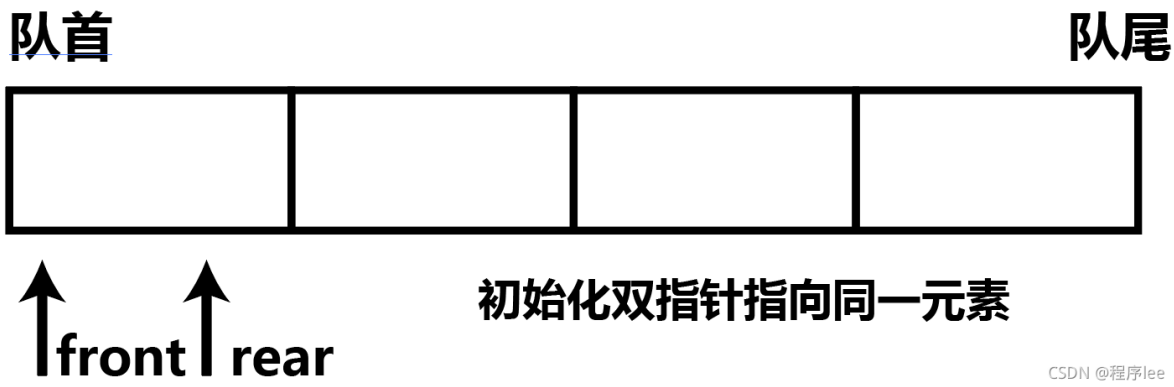
目录

队列与栈同样是一种操作受限制的线性表，队列的特点是先进先出即 FIFO，一般在尾部插入头部删除，在通常使用过程中，顺序队列经常产生假溢出等情况，因此时常采用**顺序循环队列。**

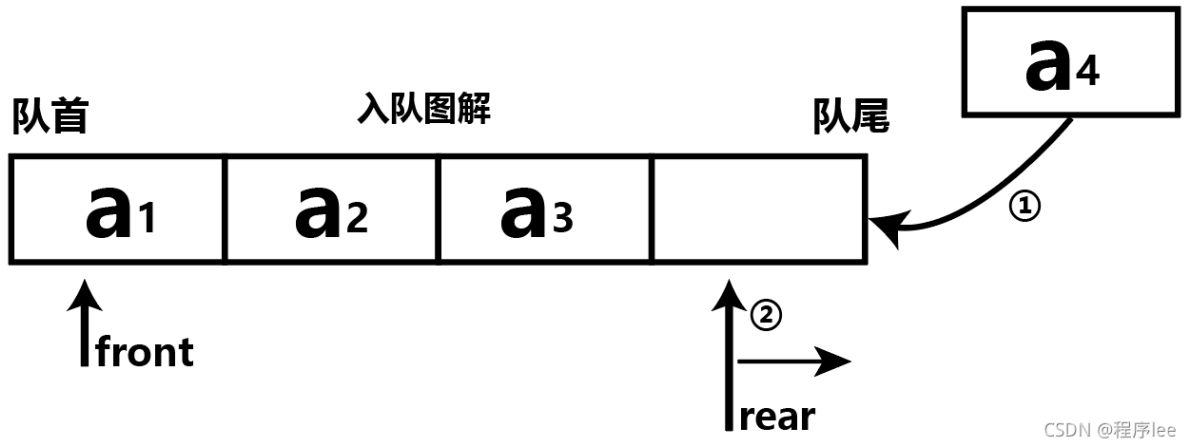
除顺序队列外还有链队，双端队列等。接下来先从顺序循环队列开始……

顺序队列

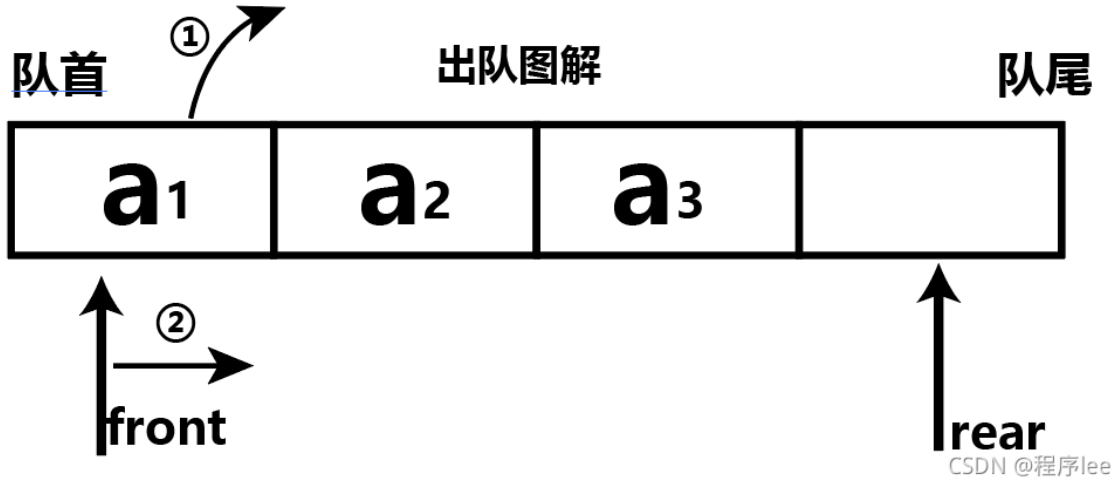
头尾指针与顺序栈是一个思路，本质其实是**数组的下标**，如图所示，起始双指针指向同一位置，因此也可以推导出队列**判空**的条件即**Q.front=Q.rear。**始终保持rear指向后继元素



接下来可以进行**入队**操作，即：先在rear位置插入元素，然后再将rear向后移动。始终保持rear指向后继空元素位置，入队过程中front指针是始终没有变化的

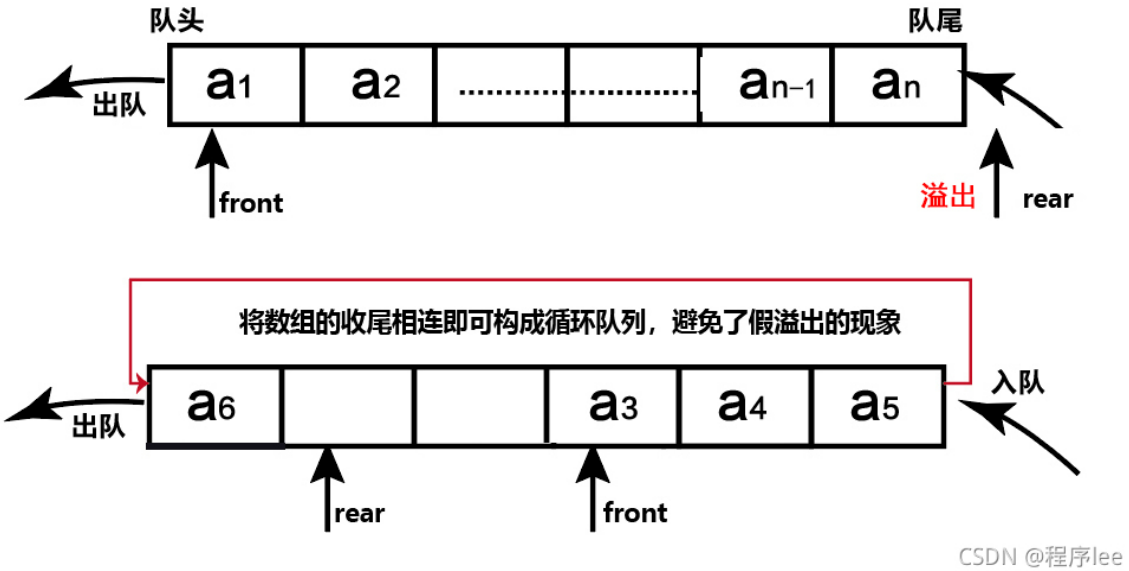


出队操作与之相反，因队列的特性。出队应在队首进行即：先将front位置的元素删除，再将front移动，出队过程中rear无变化如图所示



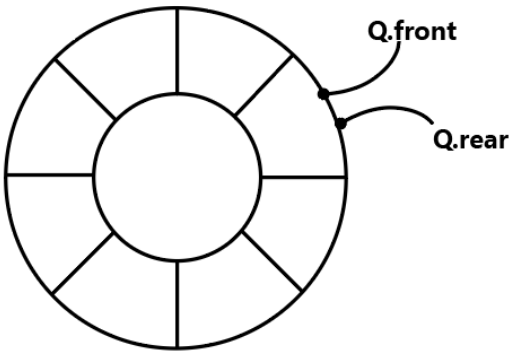
由下图可以看出单纯的顺序队列会产生假溢出的情况，即当**Q.rear=Q.MAX**时，已无法再入队，但队列前部分还有空缺。因此采用循环队列可解决此情况。

*

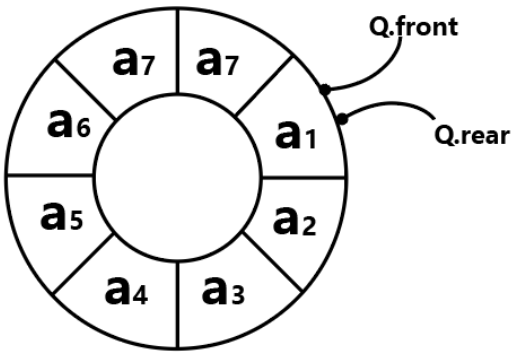


*

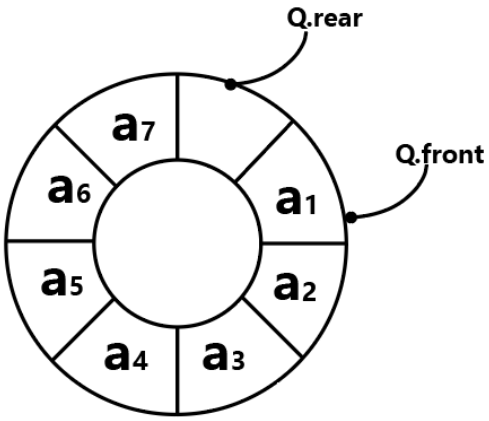
除改为循环队列外，还需要进行一些改进方便使用，即始终保留一空位。原因如下图，如不保留一块空位则会导致**队空队满**的条件皆为：**Q.front=Q.rear 不合理**，由此图也可看出，Q.rear指向当前尾结点的后继元素，而非当前尾结点



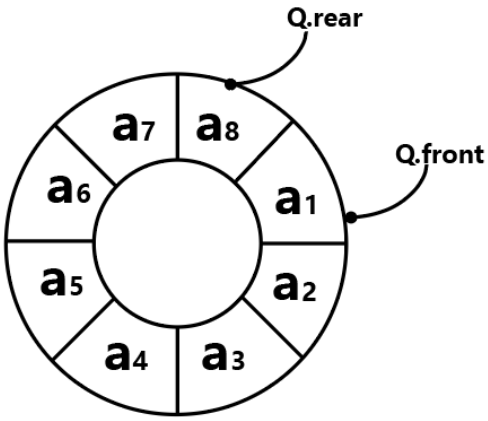
队空情况：Q.front=Q.rear



队满不合理情况：Q.front=Q.rear



队满合理情况：Q.front=（Q.rear+1）%MAXSIZE



如Q.rear指向尾结点，找不到队列空的条件，因此需将Q.rear指向后继节点

@程序lee

饼状表示图

顺序循环队列实现

一、求队列长度

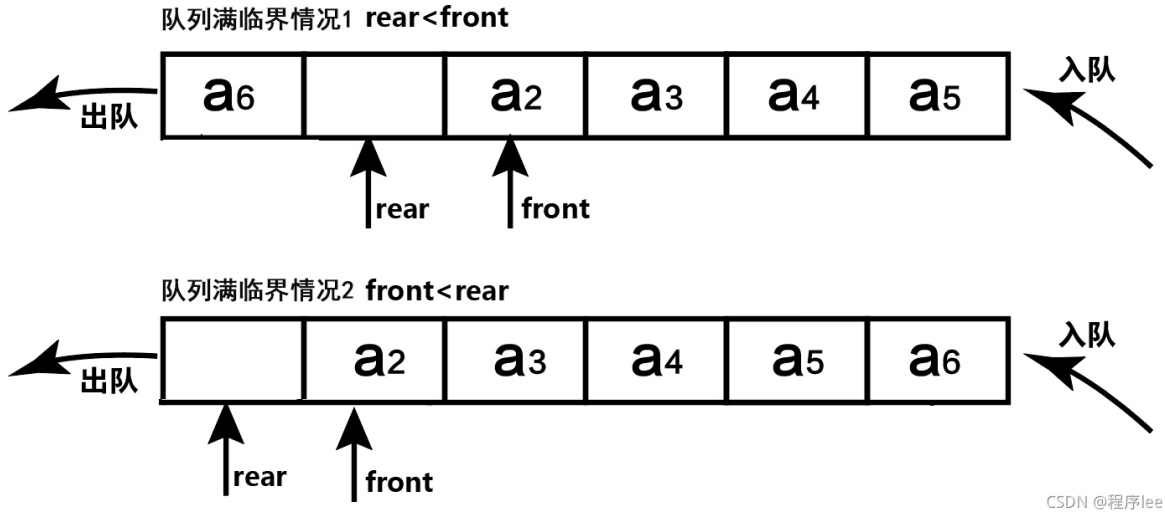
如上图所示，正常队列的队列长度为**front-rear**， 当变为顺序循环队列后，长度已经不再是**front-rear**， 经计算要满足两种情况的长度为：

```
#define MAXSIZE  6
（Q.rear-Q.front+MAXSIZE）%MAXSIZE     //MAXSIZE 为数组长度
```

二、队满

```
#define MAXSIZE  6
(Q.rear+1) % MAXSIZE = Q.front
```

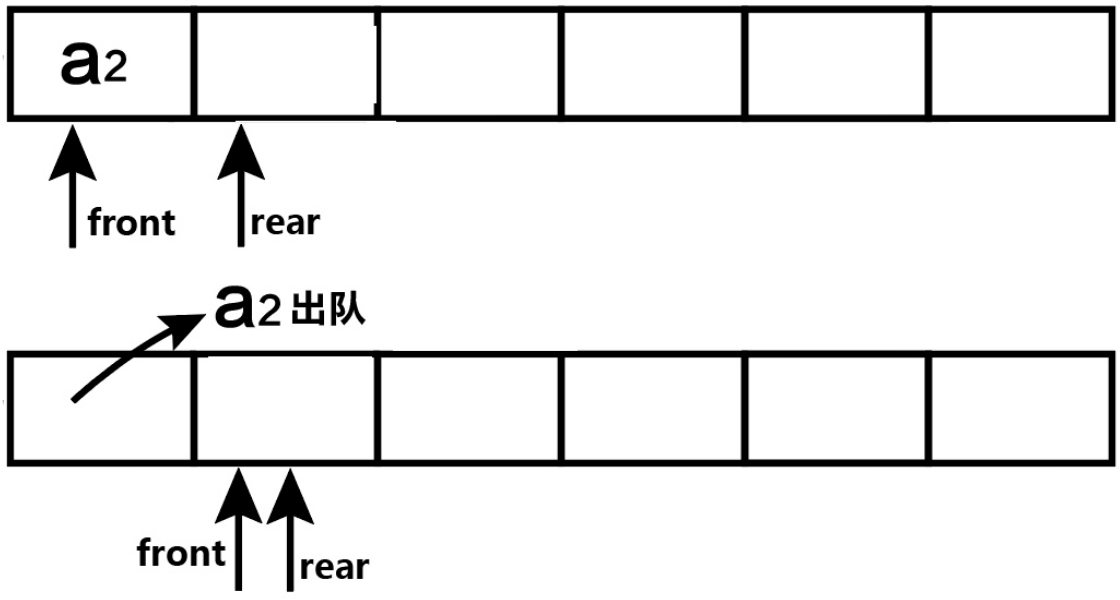
*



CSDN @程序lee

*三、队空

```
rear=front
```



CSDN @程序lee

创建循环队列

静态创建（不可扩容）：步骤：① front=rear=-1，初始化数组下标位置 ②rear+1 ③赋值

创建方法各不相同，基本合理即可，图解参照上图栈空情况反推即可

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 5
typedef struct LinkQue {
    int data[MAXSIZE];
    int front;      //头下标
    int rear;      // 尾下标
}LinkQue;
```

```
LinkQue CreatQue(LinkQue Q,int arr[]) {
    Q.front = Q.rear = 0;              //首尾下标为0
    for (int i = 0;i < MAXSIZE-1;i++) {  //循环队列有一个空位置
        Q.rear = Q.rear + 1;         //尾指针向后移动
        Q.data[i] = arr[i];            //赋值
        printf("%d\n",Q.data[i]);
    }
    return Q;
}
```

循环队列的入队

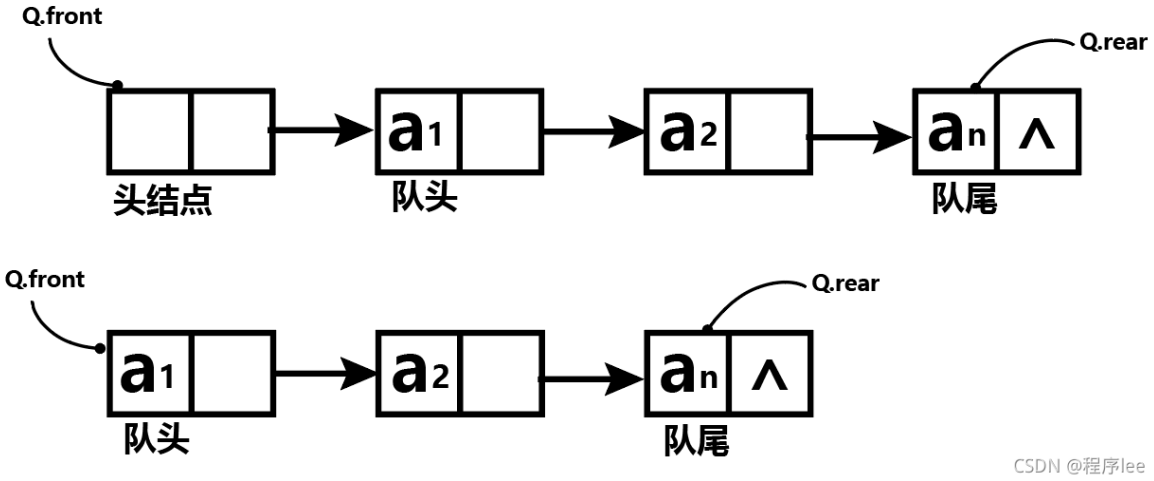
```
void AddQue(LinkQue* Q1, int e) {   //入队 e
    if ((Q1->rear + 1) % MAXSIZE == Q1->front)  //判断队满条件
        printf("xww");
    else {
        Q1->data[Q1->rear] = e;                //新元素加入队尾
        Q1->rear = (Q1->rear + 1) % MAXSIZE;   //队尾指针加一
}
```

循环队列的出队

```
void Delete(LinkQue*Q,int e) {   //入队 e
    if (Q.rear==Q.front)          //判断队空条件
    return false;
    else {
        e=Q.data[Q.front];
        Q.front=(Q.front+1)%MAXSIZE;   //指针+1
    }
}
```

创建链队

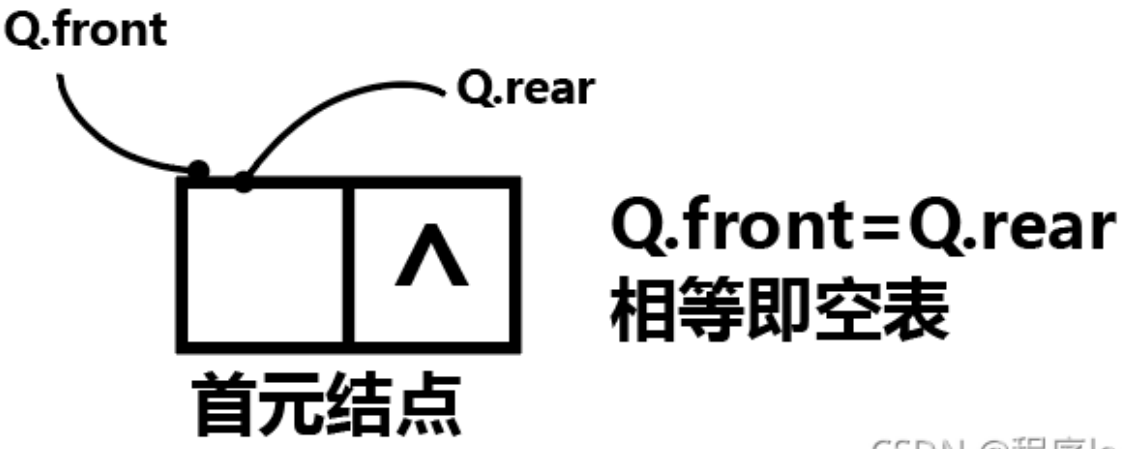
链式队列有两个指针分别指向头结点与尾结点，因队列只能头插尾删所以相对带头尾节点的链表简单很多。链队也分为带头结点与不带头结点两种。



CSDN @程序lee

链队为链式存储结构，因此与循环队列，顺序队列不同在于，可以无需考虑**队满**的情况，但仍需考虑空链表的情况。因此首先考虑链表为空的条件。

一、链队为空



CSDN @程序lee

二、链队初始化

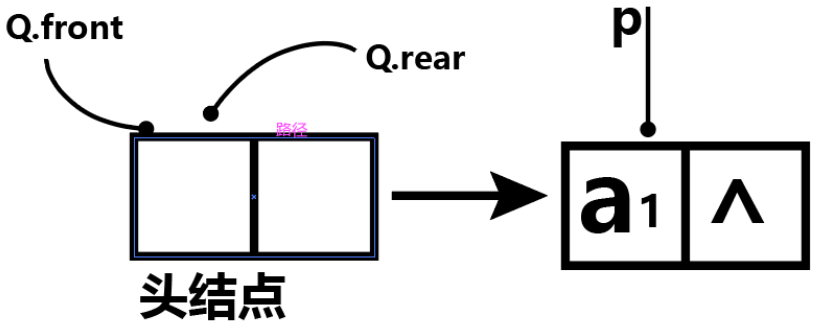
初始化即创建只有头结点的链队，头尾指针指向头结点。方便后期插入以建立链队。如上图

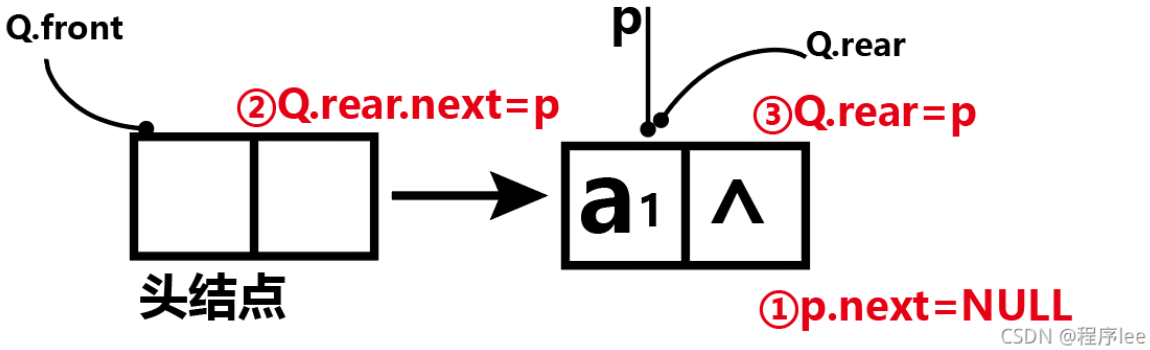
```
typedef struct Qnode {
    int data;                //存放指针front，rear
    QueuePtr next;           //与链表相似
}Qnode,*QueuePtr;
typedef struct {
    QueuePtr front;         //头指针
    QueuePtr rear;         //尾指针
}LinkQueue;
```

```
LinkQueue InitQueue(LinkQueue Q) {
    Q.front = Q.rear = (QueuePtr)maclloc(sizeof(Qnode));  //开辟空间并双指针指向
    Q.front->next = NULL;                                 //头节点为NULL
    return Q;
}
```

二、链队建立/入队

链队插入只能在队尾插入，每次插入的元素next域都为NULL，需要将尾指针Q.rear指向尾结点





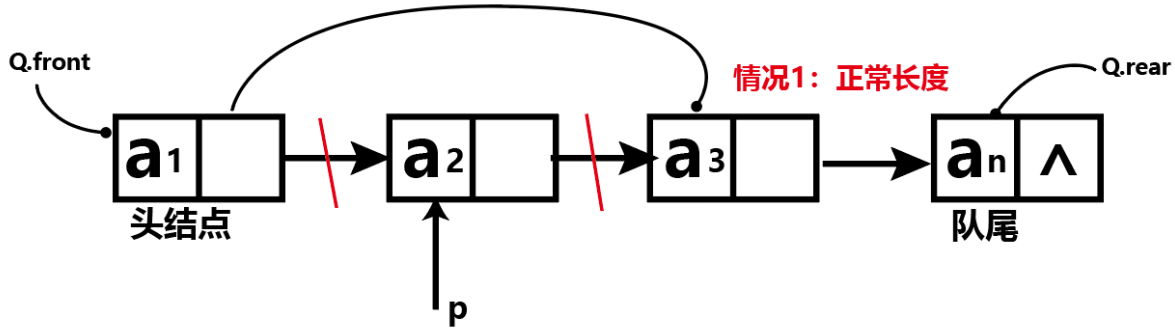
CSDN @程序lee

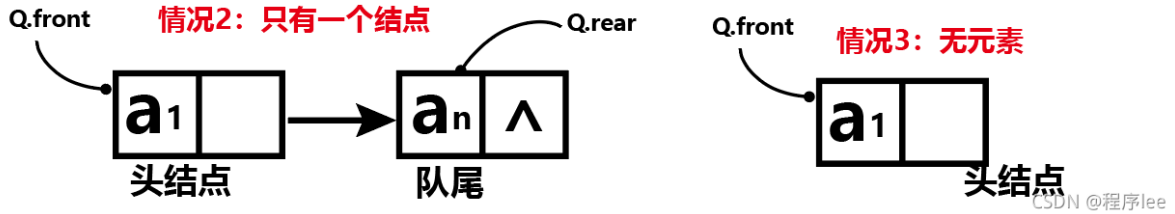
代码实现

```
LinkQueue CreatQueue(LinkQueue Q,int arr[]) {
    for (int i = 0;i < 5;i++) {
        QueuePtr p = (QueuePtr)malloc(sizeof(Qnode));  //开辟空间
        p->data[i] = arr[i];                //赋值
        p->next = NULL;                     //先将新建节点的next置空
        Q.rear->next= p;                    //将前一节点与p相连
        Q.rear = p;                         //尾指针 指向尾结点
    }
    return Q;
}
```

三、链队出队

出队除了要进行队空判断外还有一种特殊情况，即：只有头结点与首元结点





CSDN @程序lee

代码实现

```
void DeleteQueue(LinkQueue Q) {
    if (Q.front==Q.rear) {
        printf("空了");     //判断是否为空队
    }
    else {
        QueuePtr p = Q.front->next;    //p为首元结点 即要删除的节点
        int *e = p->data;               //存下要删除的地址
        Q.front->next = p->next;
        if (Q.rear==p) { Q.front = Q.rear; }  //如果只有尾结点一个元素
    }
}
```