# 队

## 队的定义

队列是只允许在一端进行插入操作，而在另一端进行删除操作的线性表。允许插入的一端称为队尾，允许删除的一端称为队头。队是一种先进先出的线性表，可以用数组和链表进行模拟。数据插入称为入队，删除称为出队。

## 循环队列

### 循环队列的定义

用数组模拟队列，数组下标为0的元素即为队头。引入两个指针front和rear，分别指向队头元素的位置和队尾元素的下一个位置。当front=rear时队列为空。元素入队时rear指针向后移动一位，出队时front指针向后移动一位。这样进行下去，rear和front最终会指向数组的最后一个位置。此时虽然队中已经没有元素了，但仍无法再令元素进队，这种现象为假溢出。

可以构造头尾相接的队列，解决假溢出问题。这种队列称为循环队列。rear指向队尾后，再有元素入队就令rear指向队头的位置，这样就可以循环利用空间。但是当数组中所有位置都填满元素时，rear=front，与刚才队列为空的判定条件一样了。如何区别这两种情况呢?

1.设置一个标志变量flag，rear=front且flag为0表示空，rear=front且flag为1表示满。

2.修改队列为满的条件：保留一个空元素，则(rear+1)%最大容量=front表示满，rear=front表示空。队列长度可用(rear-front+最大容量)%最大容量计算，范围0至最大长度-1。

```
typedef struct Q{
    int Data[MaxSize];//也可以只定义数组的头指针，之后用malloc分配内存。
    int front;//队头的位置
    int rear;//队尾的位置
}Queue;
```

front和rear初始值均为0。

### 入队

```
int EnQueue(Queue *Q,int InData){
    if((Q->rear+1)%MaxSize==Q->front){
        return 0;
    }
    Q->Data[Q->rear]=InData;
    Q->rear=(Q->rear+1)%MaxSize;
    return 1;
}
```

### 出队

```
int DeQueue(Queue *Q,int *OutData){
    if(Q->front==Q->rear){
        return 0;
    }
    *OutData=Q->Data[Q->front];
    Q->front=(Q->front+1)%MaxSize;
    return 1;
}
```

#### 求队列长度

```
int QueueLength(Queue *Q){
    return (Q->rear-Q->front+MaxSize)%MaxSize;
}
```

### 链式队列

只能从尾部进从头部出的单链表。队列的头指针指向链表的头结点，尾指针指向最后一个结点。队列为空时front和rear都指向头结点。

```
typedef struct QN{
    int Data;
    struct QN *pNext;
}QNode;//链表的结点

typedef struct Q{
    QNode *front,rear;//头指针，尾指针
}LinkQueue;
```

#### 入队

入队操作即为在链表尾端插入结点。

```
int EnQueue(LinkQueue *Q,int InData){
    QNode *Node=(QNode *)malloc(sizeof(QNode));
    if(Node==NULL){
        return 0;
    }
    Node->Data=InData;
    Node->pNext=NULL;
    Q->rear->pNext=Node;
    Q->rear=Node;
    return 1;
}
```

#### 出队

出队操作即为删除头结点的后置结点，头结点的指针域指向删除前的第二个结点。若删除后队为空，则rear指向头结点。

```
int DeQueue(LinkQueue *Q,int *OutData){
    if(Q->front==Q->rear){
        return 0;
    }
    QNode *Node=Q->front->pNext;
    *OutData=Node->Data;
    Q->front->pNext=Node->pNext;
    if(Node==Q->rear){
        Q->rear=Q->front;//若出队后队列为空，则rear指向头结点
    }
    free(Node);
    return 1;
}
```

**滑动窗口**

有一个长为 n 的序列 a，以及一个大小为 k 的窗口。现在这个从左边开始向右滑动，每次滑动一个单位，求出每次滑动后窗口中的最大值和最小值。

输入一共有两行，第一行有两个正整数 n,k。 第二行 n 个整数，表示序列 a。输出共两行，第一行为每次窗口滑动的最小值，第二行为每次窗口滑动的最大值。

对于 50% 的数据，1≤$n$≤10^5；对于 100% 的数据，1≤$k$≤$n$≤10^6，ai∈[−2^31,2^31)。

| Window Position | Min | Max |
|---|---|---|
| [1 3 -1] -3 5 3 6 7 | -1 | 3 |
| 1 [3 -1 -3] 5 3 6 7 | -3 | 3 |
| 1 3 [-1 -3 5] 3 6 7 | -3 | 5 |
| 1 3 -1 [-3 5 3] 6 7 | -3 | 5 |
| 1 3 -1 -3 [5 3 6] 7 | 3 | 6 |
| 1 3 -1 -3 5 [3 6 7] | 3 | 7 |

**单调队列**

单调队列内的元素满足单调递增/单调递减，既可以从队头弹出元素，也可以从队尾弹出元素。

以单调递增为例，若当前元素大于等于队尾元素，则直接入队，否则将队尾元素依次弹出，直到队尾元素小于等于当前元素。如果所有元素都被弹出，则当前元素作为新的最小值入队。

滑动窗口中队头弹出元素的规则为：窗口中元素个数大于k，即队尾元素序号-队头元素序号+1大于k时，将队头元素弹出。

所以对于单增队列，元素入队的条件有两个：

1.弹出队尾元素，直至队尾元素小于等于当前元素或队列为空；

2.弹出队头元素，直至队尾元素序号-队头元素序号+1小于等于k；

每次入队操作结束后，窗口最小值即为队头元素。

同理可以构建一个单减队列求出每次窗口的最大值。

解题思路：定义一个数组存储序列中的数值，数组下标代表这个数在序列中的位置。再按照上述思路构建单调队列，队列中存储数据的位置，每次后移时求出最大值和最小值即可。

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int *Data;//数组模拟队
int front,rear;//头指针和尾指针
int Num[1000005];//用于存储序列a

int main(){
    int n,k,i;
    scanf("%d%d",&n,&k);
    Data=(int *)malloc((k+1)*sizeof(int));
    for(i=0;i<n;i++){
        scanf("%d",&Num[i]);
    }
    Data[0]=0;
    front=0;
    rear=1;
    if(k==1){
        printf("%d ",Num[0]);
    }//为防止非法访问，后面的循环从第二个开始。但如果k=1，就需要输出第一个数。
    for(i=1;i<n;i++){
        while(Num[i]<Num[Data[(rear-1+k+1)%(k+1)]]){//队尾元素大于当前元素且队列不为空，依次弹出
            rear=(rear-1+k+1)%(k+1);
            if(rear==front){//队列为空则退出循环
                break;
            }
        }
        //队尾元素小于等于当前元素或队列为空就入队
        Data[rear]=i;
        rear=(rear+1)%(k+1);
        while((i-Data[front]+1)>k){//总个数大于k，队头出栈
            front=(front+1)%(k+1);
        }
        if(i>=k-1){
            printf("%d ",Num[Data[front]]);
        }
    }
    printf("\n");
    //同理求最大值
    memset(Data,0,sizeof(Data));
    front=0;
    rear=1;
    if(k==1){
        printf("%d ",Num[0]);
    }
    for(i=1;i<n;i++){
        while(Num[i]>Num[Data[(rear-1+k+1)%(k+1)]]){//队尾元素小于当前元素且队列不为空，依次弹出
            rear=(rear-1+k+1)%(k+1);
            if(rear==front){//队列为空则退出循环
                break;
            }
        }
        //队尾元素大于等于当前元素或队列为空就入队
        Data[rear]=i;
```

```
            rear=(rear+1)%(k+1);
            while((i-Data[front]+1)>k){//总个数大于k，队头出栈
                front=(front+1)%(k+1);
            }
            if(i>=k-1){
                printf("%d ",Num[Data[front]]);
            }
        }
        return 0;
}
```