第二章　栈和队列(2)

- 栈和队列也是线性表，只是操作受限的线性表。
- 由于操作上的限制，使他们的行为不同于一般的线性表，而有自己的特点。
  - 栈的定义
  - 栈的顺序存储与实现---顺序栈
  - 栈的链式存储与实现---链式栈
  - 队列的定义
  - 队列的顺序存储与实现
  - 队列的链式存储与实现
  - 栈与队列的应用

2.2 队列（Queue)

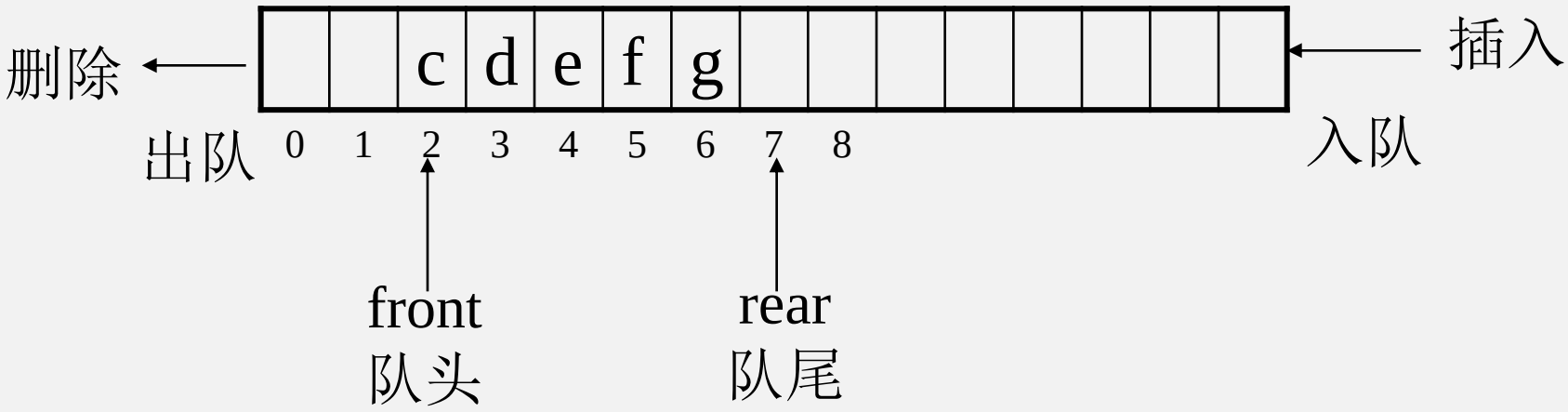
1、逻辑定义及特点

定义：只能在一端插入，在另一端删除的线性表。
（**操作限制**）

队尾：允许插入的一端称为**队尾**(rear)

队头：允许删除的另一端称为**队头**(front)

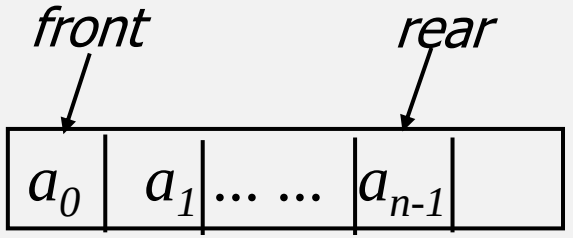
特点：**先进先出 FIFO**（First In First Out）



队列（Queue)

插入：入队、进队

删除：出队、出队



队列的基本操作：

1）创建一个空队列

2）入队

3）出队

4）取队头元素

5）判断队列空、队满、清空队列

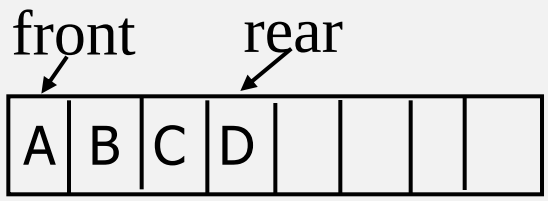
队列的存储实现方式：

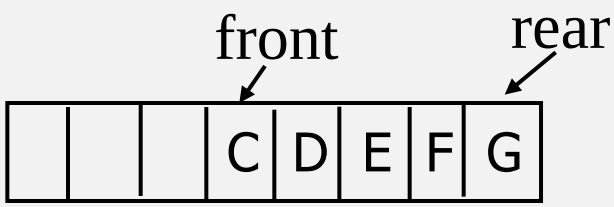
1、基于链式的队列----链式队列

2、基于数组的队列----循环队列（顺序队列）

队列的顺序存储实现(循环队列)

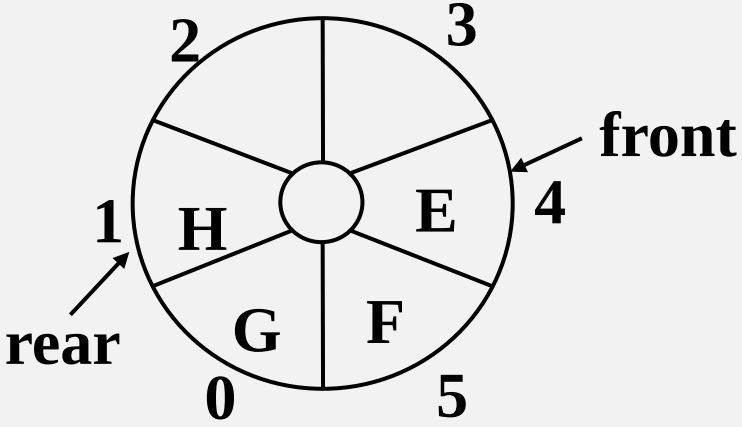
2、队列的顺序存储实现





队列满吗?

循环队列:



front向后移动:

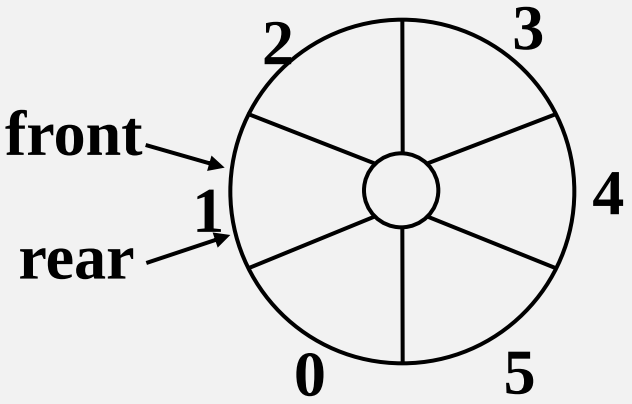
$$front = \begin{cases} front + 1 & front < size - 1 \\ 0 & front = size - 1 \end{cases}$$

$$front = (front + 1) \% size$$

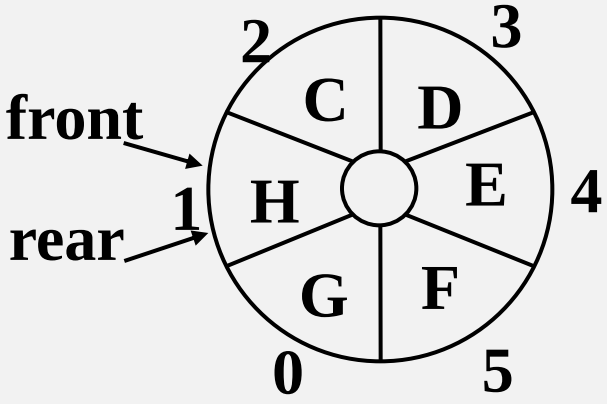
Rear向后移动:

$$rear = (rear + 1) \% size$$

队列的顺序存储实现(循环队列)



队空：front =1，rear =1

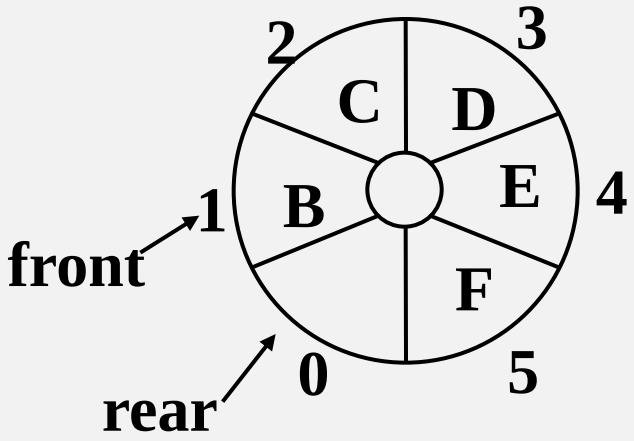


队满：front =1，rear =1

如何区分队空或队满呢?

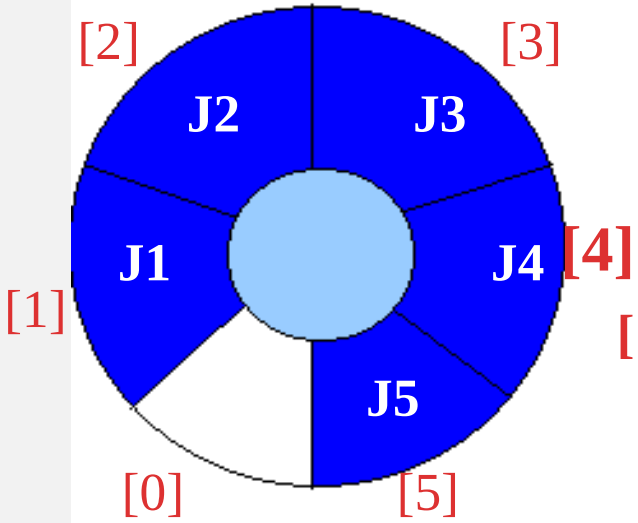
a) 设置标志变量记录队空或队满；

b) 分配存储空间=最大元素+1，
让front元素之前空闲一个单元。



队列的顺序存储实现(循环队列)

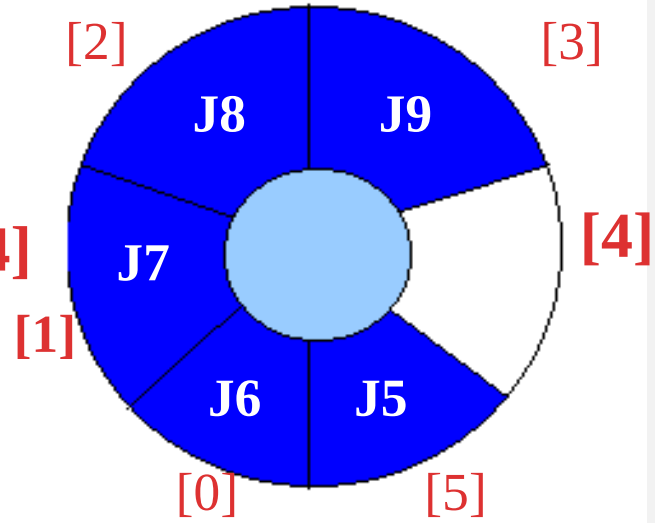
队列满



front =1

rear = 0

队列满



front =5

rear =4

队列为空的条件： rear == front

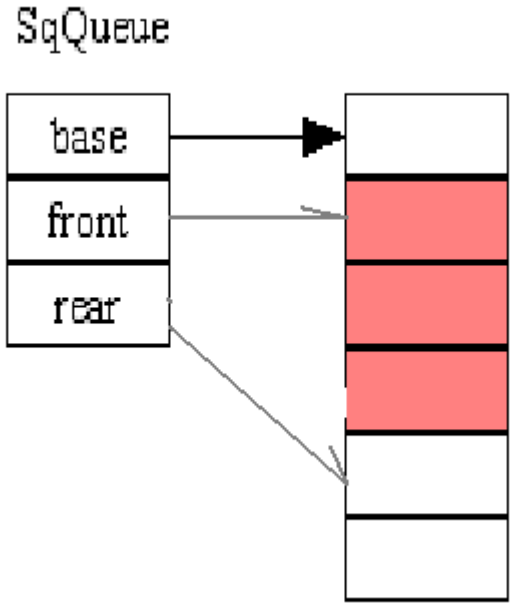
队列未满的条件： (rear+1) % size == front

队列的顺序存储实现(循环队列)

（1）循环队列定义（顺序存储结构上的C语言实现）

- 队头、队尾和队列存储位置要指示出来

```
#define  DefaultSize   30;
Typedef int ElemType;
Typedef struct {
    ElemType  *Elem；//队列数组存区指针
    int  Raer, Front;     //队列的尾指针、头指针
    int  MaxSize;  //队列存储区大小
} SeqQueue;
SeqQueue  SQueue;  //定义队列变量
```



栈的顺序存储实现(顺序栈)

（2）、创建一个空队列（队列的初始化）

```
bool  Init_SeqQueue(SeqQueue &Q, int Qsize=DefaultSize) // 队变量，队区大小
{   // 动态分配队列的顺序存储空间
    Q.Elem = ( ElemType *)malloc( (Qsize+1)*sizeof(ElemType) );
    if (Q.Elem == NULL) return FALSE;      //队列存储区分配失败
    Q.Front =  Q.Rear = 0;                 // 队尾、队头指示器为0
    Q.MaxSize = Qsize+1;                   //队列最大空间元素数
    return TURE;
}
```

调用方式： Init_SeqQueue(SQueue, 50)　其中：SQueue为队列变量

算法复杂度为： O(1)

（3）、判断队列为空

```
bool  Empty_SeqQueue(SeqQueue *Q)        // 队列变量指针
{   if (Q->Front == Q->Raer) return TURE;  //队列空，返回真值
    else  return  FALSE;                   // 队列非空，返回假
}
```

调用方式： Empty_SeqQueue (&SQueue) ;　　算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（4）、判断队列满

```
bool  Full_SeqQueue(SeqQueue *Q)          // 队列变量指针
{   if ( (Q->Rear+1)%Q->MaxSize == Q->Front)  return TURE;//队满，返回真值
    else  return  FALSE;                  // 队未满，返回假
}
```

调用方式： Full_SeqQueue(&SQueue) ;

算法复杂度为： O(1)

（5）、返回队列大小（当前元素个数）

```
int  Size_SeqQueue(SeqQueue *Q) // 队列变量指针
{   //返回栈中现有的元素个数
    return  (Q->Rear - Q->Front + Q->MaxSize)%Q->MaxSize ;
}
```

调用方式： Size_SeqQueue (&SQueue) ;

算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（6）、取队头元素值（取队头---GetFront操作）

```
bool  GetFront_SeqQueue(SeqQueue *Q, ElemType &x){ //队变量指针，元素变量
{
    if ( Empty_SeqQueue(Q) ) return FALSE;          //返回取队头失败
    else {
          x = Q->Elem[Q->Front];  //非空队列，取队头元素值
          return TURE;                //返回成功
    }
}
```

调用方式： GetFront_SeqQueue(&SQueue, y) 其中：y为元素变量

算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（7）、入队（进队列---InQueue操作）

```
bool  In_SeqQueue(SeqQueue *Q, ElemType x) // 队列变量指针，待存元素
{
    if ( Full_SeqQueue(Q) ) return FALSE;    //队列已满，入队操作失败
    else {
            Q->Elem[Q->Rear] = x ;                          //在队尾空位存入元素
            Q->Rear = (Q->Rear+1)%Q->MaxSize;       // 调整队尾位置
            return TURE;                                        //返回成功
    }
}
```

调用方式： In_SeqQueue(&SQueue, 55)　其中：Squeue为队列变量

算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（8）、出队（离开队列---OutQueue操作）

```
bool  Out_SeqQueue(SeqQueue *Q, ElemType &x) //队列变量指针，元素变量
{
    if ( Empty_SeqQueue(Q) ) return FALSE;          //队空，操作失败
    else {
          x = Q->Elem[Q->Front];   //取出栈顶元素
          Q->Front = (Q->Front+1)%Q->MaxSize;     // 调整队头位置
          return TURE;                    //返回成功
    }
}
```

调用方式： Out_SeqQueue(&SQueue, y) 其中:Squeue为队变量,y为元素变量

算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（9）、清空队列（清除现有元素，回复空队列）

```
bool  Clear_SeqQueue(SeqQueue *Q)    // 队列变量指针
{  Q->Front = Q->Rear = 0;    //设置队头、队尾为0
   return  TURE;                 // 返回清空成功
}
```

调用方式： Clear_SeqQueue(&SQueue) ;

算法复杂度为： O(1)

（10）、删除队列（清除元素，并释放空间）

```
bool  Delete_SeqQueue(SeqQueue *Q)  // 队列变量指针
{   Q->Front = Q->Rear = 0;  //清除元素
   free(Q->Elem);      //释放队列存储空间
   return  TURE;                 //返回成功
}
```

调用方式： Delete_SeqQueue(&SQueue) ;

算法复杂度为： O(1)

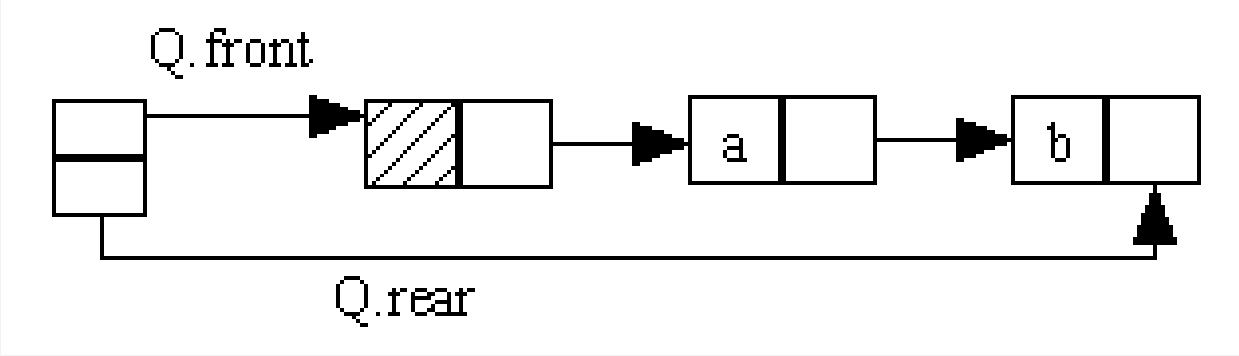
队列的链式存储实现(链队列)

3、队列的链式存储实现

（1）链队列程序定义（链式存储结构上的C语言实现）

- 队头、队尾和队列大小要指示出来

```
Typedef struct {   //其中：Node为单链表结点结构类型
   Node *Front;    //队头指针，指向队列第一个元素。
   Node *Rear;     // 队尾指针，指向队列最后一个元素
   int  QSize;     //队列当前元素个数
} LinkQueue;       //队列类型
LinkQueue  LQueue;  //定义队列类型变量
```



队列的链式存储实现(链队列)

（2）、创建一个空队列（队列的初始化）

```
bool  Init_LinkQueue(LinkQueue *Q ) // 链式队列变量指针
{
    Q->Front = Q->Rear = NULL;  //队头、队尾指针置为空
    Q->QSize =  0;                          // 填写大小计数初值为0
    return TURE;
}
```

调用方式： Init_LinkQueue(&LQueue)　其中：LQueue为链式的队列变量

算法复杂度为： O(1)

（3）、判断空队列

```
bool  Empty_LinkQueue(LinkQueue *Q)     // 队列变量指针
{   if (Q->Front == NULL) return TURE;     //队列为空，返回真值
    else  return  FALSE;                          // 队列非空，返回假
}
```

调用方式： Empty_LinkQueue(&LQueue) ;

算法复杂度为： O(1)

队列的链式存储实现(链队列)

（4）、返回队列大小

```
int  Size_LinkQueue(LinkQueue *Q)    // 队列变量指针
{  return  Q->QSize;  //返回队列中现有的元素个数
}
```

调用方式： Size_LinkQueue(&LQueue) ;

算法复杂度为： O(1)

（5）、取队头元素值（取队头---GetFront操作）

```
bool  GetFront_LinkQueue(LinkQueue *Q, ElemType &x) // 队变量，元素变量
{   if ( Empty_LinkQueue(Q) ) return FALSE;          //返回失败
    else  x = Q->Front->data;          //非空队列，取队头元素的值
    return TURE;                       //返回成功
}
```

调用方式： GetFront_LinkQueue(&LQueue, y)

其中：Lqueue为队变量，y为元素变量

算法复杂度为： O(1)

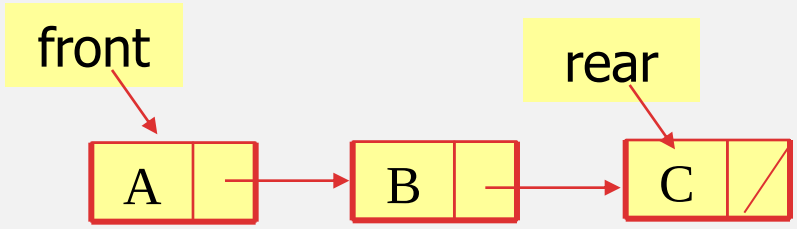
队列的链式存储实现(链队列)

（6）、入队（进队列---InQueue操作）

```
bool In_LinkQueue(LinkQueue *Q, ElemType x) //队列变量指针，待存元素
{  Node *p;
   p = (Node *)malloc( sizeof(Node) );
   if ( !p ) return FALSE;      //申请分配结点失败，操作失败
   p->data = x;                 //在新节点存入元素
   if  (Empty_LinkQueue(Q)) {p->next=NULL; Q->Front=Q->Raer=p;}
   else {p->next = Q->Front; Q->Front = p; }
   Q->QSize++;                  // 调整队列大小
   return TURE;                 //返回成功
}
```

调用方式： In_LinkQueue(&LQueue, 75)　　其中：LQueue为栈变量

算法复杂度为： O(1)



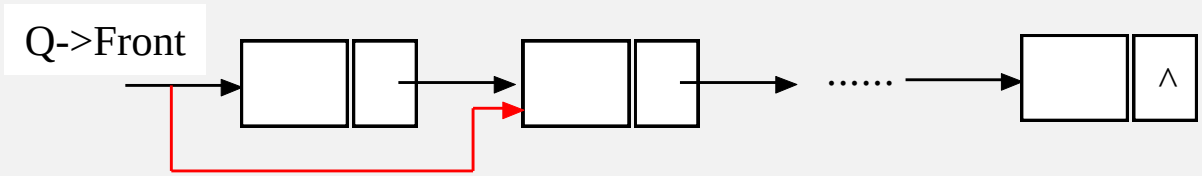
队列的链式存储实现(链队列)

（7）、出队（离开队列---OutQueue操作）

```
bool  Out_LinkQueue(LinkQueue *S, ElemType &x) // 队变量指针，待删元素值
{   Node *p;
    if ( Empty_LinkQueue(Q) ) return FALSE;          //操作失败
    p = Q->Front;              //将队头元素结点摘除
    Q->Front = p->next;
    if (p ==Q->Rear) Q->Rear = NULL;
    x = p->data;                //取出队头元素
    free( p ) ;                 //释放元素结点
    Q->QSize--;                 // 调整队列大小
    return TURE;                //返回成功
}
```

调用方式： Out_LinkQueue(&LQueue, y)　其中：Lqueue为队列变量

算法复杂度为： O(1)



队列的链式存储实现(链队列)

（8）、清空队列（清除现有元素，释放结点）

```
bool  Clear_LinkQueue(LinkQueue *Q)       // 队列变量指针
{   Node *p, *q;
    p=Q->Front;                 //让p指向队头
    while(p){                   //链不空
          q = p;                //保存待删除结点
          p = p->next;          //p指针后移
          free(q);              //释放被删除结点空间
    }
    Q->Front = Q->Rear = NULL;  //设置队头、队尾指针为NULL
    Q->QSize = 0;
    return  TURE;               // 返回清空成功
}
```

调用方式： Clear_LinkQueue(&LQueue) ;
算法复杂度为： O(1)

队列的应用(杨辉三角形)

4、队列应用（杨辉三角形，输出二项式系数）

输出$(a+b)^i$的各次幂系数：

i=1　　1　　1

2　　1　　2　　1

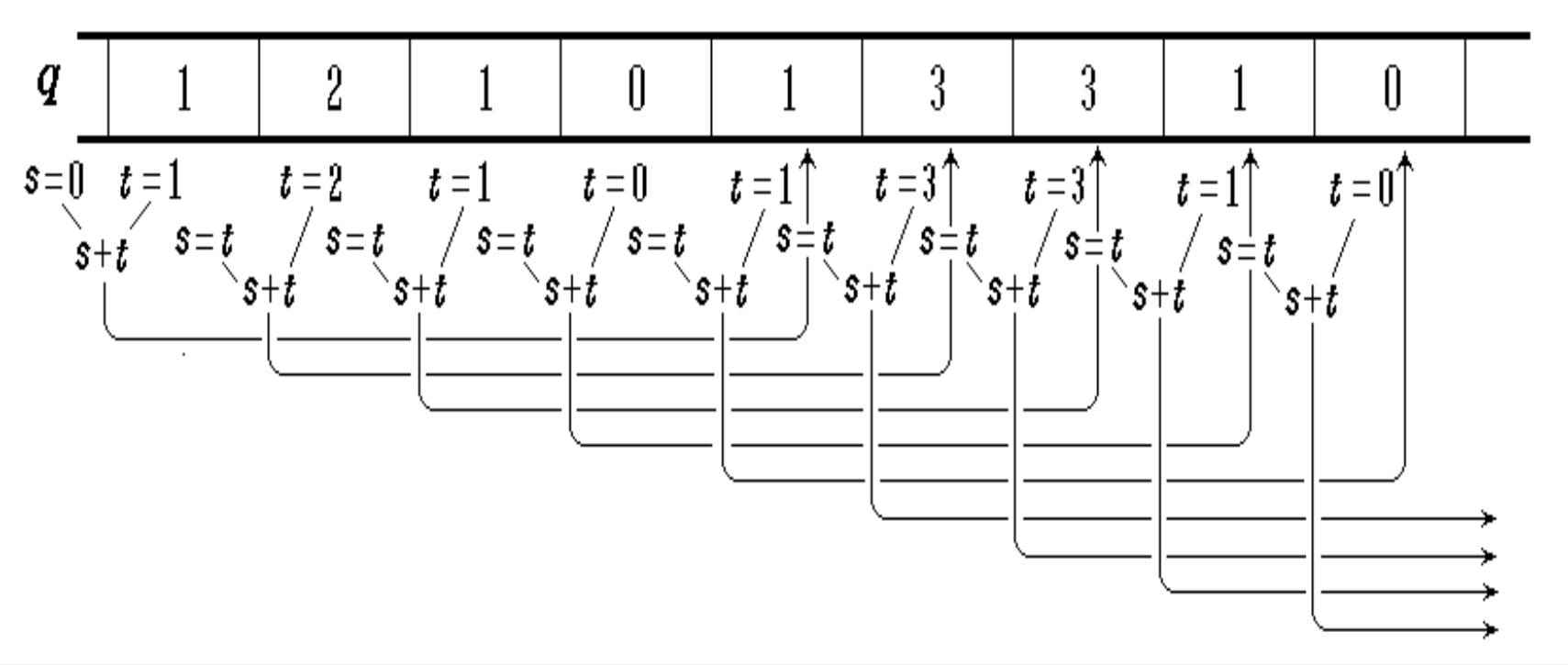
3　　1　　3　　3　　1

4　　1　　4　　6　　4　　1

5　　1　　5　　10　　10　　5　　1

…

队列的应用(杨辉三角形)



队列的应用(杨辉三角形)

```
void YANGVI ( int n ) {
    SeqQueue  q;
    Init_SeqQueue(q, n+2);
    In_SeqQueue (&q, 1);    In_SeqQueue(&q, 1);
    int s = 0;
    for ( int i=1; i<=n; i++ ) {
        printf("\n");
        In_SeqQueue(&q, 0);
        for ( int j=1; j<=i+2; j++ )  {
            int t;
            Out_SeqQueue(&q, t );
            In_SeqQueue(&q, s + t );
            s = t;
            if ( j != i+2 ) Printf("\d  ", s);
        }
    }
}
```

栈与队列的总结

1、 栈

- 栈的概念、特性：先进后出
- 栈的顺序实现、栈的链表实现
- 栈满及栈空条件
- 栈的应用：主要利用栈的特性（数值转换、表达值求值等）

2、 队列

- 队列的概念、特性：先进先出
- 队列的数组实现：
  - 循环队列中队头与队尾指针的表示，队满及队空条件，队头和队尾指针的移动
- 队列的应用：利用队列的特性

Thank you