国家计算机等级考试二级笔试试题-

基础知识

真题考点分析与讲解

主　编　郎六琪
副主编　李　昱　王　婧

吉林大学出版社

国家计算机等级考试二级笔试试题——基础知识部分

真题考点分析与讲解

主编：郎六琪
副主编：李昱、王婧
参编人员：崔树林、冯广慧、程宇、李皓

吉林大学出版社

图书在版编目（CIP）数据

国家计算机等级考试二级笔试试题真题考点分析与讲解. 基础知识/ 郎六琪主编.
—长春：吉林大学出版社，2011.1
ISBN 978-7-5601-7004-6
Ⅰ. ①国… Ⅱ. ①郎… Ⅲ. ①电子计算机—水平考试—自学参考资料 Ⅳ. ①TP3

中国版本图书馆 CIP 数据核字（2011）第 013041 号

书　名：国家计算机等级考试二级笔试试题真题考点分析与讲解—基础知识
作　者：郎六琪　主编

责任编辑、责任校对：李国宏　郑宇
吉林大学出版社出版、发行
开本：787×1092　毫米　1/16
印张：9.375　　字数：178 千字
ISBN 978-7-5601-7004-6

封面设计：创意广告
长春华艺印刷有限公司　印刷
2011 年 1 月　第 1 版
2011 年 1 月　第 1 次印刷
定价：19.00 元

社　址：长春市明德路 421 号　邮编：130021
发行部电话：0431-88499826
网　址：http://www.jlup.com.cn
E-mail:jlup@mail.jlu.edu.cn

目　录

目　录

第一部分　算法与数据结构

基础知识概述

一、算法

算法是指为完成某个任务或解决某个特定问题所需要的具体步骤和方法。也就是说，给定初始状态或输入数据，通过计算机程序的有限次运算，能够得出所要求或期望的终止状态或输出数据。

1. 算法的特征

一个算法应该具有以下五个重要的特征：

1）有穷性：有穷性是算法的重要特征，不论采用什么算法其目的都是解决问题。因此，一个算法必须在有限的操作步骤内完成。如果一个算法的执行时间是无限的，或在期望的时间内没有完成，那么这种算法就是无用和徒劳的，我们不能称其为算法。

2）确定性：算法中的每一个步骤必须有确切的定义，即算法的描述必须无歧义，以保证算法的执行结果是确定的。

3）输入：一个算法通常有 0 个或多个输入，以此刻画运算对象的初始情况。

4）输出：算法的目的是解决问题，一个算法通常有一个或多个输出，以便反映算法运算的结果。

5）可行性：又称有效性，算法中的每一个步骤原则上都能够精确地运行，算法中描述的操作都可以通过已经实现的基本运算执行有限次来实现。

2. 算法的组成要素

算法通常由两种基本要素组成：一个是对数据对象的运算和操作；另外一个是算法的控制结构。

1)算法中对数据的运算和操作:对于所有算法都是按照要求从环境能够运行的所有操作中选择合适的操作所组成的一组指令序列。基本的运算和操作包括四类:算术运算、逻辑运算、关系运算和数据传输。

2)算法的控制结构:即算法中各操作之间的执行顺序。包括顺序、选择、循环三种结构。

3. 算法的复杂度

同一问题可用不同算法解决,而一个算法的质量优劣将影响到程序的效率。算法分析的目的在于选择合适算法和改进算法。对一个算法的评价主要从时间复杂度和空间复杂度来考虑。

1)时间复杂度

所谓算法的时间复杂度,是指执行算法所需要的计算工作量。通常,一个算法所用的时间等于编译时间加上运行时间。可以用算法在执行过程中所需基本运算的执行次数来度量算法的工作量。

一般来说,计算机算法是问题规模 n 的函数 $f(n)$,算法的时间复杂度也因此记作

$$T(n) = O(f(n))$$

因此,问题的规模 n 越大,算法执行的时间的增长率与 $f(n)$ 的增长率正相关,称作渐进时间复杂度(Asymptotic Time Complexity)。通常用 表示常数计算时间,常见的时间复杂度有:

$$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) < O(2^n)$$

2)空间复杂度

算法的空间复杂度是指执行算法需要消耗的内存资源。其计算和表示方法与时间复杂度类似,一般都用复杂度的渐近性来表示。同时间复杂度相比,空间复杂度的分析要简单得多。

二、数据结构

数据结构是指反映数据元素之间关系的数据集合的表示,即带有结构的数据元素之间的前后件关系。是整个计算机科学与技术领域的一个重要概念。它用来反映一个数据的内部构成,即一个数据包含什么内容,以什么方式构成,呈什么结构。数据结构作为计算机科学的一门学科,主要研究以下三个方面的内容:

1)数据元素之间的逻辑关系,也称数据的逻辑结构(Logical Structure)。

2)数据元素及其关系在计算机存储器内的表示,称为数据的存储结构(Storage Structure)。

3)数据的运算,对数据施加的操作,即数据的运算。

为更好地理解数据结构,还需理解以下几个基本概念:

数据(data)——所有能输入到计算机中去并被计算机处理的描述客观事物的符号。

数据元素(data element)——数据的基本单位,也称节点(node)或记录(record)。

数据项(data item)——有独立含义的数据最小单位,也称域(field)。

数据对象(data object)——具有相同特性的数据元素的集合,是数据的子集。

1. 数据的逻辑结构

数据的逻辑结构是指数据元素之间的逻辑关系,它可以用一个数据元素的集合和定义在此集合上的若干关系来表示。

数据的逻辑结构是从逻辑关系上描述数据,它与数据在计算机中的存储位置无关,是独立于计算机的。数据元素之间通常有4类基本逻辑结构,如图1-1所示。

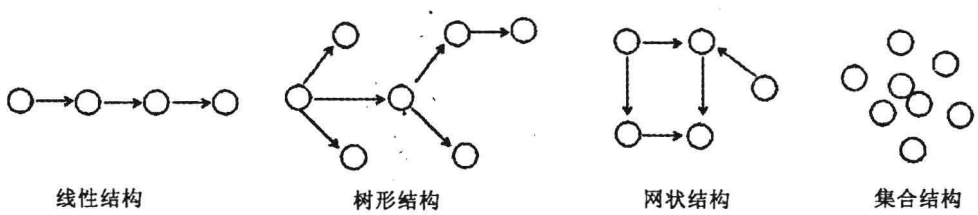


图1-1　数据结构图

2. 数据的存储结构

数据的存储结构是数据元素及其关系在计算机存储器内的表示。存储结构的主要内容是指在存储空间中使用一个存储结构点来存储一个数据元素;在存储空间中建立各存储结构点之间的关联,以表示数据元素之间的逻辑关系。

常用数据的存储结构有如下4种:

1)顺序存储方式。

2)链式存储方式。

3)索引存储方式。

4)散列存储式。

3. 数据结构的表示形式

1)用二元关系表示

数据的逻辑结构有两个要素:一是数据元素的集合,通常记为D;二是D上的关系,以反映D中数据元素之间的前后件关系,通常记为R。一个数据结构可以表示成:

$$B=(D,R)$$

其中,B表示数据结构,为了反映D中各数据元素之间的前后件关系,一般用二元组表示。例如家庭成员数据结构可以表示成

B=(D,R)

D={父亲,儿子,女儿}

R={(父亲,儿子),(父亲,女儿)}

2)用图形表示

在数据结构的图形表示中,对于集合D中的每个数据元素用标有元素值的方框表示,通常称为数据结点;对于关系R中的每个二元组,用一条有向线段从前件结点指向后件结点。例如家庭成员数据结构也可用图1-2所示方式表示。

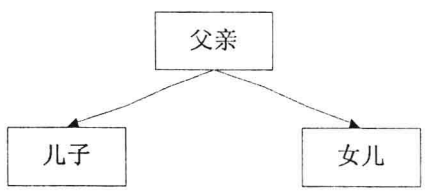


图1-2 数据关系

4. 线性结构与非线性结构

根据结构中各数据元素之间前后件关系的复杂程度,将数据结构分为两个类型:线性结构和非线性结构。

1)线性结构:在数据元素的非空有限集中,线性结构有如下特征:

① 存在唯一的一个被称作"根结点"的数据元素,该节点无前件。

② 存在唯一的一个被称作"终端结点"的数据元素,该结点无后件。

③ 除根结点外,集合中的每个数据元素均只有一个直接前件。

④ 除终端结点外,集合中的每个数据元素均只有一个直接后件。

2)非线性结构:其特征是一个结点可能有多个直接前驱和直接后继,例如,树和图都是非线性结构。

三、线性表

1. 线性表的定义

线性表是具有相同数据类型的 n 个数据元素组成的有限序列，通常记为：

$$(a_1, a_2, a_3, \cdots, a_{i-1}, a_i, \cdots, a_n)$$

其中 $n(n \geq 0)$ 称为线性表的长度，$n=0$ 时称为空表，在一个非空表中，每个数据元素都有一个确定的位置。如 a_1 是第一个数据元素，a_n 是最后一个数据元素，a_i 是第 i 个数据元素，称 i 为数据元素 a_i 在线性表中的位序。

线性表是一种非常灵活的数据结构，它的数据元素可以是一个数或一个符号，也可以由若干个数据项组成。线性表的长度可以根据需要增长或缩短，即对线性表的数据元素不仅可以进行访问，还可进行插入或删除等。

2. 线性表的顺序存储结构

线性表的顺序存储是指在内存中用地址连续的一块存储单元依次存储线性表的各数据元素。这类顺序存储结构的线性表也称为顺序表。顺序表中所有结点的类型相同，每个结点所占用存储空间大小亦相同。假设表中每个结点占用 c 个存储单元，其中第一个单元的存储地址是该结点的存储地址，并设表中开始结点 a_1 的存储地址（简称为基地址）是 $\mathrm{LOC}(a_1)$，那么结点 a_i 的存储地址 $\mathrm{LOC}(a_i)$ 可通过下式计算：

$$\mathrm{LOC}(a_i) = \mathrm{LOC}(a_1) + (i-1) * c \qquad 1 \leq i \leq n$$

由上式可以得出：顺序表中每个结点 a_i 的存储地址都是该结点在表中的位置 i 的线性函数，只要知道基地址和每个结点的大小，就可以找到任一结点的存储地址。也就是说，线性表是以元素在计算机内的物理位置关系来表示表中数据元素之间的逻辑关系。线性表表长可变，在设计数组容量时应设计得足够大，假设用 data [MAXSIZE] 来表示线性表的表长，那么 MAXSIZE 就是一个根据实际需要定义的足够大的整数，线性表中的数据元素从 data[0] 开始顺序存储。但如果线性表中的数据元素没能达到 MAXSIZE 时，还需设置一个变量 last 记录当前顺序表中最后一个元素的位置。

通常将 data 和 last 封装在一个结构中定义顺序表的类型：

```
typedef struct
{
    DataType data[MAXSIZE];      //data 数组用来存放表结点
    Int last;                    //当前顺序表中最后一个结点的位置
```

} SeqList

SeqList L;

这样就定义了一个顺序表 L,它所表示的顺序表如图 1 -3 所示。

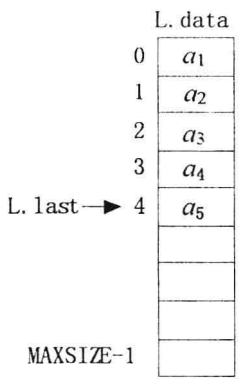


图 1 -3　线性表的顺序存储

该线性表表长为 L. last + 1,线性表中的数据元素 $a_1, a_2, a_3, \cdots, a_n$,分别存放在 L. data[0]到 L. data[L. last]中。

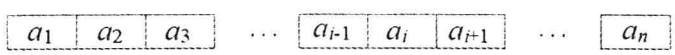
顺序存储结构的特点:

1)利用数据元素的存储位置表示线性表中相邻数据元素之间的前后关系,即线性表的逻辑结构与存储结构(物理结构)一致。

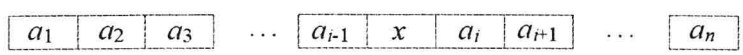
2)在访问线性表时,可以利用上述给出的数学公式,快速地计算出任何一个数据元素的存储地址。因此,我们可以粗略地认为,访问每个数据元素所花费的时间相等。这种存取元素的方法被称为随机存取法,使用这种存取方法的存储结构被称为随机存储结构。

顺序表的插入操作:

顺序表的插入是指在表中第 $i(1 \leqslant i \leqslant n+1)$ 个位置上插入一个值为 x 的新元素,插入后使原表长为 n 的顺序表:



变为表长为 n + 1 的顺序表:



顺序表的插入算法:

①将 a_i 到 a_n 顺序向后移动;

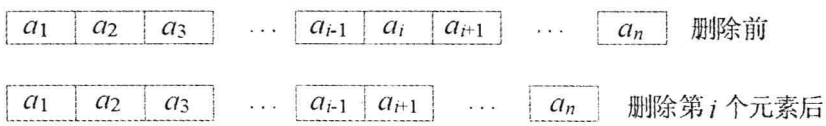
②将 x 插入到第 i 个位置;

③修改表长,使之仍然指向最后一个元素。

```
void ListInsert(SeqList *L,int i,DataType x)
{
DataType *p,*q;
if (i<1||i>L.last+2) return ERROR;          //L.last+1 为表长
if(L.last+1+1>=MAXSIZE) return ERROR;       //元素个数大于最大个数
q=&(L.data[i-1]);
for(p=&L.data[L.last];p>=q;--p)             //从最后一个元素开始结点后移
*(p+1)=*p;                                  //结点后移
*q=x;                                       //插入 x
++L.last;                                   //修改表长
return OK;
}/*ListInsert Before i */
```

顺序表的删除操作:

顺序表的删除是指将表中第 i($1<=i<=n+1$)个位置上的元素从表中去掉,删除后使原表长为 n 的顺序表表长减小为 $n-1$。如下图所示:



顺序表的删除算法:

```
ListDelete(SeqList *L,int i)
{
DataType *p,*q;
if (i<1||i>L.last+2) return ERROR;          //L.last+1 为表长,位置非法出错
q=&(L.data[i]);                             //从第 i+1 个元素开始结点前移
for(p=&L.data[i+1];p<=&L.data[L.last+1];q++,p++)
*q=*p;                                      //结点前移
--L.last                                    //修改表长
return OK;
}/*ListDelete */
```

3. 线性表的链式存储结构

第一部分

以链式结构存储的线性表称之为线性链表。线性链表的特点是该线性表中的数据元素可以用任意的存储单元来存储。线性表中逻辑相邻的两元素的存储空间可以是不连续的。为表示逻辑上的顺序关系,对表的每个数据元素除存储本身的信息之外,还需存储一个指示其直接后继的信息。这两部分信息组成数据元素的存储映象,称为结点。线性链表结点的结构类型可定义如下:

```
struct LNode
{
DataType data;          //数据域
struct LNode * next;    //指针域
};
```

其中,data 域是用来存放结点值的数据;next 域是用来存放结点的直接后继的地址(位置)的指针域(链域)。线性链表通过每个结点的指针域将线性表的 n 个结点按其逻辑顺序链接在一起的;每个结点只有一个指针域的线性链表称为单链表(Single Linked List)。

例如:线性表(data1、data2、data3、data4、data5)可用如图 1-4 所示单链表表示。

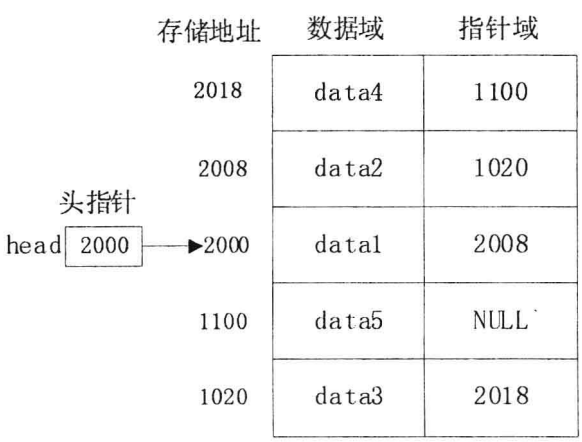


图 1-4　单链表表示线性表

图中 head 是链表的头指针,头指针只相当于结点的指针域,指向链表的第一个结点,链表中的最后一个结点是尾结点,尾结点没有后继结点,其指针域为空,表示链表的结束。

1)单链表简化的图形描述形式,参看图 1-5。

其中,head 是头指针,它指向单链表中的第一个结点,这是单链表操作的入口点。由

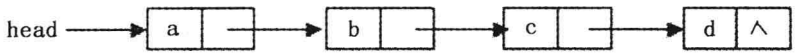


图 1-5　链表

于最后一个结点没有直接后继结点，所以，它的指针域放入一个特殊的值 NULL。NULL 值在图示中常用(^)符号表示。

2)带头结点的单链表

为了简化对链表的操作，人们经常在链表的第一个结点之前附加一个结点，并称为头结点。这样可以免去对链表第一个结点的特殊处理。如图 1-6 所示。

图 1-6　链表

3)链式存储结构的特点

①线性表中的数据元素在存储单元中的存放顺序与逻辑顺序不一定一致，即链式存储逻辑上是连续的，而物理存储空间可能是不连续的；

②在对线性表操作时，只能通过头指针进入链表，并通过每个结点的指针域向后扫描其余结点，这样就会造成寻找第一个结点和寻找最后一个结点所花费的时间不等，具有这种特点的存取方式被称为顺序存取方式。

4. 循环链表

若将链表中最后一个结点的 next 域指向头结点，则为循环链表，参看图 1-7。

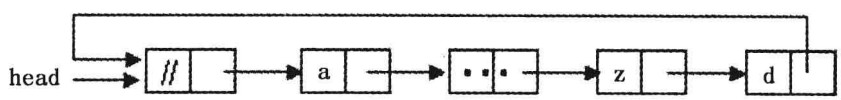


图 1-7　带头结点的循环链表示意图

实现循环链表的类型定义与单链表完全相同，它的所有操作也都与单链表类似。只是判断链表结束的条件有所不同。下面我们就列举两个循环链表操作的算法示例。

在循环链表中，访问结点的特点是访问后继结点，只需要向后走一步，而访问前驱结点，就需要转一圈。

结论：循环链表并不适用于经常访问前驱结点的情况。

解决方法：在需要频繁地同时访问前驱和后继结点的时候，使用双向链表。

5. 双向链表

双向链表也叫双链表，是链表的一种，它的每个数据结点中都有两个指针，分别指向直接后继和直接前驱。所以，从双向链表中的任意一个结点开始，都可以很方便地访问它的前驱结点和后继结点。双向链表的结构如图 1 - 8 所示。

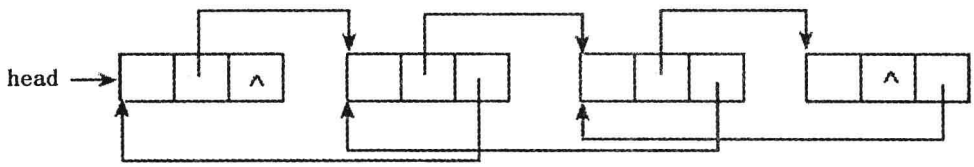


图 1 - 8　双向链表

6. 双向循环链表

双向循环链表是在双向链表基础上，将头尾指针相连，构成循环，成为双向循环链表，参看图 1 - 9。

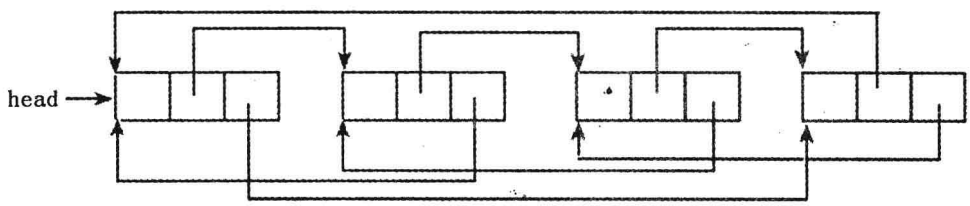


图 1 - 9　双向循环链表

四、栈和队列

1. 栈

1)栈的定义

栈(Stack)是限制在表的一端进行插入和删除运算的线性表，通常称插入、删除的这一端为栈顶(Top)，另一端为栈底(Bottom). 当表中没有元素时称为空栈。

2)栈的性质

假设栈 $S=(a_1,a_2,a_3,\cdots,a_n)$，则 a_1 称为栈底元素，a_n 为栈顶元素。栈中元素按 a_1，$a_2,a_3,\cdots,a_n$ 的次序进栈，退栈的第一个元素应为栈顶元素。换句话说，栈的修改是按后进先出的原则进行的。因此，栈称为后进先出表(LIFO)。

2. 队列

1)队列的定义

队列(Queue)也是一种运算受限的线性表。它只允许在表的一端进行插入，而在另

一端进行删除。允许删除的一端称为队头(front),允许插入的一端称为队尾(rear)。

例如:排队购物,操作系统中的作业排队。

2)队列的性质

先进入队列的成员总是先离开队列。因此队列亦称作先进先出(First In First Out)的线性表,简称 FIFO 表。

当队列中没有元素时称为空队列。在空队列中依次加入元素 $a_1, a_2, \cdots, a_n$ 之后,a_1 是队头元素,a_n 是队尾元素。显然退出队列的次序也只能是 $a_1, a_2, \cdots, a_n$,也就是说队列的修改是依先进先出的原则进行的。

3)循环队列

循环队列是把队列的头和尾在逻辑上连接起来,构成一个环。循环队列中首尾相连,分不清头和尾,此时需要两个指示器分别指向头部和尾部。插入就在尾部指示器的指示位置处插入,删除就在头部指示器的指示位置删除。

五、二叉树

二叉树是一种很重要的非线性数据结构,它的特点是每个结点最多有两个后件,且其子树有左右之分(次序不能任意颠倒)。

1. 二叉树的递归定义和基本形态

1)二叉树的递归定义

二叉树(Binary Tree)是 $n(n \geq 0)$ 个结点的有限集,它或者是空集($n=0$),或者由一个根结点及两棵互不相交的、分别称作这个根的左子树和右子树的二叉树组成。

2)二叉树的五种基本形态

二叉树可以是空集;根可以有空的左子树或右子树;或者左、右子树皆为空。二叉树的五种基本形态如图 1-10 所示。

2. 二叉树的两个特殊形态

1)满二叉树:如果一棵二叉树的任何结点,或者是树叶,或者恰有两棵非空子树,则此二叉树称作满二叉树。可以验证具有 n 个叶结点的满二叉树共有 $2n-1$ 个结点。如图 1-11 所示。

2)完全二叉树:如果一棵二叉树最多只有最下面两层结点度数可以小于 2,并且最下面一层的结点都集中在该层最左边的若干位置上,则称此二叉树为完全二叉树。如图 1-12 所示。其中:0,1,2,3,4 是度为 2 的节点;5 是度为 1 的节点;6,7,8,9,10,11 是度为

 0 的节点(也称叶子节点)。

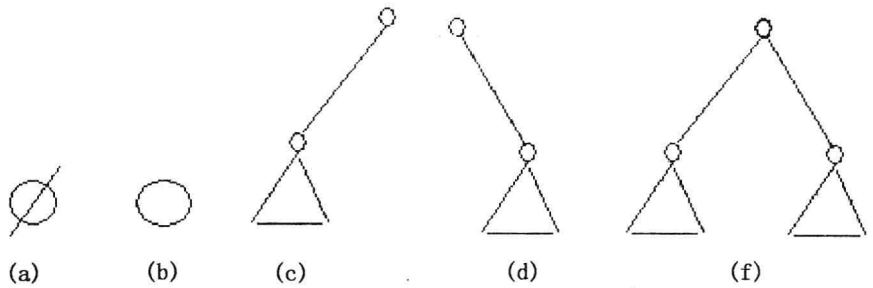


图 1-10 二叉树种类

(a)空二叉树,(b)仅有一个根节点的二叉树,(c)右子树为空的二叉树,(d)左子树为空的二叉树,(f)左右子树均非空的二叉树。

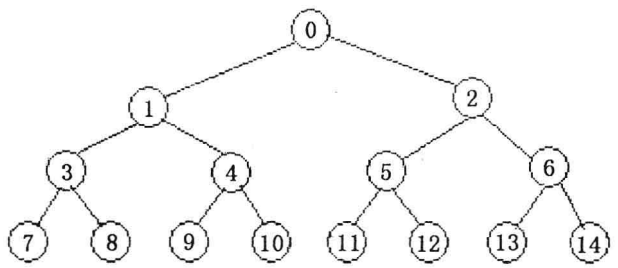


图 1-11 满二叉树

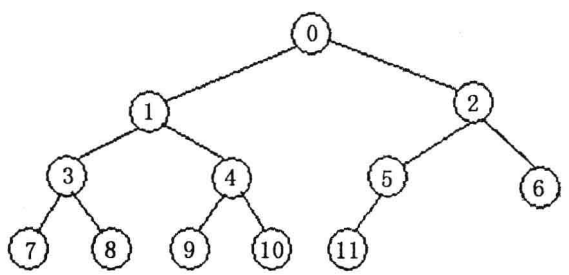


图 1-12 完全二叉树

3. 二叉树的三个主要性质

性质 1:在二叉树的第 $i(\geq 1)$ 层上,最多有 2^{i-1} 个 结点。

证明:我们采用数学归纳法证明:当 $i=1$ 时只有一个根结点,即 $2^{i-1}=2^0=1$,结论成立。假设第 $k(i=k)$ 层上最多有 2^{k-1} 个结点,考虑 $i=k+1$。由归纳假设,在二叉树第 k 层上最多有 2^{k-1} 个结点,而每一个结点最多有两个子结点,因此在第 $k+1$ 层上最多有 $2*2^{k-1}=2^{(k+1)-1}=2^i$,结论成立。综上所述,性质 1 成立。