数据结构

深圳技术大学
大数据与互联网学院

第三章 栈和队列

3.1 栈
3.2 栈的应用举例
3.3 栈与递归的实现
3.4 队列

上节小结

- 栈是限定操作的性表，特点先进后出
- 堆栈包括top栈顶、base栈底，插入、删除、访问都在top进行
  - 插入就是进栈push，top+1
  - 删除就是出栈pop，top-1
  - 访问就是取栈顶元素GetTop
  - top=base或top->next = NULL表示栈空
  - base=NULL栈不存在
  - top>stacksize时，栈满
- 堆栈的应用：数制转换、行编辑、括号匹配、迷宫求解

3.4 队列

一. 队列的概念

- 队列是只允许在表的一端进行插入，而在另一端删除元素的线性表
- 在队列中，允许插入的一端叫队尾（rear），允许删除的一端称为队头(front)
- 特点：先进先出 (FIFO)

3.4 队列

二. 顺序队列

- 顺序队列是队列的一种实现
- 顺序队列采用一组地址连续的存储单元依次存储从队列头到队列尾的元素
- 顺序队列有两个指针：队头指针front和队尾指针rear在队列中，允许插入的一端叫队尾（rear），允许删除的一端称为队头(front)

3.4 队列

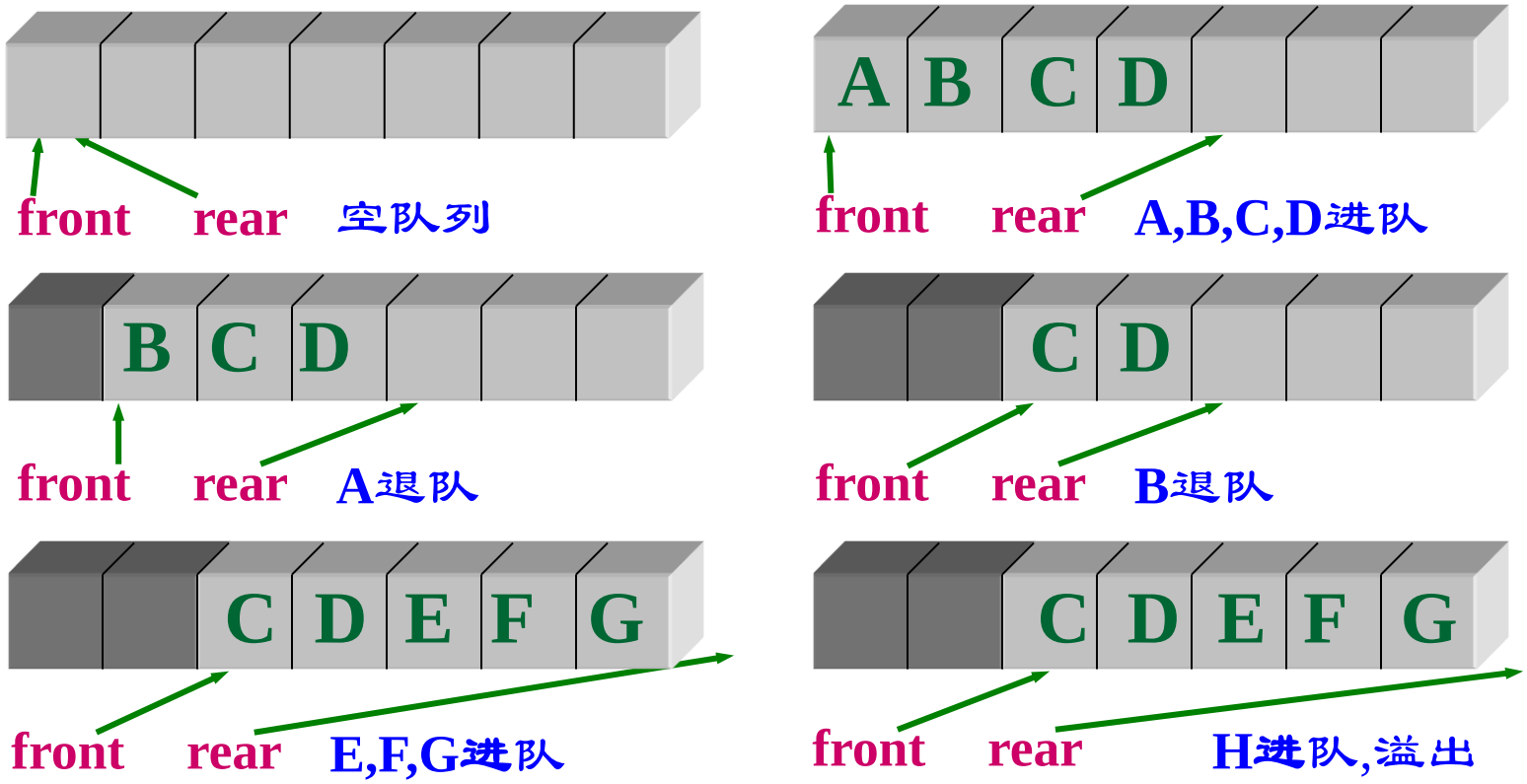
二. 顺序队列

- 顺序队列的进队和出队原则：
  - 进队时，新元素按rear指针位置插入，然后队尾指针增一，即 rear = rear + 1
  - 出队时，将队头指针位置的元素取出，然后队头指针增一，即 front = front + 1
  - 队头指针始终指向队列头元素
  - 队尾指针始终指向队列尾元素的下一个位置

3.4 队列

二. 顺序队列

- 顺序队列的进出队举例：



3.4 队列

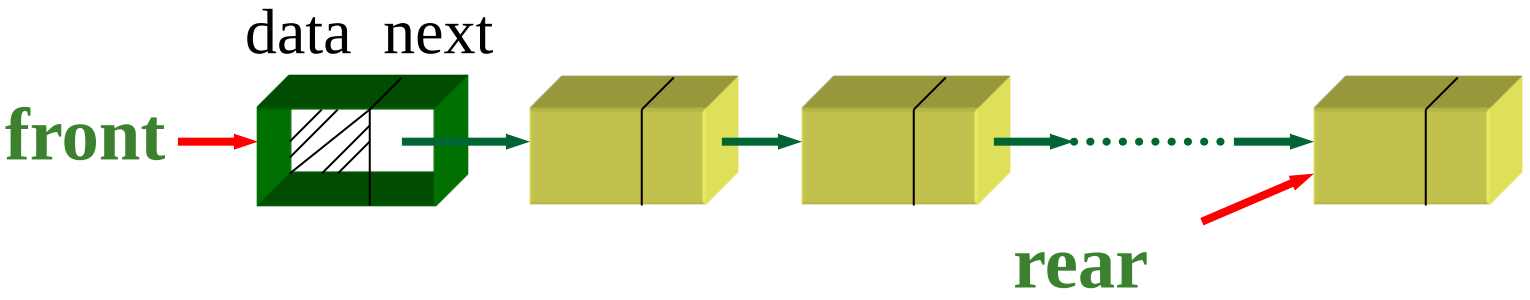
二. 顺序队列

- 顺序队列的问题：
  - 当队尾指针指向队列存储结构中的最后单元时，如果再继续插入新的元素，则会产生溢出
  - 当队列发生溢出时，队列存储结构中可能还存在一些空白位置（已被取走数据的元素）
- 解决办法之一：队列用链表实现
- 解决办法之二：将队列存储结构首尾相接，形成循环(环形)队列

3.4 队列

三. 单链队列

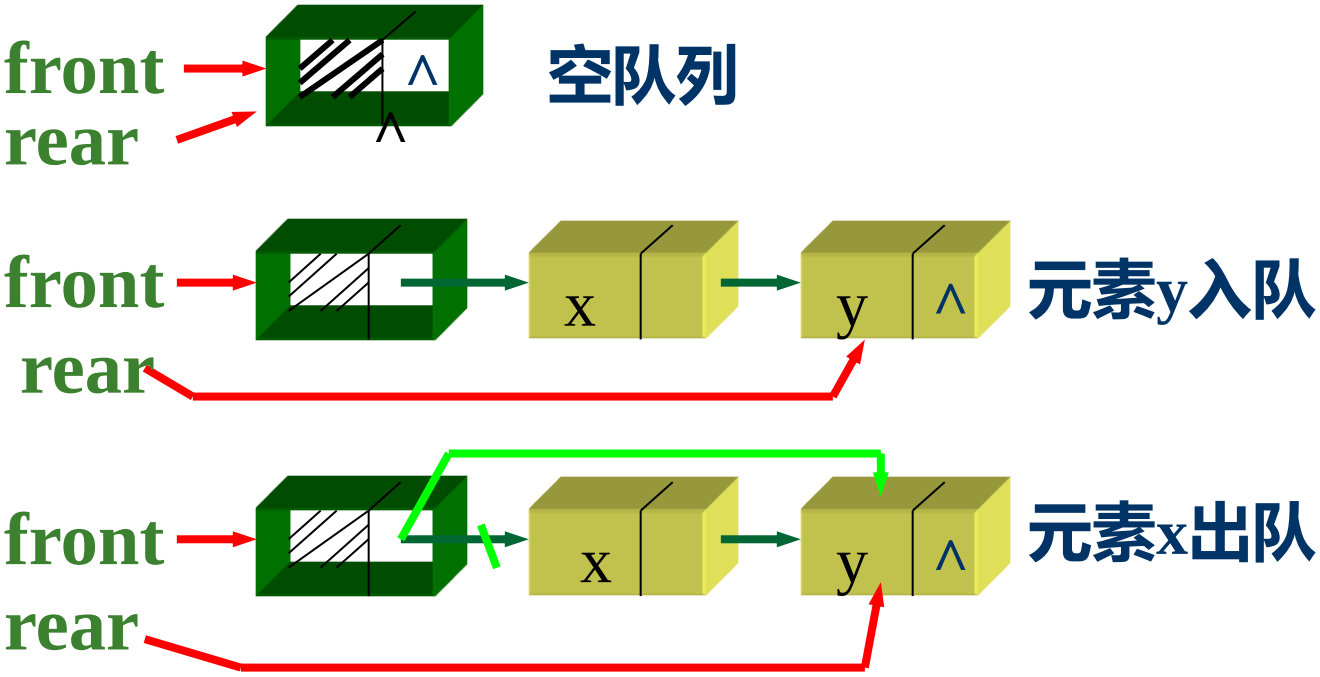
- 链队列采用链表存储单元
- 链队列中，有两个分别指示队头和队尾的指针
- 链式队列在进队时无队满问题，但有队空问题



3.4 队列

三. 单链队列

- 链队列操作实质和链表操作一样，只是多了front和rear的操作



3.4 队列

三. 单链队列

- 单链队列的定义

```
class QNode{
    QElemType date;
     QNode *next;
};
class LinkQueue{
    QNode *front;
    QNode * rear;
    ... // 其他成员
};
```

3.4 队列

三. 单链队列

- 单链队列的初始化

```
Status LinkQueue::InitQueue(){
    front = new QNode;
      if(!front) exit(OVERFLOW);
      front->next = NULL;
      rear = front;
      return OK;
}
```

3.4 队列

三. 单链队列

- 单链队列的插入

```
Status LinkQueue::EnQueue(QElemType e){
    if(!front) InitQueue(); // 列表未初始化则初始化
    QNode *p = new QNode;
    if(!p) exit(OVERFLOW);
    p->date = e;
    p->next = NULL;
    rear->next = p; // 当前队尾结点指向新结点
    rear = p;  // 更新新结点为队尾
}
```

3.4 队列

三. 单链队列

- 单链队列的删除

```
Status LinkQueue::DeQueue(QElemType &e){
    if (front == rear)  return ERROR;  // 空队列
    QNode *p = front->next; // 获得当前队首结点
    Q.front->next = p->next;
    e = p->data;
    if (rear==p)  rear = front; // 取出结点恰好为队尾
    delete p;
    return OK;
}
```

3.4 队列

三. 单链队列

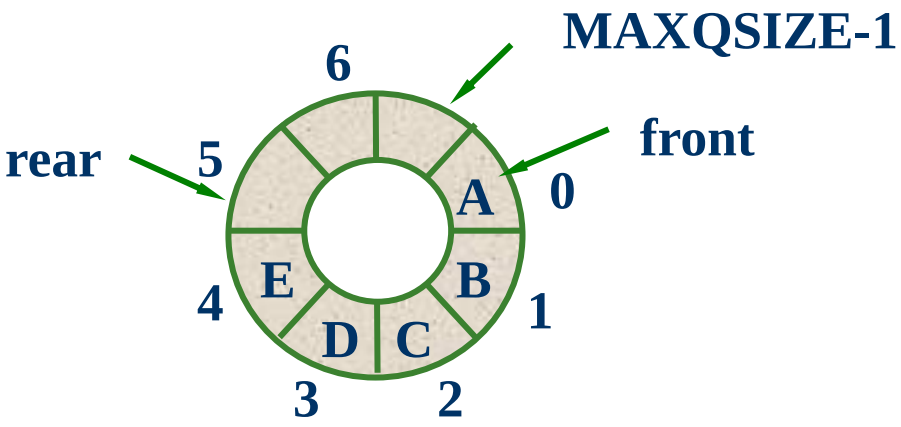
- 单链队列的销毁

```
void LinkQueue::DestoryQueue(LinkQueue &Q){
    while(front != rear){
        QElemType e;
        DeQueue(e);
    }
}
```

3.4 队列

四. 循环队列

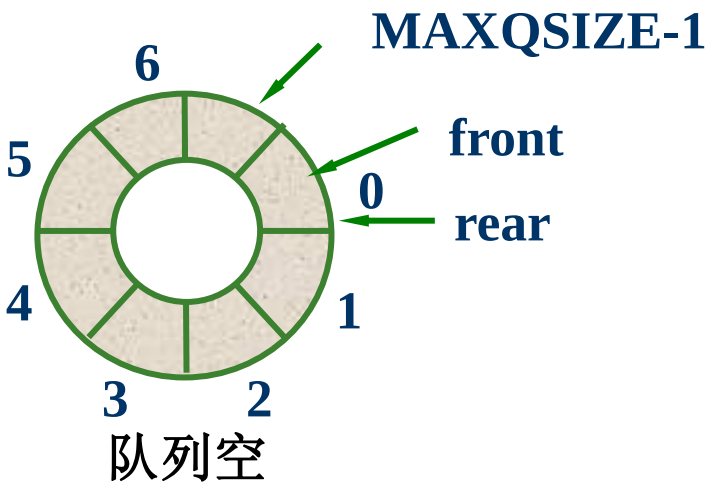
- 循环队列采用一组地址连续的存储单元
- 将整个队列的存储单元首尾相连



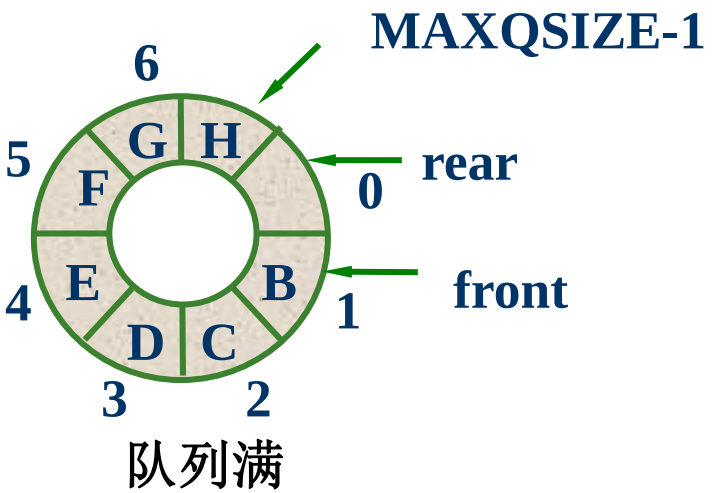
3.4 队列

四. 循环队列

- 循环队列的空与满
  - front = rear，循环队列空
  - （rear+1）% MAXQSIZE = front，循环队列满，少用一个元素空间



队列空



队列满

3.4 队列

四. 循环队列

- 循环队列的定义

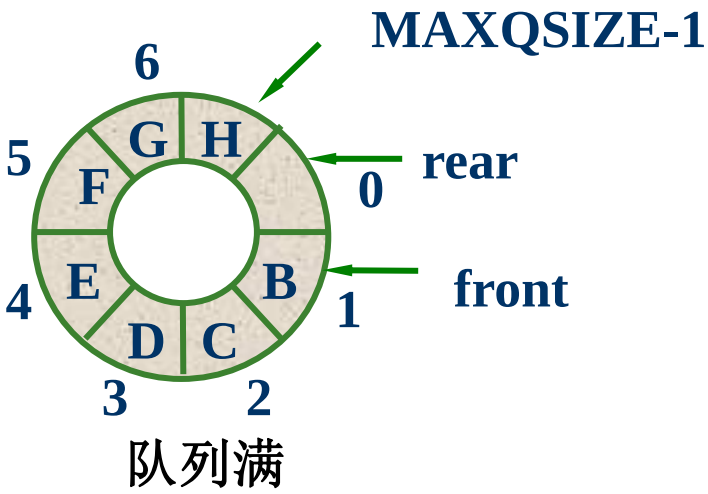
```
#define MAXQSIZE 100
class SqQueue {
    QElemType  *base;
    int front;  // 指向队首的下标
    int rear;  // 指向队尾的下标
};
Status SqQueue::InitQueue() {
    base = new QElemType[MAXQSIZE];
    if (!base) exit(OVERFLOW);     // 存储分配失败
    front = rear = 0;  // 下标从0开始
    return OK;
};
```

3.4 队列

四. 循环队列

- 循环队列的插入

```
Status SqQueue::EnQueue(QElemType e) {
    if ((rear + 1) % MAXQSIZE == front)
        return ERROR; //队满
    base[rear] = e;
    rear = (rear + 1) % MAXQSIZE;
    return OK;
}
```



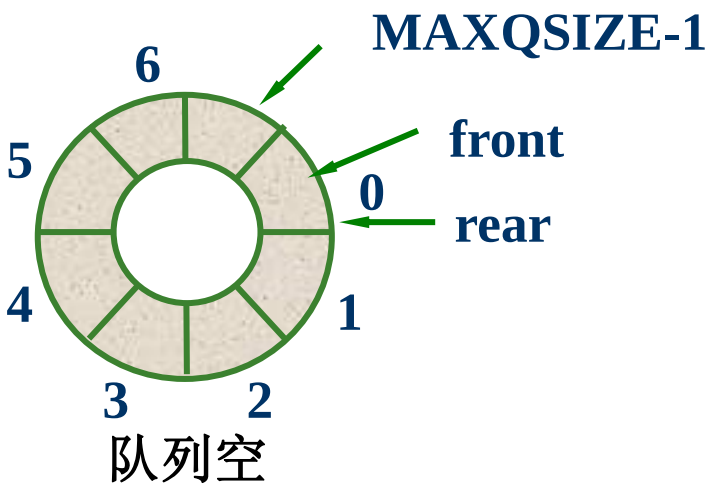
队列满

3.4 队列

四. 循环队列

- 循环队列的删除

```
Status SqQueue::DeQueue(QElemType &e) {
        if (rear == front) return ERROR; //队空
        e = base[front];
        front = (front + 1) % MAXQSIZE;
        return OK;
}
```



队列空

练习

一. 循环队列的队首和队尾下标分别为f，r，最大长度为n，写出判断队满和队空的条件

- 若n=8，f=3，r=2，请判断队列是否满或空
- 若n=9，f=3，r=6，队列从0开始编号，请指出哪几个位置为空

Take Home Message

- 队列：先进先出
  - 只在队尾端进行数据进队，只在队首进行数据出队
- 队列的种类
  - 顺序表示：使用简单但数组容易溢出
  - 链表表示：保留front和rear两个指针
  - 循环顺序表示：front==rear时为空

    ((rear + 1) % MAXQSIZE == front时为满
- 队的应用：
  - 活用先进先出特点