# 第一张概论

## 1.1 引言

两项基本任务：数据表示，数据处理

软件系统生存期：软件计划，需求分析，软件设计，软件编码，软件测试，软件维护

由一种逻辑结构和一组基本运算构成的整体是实际问题的一种数学模型，这种数学模型的建立，选择和实现是数据结构的核心问题。

机外表示 ------逻辑结构 ------存储结构

处理要求 -----基本运算和运算 ------- 算法

### 1.2.1 数据，逻辑结构和运算

数据：凡是能够被计算机存储，加工的对象通称为数据

数据元素：是数据的基本单位，在程序中作为一个整体加以考虑和处理。又称元素、顶点、结点、记录。

数据项： 数据项组成数据元素， 但通常不具有完整确定的实际意义， 或不被当作一个整体对待。 又称字段或域，是数据不可分割的最小标示单位。

### 1.2.2 数据的逻辑结构

逻辑关系：是指数据元素之间的关联方式，又称“ 邻接关系 ”

逻辑结构 :数据元素之间逻辑关系的整体称为逻辑结构。即数据的组织形式。

四种基本逻辑结构：

1. 集合：任何两个结点间没有逻辑关系，组织形式松散
2. 线性结构：结点按逻辑关系依次排列成一条“ 锁链 ”
3. 树形结构：具有分支，层次特性，形态像自然界中的树
4. 图状结构：各个结点按逻辑关系互相缠绕，任何两个结点都可以邻接。

注意点：

1. 逻辑结构与数据元素本身的形式，内容无关。
2. 逻辑结构与数据元素的相对位置无关
3. 逻辑结构与所含结点个数无关。

运算：运算是指在任何逻辑结构上施加的操作，即对逻辑结构的加工。

加工型运算：改变了原逻辑结构的“ 值 ” ，如结点个数，结点内容等。

引用型运算 :不改变原逻辑结构个数和值，只从中提取某些信息作为运算的结果。

引用：查找，读取

加工：插入，删除，更新

同一逻辑结构 S 上的两个运算 A 和 B， A 的实现需要或可以利用 B，而 B 的实现不需要利用 A，则称 A 可以归约为 B。

假如 X 是 S上的一些运算的集合， Y是 X 的一个子集， 使得 X 中每一运算都可以规约为 Y 中的一个或多个运算，而 Y 中任何运算不可规约为别的运算，则称 Y 中运算（相对于 X）为基本运算。

将逻辑结构 S和在 S 上的基本运算集 X 的整体（ S,X）称为一个数据结构。数据结构包括逻辑结构和处理方式。

## 1.3 存储实现和运算实现

由于逻辑结构是设计人员根据解题需要选定的数据组织形式， 因此存储实现建立的机内表示应遵循选定的逻辑结构。另一方面，由于逻辑结构不包括结点内容即数据元素本身的表示，因此存储实现的另一主要内容是建立

数据元素的机内表示。按上述思路建立的数据的机内表示称为数据的存储结构。

存储结构包括三部分：

1. 存储结点 ，每个存储结点存放一个数据元素。
2. 数据元素之间关联方式的表示，　　也就是逻辑结构的机内表示。
3. 附加设施 ，如方便运算实现而设置的“ 哑结点 ”等。

四种基本存储方式：

1. 顺序存储方式 ：每个存储结点只含一个数据元素。所有存储结点相继存放在一个连续的存储区里。用存储结点间的位置关系表述数据元素之间的逻辑关系。
2. 链式存储方式 ：每个存储结点不仅含有一个数据元素，还包含一组指针。每个指针指向一个与本结点有逻辑关系的结点，即用附加的指针表示逻辑关系。
3. 索引存储方式 ：每个存储结点只含一个数据元素，所有存储结点连续存放。此外增设一个索引表，索引指示各存储结点的存储位置或位置区间端点。
4. 散列存储方式 ：每个结点含一个数据元素，各个结点均匀分布在存储区里，用散列函数指示各结点的存储位置或位置区间端点。

### 1.3.2 运算实现

运算只描述处理功能，不包括处理步骤和方法；运算实现的核心是处理步骤的规定，即算法设计。

算法：算法规定了求解给定问题所需的所有处理步骤及其执行顺序，使得给定类型的任何问题能在有限时间内被机械的求解。

算法分类：

**1**：运行终止的程序可执行部分：又称为程序

**2**：伪语言算法 ：不可以直接在计算机上运行，但容易编写和阅读。

3：非形式算法 ：用自然语言，同时可能还使用了程序设计语言或伪语言描述的算法。

## 1.4 算法分析

算法质量评价指标：

1. 正确性 ：能够正确实现处理要求
2. 易读性 ：易于阅读和理解，便于调试，修改和扩充
3. 健壮性 ：当环境发生变化，算法能够适当做出反应或处理，不会产生不需要的运行结果
4. 高效率 ：达到所需要的时空性能。

如何确定一个算法的时空性能，称为算法分析

一个算法的时空性能是指该算法的时间性能和空间性能，　　前者是算法包含的计算量，　后者是算法需要的存储量。

算法在给定输入下的计算量：

1. 根据该问题的特点选择一种或几种操作作为“ 标准操作 ”　　。
2. 确定每个算法在给定输入下共执行了多少次标准操作，并将此次数规定为该算法在给定输入下的计算量。

若无特殊说明，将默认以赋值语句作为标准操作。

最坏情况时间复杂性：算法在所有输入下的计算量的最大值作为算法的计算量

平均时间复杂性：算法在所有输入下的计算量的加权平均值作为算法的计算量。

算法的输入规模 (问题规模 )是指作为该算法输入的数据所含数据元素的数目，或与此数目有关的其他参数。

常见时间复杂性量级：

1. 常数阶： O(1)即算法的时间复杂性与输入规模 N 无关或 N 恒为常数。
2. 对数阶： $O\log_2 N$
3. 线性阶： O(N)
4. 平方阶： $O(N^2)$
5. 指数阶： $O(2^N$ 次方 )

   通常认为指数阶量级的算法实际是不可计算的，而量级低于平方阶的算法是高效率的

## 第二章线性表

### **2.1** 线性表的基本概念

线性结构：线性结构是 N（N 大于等于 0）个结点的有穷序列。

Ai 称为 Ai+1 的直接前驱， Ai+1 称为 Ai 的直接后继。 为满足运算的封闭性， 通常允许一种逻辑结构出现不含任何结点的情况。不含任何结点的线性结构记为（）或

线性结构的基本特征： 若至少含有一个结点， 则除起始节点没有直接前驱外， 其他结点有且只有一个直接前驱，除终端结点没有直接后继外，其他结点有且只有一个直接后继。

#### 2.1.2 线性表

线性表的逻辑结构是线性结构。所含结点个数称为线性表的长度 (表长 )。表长为 0 的是空表。

线性表的基本运算：

1. 初始化 initiate （L）：加工型运算，其作用是建立一个空表 L。
2. 求表长 length （L）：引用型运算，其结果是线性表 L 的长度。
3. 读表元 get（L, i）：引用型运算。若 i 小于等于 length（L），其结果是 L 的第 i 个结点，否则为一特殊值。
4. 定位（按值查找） locate（L, X）：引用型运算。若 L 中存在一个或多个值与 X 相等，结果为这些结点的序号最小值，否则，运算结果为 0。
5. 插入 insert (L, X, i)：加工型运算。在 L 的第 i 个位置上增加一个值为 X 的新结点，参数 i 的合法取值范围是 1---L+1。
6. 删除 delete （L, i）：加工型运算。撤销 L 的第 i 个结点 Ai, i 的合法取值范围是 1---N。

### 2.2 线性表的顺序实现

#### 2.2.1 顺序表

顺序表是线性表的顺序存储结构，即按顺序存储方式构造的存储结构。

顺序表基本思想：

顺序表的一个存储结点存储线性表的一个结点的内容，即数据元素 (不含其他信息 )，所有存储结点按相应数据元素间的逻辑关系决定的次序依次排列。

顺序表的特点：逻辑结构中相邻的结点在存储结构中仍相邻。

顺序表的类 C 语言描述： p17

```
Constmaxsize=顺序表的容量
Typedefstruct
    { datatype date [maxsize]
    Int   last;
     } sqlist;
    Sqlist L;
```

**L** 表示线性表的长度， **last-1** 是终端结点在顺序表中的位置。常数 **maxsize** 为顺序表的容量。

表长 **L.last ,** 终端结点 **L.data[L.last-1]**

2.2.2 基本运算在顺序表上的实现

1. 插入

```
Void inset_sqlist (sqlistL,datatype x, inti)
{ if (L.last == maxsize) error( 表满 ’ );     /*‘溢出 */
If (((i<1)!!(i>L.last+1)) error (  非法位置  ’ );  ‘
For     (j=L.last ; j=I; j--)
L.data[j] = L.data [j-1];            /* 依次后移 */
L.data[i-1 ]= x;           /* 置入 */
L.last =L.last+1           /* 修改表长 */
}
```

2. 删除

```
Void    delete_sqlist ( sqlist L, int I )       /* 删除顺序表   L 中第 i 个位置上的结点   */
{
  If    ( ( i<1 ) !! (I >L.last)) error (  非法位置  ’ );     ‘
For    ( j= i+1; j= L.last; j++)
L.data [j-2 ] = L.data [j-1 ];       /* 依次前移，注意第一个     L.data[j-2] 存放  ai*/
L.last=L.last-1                      /* 修改表长 */
```

3. 定位

```
Intlocate_sqlist (sqlist L , datatype X)
  /* 在顺序表中从前往后查找第一个值等于        X 的结点。若找到则回传该结点序号，否则回传         0*/
{
   I=1 ;
While ( ( i<= L.last) && (L.data[i-1]!=x) )       /* 注意： ai 在 L.data[i-1] 中*/
i++;                                              /* 从前往后查找   */
if (i<=L.last) return (i)
  else return (0)
}
```

2.2.3 顺序实现的算法分析

插入 :平均时间复杂性：  =n/2

平均时间复杂性量级为  O（n）

删除：平均时间复杂性：  n-1/2

平均时间复杂性量级：  O（n）

定位：平均时间复杂性量级：  O（n）

求表长，读表元：量级  O（1）

以上分析得知：顺序表的插入，删除算法的时间性能方面是不理想的。

2.3 线性表的链接实现

顺序表的优缺点：

优点： 1。无需为表示结点间的逻辑关系而增加额外的存储空间。

2．可以方便地随机存取表中的任一结点。

缺点： 1。插入，删除运算不方便，除表尾位置外，其他位置上进行插入和删除操作都必须移动大量结点，效

率较低。

2．由于顺序表要求占用连续的空间，存储分配职能预先进行（静态分配），因此当表长变化较大时，可能造成空间长期闲置或空间不够而溢出。

链表：采用链接方式存储的线性表称为链表

一种数据结构的链接实现是指按链式存储方式构建其存储结构，并在此链式存储结构上实现其基本运算。

### 2.3.1 单链表

单链表表示法的基本思想：用指针表示结点间的逻辑关系。

一个存储结点包含两部分：

data 部分：称为数据域，用于存储线性表的一个数据元素。

Next 部分：称为指针域或链域，用于存放一个指针，指向本结点所含数据元素的直接后继所在的结点

终端结点的指针 NULL 称为空指针，不指向任何结点，只起标志作用。

Head 称为头指针变量，指向单链表的第一个结点的，称为头指针。

头指针具有标识单链表的作用，故常用头指针变量来命名单链表。

单链表的类 C 语言描述：

```
Typedefstruct node     *pointer;
Struct node
      { datatype data;
        Pointer next;
        };
        Typedef pointer lklist;
```

### 2.3.2 单链表的简单操作

为了便于实现各种运算，通常在单链表第一个结点前增设一个类型相同的结点，称为头结点。其他结点称为表结点。表结点中第一个和最后一个称为首结点和尾结点。头结点的数据域可以不存储任何信息，也可以存放一个特殊标志或表长。

1 初始化：

```
Lklistinitiate_lklist()     /* 建立一个空表  */
        {
            t = malloc(size);
            t - > next = NULL;
            return(t);     }
```

此算法说明的问题：

1. 语句 t = malloc(size); 有双重作用： 1 由 malloc 自动生成一个类型为 node 的新结点。 2 指针型变量 t 得到一个值即指针，该指针指向上述新结点。
2. 要生成新结点必须通过调用 malloc 才能实现。
3. 语句 t - > next=NULL 的作用是将头结点 *t 的链域置为 NULL。
4. 为了建立一个空表，可定义一个 lklist 类型的变量 head,并通过调用 head =initiate_lklist( )使 head 成为指向一个空表的头指针。

2 求表长

```
Intlength_lklist(lklist head)       /* 求表 head 的长度,  P 是 pointer 类型的变量 */
      { p=head;
        J=0;
```

```
While (p ->next!=NULL)
    { p=p->next;
      J++;   }
Return (j );    }
```

3 按序号查找

Pointer find_lklist ( lklist head ,int I ) /* 在单链表 head 中查找第 i 个结点。若找到则回传指向该结点的指针，否则回传 null*/

```
{   p= head; j=0;
    While ( p->next !=NULL)&&( j<i)
    { p= p->next; j++; }
      If (i==j) return (p);
      Else return(NULL);   }
```

4 定位

```
Intlocate_lklist ( lklist head, datatype x)
   { p=head ; j= 0;
     While ( (p->next!=NULL)&&(p->data!=x))
     { p= p->next; j++;}
       If p->data ==x      return(j);
       Else     return (0);
     }
```

5 删除

```
  Void delete_lklist ( lklist head, inti)
   { p= find_lklist (head,i-1);     /* 调用按序号查找算法   */
     If ((p!=NULL)&&(p->next!=NULL))
     {q=p->next;
     p->next = q->next;
     free (q);    }
     else error( 不存在第 i 个结点 ’) }
```

free 是库函数，结果是释放 q 所指结点占用的内存空间，同时 q 的值变成无定义。

6 插入

```
 Void insert_lklist( lklisthead,datatyped x ,inti)
 {
    P=find_lklist (head, i-1);
     If ( p==NULL)
    Error ( 不存在第 i 个位置 ’)
      Else
      { s= malloc (size); s->data= x;
        s->next=p->next;
        p->next =s;     }
```

2.5 其他链表

循环链表

尾结点的链域值不是 NULL,而是指向头结点的指针。优点是从任一结点出发都能通过后移操作而扫描整个循环链表。

但为找到尾结点，必须从头指针出发扫描表中所有结点。改进的方法是不设头指针而改设尾指针。这样，

头结点和尾结点的位置为：　　rear->next->next　和 rear.

双链表：在每个结点中增加一个指针域，所含指针指向前驱结点。

双链表的摘除　*P 的操作：　p->prior->next=p->next;

p->next->prior=p->prior;

链入操作：　P 后面链入 *q:　　q->prior=p;　　q->next=p->next;　　p->next->prior=q; p->next =q;

2.6 顺序实现与链接实现的比较

空间性能的比较：

存储结点中数据域占用的存储量与整个存储结点占用存储量之比称为存储密度。顺序表　　=1，链表 <1，所有顺序表空间利用率高。但顺序表要事先估计容量，有时造成浪费。

时间性能的比较：

一种实现的时间性能是指该实现中包含的算法的时间复杂性。

定位：顺序表和链表都是　　O（n）

读表元：顺序表　O（1），链表　O（n），故当需要随机存取时，不宜采用链表。

摘除，链入：顺序表　　O（n），链表 O（1），经常需要插入删除时不宜采用顺序表。

2.7 串

串是由零个或多个字符组成的又穷序列。含零个字符的串称为空串。串中所含字符的个数称为该串的长度。

两个串完全一样时称为相等的。串中任意个连续字符组成的子序列称为该串的子窜，该窜称为主窜。

字符串常量按字符数组处理，　　它的值在执行过程中不能改变。　　串变量与其他变量不一样，　　不能由赋值语句赋值。

串的基本运算：

1. 赋值： ASSIGN(S,T)加工型运算。将串变量或串常量的值传给串变量。
2. 判等： EQUAL(S,T): 引用型运算，若相等返回　　1，否则返回　0。
3. 求长： LENGTH(S): 引用型运算
4. 联接： CONCAT(S,T) 引用型运算。运算结果是联接在一起形成的新串。
5. 求子串： SUBSTR(S,I,j): 引用型运算：结果是串　　S中从第　i 个字符开始，由连续　　j 个字符组成的子串。当 I,j 参数超过范围时，运算不能执行，也没有结果。
6. 插入： INSERT(S1,I,S2) 加工型运算。将串　　2 整个插到　S1的第 i 个字符之后从而产生一个新串。
7. 删除 DELETE(S,I,J)加工型运算。从串　　S中删去第　I 个字符开始的长度为　　J的子串。
8. 定位： INDEX(S,T): 引用型运算。若串　　S中存在一个与　T 相等的子串。则结果为　　S中第一个这样的子串的第一个字符在　S 中的位置，否则，结果为　　0。（要求　T 不是空串）
9. 替换： REPLACE(S,T,R)加工型运算。在　　S中处处同时以串　　R置换 T 的所有出现所得的新串。

串的存储：

1. 串的顺序存储：紧缩格式，非紧缩格式
2. 串的链接存储：将串中每个存储结点存储的字符个数称为结点大小。结点为　　1 时存储密度低但操作方便，大于　1时存储密度高但操作不方便。

## 第三章栈，队列和数组

3.1 栈

栈是一种特殊的线性表，栈上的插入删除操作限定在表的某一端进行，称为栈顶。另一端称为栈底。不含任何元素的栈称为空栈。

栈又称为先进后出线性表。在栈顶进行插入运算，被称为进栈，删除被称为出栈。

栈的基本运算：

1. 初始化： InitStack(S):加工型运算，设置一个空栈 S.
2. 进栈： push（S,X）加工型运算，将元素 X 插入 S 中，使 X 称为栈顶元素。
3. 退栈： pop （S）加工型运算，当栈不空时，从栈中删除当前栈顶。
4. 读栈顶： top （S）：引用型运算，若 S不空，由 X 返回栈顶元素， S 为空时，结果为一特殊标志。
5. 判栈空 empty （S）：引用型运算，若 S 为空栈，结果为 1，否则为 0

### 3.1.2 栈的顺序实现

顺序栈由一个一维数组和一个记录栈顶位置的变量组成。

空栈中进行出栈操作，发生下溢，满栈中进行入栈操作，发生上溢。

类 C 语言定义：

```
#define sqstack_maxsize 6    /*6 是栈的容量 */
Typedefstructsqstack
   { DataType dada[sqstack_maxsize];
Int top;
    }SqStackTP;
```

栈的基本运算的实现：

1. 初始化

```
IntInitStack(InitStackTp *sq)
{sq->top=0;
  Return(1);    }
```

2. 进栈

```
Int push(sqstackTp *sq, datatype x)
  { if (s->top == sqstack_maxsize-1)
   {error( 栈满 ”);return 0;}
  Else{sq-top++;
  Sq->data[sq->top]=x;
    Return(1);    }
```

3 退栈

```
Int pop(sqstackTp *sq,datatype *x)
  {if (sq->top==0)
  {error( 下溢 ”);return(0);}
  Else {*x=sq->data[sq->top];
  Sq->top--;
  Return(1);    }
```

4 判栈空

```
Intemptystack(stackTp    *sq)
{if sq->top==0}
   Return(1);
Else return(0); }
```

5 取栈顶元素

```
Intgettop( sqstackTp *sq, datatype *x)
{if(sq->top=0) return(0);
Else{*x =sq->data[sq->top];
Return(1); }
```

### 3.1.3 栈的链接实现

链栈由栈顶指针 ls 唯一确定。栈中其他结点通过他们的 next 域链接起来。栈底结点的 next 域为 NULL。
因为链栈本身没有容量限制，所以不会出现栈满情况。

3. 1.5 栈的简单应用和递归

栈与函数调用 ：

函数调用时，先保存的位置后返回，后保存的位置先返回。所以每遇到一个函数调用便立刻将相应的返回位置进栈，调用结束时，栈顶元素正好是此函数的返回位置。

递归与栈：

满足递归的条件：

1. 被定义项在定义中的应用具有更小的尺度。
2. 被定义项在最小尺度上的定义不是递归的。

## 3.2 队列

队列也可以看成一种受限的线性表，插入限定在表的某一端进行（队尾） ，删除限定在另一端进行（队头）
队列又称先进先出线性表。

队列的基本运算：

1. 队列初始化 initQueue(Q) 加工型运算，设置一个空队列 Q
2. 入队列 enQueue(Q,X)加工型运算，将 X 插入到队列 Q 的队尾。若原队列为空，则 X 为原队尾结点的后继，同时是新队列的队尾结点。
3. 出队 outQueue(Q,X) 加工型运算，若队列 Q 不空，则将队头元素赋给 X,并删除队头结点，其后继成为新的队头结点。
4. 判队列空 emptyQueue(Q) 引用型运算，若队列 Q 为空，则返回 1，否则为 0
5. 读队头 gethead(Q,x) 引用型运算， Q 不空时由参数 X 返回队头结点的值，否则给一特殊标志。

队列的顺序实现：
队列的顺序实现由一个一维数组及两个分别指示队头和队尾的变量组成，称为队头指针和队尾指针。约定队尾指针指示队尾元素在一维数组中的当前位置，队头指针指示队头元素在一维数组中的当前位置的前一个位置。

如果按 sq.rear=sq.rear+1; sq.data[sq.rear]=x 和 sq.front=sq.front+1 分别进行入队和出队操作， 则会造成“ 假溢出。 ”

循环队列的入队操作： sq.rear=(sq.rear+1)%maxsize; sq.data[sq.rear]=x
出队操作： sq.front=(sq.front+1)%maxsize;
判断循环队列队满的条件： ((sq.rear+1)%maxsize)==sq.front
队空的条件： sq.rear==sq.front

## 3.3 数组

二维数组可以看成是一个被推广的线性表，这种线性表的每一个数据元素本身也是一个线性表。

数组只有两种基本运算：

1. 读：给定一组下标，读取相应的数据元素
2. 写：给定一组下标，修改相应的数据元素

数组元素的存储位置是下标的线性函数，计算存储位置所需的时间取决于乘法的时间，因此，存取任一元素的时间相等。通常将具有这一特点的存储结构成为随机存储结构。

### 3.3.3 矩阵的压缩存储

压缩存储的基本思想：值相同的多个元素只分配一个存储空间，零元素不分配空间。

要压缩的矩阵分为两种

1. 特殊矩阵：对称矩阵，三角矩阵。值相同的元素或零元素的分布有一定规律叫特殊矩阵。
   对称矩阵通常存储下三角，　　n 阶方阵需要　n*(n+1)/ 2 个存储单元
   三角矩阵需要　n*(n+1)/ 2+1 个存储单元，最后一个单元存储相同的常数。
2. 稀疏矩阵：零元素远多于非零元素，且非零元素的分布没有规律。
   用三元组表存储稀疏矩阵，只存储非零元素。　　(I,j,aij)

# 第四章树

## **4.1** 树的基本概念

树的定义：树是 n(n>0)个结点的有穷集合，满足：树——是 n(n>=0)个结点的有限集 T，满足：

(1)有且仅有一个特定的称为根的结点；

(2)其余的结点可分为 m(m>=0)个互不相交的子集， T1,T2,T3 Tm，其中每个子集 Ti 又是一棵树，并称其为子树。

有关术语：

树上任一结点所拥有的子树的数目称为该结点的度。度为 0 的结点称为叶子或终端结点。度大于 0 的结点称为非终端结点或分支点。一棵树中所有结点度的最大值称为该树的度。

若结点 A 是 B 的直接前驱，则称 A 是 B 的双亲或父节点，称 B 为 A 的孩子或子节点。父节点相同的结点互称为兄弟。

一棵树的任何结点（不包括根节点）称为根的子孙，根节点称为其他节点的祖先。

结点的层数（或深度）从根开始算，根的层数为 1，其他节点的层数为其双亲的层数加 1。树中结点层数的最大值称为该树的高度或深度。

树的基本运算：

1. 求根 ROOT(T)引用型运算，结果是树 T 的根结点。
2. 求双亲 PARENT (T,X)引用型运算，结果是结点 X 在树 T 上的双亲， 若 X 是树 T 的根或 X 不在 T 上，则结果为一特殊标志。
3. 求孩子 CHILD(T,X,I)引用型运算， 结果是树 T 上结点 X 的第 i 个孩子， 若 X 不在 T 上或 X 没有第 i 个孩子，结果为一特殊标志。
4. 建树 CREATE (X,T1 ‥;‥TK) ， K>=1，加工型运算，建立一棵以 X 为根，以 T1…… .TK为第 1…… K棵子树的树。
5. 剪枝 DELETE（T,X,i）加工型运算，删除树 T 上结点 X 的第 i 棵子树，若 T 无第 i 棵子树，则操作为空。
6. 遍历 Traverse Tree(T)遍历树，即访问树中每个结点，且每个结点仅被访问一次。

4.2 二叉树是 n(n>=0)个结点的有限集合，它或为空 (n=0)，或是由一个根结点及最多有两棵互不相交的左、右子树组成，且每棵子树都是二叉树，或者同时满足下述两个条件：

1. 有且仅有一个称为根的结点。
2. 其余结点分为两个互不相交的集合 T1,T2，都是二叉树，并且 T1,T2 有顺序关系， T1 在 T2 之前，他们分别称为根的左子树和右子树。

二叉树的五种形态：

空二叉树，

只含根的二叉树，

只有非空左子树的二叉树，

只有非空右子树的二叉树，

同时有非空左右子树的二叉树。

二叉树的基本运算：

1. 初始化 INITIATE(BT)加工型运算，作用是设置一棵空二叉树
2. 求根 ROOT(BT)引用型运算，结果是二叉树 BT 的根节点，若 BT 为空二叉树，结果为一特殊标志。
3. 求双亲 PARENT(BT,X)引用型运算， 结果是结点 X 在二叉树 BT 上的双亲， 若 X 是根或不在 BT 上，结果为一特殊标志
4. 求左孩子 LCHILD(BT,X)右孩子 RCHILD(BT,X)引用型运算，结果分别为结点 X 在 BT 上的左，右孩子。 若 X 为 BT 的叶子或 X 不在 BT 上，结果为一特殊标志。
5. 建树 CREAT(X,LBT,RBT)加工型运算，建立一棵 X 为结点， LBT,RBT为左右子树的二叉树。
6. 剪枝 DELETEFT(BT,X) DELETEHT(BT,X)加工型运算，删除 BT 结点 X 的左右子树，若无，运算为空。

二叉树的性质：

性质 1、二叉树第 i(i>=1)层上至多有 $2^{i-1}$ 个结点。

性质 **2**、深度为 **K**（**k>=1**）的二叉树至多有 **$2^k$ -1** 个结点。
性质 **3**、对任何二叉树，若 **2** 度结点数为 **n2,**则叶子数 **n=n2+1**

深度为 K（K>=1）且有 $2^k$-1 结点的二叉树为满二叉树，在第 K 层删去最右边连续 J 个结点，得到一棵深度为 K 的完全二叉树。

完全二叉树的性质： |_x| 表示不大于 X 的最大整数。

性质 **4**、具有 **N** 个结点的完全二叉树的深度为 **$\lfloor \log_2 n \rfloor$+1**

性质 **5**、将一棵有 **n** 个结点的完全二叉树按层编号，则对任一编号为 **i** 的结点 **X** 有：
若 **i=1**，则结点 **X** 是根，若 **i>1**，则 **X** 的双亲 **parent**（**X**）的编号为 **|_i/ 2|**
若 **2i>n,**则结点 **X** 无左孩子（且无右孩子），否则， **X** 的左孩子 **LCHILD(X)** 的编号为 **2i.**
若 **2i+1>n,**则结点 **X** 无右孩子，否则 **,X** 的右孩子 **RCHILD(X)** 的编号为 **2i+1**

4.3 二叉树的存储结构
二叉树的链式存储结构：
二叉链表在做求双亲运算时效率不高，此时可采用三叉链表。
具有 **n** 个结点的二叉树中，一共有 **2n** 个指针域，其中只有 **n-1** 个用来指向结点的左右孩子，其余的 **n+1** 个指针域为 **NULL.**
**P81**

4.3.2 二叉树的顺序存储结构
按层编号然后存储。
对于非完全二叉树， 可采用增加虚结点的方式转化成完全二叉树再进行存储。 虚结点在数组中用特殊记号表示。
但同时会浪费存储空间。

4.4. 二叉树的遍历
遍历一棵二叉树就是按某种次序系统地“ 访问 ”二叉树上所有结点，使每个节点恰好被访问一次。
先根遍历： **1** 访问根结点 **2** 先根遍历左子树 **3** 先根遍历右子树
中根遍历： **1** 中根遍历左子树 **2** 访问根结点 **3** 中根遍历右子树
后根遍历： **1** 后根遍历左子树 **2** 后根遍历右子树 **3** 访问根结点。

4.6 树和林
树的存储结构： P93

1. 孩子链表示方法 ：头结点分为数据域和指针域，表结点分为孩子域和指针域
   可以在头结点中增加双亲域，称为带双亲的孩子链表示方法。
2. 孩子兄弟链表示法 ：存储结点均含三个域：数据域，孩子域（存放指向本结点第一个孩子的指针） ，兄弟域（存放指向本结点下一个兄弟的指针） 。
3. 双亲表示法 ：数据域，指针域（指示本结点的双亲所在的存储结点）

将指针域定义为高级语言中的指针类型的链式存储结构成为“ 动态链表 ” ，相应的指针成为动态指针。
将指针域定义为整形，子界型的链式存储结构成为静态链表，相应的指针称为静态指针。
动态链表的结构通过库函数 malloc（size）动态生成，无需事先规定表的容量。而静态链表容量须事先说明。

**4.6.2** 树的遍历
**1**. 先根遍历：若树非空 **1** 访问根结点 **2** 依次先根遍历根的各个子树
**2**. 后根遍历： **1** 依次后根遍历根的各个子树 **2** 访问根结点

**3**　层次遍历：　　**2** 访问根结点　**2** 从左到右，从上到下依次访问每层。

二叉树与树，林的关系　　P97
将二叉树的二叉链表和数的孩子兄弟链表的左孩子指针，右孩子指针和孩子指针，兄弟指针对应起来。
与树对应的二叉树的右子树一定为空。

判定树和哈夫曼树
用于描述分类过程的二叉树称为判定树。　　判定树的每个非终端结点包含一个条件，　　因而对应于一次比较火判断，每个终端结点包含一个种类标记，对应于一种分类结果。

哈夫曼树：
给定一组值　p1…… pĶ 如何构造一棵有　k 个叶子且分别以这些值为权的判定树，使得其平均比较次数最小。满足上述条件的判定树称为哈夫曼树。的

## 第五章图

图中的 小圆圈称为顶点，连线称为边，连线附带的数值称为边的权。
任何两点间相关联的无向图称为无向完全图，一个　　N 个顶点的完全无向图的边数为　　n(n-1)/ 2.
任何两顶点间都有弧的有向图称为有向完全图。一个　　N 个顶点的有向完全图弧数位　　n(n-1)
每条边或弧都带权的图称为带权图或网。

一个连通图的生成树，是含有该连通图的全部顶点的一个极小联通子图。
若连通图的顶点个数位　N,则生成树的边数为　N-1,如果它的一个子图的边数大于　N-1,则其中一定有环，　如果小于，则一定不连通。

### 5.2 图的存储结构

邻接矩阵
对于无向图，顶点　VI 的度是矩阵中第　I 行（或列）的元素之和。
对于有向图，行元素之和为出度，列元素之和为入度。

邻接表
为每个顶点建立一个单链表，单链表中每个结点称为表结点，包括两个域，邻接点域，用以存放与　　**VI** 相邻接的顶点序号，链域，用以指向同　　**VI** 邻接的下一个的顶点。
另外，每个单链表设一个表头结点。每个表头结点有两个域，一个存放顶点　　**VI** 的信息，另一个指向邻接表中的第一个结点。

若一个无向图有　N 个顶点，　E 条变，则它的邻接表需要　　N 个头结点和　2E 个表结点，所以在边稀疏的情况下，用邻接表比邻接矩阵更节省存储空间。

对于无向图，第　I 个单链表中的结点个数即为　　VI 的度。
对于有向图，第　I 个单链表中的结点个数只是　　VI 的出度，为求入度，必须遍历整个邻接表，所有单链表中，邻接点域的值为　I 的结点个数即为入度。
有时为了方便的求入度，可以建立逆邻接表。

### 5.3 图的遍历

从图中某一顶点出发访遍图中其余顶点，每个顶点仅访问一次，叫做图的遍历。
增设 visited[n] 数组。初值为　0,vi 被访问后，置为　1
遍历方法：深度优先搜索和广度优先搜索。

最小生成树问题
拓扑排序

第六章查找表
集合的特点：在集合这种逻辑结构中，任何结点间都不存在逻辑关系。

用来标识数据元素的数据项称为关键字，简称键，该数据项的值称为键值。

静态查找表：以集合为逻辑结构，包括三种基本运算

1. 建表 CREATE(ST)加工型运算，生成一个由用户给定的若干数据元素组成的静态查找表 ST.
2. 查找 SEARCH(ST,K)引用型运算，若 ST中存在键值等于 K，结果为该键在 ST 中的位置，否则为一特殊标志
3. 读表元 GET(ST,pos)引用型运算，结果是 pos 位置上的数据元素。

动态查找表：包括查找，读表元（同上）和以下三种基本运算。

1. 插入 INSERT(ST,K)加工型运算，若 ST中不存在键值等于 K 的数据元素，则将一个键值等于 K 的数据元素插入到 ST中。
2. 删除 DELETE(ST,K)加工型运算，删除 ST中键值等于 K 的数据元素。
3. 初始化 INITIATE(ST)加工型运算，设置一个空的动态查找表。

6．2 静态查找表的实现

6.2.1 顺序表上的查找

顺序查找法： 从表的第 n 个位置开始， 从后往前依次将各个位置上的数据元素的键值与给定值 K 比较。 若相等，回送该位置作为结果。若不等，查找不成功。

在第 0 个单元设置哨岗，所有当查找不成功时，查找了 n+1 次。

平均查找长度 ASL=（N+1）/2

6.2.2 有序表的查找

二分查找法

当 N 较大时， ASL约等于 $\log_2[(n+1])$ -1

6,2,3 索引顺序表上的查找

索引顺序表由顺序表和索引表两部分组成。

两个特点： 1 顺序表中的数据元素按块有序 2 索引表反映了这些快的有关特性（块内最大键值和块的起始位置）

分块查找法

平均查找长度高于顺序查找而低于二分查找。

6,3 树表

6.3.1 二叉排序树

一棵二叉排序树或者是一棵空树，或者同时满足下列三个条件：

1. 若它的左子树不空，则左子树上所有结点的键值均小于它的根结点的键值。
2. 若它的右子树不空，则右子树上所有结点的键值均大于它的根结点的键值。
3. 它的左右子树也分别是二叉排序树。

二叉排序树的中根遍历所得排序是从小到大。

二叉排序树的插入，删除后仍要保持是一棵二叉排序树。

删除时：设待删结点为　P,双亲结点为　F.设 P 是 F 的左孩子。

1. P 为叶节点，直接置　F 的左指针域为空。
2. P 只有一棵非空子树，直接以其根节点代替　P 的位置。
3. P 有两颗非空子树，可用左子树根结点代替　P,然后将右子树变为新的根节点的右子树。

6.3. 2 平衡二叉排序树

一棵平衡二叉排序树　AVL 树，或者是空树，或者是一棵任一结点的左子树与右子树的高度至多差　1 的二叉排序树。

调整方法：

P140

6．4 散列表

散列函数是一种将键值映射为散列表中的存储位置的函数。

由散列函数决定的数据元素在散列表中的存储位置称为散列地址。

散列的基本思想是通过散列函数决定的键值与散列地址间的对应关系实现存储组织和查找运算。

散列函数的构造法：

1．数字分析法（需要事先知道大概的键值）　2 除余法　3 平方取中法　4 基数转换法　5 随机数法

第八章排序

假定待排序的序列中存在多个记录具有相同的键值。若经过排序，这些记录的相对次序保持不变，称这种排序的方法是稳定的，否则是不稳定的。

按照排序过程涉及的存储设备的不同，分为内部排序和外部排序。内部排序指排序过程中，数据全部存放在计算机内存中，并在内存中调整记录间的相对位置。外部排序指排序过程中，数据的主要部分存放在外存中，借助内存逐步调整记录间的相对位置。

排序以键值比较和记录移动为标准操作。

8.2 插入排序

分为直接插入排序，折半插入排序，表插入排序和希尔排序。

直接插入排序，方法是稳定的，时间复杂性为　$O(N^2)$,空间来看，只需要一个记录的辅助空间，故空间复杂度为 O（1）。

8.3 交换排序

根据序列中两个记录键值的比较结果来兑换这两个记录在序列中的位置。特点是，将键值较大的记录向序列尾部移动，键值较小的记录向序列前部移动。

8.3.1 冒泡排序

至多需要进行　N-1 趟起泡。

时间复杂性为　O（$n^2$），稳定的排序方法。