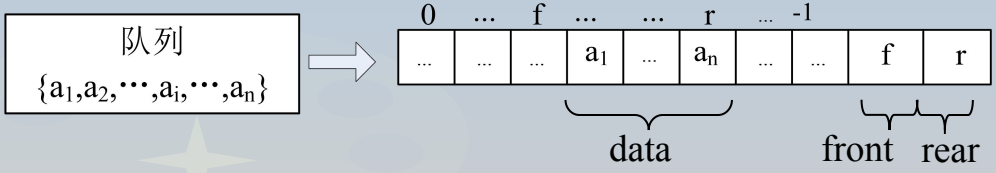
本节主题：

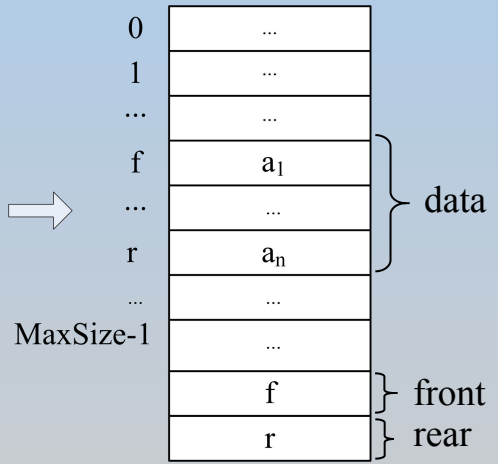
顺序队的存储及基本操作

队列的顺序存储结构

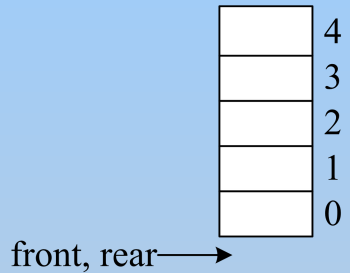
- 描述队列
  - 数据元素data：元素具有同一类型ElemType，最多MaxSize，
  - 当前队首front
  - 当前队尾rear

```
typedef struct
{
     ElemType data[MaxSize];
     int front, rear;//队首和队尾指针
} SqQueue;
```

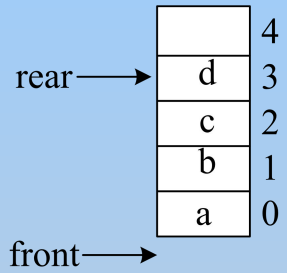




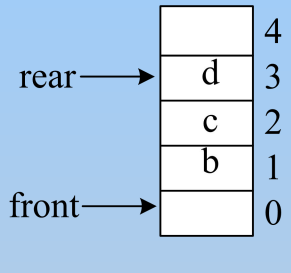
顺序队的四要素



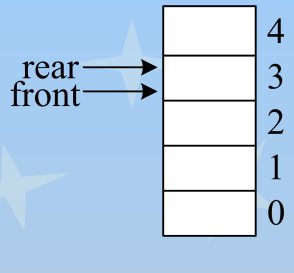
(1) 空队列



(2) a b c d入队



(3) 出队一次



(4) 再出队三次

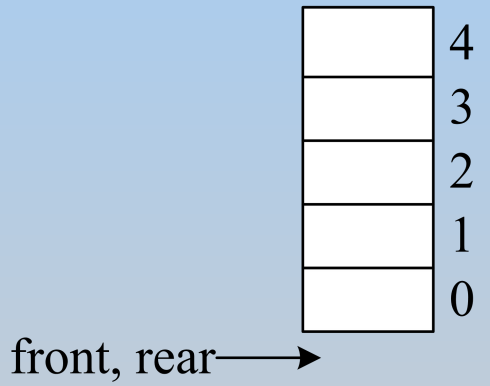
- 队空条件： `front=rear`
- 队满条件： `rear = MaxSize - 1`
- 元素e进队： `rear++; data[rear]=e;`
- 元素e出队： `front++; e=data[front];`

- rear指向队尾元素
- front指向队头元素的前一个位置。

初始化队列InitQueue(q)

- 构造一个空队列q。将front和rear指针均设置成初始状态即-1值。

```
void InitQueue(SqQueue *&q)
{
   q=(SqQueue *)malloc (sizeof(SqQueue));
   q->front=q->rear=-1;
}
```



销毁队列DestroyQueue(q)

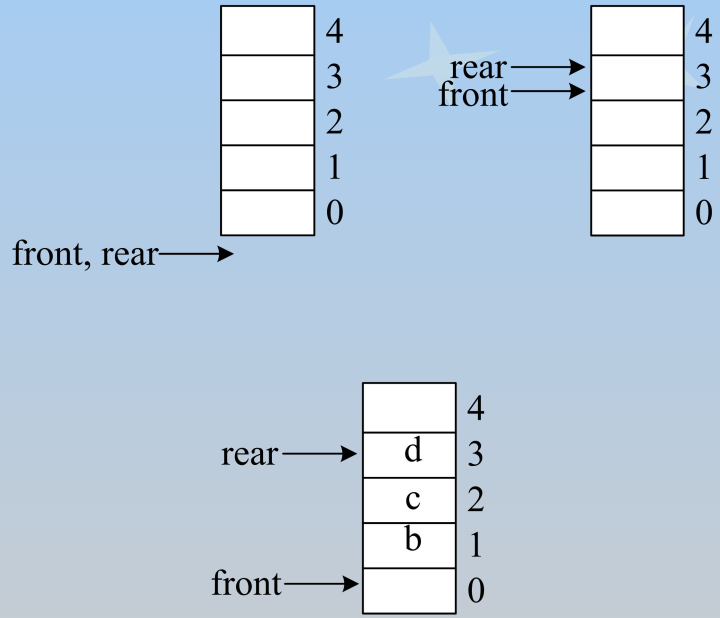
- 释放队列q占用的存储空间。

```
void DestroyQueue(SqQueue *&q)
{
   free(q);
}
```

判断队列是否为空QueueEmpty(q)

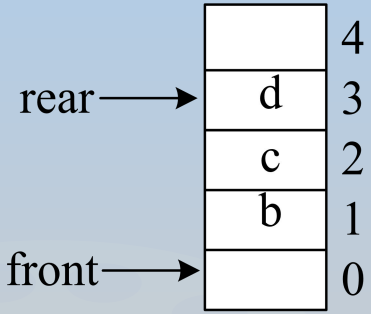
- 若队列q满足q->front==q->rear条件，则返回true；否则返回false。

```
bool QueueEmpty(SqQueue *q)
{
    return(q->front==q->rear);
}
```



进队列enQueue(q,e)

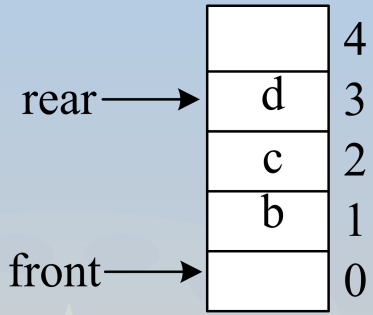
- 条件
  - 队列不满时
- 操作
  - 先将队尾指针rear循环增1
  - 然后将元素添加到该位置。




```
bool enQueue(SqQueue *&q,ElemType e)
{
     if (q->rear==MaxSize-1)
   return false;
     q->rear++;
     q->data[q->rear]=e;
     return true;
}
```

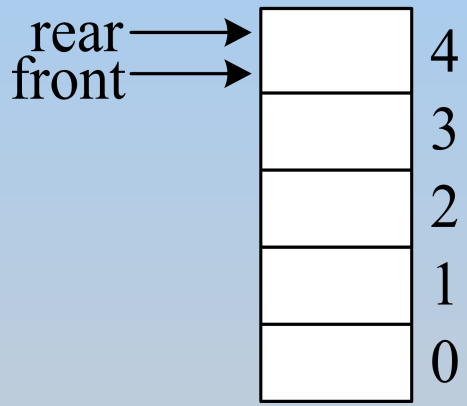
出队列deQueue(q,e)

- 条件
  - 队列q不为空
- 操作
  - 将队首指针front循环增1
  - 将该位置的元素值赋给e。




```
bool deQueue(SqQueue *&q,ElemType &e)
{
    if (q->front==q->rear)
        return false;
    q->front++;
   e=q->data[q->front];
   return true;
}
```

顺序队列的问题



队空？

队满？