

Queue

基本概念

- 队尾插入
- 队头删除

考虑平时排队找队尾

- 逻辑结构：一对一，和线性表相同
- 存储结构：顺序队、链队、循环顺序队
- 运算规则：队首、队尾运算，访问节点依照先进先出（FIFO）原则
- 实现方式：入队、出队
- 基本操作：入队、出队、建空队列、判断空、满

队列的基本概念

队列定义

只能在表的一端进行插入运算，在表的另一端进行删除运算的线性表。

队尾插入

逻辑结构

与线性表相同，仍为一对关系。

队头删除

存储结构

顺序队或链队，以循环顺序队更常见。

运算规则

只能在队首和队尾运算，且访问结点时依照先进先出（FIFO）的原则。

实现方式

关键是掌握入队和出队操作，具体实现依顺序队或链队的不同而不同。

基本操作：

入队或出队，建空队列，判队空或队满等操作。

2

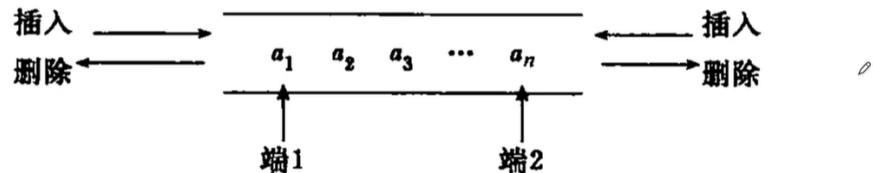
为啥设计队列：

- 模拟离散时间
- 操作系统的作业调度

双端队列

双端队列 (Deque)

- Deque是double-ended queue的缩写，区别于出队列DeQueue。
- 介于“栈”和“队列”之间，两端都可以进行操作。

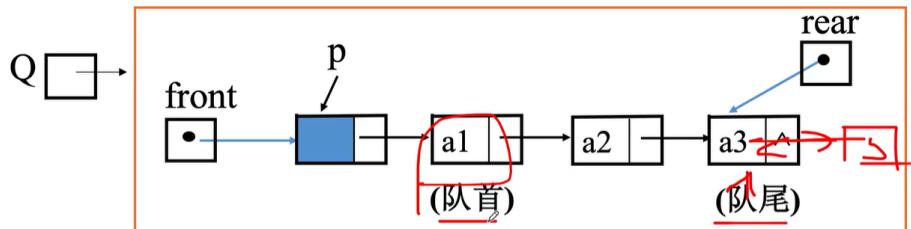


队列的链表表示和实现

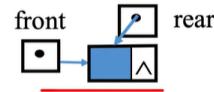
```
//结点类型的定义:  
template<typename QElemType>  
typedef struct QNode{  
    QElemType data;  
    struct QNode * next;  
}Qnode, *QueuePtr;  
  
//链队列类型的定义:  
typedef struct{  
    QueuePtr front;  
    QueuePtr rear;  
}LinkQueue; // 关于整个链队的总体描述
```

队列示意图和备注

链队示意图：

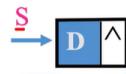


① 空链队的特征? front==rear



② 链队会满吗? 一般不会, 因为删除时有free动作。除非内存不足!

③ 怎样实现链队的入队和出队操作?

入队 (尾部插入) : rear->next=S rear=S; 

出队 (头部删除) : p=front->next; front->next=p->next; free(p);

10

入队、出队实现

```
Status EnQueue(LinkQueue &Q, QELEMType e)
{//向队列Q插入新的队尾元素e
    p = (QueuePtr) malloc (sizeof(Qnode));
    if(!p) exit(Error); // 存储分配失败
    p->data = e;
    p->next = NULL;
    Q.rear->next = p;
    Q.rear = p;
    return OK;
}
```

```
Status DeQueue(LinkQueue &Q, QELEMType &e)
{
    //若队列不空, 则删除Q的对头元素, 用e返回其值
    //并返回True, 否则Error
    if(Q.front == Q.rear) return Error;
    p = Q.front->next;
    e = p->data;
    Q.front->next = p->next;
    // 当仅有一个元素, 即rear一开始就是Q.front->next时, 需要将rear重新指向front
    // 否则删除p之后, rear悬空。
    if(Q.rear==p) Q.rear = Q.front; // Important! !
    free(p);
    return OK;
}
```

队列的顺序表示和实现

```
#define MAXQSIZE 100
typedef struct{
    QElemType *base; // 队列的基地址
    int front; // 队首指针，指向列头元素
    int rear; // 队尾指针，指向队尾元素的下一个位置
} SqQueue;
// 建对的核心语句
Q.base = (QElemType*) malloc(sizeof(QElemType) * MAXQSIZE); // 动态分配空间
```

示意图和注意事项

顺序队示意图：

讨论：

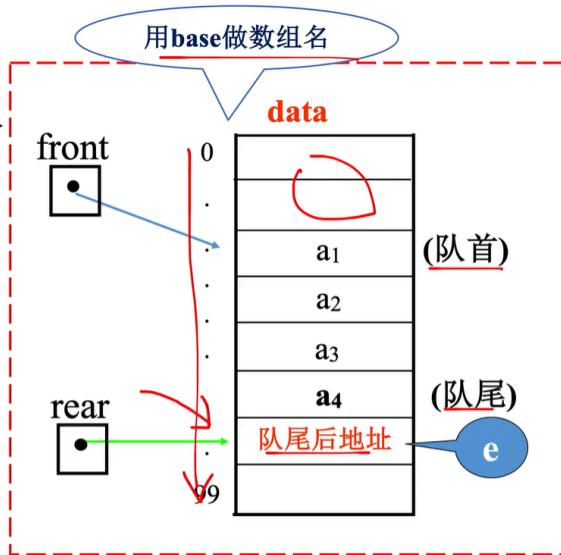
① 空队列的特征？

约定： **front=rear**

② 队列会满吗？

极易装满！ 因为数组通常有**长度限制**，而其前**端空间无法释放**。

③ 怎样实现入队和出队操作？**核心语句如下：**



入队(尾部插入): **Q.base[Q.rear]=e; Q.rear++;**

出队(头部删除): **e=Q.base[Q.front]; Q.front++;**

16

基本操作：

- 初始化空队

```
Q.front = Q.rear = 0;
```

- 队空条件

```
Q.front == Q.rear
```

说明：不一定等于0

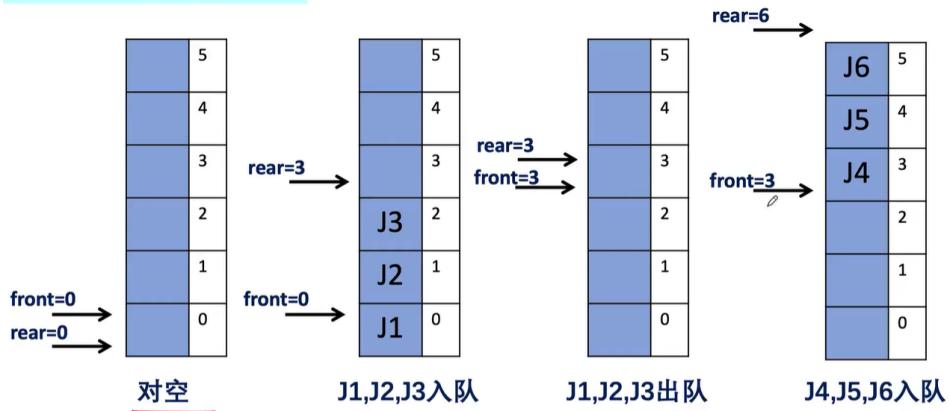
- 队满条件

```
Q.rear == MAXQSIZE
```

说明：由于"Q.rear"作为一个数组下标一样的存在，所以当Q.rear==MAXQSIZE时，就已经满了

操作示意图

顺序队操作示意图：



设两个指针front,rear,约定：
rear指示队尾元素前一位置；
front指示队头元素
初值front=rear=0

空队列条件：front==rear
入队列：base[rear++]=x;
出队列：x=base[front++];

假溢出

- 在顺序队列中，当指针已经到了数组上界，不能再有入队操作。但其实，数组中还有空的位置
- 如上图中J4, J5, J6入队之后
- 采用循环队列解决

顺序存储的循环队列

- 将顺序队列设想为首尾相连的环状空间
- 当Q.rear值超出空间最大位置时，令Q.rear=0
- Q.front "越界" 时也应有对应操作

循环队列的基本操作：

假上溢的解决办法

把顺序队列看成首尾相接的环(钟表)-循环队列

基本操作的实现

- 入队: $Q.\underline{\text{rear}} = (\underline{Q.\text{rear}} + 1) \% \underline{\text{MAXSIZE}}$
- 出队: $Q.\underline{\text{front}} = (\underline{Q.\text{front}} + 1) \% \underline{\text{MAXSIZE}}$
- 队空条件: $Q.\underline{\text{front}} == Q.\underline{\text{rear}}$
由于出队 $\underline{Q.\text{front}}$ 追上了 $\underline{Q.\text{rear}}$
- 队满条件: $Q.\underline{\text{front}} == Q.\underline{\text{rear}}$
由于入队 $\underline{Q.\text{rear}}$ 追上了 $\underline{Q.\text{front}}$

问题: 队空和队满的判断条件一样

解决对空堆满判断条件相同 (3个)

- 计数器, 记录队列元素个数
- 标志位, 删除时置1, 插入时置0
 - $\text{front} == \text{rear}$ 且 1 --> 因删除而相等 --> 队空
 - $\text{front} == \text{rear}$ 且 0 --> 因插入而相等 --> 队满
- 人为浪费一个单元: 申请 $N+1$ 个空间而认为只有 N 个
 - 则队满特征变为:

```
front = (rear + 1) % N;
```

- 队空仍是

```
front = rear;
```

- 队列长度

```
L = (N+rear-front) % N;
```

实际中常选用方案3 (人为浪费一个单元) :

即front和rear二者之一指向实元素，另一个指向空闲元素。

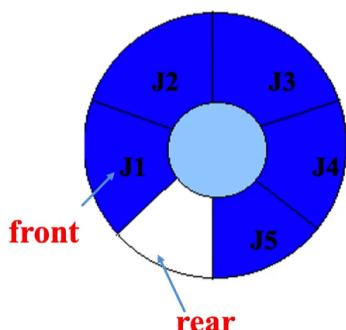
队空条件 : **front = rear**

(初始化时: **front = rear**)

队满条件: **front = (rear+1) % N**

($N = \text{maxsize}$)

队列长度 (即数据元素个数) : **$L = (\underline{N + rear - front}) \% N$**



问1: 左图中队列 **maxsize N=?** 6

问2: 左图中队列 **长度 L=?** 5

问3: 在具有 **n** 个单元的循环队列中, 队满时共有多少个元素?

N-1个

26

循环队列:

■ 队列存放数组被当作首尾相接的表处理。

■ 队头、队尾指针加1时从 $\text{maxSize} - 1$ 直接进到0, 可用语言的取模(余数)运算实现。

队空: $Q.\text{front} == Q.\text{rear}$

队满: $Q.\text{front} == (Q.\text{rear} + 1) \% \text{maxSize}$

入队: $Q.\text{rear} = (Q.\text{rear} + 1) \% \text{maxSize}$

出队: $Q.\text{front} = (front + 1) \% \text{maxSize};$

求队长: $(Q.\text{rear}-Q.\text{front}+\text{maxSize}) \% \text{maxSize}$

入队、出队的代码实现

```
// 入队
Status EnQueue(SqQueue& Q, QElemType e)
{
    if((Q.rear+1)%MAXQSIZE == Q.front) return ERROR; // 队列满
    Q.base[Q.rear] = e;
    Q.rear = (Q.rear + 1) % MAXQSIZE;
    return OK;
}

// 出队
Status DeQueue(SqQueue& Q, QElemType& e)
{
    if(Q.rear == Q.front) return ERROR; // 队列空
}
```

```
e = Q.base[Q.front];
Q.front = (Q.front + 1) % MAXQSIZE;
return OK;
}
```

队列应用

离散时间模拟

Back_Simulation_Code:

```
git clone https://github.com/xinggangw/data_struct_teaching.git
```

CPU循环调度问题

- n 个任务，按顺序到来，每个人住需要执行 $t(ms)$
- CPU的一个时间片为 $q(ms)$
- 如果 $q(ms)$ 之后任务尚未处理完成，则该任务将被移动到队尾，CPU随即开始处理下一个任务

模拟CPU的循环调度

- 举个例子，假设 q 是100，然后有如下任务队列。
- A(150) -- B(80) -- C(200) -- D(200)
- 首先A被处理100 ms，然后带着剩余的50 ms移动至队尾
- B(80) -- C(200) -- D(200) -- A(50)
- 随后B被处理80 ms，在总计第180 ms时完成处理，从队列中消失
- C(200) -- D(200) -- A(50) \nearrow
- 接下来C被处理100 ms，然后带着剩余的100 ms移动至队尾。
- D(200) -- A(50) -- C(100)
- 之后同理，一直循环到处理完所有任务。
- 请编写一个程序，模拟循环调度法。

小结

第3章小结

线性表、栈、队的异同点：

相同点：逻辑结构相同，都是线性的；都可以用顺序存储或链表存储；栈和队列是两种特殊的线性表，即**受限的线性表**（只是对插入、删除运算加以限制）。

不同点： ① **运算规则不同：**

- ◆ 线性表为随机存取；
- ◆ 而栈是只允许在一端进行插入和删除运算，因而是后进先出表**LIFO**；
- ◆ 队列是只允许在一端进行插入、另一端进行删除运算，因而是先进先出表**FIFO**。

② **用途不同。** 线性表比较通用；堆栈用于函数调用、递归和简化设计等；队列用于离散事件模拟、OS作业调度和简化设计等。

