守序前行有道，公德心中常存

第08讲 队列

信息学院（智能应用研究院）
欧 新 宇

建议课时：2

第3章 栈和队列

考核要点

- **考核大纲**

  栈和队列的基本概念、栈和队列的顺序存储和链式存储、 栈和队列的应用

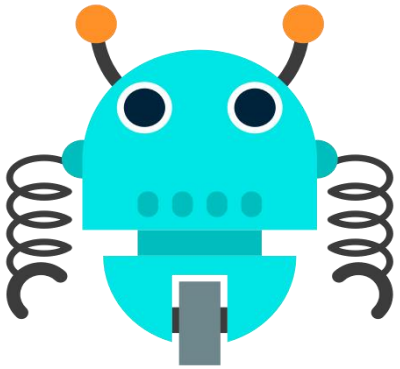
- **复习要点**

  - 掌握栈的基本操作，特别是出入站的过程、出栈序列的合法性
  - 熟练掌握栈的顺序、链式存储的基本操作实现算法，特别是栈满和栈空的条件
  - 熟练掌握标准队列、双端队列、循环队列和链队列的概念、基本操作的实现算法，特别注意队满和队空的条件
  - 掌握栈和队列的特点，并能在相应的应用问题中正确选用
  - 重点掌握算法的思想！能够使用类C或类C++语言实现

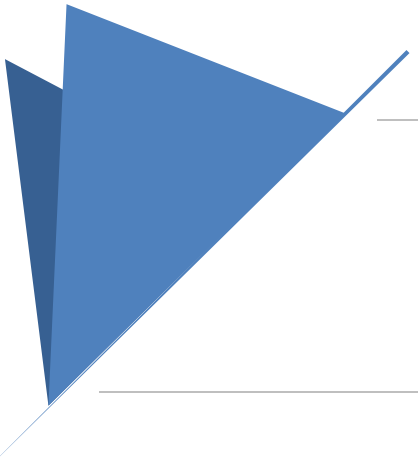
第08讲 队列

本讲作业

- **必做题**
  - ✓ 【课堂互动8.1】队列的基本概念
  - ✓ 【课堂互动8.2】循环队列
  - ✓ 【课堂互动8.3】链队列
  - ✓ 【课后作业08】[项目003] 栈和队列的综合应用
- **选做题**
  - ✓ 【课前自测08】队列
  - ✓ 【扩展练习08】队列
  - ✓ 【选做题08】[项目011] 栈和队列的综合应用（进阶-选做题）



- 队列的基本概念
- 双端队列
- 循环队列
- 链队列



队列的基本概念

/ 队列的定义（标准队列、双端队列）

/ 队列的抽象数据类型

/ 队列的顺序存储实现

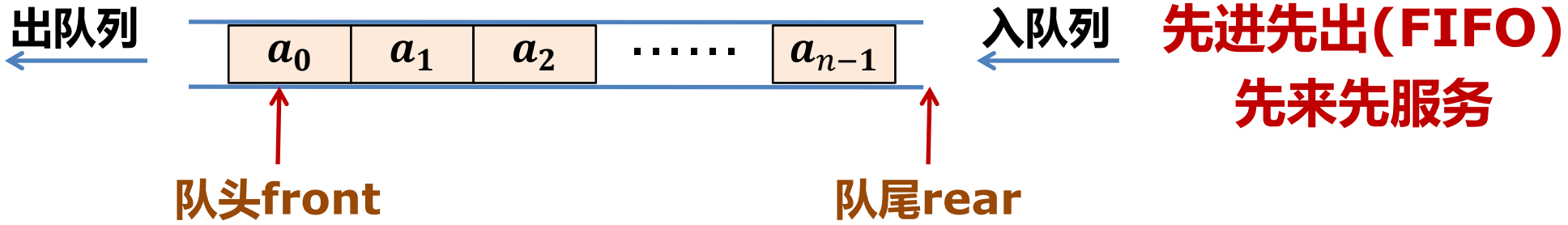
/ 队列的入队和出队

队列的基本概念

什么是队列

队列(Queue)是一种先进先出（FIFO）的线性表，其插入操作在队列的一端（队尾），而删除操作则在队列的另一端（队头）。数据的插入称为入队列；输出的删除称为出队列。

设队列 $Q=(a_0,a_1,\ldots,a_n)$，称a_0是对头元素，a_n是队尾元素。



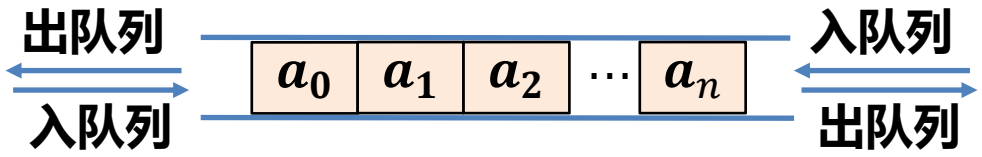
先进先出(FIFO)
先来先服务

双端队列

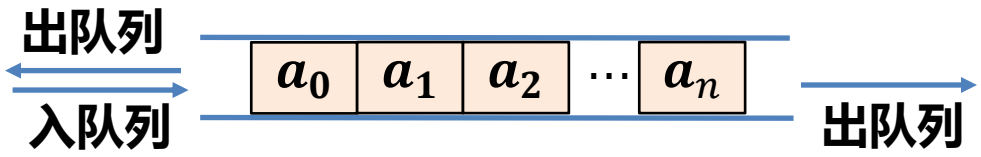
什么是双端队列

允许两端都可以进行入队和出队操作的队列称为**双端队列**。其中，

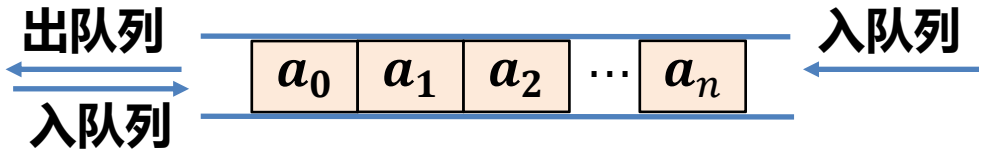
- 允许在一端进行插入和删除，但在另一端只允许删除的双端队列称为**输入受限的双端队列**；
- 允许在一端进行插入和删除，但在另一端只允许插入的双端队列称为**输出受限的双端队列**；
- 若限制双端队列从某端插入的元素只能从该端出队，则该双端队列蜕变为栈底相邻的两个邻接栈。



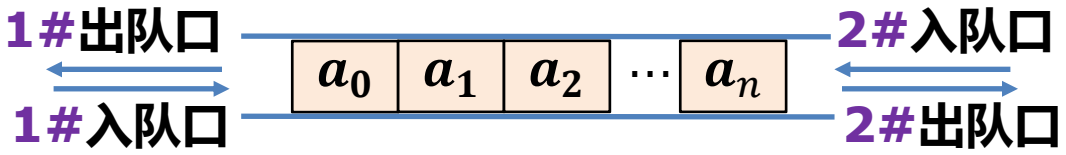
(a) 双端队列



(b) 输入受限的双端队列



(c) 输出受限的双端队列



(d) 邻接栈

队列的基本概念

队列的抽象数据类型描述

ADT Queue {

数据对象：一个包含0个或多个元素的有穷线性表。$D = \{a_i | a_i \in ElemSet, i \in \mathbb{N}, i \leq n\}$

数据关系：$R = \{< a_i, a_{i+1} > | a_i, a_{i+1} \in D, i \leq n\}$，其中$a_1$端为对头，$a_n$端为队尾

基本操作：

Status InitQueue(SqQueue &Q)：构造一个空循环队列;

bool QueueEmpty(SqQueue Q)：判断队列Q是否为空，返回 Ture|False;

bool QueueFull(&Q, int MaxSize)：判断队列是否已满;

int QueueLength(SqQueue Q)：获取循环队列的长度;

QElementType GetHead(SqQueue Q)：获取当前队头元素的值。

Status EnQueue(SqQueue &Q, QElemType e)：将元素e插入到队列Q中，成为新队尾。

QElementType DeQueue(SqQueue&Q, &e)：删除队列Q的队头元素，并用e返回其值。

} ADT Queue

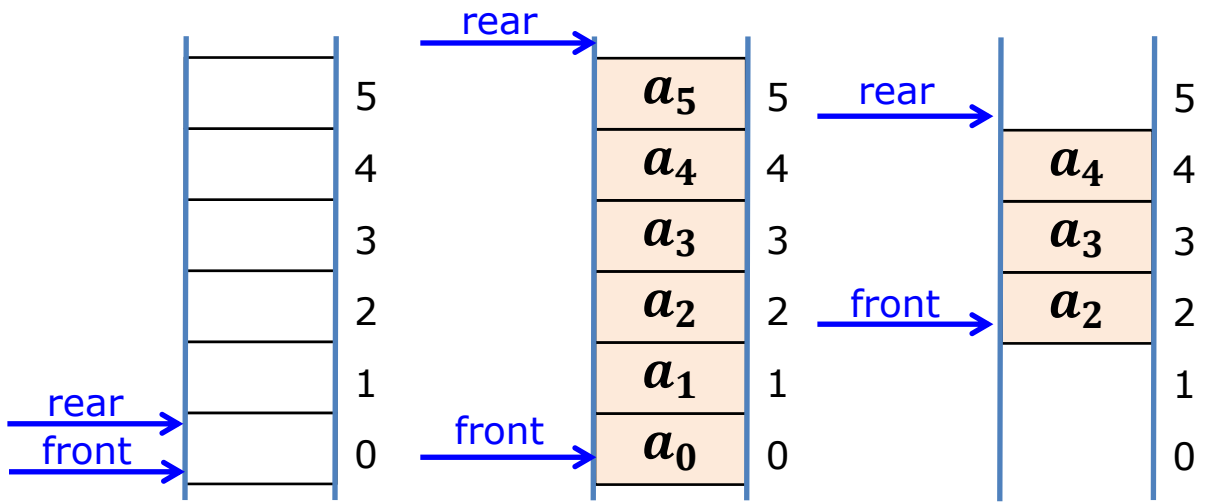
队列的基本概念

队列的顺序存储实现

队列（Queue）的顺序存储结构是由一块连续的存储单元存放队列的**一维数组**，它附设两个指针，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。

队列的顺序存储结构

```
#define MaxSize 100                 // 最大队列长
typedef struct SqQueue {            // 定义顺序队列
    QElementType data[MaxSize];     // 存储空间
    int rear;                       // 队尾指针
    int front;                      // 队头指针
};
```



空队列 **满队列** **一般情况**

Q.front=Q.rear=0

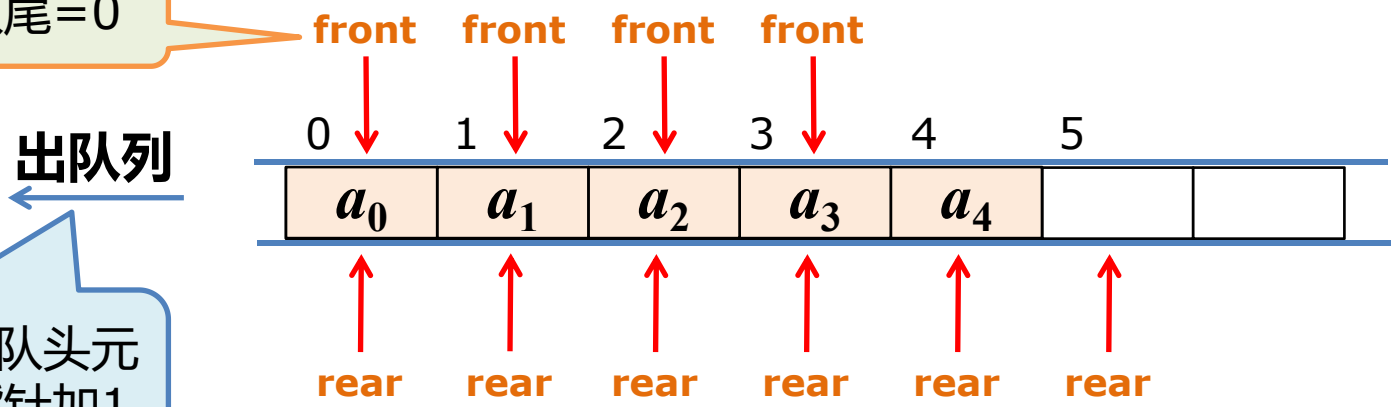
队列的基本概念

顺序队列的入队和出队

例1：假设存在一个MaxSize为7的队列，其入队和出队的顺序为 (0入,1入,2入,0出,3入,4入,1出,2出)，试给出其入对和出队的过程。

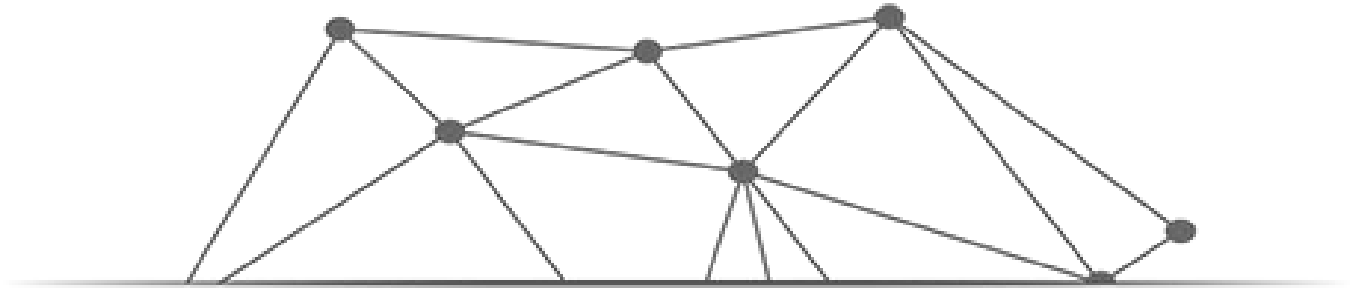
初始：队头=队尾=0

队不满时，先送值到队尾元素，再将队尾指针加1

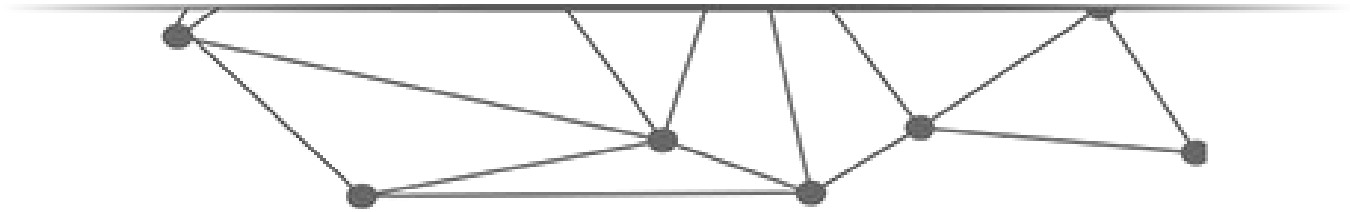


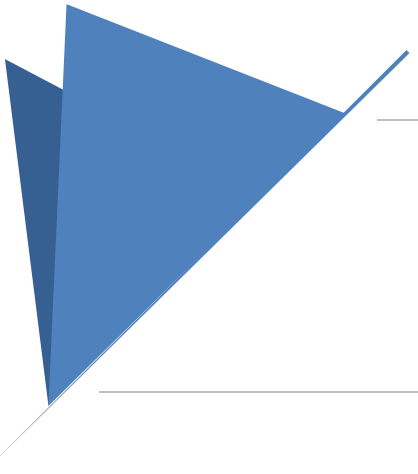
队不空时，先取队头元素，再将队头指针加1

EnQueue(a_0)	EnQueue(a_1)	EnQueue(a_2)	DeQueue(a_0)
EnQueue(a_3)	EnQueue(a_4)	DeQueue(a_1)	DeQueue(a_2)



课堂互动 8.1





循环队列

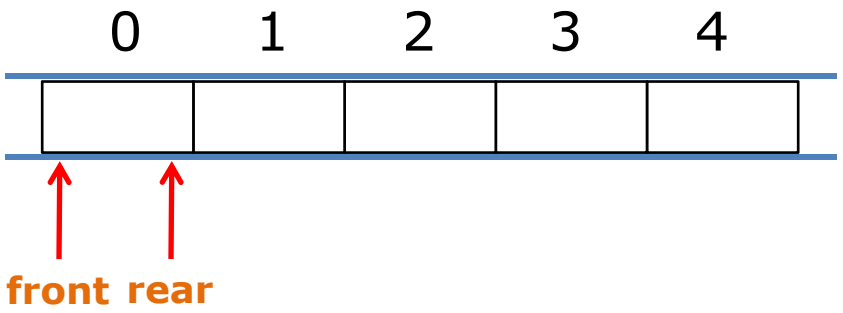
/ 顺序队列的假溢出

/ 循环队列的实现

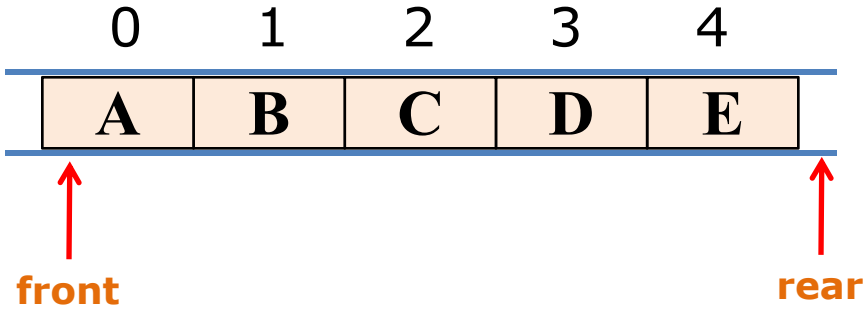
/ 循环队列的基本操作

循环队列

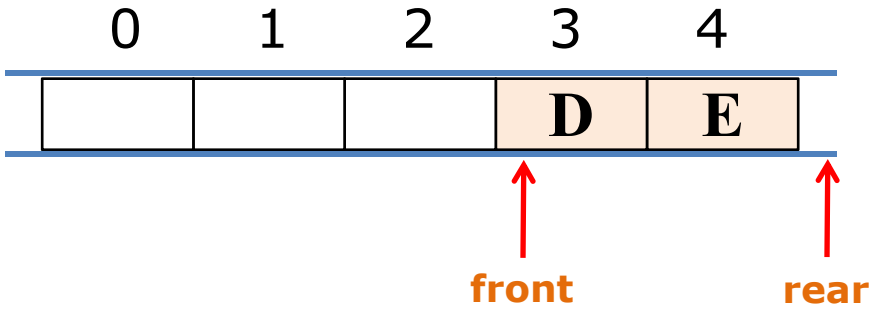
顺序队列的“假溢出”问题



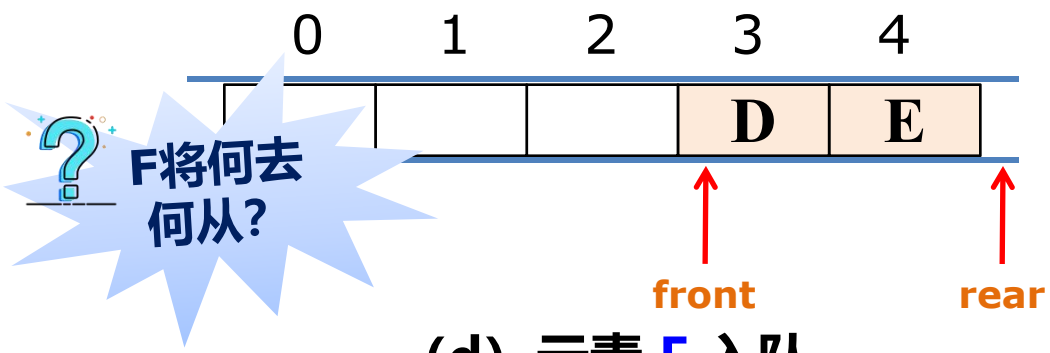
(a) 空队列



(b) 元素ABCDE依次入队



(c) 元素ABC依次出队



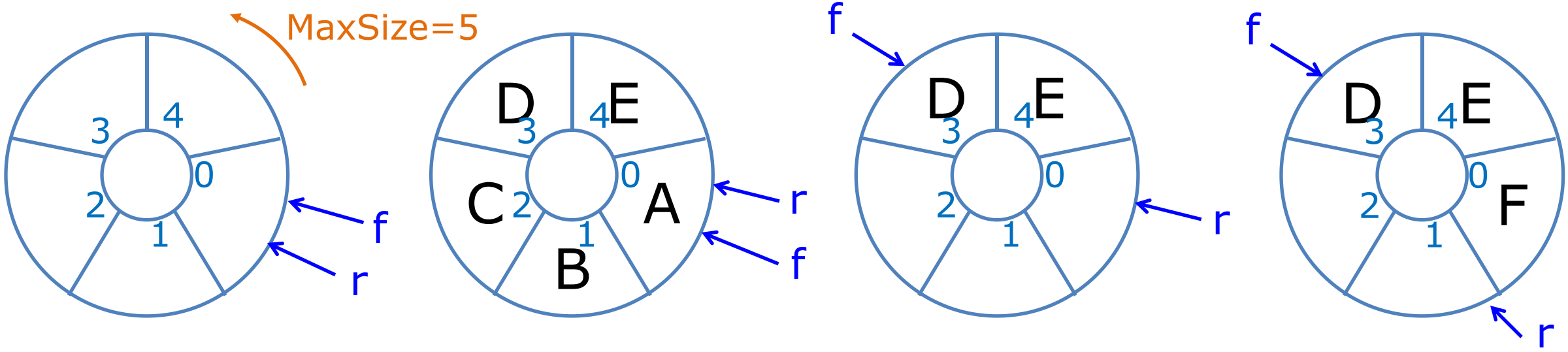
(d) 元素 F 入队

当需要将 新元素F 入队时，由于队尾非空，将产生“**溢出**”。
然而，队列中仍有“**空位**”，这种现象就称为“**假溢出**”。

循环队列

循环队列

将队列从逻辑上看成是一个首尾相连的环的队列称为**循环队列（Circular Queue）**。一般来说，循环队列也是一种顺序存储结构，使用取余运算**%**来连接队头和队尾。



（a）空队列　**（b）元素ABCDE依次入队**　**（c）元素ABC依次出队**　**（d）元素 F 入队**

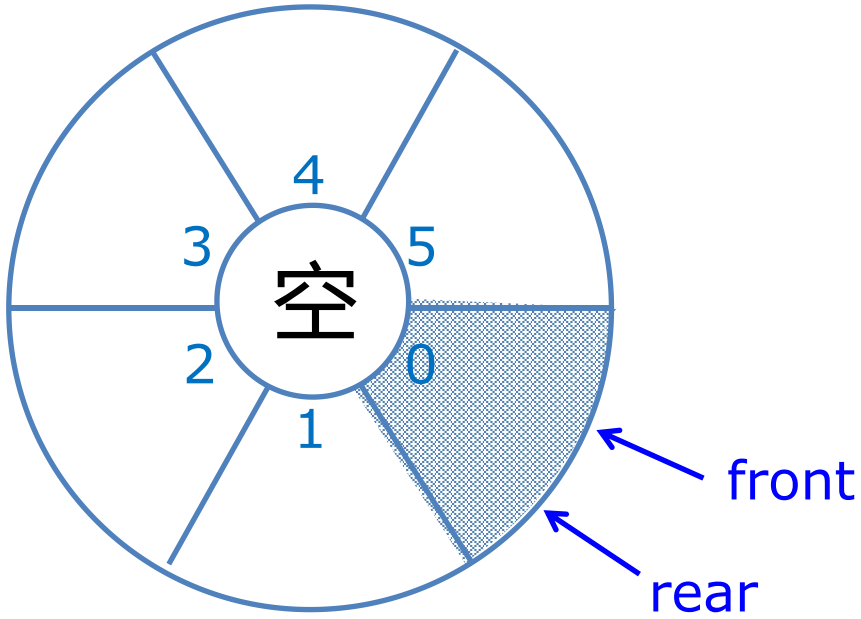
- 判断队空：Q.front == Q.rear
- 判断队满：Q.front == Q.rear

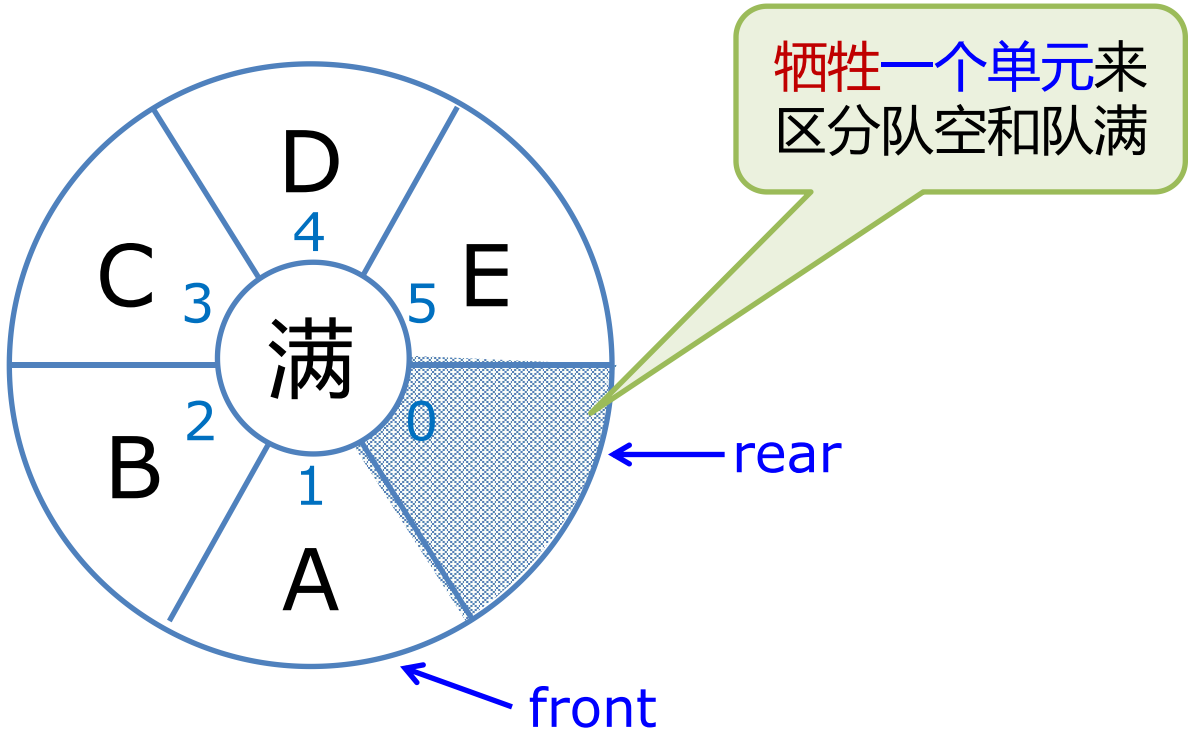
如何区分

循环队列可以有效解决假溢出问题。当队尾满后，新的队尾可以利用队列**空闲部分**。

循环队列

循环队列





- 队空：Q.front == Q.rear
- 队满：Q.front == (Q.rear+1)%MaxSize
- ★ **队列元素个数：(Q.rear - Q.front + MaxSize)%MaxSize**

循环队列

循环队列的抽象数据类型

ADT Queue {

数据对象：一个包含0个或多个元素的有穷线性表。$D = \{a_i | a_i \in ElemSet, i \in \mathbb{N}, i \leq n\}$

数据关系：$R = \{< a_i, a_{i+1} > | a_i, a_{i+1} \in D, i \leq n\}$，其中$a_1$端为对头，$a_n$端为队尾

基本操作：

Status InitQueue(SqQueue &Q)：构造一个空循环队列;

bool QueueEmpty(SqQueue Q)：判断队列Q是否为空，返回 Ture|False;

bool QueueFull(&Q, int MaxSize)：判断队列是否已满;

int QueueLength(SqQueue Q)：获取循环队列的长度;

QElementType GetHead(SqQueue Q)：获取当前队头元素的值。

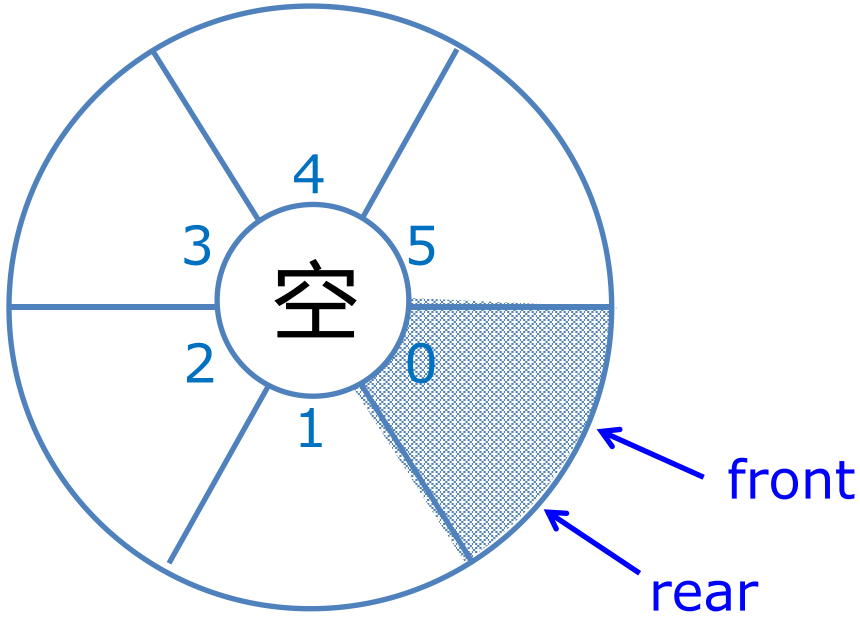
Status EnQueue(SqQueue &Q, QElemType e)：将元素e插入到队列Q中，成为新队尾。

QElementType DeQueue(SqQueue&Q, &e)：删除队列Q的队头元素，并用e返回其值。

} ADT Queue

循环队列

循环队列的初始化



初始化一个新的循环队列

```
Status InitQueue (SqQueue &Q) {
    Q.data = new QELemType[MaxSize];
    if (!Q.data)
        exit (Overflow);
    Q.front = Q.rear = 0;
    return OK;
}
```

1. 构造一个最大容量为MaxSize的数组空间用于存储队列
2. 判断存储空间是否分配成功
3. 初始化队首和队尾，使它们同时指向地址为0的空间

循环队列

循环队列的基本操作

判断循环队列是否为空

```
bool QueueEmpty(SqQueue Q) {
    if (Q.front == Q.rear)          // 满足判空条件
        return true;                // 返回true
    else    return false;
}
```

判断循环队列是否为满

```
bool QueueEmpty(SqQueue Q) {
    if (Q.front == (Q.rear+1)%MaxSize  // 满足判满条件
        return true;                   // 返回 true
    else    return false;
}
```

取队头元素

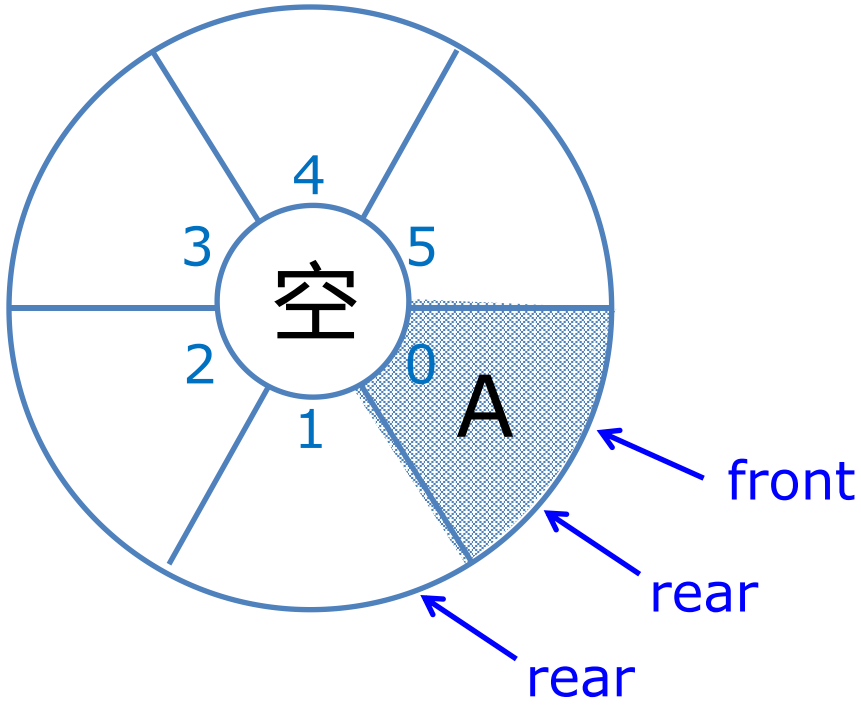
```
QElementType GetHead(SqQueue Q) {
    if (Q.front != Q.rear)          // 队列非空
        return Q.data[Q.front]      // 返回队头元素
    else    return false;
}
```

获取队列长度

```
int QueueLength(SqQueue Q) {
    QLength = (Q.rear - Q.front + MaxSize)%MaxSize
    return QLength;                 // 返回队列长度
}
```

循环队列

循环队列的入队

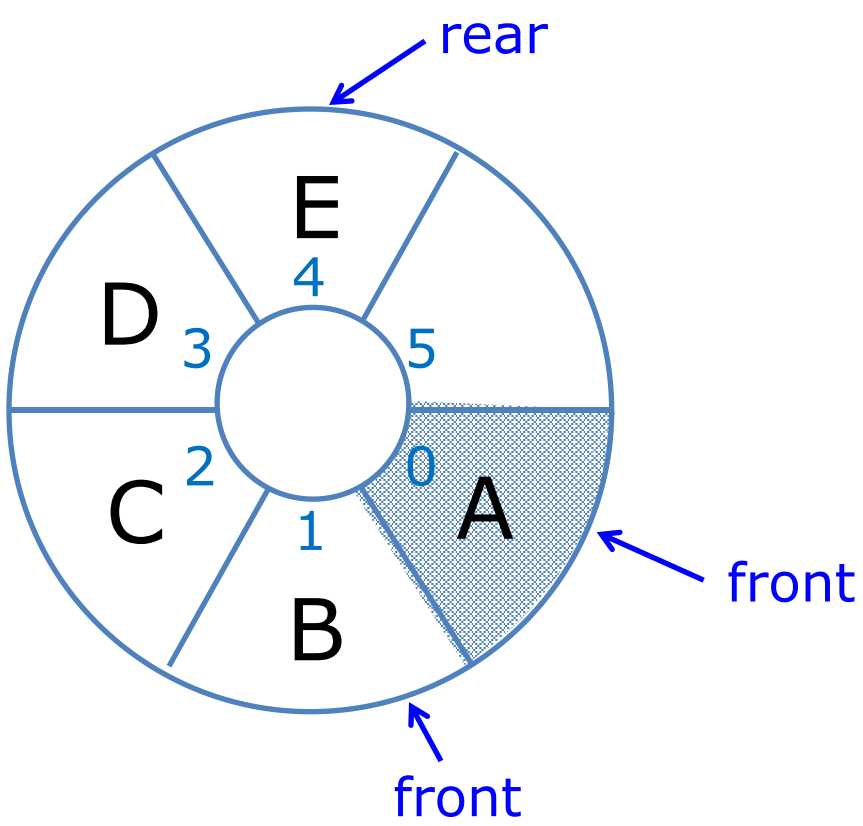


入队：在队尾插入一个新的元素

```
Status EnQueue(SqQueue &Q, QElemType e) {
    if (Q.front == (Q.rear+1)%MaxSize)
        printf( “队列满” );
        return false;
    Q.data[Q.rear] = e;
    Q.rear = (Q.rear + 1)%MaxSize;
    return ture;
}
```

1. 判断队列是否已满，若未满则执行插入，否则返回错误，
2. 将新元素插入队尾
3. 队尾指针加1

循环队列

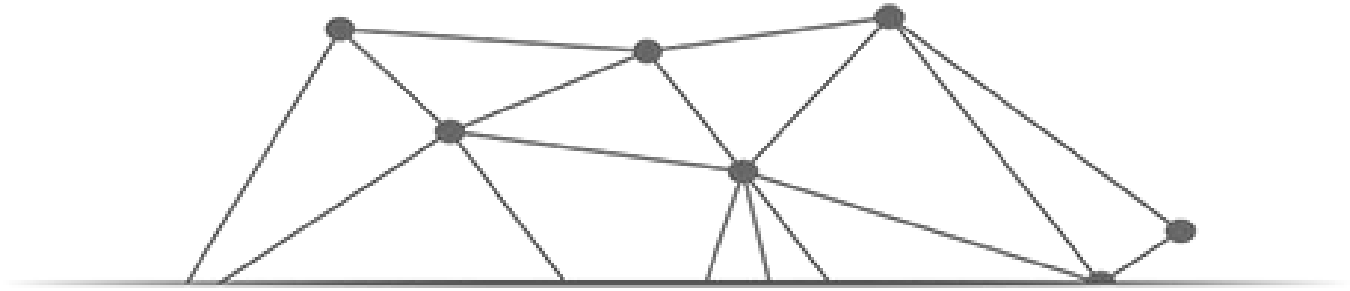


循环队列的出队

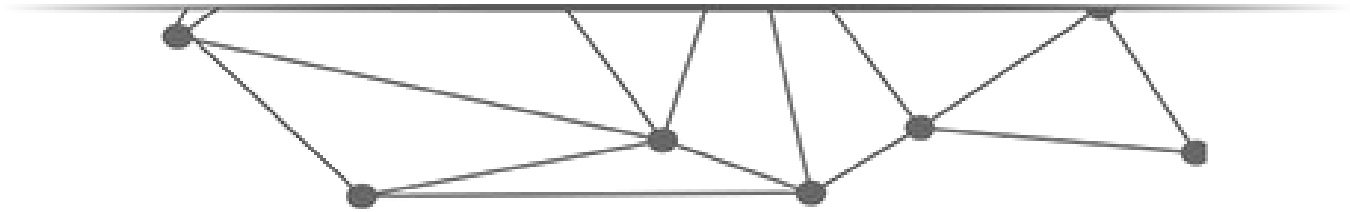
出队：将对头元素删除

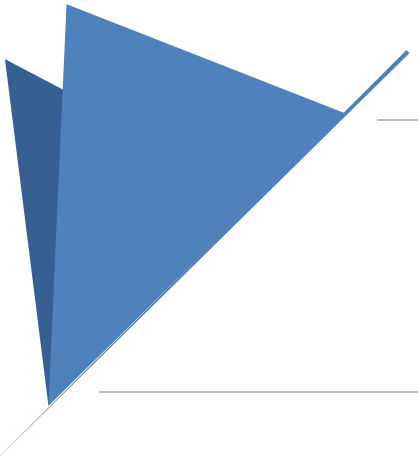
```
QElementType DeQueue(SqQueue&Q, &e) {
    if (Q.front == Q.rear)
        printf(“队列空”);
        return false;
    e = Q.data[Q.front];
    Q.front = (Q.front + 1)%MaxSize;
    return ture;
}
```

1. 判断队列是否为空，若非空则继续
2. 将队头元素保存到变量e
3. 队头指针加1



课堂互动 8.2





链式队列

/ 队列的链式表示和实现

/ 链式队列的基本操作（初始化、判空、取队头、入队、出队）

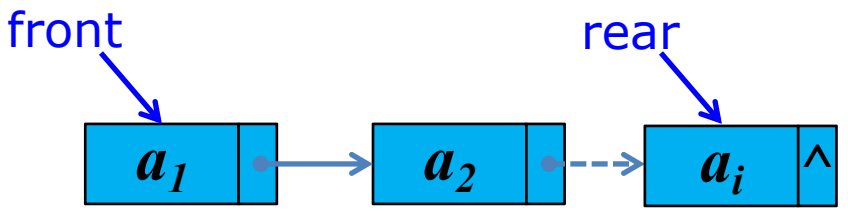
链式队列

队列的链式表示和实现

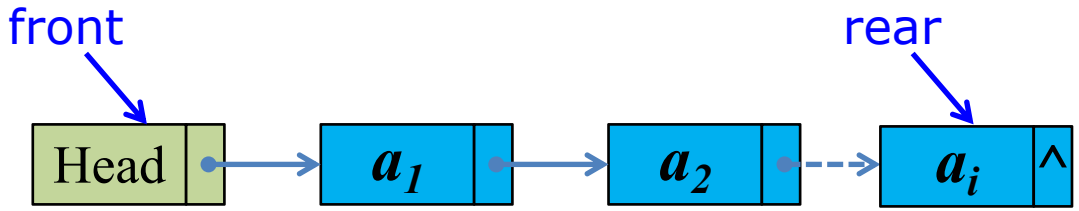
队列的链式存储结构称为**链队列（LinkQueue）**，它是一个同时带头指针和尾指针的单链表。头指针（front）指向队头结点（或头结点），尾指针（rear）指向队尾结点（顺序存储通常指向对尾结点的下一结点）。

队列的链式存储结构

```
typedef struct LinkNode{
    QElemType data;
    struct LinkNode *next;
} LinkNode;
typedef struct LinkQueue {
      LinkNode front;                  // 队头指针
      LinkNode rear;                   // 队尾指针
} LinkQueue;
```



不带头结点的链队列



带头结点的链队列

链式队列

链式队列的基本操作

初始化一个新的链式队列

```
Status InitQueue (LinkQueue &Q) {
    Q.front = Q.rear = new LinkNode;    // 创建一个新结点
    Q.front->next = NULL;               // 将新结点的next置空
    return OK;
}
```

判断链队列是否为空

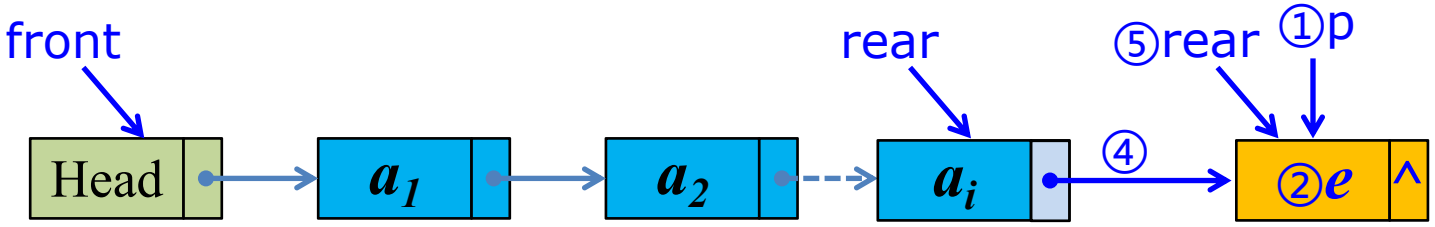
```
bool QueueEmpty(LinkQueue Q)
    if (Q.front == Q.rear)    return true;
    else    return false;
}
```

取链队列的队头元素

```
QElementType GetHead(LinkQueue Q) {
    if (Q.front != Q.rear)                    // 队列非空
        return Q.front->next->data;           // 返回队头元素的值，队头指针不变
}
```

链式队列

链式队列的入队



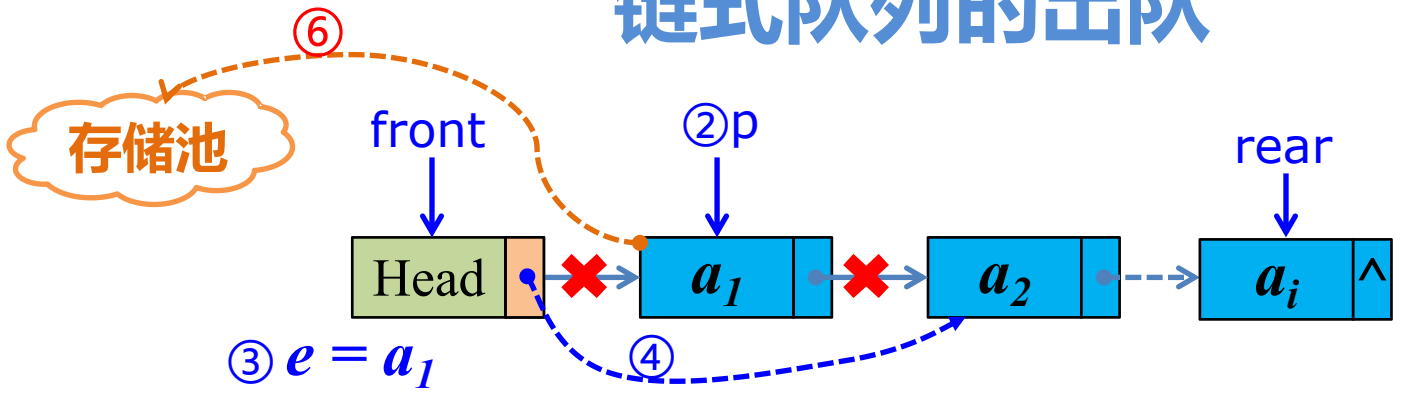
1. 创建一个新结点，并用p指向
2. 将新结点的数据域置为e
3. 将新结点的next域置空
4. 将原来尾结点的next域指向新结点
5. 尾指针指向新结点

入队：在队尾插入一个新的元素

```
Status EnQueue(LinkQueue &Q, QElemType e) {
    p = new LinkNode;
    p->data = e;
    p->next = NULL;
    Q.rear->next = p;
    Q.rear = p;
    return ture;
}
```

链式队列

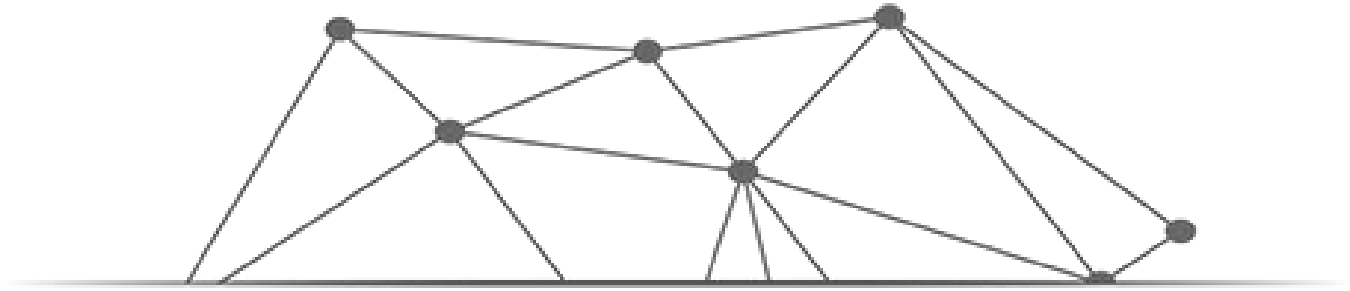
链式队列的出队



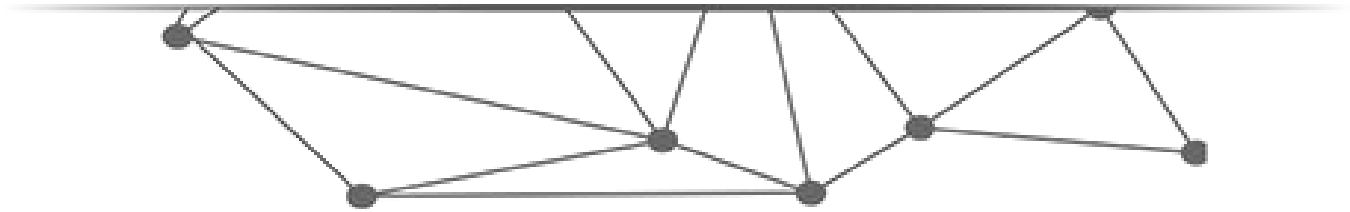
1. 判断队列是否为空，若空则返回Error
2. 将工作指针p指向队头元素
3. 暂时保存队头元素的值，以释放空间
4. 修改头结点指针，指向下一结点
5. 判断出队元素是否是最后一个元素，若是最后一个则将队尾指向队头
6. 释放原队头元素的空间

入队：在队尾插入一个新的元素

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



课堂互动 8.3



第09讲 队列

小　结

- 队列是限定仅在一端进行插入（队尾），在另一端进行删除（队头）的线性表。
- 队列具有先进先出的特点
- 循环队列可以解决标准队列假溢出问题，它通过对MaxSize求模实现队头队尾指针的连接
- 队列和栈都是都是受限制的线性表，数据元素间具有1对1的关系
- 顺序存储的栈和队列都需要预先分配存储空间，可能会出现空间闲置或栈满溢出的现象，且数据元素无法自由扩充
- 链式存储的栈和队列采用动态分配内存空间，不会出现空间闲置或栈满溢出的现象，且数据元素可自由扩充

欧老师的联系方式

读万卷书 行万里路 只为最好的修炼

QQ：14777591 (宇宙骑士)

Email：ouxinyu@alumni.hust.edu.cn

Website: http://ouxinyu.cn

Tel：18687840023

地址：安宁校区 诚远楼201

南院 智能应用研究院A306-2