#define MaxSize 10
```

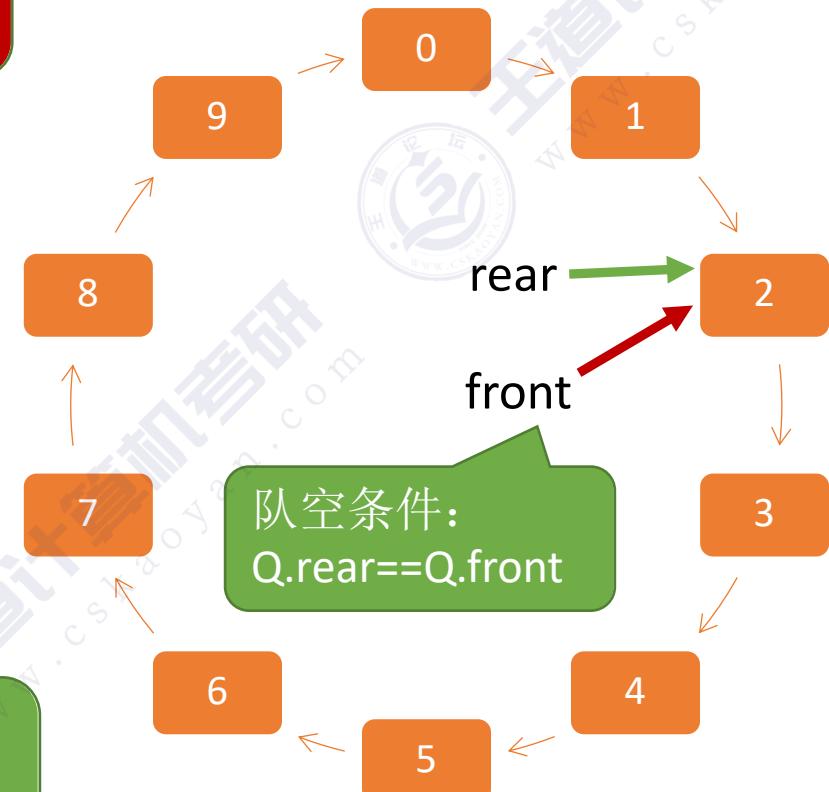
```
typedef struct{
```

```
ElemType data[MaxSize];
```

```
int front, rear;
```

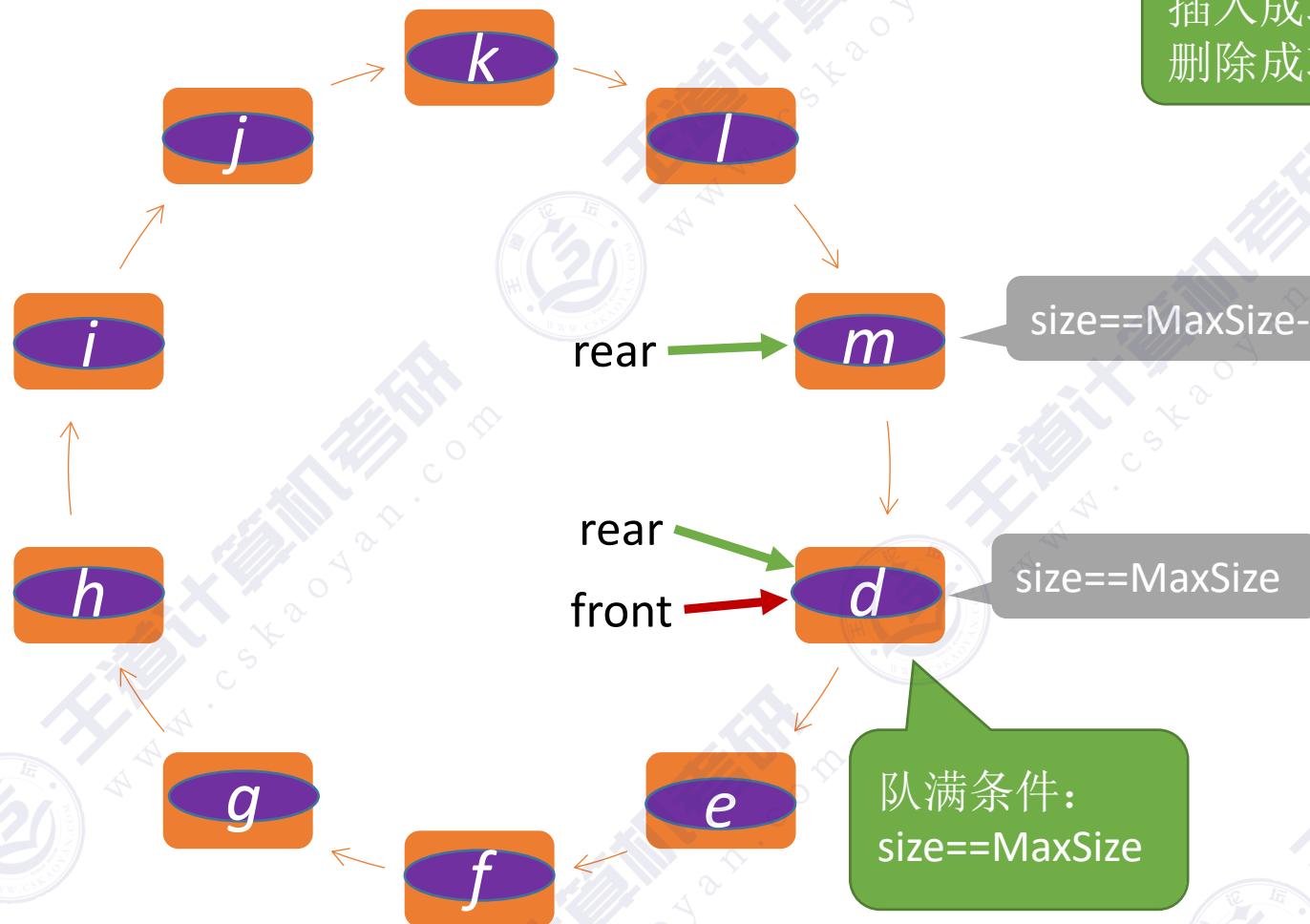
```
} SqQueue;
```

初始化时
rear=front=0



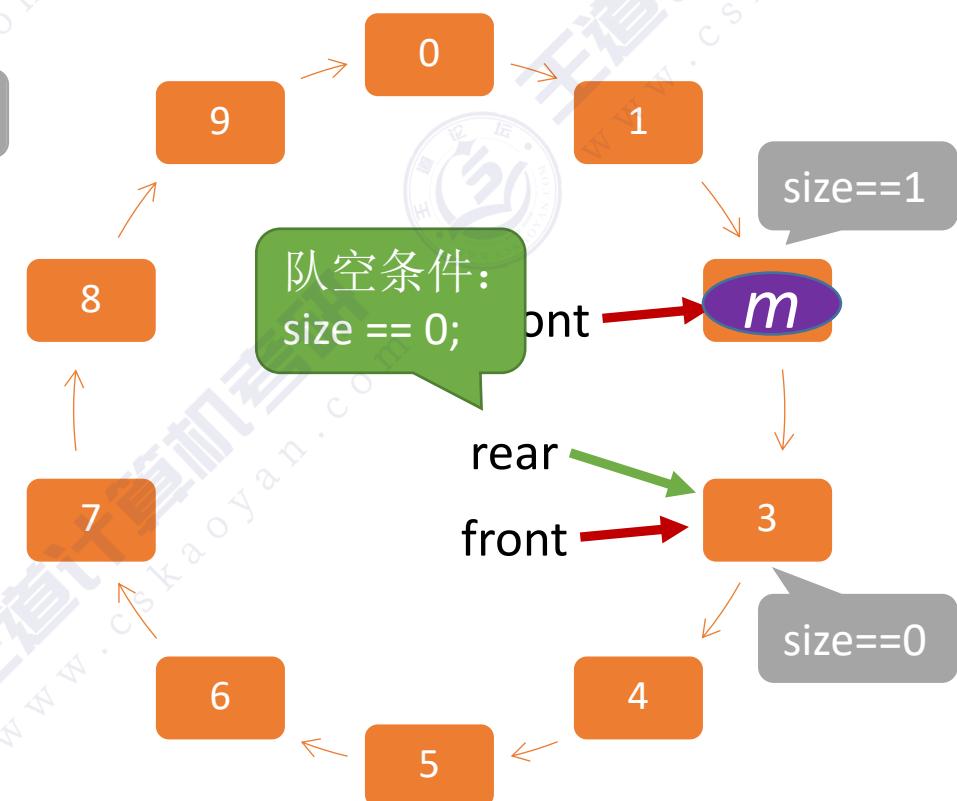
方案二：判断队列已满/已空

队列元素个数 = size



```
#define MaxSize 10
typedef struct{
    ElemType data[MaxSize];
    int front, rear;
    int size;
} SqQueue;
```

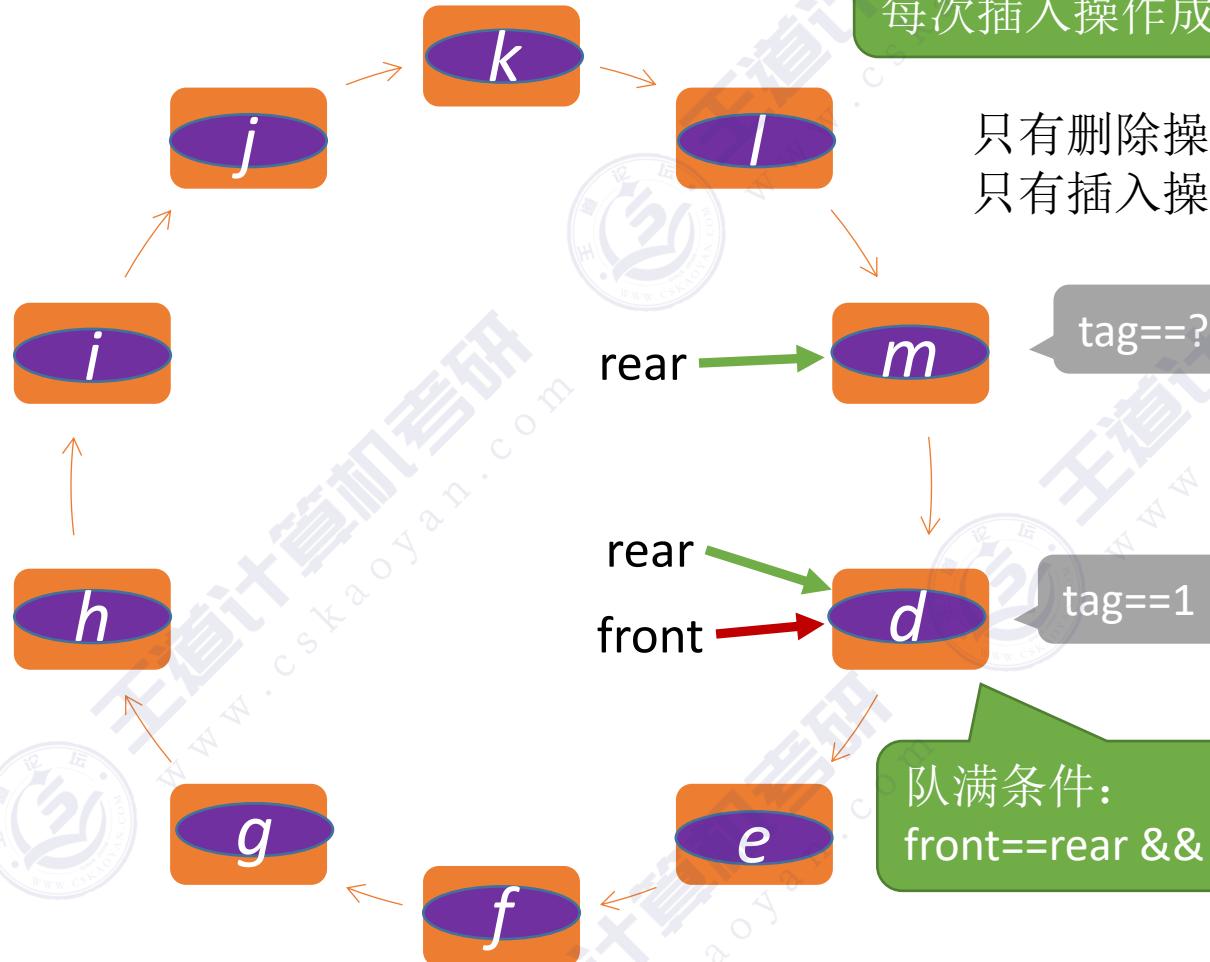
初始化时
rear=front=0;
size = 0;



方案三：判断队列已满/已空

队列元素个数：

$(\text{rear}+\text{MaxSize}-\text{front})\% \text{MaxSize}$

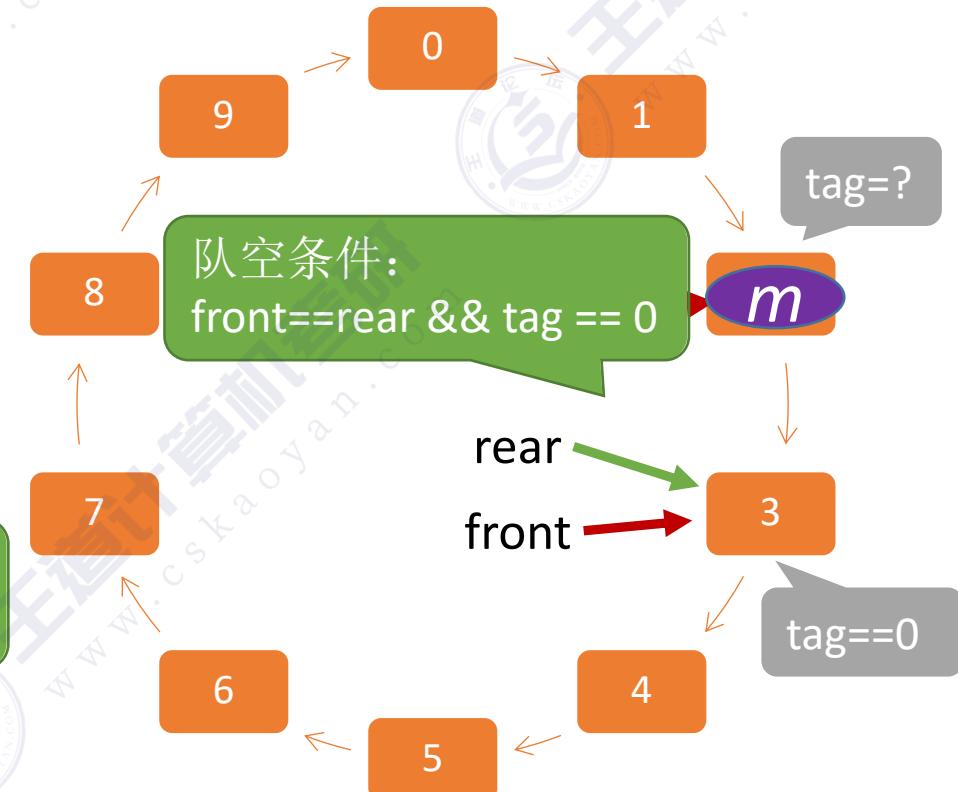


每次删除操作成功时，都令tag=0；
每次插入操作成功时，都令tag=1；

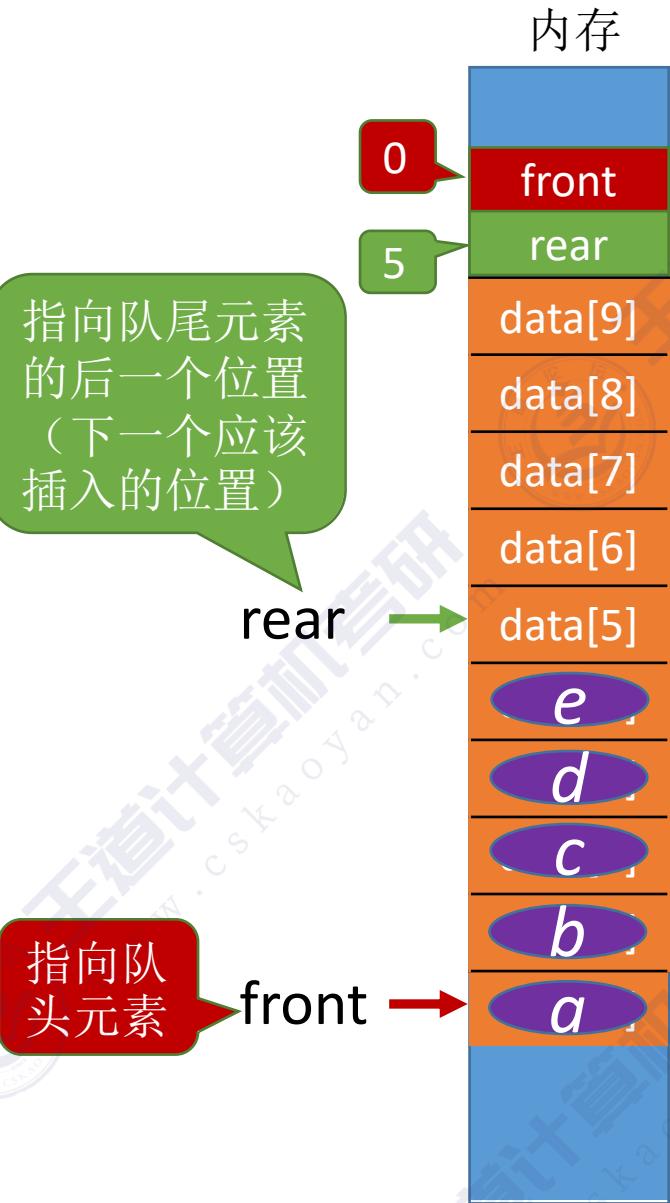
只有删除操作，才可能导致队空
只有插入操作，才可能导致队满

```
#define MaxSize 10
typedef struct{
    ElemType data[MaxSize];
    int front, rear;
    int tag; //最近进行的是删除/插入
} SqQueue;
```

初始化时
rear=front=0;
tag = 0;



其他出题方法



入队操作:

```
Q.data[Q.rear]=x;  
Q.rear=(Q.rear+1)%MaxSize;
```

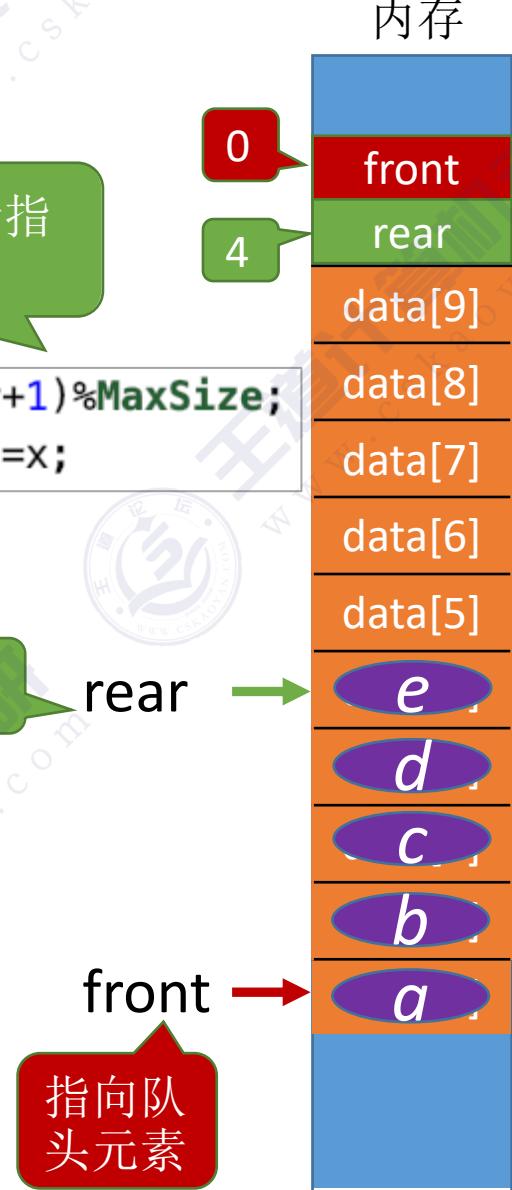
初始时队尾指针指向哪更合理?

```
Q.rear=(Q.rear+1)%MaxSize;  
Q.data[Q.rear]=x;
```

入队导致
rear 后移

出队操作

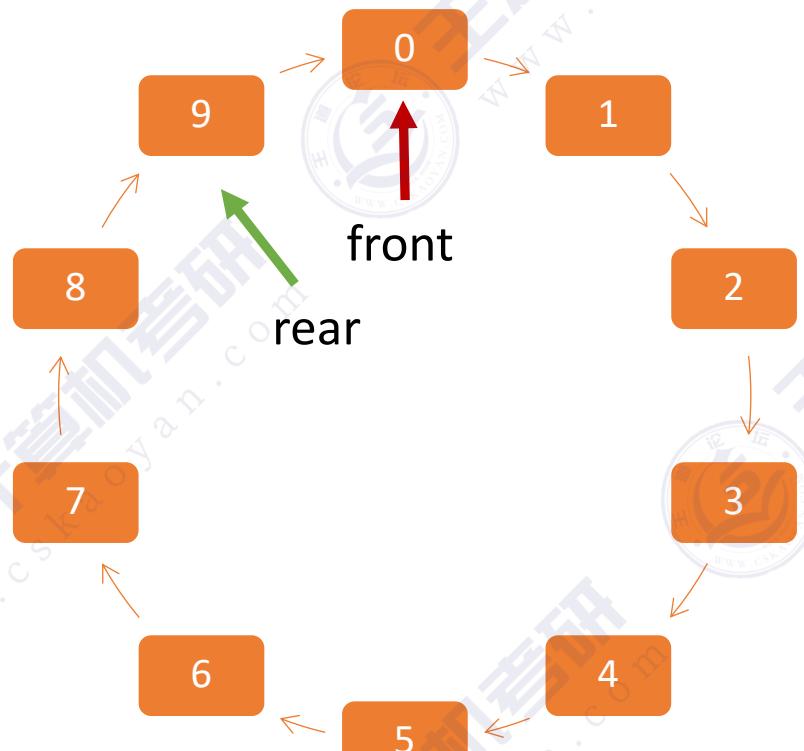
```
x=Q.data[Q.front];  
Q.front=(Q.front+1)%MaxSize;
```



其他出题方法

判空:

$(Q.\text{rear}+1) \% \text{MaxSize} == Q.\text{front}$



初始时队尾指针指向哪更合理?

```
Q.\text{rear}=(Q.\text{rear}+1) \% \text{MaxSize};  
Q.\text{data}[Q.\text{rear}]=x;
```

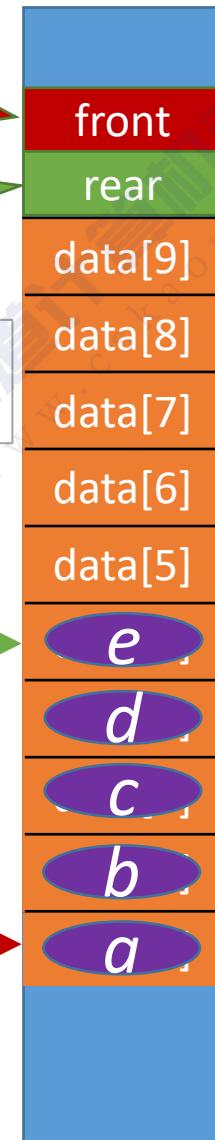
指向队尾元素

rear →

front →

指向队头元素

内存

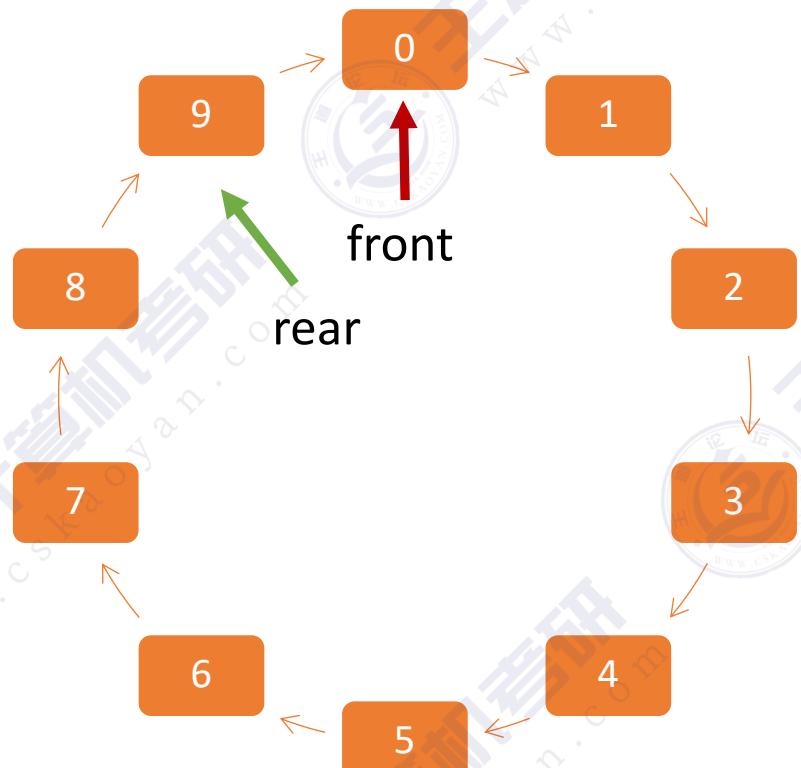


其他出题方法

方案一：牺牲一个存储单元
方案二：增加辅助变量

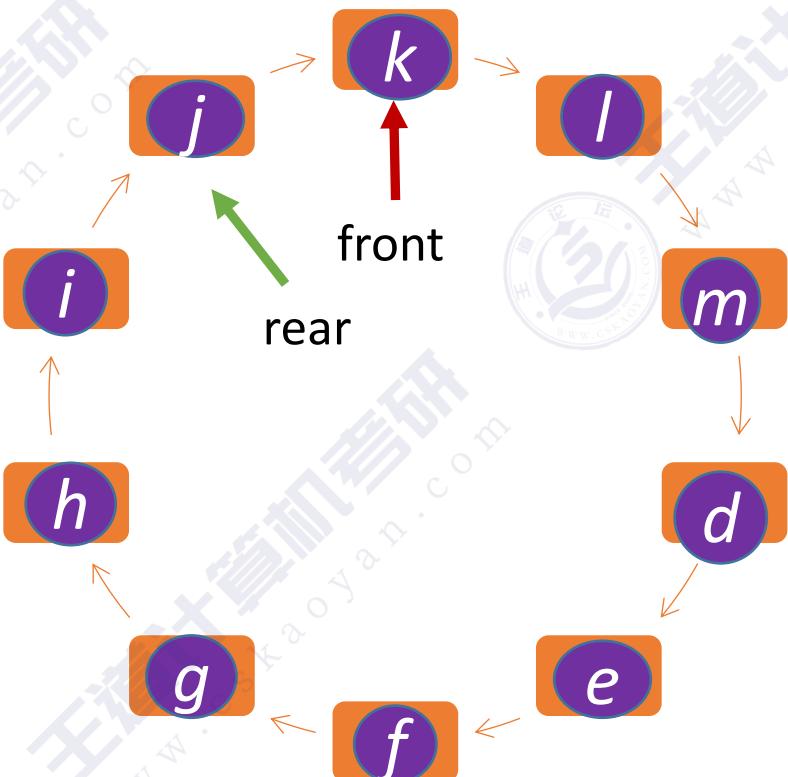
判空：

$(Q.\text{rear}+1)\% \text{MaxSize} == Q.\text{front}$



判满：

$(Q.\text{rear}+1)\% \text{MaxSize} == Q.\text{front}$?



知识回顾与重要考点

队列的顺序实现

实现思想

用静态数组存放数据元素，设置队头/队尾(front/rear)指针

重要考点

循环队列：用模运算（取余）将存储空间在逻辑上变为“环状”

`Q.rear=(Q.rear+1)%MaxSize;`

如何 初始化、入队、出队

如何 判空、判满

如何计算队列的长度

思考：分别采用

①a、①b、①c

②a、②b、②c

策略时，这些操作怎么实现？

分析思路

确定front、rear指针的指向

①rear指向队尾元素后一个位置

②rear指向队尾元素

确定判空
判满的方法

a. 牺牲一个存储单元

b. 增加 size 变量记录队列长度

c. 增加 tag = 0/1 用于标记
最近的一次操作是 出队/入队

...