**数据结构笔记**

# 第一章——绪论

## 1.1——数据结构基本概念

1. **数据：**描述客观事物属性的数，字符及所有能输入到计算机中并被计算机程序识别和处理的符号的集合（比较宏观，比较笼统）
2. **数据元素：**数据元素是数据的基本单位
3. **数据项：**数据项是构成数据元素不可再分隔的最小单位
4. **数据对象：**数据对象是具有相同性质的数据元素的集合
5. **数据类型：**数据类型是一个值的集合和定义在此集合上的一组操作的总称
6. **数据结构：**数据结构是相互之间存在一种或多种特定关系的数据元素的集合，包括三部分：①逻辑结构②存储结构③数据的运算

## 1.2——数据结构三要素

### 1.2.1——逻辑结构

1. 定义：数据元素之间的逻辑关系
2. 分类：数据结构分为线性结构和非线性结构

### 1.2.2——存储结构

1. 定义：数据结构在计算机中的表示，也成为物理结构
2. 分类：①顺序结构②链式存储③索引存储④散列存储

### 1.2.3——数据的运算

施加在数据上的运算包括运算的定义和实现

## 1.2——算法

### 1.2.1——算法的基本概念

* **算法的重要 特性：**

①有穷性②确定性③可行性④输入⑤输出

* 算法的效率度量

①时间复杂度：**T(n)=O(f(n))**

②空间复杂度: **S(n) = O(g(n))**

# 第二章——线性表

## 2.1——线性表的顺序表示

### 2.1.1——顺序表的定义

### 2.1.2——线性表的存储结构

——数据域和顺序表的当前长度55555555555555555555555

#define MaxSize 50 //定义线性表的最大长度

typedef struct{

Elemtype data[MaxSize]；//顺序表的元素

int length； //顺序表的当前长度

}**SqList** //顺序表的类型定义

### 2.1.3.——初始化顺序表

构造一个空的顺序表

Status InitiaList(SqList &L)

{

L.elem=new ElemType[MAXSIZE]；

if(!L.elem) exit(OVERFLOW)

L.length=0；

return ok；

}

### 2.1.4.——其他操作

**插入操作、删除操作、查找元素操作**

* **每插入一个元素，表的长度要记得加1**
* **每删除一个元素，表的长度要记得减一**

### 2.1.5——结构特点

* 可随机访问，存储密度高，插入和删除元素操作需要移动大量元素
* 动态分配属于顺序存储结构

## 2.2——线性表的链式表示

### 2.2.1——单链表的存储结构

1. **单链表中结点类型的存储结构**

**typedef struct LNode{ //定义单链表的结点类型**

**ElemType data； // 数据域**

**struct LNode \*next； //指针域**

**}LNode，\*LinkList //两个类型名**

1. **头结点，头指针，首元结点**

* 头节点：在首元结点之前的一个结点，指向首元结点
* 首元结点：存储第一个数据元素的节点
* 头指针：指向链表中第一个节点的指针

### 2.2.2——单链表的初始化

Status Initialist(LinKlist &L)

{

L=new LNode；

L->next=NULL； //头节点的指针置为空

return ok；

}

### 2.2.3——单链表上的其他操作

头插法建立单链表，尾插法建立单链表，按序号查找结点值，按值查找表结点，插入结点操作，删除结点操作，求表长操作，

* 删除结点操作

先用指针保存要删除结点的地址，然后才能去销毁它

### 2.2.4——双链表

定义：双链表结点中有两个指针prior和next 分别指向其前驱结点和后驱结点

### 2.2.5——循环链表

* **循环单链表：**表中最后一个结点的指针不是NULL，而改为指向头结点
* **循环单链表特点：**

**①循环单链表的判空条件不是头结点的指针是否为空，而是是否等于头指针②循环单链表可以从表中任意一个结点开始遍历整个链表③有时对单链表常做的操作是在表头和表尾进行的，此时对循环单链表不设头指针而仅设尾指针**

* **循环双链表：头结点的prior指针指向表尾结点，表尾的尾指针指向头结点**

# 第三章——栈、队列和数组

## 3.1——栈

### 3.1.1——栈的定义

* 栈是只允许在一端进行插入或删除操作的线性表。限定这种 线性表只能在一端进行插入和删除操作
* 由栈顶和栈底组成
* 空栈：不含任何元素的空表
* 栈的数学性质：n个不同元素进栈，出栈元素不同排列的个数为： ，称为卡特兰数

### 3.1.2——栈的顺序存储结构

#define MaxSize 50 //定义栈中元素的最大个数

typedef struct{

Elemtype data[MaxSize]; //存放栈中元素

int top； //栈顶指针

**} SqStack**

* **栈顶指针;S.top 栈顶元素：S.data[S. top]**
* **栈空条件：S. top==S.base(有的教材也会设置S.top==0)**
* **栈满条件：S.top==MaxSize-1**
* **栈非空时，top指针始终指向栈顶元素的上一个元素**

### 3.1.3——顺序栈的初始化

**Status Initialist(SqStack &S)**

**{ //构造一个空栈S**

**S.base==new SELemType[MAXSIZE]；**

**if(!S.base) exit(OVERFLOW)**

**S.top=S.base；**

**S.stacksize=MAXSIZE；**

**return OK；**

### 3.1.4——栈的其他操作

入栈、出栈、取栈顶元素、销毁栈

* **分配内存的时候要注意是否分配成功**
* **入栈的时候要检查是否栈满**
* **出栈的时候要检查是否栈空**

### 3.1.5——链栈的存储结构

typedef struct StackNode

{

ElemType data； //数据域

struct StackNode \*next； //指针域

}StackNode，\*LinkStack;

### 3.1.6——链栈的初始化

Status InitStack(LinkStack &S)

**{**

**S=NULL;**

**return OK;**

**}**

### 3.1.7——链栈的其他操作

入栈、出栈、栈与递归、取栈顶元素

#### 3.1.7.1.——入栈

采用头插法或尾插法

//2. 链式栈入栈操作；

int stack\_push(linkStack \*s, data\_t val)

{

**//先判断栈空不空,如果空的，先创建第一个结点：**

if (\*s == NULL)

{

\*s = (linkStack)malloc(sizeof(stackNode));

if (\*s == NULL)

{

return -1;

}

(\*s)->data = val;

(\*s)->next = NULL;

return 0;

}

**//不空就在链表头前面加上一个结点**

linkStack p = (linkStack)malloc(sizeof(stackNode));

if (p == NULL)

{

return -1;

}

p->next = \*s;

p->data = val;

\*s = p;

return 0;

}

### 3.2.1——队列的定义

**受限线性表，只允许在表的一段进行插入，而在表的另一端进行元素删除**

### 3.2.2——队列的顺序存储结构

#define MaxSize 50 //定义队列中元素的最大个数

typedef struct{

ElemType data[MaxSize]； //存放队列元素

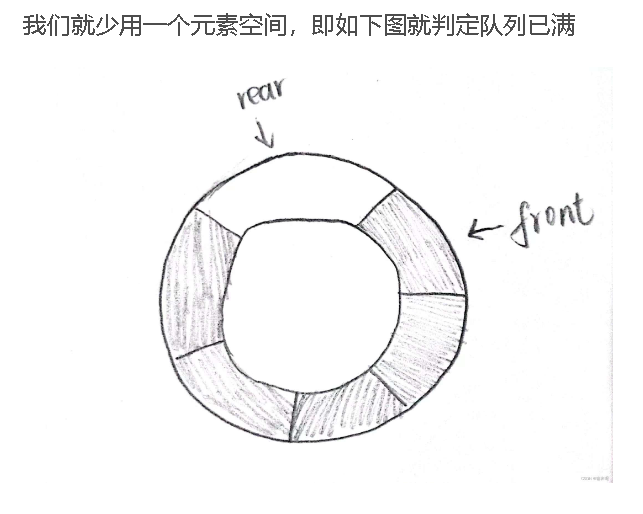
int front , int rear； //队头指针和队尾指针

}SqQueue

* **头指针始终指向队头元素**
* **尾指针始终指向队尾元素的下一个位置**

### 3.2.3——循环队列

* 队空的条件：Q.front==Q.rear
* 队满的条件：**(Q.rear+1)%MAXQSIZE==Q.front**



#### 循环队列的其他操作

1. struct SeqQueue {
2. T\* data; // 指向数据元素数组的指针
3. int front; // 下一个出队元素的数组下标
4. int rear; // 下一个入队元素应该存放的单元的数组下标
5. int max; // 队列中最多可放max-1个数据元素，留一个空数据单元以区分空和满
6. };

##### 1.判空

bool SQ\_IsEmpty(SeqQueue\* sq)

// 判断队列是否为空，为空返回true，否则返回false。

{

    // 请在Begin-End之间补充代码，完成队列是否为空的判断。

    /\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

    if(sq->front==sq->rear) return true;

    else return false;

    /\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

##### 2.判断是否满

bool SQ\_IsFull(SeqQueue\* sq)

// 判断队列是否为满。为满返回true,否则返回false。

{

    // 请在Begin-End之间补充代码，完成队列是否为满的判断。

    /\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

    if((sq->rear+1)%(sq->max)==sq->front) return true;

    else return false;

    /\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

##### 求循环队列长度

int SQ\_Length(SeqQueue\* sq)

// 队列长度。

{

    // 请在Begin-End之间补充代码，获取队列长度。

    /\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

    return (sq->rear+(sq->max)-sq->front)%(sq->max);

    /\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

##### 入队

bool SQ\_In(SeqQueue\* sq, T x)

// 将x入队。若入队失败(队列满)，则返回false，否则返回true。

{

    // 请在Begin-End之间补充代码，将 x 入队。

    /\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

    if(SQ\_IsFull(sq)) return false;

    else{

        sq->data[sq->rear++]=x;

        sq->rear=sq->rear%(sq->max);

    }

    /\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

##### 出队

bool SQ\_Out(SeqQueue\* sq, T& item)

// 从队列sq出队一个元素，返回时item为出队的元素的值。若出队成功(队列不为空)，则返回true，否则(队列空)，返回false，此时item不会返回有效值。

{

    // 请在Begin-End之间补充代码，完成元素出队操作。

    /\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

    if(SQ\_IsEmpty(sq)) return false;

    else{

        item=sq->data[sq->front++];

        sq->front=sq->front%(sq->max);

    }

    /\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

##### 打印循环队列中元素

void SQ\_Print(SeqQueue\* sq)

// 依次打印出队列中的每个元素。

{

    int i=sq->front;

    if (SQ\_IsEmpty(sq)) {

        printf("queue is emtpy");

        return;

    }

    for (i=sq->front; i!=sq->rear; i=(i+1)%sq->max) {

        printf("%d  ", sq->data[i]);

    }

    printf("\n");

}

### 3.2.4——队列的链式存储结构

typedef struct LinkNode{

ElemType data;

struct LinkNode\*next；

}LinkNode；

typedef struct{

LinkNode\*front，\*rear；

}LinkNode；

* 当Q.front==NULL或Q.rear==NULL时，链式队列为空

### 3.2.5——链式队列的初始化

void InitQueue(LinkQueue &Q){

Q.front=Q.rear=(LinkNode\*)malloc(sizeof(LinkNode)) //建立头结点

Q.front->next=NULL； //初始化为空

}

### 3.2.6——链式队列的其他操作

* 判队空，入队，出队，取队头元素
* 队列的出队一定要记得从队头出去，入队是从队尾进来的

#### 3.2.6.1——链式队列出队：

void LQ\_Out(LinkQueue \*Q, ElemType \*e)

//´Ó¶ÓÁÐQ³ö¶ÓÒ»¸öÔªËØ£¬¼´£ºÉ¾³ýQµÄ¶ÓÍ·ÔªËØ£¬ÓÃe·µ»ØÆäÖµ¡£

{

    // ÇëÔÚÕâÀï²¹³ä´úÂë£¬Íê³É±¾¹ØÈÎÎñ

    /\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

    QNode\*m=NULL;LinkQueue\*n=Q;

    if(LQ\_IsEmpty(\*Q))

    {

        //printf("为空队");

        return;

    }

    else

    {

        \*e=Q->front->next->data;

        m=Q->front->next;//保留队首元素地址

        Q->front->next=m->next;

        free(m);

        if(Q->front->next==NULL)

            Q->rear = Q->front;

}

**对于有头结点的队列，出完队列要记得判断首元结点是否为空**

# 第四章——串、数组和广义表

## 4..1——串的定义

串是由零个或多个字符组成的有限序列

* 子串：串中任意多个连续的字符组成的子序列称为该串的子串
* 主串：包含子串的串称为主串
* 空串：由一个或多个空格组成的串称为空格串

## 4.2——串的顺序存储结构

* 定长顺序存储表示

#define MAXLEN 255

typedef struct{

char ch[MAXLEN] //每个分量存储一个字符

int length //串的实际长度

}SString;

* 堆分配存储表示

堆分配存储表示仍然以一组地址连续的存储单元存放串值的字符序列，但他们的存储空间是在程序执行过程中动态分配得到的

typedef struct{

char \*ch //按串长分配存储区，ch指向串的基地址

int length //串的长度

}HString

## 4.3——串的模式匹配

* 前缀
* 后缀

### 4.3.1——BF算法

### 4.3.2——KMP算法

## 4.3——数组和特殊矩阵

### 4.3.1——数组定义

数组是由n个相同类型的数据元素构成的有限序列，是线性表的推广

* 一维数组可视为一个线性表
* 二维数组可视为其元素也是定长线性表的线性表

### 4.3.2——数组的存储结构

* 采用数组数据类型存储
* 一维数组： LOC()=LOC()+i\*L(0≤i<n) L是每个数组元素所占的存储单元
* 多维数组：按照行优先或列优先

### 4.3.3——特殊矩阵的压缩储存

1. 特殊矩阵的压缩存储方法：找出特殊矩阵中值相同的矩阵元素的分布规律，把那些呈现规律性分布的，值相同的多个矩阵元素压缩储存到一个存储空间中

* 对称矩阵、三角矩阵、三对角矩阵

1. 稀疏矩阵：矩阵中元素个数为t，相对矩阵元素个数S来说非常少，即s远大于t的矩阵

* 采用三元组（行标，列标，值）的形式，将这些三元组按某种规律去储存它

# 第五章——树与二叉树