第二章　　习题与解答

一 判断题

错 1．线性表的逻辑顺序与存储顺序总是一致的。
对 2．顺序存储的线性表可以按序号随机存取。
错 3．顺序表的插入和删除操作不需要付出很大的时间代价，因为每次操作平均只有近一半的元素需要移动。
错 4．线性表中的元素可以是各种各样的，但同一线性表中的数据元素具有相同的特性，因此是属于同一数据对象。
错 5．在线性表的顺序存储结构中，逻辑上相邻的两个元素在物理位置上并不一定紧邻。
对 6．在线性表的链式存储结构中，逻辑上相邻的元素在物理位置上不一定相邻。
对 7．线性表的链式存储结构优于顺序存储结构。
错 8．在线性表的顺序存储结构中，插入和删除时，移动元素的个数与该元素的位置有关。
对 9．线性表的链式存储结构是用一组任意的存储单元来存储线性表中数据元素的。
错 10．在单链表中，要取得某个元素，只要知道该元素的指针即可，因此，单链表是随机存取的存储结构。

二 单选题（请从下列 A，B，C，D 选项中选择一项）

1．线性表是(　A)　。

(A) 一个有限序列，可以为空；　　(B) 一个有限序列，不能为空；
(C) 一个无限序列，可以为空；　　(D) 一个无序序列，不能为空。

2．对顺序存储的线性表，设其长度为 n，在任何位置上插入或删除操作都是等概率的。插入一个元素时平均要移动表中的（C ）个元素。

(A) n/2　　(B)　n+1/2　　(C)　n -1/2　　(D)　n

3．线性表采用链式存储时，其地址(D)　。

(A) 必须是连续的；　　(B) 部分地址必须是连续的；
(C) 一定是不连续的；　　(D) 连续与否均可以。

4．用链表表示线性表的优点是　（ C)。

(A)便于随机存取
(B)花费的存储空间较顺序存储少
(C)便于插入和删除
(D)数据元素的物理顺序与逻辑顺序相同

5. 某链表中最常用的操作是在最后一个元素之后插入一个元素和删除最后一个元素，则采用(D)存储方式最节省运算时间。

(A)单链表

(B)双链表

(C)单循环链表

(D)带头结点的双循环链表

6. 循环链表的主要优点是(D) 。

(A)不在需要头指针了

(B)已知某个结点的位置后，能够容易找到他的直接前趋

(C)在进行插入、删除运算时，能更好的保证链表不断开

(D)从表中的任意结点出发都能扫描到整个链表

7. 下面关于线性表的叙述错误的是(B D)。

(A)线性表采用顺序存储，必须占用一片地址连续的单元；

(B)线性表采用顺序存储，便于进行插入和删除操作；

(C)线性表采用链式存储，不必占用一片地址连续的单元；

(D)线性表采用链式存储，不便于进行插入和删除操作；

8. 单链表中，增加一个头结点的目的是为了（C ）。

(A) 使单链表至少有一个结点 (B)标识表结点中首结点的位置

（C）方便运算的实现 (D) 说明单链表是线性表的链式存储

9. 若某线性表中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素，则采用（ D）存储方式最节省运算时间。

(A) 单链表 (B) 仅有头指针的单循环链表

(C) 双链表 (D) 仅有尾指针的单循环链表

10. 若某线性表中最常用的操作是取第 i 个元素和找第 i 个元素的前趋元素，则采用（ ）存储方式最节省运算时间（B ）。

(A) 单链表 (B) 顺序表

(C) 双链表 (D) 单循环链表

三 填空题

1.带头结点的单链表 H 为空的条件是_____H->next==NULL _____。

2.非空单循环链表 L 中*p 是尾结点的条件是_____p->next==L_____。

3. 在一个单链表中 p 所指结点之后插入一个由指针 f 所指结点，应执行 f->next=___p->next_____;和 p->next=_____f_____的操作。

4.在一个单链表中 p 所指结点之前插入一个由指针 f 所指结点，可执行以下操作：

s->next=____ p->next ____;

p->next=s;

t=p->data;

p->data=_____ s->data _____;

s->data=_____t_____;

5.在顺序表中做插入操作时首先检查_____是否满_____。

四　算法设计题

1. 设线性表存放在向量 A[arrsize]的前 elenum 个分量中，且递增有序。试写一算法，将 x 插入到线性表的适当位置上，以保持线性表的有序性。并且分析算法的时间复杂度。
2. 已知一顺序表 A，其元素值非递减有序排列，编写一个函数删除顺序表中多余的值相同的元素。
3. 编写一个函数，从一给定的顺序表 A 中删除值在 x~y(x<=y)之间的所有元素，要求以较高的效率来实现。

   提示：可以先将顺序表中所有值在 x~y 之间的元素置成一个特殊的值，并不立即删除它们，然后从最后向前依次扫描，发现具有特殊值的元素后，移动其后面的元素将其删除掉。
4. 线性表中有 n 个元素，每个元素是一个字符，现存于向量 R[n]中，试写一算法，使 R 中的字符按字母字符、数字字符和其它字符的顺序排列。要求利用原来的存储空间，元素移动次数最小。（研 54）
5. 线性表用顺序存储，设计一个算法，用尽可能少的辅助存储空间将顺序表中前 m 个元素和后 n 个元素进行整体互换。即将线性表

   （$a_1, a_2, \dots, a_m, b_1, b_2, \dots, b_n$）　　改变为：

   （$b_1, b_2, \dots, b_n, a_1, a_2, \dots, a_m$）。
6. 已知带头结点的单链表 L 中的结点是按整数值递增排列的，试写一算法，将值为 x 的结点插入到表 L 中，使得 L 仍然有序。并且分析算法的时间复杂度。
7. 假设有两个已排序的单链表 A 和 B，编写一个函数将他们合并成一个链表 C 而不改变其排序性。
8. 假设长度大于 1 的循环单链表中，既无头结点也无头指针，p 为指向该链表中某一结点的指针，编写一个函数删除该结点的前趋结点。
9. 已知两个单链表 A 和 B 分别表示两个集合，其元素递增排列，编写一个函数求出 A 和 B 的交集 C，要求 C 同样以元素递增的单链表形式存储。
10. 设有一个双向链表，每个结点中除有 prior、data 和 next 域外，还有一个访问频度 freq 域，在链表被起用之前，该域其值初始化为零。每当在链表进行一次 Locata(L,x)运算后，令值为 x 的结点中的 freq 域增 1，并调整表中结点的次序，使其按访问频度的递减序列排列，以便使频繁访问的结点总是靠近表头。试写一个算法满足上述要求的 Locata(L,x)算法。

五 上机实习题目

1．**Josephu 问题**

Josephu 问题为：设编号为 1，2，… n 的 n 个人围坐一圈，约定编号为 k（1<=k<=n）的人从 1 开始报数，数到 m 的那个人出列，它的下一位又从 1 开始报数，数到 m 的那个人又出列，依次类推，直到所有人出列为止，由此产生一个出队编号的序列。

提示：用一个不带头结点的循环链表来处理 Josephu 问题：先构成一个有 n 个结点的单循环链表，然后由 k 结点起从 1 开始计数，计到 m 时，对应结点从链表中删除，然后再从被删除结点的下一个结点又从 1 开始计数，直到最后一个结点从链表中删除算法结束。

2．**一元多项式的相加**

提示：

（1） 一元多项式的表示问题：对于任意一元多项式：

$$P_n(x)= P_0+ P_1X^1+ P_2X^2+ \dots + P_iX^i+ \dots + P_nX^n$$

可以抽象为一个由“系数-指数”对构成的线性表，且线性表中各元素的指数项是递增的：

$$P=((P_0,0), (P_1,1), (P_2,2), \dots , (P_n,n))$$

(2) 用一个单链表表示上述线性表，结点结构为：

```
typedef   sturct   node
{   float   coef;   /*系数域*/
    int       exp;   /*指数域*/
    struct node     *next;   /*指针域*/
} Ploy Node;
```

coef	exp	next

约瑟夫环问题

约瑟夫环问题：设编号为 1，2，3，……，n 的 n(n>0)个人按顺时针方向围坐一圈，每个人持有一个正整数密码。开始时任选一个正整数做为报数上限 m，从第一个人开始顺时针方向自 1 起顺序报数，报到 m 是停止报数，报 m 的人出列，将他的密码作为新的 m 值，从他的下一个人开始重新从 1 报数。如此下去，直到所有人全部出列为止。令 n 最大值取 30。要求设计一个程序模拟此过程，求出出列编号序列。

源程序代码：（在 Tubro C 2.0 测试通过）

```
#include <stdlib.h>
#include <alloc.h>

struct node
{
```

```
    int number; /*  人的序号  */
    int cipher; /*  密码  */
    struct node *next; /*  指向下一个节点的指针  */
};

struct node *CreatList(int num) /*  建立循环链表  */
{
    int i;
    struct node *ptr1,*head;

    if((ptr1=(struct node *)malloc(sizeof(struct node)))==NULL)
    {
        perror("malloc");
        return ptr1;
    }
    head=ptr1;
    ptr1->next=head;
    for(i=1;i<num;i++)
    {
        if((ptr1->next=(struct node *)malloc(sizeof(struct node)))==NULL)
        {
            perror("malloc");
            ptr1->next=head;
            return head;
        }
        ptr1=ptr1->next;
        ptr1->next=head;
    }
    return head;
}

main()
{
    int i,n=30,m; /*  人数 n 为 30 个  */
    struct node *head,*ptr;
    randomize();
    head=CreatList(n);
    for(i=1;i<=30;i++)
    {
```

```
        head->number=i;
        head->cipher=rand();
        head=head->next;
    }
    m=rand(); /* m 取随机数 */
    i=0; /* 因为我没办法删除 head 指向的节点，只会删除 head 的下一节点，所以只能从
0 数起。*/
    while(head->next!=head) /* 当剩下最后一个人时，退出循环 */
    {
        if(i==m)
        {
            ptr=head->next; /* ptr 记录数到 m 的那个人的位置 */
            printf("number:%d\n",ptr->number);
            printf("cipher:%d\n",ptr->cipher);
            m=ptr->cipher; /* 让 m 等于数到 m 的人的密码 */
            head->next=ptr->next; /* 让 ptr 从链表中脱节，将前后两个节点连接起来 */
            head=hea/d->next; /* head 移向后一个节点 */
            free(ptr); /* 释放 ptr 指向的内存 */
            i=0; /* 将 i 重新置为 0，从 0 再开始数 */
        }
        else
        {
            head=head->next;
            i++;
        }
    }
    printf("number:%d\n",head->number);
    printf("cipher:%d\n",head->cipher);
    free(head); /* 让最后一个人也出列 */
}
```

第三章 习题

一、 基本题

1．填空：线性表、栈和队列都是__线性___结构，可以在线性表的___任何__位置插入和删除元素，对于栈只能在___栈顶____位置插入和删除元素，对于队只能在___队尾___位置插入和___队头___位置删除元素。

2. 栈和队列数据结构的特点，什么情况下用到栈，什么情况下用到队列？

3．设有编号为 1，2，3，4 的四辆车，顺序进入一个栈式结构的站台，试写出这四辆车开出车站的所有可能的顺序（每辆车可能入站，可能不入站，时间也可能不等）。

1234,1243,1324,1342,1432,2134,2143,2314,2341,2431,3214,3241,3421,4321.

4．试证明：若借助栈由输入序列 1，2，…，n 得到输出序列为 $p_1p_2…p_n$ (它是输入序列的一个排列)，则在输出序列中不可能出现这样的情形：存在着 $i<j<k$，使得 $p_j<p_k<p_i$ 。

存在着 $i<j<k$，使得 $p_k<p_i<p_j$

二、 设计算法题

1. 假设称正读和反读都相同的字符序列为“回文”，例如，“abcddcba”、 “qwerewq”是回文，“ashgash”不是回文。是写一个算法判断读入的一个以‘@’为结束符的字符序列是否为回文。
2. 设以数组 se[m]存放循环队列的元素，同时设变量 rear 和 front 分别作为队头队尾指针，且队头指针指向队头前一个位置，写出这样设计的循环队列入队出队的算法。
3. 假设以数组 se[m]存放循环队列的元素，同时设变量 rear 和 num 分别作为队尾指针和队中元素个数记录，试给出判别此循环队列的队满条件，并写出相应入队和出队的算法。
4. 假设以带头结点的循环链表表示一个队列，并且只设一个队尾指针指向尾元素结点（注意不设头指针），试写出相应的置空队、入队、出队的算法。
5. 设计一个算法判别一个算术表达式的圆括号是否正确配对。
6. 写一个算法，借助于栈将一个单链表置逆。
7. 两个栈共享向量空间 v[m]，它们的栈底分别设在向量的两端，每个元素占一个分量，试写出两个栈公用的栈操作算法：push(i,x)、pop(i)和 top(i)，i=0 和 1 ,用以指示栈号。

三、 实验题目

1. 背包问题的求解：
   假设有一个能装入总体积为 T 的背包和 n 件体积分别为 $w_1, w_2, \ldots, w_n$ 的物品，能否从 n 件物品中挑选若干件恰好装满背包，即使 $w_1+w_2+\ldots+w_n=T$，要求找出所有满足上述条件的解。例如：当 T=10，各件物品的体积{1，8，4，3，5，2}时，可找到下列 4 组解：（1，4，3，2）
   （1，4，5）
   （8，2）
   （3，5，2）。

提示：可利用回溯法的设计思想来解决背包问题。首先将物品排成一列，然后顺序选取

物品装入背包，假设已选取了前 i 件物品之后背包还没有装满，则继续选取第 i+1 件物品，若该件物品“太大”不能装入，则弃之而继续选取下一件，直至背包装满为止。但如果在剩余的物品中找不到合适的物品以填满背包，则说明“刚刚”装入背包的那件物品“不合适”，应将它取出“弃之一边”，继续再从“它之后”的物品中选取，如此重复，，直至求得满足条件的解，或者无解。

由于回溯求解的规则规则是“后进先出”因此自然要用到栈。

2. 模拟停车厂管理的问题。

设停车厂只有一个可停放几辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达的先后顺序依次排列，若车场内已停满几辆汽车，则后来的汽车只能在门外的便道上等候，一旦停车场内有车开走，则排在便道上的第一辆车即可进入；当停车场内某辆车要离开时，由于停车场是狭长的通道，在它之后开入的车辆必须先退出车场为它让路，待该辆车开出大门后，为它让路的车辆再按原次序进入车场。在这里假设汽车不能从便道上开走。试设计一个停车场管理程序。

第四章习题

算法设计题

1．利用 C 的库函数 strlen,strcpy 和 strcat 写一个算法 void StrInsert(char *S,char *T,int t) ，将串 T 插入到 S 的第 i 个位置上。若 i 大于 S 的长度，则插入不执行。

2.利用 C 的库函数 strlen,strcpy（或 strncpy）写一个算法 void StrDelete(char *S,int t,int m) ，删除串 S 中从位置 I 开始的连续的 m 个字符。若 i≥strlen(S)，则没有字符被删除；若 i+m≥strlen(S)，则将 S 中从位置 i 开始直至末尾的字符均被删去。

3．采用顺序结构存储串，编写一个函数，求串和串的一个最长的公共子串。

4．采用顺序存储结构存储串，编写一个函数计算一个子串在一个字符串中出现的次数，如果该子串不出现则为 0。

第五章习题

一、单项选择题

1. 二维数组 M 的成员是 6 个字符（每个字符占一个存储单元）组成的串，行下标 i 的范围从 0 到 8，列下标 j 的范围从 1 到 10，则存放 M 至少需要（1）个字节；M 的第 8 列和第 5 行共占（2）个字节；若 M 按行优先方式存储，元素 M[8][5]的起始地址与当 M 按列优先方式存储时的（3）元素的起始地址一致。（）

（1） A.90 B.180 C.240 D.540

（2） A.108 B.114 C.54 D.60

（3） A.M[8][5] B.M[3][10] C.M[5][8] D.M[0][9]

2. 二维数组 M 的元素是 4 个字符（每个字符占一个存储单元）组成的串，行下标 i 的范围从 0 到 4，列下标 j 的范围从 0 到 5，M 按行存储时元素 M[3][5]的起始地址与 M 按列存储时元素（1）的起始地址相同。（）

A.m[2][4] B.M[3][4] C.M[3][5] D.M[4][4]

3. 数组 A 中，每个元素 A 的存储占 3 个单元，行下标 i 从 1 到 8，列下标 j 从 1 到 10，从首地址 SA 开始连续存放在存储器内，存放该数组至少需要的单元个数是（1），若该数组按行存放时，元素 A[8][5]的起始地址是（2），若该数组按列存放时，元素 A[8][5]的起始地址是（ 3 ）。

（1） A. 80 B.100 C.240 D.270

（2） A.SA+141 B.SA+144 C.SA+222 D.SA+225 7*10+5=75*3-3

（3） A.SA+141 B.SA+180 C.SA+222 D.SA+225 7*8+5=61*3-3

4. 稀疏矩阵一般的压缩存储方法有两种，即（）

A.二维数组和三维数组 B. 三元组和散列

C.三元组和十字链表 D. 散列和十字链表

5.若采用三元组压缩技术存储稀疏矩阵，只要把每个元素的行下标和列下标互换，就完成了对该矩阵的转置运算，这种观点（）

A.正确 B.错误

6.假设按行优先存储整数数组 A[9][3][5][8]时，第一个元素的字节地址时 1 0 0，每个整数占 4 个字节。问下列元素的存储地址是什么。

(1) a_{0000} (2)a_{1111} (3)a_{3125} (4)a_{8247}

7.设有三对角矩阵 $A_{n\times n}$，将其三条对角线上的元素存于数组 B[3][n]中，使得元素 B[u][v]=a_{ij}，试推倒出从（i,j）到 (u,v)的下标变换公式。

8.假设一个准对角矩阵：

$$\begin{pmatrix} a_{11} & a_{12} & & & & & & & \\ a_{21} & a_{22} & & & & & & & \\ & & a_{33} & a_{34} & & & & & \\ & & a_{43} & a_{44} & & & & & \\ & & & & \cdots & & & & \\ & & & & & a_{ij} & & & \\ & & & & & & a_{2m-1,2m-1} & a_{2m-1,2m} \\ & & & & & & a_{2m,2m-1} & a_{2m,2m} \end{pmatrix}$$

按以下方式存储于一维数组 B[4m]中:

0	1	2	3	4	5	6	…	k	…	4m-1	4m
A_{11}	a_{12}	a_{21}	a_{22}	a_{33}	a_{34}	a_{43}	…	a_{ij}	…	$a_{2m-1,2m}$	$a_{2m,2m-1}$	$a_{2m,2m}$

写出由一对下标（i,j）求 k 的转换公式。

9.画出下列广义表的存储结构式意图。

（１） A=((a,b,c),d,(a,b,c))

（２） B=(a,(b,(c,d),e),f)

二、算法设计

1．对于二维数组 A[m][n],其中 m<=80,n<=80，先读入 m,n，然后读该数组的全部元素，对如下三种情况分别编写相应函数：

(1)求数组 A 靠边元素之和

(2)求从 A[0][0]开始的互不相邻的各元素之和

(3)当 m=n 时，分别求两条对角线的元素之和，否则打印 m!=n 的信息

2.有数组 A[4][4]，把 1 到 16 个整数分别按顺序放入 A[0][0]...A[0][3]，A[1][0]...A[1][3] A[2][0]...A[2][3]，A[3][0]...A[3][3]中，编写一个函数获取数据并求出两条对角线元素的乘积。

3．只猴子要选大王，选举办法如下：所有猴子按 1，2，...，n 编号围坐一圈，从第 1 号开始按 1、2、...、m 报数，凡报 m 号的退出圈外，如此循环报数，直到圈内剩下一只猴子时，这只猴子就是大王。n 和 m 由键盘输入，打印出最后剩下的猴子号。编写一个程序实现上述函数。

4．如果矩阵 A 中存在这样的一个元素 A[i][j]满足下列条件：A[i][j]是第 i 行中值最小的元素，且又是第 j 列中最大的元素，则称之为该矩阵的一个马鞍点。编写一个函数计算出 m*n 的矩阵 A 的所有马鞍点。

5．现有如下的稀疏矩阵 A（如图所示），要求画出以下各种表示方法。

（1）三元组表示法

（2）十字链表法

15	0	0	22	0	-15
0	13	3	0	0	0
0	0	0	-6	0	0
0	0	0	0	0	0
91	0	0	0	0	0
0	0	28	0	0	0

6．假设稀疏矩阵 A 和 B（具有相同的大小 m*n）都采用三元组表示，编写一个函数计算 C=A+B，要求 C 也采用三元组表示。

7．假设稀疏矩阵 A 和 B（分别为 m*n 和 n*1 矩阵）采用三元组表示，编写一个函数计算 C=A*B，要求 C 也是采用稀疏矩阵的三元组表示。

8．假设稀疏矩阵只存放其非 0 元素的行号、列号和数值，以一维数组顺次存放，行号为-1 结束标志。

例如：如图所示的稀疏矩阵 M，则存在一维数组 D 中：

D[0]=1， D[1]=1，D[2]=1，D[3]=1，D[4]=5

D[5]=10，D[6]=3，D[7]=9，D[8]=5，D[9]= -1

M =:

1	0	0	0	10	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	5
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

现有两个如上方法存储的稀疏矩阵 A 和 B，它们均为 m 行 n 列，分别存放在数组 A 和 B 中，编写求矩阵加法 C=A+B 的算法，C 亦放在数组 C 中。

9．已知 A 和 B 为两个 n*n 阶的对称矩阵，输入时，对称矩阵只输入下三角形元素，按压缩存储方法存入一维数组 A 和 B 中，编写一个计算对称矩阵 A 和 B 的乘积的函数。

10．假设 L 为非递归并且不带共享子表的广义表，设计一个复制广义表 L 的算法。

同步综合练习及参考答案

（一） 基础知识题

6.1 加设在树中，结点 x 是结点 y 的双亲时，用(x,y)来表示树变。已知一棵树边的集合为：{(i,m),(i,n),(b,e),(e,i),(b,d),(a,b),(g,i),(g,k),(c,g),(c,f),(h,l),(c,h),(a,c)}用树形表示法画出此树，并回答下列问题：

（1） 哪个是根结点？

（2） 哪些是叶结点？

（3） 哪个是 g 的双亲？

（4） 哪些是 g 的祖先？

（5） 哪些是 g 的孩子？

（6） 哪些是 e 的子孙？

（7） 哪些是 e 的兄弟？哪些是 f 的兄弟？

（8） 结点 b 和 n 的层次各是多少？

（9） 树的深度是多少？

（10） 以结点 c 为根的子树的深度是多少？

（11） 树的度数是多少？

解：

（1） a 是根结点。

（2） m,n,j,k,l 是叶结点。

（3） c 是 g 的双亲。

（4） a,c 是 g 的祖先。

（5） g 的孩子是 j,k.。

（6） e 的子孙是 i,m,n.。

（7） e 的兄弟是 a,f 的兄弟是 g。

（8） h,b 在第五层。

（9） 树的深度为 3。

（10） 以 C 为根的子树的深度为 3。

（11） 树的度数为 3。

6．2 一棵度为 2 的有序属于一棵二叉树有何区别？

解：

区别：度为 2 的树有二个分支，没有左右之分；以可二叉树也有两个分支，但有左右之分，且左右不能交换。

6．3 试分别画出具有 3 个结点的树和 3 个结点的二叉树的所有不同形态。

解：

3 个结点树形态：

3 个结点的二叉树：

6．4 已知一棵树为 m 的树中有 n_1 个度为 1 的结点，n2 个度为 2 的结点，…nm 个度为 m 的结点，问该书中有多少片叶子？

解：

因为 $n_1+n_2+\ldots+n_M+n_0+=1+n_1+2n_2+\ldots+mn_M$

$=>n_0+=1+n_2+\ldots+(m-1)n_M$

6.5 一个深度为 h 的满 k 叉树有如下性质：第 h 层上的结点都是叶子结点，其余各层上每个结点都有 k 棵非空子树。如果按层次顺序（同层自左至右）从未有过开始对全部结点编号，问：

（1） 各层的结点数目是多少？
（2） 编号为 i 的结点的双亲结点（若存在）的编号是多少？
（3） 编号为 i 的结点的第 j 个孩子结点（若存在）的编号是多少？
（4） 编号为 i 的结点有右兄弟的条件是什么？其右兄弟的编号是多少？

解：

（1） K_{i-1}

（2）

（3） K_{i+j-1}

（4） (i-1)MOD K<>0 ,i+1

6．6 高度为 h 的完全二叉树至少有多少个结点？至多有多少个结点？

解：

至少有：2^{h-1}， 至多有：2^h-1

6．7 在具有 n 个结点的 k 叉树（k≥2）的 k 叉树链表表示中，有多少个空指针？

解：

（k-1）n+1 个空指针

6．8 假设二叉树包含的结点数据为 1，3，7，2，12。

（1） 画出两棵高度最大的二叉树；

（2） 画出两棵完全二叉树，要求每个双亲结点的值大于其孩子结点的值。

6．9 试找出分别满足下面条件的所有二叉树：

（1）前序序列和中序序列相同； （2）中序序列和后序序列相同；

（3）前序序列和后序序列相同； （4）前序、中序、后序序列均相同。

解：

（1） 空二叉树或任一结点均无左子树的非空二叉树

（2） 空二叉树或任一结点均无左子树的非空二叉树

（3） 空二叉树或仅有一个结点的二叉树

（4） 同（3）

6．10 试采用顺序存储方法和链接存储方法分别画出 6.30 所示各二叉树的存储结构。

解：①顺序存储：

(a)1 2 Φ Φ 3 Φ Φ Φ Φ 4 Φ Φ Φ Φ Φ Φ Φ Φ Φ Φ 5

(b)1 Φ 2 Φ Φ 3 Φ Φ Φ Φ Φ Φ Φ 4 Φ Φ Φ Φ Φ Φ Φ Φ Φ Φ Φ Φ 5

(c)1 Φ 2 Φ Φ 3 4 Φ Φ Φ Φ 5 6 Φ Φ Φ Φ Φ Φ Φ Φ Φ Φ Φ 7 8

(d)1 2 3 4 Φ 5 6 Φ 7 Φ Φ Φ Φ 8 9

②连接存储：

6．11 分别写出图 6.30 所示各二叉树的前序、中序和后序序列。

解：

6．12 若二叉树中个结点的值均不相同，则由二叉树的前序序列和中序序列，或由其后序序列的中序列均能惟一地确定一棵二叉树，但由前序序列和后序序列却不一定能惟一地确定一棵二叉树。

（1） 已知一棵二叉树的前序序列和中序序列分别为 ABDGHCEFI 和 GDHBAECIF，请画出此二叉树。

（2） 已知一棵二叉树的中序序列和后序序列分别为 BDCEAFHG 和 DECBHGFA，请画出此二叉树。

（3） 已知两棵二叉树前序序列和后序序列均为 AB 和 BA，请画出这两棵不同的二叉树。

解：

6．13 对二叉树中结点进行按层次顺序（每一层自左至右）的访问操作称为二叉树的层次遍历，遍历所得到的结点序列称为二叉树的层次序列。现已知一棵二叉树的层次序列为ABCDEFGHIJ，中序序列为 DBGEHJACIF，请画出该二叉树。

解：

6．14 试画出图 6.30 所示各二叉树的前序、中序和后序线索树及相应的线索链表。

解：

（以 c 为例）

① 前序：1 2 3 5 7 8 6 4

② 前序：1 7 5 8 3 6 2 4

6．15 在何种线索树中，线索对所求指定结点在相应次序下的前趋和后继并无帮助？

解：

在前序线索树中找某一点的前序前趋以及在后序线索树中寻找某一点的后继，线索并无多大帮助。

6．16 对图 6.31 所示的森林：

（1） 求各树的前序序列和后序序列：

（2） 求森林的前序序列和后序序列：

（3） 将此森林转换为相应的二叉树：

（4） 给出（a）所示树的双亲链表表示、孩子链表表示、双亲孩子链表表示及孩子兄弟链表表示等四种存储结构，并指出哪些存储结构易于求指定结点的祖先，哪些易于求指定结点的后代？

解：

(1)

	a	b	c
前序	ABCDEF	GHIJK	LMPQRNO
后序	BDEFCA	IJKHG	QRPMNOL

(2) 前序： ABCDEFGHIGKLMPQRNO

后序： BDEFCAIJKHGQRPMNOL

(3) 二叉树

(4) 1

① 孩子链表表示发：

② 双亲链表表示发：

结点	0	1	2	3	4	5	6
data		A	B	C	D	E	F
parent	-1	0	1	1	3	3	3

③ 双亲孩子链表：

④ 孩子兄弟链表表示：

⑤ 易于求祖先：双亲链表面 双亲孩子

⑥ 易于求后代：孩子链表 双亲孩子

6．17 画出图 6.32 所示的各二叉树所应的森林

6．18 高度为 h 的严格二叉树至少有多少个结点？至多有多少个结点？

解：

最多有 2^n-1 最少有 2^{n-1}

6．19 在什么样的情况下，等长编码是最优的前缀码？

解：

当字符集中的各字符使用频率均匀时。

6．20　下属编码哪一组不是前缀码？

{00，01，10，11}，{0，1，00，11}，{0，10，110，111}

解：

因为前缀码中不可能存在一个元素是另一个的前面部分。　所以第二组不是。

6．20　假设用于通信的电子由字符集{a,b,c,d,e,f,g,h}中的字母构成，这 8 个字母在电文中出现的概率分别为{0.07，0.19，0.02，0.06，0.32，0.03，0.21，0.10}

（1）　为这 8 个字母设计哈夫曼编码。

（2）　若用三位二进制数（0~7）对这个 8 个字母进行等长编码，则哈夫曼编码的平均码长是等长编码的百分之几？它使电文总长平均压缩多少？

解：

①

②哈夫曼编码码长：

4*0.07+2*0.19+5*0.02+4*0.06+5*0.03+2*0.21+4*0.1=2.71

等长码长：　3

905　平均缩了 10%

（二）　算法设计题

6.22 二叉树的遍历算法可写为通用形式。例如，通用的中序遍历为：

```
void Inorder(BinTree T,void(*Visit)(Datatype x))
{ if (T)
    {Inorder(T->lchild,Visit);          /*遍历左子树*/
     Visit(T->data);          /*通过函数指针调用它所指的函数来访问结点*/
     Inorder(T->rchild,Visit);            /*遍历右子树*/
     }
}
```

其中 Visit 是一个函数指针，它指向形如 void f（DdataType x）的函数。因此我们可以将访问结点的操作写在函数 f 中，通过调用语句 Inorder（root，f）将 f 的地址传递给 Visit，来执行遍历操作。请写一个打印结点的数据的函数，通过调用上述算法来完成书中 6.3 节的中序遍历。

解：

```
#include“stdio.h”
#define Null 0
typedef char DataType;
typedef struct node
  {
    DataType data;
    Struct node lchild,rchild;
    }BinTree;
BinTree *root;
BinTree *Q[100];
BinTree CreateBinTree()            /*建立二叉树*/
{
 char ch;
 int front,rear;
```

```
        BinTree root,s;
        Root=Null;
        front=1;
        rear=0;
        ch=getchar();
        while(ch!=' #' )
         {
           s=Null;
           if(ch!=' @' )
             {
              s=(BinTree*)malloc(sizeof(BinTree));
              s->data=ch;
              s->lchild=Null;
              s->rchild=Null;
              }
             rear ++;
             Q[rear]=s;
             if(rear==1) root=s;
             else
             {
               if(s&&Q[front])
                  if(rear%2==0)  Q[front]->lchild=s;
                  else
                   Q[front]->rchild=s;
               if(rear%2==1) front++;
              }
              ch=getchar();
             }
             return root;
            }

main()
{
 root=CreateBinTree();
 Inorder(root);
}
```

① 中序遍历法之一

```
Inorder(BinTree *t)
{
  if(t)
   {
     Inorder(t->lchild);
     Visit(t->data);
```

```
    Inorder(t->rchild);
   }
  }

  Vist(int i)
  {
   printf(“%c”,i);
  }
```

② 中序遍历法之二

```
Inorder(BinTree *t)
{
 if(t)
  {
   Inorder(t->lchild);
   printf(“%c”,t->data);
   Inorder(t->rchild);
  }
}
```

6.23 以二叉链表为存储结构，分别写出求二叉树结点总数及叶子总数的算法。

解：

① 计算结点总数

```
int CountNode(BinTree *root)
{
 int num1,num2;
 if(root==Null) return(0);
 else if(root->lchild==Null&&rooot->rchild==Null)
 return(1);
 else
  {
   num1=CountNode(root->lchild);
   num2=CountNode(root->rchild);
   return(num1+num2+1);
  }
}
```

② 计算叶子总数

```
int CountLeafs(BinTree *root)
{
 int num1,num2;
 if(root==Null) return(0);
 else if(root->lchild==Null&&root->rchild==Null)
 return(1);
 else
```

```
    {
      num1=CountLeafs(root->lchild);
      num2=CountLeafs(root->rchild);
      return(num1+num2);
    }
  }
```

6.24 以二叉链表为存储结构，分别写出求二叉树高度及宽度的算法。所谓宽度是指在二叉树的各层上，具有结点数最多的那一层上的结点总数。

解：

① 求树的高度

思想：对非空二叉树，其深度等于左子树的最大深度加 1。

```
Int Depth(BinTree *T)
{
  int dep1,dep2;
  if(T==Null) return(0);
  else
     {
      dep1=Depth(T->lchild);
      dep2=Depth(T->rchild);
      if(dep1>dep2) return(dep1+1);
      else return(dep2+1);
}
```

② 求树的宽度

思想：按层遍历二叉树，采用一个队列 q，让根结点入队列，最后出队列，若有左右子树，则左右子树根结点入队列，如此反复，直到队列为空。

```
int Width(BinTree *T)
 {
  int front=-1,rear=-1;              /* 队列初始化*/
  int flag=0,count=0,p;
  /* p 用于指向树中层的最右边的结点，标志 flag 记录层中结点数的最大值。*/
   if(T!=Null)
    {
     rear++;
     q[rear]=T;
     flag=1;
     p=rear;
    }
   while(front<p)
    {
     front++;
     T=q[front];
     if(T->lchild!=Null)
      {
```

```
            rear++;
            q[rear]=T->lchild;
            count++;
           }
          if(T->rchild!=Null)
           {
            rear++;
            q[rear]=T->rchild;
            count++;
           }
          if(front==p)                          /* 当前层已遍历完毕*/
           {
            if(flag<count) flag=count;
            count=0;
            p=rear;                             /* p 指向下一层最右边的结点*/
           }
         }                                      /* endwhile*/
       return(flag);
      }
```

6.25 以二叉链表为存储结构，写一算法交换各结点的左右子树。

解：

思想：借助栈来进行对换。

```
Swap(BinTree*T)
{
  BinTree  *stack[100], *temp;
  int top=-1;
  root=T;
  if(T!=Null)
}
 top++;
 stack[top]=T;
 while(top>-1)
 {
  T=stack[top];
  top--;
  if(T->child!=Null||T->rchild!=Null)
  {                                 /*交换结点的左右指针*/
   temp=T->lchild;
   T->lchild=T->rchild;
   T->rchild=temp;
  }
 if(T->lchild!=Null)
  {
   top++;
```

```
        stack[top]=T->lchild;
       }
      if(T->rchild!=Null)
       {
        top++;
        stack[top]=T->rchild;
       }
      }
     }                              /*endwhile*/
    }                               /*endif*/

main()
{
 int I,j,k,l;
 printf(“\n”);
 root=CreateBinTree();
 Inorder(root);
 i=CountNode(root);
 j=CountLeafs(root);
 k=Depth(root);
 l=Width(root);
 printf(“\nThe Node ’ s Number:%d”,i);
 printf(“\nThe Leafs’ s Number:%d”,j);
 printf(“\nThe Depth is:%d”,k);
 printf(“\nThe width is:%d”,l);
 Swap(root);
 Printf(“\nThe swapTree is:”);
Inorder(root);
}
```

6.26 以二叉表为存储结构,写一个拷贝二叉表的算法 哦(BinTree root,BinTree *newroot),其中新树的结点是动态申请的，为什么 newroot 要说明为 BinTree 形指针的指针？

解：

```
     CopyTree(BinTree root,BinTree *(newroot))
     }
      if(root!=Null)
       {
        *newroot=(BinTree *)malloc(sizeof(BinTree));
        (*newroot)->data=root->data;
        CopyTree(root->lchild,&(*newroot)->lchild);
        CopyTree(root->rchild,&(*newroot)->rchild);
        Inorder(*newroot);
       }
      else return(Null);
     }
```

```
main()
  {
   BinTree *newroot;
   int I,j,k,l;
   printf(“\n”);
   root=CreateBinTree();
   Inorder(root);
   Printf(“\n”);
   /*Swap(root);*/
   &(*newroot)=Null;
   CopyTree(root,&*newroot);
  }
```

6．27 以二叉树表为存储结构，分别写处在二叉树中查找值为 x 的结点在树中层数的算法。

解：

```
int h=-1,lh=1,count=0;charx=’ c’ ;   /*赋初值*/
Level(BinTree T,int h,int lh)        /*求 X 结点在树只的层树*/
 {
   if(T==Null)h=0;
   else if (T->data==x)
    {
      h=lh;
      count=h;
    }
   else
    {
      h++;
      Level(T->lchild,h,lh);
      If(h==-1)Level(T->rchild,h,lh);
    }
 }


main()
 {
   BinTree *(*newroot);
   Printf(“\n”);
   Root=CreateBinTree();
   Inorder(root);
   Printf(“\n”);
   Level(root,h,lh);
   Printf(“%d”,count);
 }
```

6.28 一棵 n 个结点的完全二叉树以向量作为存储结构，试写一非递归算法实现对该树的前序遍历。

解：

思想：采用栈，先让跟结点如栈，然后退栈，如有左右孩子，则先让右孩子如栈，然后左孩子如栈，如此反复实现前序遍历。

```
typedef struct
   {
    int data[100];
    int top;
    }seqstack;
    seqstack *s;
Perorder(char a[],int n)
 {
  int i=1,count=1;
  s->top=-1;
 if(n==0)return(0);
 else
  {
    if(I<=n)
     {
       s->top++;
       s->data[s->top]=a[I];
     }
    while(count<n)
     {
       printf(“%c”,s->data[s->top]);
       count++;
       s->top--;
       if(s->data[s->top]);==a[i])
         {   /*若栈顶结点为a[i]结点，则退栈，保证父结点比孩子结点先退栈 */
           printf(“%c”,s->data[s->top]);
           count++;
           s->top--;
         }
       if((2*i+1)<n)
         {
           i=2*i;
           s->top++;
           s->data[s->top]=a[i+1];
           s->top++;
       s->data[s->top]=a[i];
     }
    else if(a*i<n)
      {
       i=2*i;
       s->top++;
       s->data[s->top]=a[i];
```

```
            }
          else if(i/2%2==1)i=i/2/2+1;
              /*父结点没有右兄弟，回到祖父结点大右兄弟*/
          else i=i/2+1;       /*回到父结点的右兄弟*/
         }
       }
     }
  main()
   {
     char A[]=“kognwyuvb”;
     int n=strlen(A);
     s=(seqstack *)malloc(sizeof(seqstack));
     printf(“\n”);
     Perorder(A,n);
  }
```

6．29 以二叉树表为存储结构，写一算法对二叉树进行层次遍历（定义见习题 6.13）。提示：应使用队列来保存各层的结点。

解：

```
     void TransLevel(BinTree *T)
     {
       int front=0,rear=0;
       int p;
       if(T!=Null)
        {
         printf(“%c”,T->data);
         q[rear]=T;
         rear++;
        }
       while(front<rear)
        {
          T=q[front];
          Front++;
          if(T->lchild!=Null)
           {
            printf(“%c”,T->lchild->data);
            q[rear]=T->lchild;
            rear++;
            }
            if(T->rchild!=Null)
             {
               printf(“%c”,T->rchild->dara);
               q[rear]=T->rchild;
               rear++;
              }
```

```
        }
      }
    main()
     {
      printf(“\n”);
      root=CreateBinTree();
      Inorder(root);
      Printf(“\n”);
      TransLevel(root);
     }
```

6. 30 以二叉树表为存储结构，写一算法用括号形式（keyLT,RT）打印二叉树，其中 key 是根结点数据，LT 和 RT 分别是括号形式的左右子树。并且要求：空树不打印任何信息，一给结点 x 的树打印形式是 x，而不应是（x,,）d 的形式。

解：

```
viod Print(BinTree T)              /*哟感括号形式打印二叉树*/
 {
  if(T!=Null)
   {
    if(T->lchild==Null&&T->rchild==Null)     /*只有根结点*/
     printf(“%c”,T->data);
    else  /*T->lchild!=Null||T->rchild!=Null*/
    {
     printf(“(”);
     printf(“%c”,T->data);
     if(T->lchild->lchild==Null&&T->lchild->rchild==Null)
      printf(“)”);
     Print(T->lchild);
     if(T->rchild!=Nulll)printf(“,”);
     printf(“)”);
     printf(“)”);
    }
   }
  }
main()
 {
  printf(“\n”);
  root=CreateBinTree();
  Inorder(root);
  printf(“\n”);
  Print(root);
 }
```

6.31 以线索链表为存储结构，分别写出在前序线索树中查找给定结点*p 的后继，以及在后序线索树中查找*p 的后序前趋的算法。

解：

① 找结点 p 的前序后继

```
BinTheNode * PreorderSuccessor(BinThrNode *p)
 {
   BinThrNode *q;
   if(p->rtag==Thread)
     q=p->rchild;              /*右子树为空*/
        else
     { if(p->ltag==list)
        q=p->lchild;           /*左子树非空*/
       if(p->ltag==Thread)
    q=p->rchild;             /*左子树为空*/

     }
    return(q);
 }
```

②找结点 p 的后序继

```
BinTheNode * PostorderSuccssor(BinThrNode *p)
 {
   BinThrNode *q;
   if(p->ltag==Thread)
    q=p->lchild;         /*左子树为空*/
    else
    {if(p->rtag==list)
       q=p->rchild;      /*右子树非空*/
     if(p->ltag==Thread)
           q=p->lchild;      /*右子树为空*/
    }
  return(q);
 }
/*说明：list\Thread 为特殊标志，其值分别为 0 与 1.*/
```

6.32 完成 6.6.1 节算法 CreateHuffmanTree 中调用的三个函数：InputWeight,SelecMin 和 InitHuffmanTree.

解：

① 初始化

```
InitHuffmanTree(HuffmanTree T)
{ int i;
  for(i=0;i<m;i++)
   {T[i]->parent=-1;
    T[i]->lchild=-1;
    T[i]->rchild=-1;
    T[i]->weigh=0;
    }
 }
```

② 读入叶子结点权值

```
  InputWeigh(HuffmanTree T)
{ int I;
   for(i=0;i<n;i++)
     scanf(“%d”,&T[i]->weigh);
 }
```

③ 选出两个权至最小的根结点

```
   SelecMin(HuffmanTree T,i-1,&p1,&p2)
   int i,p1,p2;
 { int j,small1,small2;
   small1=small2=max;                    /*设 max 为整型最大值*/
    for(j=0;j<i-1;j++)
         if(T[j]->parent==-1)
           if(T[j]->weigh<small1)
    {small2=small1;                     /*改变最小权、次小权及对应位置*/
     small1=t[j]->weigh;
     p2=p1;
     p1=j;
    }
   else
    if(T[j]->weigh<small2)
       {small2=T[j]->weigh;              /*改变最小权及对应位置*/
        p2=j;
        }
}
```

6.33 分别写出对文件进行哈夫谩码的算法，以及对编码文件进行解码的算法。为简单起见，可以假设文件存放在一个字符向量。

解：

① 编码算法

设哈夫曼树已求出。

```
  HuffmanCode(code,tree)
  Codetype code[];
  Huffmantype tree[]                    /*以求出*/
   { int I,j,c,p;
     codetype cd;                       /*缓冲区变量*/
  for(i=0;i<n;i++)
    { cd->start=n;
      c=i+1;
      p=tyee[i]->parent;
      while(p!=0)
       { cd->start=n;
         c=i+1;
         p=tyee[i]->parent;
         while(p!=0)
           { cd->start--;
```

```
            if(tree[p-1]->lchild==c)
              cd->bit[cd-start]=’0’;/*type[i]是左子树，生成代码为‘0’*/
            else
              cd->bit[cd->start]=’1’；/*type[i]是右子树，生成代码为‘1’*/

              c=p;
              p=tree[p-1]->parent;
            }
           code[i]=cd;
          }
        }
```

注：结构体

```
typedef struct
  { char bit[n];          /*位串*/
    int start;            /*编码在位串的起始位置*/
    char ch;              /*字库*/
  }codetype;
  codetype   code[n];
```

② 译码算法

```
Decode(code,tree)
Codetype code[];
Huffmantype tree[];
{ int I,j,c,p,b;
  int endfily=-1;          /*电文结束标志*/
  i=m;                     /*从根结点开始往下搜索*/
  scanf(“d”, &b);          /*读入一个二进制代码*/
  while(b!=endfily)
     { if(b==0)
        i=tyee[i]->lchild-1;      /*走向左孩子*/
         else
        i=tyee[i]->rchild-1;       /*走向右孩子*/
       if(tyee[i].child==0)        /*tyee[i]为叶结点*/
       { outchar(code[i]->ch);     /*译码，即输出叶结点对应的字符*/
         i=m;                      /*回归根结点*/
       P=tyee[p-1]->parent;
      }
       scanf(“%d”,&b);               /*读入下一个二进制代码*/
     }
     if(tyee[i].lchild!=0)            /*电文读完但未到叶结点*/
        printf(“\n ERRoR\n”);         /*输出电文有错信息*/
  }
```

同步综合练习及参考答案

（一） 基础知识题

7．1 在图 7.23 所是的各无向图中：

（1） 找出所有的简单环。

（2） 那些图是连通图？对非连通图给出其连通分量。

（3） 那些图是自由树（或森林）？ //自由树的概念见 7.4 节

解：

（1） 简单环

（a）(1 2 3 1)

(b) 无

(c) （1 2 3 1） （2 3 4 2） （1 2 4 3 1）

(d) 无

（2） 连通图（a）（c）（d）

非连通图（b）的连通分量为

（3） 自由树 （d）

森林 （b）

7．2 在图 7.24 所是的有向图中：

（1） 该图是强连通的码？若不是，则给出其强连通分量。

（2） 请给出所有简单路径及有向环。

（3） 请给出每个顶点的度、入度和出度。

（4） 请给出其邻接表、邻接矩阵及逆邻接表。

解：

（1） 图是强连通的。

（2） 所有简单路径（重复环未计算）

（v1）（v2）（v3）（v4）

(v1 v2)(v2 v3)(v3 v1)(v1 v4)(v4 v3)

(v1 v2 v3)(v2 v3 v1)(v3 v1 v4)(v3 v1 v2)(v1 v4 v3)(v4 v3 v1)

（v1 v2 v3 v1）（v2 v3 v1 v4）（v3 v1 v4 v3）（v4 v3 v1 v2）

有向环

（v1 v2 v4 v1）（v4 v1 v4 v4）

(3) 各顶点的度、入度、和出度。

(4) ①邻接表

① 邻接矩阵集。

② 逆邻接表

7．3 假设图的顶点是 A,B,…,请根据下属的邻接矩阵画出相应的无向图或有向图。

解：

7．4 假设一颗完全二叉树包含 A,B,…,G 第七个结点，写出其邻接表和邻接矩阵。

解：

① 完全二叉树

② 邻接矩阵

③ 邻接表

7．5 对 n 各顶点的无向图和有向图，采用邻接矩阵和邻接表表示，如何判别下列有关问题：

（1） 图中有多少条边？

（2） 任意两个顶点 I 和 j 是否有边相连？

（3） 任意一个顶点的度是多少？

解：

① 对于无向图

（1） 图中边数等于邻接矩阵中 1 的各数的一半；邻接表中的边表中结点各数的一半。

（2） 若邻接矩阵中 A[i,j]≠0 则 I 和 j 两个顶点有边相连。

（3） 顶点 I 的度为第 I 行中 1 的个数；邻接表中 I 的边表结点个数 j.

② 对于有向图

（1） 图中边树等于邻接矩阵中的个数；邻接表中的出边表中结点数。

（2） 若邻接矩阵中 A[i,j]>0 则 I 和 j 两个顶点有边相连 ；
邻结表中 I 得出边表是否有结点 j，决定 I 和 j 两个顶点有边相连。

（3）顶点 I 的度为第 I 行中 1 的个数加上第 I 列中 1 的个数之和；
邻接表中 i 得出边表结点个数加上边表中结点 I 的个数之和。

7．6 n 个顶点的连通图至少有几条边？强连通图呢？

解：

①n 个顶点的连通图至少有 n-1 条边。

②n 个顶点的强连通图至少有 n 条边。

7．7 DFS 和 BFS 遍历采用什么样的数据结构来暂存顶点？当要求连通图的生成树的高度最小，应采用何种遍历？

解：

①DFS 采用了栈；BFS 采用了队列。

②采用 BFS。

7．8 画出以顶点 v1 为初始源点遍历图 7。25 所示的有向图所得到的 DFS 和 BFS 生成森林。

解：

7．9 按顺序输入顶点对：（1，2），（1，6），（2，6），（1，4），（6，4），（1，3），（3，4），（6，5），（4，5），（1，5），（3，5），根据第 7.2.2 节中算法 CreateALGraph 画出相应的邻接表，并写出在该邻接表上，从顶点 4 开始搜索所得的 DFS 和 BFS 序列，及 DFS 和 BFS 生成树。

解：

依题意构造有向图：

①邻接表

②DFS 和 BFS 序列

DFS 序列：4 5 3 1 6 2

BFS 序列：4 5 3 6 1 2

③DFS 和 BFS 生成树

DFS 生成树

BFS 生成树

7.10 什么样的图其最小生成树是唯一的？用 Prim 和 Kruskal 求最小生成数的时间各为多少？他们分别适合于哪个图？

解：①具有 n 的结点，n-1 条边的连通图其生成树是唯一的。

① 的结点，e 条边的无向连通图求最小生成树的时间分别为

Prim: O(n2)； Kruskal:O(e*log2e)

② Prim 适应于稠密图（点少边多）；Kruskal 适应于稀疏图（点多边少）。

7．11 对图 7.26 所示的连通图，请分别用 Pirm 和 Kruskal 算法构造其最小生成树。

解：

① Pirm 算法构造的生成树。

② Kruskal 算法构造的生成树。

7．12 对图 7.27 所示的有向网，试利用 Dijkstra 算法求出从源点 1 到其它各顶点的最短路径，并写出执行短发过程中扩充红点集的每天次循环状态（类似表 7.2）.

解：

7．13 是写出图 7.28 所示有向图的所有拓扑序列，并指出应用教材中 NonPrefirtTopSort 算法求得的是哪个序列，设邻接表的边表结点中的邻接点序号是递增的。

解：

NonPreFirtTopSort 算法求得的序列是：

V0 V1 V5 V2 V6 V3 V4

其余序列有多种（略）。

7．14 什么样的 DAG 的拓扑序列是唯一的？

解：

设 DAG 中有 n 个结点，则当 DAG 中有至少 n-1 个不同的后继结点且至少有 n-1 个前缀结点时其拓扑序列是唯一的。

7．15 以 V4 为源点，给出用 DFS 搜索 7.28 的道德你拓扑序列。

V4 V3 V2 V0 V1 V5 V6.

（二） 算法设计题

7．16 试在无向图的邻接矩阵和邻接表上实现如下算法：

（1） 往图中插入一个顶点

（2） 往图中插入一条边

（3） 删去图中某顶点

（4） 删去图中某条边

解：

（1） 插入给定结点

① 邻接表

```
viod Inser-ALGraph(ALGraph *G,int k)
 {
   int i;
   G->n++;
   for(i=G->n;i>k;i--)
    {
      G->adjlist[i].vertex=G->adjlist[i-1].vertex;
      G->adjlist[i].firstedge=G->adjlist[i-1].Firstedge;
     }
   G->adjlist[i].vertex=getchar();
   G->adjlist[i].firstedge=Null;
 }
```

② 邻接矩阵

```
void Inser-Mgraph (mGraph *G,int k)
 {int i,j;
```

```
  G->n++;
 for(i=G->n;i>k;i--)G->vexs[i]=G->vexs[i-1];
    G->cexs[k]=getchar();
 for(i=G->n;i>k;i--)
  {  for(j=G->n;j>k;j--)
        G->e[i][j]=G->e[i-1][j-1];
     G->e[i][j]=0;
     for(j=k-1;j>=0;j--)
        G->e[i][j]=G->e[i-1][j];
  }
 for(j=G->n;j>=0;j--)G->e[i][j]=0;
 for(i=k-1;i>=0;j--)
  { for(j=g->n;j>k;j--)
      G->e[i][j]=G->e[i][j-1];
    G->e[i][j]=0
  }
}
```

⑵插入一条边

① 邻接表

```
InsertLAGraph (ALGraph *G,int m,INT n)
 { EdgeNode *E;
   EdgeNode *sm=new EdgeNode;
   EdgeNode *sn=new Edgenode;
   Sm->adjvex=n;sm->next=Null;
   Sn->adjvex=m;sn->next=Null;
   for(E=G->adjlist[m].firstedge;E!=NULL;E=E->next) { }
     E=sm;
   for(E=G->adjlist[n].firstedge;E!=NULL;E=E->next) { }
     E=sn;
 }
```

② 邻接矩阵

```
void InsertLMGraph(mGraph *G,int m,int n)
  { G->e[m][n]=1;
    }
```

（3）删除某结点

① 邻接表

```
  void DelVALGraph(ALGraph *g,int k)
  { int I;
    G->n--;
    EdgeNode *e *s;
    for(E=G->adjlist[k].firstdge;E!=NULL;E=E->next)
      { for (s=G->adjlist[E->adjvxe].firstedge;E!=NULL;E->E->next)
        if(s->adjvex==k)
        {s=s->next;break;}
```

```
    }
  for(i=k;i<G->n;i++)
    { G->adjlist[i].vertex=G->adjlist[i+1].vextex;
      G->adjlist[i].first[i].firstedge=G->adjlist[i+1].firstedge;
    }
  }
```

② 邻接矩阵

```
void delvMGraph(mGraph *G,int k)
{int i,j;
 G->n--;
 for(i=k;i<=g->n;i++) G->vexs[i]=G->vexs[i+1];
 for(i=0;i<k;i++)
     for(j=k;j<=n;j++) G->e[i][j]=G->e[i][j];
  for(i=k;i<=G->n;i++)
   {for(j=0;j<k;j++) G->e[i][j]=G->e[i+1][j];
    for(j=k;j<=n;j++) G->e[i][j]=G->e[i+1][j];
   }
 }
```

(4)删除某条边

① 邻接表

```
void dellALGraph(ALGraph *G,int m,int n)
{EdgeNode *s;
 for(S=G->adjlist[m].firstedge;
 s->adjvex!=k;s=s->next) s=s->next;
 for(s=G->adjlist[n].firstedge;
  s->adjvex!=k;s=s->next) s=s->next;
}
```

②邻接矩阵

```
void InsertLMGraph(mGraph *G,int m,int n)
 {G->e[m][n]=0;
  G->e[n][m]=0;
 }
```

7.17 下面的伪代码是一个广度优先搜索算法,试以图 7.29 中的 V4 为源点执行该算法,请回答下述问题:

(1) 对图中顶点 Vn+1,它需要入队多少次?它被重复访问多少次?

(2) 若要避免重复访问同一个顶点的错误,应如何修改此算法?

```
Void BFS(ALGraph *G,int k)
{//以下省略局部变量的说明,visited 各分量初值为假
 InitQueue(&Q);                          //置空队列
 EnQueue(&Q,k);                          //k 入队
 While(!QueueEmpty(&Q))
  {I=DeQueue(&Q);                        //vi出队
  visited[I]=true                        //置访问标记//
  printf(“%c”,G->adjilist[i],vertes); //访问 j
```

```
    for(p=G->adjilist[i],firstedge;p;p=p->adjves=j)//依次搜索 vi的邻接点 vj
    (不妨设 p->adjves=j)
    if(!Visited[p->adjves])                    //若 vj未被访问
       EnQueue(&Q,p->adjves);                  //vj 入队
    }                                          //endwhile
    }                                          //BFS
```

解：

对图中顶点 v_{n+1},它需入队 n 次？它被重复访问 n-1 次。

若要避免重复访问同一个顶点的错误，应修改算法如下：

```
  Void  BFS(ALGraph*G,int K)
{ /*以下省略局部变量得说明，visited 各分量初值为假*/
 InitQueue(&Q);                          /*置空队列*/
 EnQueue(&Q,k);                          /*k 队列*/
 While(!QueueEmpty(&Q))
     {I=DeQueue(&Q);                     /*vi 出队*/
      visited[I]=true                    /*置访问标记*/
 printf(“%c”,G->adjilist[i],vertes);/*访问 vi*/
 for(p=G->adjlist[i],firstedge;p;p->adjves=j)
                              /*依次搜索 vi 的邻接点 vj（不妨设 p->adjves=j）*/
 if(!Vvisit[p->adjves])           /*若 vj 未访问过*/
     {
      EnQueue(&Q,p->adjves);         /*/vj 入队*/
      Visited[p->adjvex]=TRUE;
        }
    }                                   /*endwhile*/
                }                       /*BFS*/
```

7.18 试以邻接表和邻接矩阵为存储结构，分别写出基于 DFS 和 BFS 遍历的算法来判别顶点 v_i 和 vj(I<>j)之间是否有路径。

解：

```
/*基于邻接表方式*/
/*所有数据类型*/
#define   maxvn 10
typedef struct node
  { int vertex;
    int vertex;
    setuct node *list;
    } vertexNode
  vertexNode *head[maxvn];
  bool JUDGE(vertexNode *adjl[maxvn],int n,int j);
     /*深度优先搜索判别 n 个顶点的有向图中顶点 I 到顶点 j 是否存在路径*/
   { int stack[maxvn];
     bool visit[maxvm];
     int top,k;
     vertexNODE *p;
```

```
 bool yes;
for(k=1;k<=n;k++)   visit[k]=false;
top=1;
stack[top]=i;
visit[i]=True;
yes=false;
do{
   p=adjl[stack[top]];
   while(!p=NULL&&visit[p->vertex]p=p->link);
   if(p==NULL)
      top=top-1;                    /*p 之后无邻接结点，退栈*/
    else
      {i=p->vertex;             /*p 指向的顶点未访问*/
       if(i==j)
        yes=true;
       else
         { visit[i]=true;
           top=top+1;
           stack[top]=i;
          }
   }while(top1=0&&!yes);
 return(yes);
}
```

7. 19 试分别写出求 DFS 和 BFS 生成树（或生成森林）的算法，要求打印出所有的树边。

解：

①/*以 Vi 为树根，对邻接矩阵表示的图 G 进行 DFS 搜索*/

```
void DFSM(Mgraph *G,int )
 { int j;
   printf(“visit vertex:%c”,G->vexs[i]);
   visitcd[i]=True;
   for(j=0;j<=G->n;j++)
       if(G->edges[i][j]==1&&!visit[j])
      { print(“edye:%d→%d\n”,i,j);
       DESM(G,j);
       }
 }
```

②/*以 VI 为树根，对邻接矩阵表示的图 G 进行 DFS 搜索*/

```
void DFSM(Mgraph *G,int k)
 { int i,j;
   SETNULL(Q);                /*置空队 Q*/
   Printf( ‘%/ “, G.vexs[k]);
   Visited[k]=True;           /*标志 Vk+1 已经访问过*/
   ENQUEUE(Q,k);              /*  已经访问过的顶点如队列*/
   While(!EMPTY(Q))           /*队列非空时执行*/
```

```
{i=DEQUEUE(Q);                    /*队头元素序号出队列*/
  for(j=0;j<n;j++)
    {print("edye:%d→%d\n",i,j);     /*访问过的边*/
     print("%c\n",G,vexs[j]);
     visited[j]=True;
     ENQUEUE(Q,j);                  /*访问过的顶点如队*/
     }
  }
}
```

7.20 写一算法切连通分量的个数并输出各连通分量的顶点集。

解：

/*修改 DFS 函数，每当使 visited[i]=true 时打印 printf（"%D"，i)；可输出各连通分量顶点集。*/

```
{int i;
 int m=0;
 for(i=0;i<n;i++)
     visited[i]=False;
 for(i=0;i<n;i++)
   if(!visited[i])
    {DFS(i);
     m++;
     printf("comp end\n");
     }
   printf("共有 m 个连通分量");
}
```

7.21 设图中各边上的权值均相等，是以邻接矩阵和邻接表为存储结构，分别写出算法：

（1） 求顶点 Vi 到 Vj(i<>j)顶点的最短路径；

（2） 求源点 Vi 到其余各顶点的最短路径。

要求输出路径上的所有顶点（提示：利用 BFS 遍历的思想）

解：

利用 BFS 的遍历思想，同时构造一棵树，使每个孩子结点均指向双亲，当搜索到目标结点时，则从目标回溯到根结点即可，具体算法如下 ：

```
typedef      struct        node
{ int data;
  struct node *parent;
}Tree;
void  Path(ALGraph,*G,int i,int j)   /*以邻接点存储*/
{ int k;              /*求 vi 到 vj 的最短路径*/
  CirQueue  Q;        /*将 DataType 改为 int */
  EdyeNode  P;
  InitQueue(&Q)       /*队列初始化*/
  Visit[i]=TRVE;        /*源点*/
  S=(tree)mallo((sizeof(Tree));        新建一个结点
  s->data=i                  /*树根*/
```

```
      s->patent=NULL;
  EnQueue(&Q,i);                    /*入队列*/
  While(!Queue  Empty(&Q))
  { k=DeQueue(&Q);
    p=G->dajlist[k]->firstdeye;
    while(P)
    { if(! Visited[p->adjvex])
     { visited[p->adjvex]=TRVE;
       *S=(Tree*)mallo((sizeof(Tree));       /*新结点*/
       s->data=p->adkvex;
       s->parent->data=k;
       EnQueue(&Q,p->adjvex);
      }                                     /*end if */
      if(p->adjvex==j)bread;                /*已搜索到 j 点*/
        p-=p->next;
      }                                     /*end if */
      if(p->adjvex==j)break;               /*已搜索到 j 点，结束搜索*/
     }                                      /*end if */
     while(s!=NULL)                       /*此时打印出来的路径为反序*/
     { printf(“%d”,G->adjlist[s->data]->vertex)
       s=s->parent;
      }
```

（2）求源点到其余各顶点最短路径，则可调用上面算法。

```
Void PathALL(ALGraph *G.int i)
{int j,
 for(j=0;f<G->n;j++)
 if(j!=i)
 path(G,i,j);
}
```

对于用邻接矩阵存储的图:

(1) 求顶点 v_i 到顶点 v_j(i≠j)最短路径

算法思想，采用弗洛伊德算法，可表示为

A[K][i,j]=Min(A k-1[i,j],A k-1[i,k]+A k-1[k.j])(1<=i<=n,1<=j<=n)

其中 k 表示第 k 次迭代运算。A[0][i,j]=A[i,j].

```
  #define MAXVEX 100
  void floyd(a,c,n)     /*c 为已知邻接矩阵，a 为待求矩阵，n 为源点数*/
  int a[MAXVEX][MAXVEX] c[MAXVEX][MAXVEX],n;
  {int i,j,k;
   for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
   a[i][j]=c[i][j];
   for(i=1;i<=n;i++) a[i][j]=0;
   for(k=1;k<=n;k++)
   for(i=1;i<=n;i++)
```

```
        for(j=1;j<=n;j++)
        if(a[i][k]+a[k][j]<a[i][j])
        a[i][j]=a[i][k]+a[k][j];
       }
```

7.22 以邻接表为存储结构，写一个基于 DFS 遍历策略的算法，求图中通过某顶点 v_k 的简单回路（若存在）。

解：

算法思想，从给定顶点 v4 出发进行深度优先搜索，在搜索过程中判断当前访问的顶点是否为 v4。若是，则找到一条回路。否则继续搜索。为此设一个顺序栈 cycle 记录构成回路的顶点序列，把访问顶点的操作改为将当前访问的顶点入栈；相应地，若从某一顶点出发搜索完再回溯，则做退栈操作，同时要求找到的回路的路径应大于 2。另外还设置一个 found，出值为 0，当找到回路后为 1。

```
Void  dfscycle (ALGrph *G,int   V4)
{int i,j,top=0,V=V4,found=0,w;
 int Visitde[100],cycle[100];
 EdgeNode  *P;
 i=1;
 cycle[i]=Vi                  /*从 V 是开始搜索*/
 Visitde[v]==1;
 P=G[v]->firstedge;
 While(p!=NULL!!top>0)&&!found)
 { while(p!=NULL&&!found)
   if(p->adjvex==V4&&i<2)found=1;    /*找到回路*/
   else if(visited[p->adjvex]==0)p=p->next; /*找到下一个邻接点*/
   elst
   {w=p->adjvex;            /*记下路径，继续搜索*/
    visited[w]=1;
    i++;
    cycle[i]=w;
    top++;
    stack[top]=p;
    p=G[w]->firstedge;
   }
   if(!found&&top>0)       /*沿原路径退回后，另选路径进行搜索*/
   { p=attack[top];
     top--;
     p=p->next;
     i--;
    }
   }                           /*end while*/
   if(found)
    {for(j=1;j<=i;j++)
     printf(“%d,”,cycle[j]);          /*打印回路的顶点序列*/
     printf(“%d,\n”,V);
```

```
    }
   else printf(“设有通过点 V4 的回路！\n”)
   }
```

7．23 写一算法求有向图的所有根（若存在），分析算法的时间复杂度。

解：

算法思想：以有向图的每一个结点为源点对圆进行搜索，若能搜索到每个结点。则该结点为根。否则不是。

```
Void searchroot （ALGraph  *G）
{ int i;
  for(i=0;i<G->n;i++)
  {for(j=0;j<G->n;j++)
   visited[j]=false;          /*标志向量初始化*/
   DPS(G,i);         /*以 Vi 为源点开始 DPS 搜索*/
   }
 }
void  DPS(ALGtaph  *G,int i)
{ int count=0;
  EdgeNode   *P;
  Visited[i]=true;
  count ++;
  p=G->adjlist[i]->firstedge;
  while(p)
  {if(!Visited[p->adjvex])
   DPS(G,p->adjvex);
   P=p->next;
  }
  if(count==G->n)                 /*该结点是根结点*/
  printf(“%c”,G->adjlist[i]->vertex);
  }
```

7．24 改写 7.5 节的算法 print,试输出的从源点到各终点的最短路径是正想。（提示：使用栈暂存路径）。

解：

使用栈暂存路径

```
 void Ptint(PathP,Distance D)
 { int i,pre;
   seqstack *S;
   s->top=-1                /*置空栈*/
   for(i=0;i<n;i++)
   {printf(“\n distanck:%d,Path;”,D[i]);
                                  /*输出终点 I 的最短距离*/
    s->top++;
    s->stack[s->top]=i;       /*i 入栈*/
    pre=p[i];                 /*栈终点的前趋*/
    while(s->top>1)
```

```
    {printf(“%d”,s->stack[s->top]);    /*输出路径*/
     s->top--;
     }                  /*end while */
   }                    /*end for*/
  }
```

7.24 改写 7.5 节的算法 print，使输出的从源点到各终点的最短路径是正向。（提示：使用栈暂存路径）。

解：

使用栈暂存路径

```
   void Print(PathP,Distance  D)
{ int ,pre;
  seqstack      *s;
   s->top=-1                        /*置空栈*/
  for(i=0;i<n;i++)
  {printf(“\n distanck:%d,path:”,D[i]);
                                                  /*输出终点 i 的最短距离*/
   s->top++;
   s->stack[s->top]=i;              /*I 入栈*/
   pre=p[i];                        /*栈终点的前趋*/
   while(pre!=-1)
   {s->top++;
   s->stack[s->top]=pre;            /*路径入栈保存*/
   pre=p[pre];                       /*继续上溯前趋*/
   }
   while(s->top>-1)
   {printf(“%d”,s->stack[s->top]);  /*输出路径*/
   s->top--;
      }                             /*end while*/
    }                                /*end for*/
   }
```

7.25 对 7.6 节的 NonSuccFirstTopSort 算法，分别以邻接矩阵和邻接表作为存储结构，写出其具体算法，并分析算法的时间。

解：

① 用逆邻接表作为 G 的存储结构。

```
Void NonSuccPirstTopSort(G)
{int  outdehree[maxvertexNum];     /*出度向量，Maxvertexnum>=G,n*/
 seqstack  S,T;                    /*应将栈中 data 向量改为 int 类型*/
 int i,j,count=0;
 Edgenode   *P;
 for(i=0;i<n;i++)                 /*以下为初始化*/
outdegree[i]=0;
for(p=G->adjlist[i]->firstedge;p;p->next)  /*扫描的入边表*/
outdegree[p->adjvex]++;                    /*出度为 1*/
Initstack(&S);                      /*置空栈*/
```

```
    Initstack(&T);
 for(i=0;i<G->n;i++)
 if(outdegree[i]==0)
 push（&S,i）                          /*出度为零的顶点 i 入栈 */
 while(!stackEmpty(&S))                 /*栈非空，即图中有出度为 0 的顶点*/
 {i=pop(&S);
  push(&T,i);
  count++;
  for(p=G->adjlist[i]->firstedge;P;p=p->next);  /*扫描的 i 入边表*/
  {j=p->adjvex;                         /*j 是 i 的入边的<i,j>起点*/
   outdegree[j]--;        /*j 的出度减肥。相当于删去边<i,j>*/
   if(!outdegree[i])          /*j 无后继*/
  push(&S,j);
  }/*end for*/
}  /*end  while*/
if(count<G->n)
printf(“\n The Graph is  not a DAG.\n”); /*图中有环，排序失败*/
else
{while(!stackEmpty(&S))                 /*输出拓扑序列*/
 {i=po(&T);print(“%d”,G->adjlist[i]->vertex);}
 }
 }
 ②用邻接矩阵作为存储结构。
    Void   NoSucePirstTopSort(Mgraph   *G)
    {seqstack s,T;
    int i,j,count=0;              /*用 i,j 代表 Vi,Vj*/
    Initstack(&S);
    Initstack(&T);
    for(i=0;i<G->n;i++)
    for(j=0;j<G->n;j++)
    if(G->edges[i][j]= =0)
    push(&S,i);                           /*出度为 0，入栈*/
    while(!sruckEmpty(&S))
    { i=po(&S);
     push(&T,i);
     count+ +;
     for(j=0;j<G->n;j++)
     G->edges[i][j]=0;                    /*修改邻接矩阵*/
    for(i=0;i<G->n;i++)
    for(j=0;j<G->n;j++)
    if(G->edges[i][j]= =0)
    push(&S,i);
   }                                  /*end while */
   if(count<G->n)
```

```
 printf("\n The Graph is not a DAG.\n");
else
{while(!stackEmpty(&T))                    /*输出拓扑序列*/
 {i=pop(&T);
  printf("%d",G->adjlist[i]->vertex);
}
 }
}
```

7.26 设有向图一个 DAG，试以邻接矩阵和邻接表作为存储结构，写出对 7.6 节的 DFSTopSort 求精的算法。问什么有向图不是 DAG 时，该算法不能正常工作？

解：

① 用邻接矩阵作为存储结构。

```
Void DPSTopSort(Mgraph*G,i,Setstack T)
{int j;
 visited[i]=true;
 for(j=0;j<G->m;j++)                  /*栈所 i 有的邻接点 j*/
if(G->edges[i][j]= =1)
if(!visited[j])
DPSTopSort(G,j,T);
Push(&T,i);                           /*从 I 出发的搜索已完成，保存 i*/
}
```

② 用邻接表作为存储结构。

```
Void DPSTopSort(ALGraph   G,i,T)
{int j;
 visited[i]=true;
for(p=G->adjlist[i]->firstedge;p;p=p->next)
if(!visited[p->adjvex])
DPSTopSort(G,p->adjvex,T);
Push(&T,i);
}
```

由于有向图不是 DAG 时，从某点出发的搜索将陷入循环状态，所以算法不能正常工作。

7．27 利用拓扑排序算法的思想写一算法判别有向图中是否存在有向环，当有向环存在时，输出构成环的顶点。

解：

设有向图用邻接表存储

```
Void SearckCycle(ALGraph G)
{int   i;
EdgeNode *P;
 for(i=0;i<G->n;i++)                          /*对每一个顶点 i*/
for(p=G->adjlist[i]->firstedge;p;p=p->next)   /*扫描 i 的出边表*/
if(p->next=G->adjlist[i]->firstedge)          /*存在环*/
printf("%d",G->adjlist[i]->vertex);
}
```

第八章　　习　题

一 、判断题

1．求最小生成树的 Prim 算法在边较少、结点较多时效率较高。（ 错 ）

2．图的最小生成树的形状可能不唯一。（ 对 ）

3．用邻接矩阵法存储一个图时，在不考虑压缩存储的情况下，所占用的存储空间大小只与图中结点个数有关，而与图的边数无关。(对)

4．邻接表法只用于有向图的存储，邻接矩阵对于有向图和无向图的存储都适用。(错)

5．任何有向网络（AOV-网络）拓扑排序的结果是唯一的。(错)

6．有回路的图不能进行拓扑排序。(对)

7．存储无向图的邻接矩阵是对称的，故只存储邻接矩阵的下(或上)三角部分即可。(对)

8．用邻接矩阵 A 表示图，判定任意两个结点 Vi 和 Vj 之间是否有长度为 m 的路径相连，则只要检查 A^m 的第 i 行第 j 列的元素是否为 0 即可。(对)

9．在 AOE 网中一定只有一条关键路径。(错)

10．缩短关键路径上活动的工期一定能够缩短整个工程的工期。(错)

11．连通分量是无向图中的极小连通子图。(错)

12．强连通分量是有向图中的极大强连通子图。(对)

二、选择题

1．N 条边的无向图的邻接表的存储中，边表的个数有（A ）。

A）N　　B）2N　　C）N/2　　D）N*N

2．N 条边的无向图的邻接多重表的存储中，边表的个数有（A ）。

A）N　　B）2N　　C）N/2　　D）N*N

3．最短路径的生成算法可用（C ）。

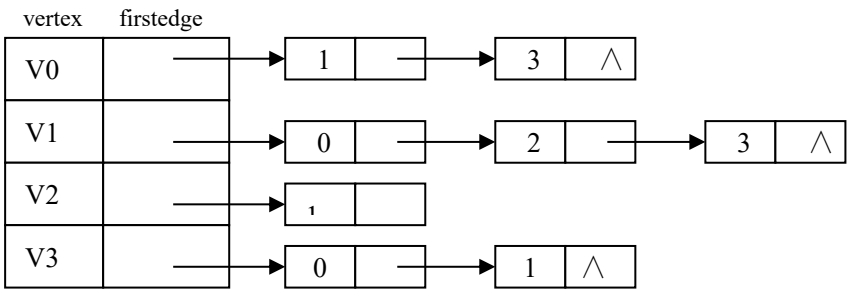
A）普里姆算法　B）克鲁斯卡尔算法　C）迪杰斯特拉算法　D）哈夫曼算法

(常见的求最小生成树的方法有两种：克鲁斯卡尔(Kruskal)算法和普里姆（Prim）算法)

4．有拓扑排序的图一定是（D ）。

A）有环图　　B）无向图　　C）强连通图　　D）有向无环图

5．图的邻接表如下图所示：



（选择题 5 图）

（1）从顶点 V0 出发进行深度优先搜索，经历的结点顺序为（B ）。

A）V0,V3,V2,V1　　B）V0,V1,V2,V3　　C）V0,V2,V1,V3　　D）V0,V1,V3,V2

（2）从顶点 V0 出发进行广度优先搜索，经历的结点顺序为（B ）。

A）V0,V3,V2,V1　　B）V0,V1,V2,V3　　C）V0,V2,V1,V3　　D）V0,V1,V3,V2

6．设有向图 n 个顶点和 e 条边，进行拓扑排序时，总的计算时间为（D ）。

A）O (nlog$_2$e)　　B）O (en)　　C）O (elog$_2$n)　　D）O (n+e)

7．对于含有 n 个顶点 e 条边的无向连通图，利用 Kruskal 算法生成最小代价生成树其时间复杂度为（A）。

A）O (elog$_2$e)　　B）O (en)　　C）O (elog$_2$n)　　D）O (nlog$_2$n)

8．关键路径是事件结点网络中（A ）。

A）从源点到汇点的最长路径　　B）从源点到汇点的最短路径

C）最长的回路　　D）最短的回路

9．N 个顶点的强连通图至少有（(1)）条边，这样的有向图的形状是（(2)）。

（1）A）n　　B）n＋1　　C）n－1　　D）n（n－1）

（2）A）无回路　　B）有回路　　C）环状　　D）树状

(不知道对不对)10．设 G 有 n 个顶点和 e 条边，进行深度优先搜索的时间复杂度至多为（(1)）。当 G 是非孤立顶点的连通图时有 2e≥n,故可推得深度优先搜索的时间复杂度为（(2)）。

（1）A）O (ne)　　B）O (n＋e)　　C）O (elog$_2$n)　　D）O (e)

（2）A）O (ne)　　B）O (n＋e)　　C）O (elog$_2$n)　　D）O (e)

三、填空题

1．设无向图 G 中顶点数为 n，则图 G 最少有____n-1__(不知道对不对)_条边，最多有___ n(n-1)/2____条边。若 G 为有向图，有 n 个顶点，则图 G 最少有___n__(不知道对不对)_条边；最多有____ n(n-1)___条边。

2．具有 n 个顶点的无向完全图，边的总数为___ n(n-1)/2____条；而在 n 个顶点的有向完全图中，边的总数为____ n(n-1)___条。

3.图的邻接矩阵表示法是表示_____顶点_____之间相邻关系的矩阵。

4.一个图的生成树的顶点是图的____全部_____顶点。(这个真的不知道填啥)

5．写出下图所示的所有拓扑序列_____(共计 24 种)_____。

0,1,5,2,3,6,4　0,1,5,2,6,3,4　0,1,5,6,2,3,4

0,5,1,6,2,3,4　0,5,1,2,6,3,4　0,5,1,2,3,6,4　0,5,6,1,2,3,4

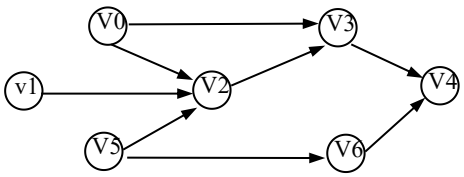
1,0,5,2,3,6,4　1,0,5,2,6,3,4　1,0,5,6,2,3,4

1,5,6,0,2,3,4　1,5,0,6,2,3,4　1,5,0,2,6,3,4　1,5,0,2,3,6,4

5,0,1,6,2,3,4　5,0,1,2,6,3,4　5,0,1,2,3,6,4　5,0,6,1,2,3,4

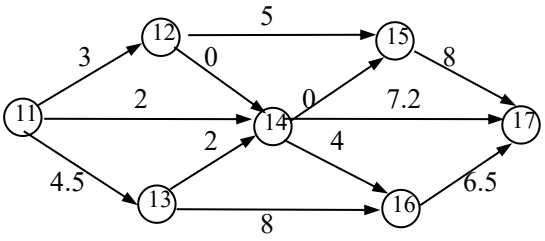
5,1,6,0,2,3,4　5,1,0,6,2,3,4　5,1,0,2,6,3,4　5,1,0,2,3,6,4

5,6,1,0,2,3,4　5,6,0,1,2,3,4



(填空题 5 图)

6．在如下图所示的网络计划图中关键路径是_____11.13.16.17_____，全部计划完成的时间是_____19_____。



（填空题 9 图）

7．设 G 有 n 个顶点和 e 条边，进行广度优先搜索的时间复杂度至多为___n+e____。当 G 是非孤立顶点的连通图时有 2e≥n,故可推得广度优先搜索的时间复杂度为____n___。这个前面好像有

8．一个连通图的生成树是一个___强____连通子图，n 个顶点的生成树有___(n-1)____条边。

9．对于含有 n 个顶点 e 条边的无向连通图，利用 prim 算法生成最小代价生成树其时间复杂度为___ne_____。求解

10．邻接表和十字链表适合于存储___稀疏____图，邻接多重表适合于存储___稠密____图。

四、简答题

1．对于如图所示的有向图，试给出

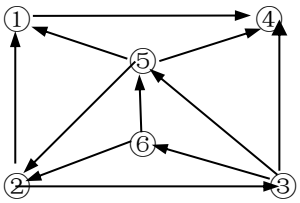
（1）每个顶点的入度和出度；

（2）邻接矩阵；

（3）邻接表；

（4）逆邻接表；

（5）强连通分量。



（简答题 1 图）

2．设无向图 G 如图所示

試给出

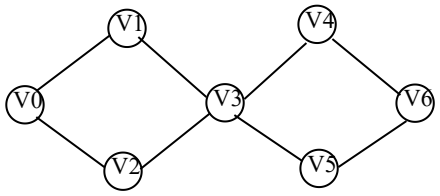
（1）该图的邻接矩阵；

（2）该图的邻接表；

（3）该图的多重邻接表；

（4）从 A 出发的“深度优先”遍历序列；

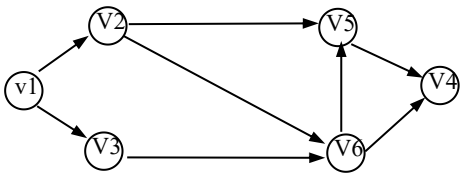
（5）从 A 出发的“广度优先”遍历序列。



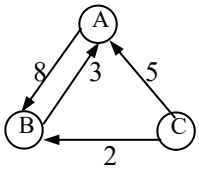
（简答题 2 图）

3．设有向图 G 如图所示

试画出图 G 的十字链表结构，并写出图 G 的两个拓扑序列。



（简答题 3 图）



（简答题 4 图）

4．试利用弗洛伊德（R.W.Floyed）算法，求图所示有向图的各对顶点之间的最短路径，并写出在执行算法过程中，所得的最短路径长度矩阵序列和最短路径矩阵序列。

5．下表列出了某工序之间的优先关系和各工序所需时间，求：

工序代号	所需时间	前序工序	工序代号	所需时间	前序工序
A	15	无	H	15	G,I
B	10	无	I	120	E
C	50	A	J	60	I
D	8	B	K	15	F,I
E	15	C,D	L	30	H,J,K
F	40	B	M	20	L
G	300	E			

（简答题 5 表）

(1)画出 AOE 网；

(2)列出各事件中的最早、最迟发生时间；

(3)找出该 AOE 网中的关键路径，并回答完成该工程需要的最短时间。

五、算法设计题

1．试以邻接矩阵为存储结构实现图的基本操作：InsertVex (G,v)、InsertArc (G,v,w)、DeleteVex (G,v)和 DeleteArc (G,v,w)。

2． 试以邻接表为存储结构实现算法设计题 1 中所列图的基本操作。

3．试以十字链表为存储结构实现算法设计题 1 中所列图的基本操作。

4．试以邻接多重表为存储结构实现算法设计题 1 中所列图的基本操作。

5．试写一算法由图的邻接链表存储得到图的十字链表存储。

6．写一算法，由依次输入图的顶点数目、边的数目、各顶点的信息和各条边的信息建立无向图的邻接多重表。

7．试写一个算法，判别以邻接表方式存储的有向图中是否存在由顶点 vi 到顶点 vj 的路径（i≠j）。假设分别基于下述策略：

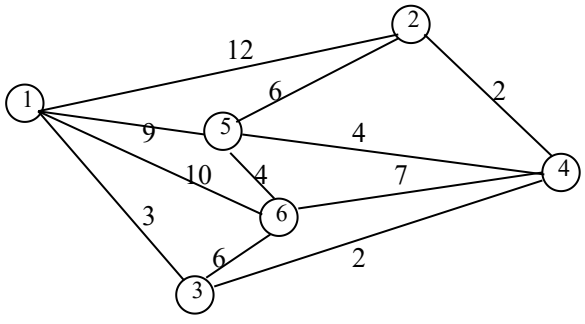
（1）图的深度优先搜索;

（2）图的宽度优先搜索。

8．试修改 Prim 算法，使之能在邻接表存储结构上实现求图的最小生成森林，并分析其时间复杂度（森林的存储结构为孩子－兄弟链表）。

9．以邻接表作存储结构实现求从源点到其余各顶点的最短路径的 Dijkstra 算法。

10．给定 n 个村庄之间的交通图，若村庄 i 和村庄 j 之间有道路，则将顶点 i 和顶点 j 用边连结，边上的权 Wij 表示这条道路的长度。现在要从这 n 个村庄中选择一个村庄建一所医院，问这所医院应建在那个村庄，才能使离医院最近的村庄到医院的路程最短？试设计一个解答上述问题的算法，并应用该算法解答如图所示的实例。



景点不少于 10 个。以图中顶点表示校内各景点，

六、上机实习题目

设计一个校园导游程序，为来访的客人提供各种信息查询服务。

基本要求：

（1）设计你所在学校的校园平面图，所含场所不少于 10 个。以图中顶点表示校内各场所，存放场所名称、代号、简介等信息；以边表示路径，存放路径长度等相关信息。

（2）为来访客人提供图中任意场所相关信息的查询。

（3）为来访客人提供图中任意场所的问路查询，即查询任意两个景点之间的一条最短的简单路径。

选作：

（1）提供图中任意场所得问路查询，即求任意两个场所之间的所有路径。

（2）校园导游图的场所与道路的修改与扩充功能。

2.1. 一元稀疏多项式的求导算法

写出一元稀疏多项式的求导算法，用带表头结点的单链表存储该一元稀疏多项式，Lb 为头指针，用类 C 语言描述该求导算法，不另行开辟存储空间，删除无用结点，并分析算法的时间复杂度。该链表的数据结构如下：

```
typedef struct LNode{
    float   coe;、//系数
    int     exp;   //指数
    struct  LNode   *next;   //指针
} LNode , *LinkList ;
```

求导算法如下：

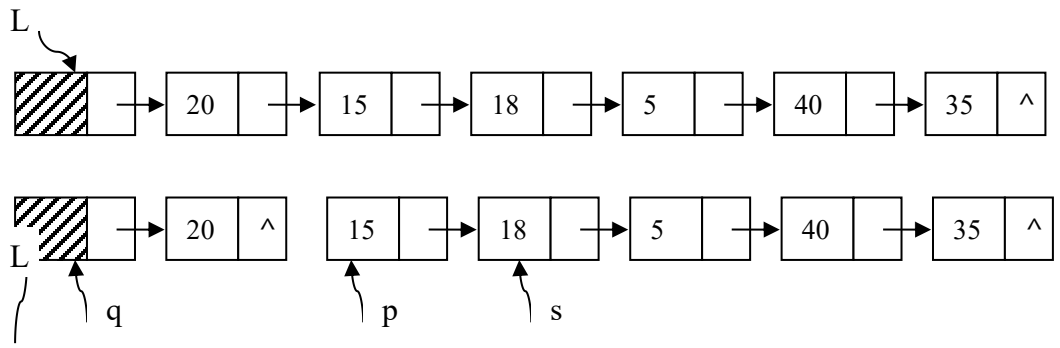
```
void   Differential(LinkList &Lb)
 { //求导算法，Lb 为链表头指针
 LinkList p, pre; //定义指针变量 p, pre
 pre=Lb; //初始化变量，pre 为 p 的前驱
 p=pre->next; // p 为当前待处理的结点
 while ( p ) //遍历链表
  {//逐个结点（数据项）求导处理
   if ( p->exp != 0 )//指数不等于零，非常数项
    {   p->coe = p->coe * p->exp ; //修改系数
        p->exp = p->exp – 1 ; //修改指数
        pre = pre->next ; //后移前驱指针 pre
    }
   else//指数等于零，常数项
    {   pre->next = p->next ; //逻辑删除常数项结点
        free ( p ); //物理删除常数项结点，即释放所占内存
    }
   p = pre->next ; //处理下一结点
  }
 }// Differential
```

时间复杂度为: O(n)

2.2. 单链表存储结构的排序算法

排序算法：将一组整数排序成非递减有序序列。用带头结点的单链表存储，L 为头指针，用类 C 语言写出该排序算法，不另行开辟存储空间，并分析算法的时间复杂度。该单链表的数据结构如下：

```
typedef struct LNode{
        int    data; //数据域
    struct    LNode    *next;    //指针域
        } LNode , *LinkList ;
void    Sort(LinkList    &L)//L 为链表头指针
{    //排序算法如下：将 L 排序成非递减单链表
    LinkList    q,p,s;    //定义变量
    if (L->next==NULL) { printf("空表\n");    return;}
//拆成 2 个链表，第 1 个链表只有头结点和首元结点，其余结点为第 2 个链表
    q=L;//q 为待处理结点的前驱
    p=q->next->next;// 第 2 个链表的头指针为 p
    q->next->next=NULL; // 第 1 个链表的头指针为 L
    //直接插入排序方法：第 2 个链表逐个结点插入到第 1 个链表
    while(p) //遍历链表
    { //比较数据域大小，定位合适的插入位置的前驱 q
        while(q->next && p->data >= q->next->data)    q=q->next;
        s=p->next; //保留下一个待处理结点
        p->next=q->next; //本条语句和下一条语句：p 结点插入到 q 结点的后面
        q->next=p;
        p=s; //处理下一结点
        q=L; //q 归位链表头结点 L
    }
}//sort
```



2.3. 设 H 为具有 n(n>0，n 很大且未知)个数据元素的单链表的头结点指针，试采用 C 语言编写一个程序，完成将单链表中第 n/2 个数据元素之后的全部数据元素倒置的功能。要求不另行开辟存储空间，算法的时间复杂度不超过 O(n)。

单链表结点的数据类型描述如下：

```
typedef struct Lnode {
  int data;    //数据元素为整数
  struct Lnode    *next;
}Lnode, *LinkList;

void ReverseN2(LinkList &H)
{//将单链表的正中间位置结点之后的全部结点倒置的功能
   LinkList p,q,s;
   p=H;   //初始化变量，p 指向头指针
   q=H;   //初始化变量，q 指向头指针
//定位链表正中间位置结点。p 结点走到表尾，q 结点在正中间位置
   while (p)
   {   if (p->next) //如果不是表尾结点
           p=p->next->next;   // p 指向下一结点的下一结点
       else//如果是表尾结点
           break;     //退出
       q=q->next;    //处理下一结点 q 指向下一结点
   }

   //拆成 2 个链表，第 1 个链表：头结点开始到正中间位置 q 结点
   //第 2 个链表：q 结点（不含 q）之后结点组成第 2 个链表
   p=q->next; // p 指向 q 结点的后继结点，p 为第 2 个链表的头指针
   q->next=NULL; //第 1 个链表的尾结点指针为 q
   //头插法（逆序），p 指向的第 2 个链表逐个逆序插入 q 结点的后面
   //实现倒置功能
   while (p)
   {   s= p->next; //保留下一个待处理结点
       p->next=q->next; //本条语句和下一条语句：p 结点插入到 q 结点的后面
       q->next=p;
       p=s; //处理下一结点
   }
}//ReverseN2
```

3.1．请写出如下递归程序的输出结果。

```
#include <stdio.h>
#include <stdlib.h>
int Fb(int x);
void main( )
{ int k=40, val;
   val= Fb(k);
  printf(" %d , %d 。" , val, k );
}

int Fb(int x)
{ int y;
  if (x <= 5 )   y = x / 5;     /* x/5 表示 x 整除 5*/
  else    y = Fb(x - 26) + x/3;
  printf("%d , ", x );
  printf("%d ; ", y );
  return ( y );
}
```

（1）主程序执行【int k=40, val; val= Fb(k);】后，调用递归函数 **Fb(40)**，子程序 **Fb(40)**入栈顺序为：

入栈顺序	堆栈		
	x	y	
3	**-12**	**Fb(-12)=-12/5=-2**	**←栈顶 Top**
2	**14**	**Fb(14)=Fb(14-26)+14/3= Fb(-12)+4**	
1	**40**	**Fb(40)=Fb(40-26)+40/3= Fb(14)+13**	**←栈底 Bottom**

即：

栈顶 Top→	**x=12，y=Fb(-12)=-12/5=-2**
	x=14，y=Fb(14)=Fb(14-26)+14/3= Fb(-12)+4
栈底 Bottom→	**x=40，y=Fb(40)=Fb(40-26)+40/3= Fb(14)+13**

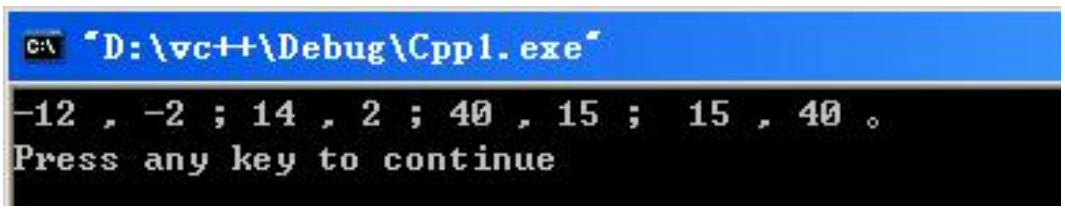
（2）子程序 **Fb( )**出栈顺序：

出栈次序	当前栈顶元素		输出数据	
	x	y	printf("%d , ", x);	printf("%d ; ", y);
1	**-12**	**Fb(-12)=-12/5=-2**	**-12，**	**-2；**
2	**14**	**Fb(14)= Fb(-12)+4=-2+4=2**	**14，**	**2；**
3	**40**	**Fb(40)= Fb(14)+13=2+13=15**	**40，**	**15；**

（3）主程序【**val= Fb(k);**】执行后【**printf(" %d , %d 。" , val, k);**】的输出结果：**15，40**。

因此，该程序执行结果为：**-12，-2；14，2；40，15；**15，40。

程序运行结果截图如下：



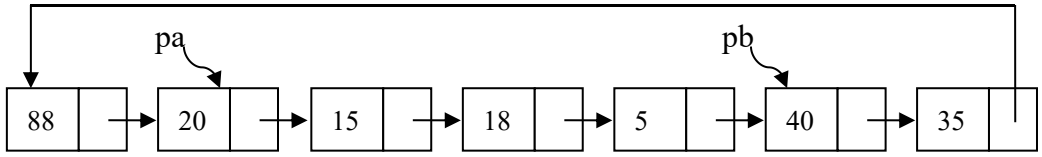
3.2. 简述下列算法的功能

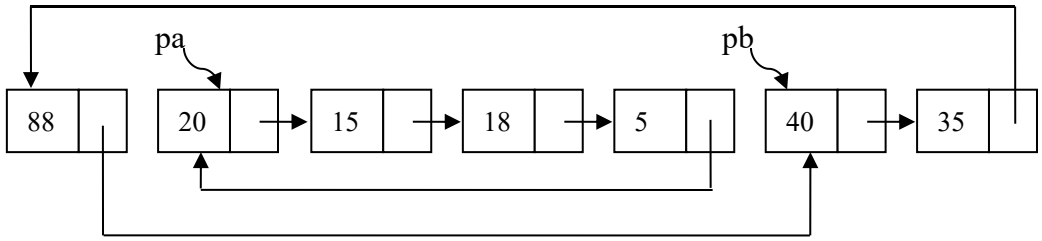
```
void Split(Lnode *s , Lnode *q )
{
   Lnode *p;
   p=s;
   while ( p->next !=  q ) p = p->next;
   p->next = s;
}

void AtoBB(Lnode *pa , Lnode *pb )
{
   //pa和pb分别指向单循环链表( 结点数 > 1 )中的两个结点.
   split(pa,pb);
   split(pb,pa);
}
```

解：

该算法的功能为：将原单循环链表从 pa 和 pb 所指向的结点断裂，即原来指向 pb 结点的链，转向指 pa，而原来指向 pa 结点的链，转向指 pb。这样就形成两个独立的单循环链表。

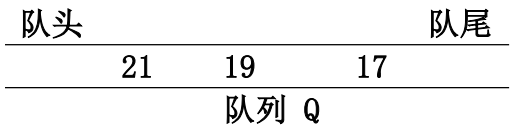




3.3. 试读下列栈和队列操作的算法，请给出调用并执行 testit(3)后的所有输出结果。

```
//Stack为栈，Queue为队列
//InitStack(S)为构造一个空栈，InitQueue(Q)为构造一个空队列。
//EnQueue( Q , nb )表示入队列操作；DeQueue(Q)表示出队列操作。
void testit( int m )
{
   Stack S;
   Queue Q;
   int  na , nb , nm ;
   InitStack(S);   InitQueue(Q);
   na=12;   nb=21;   nm=m;
   while( na < nb )
   {
      Push( S ,na ); EnQueue( Q , nb );
      na++; nb -=2;  nm++;
   }
   printf("nm=%d,na=%d,nb=%d\n",nm,na,nb);
   while ( nm > 0 )
   {
      nm -= 2;
      na=POP(S);
      nb=DeQueue(Q) + na;
      printf("Out:%d\n",nb);
   }
}
```

解：



入栈/入队：

nm	na	nb
3	12	21
4	13	19
5	14	17
6	15	15

输出：nm=6, na=15, nb=15

出栈/出队：

nm	na	nb	输出
6 – 2 = 4	14	21 + 14 = 35	35
4 – 2 = 2	13	19 + 13 = 32	32
2 – 2 = 0	12	17 + 12 = 29	29

输出：Out:35

Out:32

Out:29

所以，输出结果为：

nm=6, na=15, nb=15

Out:35

Out:32

Out:29

4.1.1 模式匹配

设主串 S='aaabdaaabfaaabdaaabbdaaaabdaaabbda'，子串 T='aaabdaaabbda'，求解下列问题：

（1）求出模式 T 的 next[j]值；

（2）求出模式 T 的 nextval[j]值；

（3）在 S 中查找 T 至少需要几趟匹配？至少需要几次比较？模式匹配成功的位置序号？请给出详细的匹配过程。

提示：KMP 算法

$$\text{next}[j]=\begin{cases}0 & \text{当 } j=1 \text{ 时}\\ \max\{k \mid 1<k<j \text{ 且 } 'T_1\ldots T_{k-1}'='T_{j-(k-1)}\ldots T_{j-1}'\} & \\ 1 & \text{其他情况}\end{cases}$$

$$\text{nextval}[j]=\begin{cases}\text{next}[j] & \text{当 } T_j!=T_{\text{next}[j]}\\ \text{nextval}[\text{next}[j]] & \text{当 } T_j==T_{\text{next}[j]}\end{cases}$$

答：

j	1	2	3	4	5	6	7	8	9	10	11	12
模式串 T	*a*	*a*	a	b	d	a	a	a	b	b	d	a
Next[j]	0	1	2	3	1	1	2	3	4	5	1	1
NextVal[j]	0	0	0	3	1	0	0	0	3	5	1	0

匹配过程如下：

a a a b d a a a b f a a a b d a a a b b d a a a a b d a a a b b d a

a a a b d a a a b b …i=1→10, j=1→10, 比较 10 次。j=NextVal[10]=5

(a a a b)d …i=10→10, j=5→5, 比较 1 次。j=NextVal[5]=1

a …i=10→10, j=1→1, 比较 1 次。j=NextVal[1]=0，则 i++, j++

a a a b d a a a b b d a…i=11→23, j=1→13, 比较 12 次，匹配成功。

因此，至少需要 4 趟匹配，至少需要 10+1+1+12=24 次比较，模式匹配成功的位置序号为 i-T[0]=23-12=11。

4.2.设四维数组 B[1..3，2..8，0..5，1..8]以行主序顺序方法存储在一个连续的存储空间内,每一个数据元素占 2 个存储单元,且B[1,2,4,1]的存储地址是2000， 则B[2,3,4,5]的存储地址是 _____。

若为列主序存储，则[2,3,4,5]的存储地址是 _____。

解答:

行序存储，B[2,3,4,5]的存储地址是：

2000+((2-1)×(8-2+1)×(5-0+1)×(8-1+1)+(3-2)×(5-0+1)×(8-1+1)+(4-4) ×(8-1+1)+(5-1))×2

=2000+(336+48+0+4)×2=2776

列序存储，B[2,3,4,5]的存储地址是：

2000+((5-1)×(5-0+1)×(8-2+1)×(3-1+1)+(4-4)×(8-2+1)×(3-1+1)+(3-2) ×(3-1+1)+(2-1))×2

=2000+(504+0+3+1)×2=3016

4.3.已知广义表 A=((a, (b ,c)), (a, (b ,c),d)),则运算 head(head (tail (A)))的结果是_____ 。

解答:

head(head (tail (A)))

= head(head (((a, (b ,c),d))))

= head((a, (b ,c),d))

= a

4.4. 利用广义表的 Head(L)和 Tail(L)的运算，将元素 c 从广义表 L=(((a, b), e,(c, d)))中分离出来，其运算表达式为_____ 。

解答:

Head(Head(Tail(Tail(Head(Head(L))))))

= Head(Head(Tail(Tail(Head(((a , b) , e,(c , d))))))))

= Head(Head(Tail(Tail(((a, b), e,(c, d))))))

= Head(Head(Tail((e,(c, d)))))

= Head(Head(((c, d))))

= Head((c, d))

= c

5.1. 某通信系统有八种字符： d、e、f、g、h、i、j，k，其权值分别为：0.24、0.10、0.03、0.15、0.09、0.06、0.29、0.04，请完成：

（1）构造 Huffman 树（要求所有结点左孩子的权值不大于右孩子的权值）；

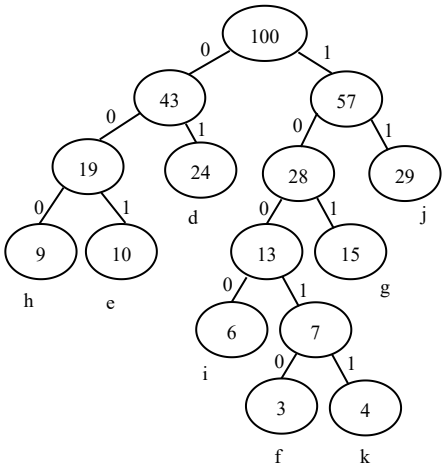
（2）据此设计出各个字符的 Huffman 编码；

（3）求出该 Huffman 树的带权路径长度 WPL；

（4）并译出下列报文：111000011011001010011。

解：

（1）Huffman 树：为表达方便，权值放大 100



（2）Huffman 编码

字符	Huffman 编码
d	01
e	001
f	10010
g	101
h	000
i	1000
j	11
k	10011

（3）求带权路径长度 WPL

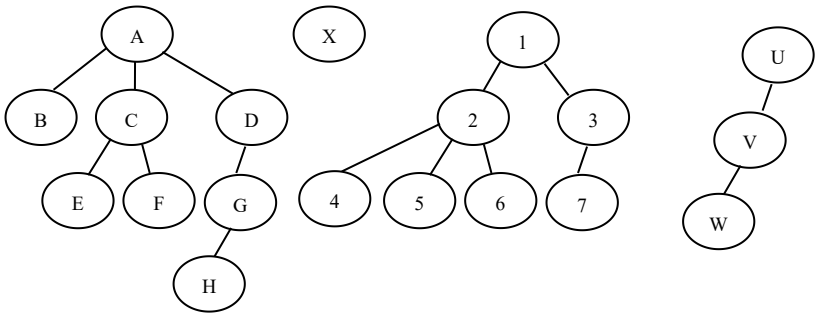
WPL=（（0.03+0.04）×5+0.06×4+（0.09+0.10+0.15）×3+（0.24+0.29）×2）=2.67

（4）译报文

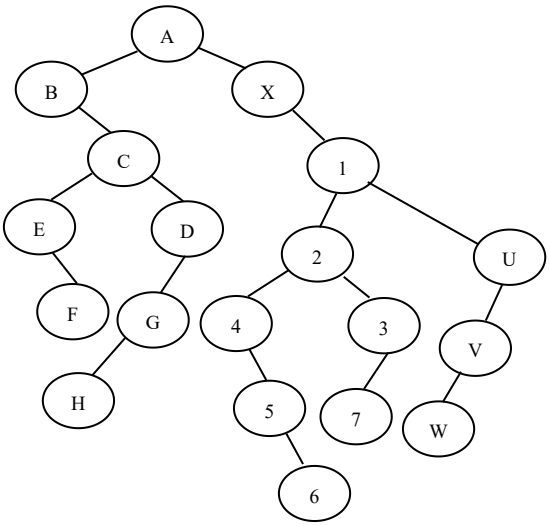
11	1000	01	101	10010	10011
j	i	d	g	f	k

报文译码为：

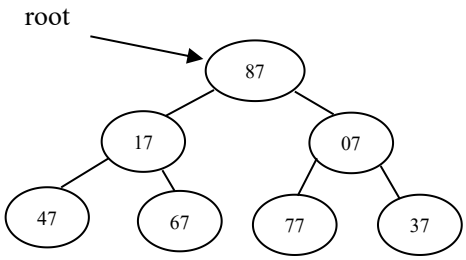
5.2.将下面的森林（F={T1，T2，T3，T4}）转换为对应的二叉树。



解：

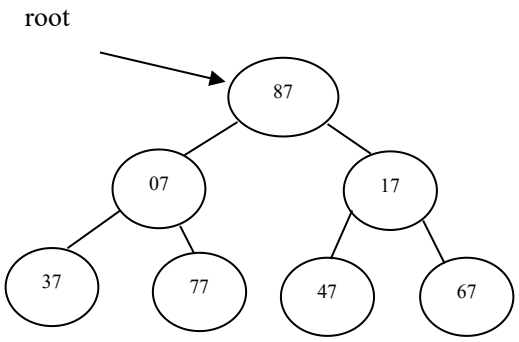


5.3. 以指向左侧二叉树的指针 root 作初始值，执行递归算法 ShiftTree(root)，请指出执行后的二叉树结构图。




```
ShiftTree( T )
{
      if (T == Null ) return( 0 );
      TL = T->Lchild;
      ShiftTree( TL);
      TR = T->Rchild;
      if ( ( TL <> Null ) && ( TR <> Null ) && ( TL->data > TR->data ) )
            {     T->Lchild = TR;
                  T->Rchild = TL;
            }
      ShiftTree( TR );
}
```

解：




```
0   ShiftTree( T )
1   {
2       if (T == Null ) return( 0 );
3       TL = T->Lchild;
4       ShiftTree( TL );
5       TR = T->Rchild;
6       if ( ( TL <> Null ) && ( TR <> Null ) && ( TL->data > TR->data ) )
7           {     T->Lchild = TR;
8                 T->Rchild = TL;
9           }
10      ShiftTree( TR );
11  }
```

5.4. 试证明：一棵非空的满 m 叉树上叶子结点数 n_0 和非叶子结点数 N 之间满足以下关系：$n_0 =（m - 1）*N + 1$

证明：

设分支总数为 B，结点总数为 M。

因为在满 m 叉树上，只存在度为 m 和度为 0 的结点，

所以，$B = m*N$

$M=n_0 + N$

又因为除了根结点外，每个结点有唯一的分支与之对应。

所以，$M = B + 1 = m*N + 1$

即有，$n_0 + N = m*N + 1$

也即，$n_0 =（m - 1）*N +1$

证毕。

总结：

1. 除了根结点外，每个结点有唯一的分支与之对应；
2. 满 m 叉树上，只存在度为 m 和度为 0 的结点。

5.5. 证明题

设结点 u 和结点 v 是树中的两个结点，且在对该树的先序遍历序列中 u 在 v 之前，而在其后序遍历序列中 u 在 v 之后，试证明结点 u 是结点 v 的祖先。

证明：用反证法证明本题：

假设 u 结点不是 v 结点的祖先结点，并设该树为 BT，其根结点为 r 结点。则 u 结点不在从 r 结点到 v 结点的路径上。

分两种情况讨论：

1. 若 u 结点是结点 v 的子树上的结点，则，在 BT 的先序遍历序列中，子树上的所有结点都在 v 结点之后，即 u 结点在 v 结点之后出现，故与原题条件矛盾，所以 u 结点不能是 v 的子树上的结点。

2. 若 u 结点不是结点 v 的子树上的结点，即 u 结点不为 v 结点的子孙结点,可设从 r 结点到 v 结点的路径序列为：r，r_1，r_2，…，r_k，v

即，r，r_1，r_2，…，r_k 是从 r 结点到 v 结点的路径上的结点，都是 v 结点的祖先结点。

则，u 结点只能在以 r ，r1，r2，…，rk 为根的，且不包含 v 结点作为子孙结点的子树中。

对 r，r_1，r_2，…，r_k 中的任意一个结点 x，

若 v 结点在 x 结点的子树 Xi 上，u 结点在 x 结点的子树 Xj 上，其中 i<j。于是，在对以 x 结点为根的树做先序遍历时，v 结点应在 u 结点之前出现（x 结点为根的先序遍历是 BT 的先序遍历序列的子序列），从而与原题的条件矛盾；

若 u 结点在 x 结点的子树 Xi 上，v 结点在 x 结点的子树 Xj 上，其中 i<j。于是，在对以 x 结点为根的树做后序遍历时，u 结点应在 v 结点之前出现（x 结点为根的后序遍历是 BT 的后序遍历序列的子序列），从而与原题的条件矛盾。

所以，u 结点不能是 r，r_1，r_2，…，r_k 以外的结点，只能是 r，r_1，r_2，…，r_k 中的某个结点，即 u 结点是 v 结点的祖先结点。证毕。

要点：1. 先序遍历和后序思想；

2. 结点的层次关系；

3. 证明思路清晰。

5.6．证明题

设结点 u 和结点 v 是树中的两个结点，且结点 u 是结点 v 的祖先。试证明在对该树的先序遍历序列中 u 在 v 之前，而在其后序遍历序列中 u 在 v 之后。

证明：

树的先序遍历算法：先访问树的根结点，然后依次先序遍历根的每棵子树。

树的后序遍历算法：先依次后序遍历根的每棵子树，然后访问树的根结点。

因为结点 u 是结点 v 的祖先，则以结点 u 为根的子树必包括结点 v，v 是 u 的子树。

根据树的先序遍历算法，当遍历到以结点 u 为根的子树时，第一个遍历的结点为 u，v 必然在 u 的后面，即对该树的先序遍历序列中 u 在 v 之前。

根据树的后序遍历算法，当遍历到以结点 u 为根的子树时，最后一个遍历的结点为 u，v 必然在 u 的前面，即对该树的后序遍历序列中 u 在 v 之后。

故命题得证。

要点：

1、说明树的先序遍历算法、后序遍历算法。

2、论证树的先序遍历序列中 u 在 v 之前。

3、论证树的后序遍历序列中 u 在 v 之后。

5.7．证明题

设一棵度为 k 的非空树上的叶子结点数为 n_0，度为 i 的结点数为 n_i（$1 \le i \le k$），试证明以下关系成立。

$$n_0 = 1 + \sum_{i=1}^{k}(i-1)* n_i$$

证明：

设分支总数为 B，结点总数为 N。

根据题意，有

$B= n_1 + 2*n_2 + \ldots + k*n_k$

$N= n_0 + n_1 + n_2 + \ldots + n_k$

又因为**除了根结点外，每个结点有唯一的分支与之对应**。

所以，$N = B + 1$

即有，$n_0 + n_1 + n_2 + \ldots + n_k = n_1 + 2*n_2 + \ldots + k*n_k + 1$

也即，

$$n_0 = 1 + \sum_{i=1}^{k}(i-1) * n_i$$

证毕。

要点：1. **除了根结点外，每个结点有唯一的分支与之对应**；

2. 结点总数 = 分支总数+1。

5.8．证明题

试证明：若一个具有 p 个结点、q 条边的非连通无向图是一个森林(p>q)，则该森林中必有 p-q 棵树。

证明：

由题目知，一个具有 p 个结点、q 条边的非连通无向图是一个森林(p>q)。

设该森林有 x 棵树。这些树的结点数分别为 n_1、n_2、n_3、……、n_x。

因为除了根结点以外，树的每个结点有唯一的分支（即边）与之对应，即结点数=边数+1。

所以这些树的边数分别为 n_1-1、n_2-1、n_3-1、……、n_x-1。

因为 $p=n_1+n_2+n_3+\ldots\ldots+n_x$

所以

$q= (n_1-1)+(n_2-1)+(n_3-1)+\ldots\ldots+(n_x-1)=(n_1+n_2+n_3+\ldots\ldots+n_x) – x=p – x,$

即 $x= p– q$

故该森林中必有 p-q 棵树。

证毕。

关键点：结点与分支（边）的关系；计算结点数与边数；推导过程。

5.9. 试用归纳法证明：高度为 h 的二叉树的结点总数不超过 2^h-1

证明：

对高度 h 采用第一归纳法。令 n 为二叉树的结点总数。

当 h=1 时，二叉树只含有根结点，所以，$n=2^1-1=1$ 成立。

假设 h=m-1 时（m>1），有 $n\leq 2^{m-1}-1$。

则当 h=m 时，由于高度为 m 的二叉树可在高度为 m-1 的二叉树上增加一层结点构成。

由于高度为 m-1 的二叉树的第 m-1 层上最多有 2^{m-2} 个结点，而每个结点最多引出两个分支，所以第 m 层上最多有 $2\times 2^{m-2}=2^{m-1}$ 个结点。

所以 $n\leq 2^{m-1}-1+2^{m-1}=2^m-1$ 结点。结论成立，即高度为 h 的二叉树的结点总数不超过 2^h-1。

证毕。

5.10. 对于那些所有非叶子结点（分支结点、非终端结点）均含有左右子树的二叉树 BT(即只有度为 2 和度为 0 的结点)：

（1）试问：具有 n 个叶子结点的这样的二叉树中共有多少个结点？

（2）试证明：$\sum_{i=1}^{n}2^{-(h_i-1)}=1$，其中 n 为叶子结点的个数，$h_i$ 表示第 i 个叶子结点所在的层次（设根结点所在层次为 1）

解答：

（1）设度为 2 的结点数为 n_2，由二叉树的性质 3（任何一棵二叉树的叶子结点数等于度为 2 的结点数+1），可得 $n=n_2+1$，则 BT 的结点总数为 $n+n_2=n+(n-1)=2n-1$。

（2）证明：

该问题的证明似乎可以采用归纳法证明，但由于本问题的问题规模（n，h_i）的规律很难掌握，所以归纳法证明非常困难。因为很难确定 n 个叶子结点分布在 BT 的哪个层次上，所以对 n 采用归纳法有难度，对 h_i 采用归纳法也受制于叶子结点数 n。下面采用计算的方法加以证明。

设叶子结点的最深层为 h。对出现在小于 h 层的每一个叶子结点，以该结点为根，向下补充结点，构造一棵满二叉树且使叶子结点出现在 BT 的第 h 层，且 BT 中不存在度为 1 的结点，所以可以得到一个深度为 h 的满二叉树 BT'。

根据满二叉树的性质，深度为 h 的满二叉树 BT'上共有 2^h-1 个结点，原 BT 的结点数为 2n-1，故得新补充的结点总数为$(2^h-1)-(2n-1)$。另外，以 BT 中每个原叶子结点为子树的根与它向下所补充的结点，一起也构成一棵深度为 k 的满二叉树，具有 2^k-1 个结点（每个原叶子结点向下所补充的结点数为 $2^k-1-1=2^k-2$，叶子结点在最深层 h 时，补充了 0 个结点），其中，深度 $k=h-h_i+1=h-(h_i-1)$。所以新补充的结点总数为：

$\sum_{i=1}^{n}(2^k-2)=\sum_{i=1}^{n}2^{h-(h_i-1)}-2n=(2^h-1)-(2n-1)$，整理得 $\sum_{i=1}^{n}2^{h-(h_i-1)}=2^h$，即

$$\sum_{i=1}^{n}2^{-(h_i-1)}=1$$

证毕。

5.11. 试证明：在任意非空二叉树 T 上，分支结点数 < 叶子结点数的充分必要条件是二叉树 T 上不存在度为 1 的结点。

证明：

在任意非空二叉树 T 上，设度为 2 的结点数是 n_2，度为 1 的结点数是 n_1，度为 0 的结点数（即叶子数）是 n_0，结点总数 n， 分支总数 B。所以分支结点数=n_2+n_1。

∵ 二叉树中结点总数 $n=n_0+n_1+n_2$（叶子数＋1 度结点数＋2 度结点数）

又∵二叉树中结点总数 n＝B+1 （ 分支总数＋根结点 ）

（因为除根结点外，每个结点必有一个直接前趋，即一个分支）

而分支总数 $B= n_1+2n_2$（1 度结点必有 1 个直接后继，2 度结点必有 2 个直接后继）

上面三式联立可得： $n_0+n_1+n_2= n_1+2n_2 +1$, 即 $n_0=n_2+1$

（1）必要条件（分支结点数 < 叶子结点数的必要条件是二叉树 T 上不存在度为 1 的结点），即证明：若分支结点数 < 叶子结点数，则二叉树 T 上不存在度为 1 的结点。

因为分支结点数 < 叶子结点数，即 $n_2+n_1 < n_0=n_2+1$，整理得 $n_1 < 1$，由于 n_1 为非负整数，故 $n_1 = 0$，即二叉树 T 上不存在度为 1 的结点。必要条件得证。

（2）充分条件（分支结点数 < 叶子结点数的充分条件是二叉树 T 上不存在度为 1 的结点），即证明：若二叉树 T 上不存在度为 1 的结点，则分支结点数 < 叶子结点数。

因为二叉树 T 上不存在度为 1 的结点，即 $n_1 = 0$，由于 n_1 为非负整数，

得 $n_1 < 1$，两边加 n_2，得 $n_2+n_1 < n_2+1$，因为 $n_0=n_2+1$，故 $n_2+n_1 < n_0$，即分支结点数 < 叶子结点数。

充分条件得证。

故命题得证。

5.12. 试证明：若森林 F 中有 n 个非终端结点，B 是由 F 变换得到的二叉树，则 B 中右指针域为空的结点有 n+1 个。

证明：

本题思路：右指针域为空的个数 =空指针域总数 - 左指针域为空的个数

（1）题目非终端结点数为 n，那么假设森林总结点数为 m，终端结点（即叶子结点）数为 m-n，指针域总数就是 2*m。

（2）除根结点外，每个结点直接都被其父结点指向，使用的指针域为 m-1 个，所以空指针域总数为 2*m -（m-1） = m+1

（3）m-n 个终端结点转化为二叉树后，都没有左孩子，左指针域就为空，即左指针域为空的个数是 m-n。因为森林转二叉树后，左孩子是第一个孩子，右孩子是兄弟，所以终端结点转化后一定无左孩子，右孩子可能有也可能没有，即转化后有两种可能：a）仍为叶子结点，b）不再是叶子结点，但只有右孩子。

所以，右指针域为空的个数=空指针域总数 - 左指针域为空的个数

=m + 1 -（m - n） = n + 1。

命题得证。

7.1.1. 哈希表问题

哈希函数 Hash (Key)=Key MOD 13，处理冲突函数 Hi=(Hash(key)+di) mod m (m 为哈希表长度，d_i=1,2,3,4,5,……)。

给定关键字序列：14，02，24，29，01，33，79，41，53，46，68，90。

试在 S.elem[0..15] 存储空间上，解答如下问题：

（1）构造如上给定关键字序列的哈希表，统计每个关键字的查找次数

（2）计算其查找成功时的平均查找长度 ASL

（3）计算其查找不成功时的平均查找长度 ASL

7.1.1 解：处理冲突函数 Hi=(Hash(key)+di) mod m (m 为哈希表长度 16，d_i=1,2,3,……)

（1）构造哈希表及每个关键字的查找次数

哈希表 S.elem[0..15]

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
关键字		14	02	29	01	79	41	33	53	46	68	24	90			
查找成功时的比较次数		1	1	1	4	5	5	1	8	3	8	1	1			

（2）计算关键字查找成功时的平均查找长度 ASL=（Σ各关键字查找成功时的比较次数）/（关键字的个数）

查找成功时的 ASL=（1+1+1+4+5+5+1+8+3+8+1+1）/ 12 = 39/12 =13/4= 3.25

（3）计算关键字查找不成功时的平均查找长度 ASL=（Σ各地址查找不成功时的比较次数）/（不成功地址个数）

根据哈希函数 Hash (Key)=Key MOD 13，则第一次计算的哈希地址值只可能是 0~12，这些哈希地址查找不成功的比较次数为它到第一个没有存储关键字的地址的比较次数。因此查找不成功的比较次数如下表所示：

地址	0	1	2	3	4	5	6	7	8	9	10	11	12
查找不成功时的比较次数	1	13	12	11	10	9	8	7	6	5	4	3	2

查找不成功时的 ASL=（1+13+12+11+10+9+8+7+6+5+4+3+2）/13 = 91/13=7

7.1.2 哈希表问题

哈希函数 Hash (Key)=Key MOD 13，处理冲突函数 Hi=(Hash(key)+di) mod m（若 Hi<0，则 Hi=Hi+m），m 为哈希表长度 16，d_i=1，-1，2，-2，3，-3，4，-4，……）。

给定关键字序列：14，02，24，29，01，33，79，41，53，46，68，90。

试在 S.elem[0..15] 存储空间上，解答如下问题：

（1）构造如上给定关键字序列的哈希表，统计每个关键字的查找次数

（2）计算其查找成功时的平均查找长度 ASL

（3）计算其查找不成功时的平均查找长度 ASL

7.1.2. 解：处理冲突函数 Hi=(Hash(key)+di) mod m（若 Hi<0，则 Hi=Hi+m），m 为哈希表长度 16，d_i=1，-1，2，-2，3，-3，……

（1）构造哈希表及每个关键字的查找次数。给定关键字序列为 14，02，24，29，01，33，79，41，53，46，68，90。

哈希表 S.elem[0..15]={01,14,02,29,41,68,#,33,46,#,#,24,90,#,53,79}

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
关键字	01	14	02	29	41	68		33	46			24	90		53	79
查找成功时的比较次数	3	1	1	1	4	4		1	2			1	1		7	5

（2）计算关键字查找成功时的平均查找长度 ASL=（Σ各关键字查找成功时的比较次数）/（关键字的个数）

查找成功时的 ASL=（3+1+1+1+4+4+1+2+1+1+7+5）/ 12 = 31/12 ≈ 2.58

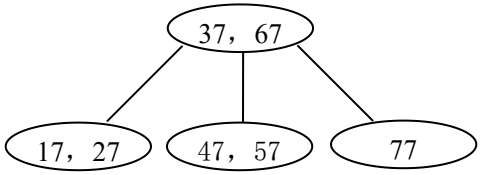
（3）计算关键字查找不成功时的平均查找长度 ASL=（Σ各地址查找不成功时的比较次数）/（不成功地址个数）

根据哈希函数 Hash (Key)=Key MOD 13，则第一次计算的哈希地址值只可能是 0~12，这些哈希地址查找不成功的比较次数为它到第一个没有存储关键字的地址的比较次数。因此查找不成功的比较次数如下表所示：

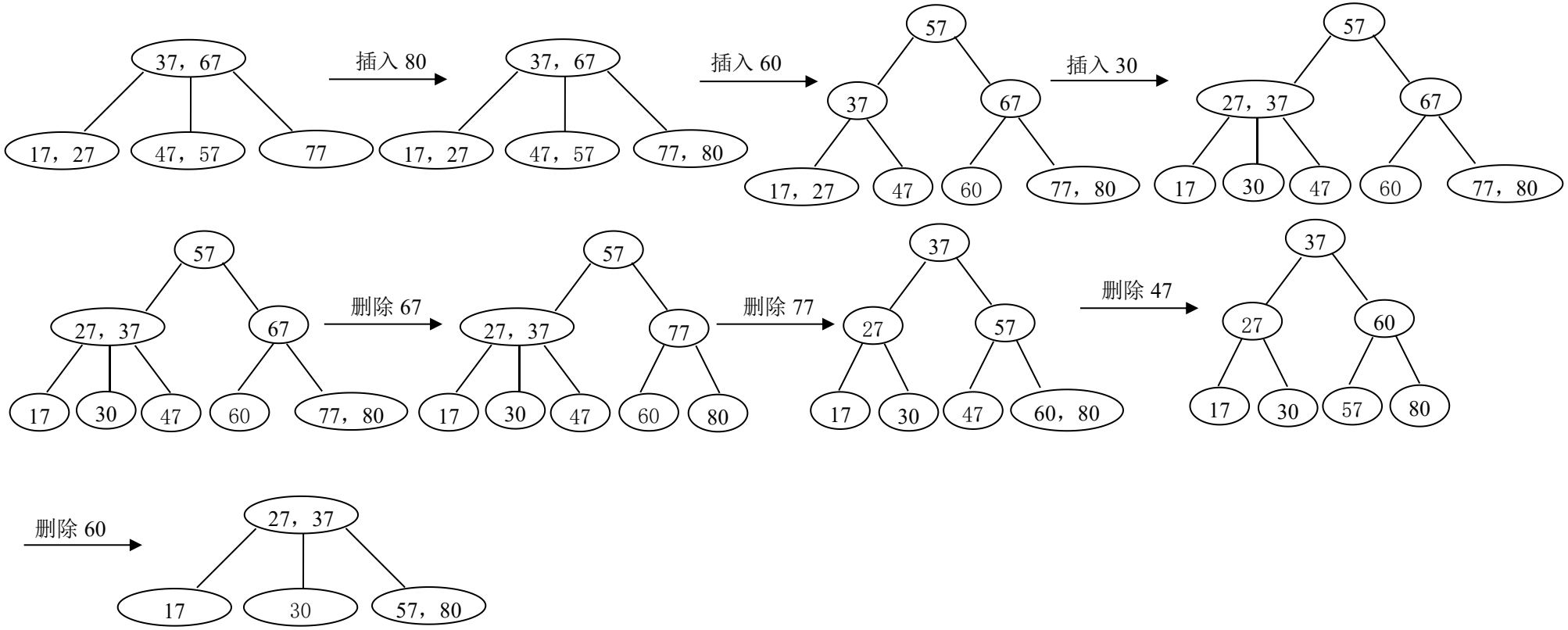
地址	0	1	2	3	4	5	6	7	8	9	10	11	12
查找不成功时的比较次数	7	9	8	6	4	2	1	3	2	1	1	3	2

查找不成功时的 ASL=（7+9+8+6+4+2+1+3+2+1+1+3+2）/13 = 49/13 ≈ 3.77

7.2. 请在下面的 3 阶（2-3）B_树上依次插入关键字 80、60、30，然后再依次删除关键字 67、77、47、60。试画出每次操作后的 B_树结构。



7.2. 解：


37，67
17，27
47，57
77
插入 80
37，67
17，27
47，57
77，80
插入 60
57
37
67
17，27
47
60
77，80
插入 30
57
27，37
67
17
30
47
60
77，80
57
27，37
67
17
30
47
60
77，80
删除 67
57
27，37
77
17
30
47
60
80
删除 77
37
27
57
17
30
47
60，80
删除 47
37
27
60
17
30
57
80
删除 60
27，37
17
30
57，80

7.3. 平衡二叉树

已知关键字序列为｛47, 57, 87, 77, 67, 27, 07, 97, 57, 37, 17｝，其中，_x_表示值为 *x* 的另一关键字。按如下要求完成本题。

（1）针对给定关键字序列的不同排列，所构造出的不同形态的二叉排序树中，在最好和最坏情况下，该二叉排序树的高度各是多少？

（2）根据给定的关键字序列，构造一棵平衡二叉排序树。

（3）在等概率的情况下，计算查找成功时该平衡二叉排序树的 $ASL_{成功}$。

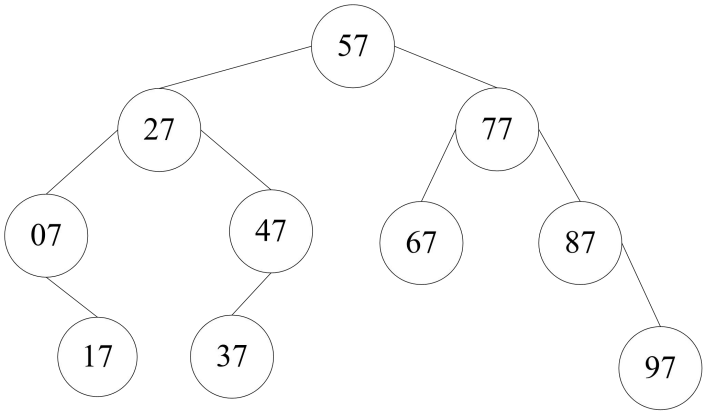
（4）在等概率的情况下，计算查找不成功时该平衡二叉排序树的 $ASL_{不成功}$。

解：

（1）最好情况下的二叉排序树的高度为：4，最坏情况下的二叉排序树的高度为：10

注意：二叉排序树最好情况下的高度为平衡二叉树的高度，最坏情况下的高度为不同的关键字个数。

（2）构造一棵平衡二叉排序树



详细构造过程如下页：

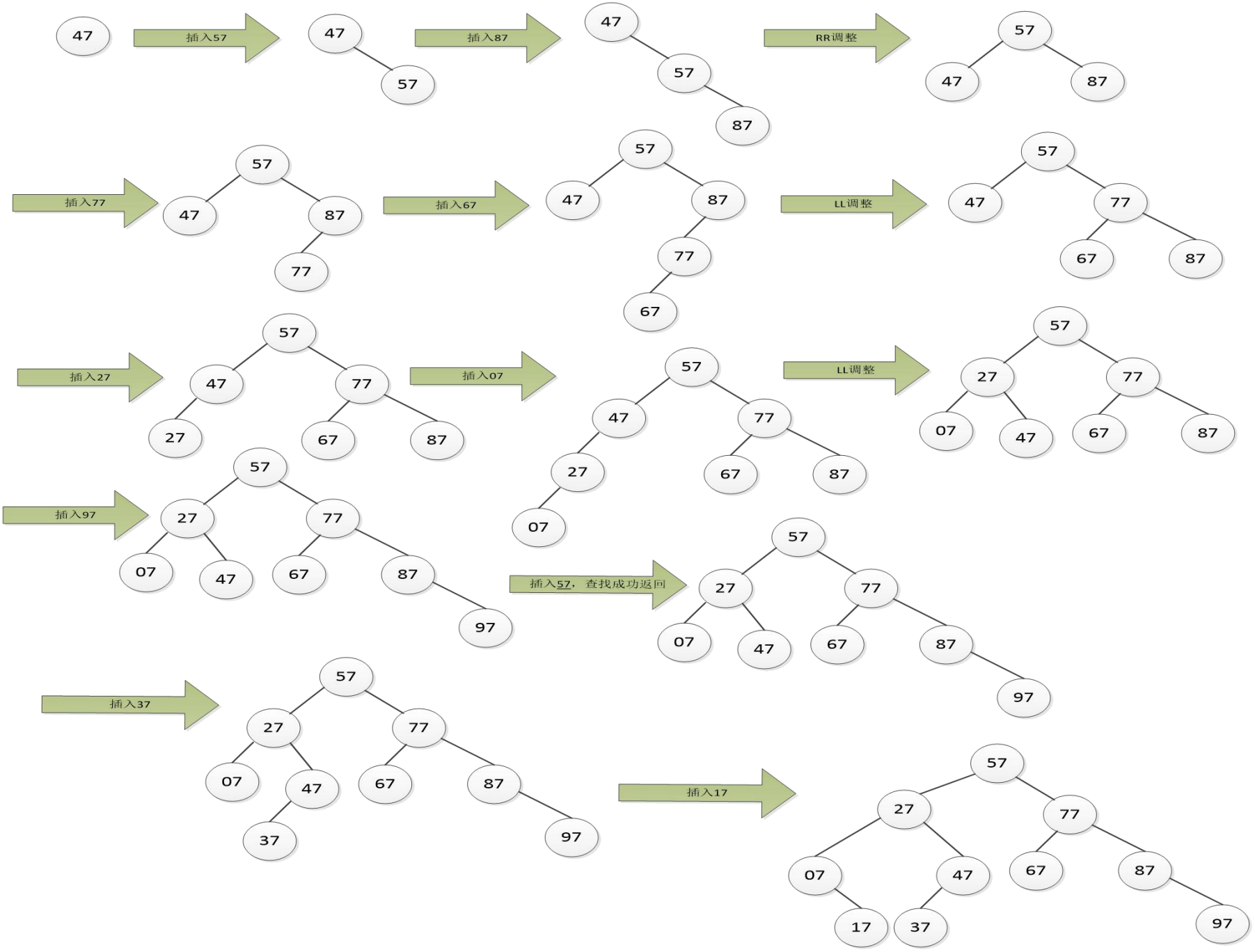
（3）查找成功时的平均查找长度

$ASL_{成功} = (3\times4+4\times3+2\times2+1\times1)/10=2.9$

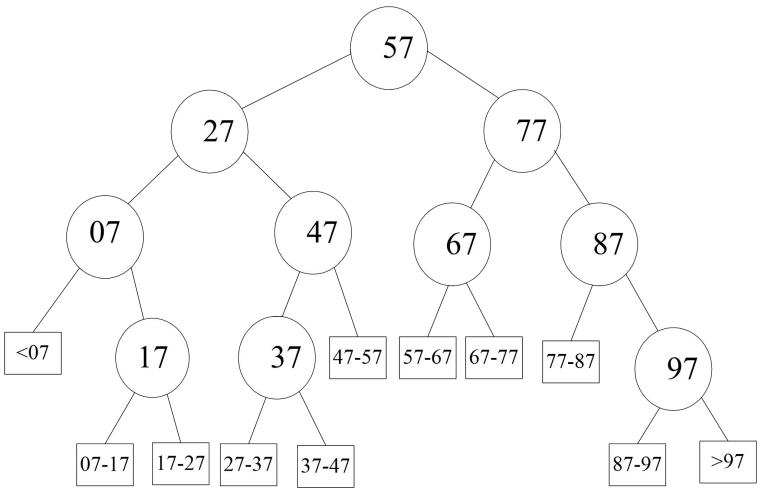
（4）查找不成功时的平均查找长度

$ASL_{不成功} = (5\times3+6\times4)/11=39/11$

平衡二叉树构造过程：关键字序列为｛47, 57, 87, 77, 67, 27, 07, 97, 57, 37, 17｝，构造一棵平衡二叉树。



7.4. 求折半查找判定树的平均查找长度



根据上图的折半查找判定树，解答如下问题：

（1）在等概率的情况下，计算查找成功时该判定树的平均查找长度 ASL。

（2）在等概率的情况下，计算查找不成功时该判定树的平均查找长度 ASL。

解：

折半查找判定树满足以下两点：

1. 折半查找判定树是一棵二叉排序树，即每个结点的值均大于其左子树上所有结点的值，小于其右子树上所有结点的值；

2. 折半查找判定树中的结点（圆形结点）都是查找成功的情况，将每个结点的空指针指向一个实际上并不存在的结点——称为外结点，所有外结点即是查找不成功的情况（方框形状的结点）。如果有序表的长度为 n，则外结点一定有 n+1 个。

（1）在等概率的情况下，查找成功时该判定树的 ASL。

在折半查找判定树中，某结点所在的层数即是查找该结点的比较次数，整个判定树代表的有序表的平均查找长度即为查找每个结点的比较次数之和除以有序表的长度。

ASL=(1×1+2×2+3×4+4×3)/10=29/10

（2）在等概率的情况下，查找不成功时该判定树的 ASL。

在折半查找判定树中，查找不成功时的比较次数即是查找相应外结点时与内结点的比较次数。整个判定树代表的有序表在查找失败时的平均查找长度即为查找每个外结点的比较次数之和除以外结点的个数。

ASL=(3×5+4×6)/11=39/11

7.5. 试证明：在由 n(n>0)个结点构成的有序表中进行折半查找，其最坏情况下与关键字比较的次数不超过由 n 个结点所构成的完全二叉树的深度。

证明：

根据完全二叉树的性质 4，可以知道：

具有 n 个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor+1$ 。

考虑具有 n 个关键字的有序表 S[1..n]，对 S 采用折半查找方法查找表中的元素是，首先查找的是 S[(1+n)/2]。

当查找不成功时，继续在左或右边的一半中继续查找。如此重复，直到查找成功或查找不成功时，查找过程结束。

当查找成功时，查找过程结束在一个内部结点上；

当查找不成功时，查找过程结束在一个外部结点上。

根据上述查找过程，可以得到一棵具有 n 个结点的判定树。

设判定树的高度为 h，则外部结点全部出现在第 h 层和第 h+1 层上

考虑最坏的情况，查找不成功时需要比较的最多次数 h

由于**具有 n 个结点的二叉判定树的深度为$\lfloor \log_2 n \rfloor+1$，即 $h=\lfloor \log_2 n \rfloor+1$**。

所以，最坏情况下，与关键字比较的次数不超过由 n 个结点所构成的完全二叉树的深度。

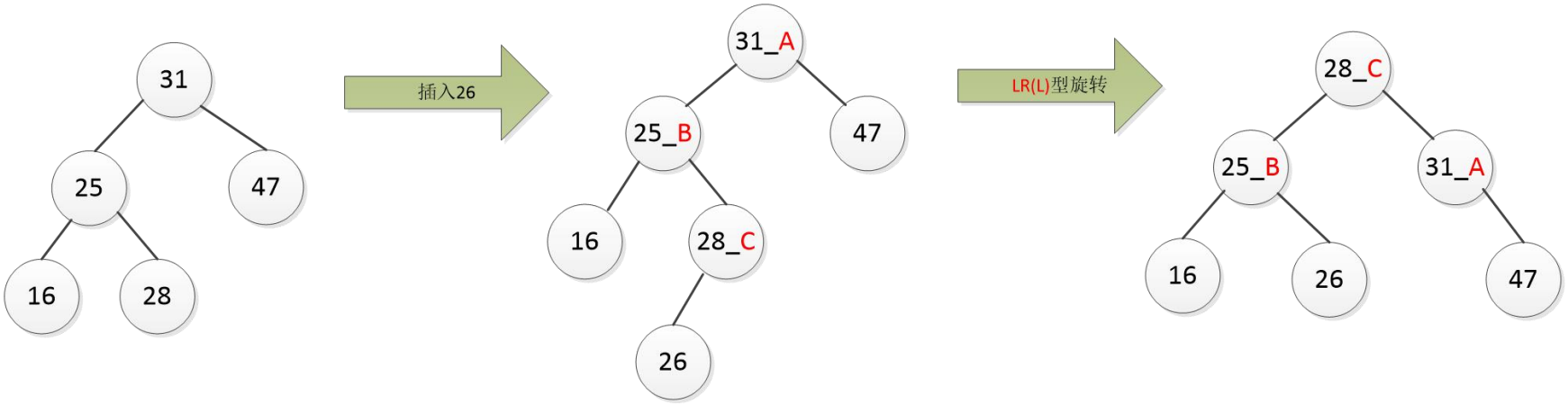
证毕。

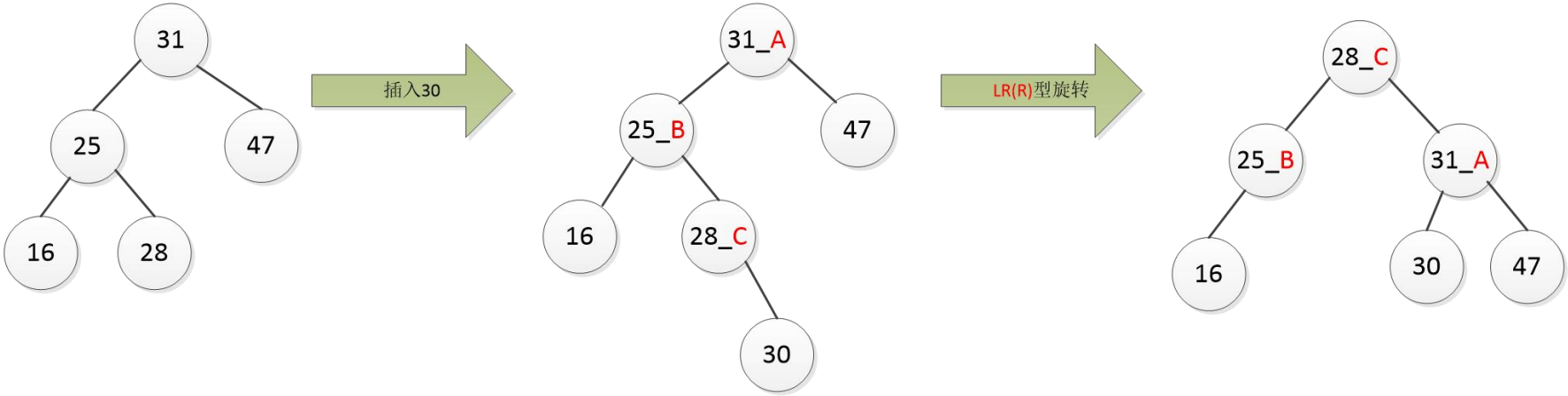
要点：1. **具有 n 个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor+1$**；

2. **具有 n 个结点的二叉判定树的深度为$\lfloor \log_2 n \rfloor+1$**。

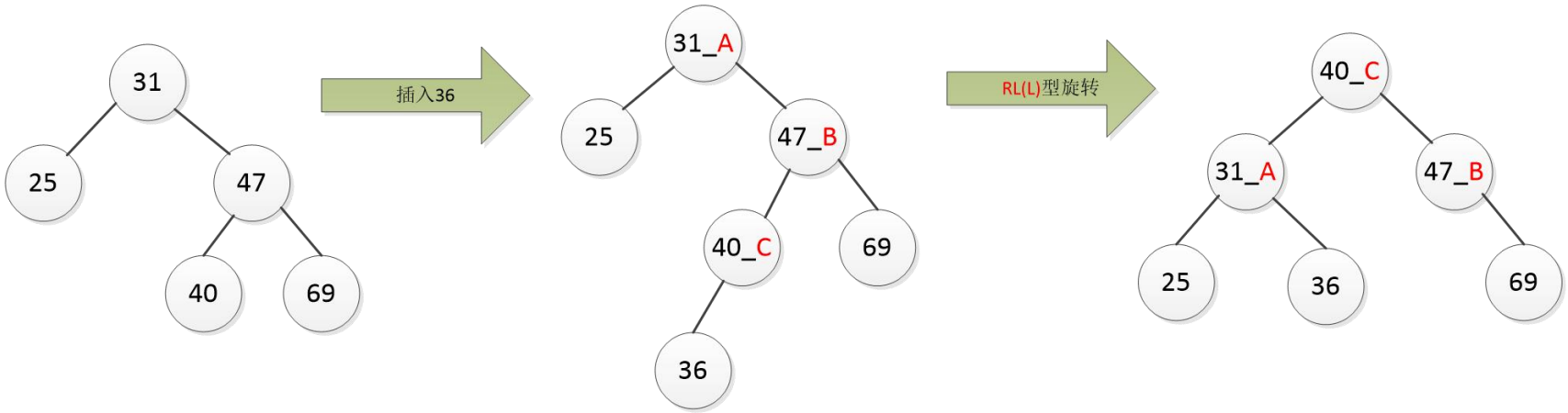
7.6 平衡二叉树（教材例题第 207 页~209 页）

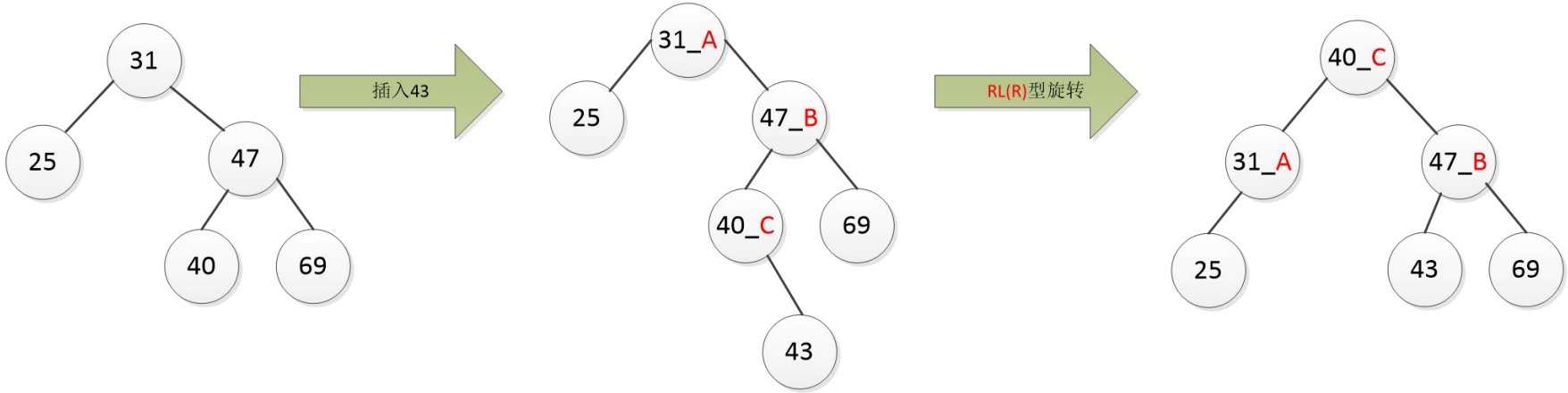
（1）LR 型旋转（LR(L)型、LR(R)型）（教材第 208 页图 7.18）





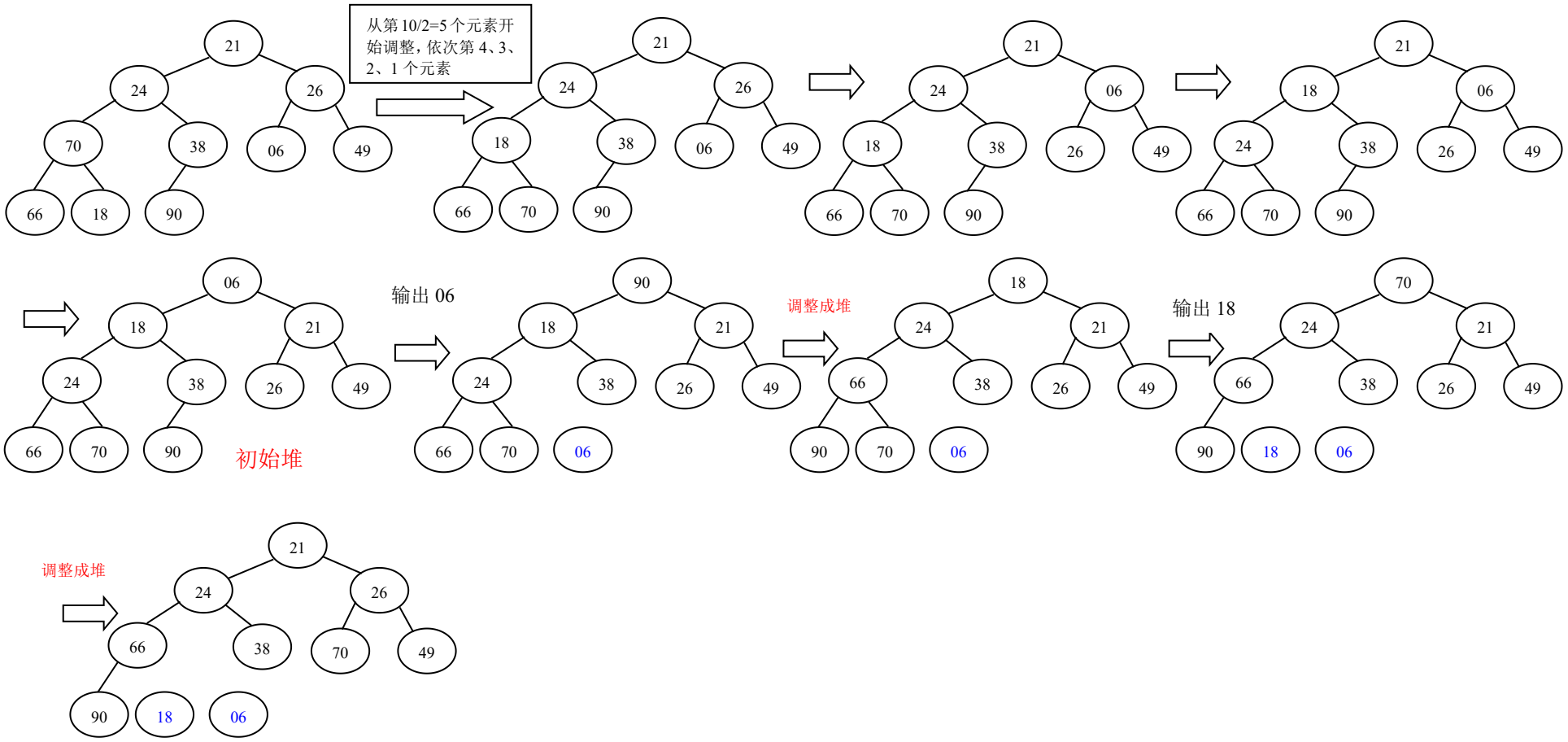
（2）RL 型旋转（RL(L)型、RL(R)型）（教材第 209 页图 7.20）





8.1. 将下列给定的关键字序列（21，24， 26， 70， 38， 06， 49， 66， 18， 90）调整成一个堆，使其满足 $K_i \leqslant K_{2i}$ 且 $K_i \leqslant K_{2i+1}$ ，并画出从初始堆到输出关键字 18 后的过程中形成的所有堆，并给出堆的完全二叉树的顺序存储表示形式。

解：



给出堆的完全二叉树的顺序存储表示形式											
初始堆	06	18	21	24	38	26	49	66	70	90	
输出第 1 个元素 06 后的堆	18	24	21	66	38	26	49	90	70	06	
输出第 2 个元素 18 后的堆	21	24	26	66	38	70	49	90	18	06	本题已完成
输出第 3 个元素 21 后的堆	24	38	26	66	90	70	49	21	18	06	继续排序
输出第 4 个元素 24 后的堆	26	38	49	66	90	70	24	21	18	06	
输出第 5 个元素 26 后的堆	38	66	49	70	90	26	24	21	18	06	
输出第 6 个元素 38 后的堆	49	66	90	70	38	26	24	21	18	06	
输出第 7 个元素 49 后的堆	66	70	90	49	38	26	24	21	18	06	
输出第 8 个元素 66 后的堆	70	90	66	49	38	26	24	21	18	06	
输出第 n-1=9 个元素 70 后	90	70	66	49	38	26	24	21	18	06	排序全部结束