队列的基本概念

队列的定义

队列（Queue）简称队，也是一种操作受限的线性表，只允许在表的一端进行插入，而在表的另一端进行删除。向队列中插入元素称为入队或进队，删除元素称为出队或离队。这和我们日常生活中的排队是一致的，最早排队的也是最早离队的，其操作的特性是先进先出（First In First Out, FIFO）。

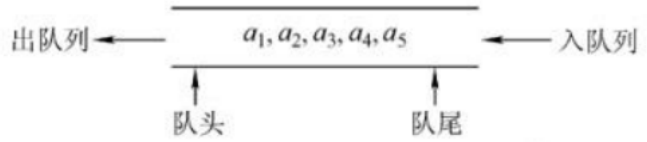

图 3.5　队列示意图

队头（Front）。允许删除的一端，又称队首。

队尾（Rear）。允许插入的一端。

空队列。不含任何元素的空表。

队列常见的基本操作

InitQueue(&Q)	初始化队列，构造一个空队列Q。
QueueEmpty(Q)	判队列空，若队列Q为空返回true，否则返回false。
EnQueue(&Q,x)	入队，若队列Q未满，将x加入，使之成为新的队尾。
DeQueue(&Q,&x)	出队，若队列Q非空，删除队头元素，并用x返回。
GetHead(Q,&x)	读队头元素，若队列Q非空，则将队头元素赋值给x。

需要注意的是，栈和队列是操作受限的线性表，因此不是任何对线性表的操作都可以作为栈和队列的操作。比如，不可以随便读取栈或队列中间的某个数据。

队列的顺序存储结构

队列的顺序存储

队列的顺序实现是指分配一块连续的存储单元存放队列中的元素，并附设两个指针：队头指针front指向队头元素，队尾指针rear指向队尾元素的下一个位置（不同教材对front和rear的定义可能不同，例如，可以让rear指向队尾元素、front指向队头元素。对于不同的定义，出队入队的操作是不同的，本节后面有一些相关的习题，读者可以结合习题思考）。

队列的顺序存储类型可描述为

```
#define MaxSize 50//定义队列中元素的最大个数
typedef struct{
    ElemType data[MaxSize];//用数组存放队列元素
    int front, rear;//队头指针和队尾指针
}SqQueue;
```

初始时：Q.front=Q.rear=0。

进队操作：队不满时，先送值到队尾元素，再将队尾指针加1。

出队操作：队不空时，先取队头元素值，再将队头指针加1。

a所示为队列的初始状态，有Q.front==Q.rear==0成立，该条件可以作为队列判空的条件。但能否用Q.rear==MaxSize作为队列满的条件呢？显然不能，d中，队列中仅有一个元素，但仍满足该条件。这时入队出现“上溢出”，但这种溢出并不是真正的溢出，在data数组中依然存在可以存放元素的空位置，所以是一种“假溢出”。

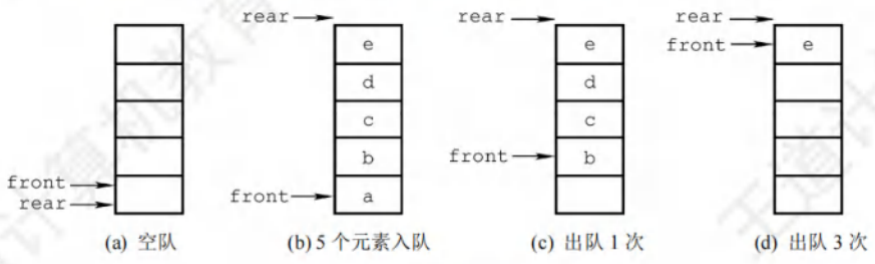

图 3.6　队列的操作

循环队列

上面指出了顺序队列“假溢出”的问题，这里引出循环队列的概念。将顺序队列臆造为一个环状的空间，即把存储队列元素的表从逻辑上视为一个环，称为循环队列。当队首指针Q.front=MaxSize-1后，再前进一个位置就自动到0，这可以利用除法取余运算（%）来实现。

初始时	Q.front=Q.rear=0
队首指针进1	Q.front=(Q.front+1)%MaxSize
队尾指针进1	Q.rear=(Q.rear+1)%MaxSize
队列长度	(Q.rear+MaxSize-Q.front)%MaxSize
出队入队时	指针都按顺时针方向进1

那么，循环队列队空和队满的判断条件是什么呢？显然，队空的条件是Q.front==Q.rear。若入队元素的速度快于出队元素的速度，则队尾指针很快就会赶上队首指针，如d1所示，此时可以看出队满时也有Q.front==Q.rear。循环队列出入队示意图如下图所示。

为了区分队空还是队满的情况，有三种处理方式：

1. 牺牲一个单元来区分队空和队满，入队时少用一个队列单元，这是一种较为普遍的做法，约定以“队头指针在队尾指针的下一位置作为队满的标志”，如d2所示。
2. 类型中增设size数据成员，表示元素个数。删除成功size减1，插入成功size加1。队空时Q.size==0；队满时Q.size==MaxSize，两种情况都有Q.front==Q.rear。

3. 类型中增设tag数据成员，以区分是队满还是队空。删除成功置tag=0，若导致Q.front==Q.rear，则为队空；插入成功置tag=1，若导致Q.front==Q.rear，则为队满。

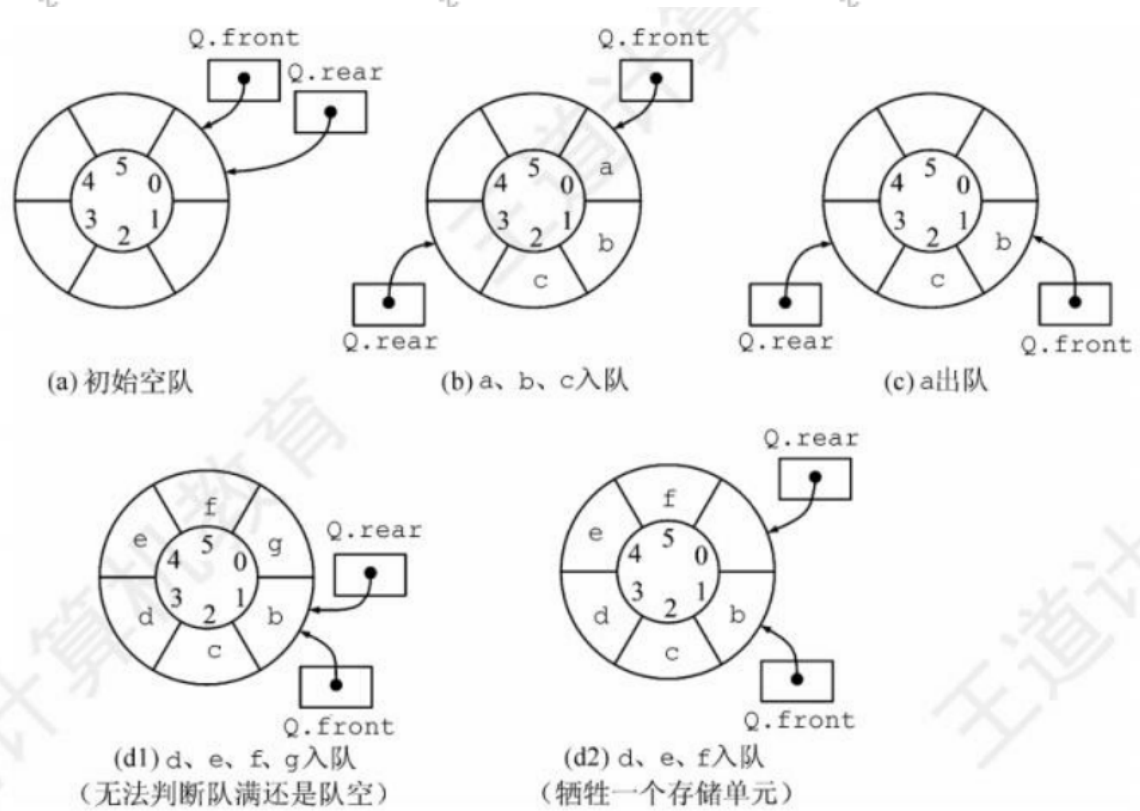

图 3.7　循环队列出入队示意图

循环队列的操作

1. 初始化

```
void InitQueue(SqQueue &Q){
    Q.rear=Q.front=0;//初始化队首、队尾指针
}
```

2. 判队空

```
bool isEmpty(SqQueue Q){
    if(Q.rear==Q.front)//队空条件
        return true;
    else
        return false;
}
```

3. 入队

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

4. 出队

```
bool DeQueue(SqQueue &Q, ElemType &x){
    if(Q.rear==Q.front)//队空则报错
        return false;
    x=Q.data[Q.front];
    Q.front=(Q.front+1)%MaxSize;//队头指针加1取模
    return true;
}
```

队列的链式存储结构

队列的链式存储

队列的链式表示称为链队列，它实际上是一个同时有队头指针和队尾指针的单链表，如下图所示。头指针指向头结点，尾指针指向尾结点，即单链表的最后一个结点。

图 3.8　不带头结点的链式队列

队列的链式存储类型可描述为

```
typedef struct LinkNode{//链式队列结点
    ElemType data;
    struct LinkNode *next;
}LinkNode;
typedef struct{//链式队列
    LinkNode *front,*rear;//队列的队头和队尾指针
}LinkQueue;
```

不带头结点时，当Q.front==NULL或Q.rear==NULL时，链式队列为空。

入队时，建立个新结点，将新结点插入到链表的尾部，并让Q.rear指向这个新插入的结点（若原队列为空队，则令Q.front也指向该结点）。出队时，首先判断队是否为空，若不空，则取出队头元素，将其从链表中摘除，并让Q.front指向下一个结点（若该结点为最后一个结点，则置Q.front和Q.rear都为NULL）。

不难看出，不带头结点的链式队列在操作上往往比较麻烦，因此通常将链式队列设计成一个带头结点的单链表，这样插入和删除操作就统一了。

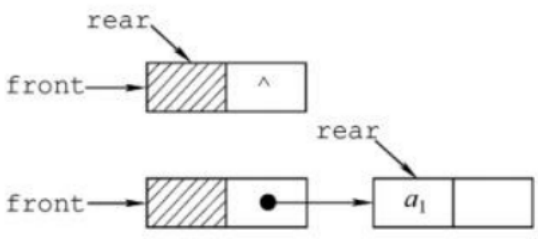

图 3.9　带头结点的链式队列

用单链表表示的链式队列特别适合于数据元素变动比较大的情形，而且不存在队列满且产生溢出的问题。另外，假如程序中要使用多个队列，与多个栈的情形一样，最好使用链式队列，这样就不会出现存储分配不合理和“溢出”的问题。

链式队列的基本操作

1. 初始化

```
void InitQueue(LinkQueue &Q){
    Q.front=Q.rear=(LinkNode*)malloc(sizeof(LinkNode));//建立头结点
    Q.front->next=NULL;//初始为空
}
```

2. 判队空

```
bool IsEmpty(LinkQueue Q){
    if(Q.front==Q.rear)//判空条件
        return true;
    else
        return false;
}
```

3. 入队

```
void EnQueue(LinkQueue &Q, ElemType x){
    LinkNode *s=(LinkNode *)malloc(sizeof(LinkNode));//创建新结点
    s->data=x;
    s->next=NULL;
    Q.rear->next=s;//插入链尾
    Q.rear=s;//修改尾指针
}
```

4. 出队

```
bool DeQueue(LinkQueue &Q, ElemType &x){
    if(Q.front==Q.rear)
        return false;//空队
    LinkNode *p=Q.front->next;
    x=p->data;
    Q.front->next=p->next;
    if(Q.rear==p)
        Q.rear=Q.front;//若原队列中只有一个结点，删除后变空
    free(p);
    return true;
}
```

双端队列

双端队列是指允许两端都可以进行插入和删除操作的线性表，如下图所示。双端队列两端的地位是平等的，为了方便理解，将左端也视为前端，右端也视为后端。

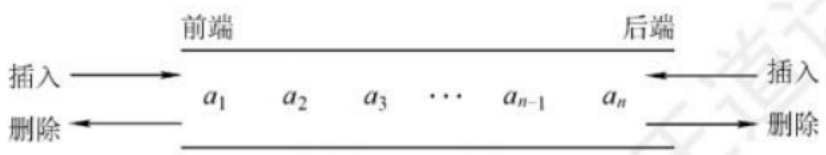

图 3.10　双端队列

在双端队列进队时，前端进的元素排列在队列中后端进的元素的前面，后端进的元素排列在队列中前端进的元素的后面。在双端队列出队时，无论是前端还是后端出队，先出的元素排列在后出的元素的前面。思考：如何由入队序列a,b,c,d得到出队序列d,c,a,b?

输出受限的双端队列：允许在一端进行插入和删除，但在另一端只允许插入的双端队列称为输出受限的双端队列称为输出受限的双端队列。

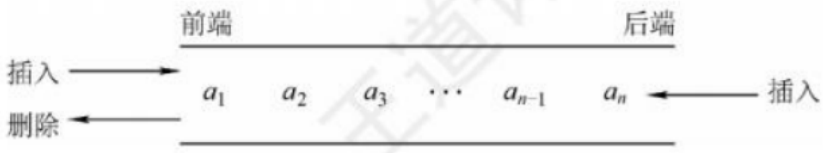

图 3.11　输出受限的双端队列

输出受限的双端队列：允许在一端进行插入和删除，但在另一端只允许删除的双端队列称为输入受限的双端队列。若限定双端队列从某个端点插入的元素只能从该端点删除，则该双端队列就蜕变为两个栈底相邻接的栈。

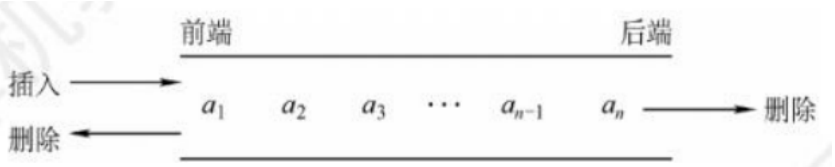

图 3.12　输入受限的双端队列

设有一个双端队列，输入序列为$1, 2, 3, 4$，试分别求出以下条件的输出序列。

(1)能由输入受限的双端队列得到，但不能由输出受限的双端队列得到的输出序列。

(2)能由输出受限的双端队列得到，但不能由输入受限的双端队列得到的输出序列。

(3)既不能由输入受限的双端队列得到，又不能由输出受限的双端队列得到的输出序列。

先看输入受限的双端队列，如下图所示。假设end1端输入1，2，3，4，则end2端的输出相当于队列的输出，即1，2，3，4；而end1端的输出相当于栈的输出，n=4时仅通过end1端有14种输出序列（由Catalan公式得出），仅通过end1端不能得到的输出序列有4!-14=10种：

图 3.13　输入受限的双端队列

1, 4, 2, 3	2, 4, 1, 3	3, 4, 1, 2	3, 1, 4, 2	3, 1, 2, 4
4, 3, 1, 2	4, 1, 3, 2	4, 2, 3, 1	4, 2, 1, 3	4, 1, 2, 3

通过end1和end2端混合输出，可以输出这10种中的8种，参看下表。其中，SL，XL分别代表end1端的进队和出队，XR代表end2端的出队。

输 出 序 列	进队出队顺序	输 出 序 列	进队出队顺序
1, 4, 2, 3	$S_LX_RS_LS_LS_LX_LX_RX_R$	3, 1, 2, 4	$S_LS_LS_LX_LS_LX_RX_RX_R$
2, 4, 1, 3	$S_LS_LX_LS_LS_LX_LX_RX_R$	4, 1, 2, 3	$S_LS_LS_LS_LX_LX_RX_RX_R$
3, 4, 1, 2	$S_LS_LS_LX_LS_LX_LX_RX_R$	4, 1, 3, 2	$S_LS_LS_LS_LX_LX_RX_LX_R$
3, 1, 4, 2	$S_LS_LS_LX_LX_RS_LX_LX_R$	4, 3, 1, 2	$S_LS_LS_LS_LX_LX_LX_RX_R$

剩下两种是不能通过输入受限的双端队列输出的，即4，2，3，1和4，2，1，3。

再看输出受限的双端队列，如下图所示。假设end1端和end2端都能输入，仅end2端可以输出。若都从end2端输入，就是一个栈了。当输入序列为1，2，3，4时，输出序列有14种。对于其他10种不能得到的输出序列，交替从end1和end2端输入，还可以输出其中8种。设SL代表end1端的输入，SR、XR分别代表end2端的输入和输出，则可能的输出序列见下表。

图 3.14　输出受限的双端队列

输 出 序 列	进队出队顺序	输 出 序 列	进队出队顺序
1, 4, 2, 3	$S_LX_RS_LS_LS_RX_RX_RX_R$	3, 1, 2, 4	$S_LS_LS_RX_RX_RS_LX_RX_R$
2, 4, 1, 3	$S_LS_RX_RS_LS_RX_RX_RX_R$	4, 1, 2, 3	$S_LS_LS_LS_RX_RX_RX_RX_R$
3, 4, 1, 2	$S_LS_LS_RX_RS_RX_RX_RX_R$	4, 2, 1, 3	$S_LS_RS_LS_RX_RX_RX_RX_R$
3, 1, 4, 2	$S_LS_LS_RX_RX_RS_RX_RX_R$	4, 3, 1, 2	$S_LS_LS_RS_RX_RX_RX_RX_R$

通过输出受限的双端队列不能得到的两种输出序列是4，1，3，2和4，2，3，1。

综上所述：

1. 能由输入受限的双端队列得到，但不能由输出受限的双端队列得到的是4，1，3，2。
2. 能由输出受限的双端队列得到，但不能由输入受限的双端队列得到的是4，2，1，3。
3. 既不能由输入受限的双端队列得到，又不能由输出受限的双端队列得到的是4，2，3，1。

提示：实际双端队列的考题不会这么复杂，通常仅判断序列是否满足题设条件，代入验证即可。