队列（Queue）

优先级队列

- 定义
  - 带有优先级的队列

双端队列

- 定义
  - 所有的插入和删除操作都在线性表的两端进行，可以看成是两个栈的一起的两个栈
  - 与两栈共享存储空间不同的是，两个栈的栈顶指针是望两端延伸的
- 图示

定义

- 与堆栈一样，队列也是一种特殊的线性表，这种表在一端进行插入操作，而在另一端进行删除操作
  - 允许插入的一端叫队尾(rear)，在下面的上方
  - 允许删除的一端叫队首(front)，在下面的下方
  - 位于队首和队尾的元素分别称为队首元素和队尾元素
  - 当表中无元素时，称为空队
- 图示

- 注意事项
  - 队列的插入称为进队，队列的删除称为出队
  - 数据元素从**rear（队尾，上方）进队，从front（队首，下方）**出队
  - 先进入队列的元素比后进入队列的元素先出队列（后进入队列的元素比先进入的元素后出队列），即队列是一个先进先出的FIFO表
  - 队列空条件为：front == rear,而front=rear=0是队列初始状态（未书初定）
  - 进队时，先把元素插入到rear所指示的位置，然后rear++(保证rear始终指向队尾元素的后一位)
  - 出队时，先取出front所指示的队首元素，然后front++(保证front始终指向真正的队首元素)
  - 如果存储队列长度为maxsize，则rear >= maxsize时，队满；当rear == front时，队空
- 抽象数据类型ADT
  - ADT Queue(
    Data：数据元素相同数据类型，相邻元素具有前驱和后继的关系。
    Operation：
    InitQueue(&Q，maxsize，incresize)
    操作结果：构造一个容量为maxsize的空队列Q。
    ClearQueue(&Q)
    初始条件：队列Q已存在。
    操作结果：将Q清为空队列。
    QueueLength(Q)
    初始条件：队列Q已存在。
    操作结果：返回Q的元素个数，即队列的长度。
    EnQueue(&Q，e)
    初始条件：队列Q已存在。
    操作结果：插入元素e为Q的新的队尾元素。
    DeQueue(&Q，&e)
    初始条件：Q为非空队列。
    操作结果：删除Q的队首元素，并用e返回其值。
    GetHead(Q，&e)
    初始条件：Q为非空队列。
    操作结果：用e返回Q的队头元素。
    QueueTraverse(Q)
    初始条件：队列Q已存在且非空。
    操作结果：从队头到队尾依次输出Q中各个数据元素。
    QueueEmpty(Q)
    初始条件：队列Q已存在。
    操作结果：若Q为空队列，则返回true；否则返回false。
    DestroyQueue(&Q)
    初始条件：队列Q已存在。
    操作结果：队列Q被删除，不再存在。
)ADT Queue

顺序队列

- 定义
  - 顺序队列就是队列以顺序存储结构存储
  - 顺序队列包括顺序非循环队列和顺序循环队列
  - 顺序队列与顺序线一样，都是特殊的顺序表，只是顺序线的插入操作在队尾进行，删除操作在队首进行
- 注意事项
  - 顺序队列中，front和rear的值其实都是数组下标（假指针）
  - 顺序队列的判空条件是：rear == front，当其都为0时，队列为初始状态

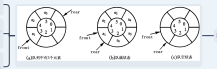
假溢出

- 定义
  - 当元素被插入到数组中下标最大的位置上时，数组的末端还有空间空间，此时如果还有元素入队，就会发生“溢出”现象，虽然这种溢出并不是真正的溢出，而是**“假溢出”**。
  - 相当于：“上面满了，下面还有空的”。
  - 假溢出的判断条件是：rear >= maxsize && front > 0。
  - 假溢出是队满时候的一种情况，另一种是真溢出。
  - 假溢出只存在于非循环顺序队列中。

- 解决方法
  - 1. 修改出队算法（事先避免假溢出），使每次出队列后都把队列中剩余的元素向front方向移动一个位置。O(n)
  - 2. 修改进队算法（假溢出发生时的解决方法），使得当真溢出时返回true；当假溢出时，把队列中的所有元素向front方向移动一个位置。O(n)
  - 3. 顺序循环队列。

顺序循环队列

- 定义
  - 顺序循环队列就是把顺序队列改造成一个头尾相连的循环表
  - 顺序循环队列初始状态为front = rear = 0
  - 入队时，把元素插到rear指示位置，然后rear++
  - 出队时，把front指示位置元素删除，然后front++
  - 顺序循环队列可以解决假溢出问题，因为如果maxsize-1个位置被占用后，只要还有空间，就可以将新的元素加入下标为0的位置。（如图3.13）
  - 判断下次进入的位置：例如若maxsize = 6，rear = 5，则下次进队的位置为0，因为(rear+1)% 6 = 0
- 状态图示
  - 第一种状态：队列中有元素（既非空，也非满）
    rear != front
  - 第二种状态：队满
    (rear == front) && front与rear所指示的位置有元素
  - 第三种状态：队空
    (rear == front) && front与rear所指示的位置无元素



- 注意事项
  - 顺序循环队列可以解决“假溢出”问题，但是会带来“无法判空”的问题。
  - 顺序队列判空条件是rear == front，而满足这个条件的顺序循环队列有可能为空，有可能为满。
  - ❓ 如何解决“无法判空”问题？
    - 解决方案1：少用一个存储单元(源代码所用方案)
      即修改判满操作，使得进行插入操作时，首先判断rear所指示的下一个位置是否是front，如果是，则停止插入。（队尾留一个单元）
      判满条件：(rear+1)%maxsize = front
      判空条件：rear == front

    - 解决方案2：设置一个标志位
      即设置一个标志位tag=0,当进队成功时tag=1；当出队成功时tag=0
      判满条件：(rear==front) && (tag==1)
      判空条件：(rear==front) && (tag==0)
    - 解决方案3：设置计数器 即设置一个计数器count=0，当进队成功时count++，当出队成功时count--
      判满条件：(rear==front) && count>0** 判空条件：count == 0
- 基本操作
  - 顺序循环队列判队空

```
[In SqQueue.h]
bool QueueEmpty_Sq(SqQueue Q)
{    // 判空函数True，队满函数False
    return Q.rear==Q.front;
}// QueueEmpty_Sq
```

  - 顺序循环队列销毁队列

```
[In SqQueue.h]
void DestroyQueue_Sq(SqQueue &Q)
{
    free(Q.queue);
    Q.queue=NULL;
    Q.queueSize=0;
    Q.front=Q.rear=0;
}// DestroyQueue_Sq
```

  - 顺序循环队列的数据结构

```
[In SqQueue.h]
typedef *queue;                 // 指针数组(数组一维数组)
typedef struct {
    int *queue;                 // 指针数组(数组一维数组)
    int front;                  // 队首指针，指向队首第一个元素
    int rear;                   // 队尾指针，指向队尾第一个元素
    int queueSize;              // 数组实际容量大小
    int increasesize;           // 数组增量
}SqQueue;
```

  - 顺序循环队列的初始化

```
[In SqQueue.h]
void InitQueue_Sq(SqQueue &Q,
                  int queueSize=DEF_INIT_SIZE,
                  int incresize=DEFINCREMENT )
{
    Q.queue=(ElemType *)malloc(queueSize*sizeof(ElemType));
    if(!Q.queue)  exit(0);
    Q.front=Q.rear=0;
    Q.queueSize=queueSize;
    Q.incresize=incresize;
}// InitQueue_Sq
```

  - 求顺序循环队列长度

```
[In SqQueue.h]
int QueueLength_Sq(SqQueue Q)
{
    return (Q.rear-Q.front+Q.queueSize) % Q.queueSize;
}// QueueLength_Sq
```

    元素个数N = (rear-front+maxsize) % maxsize
  - 顺序循环队列进队

```
[In SqQueue.h]
bool EnQueue_Sq(SqQueue &Q,ElemType &e)
{
    if(Q.front==Q.rear)  return false;   // 队满
    e=Q.queue[Q.front];                  // 取出队首元素
    Q.front=(Q.front+1) %Q.queueSize;    // 队首指针顺时针移动一个位置
    return true;
}// EnQueue_Sq
```

  - 顺序循环队列出队

```
[In SqQueue.h]
bool DeQueue_Sq(SqQueue &Q, ElemType &e)
{
    if(Q.front==Q.rear)  return false;   // 队空
    e=Q.queue[Q.front];
    Q.front=(Q.front+1) %Q.queueSize;    // 队首指针顺时针移动一个位置
    return true;
}// DeQueue_Sq
```

  - 顺序循环队列取队首元素

图例
- Question

XMind:ZEN 试用模式

XMind:ZEN | 试用模式