第二章

第二章

2.2设有数据结构（D，R），其中

D={d1, d2, d3, d4 }

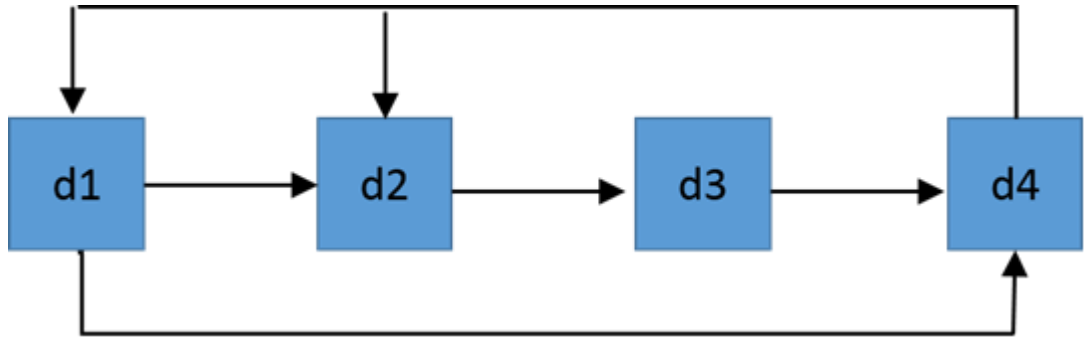
R={r1, r2}

r1={ <d1, d2>, <d2, d3>, <d3, d4>, <d1, d4>, <d4, d2>, <d4, d1> }

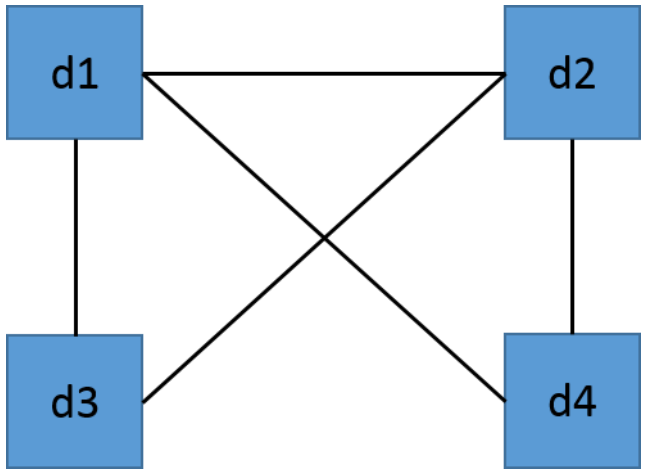
r2={ (d1, d2), (d1, d3), (d1, d4), (d2, d4), (d2, d3) }

试绘出其逻辑结构示意图。

r1:



r2:



2.3设n是正整数。试写出下列程序段中用记号“△”标注的语句的频度：

```
(1)     i=1; k=0;
        while(i<=n-1) {
        △       k+=10*i;
                i++;
        }
```

频度：n-1(n>=1)

```
(2)     i=1; k=0;
        do {
        △       k+=10*i;
                i++;
        }while(i<=n-1)
```

频度： n>=2时n-1； n=1时1 // do 语句先执行

```
(3)     i=1; k=0;
        do {
        △       k+ = 10*i; i++;
        }while(i==n);
```

频度：n=2时候，频度为2，否则频度为1 //分类讨论

(4)　　i=1; j=0;
　　while(i+j≤n) {
　　△　　if(i<j) i++; else j++;//此处是执行if else语句的频度
　　}

频度：n

(5)　　x=n; y=0; //n是不小于1的常数
　　while(x>=(y+1)*(y+1)){
　　△　　y++;
　　}

频度：round(sqrt(n)) //round代表向下取整函数

(6)　　x=91; y=100;
　　while (y>0)
　　△　　if(x>100) { x-=10; y--; }
　　else x++ ;
　　}

频度：1100次 //x每次增加到 101，y--，注意计算if else 语句 频度

(7)
```
for( i=0; i<n; i++)
    for( j=i; j<n; j++)
        for( k=j; k<n; k++)
△           x+=2;
```

频度：

$$\sum_{i=1}^{n}\frac{i^2}{2}+\frac{1}{4}n(n+1)=\frac{n(n+1)(n+2)}{6}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}\frac{i\times(i+1)}{2}=\frac{1}{2}\sum_{i=1}^{n}(i^2+i)$$

$$=\frac{1}{2}\left(\frac{n\times(n+1)\times(2n+1)}{6}+\frac{n\times(n+1)}{2}\right)$$

$$=\frac{n\times(n+1)\times(2n+1)}{12}+\frac{n\times(n+1)}{4}$$

$$=\frac{n(n+1)(n+2)}{6}$$

第三章 栈和队列

3.7 假设一个算术表达式中可以包含三种符号：圆括号“(”和“)”、方括号“[”和“]”、花括号“{”和“}”，且这三种括号可按任意次序嵌套使用。编写判别给定表达式中所含的括号是否正确配对的算法(已知表达式已存入数据元素为字符的顺序表中)。

```
Status Compare(){
        char c,e;
        SqStack S;
        InitStack(S);
        printf("请输入算术表达式:\n");
        while((c = getchar()) != '\n'){
                switch(c){
                        case '(':
        //左括号，直接入栈
                        case '[':
                        case '{':
                                Push(S,c);
                                break;
                        case ')':
        //右括号，与栈顶元素做比对
                                if(Pop(S,e) == ERROR || e != '(')
                                        return FALSE;
                                break;
                        case ']':
                                if(Pop(S,e) == ERROR || e != '[')
                                        return FALSE;
                                break;
                        case '}':
                                if(Pop(S,e) == ERROR || e != '{')
                                        return FALSE;
                                break;
                        default :
                                break;
                }
        }
        if(isEmpty(S))
                return TURE;
        else
                return FALSE;
}
```

3.8 设表达式由单字母变量、双目运算符和圆括号组成（如:“(a*(b+c)-d)/e)”。试写一个算法，将一个书写正确的表达式转换为逆波兰式。

(1)首先，需要分配2个栈，栈s1用于临时存储运算符（含一个结束符号），此运算符在栈内遵循越往栈顶优先级越高的原则；栈s2用于输入逆波兰式，为方便起见，栈s1需先放入一个优先级最低的运算符，在这里假定为'#'；

(2)从中缀式的左端开始逐个读取字符x，逐序进行如下步骤：

1.若x是操作数，则分析出完整的运算数（在这里为方便，用字母代替数字），将x直接压入栈s2；

2.若x是运算符，则分情况讨论：

若x是'('，则直接压入栈s1；

若x是')'，则将距离栈s1栈顶的最近的'('之间的运算符，逐个出栈，依次压入栈s2，此时抛弃'('；

若x是除'('和')'外的运算符，则再分如下情况讨论：

若当前栈s1的栈顶元素为'('，则将x直接压入栈s1；

若当前栈s1的栈顶元素不为'('，则将x与栈s1的栈顶元素比较，若x的优先级大于栈s1栈顶运算符优先级，则将x直接压入栈s1。否者，将栈s1的栈顶运算符弹出，压入栈s2中，直到栈s1的栈顶运算符优先级别低于（不包括等于）x的优先级，或栈s2的栈顶运算符为'('，此时再则将x压入栈s1;

(3)在进行完(2)后，检查栈s1是否为空，若不为空，则将栈中元素依次弹出并压入栈s2中（不包括'#'）；

(4)完成上述步骤后，栈s2便为逆波兰式输出结果。但是栈s2应做一下逆序处理，因为此时表达式的首字符位于栈底；

```
int compare_dominant(char a, char b)
{//比较运算符的优先级

    if(a == '+' || a == '-' && b == '/' || b == '*')
        return 0;
    if(a == '/' || a == '*' && b == '+' || b == '-')
        return 1;
    if(a == '+' || a == '-' && b == '+' || b == '-')
        return 0;
    if(a == '/' || a == '*' && b == '/' || b == '*')
        return 0;
}
```

```
void trans_post_exp(char exp[], char newexp[])
{//将中缀表达式（已存储在数组中）转换成后缀表达式，将后缀表达式存储在另一个数组中；
  int i=0;
  char stack[maxsize]; int top = -1;
  for(int k=0; exp[k] != '#'; k++)
  {
    //扫描到左括号，则直接入栈
    if(exp[k] == '('
      stack[++top] = exp[k];
    //若扫描到右括号，则令栈顶元素出栈，并存入新数组中
    else if(exp[k] == ')' && top != -1)
    {
      newexp[i++] = stack[top--];
      if(stack[top] == '('
          top--;
    }
    //若扫描到字母，则直接存入新数组中；
    else if(exp[k] != '+' && exp[k] != '-' && exp[k] != '/' && exp[k] != '*')
      newexp[i++] = exp[k];

    //若扫描到运算符，则将其与栈顶运算符比较优先级，如果新运算符优先级较高，将其入栈，
    //否则栈顶元素出栈并入新数组，再将新运算符入栈
    else
    {
      if(top == -1 || stack[top] == '(')  //栈空或者栈顶元素为左括号时直接入栈
        stack[++top] = exp[k];

      else if(compare_dominant(exp[k], stack[top]) == 1)
        stack[++top] = exp[k];
      else //新运算符优先级不高于栈顶元素时一直出栈，直到新运算符优先级高于栈顶元素
      {
        while(compare_dominant(exp[k], stack[top]) == 0 && top != -1)
          newexp[i++] = stack[top--];
        stack[++top] = exp[k];
      }
    }
  }
  while(top != -1 )
  {
    if(stack[top] != '('
      newexp[i++] = stack[top];
    top--;
  }
```

```
void print(char exp[])
{
  for(int i=0; exp[i] != '#'; i++)
    printf("%c", exp[i]);
}

int main()
{
  char exp[maxsize];
  char x; int i=0;
  printf("请输入正确的表达式：\n");

  while(x != '#')
  {
    x = getchar();
    exp[i++] = x;
  }
  getchar(); //用于接收回车键

  char newexp[maxsize];
  for(int i=0; i<maxsize; i++)
    newexp[i] = '#';
  trans_post_exp(exp, newexp);

  print(newexp);

  return 0;
}
```

3.9 试用类 C 写一个算法，对逆波兰式求值。

假设逆波兰式用 s 表示。

```
InitStack(S);
i = 0;
while(s[i] != '\0'){
        if(!In(s[i],OP))        //操作数，直接入栈
                Push(S,s[i]);
        else{
                Pop(S,b);
                Pop(S,a);
                Push(S,Operate(a,s[i],b));
        }
        i++;
}
```

3.10 假设以带头结点的单循环链表表示队列，只设一个尾指针指向队尾元素，不设头指针。试编写相应的队列初始化、入队和出队的算法。

初始化：

```
Status InitQueue(LinkQueue &Q){
        //带头结点，先分配头结点
        Q.rear = (QueuePtr) malloc(sizeof(QNode));
        if(!Q.rear) exit(OVERFLOW);
        Q.rear->next = Q.rear;
        return OK;
}
```

入队：

```
Status EnQueue(LinkQueue &Q, QElemType e){
        p = (QueuePtr) malloc(sizeof(QNode));
        if(!p) exit(OVERFLOW);
        p->data = e; p->next = Q.rear->next;          //插入队尾
        Q.rear->next = p;
        Q.rear = p;
        return OK;
}
```

出队：

```
Status DeQueue(LinkQueue &Q, QElemType &e){
        if(Q.rear->next == Q.rear) return ERROR;
        p = Q.rear->next->next;                 //队首结点
        e = p->data;
        Q.rear->next->next = p->next;       //删除队首结点
        free(p);
        return OK;
}
```

3.11 假设将循环队列定义为：以 rear 和 length 分别指示队尾元素和队列长度。试给出此循环队列的队满条件，并写出相应的入队和出队算法（在出队算法中要传递回队头元素的值）。

队满条件：

(Q.rear+1)%MAXQSIZE==(Q.rear-Q.length+MAXQSIZE)%MAXQSIZE; 或者 Q.length=MAXQSIZE-1

入队：

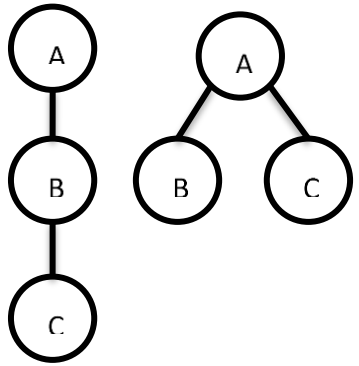
```
Status EnQueue(SqQueue &Q, QElemType e){
    if((Q.rear+1)%MAXQSIZE == (Q.rear-Q.length+MAXQSIZE)%MAXQSIZE) //队满
        return ERROR;
    Q.rear = (Q.rear+1)%MAXQSIZE;
    Q.length++;
    Q.base[Q.rear] = e;
    return OK;
}
```

出队：

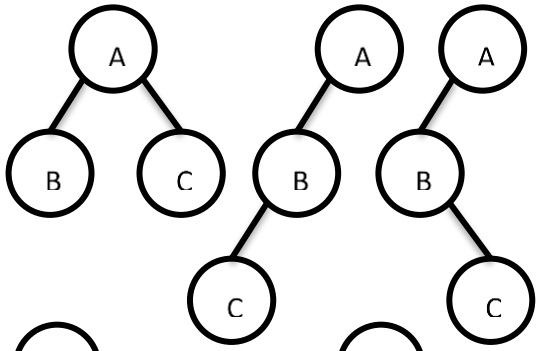
```
Status DeQueue(SqQueue &Q, QElemType &e){
    if(Q.length == 0) return ERROR;
    e = Q.base[(Q.rear-Q.length+MAXQSIZE)%MAXQSIZE];
    Q.length--;
    return OK;
```

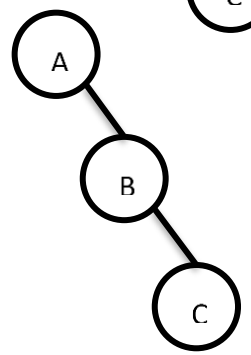
第六章

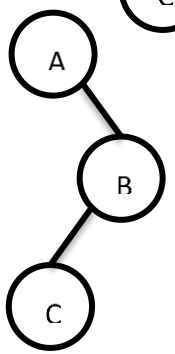
6.1试分别绘出具有 3 个结点的树和 3 个结点的二叉树的所有不同形态。



具有 3 结点的树







具有 3 结点的二叉树

6.2 设结点 X 是二叉树上一个度为 1 的结点， X 有几个子树？

2个子树（空树、本身）

6.3描述满足下列条件的二叉树形态：

(1) 先序遍历序列与中序遍历序列相同；不含左子树

(2) 后序遍历序列与中序遍历序列相同；不含右子树

(3) 先序遍历序列与后序遍历序列相同；既不含左子树，也不含右子树

6.4一个深度为 H 的满 k 叉树有如下性质：第 H 层上所有结点都是叶子结点，其余各层上每个结点都有 k 棵非空子树。如果从 1 开始按自上而下、自左向右的次序对全部结点编号，问：

(1) 各层的结点数目是多少？

(2) 编号为 i 的结点的父结点(若存在)的编号是多少？

(3) 编号为 i 的结点的第 j 个孩子(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？

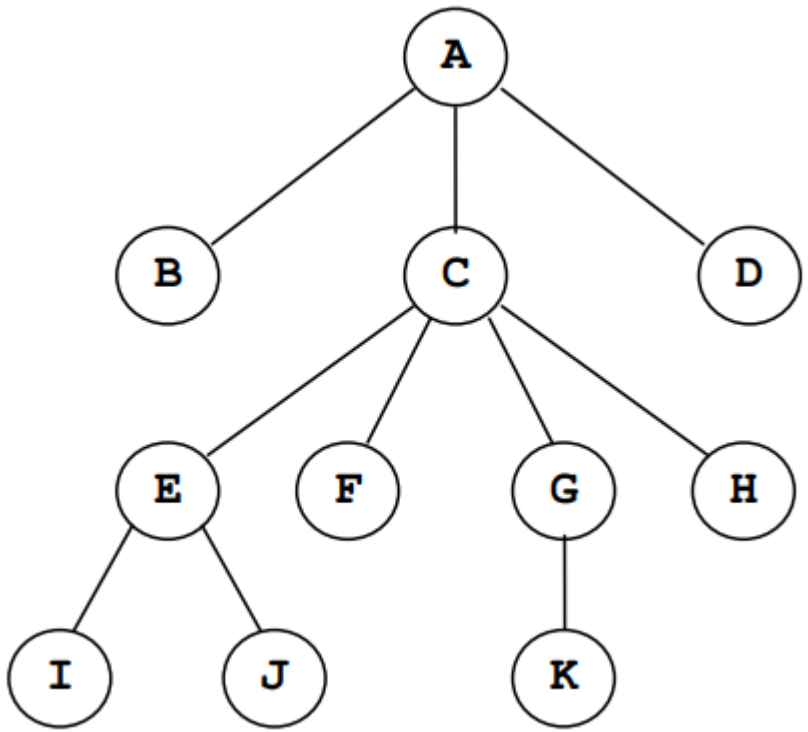
(1) k^{i-1}

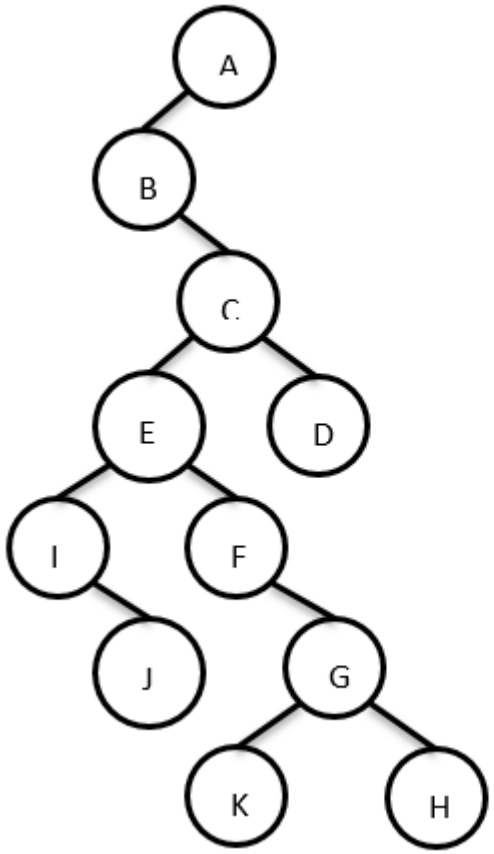
(2) $j = \left\lfloor \frac{i+k-2}{k} \right\rfloor$ 如果i是其双亲的最小的孩子,则i减去根结点的一个结点，应是k的整数倍，该整数即为所在的组数，每一组为一棵满k叉树，正好应为双亲结点的编号。如果i是其双亲的最大的孩子,则i+k-1为其最小的弟弟，再减去一个根结点，除以k，即为其双亲结点的编号

(3) 结点i的右孩子的编号为ki+1，左孩子的编号为ki+1-k+1=k(i-1)+2，第j个孩子的编号为k(i-1)+2+j-1=ki-k+j+1

(4) 当(i-1)%k != 0时，结点i有右兄弟，其右兄弟的编号为i+1。

6.8 已知一棵树如图所示，画出与该树对应的二叉树，并写出该树的先根遍历序列和后根遍历序列。

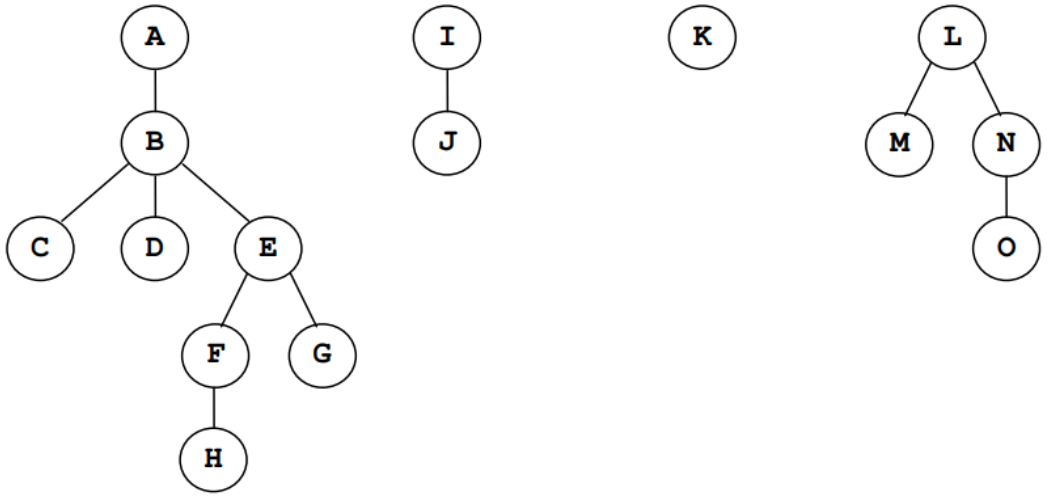


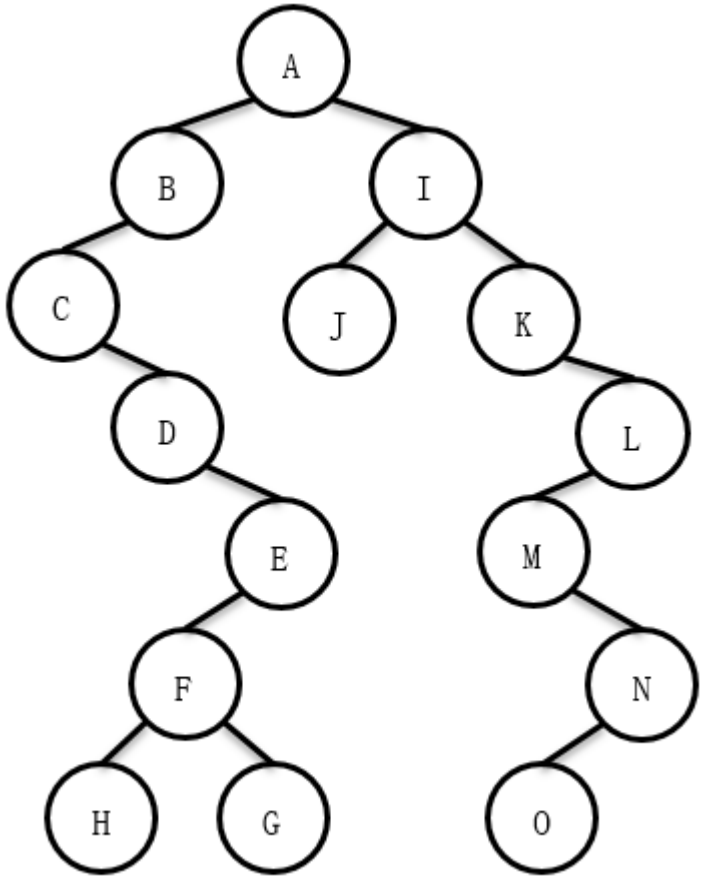


先根：ABCEIJFGKHD

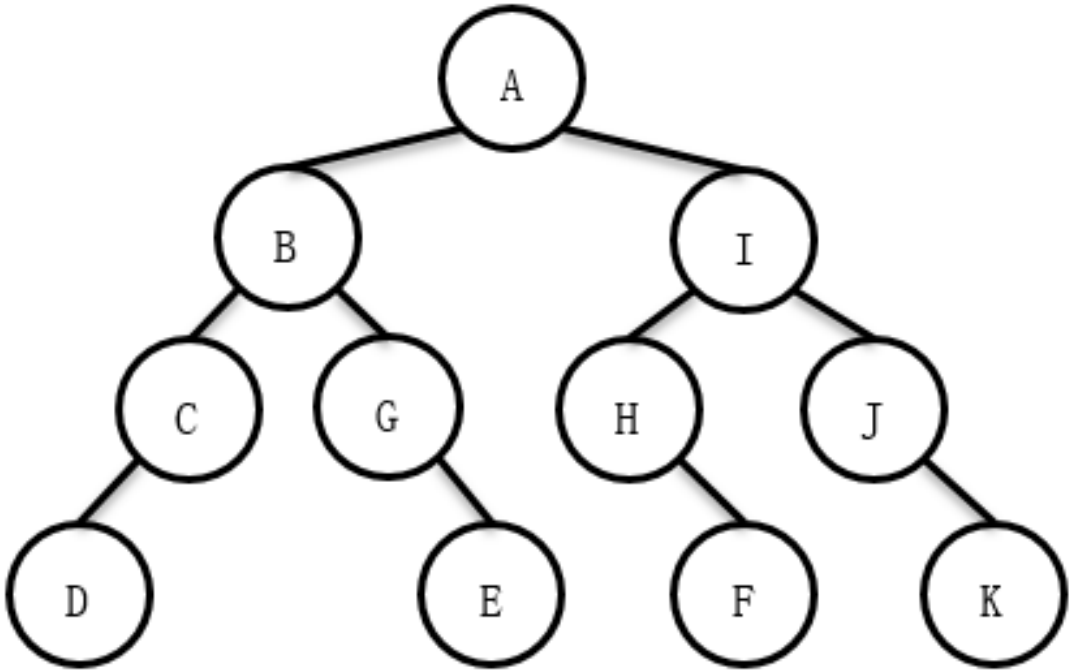
后根：BIJEFKGHCDA

6.9 将如图 6-2 所示的森林转化为对应的二叉树。

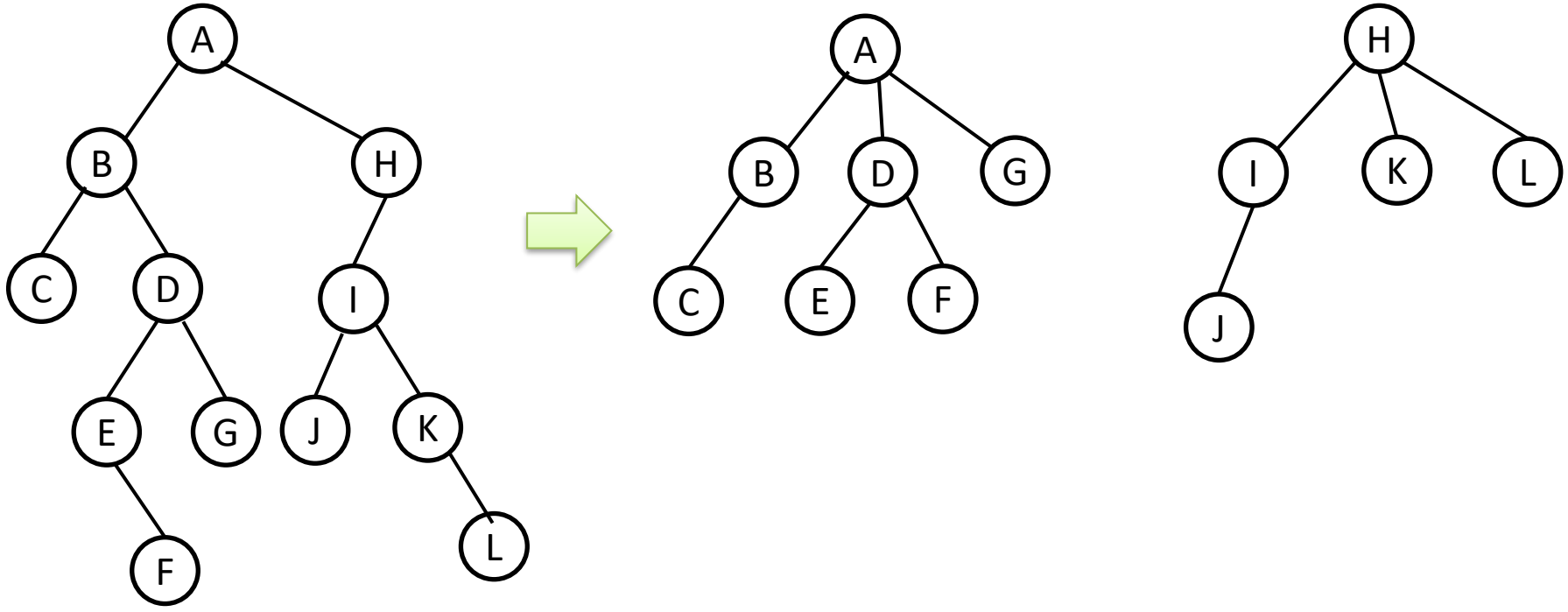




6.11已知某二叉树的中序序列为 DCBGEAHFIJK，后序序列为 DCEGBFHKJIA。请画出该二叉树。



6.13 已知森林F的先根遍历访问序列为ABCDEFGHIJKL，中根遍历访问序列为CBEFDGAJIKLH。请画出这个森林F。

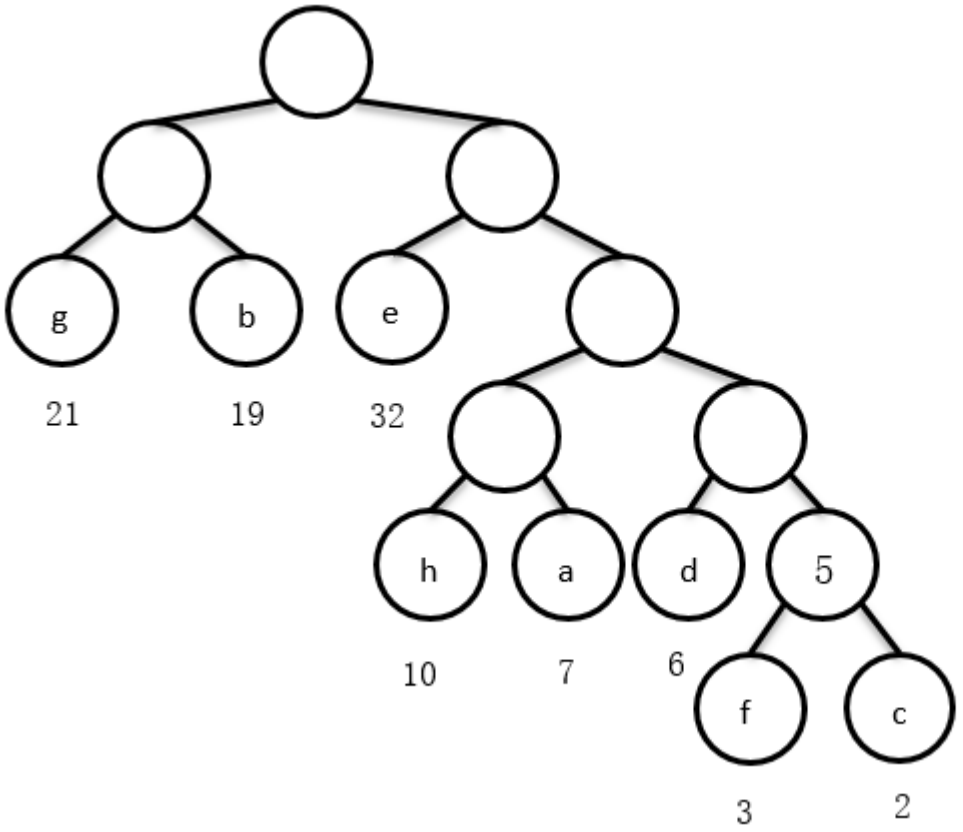


6.14假设某个电文由(a,b,c,d,e,f,g,h) 8 个字母组成，每个字母在电文中出现的次数分别为(7,19,2,6,32,3,21,10)，试解答下列问题：

(1) 画出出 Huffman 树；

(2) 写出每个字母的 Huffman 编码；

(3) 在对该电文进行最优二进制编码处理后，电文的二进制位数。



a:1101

b:01

c:11111

d:1110

e:10

f:11110

g:00

h:1100

电文二进制位数：

261=4*7+2*19+5*2+

4*6+2*32+5*3+2*21

+4*10

6.16 试编写算法，实现将二叉树所有结点的左右子树互换。

```
Status ExchangeBiTree(BiTree &T)
{
    BiTree p;
    if(T){
        p=T->lchild;
        T->lchild=T->rchild;
        T->rchild=p;
        ExchangeBiTree(T->lchild);
        ExchangeBiTree(T->rchild);
    }
    return OK;
}
```

6.12 写出计算树的深度的算法，树用孩子兄弟链表表示。

```
int GetDepth_CSTree(CSTree T)//求孩子兄弟链表表示的树 T 的深度
{
    if(!T) return 0; //空树
    else
    {
        for(maxd=0,p=T->firstchild;p;p=p->nextsib)
            if((d=GetDepth_CSTree(p))>maxd) maxd=d; //子树的最大深度
        return maxd+1;
    }
}//GetDepth_CSTree
```

6.12写出计算二叉树第 K 层结点数的算法。

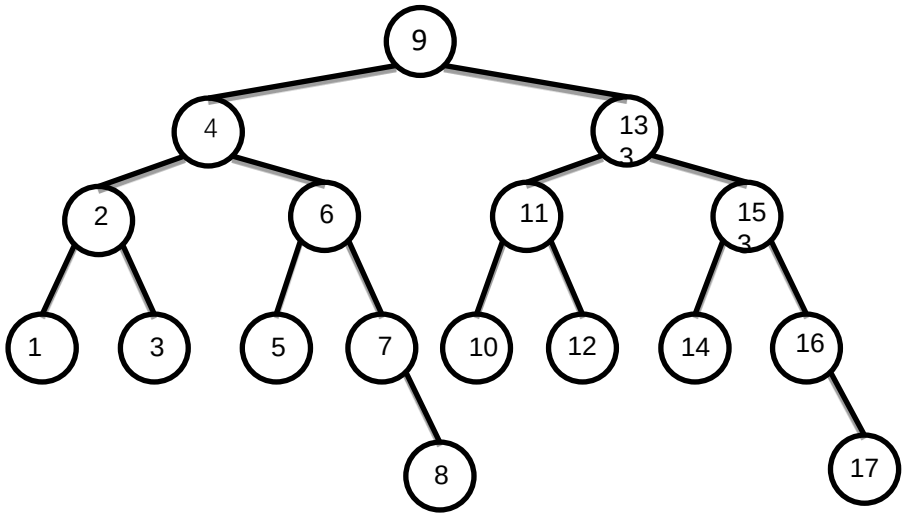
```
int GetNodeNum(Bitree T, int k)
{
    if (!T || k<1)
        return 0;
    if (k=1)
        return 1;
    return GetNodeNum(T->lchild, k-1)+ GetNodeNum(T->rchild, k-1);
}
```

第九章 查找

9.2 试分别写出在对有序线性表(a, b, c, d, e, f, g)中进行折半查找，查值等于 e、f 和 g 的元素时，先后与哪些元素进行了比较。

查找e: d, f, e
查找f: d, f
查找g: d, f, g

9.3 画出对长度为 17 的有序表进行折半查找的判定树，并分别求其等概率时查找成功和查找失败的 ASL。



折半查找的 ASL 利用二叉判定树计算

$$ASL_{成功} = \frac{每个结点的比较次数之和}{结点数}$$

$$ASL_{失败} = \frac{空指针处比较次数之和}{空指针数}$$

$$ASL_{平均} = log_2 n(该树的高度)$$

查找成功时：ASL=(1+2*2+4*3+8*4+5*2)/17=3.47

查找失败时：ASL=(2*4*6+2*1*4+2*2*5)/18=4.22

9.4 已知如下所示长度为 12 的表：

(Jan, Feb, Mar, Apr, May, Jun, July, Aug, Sep, Oct, Nov, Dec)

表中，每个元素的查找概率分别为：

(0.1, 0.25, 0.05, 0.13, 0.01, 0.06, 0.11, 0.07, 0.02, 0.03, 0.1, 0.07)

(1)若对该表进行顺序查找，求查找成功的平均查找长度；

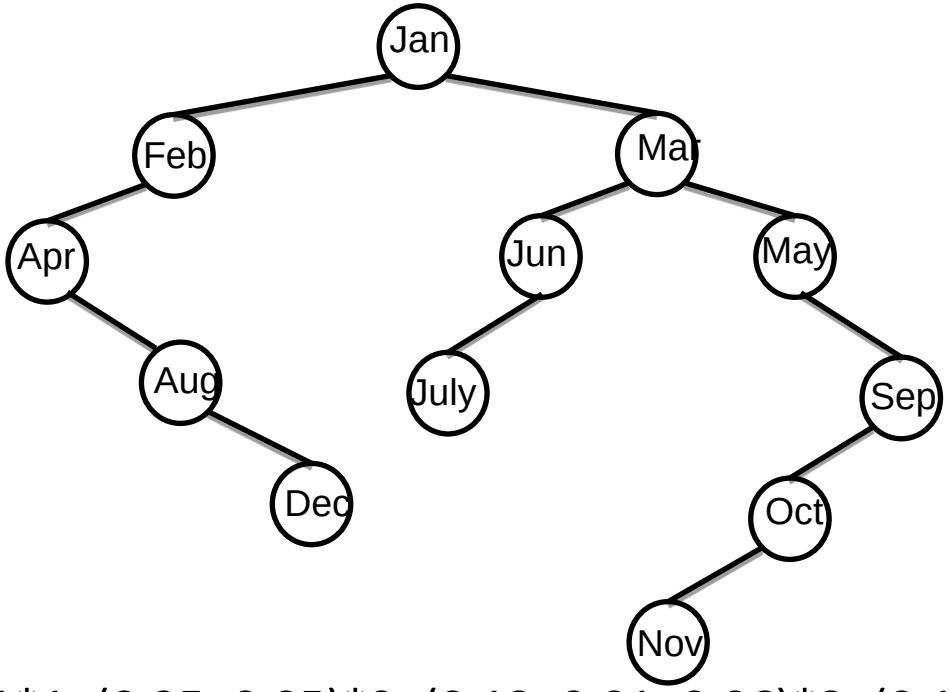
(2)画出从初态为空开始，依次插入结点，生成的二叉排序树；

(3)计算该二叉排序树查找成功的平均查找长度；

(4)将二叉排序树中的结点 Mar 删除，画出经过删除处理后的二叉排序树。

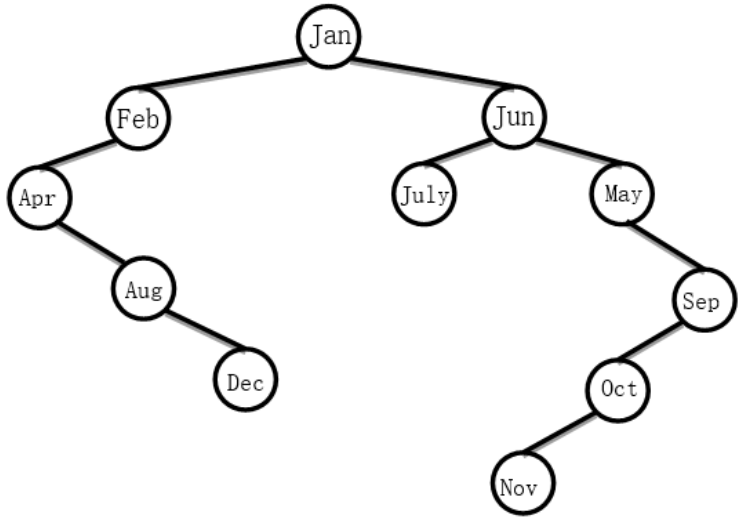
(1)0.1*12+0.25*11+0.05*10+0.13*9+0.01*8+0.06*7+0.11*6+0.07*5+0.02*4+0.03*3+0.1*2+0.07*1=7.57
或0.1*1+0.25*2+0.05*3+0.13*4+0.01*5+0.06*6+0.11*7+0.07*8+0.02*9+0.03*10+0.1*11+0.07*12=5.43

(2)

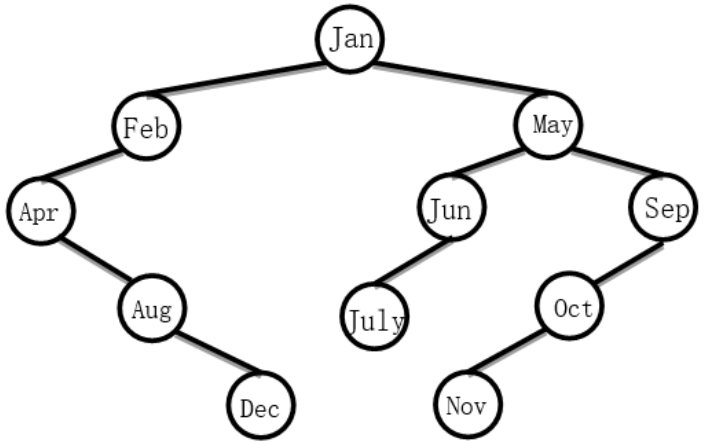


(3) 0.1*1+(0.25+0.05)*2+(0.13+0.01+0.06)*3+(0.11+0.07+0.02)*4+(0.03+0.1)*5+0.07*6=3.17

(4)



或



9.5 已知关键字序列{10, 25, 33, 19, 06, 49, 37, 76, 60}，哈希地址空间为 0～10，哈希函数为 H (Key)=Key % 11，求：

（1） 用开放定址线性探测法处理冲突，构造哈希表 HT1，分别计算在等概率情况下 HT1 查找成功和查找失败的 ASL；

（2） 用开放定址二次探测法处理冲突，构造哈希表 HT2，计算在等概率情况下 HT2 查找成功的 ASL；

（3） 用拉链法解决冲突，构造哈希表 HT3，计算 HT3 在等概率情况查找成功的 ASL。

(1) HT1

0	1	2	3	4	5	6	7	8	9	10
33	76		25	37	49	6	60	19		10
76					60	60				76

查找成功：ASL=(7*1+2*3)/9=1.44

查找失败：ASL=(3+2+1+7+6+5+4+3+2+1+4)/11=3.45

9.5 已知关键字序列{10, 25, 33, 19, 06, 49, 37, 76, 60}，哈希地址空间为 0～10，哈希函数为 H (Key)=Key % 11，求：

（1） 用开放定址线性探测法处理冲突，构造哈希表 HT1，分别计算在等概率情况下 HT1 查找成功和查找失败的 ASL；

（2） 用开放定址二次探测法处理冲突，构造哈希表 HT2，计算在等概率情况下 HT2 查找成功的 ASL；

（3） 用拉链法解决冲突，构造哈希表 HT3，计算 HT3 在等概率情况查找成功的 ASL。

(2) HT2

0	1	2	3	4	5	6	7	8	9	10
33	60		25	37	49	6		19	76	10
76				60	60	60			60	76

查找成功：ASL=(7*1+3+5)/9=1.67

(3) HT3

0	1	2	3	4	5	6	7	8	9	10
33			25	37	49	6		19		10
					60					76

查找成功：ASL=(7*1+2*2)/9=1.22

3. 散列表中地址链接法的 ASL

ASL_成功 = 每个元素被访问（查找）的次数

ASL_失败 = 每个地址被访问（查找）的次数

9.7 写出判别一棵二叉树是否为二叉排序树的算法，设二叉排序树中不存在关键字值相同的结点。

```
bool IsSearchTree(Bitree *T,Elemtype &e;)
{//递归遍历二叉树是否为二叉排序树,e 初始值为最小值
    if(!T)return TRUE;
    retl=IsSearchTree(T->lchild,e);
    if(T->data<e)return FALSE;              //当前元素小于中序序列的前一个元素
    e=T->data;
    retr=IsSearchTree(T->rchild,e);
    return retl&&retr;
}
```

Thank you!