1. 掌握栈和队列的特点，并能在相应的应用问题中正确选用
2. 熟练掌握栈的两种存储结构的基本操作实现算法，特别应注意栈满和栈空的条件
3. 熟练掌握循环队列和链队列的基本操作实现算法，特别注意队满和队空的条件
4. 理解递归算法执行过程中栈的状态变化过程
5. 掌握表达式求值 方法

六星教育
sixstaredu.com

第06讲 栈和队列之《队列表示与实现》

六星教育首席架构师：Vico老师
官方助理冰芯老师QQ：1930070991

3.1 栈和队列的定义和特点

3.2 栈的表示和操作的实现

3.3 栈与递归

3.4 队列的的表示和操作的实现

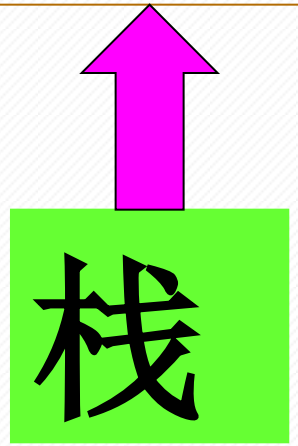
3.3 栈与递归

- **递归的定义** 若一个对象部分地包含它自己， 或用它自己给自己定义， 则称这个对象是递归的；若一个过程**直接地或间接地调用自己**，则称这个过程是递归的过程。

```
long Fact ( long n ) {
    if ( n == 0) return 1;
    else return n * Fact (n–1); }
```

当多个函数构成嵌套调用时，遵循

后调用先返回



- 以下三种情况常常用到递归方法
  - 递归定义的数学函数
  - 具有递归特性的数据结构
  - 可递归求解的问题

1. 递归定义的数学函数:

- 阶乘函数:

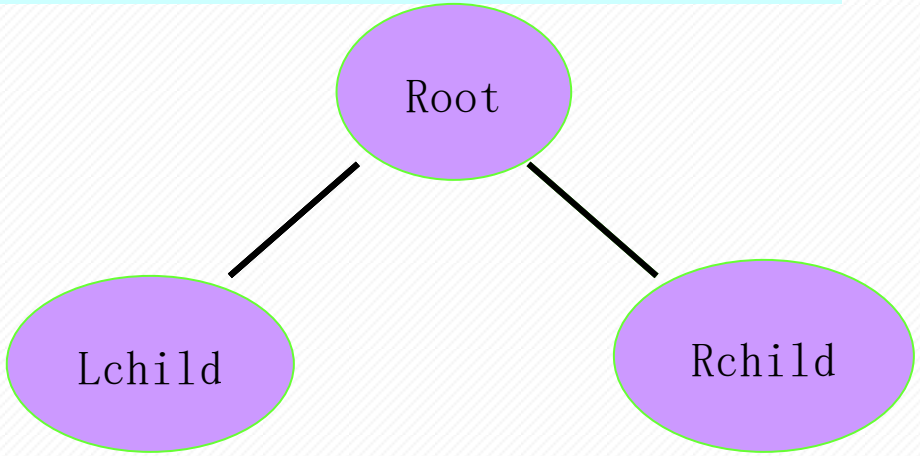
$$Fact(n)=\begin{cases}1 & 若\mathrm{n}=0\\ n\cdot Fact(n-1) & 若\mathrm{n}>0\end{cases}$$

- 2阶Fibonaci数列:

$$Fib(n)=\begin{cases}1 & 若\mathrm{n}=1或2\\ Fib(n-1)+Fib(n-2) & 其它\end{cases}$$

2. 具有递归特性的数据结构:

- 树



- 广义表

A=(a, A)

3. 可递归求解的问题:

- 迷宫问题

用分治法求解递归问题

分治法：对于一个较为复杂的问题，能够分解成几个相对简单的且解法相同或类似的子问题来求解

必备的三个条件

- 1、能将一个问题转变成一个新问题，而新问题与原问题的解法相同或类同，不同的仅是处理的对象，且这些处理对象是变化有规律的

- 2、可以通过上述转化而使问题简化

- 3、必须有一个明确的递归出口，或称递归的边界

分治法求解递归问题算法的一般形式：

```
void   p (参数表) {
  if   （递归结束条件）可直接求解步骤；-----基本项
  else  p（较小的参数）；------归纳项
 }
```

```
long Fact ( long n ) {
    if ( n == 0) return 1;//基本项
    else return n * Fact (n－1); //归纳项}
```

求解阶乘 n! 的过程

```
if ( n == 0 ) return 1;
else return n * Fact (n–1);
```

函数调用过程

调用前，系统完成：

(1)将实参,返回地址等传递给被调用函数

(2)为被调用函数的局部变量分配存储区

(3)将控制转移到被调用函数的入口

调用后，系统完成：

(1)保存被调用函数的计算结果

(2)释放被调用函数的数据区

(3)依照被调用函数保存的返回地址将控制转移到调用函数

递归的优缺点

优点：结构清晰，程序易读

缺点：每次调用要生成工作记录，保存状态信息，入栈；
返回时要出栈，恢复状态信息。时间开销大。

3.4 队列的表示和操作的实现

ADT Queue {

队列的抽象数据类型

数据对象: $D=\{a_i \mid a_i \in ElemSet, i=1,2,\cdots,n,\ n\geq 0\}$

数据关系: $R_1=\{<a_{i-1},a_i> \mid a_{i-1},a_i \in D, i=1,2,\cdots,n\}$

约定a_1端为队列头,a_n端为队列尾

基本操作:

(1) InitQueue (&Q) //构造空队列

(2) DestroyQueue (&Q) //销毁队列

(3) ClearQueue (&S) //清空队列

(4) QueueEmpty(S) //判空. 空--TRUE,

队列的抽象数据类型

(5) QueueLength(Q) //取队列长度

(6) GetHead (Q,&e) //取队头元素,

(7) EnQueue (&Q,e) //入队列

(8) DeQueue (&Q,&e) //出队列

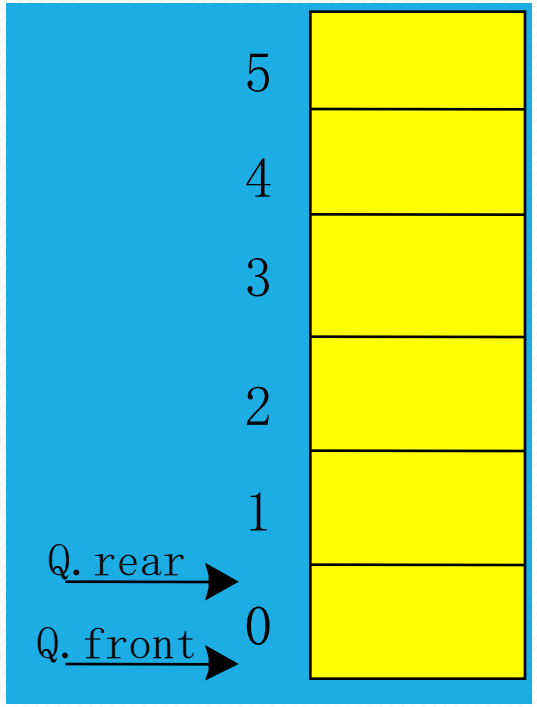
(9) QueueTraverse(Q,visit()) //遍历

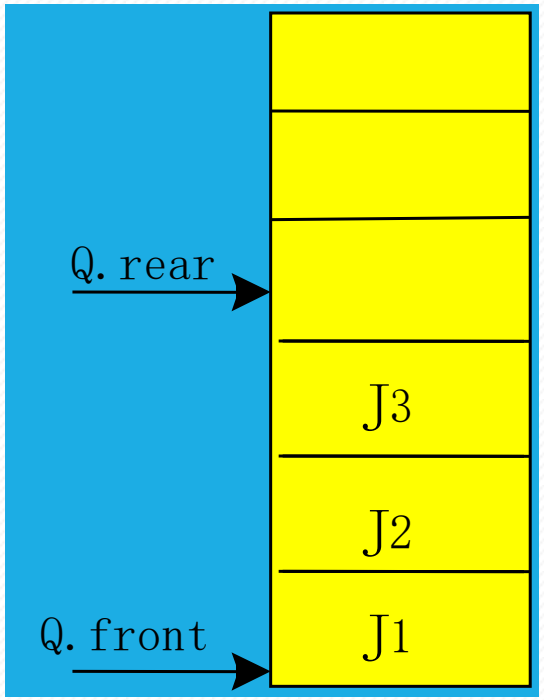
}ADT Queue

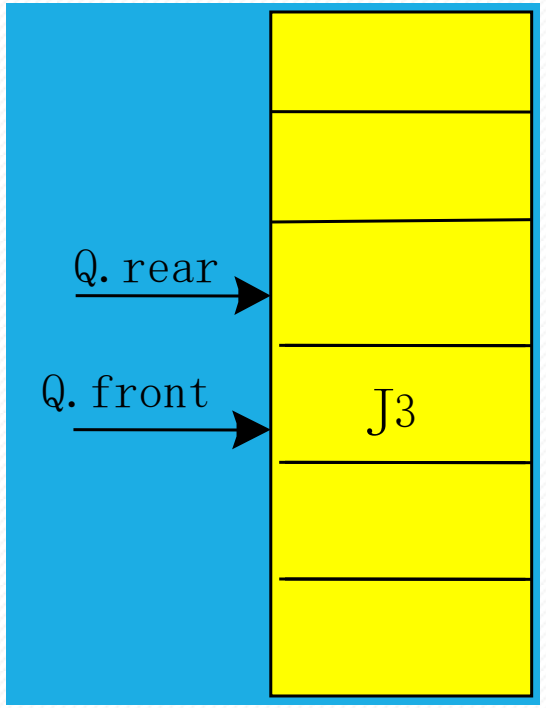
队列的顺序表示——用一维数组base[M]

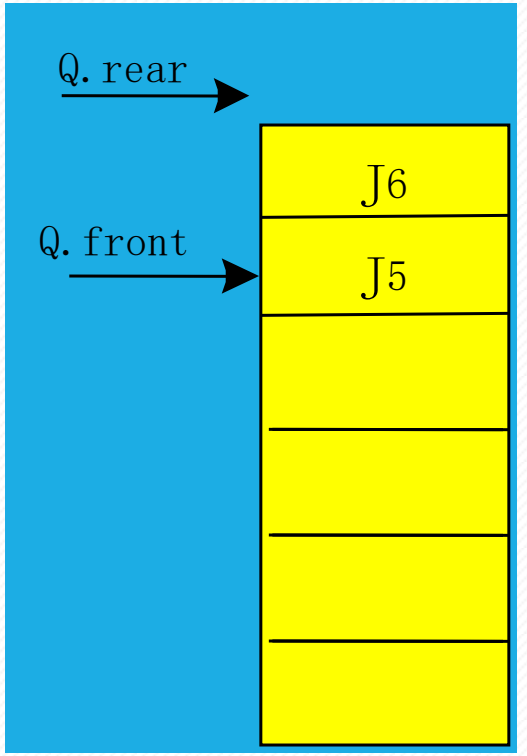

```
#define M  100   //最大队列长度
Typedef struct {
   QElemType *base;  //初始化的动态分配存储空间
   int  front;            //头指针
   int  rear;             //尾指针
}SqQueue;
```

队列的顺序表示——用一维数组base[M]









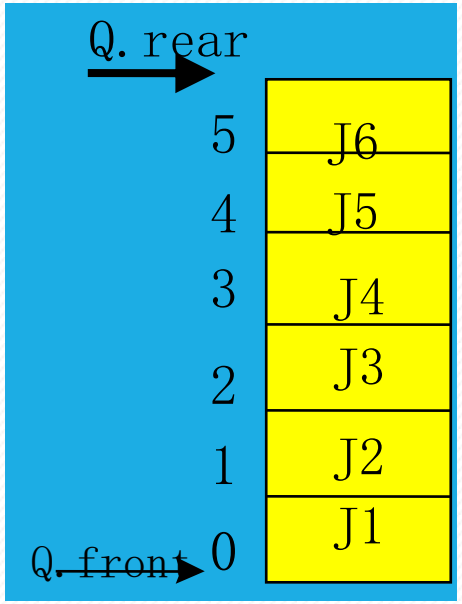
front=rear=0

空队标志：front= =rear
入队：base[rear++]=x;
出队：x=base[front++];

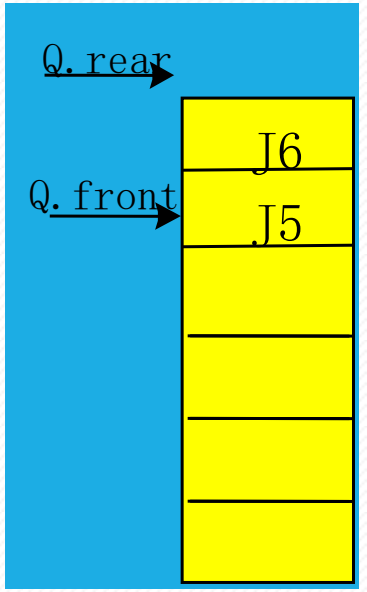
存在的问题

六星教育
sixstaredu.com

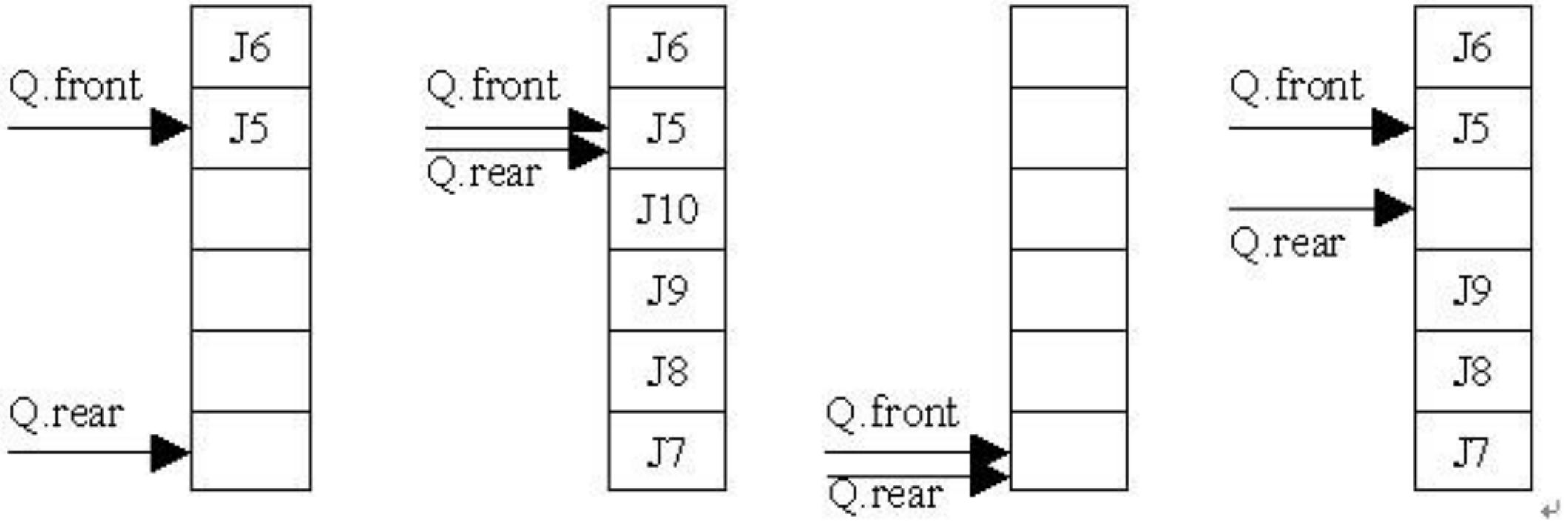
设大小为M



front=0
rear=M时
再入队—真溢出



front≠0
rear=M时
再入队—假溢出

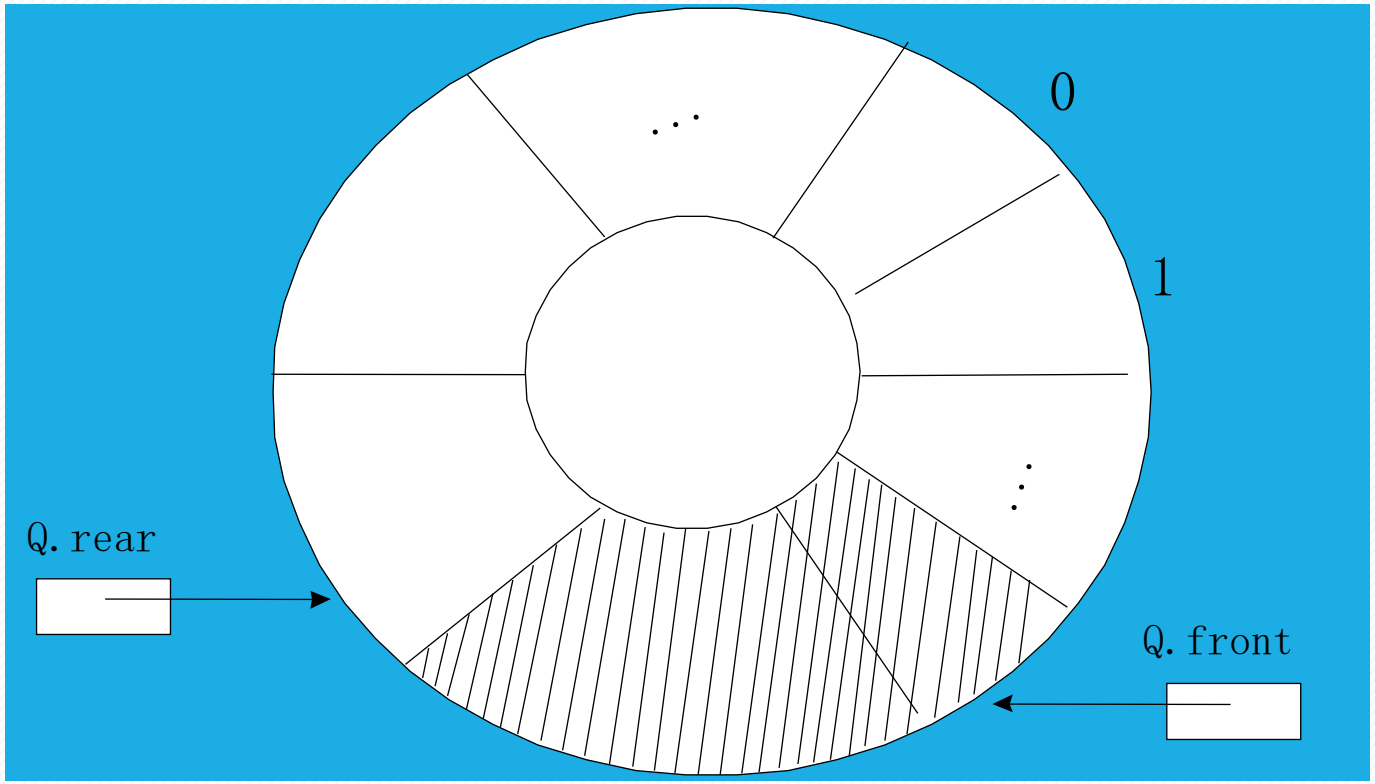


（a）一般情况　（b）队列空间被占满　（c）空队　（d）呈“满”状态的循环队列

图 3.12　循环队列中头、尾指针和元素之间的关系

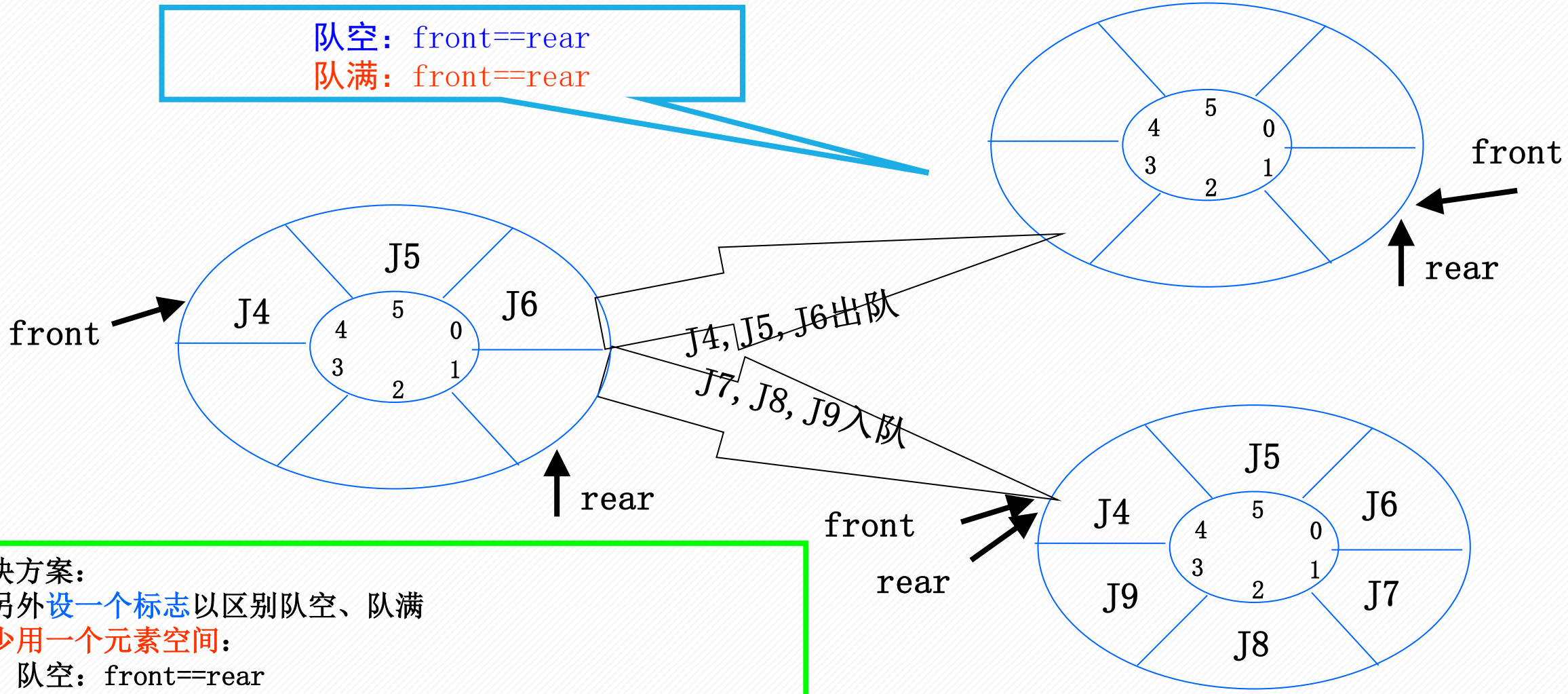
解决的方法——循环队列

base[0]接在base[M-1]之后
若rear+1==M
则令rear=0;



实现：利用“模”运算
入队：
```
base[rear]=x;
rear=(rear+1)%M;
```
出队：
```
x=base[front];
front=(front+1)%M;
```

队空：front==rear
队满：front==rear



解决方案：
1. 另外设一个标志以区别队空、队满
2. 少用一个元素空间：
   队空：front==rear
   队满：(rear+1)%M==front

循环队列


```
#define MAXQSIZE  100        //最大长度
typedef struct {
   QElemType *base;          //初始化的动态分配存储空间
   int  front;               //头指针
   int  rear;                //尾指针
}SqQueue;
```

循环队列初始化


```
Status InitQueue (SqQueue &Q){
    Q.base =new QElemType[MAXQSIZE]   //申请空间
   if(!Q.base) exit(OVERFLOW);
    Q.front=Q.rear=0;                 //队空
    return OK;
}
```

求循环队列的长度


```
int  QueueLength (SqQueue Q){
   return (Q.rear-Q.front+MAXQSIZE)%MAXQSIZE;
}
```

循环队列入队

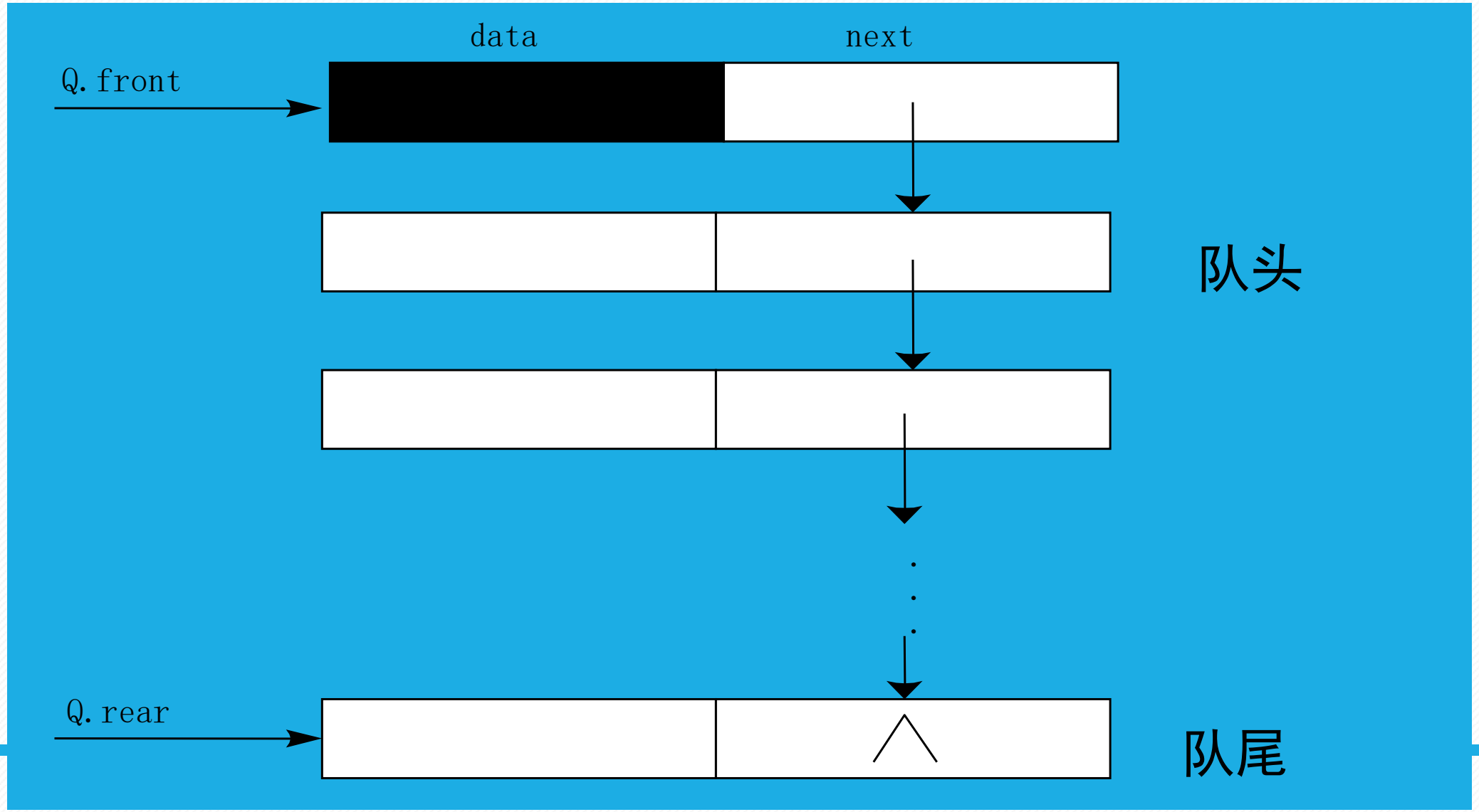

```
Status EnQueue(SqQueue &Q,QElemType e){
    if((Q.rear+1)%MAXQSIZE==Q.front)  return ERROR; //队满，无法添加
    Q.base[Q.rear]=e; //插入元素
    Q.rear=(Q.rear+1)%MAXQSIZE; //队尾指针+1
     return OK;
}
```

循环队列出队


```
Status DeQueue (LinkQueue &Q,QElemType &e){
    if(Q.front==Q.rear) return ERROR; //队空，无法删除
    e=Q.base[Q.front];
    Q.front=(Q.front+1)%MAXQSIZE; //队头指针+1
    return OK;
}
```

项目实战操作案例1

链队列

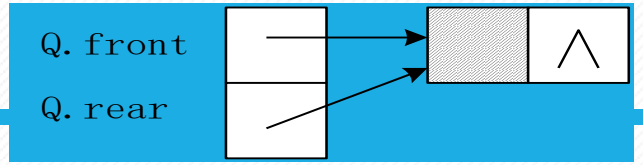


链队列

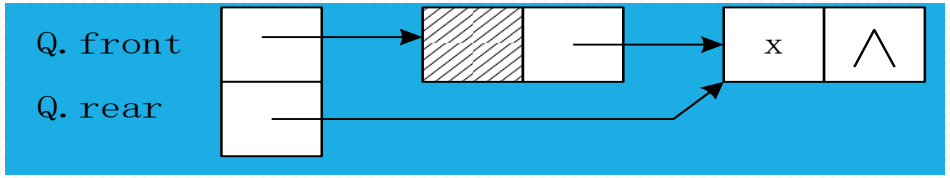

```
typedef struct QNode{
    QElemType    data;
    struct Qnode  *next;
}Qnode, *QueuePtr;

typedef struct {
    QueuePtr  front;          //队头指针
    QueuePtr  rear;           //队尾指针
}LinkQueue;
```

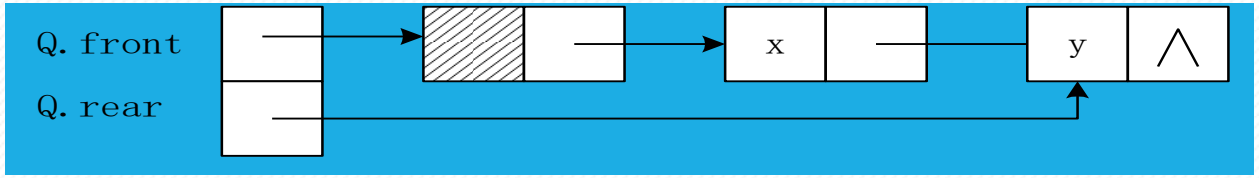
链队列



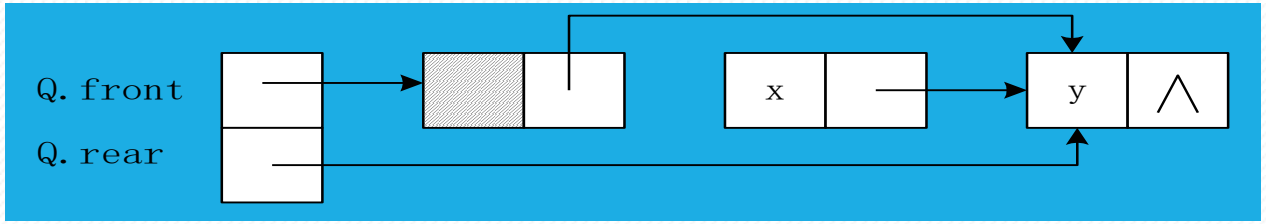
(a) 空队列



(b) 元素x入队列



(c) 元素y入队列



(d) 元素x出队列

链队列初始化


```
Status InitQueue (LinkQueue &Q){
   Q.front=Q.rear=(QueuePtr) malloc(sizeof(QNode));
    if(!Q.front) exit(OVERFLOW);
    Q.front->next=NULL;
     return OK;
}
```

销毁链队列


```
Status DestroyQueue (LinkQueue &Q){
    while(Q.front){
        Q.rear=Q.front->next;
        free(Q.front);
        Q.front=Q.rear;   }
    return OK;
}
```

判断链队列是否为空

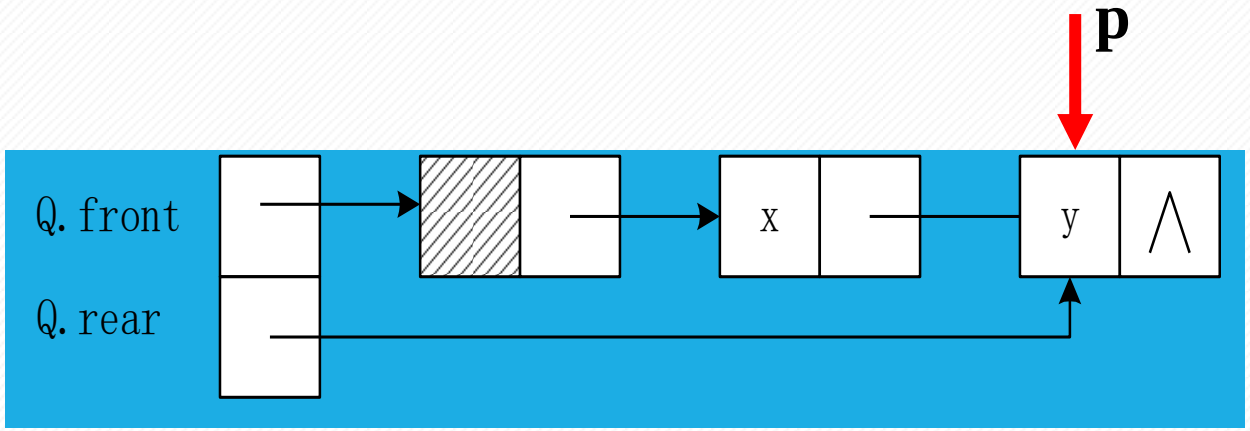

```
Status QueueEmpty (LinkQueue Q){
   return (Q.front==Q.rear);
}
```

求链队列的队头元素


```
Status GetHead (LinkQueue Q, QElemType &e){
    if(Q.front==Q.rear) return ERROR; //队空，无法删除
    e=Q.front->next->data;
    return OK;
}
```

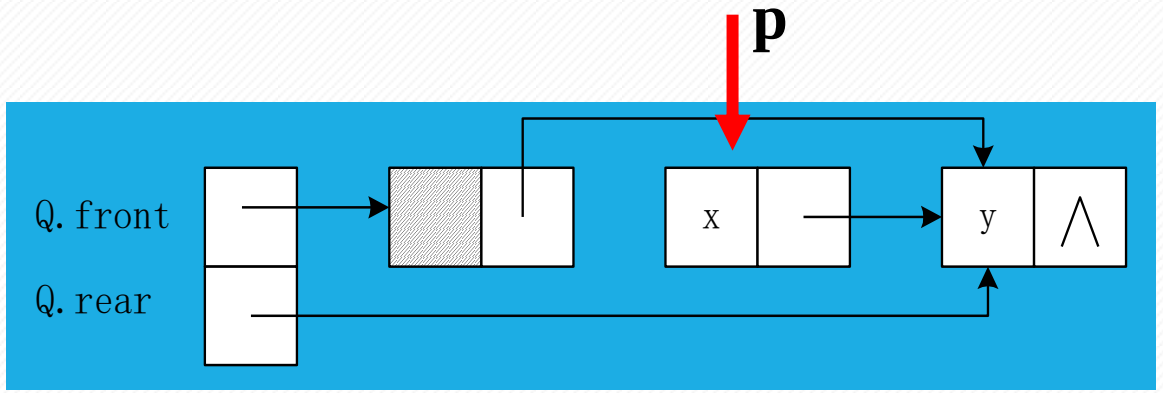
链队列入队


```
Status EnQueue(LinkQueue &Q,QElemType e){
    p=(QueuePtr)malloc(sizeof(QNode));
    if(!p) exit(OVERFLOW);
    p->data=e; p->next=NULL;
    Q.rear->next=p;
    Q.rear=p;
    return OK;
}
```



链队列出队


```
Status DeQueue (LinkQueue &Q,QElemType &e){
    if(Q.front==Q.rear) return ERROR; //队空，无法删除
    p=Q.front->next;
    e=p->data;
    Q.front->next=p->next;
    if(Q.rear==p) Q.rear=Q.front;
    delete p;
    return OK;
}
```



项目实战操作案例2

CimFAX
传真服务器常用
问答

将来的你

冲刺 加油!

让我们全力以赴

一定会感激现在拼命的自己