3.5 队列

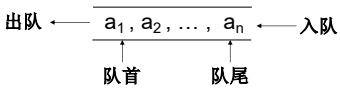
主要内容

- 队列的概念和基本操作
- 物理结构的设计
- 应用实例

3.5.1 定义

队列(Queue)：也是运算受限的线性表。是一种先进先出(First In First Out，简称FIFO)的线性表。只允许在表的一端进行插入，而在另一端进行删除。

- 队首(front)：允许进行删除的一端称为队首。
- 队尾(rear)：允许进行插入的一端称为队尾。



应用举例

- 队列在日常生活中和计算机程序设计中，有着非常重要的作用，在此，仅举出两个方面例子来说明它，其它应用在后面的学习中将会遇到。
- 第一个例子就是CPU资源的竞争问题。在具有多个终端的计算机系统中，有多个用户需要使用CPU各自运行自己的程序，它们分别通过各自终端向操作系统提出使用CPU的请求。
- 操作系统按照每个请求在时间上的先后顺序，将其排成一个队列，每次把CPU分配给队头用户使用，当相应的程序运行结束后，再把CPU分配给新的队头用户，直到所有用户任务处理完毕。

基本操作

- 初始化：InitQue
- 判断是否为空：EmptyQue
- 入队列：InsertQue
- 出队列：DeleteQue
- 取队列头：HeadQue

ADT of queue

ADT Queue{

D ={ a_i | a_i∈ElemType, i=1, 2, ..., n, n >= 0 }

R = {<a_{i-1}, a_i> | a_{i-1}, a_i∈D, i=2,3,...,n }

约定a_1端为队首，a_n端为队尾。

基本操作：

InitQue()：创建一个空队列；

EmptyQue()：若队列为空，则返回true，否则返回flase；

……

InsertQue(x)：向队尾插入元素x；

DeleteQue(x)：删除队首元素x；

HeadQue(x):读取队头元素

} ADT Queue

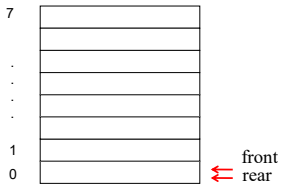
3.5.2 队列的连续设计存储表示和实现

利用一组连续的存储单元(一维数组)依次存放从队首到队尾的各个元素，称为顺序队列。

```
typedef struct queue
   { ElemType  Queue_array[MAX_QUEUE_SIZE];
      int  front;
      int  rear;
   }SqQueue;
```


设立一个队首指针front，一个队尾指针rear，非空队列中，队头指针总是指向队头元素的前一个位置，队尾指针指向队尾元素。

存储空间0..m-1
初始：front=0;rear=0。
入队：rear=rear+1
出队：front=front+1
队空：rear=front
队满：rear-front+1=m

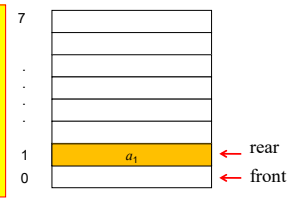


MAX_QUEUE_SIZE 8

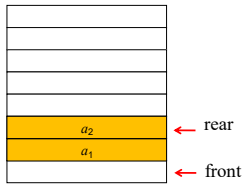
初始时是空队

入队操作：加入元素a_1

存储空间0..m-1
初始：front=0;rear=0。
入队：rear=rear+1
出队：front=front+1
队空：rear=front
队满：rear-front=m

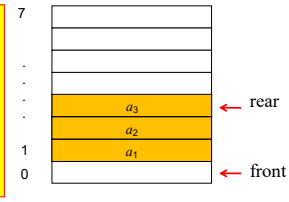


存储空间0..m-1
初始：front=0;rear=0。
入队：rear=rear+1
出队：front=front+1
队空：rear=front
队满：rear-front=m



做入队操作

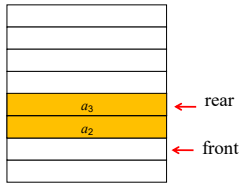
存储空间0..m-1
初始：front=0;rear=0。
入队：rear=rear+1
出队：front=front+1
队空：rear=front
队满：rear-front=m



做入队操作

做出队操作

存储空间0..m-1
初始：front=0;rear=0。
入队：rear=rear+1
出队：front=front+1
队空：rear=front
队满：rear-front=m



存储空间0..m-1

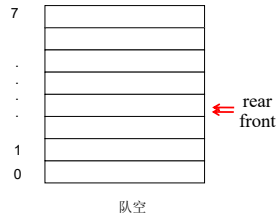
初始：front=0;rear=0。

入队：rear=rear+1

出队：front=front+1

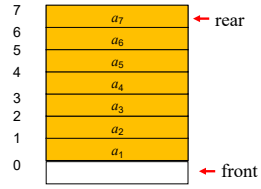
队空：rear=front

队满：rear-front=m



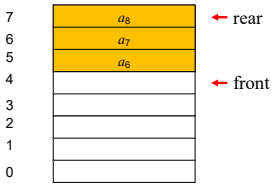
队空

队列溢出现象（1）



全部空间都已经使用，不能再加入元素，真正溢出。

队列溢出现象（2）



由于队尾指针已经指向最后位置，不能再加入元素。
但是前面还有空余，假溢出。

顺序队列中存在“假溢出”现象。因为在入队和出队操作中，头、尾指针只增加不减小，致使被删除元素的空间永远无法重新利用。因此，尽管队列中实际元素个数可能远远小于数组大小，但可能由于尾指针已经超出数组的上界而不能做入队操作。该现象称为假溢出。

3.5.3 循环队列

为充分利用空间，克服上述“假溢出”现象的方法是：将为队列分配的地址空间看成为一个首尾相接的圆环，并称这种队列为循环队列(Circular Queue)。

在循环队列中进行出队、入队操作时，队首、队尾指针仍要加1，朝前移动。当队首、队尾指针指向数组上界(MAX_QUEUE_SIZE-1)时，其加1操作的结果需要指向数组的下界0。

这种循环意义下的加1操作可以描述为：

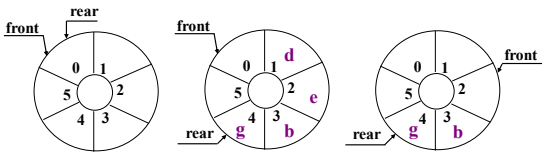
```
if (i+1==MAX_QUEUE_SIZE) i=0;
   else    i++;
```

其中：i代表队首指针(front)或队尾指针(rear)

也可以写成：

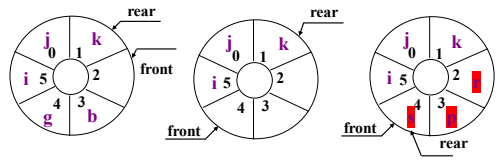
```
i=(i+1)%MAX_QUEUE_SIZE
```

例：设有循环队列QU[6]，其初始状态是front=rear=0，各种操作后队列的头、尾指针的状态变化情况如下图所示。

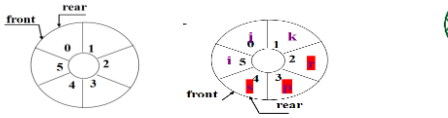


(a) 空队列　(b) d, e, b, g入队　(c) d, e出队

例：设有循环队列QU[6]，其初始状态是front=rear=0，各种操作后队列的头、尾指针的状态变化情况如下图所示。

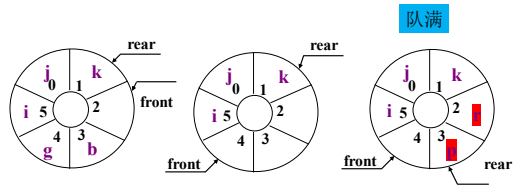


(d) i, j, k入队　(e) b, g出队　(f) r, p, s入队



(a) 空队列　(f) r, p, s入队

- 方法一：增设一个存储队列中元素个数的计数器count，当front==rear 且count==0时，队空；当front==rear 且count==MaxSize时，队满；
- 方法二：设置标志flag，当front==rear且flag==0时为队空；当front==rear且flag==1时为队满。
- 方法三：保留队空的判定条件：front==rear；把队满判定条件修改为：((rear+1)% MAX_QUEUE_SIZE ==front)
  - 代价：浪费一个元素空间，队满时数组中有一个空闲单元；



(d) i, j, k入队　(e) b, g出队　(f) r, p, s入队

循环队列的初始化

```
SqQueue Init_CirQueue(void)
{ SqQueue  Q ;
  Q.front=Q.rear=0;  return(Q) ;
}
```

入队操作

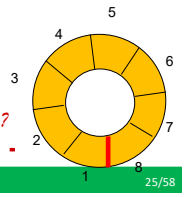
```
Status Insert_CirQueue(SqQueue  Q , ElemType  e)
 /*  将数据元素e插入到循环队列Q的队尾  */
  { if ((Q.rear+1)%MAX_QUEUE_SIZE== Q.front)
      return  ERROR;    /* 队满，返回错误标志  */
      Q.rear=(Q.rear+1)% MAX_QUEUE_SIZE ;
            /* 队尾指针向前移动 */
      Q.Queue_array[Q.rear]=e ;  /*  元素e入队  */
      return OK;      /*  入队成功  */
  }
```


出队操作

```
Status Delete_CirQueue(SqQueue  Q, ElemType  *x )
   /*  将循环队列Q的队首元素出队  */
  {  if  (Q.front+1== Q.rear)
       return ERROR ;     /* 队空，返回错误标志  */
     Q.front=(Q.front+1)% MAX_QUEUE_SIZE ;
       /*  队首指针向前移动 */
    *x=Q.Queue_array[Q.front] ;  /* 取队首元素 */
     return OK ;
  }
```

循环队列 MAX_QUEUE_SIZE =m

入队：rear=(rear+1)%*m*
出队：front=(front+1)%*m*
队空：rear=front
队满：(rear+1)%*m*=front



存储空间为1..m，上述的操作又如何？

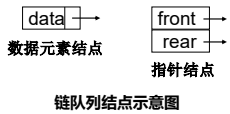
3.5.4 队列的链式表示和实现

队列的链式存储结构简称为链队列，它是限制仅在表头进行删除操作和表尾进行插入操作的单链表。

需要两类不同的结点：数据元素结点，队列的队首指针和队尾指针的结点，如图所示。

数据元素结点类型定义：

```
typedef struct Qnode
  { ElemType   data ;
    struct Qnode  *next ;
  }QNode ;
```



数据元素结点　　指针结点

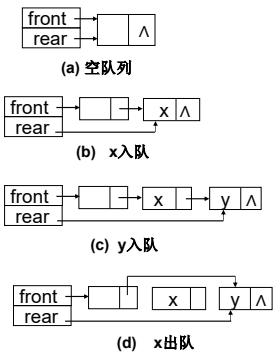
链队列结点示意图

指针结点类型定义：

```
typedef struct link_queue
  {  QNode  *front , *rear ;
  }Link_Queue ;
```

链队运算及指针变化

链队的操作实际上是单链表的操作，只不过是删除在表头进行，插入在表尾进行。插入、删除时分别修改不同的指针。



队列操作及指针变化

链队列的基本操作

(1) 链队列的初始化

```
LinkQueue *Init_LinkQueue(void)
  { LinkQueue  *Q ;  QNode  *p ;
    p=(QNode *)malloc(sizeof(QNode)) ; /* 开辟头结点 */
    p->next=NULL ;
    Q=(LinkQueue  *)malloc(sizeof(LinkQueue)) ;
        /*  开辟链队的指针结点  */
    Q.front=Q.rear=p ;
    return(Q) ;
  }
```


链队列的入队操作

在已知队列的队尾插入一个元素e，即修改队尾指针(Q.rear)。

```
Status  Insert_CirQueue(LinkQueue  *Q , ElemType  e)
      /*  将数据元素e插入到链队列Q的队尾  */
  {  p=(QNode *)malloc(sizeof(QNode)) ;
     if (!p)  return  ERROR;
        /*  申请新结点失败，返回错误标志 */
     p->data=e ; p->next=NULL ;      /*  形成新结点 */
     Q.rear->next=p ;  Q.rear=p ;  /*  新结点插入到队尾  */
     return OK;
  }
```

链队列的出队操作

```
Status  Delete_LinkQueue(LinkQueue  *Q, ElemType *x)
  {  QNode *p ;
     if  (Q.front==Q.rear)  return ERROR ;   /*  队空  */
     p=Q.front->next ;  /*  取队首结点  */
     *x=p->data ;
     Q.front->next=p->next ;     /*  修改队首指针  */
     if  (p==Q.rear)  Q.rear=Q.front ;
          /*  当队列只有一个结点时应防止丢失队尾指针  */
       free(p) ;
     return OK ;
  }
```


(4) 链队列的撤消

```
void  Destroy_LinkQueue(LinkQueue  *Q )
 /*  将链队列Q的队首元素出队  */
 {  while  (Q.front!=NULL)
      {  Q.rear=Q.front->next;
           /*  令尾指针指向队列的第一个结点  */
         free(Q.front);     /*  每次释放一个结点  */
           /*  第一次是头结点，以后是元素结点  */
         Q.ront=Q.rear;
      }
 }
```


3.6 离散事件的模拟

银行有四个窗口对外服务，从开门起不断有客户进入银行。每个窗口在某一时刻只能接待一位客户，因此在客户人数众多时，需要在每个窗口顺序排队。对于刚进入的客户，如果某个窗口正在空闲，则可上前办理业务，否则，排在人数最少的队列后面等待。

编制程序，模拟银行的业务活动，并计算客户的平均逗留时间。

- 离散事件模拟
- 问题：
  - 一个银行，有N个窗口；
  - 每分钟来一个客户，客户业务处理时间为一个随机数M；
  - 每个客户总是排到最短的队上。
- 要求：
  - 模拟一段时间内的排队情况，并进行定量统计（平均逗留时间）

若只有一个队列时，平均逗留时间又是多少？

分　析

事件：客户到达银行和离开银行时发生的事情。

　　事件的类型，事件发生的时刻。

事件的发生：

　　到达事件：客户的到来时形成（0）。

　　离开事件：由客户服务时间和等待时间决定。(1..4)

建立事件链表，记录模拟过程中发生的事件。按照事件发生的时刻的先后次序存储。

分　析

设立四个队列，存储客户到达的时刻和服务所需要的时间。队头元素为窗口正在服务的客户。每个队头客户都存在一个将要离开的事件。队列的结构如下：

到达的时刻	需要服务的时间
ArrivalTime	Duration

任意时刻发生的事件，事件结点结构如下：

新客户的到来
1号窗口客户离开
2号窗口客户离开
3号窗口客户离开
4号窗口客户离开

到达0
离开1,2,3,4

事件发生的时刻	事件类型

37/58

ev：事件链表；记录将要发生的事件（到达/离开）
如果是到达的类型，则找个队进行排队等待，
如果是离开的事件，则删除对应队列中的元素。
仿真器总是从事件链表中获得事件进行处理。
结构：事件发生时间(OccurTime)，事件类型(Ntype)。

en: 事件结点；记录要处理的事件信息，
结构：事件发生时间(OccurTime)，事件类型(Ntype) 。

q[i]： 队列； 客户排队等待
结构：到达时间(ArrivalTime)，服务时间(Duration)。

38/58

```
Void BankSimulation()
 {
   OpenForday();
   While ev非空/事件链表中有待处理的事件
    {
      DelList(ev, en);/获得要处理的事件
      if (en.NType==0) /处理事件
        CustomerArrived();
      else CustomerDeparture();
    }
   计算平均逗留时间；
 }
```

39/58

```
Void OpenForday()
{
  Totaltime=0;
   CustomerNum=0;
   InitList(ev);    //事件链表置空
   for (i=1;i<=4;i++)
     InitQue(q[i]);  //队列置空
   en.OccurTime=0;
   en.NType=0;          //设定第一个事件（客户到达事件）；
   InsertList(ev,en)   //插入事件表中
 }
```

40/58

```
Void CustomerArrived()  //客户到达处理模块，同时生成下一个事件的到达时刻
{ /* en.NType=0，  （处理当前事件，产生新的到达事件）
  CustomerNum++;
  Random(Durtime, intertime)  //（需要服务时间, 下一到达时刻间隔 ）
  t=en.OccurTime+intertime;    //下一个客户到达的时刻；
  if (t<Closetime)
       InsertList(ev, (t,0));      //插入到事件链表，
                                   表示将在t时刻有新客户到达
   i=Minimum(q);    //为当前事件查找最短的队列
   InsertQue(q[i], (en.OccurTime, Durtime));  // 入队
   if (QueueLength(q[i])==1)
     InsertList(ev, (en.Occurtime+Durtime,i));
     //产生离开事件，插入到事件链表
 }
```

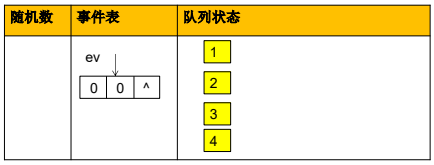
41/58

```
Void CustomerDeparture()  //客户离开事件的处理模块
 {
   i=en.NType;
   DeleteQue(q[i], Customer); //删除第i队头元素，
                              值存储在Customer中
   TotalTime=TotalTime+en.OccurTime-Customer.ArrivalTime;
   if (! EmptyQue(q[i])
    //将第i个队列的队头元素作为离开事件插入到事件表
     {
       HeadQue(q[i],Customer);
       InsertList(ev, (en.OccurTime+Customer.Durtation, i));
 }
```

42/58

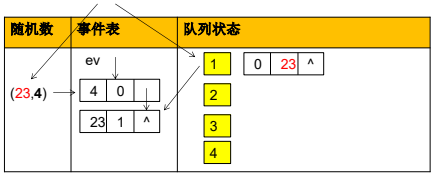
仿真示例

- 初始状态



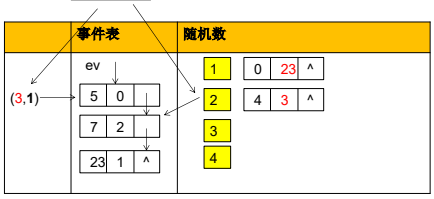
仿真示例

- 事件 0 0



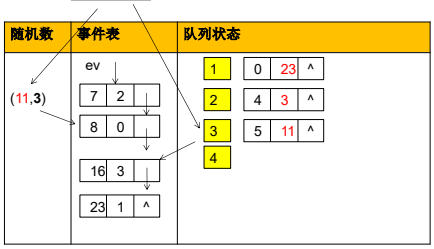
仿真示例

- 事件 4 0



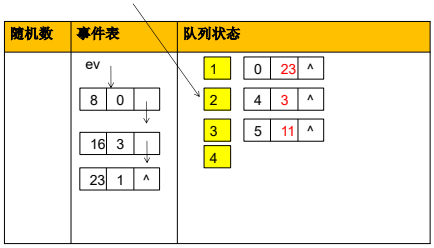
仿真示例

- 事件 5 0



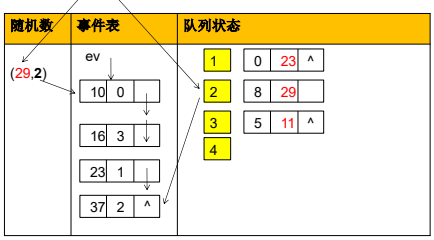
仿真示例

- 事件 7 2

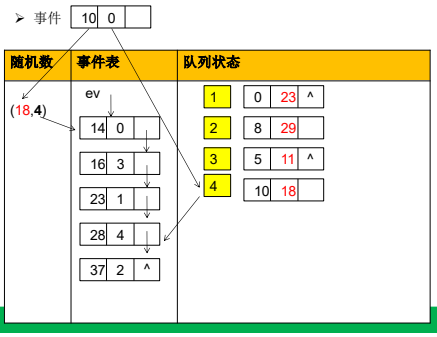


仿真示例

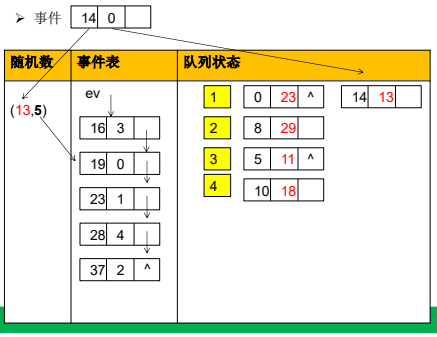
- 事件 8 0



仿真示例



仿真示例



3.7 其它操作受限的线性表

输入受限的队列：限定在一端进行输入，可以在两端进行删除的队列。

输出受限的队列：限定在一端进行输出，可以在两端进行加入的队列。

示　例（1）

若以1234作为双端队列的输入序列，分别求出满足下列条件的输出序列：

（1）能由输入受限的双端队列得到的输出序列；

在end2输入，从end1的输出相当于队的输出，即1234。
在end2输入，从end2的输出相当于栈的输出，当n=4时，仅经过end2的输出有14种。仅通过end2端不能得到的输出序列有4！-14=10种。它们是：

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

若以1234作为双端队列的输入序列，分别求出满足下列条件的输出序列：

（1）能由输入受限的双端队列得到的输出序列；

通过end1和end2对以下10种序列进行混合输出，还可以输出8种，

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

不能得到的输出序列： 4213 和 4231，其余的都是可以得到的输出序列。

示　例（2）

若以1234作为双端队列的输入序列，分别求出满足下列条件的输出序列：

（2）能由输出受限的双端队列得到的输出序列；

从end1和end2都能输入。若从end1输入，就是一个栈。能够得到的输出序列有14种。不能得到的输出序列如下：

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

若以1234作为双端队列的输入序列，分别求出满足下列条件的输出序列：

（2）能由输出受限的双端队列得到的输出序列；

对这10种输出序列，交替从end1和end2输入，还可以输出其中的8种。

则不能通过输出受限的双端队列得到的输出序列：4231，4132

常规题目

若以1234作为双端队列的输入序列，分别求出满足下列条件的输出序列：

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

优 先 队 列（堆排序）

- 普通的队列是一种先进先出的数据结构，元素在队列尾追加，从队列头删除。在优先队列中，每个元素都有一个优先级。具有最高优先级的元素最先出队。优先队列具有最高进先出（largest-in，first-out）的行为特征。
- 对优先级队列执行的操作入队（插入）、出队（删除队头元素）。
- 例子：
  - 操作系统的任务调度
  - 多媒体通讯网络中的数据包调度
  - 集合中的元素搜索

优 先 队 列 的 存 储 设 计