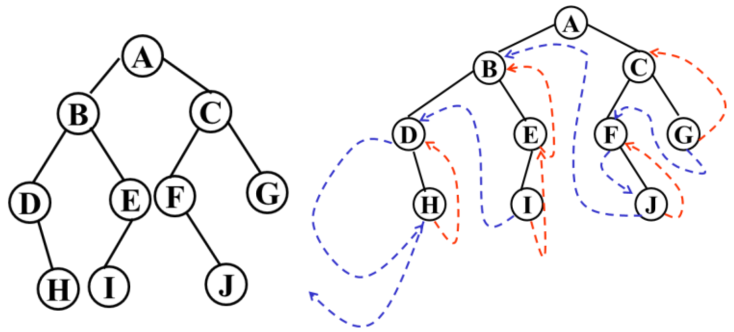
一、基础题

1. 请对下图所示二叉树进行后序线索化，为每个空指针建立相应的前驱或后继线索。



解：后序线索化树如右图所示（头结点在绘制后序线索链表时才需要给出）。其中，蓝色虚线表示后序遍历下，各结点的前驱线索；红色虚线表示后继线索。

2. 将下列二叉链表改为先序线索链表（不画出树的形态），并写出为结点 D、F、K 线索化的过程。

序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Info	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Ltag														
Lchild	2	4	0	0	8	0	10	0	0	0	14	0	0	0
Rtag														
Rchild	3	5	6	7	0	9	11	0	12	13	0	0	0	0

解：**先序线索链表如下：**

序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Info	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Ltag	0	0	1	1	0	1	0	1	1	1	0	1	1	1
Lchild	2	4	8	2	8	3	10	5	6	7	14	9	10	11
Rtag	0	0	0	0	1	0	0	1	0	0	1	1	1	1
Rchild	3	5	6	7	8	9	11	3	12	13	14	0	11	5

D、F 和 K 的线索化为：

D：结点 D 的左指针应该指向前驱。结点 D 是结点 B 的左孩子，先序遍历结点 B 后就访问结点 D。所以结点 D 的前驱是结点 B。

F：结点 F 的左指针应该指向前驱。结点 F 是结点 C 的右孩子，应该先序遍历结点 C 的左子树后才访问结点 F。而结点 C 的左子树为空，因此，结点 F 的前驱即为结点 C。

K：结点 K 的右指针应该指向后继。访问了结点 K 后，应该访问其左子树树根结点，即结点 N。

所有结点的线索化过程为：

（1） 首先标识左右指针域，置 0 表示指向孩子，置 1 表示指向前驱或后继。

（2） 结点 C 的左指针应该指向前驱。结点 C 是结点 A 的右孩子，应该先序遍历结点 A 的左子树后才访问结点 C。所以结点 C 的前驱是结点 A 的左子树的最后遍历结点，即结点 H。

（3） 结点 D 的左指针应该指向前驱。结点 D 是结点 B 的左孩子，先序遍历结点 B 后就访问结点 D。所以结点 D 的前驱是结点 B。

（4） 结点 E 的右指针应该指向后继。访问了结点 E 后，应该访问其左子树树根结点，即

结点 H。

（5） 结点 F 的左指针应该指向前驱。结点 F 是结点 C 的右孩子，应该先序遍历结点 C 的左子树后才访问结点 F。而结点 C 的左子树为空，因此，结点 F 的前驱即为结点 C。

（6） 结点 H 的左指针应该指向前驱。结点 H 是结点 E 的左孩子，先序遍历结点 E 后就访问结点 H。所以结点 H 的前驱是结点 E。

（7） 结点 H 的右指针应该指向后继。

a) 访问了结点 H 之后，应该先序遍历其左右子树。然而结点 H 是叶结点，即以 H 为根的子树的先序遍历已经完成。后续待访问结点应结合 E 与双亲的关系确定。

b) 因为结点 H 是结点 E 的左孩子，所以后续应该访问结点 E 的右子树的树根，而结点 E 没有右子树，再一次，需要结合结点 E 与其双亲的关系判定后续待访问结点。

c) 结点 E 是结点 B 的右子树树根，结束了对 B 的右子树的访问，后续待访问结点需要结合 B 与其双亲的关系判定。

d) 结点 B 是结点 A 的左子树树根，先序遍历了 A 的左子树后，接下来应该访问 A 的右子树树根，即结点 C。所以结点 H 的后继是结点 C。

（8） 结点 I 的左指针应该指向前驱。结点 I 是结点 F 的右孩子，应该先序遍历结点 F 的左子树后才访问结点 I。而结点 F 的左子树为空，因此，结点 I 的前驱即为结点 F。

（9） 结点 J 的左指针应该指向前驱。结点 J 是结点 G 的左孩子，应该先序遍历结点 G 后就访问结点 J。结点 J 的前驱即为结点 G。

（10） 结点 K 的右指针应该指向后继。访问了结点 K 后，应该访问其左子树树根结点，即结点 N。

（11） 结点 L 的左指针应该指向前驱。结点 L 是结点 I 的右孩子，应该先序遍历结点 I 的左子树才访问结点 L。而结点 I 的左子树为空，因此，结点 L 的前驱即为结点 I。

（12） 结点 L 的右指针应该指向后继。

a) 访问了结点 L 之后，应该先序遍历其左右子树。然而结点 L 是叶结点，即以 L 为根的子树的先序遍历已经完成。后续待访问结点应结合 L 与双亲的关系确定。

b) 因为结点 L 是结点 I 的右孩子，完成了对 I 的右子树的遍历后，结合结点 I 与其双亲的关系判定后续待访问结点。

c) 结点 I 是结点 F 的右孩子，完成了对 F 的右子树的遍历后，结合结点 F 与其双亲的关系判定后续待访问结点。

d) 结点 F 是结点 C 的右孩子，完成了对 C 的右子树的遍历后，结合结点 C 与其双亲的关系判定后续待访问结点。

e) 结点 C 是结点 A 的右孩子，完成了对 A 的右子树的遍历后，整棵树的遍历结束。结点 L 为遍历的最后结点，其后继为空。

（13） 结点 M 的左指针应该指向前驱。结点 M 是结点 J 的右孩子，应该先序遍历结点 J 的左子树才访问结点 M。而结点 J 的左子树为空，因此，结点 M 的前驱即为结点 J。

（14） 结点 M 的右指针应该指向后继。

a) 访问了结点 M 之后，应该先序遍历其左右子树。然而结点 M 是叶结点，即以 M 为根的子树的先序遍历已经完成。后续待访问结点应结合 M 与双亲的关系确定。

b) 因为结点 M 是结点 J 的右孩子，完成了对 J 的右子树的遍历后，结合结点 J 与其双亲的关系判定后续待访问结点。

c) 结点 J 是结点 G 的左孩子，遍历了 G 的左子树后，应该先序遍历 G 的右子树树根，即结点 K。

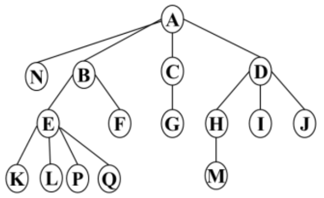
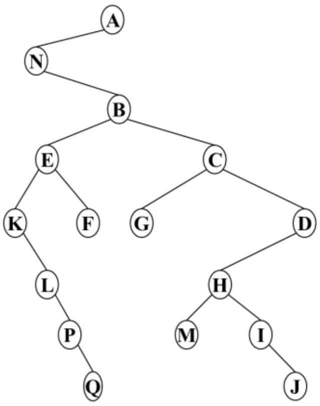
（15） 结点 N 的左指针应该指向前驱。结点 N 是结点 K 的左孩子，先序遍历结点 K 后即访问结点 N，因此，结点 N 的前驱即为结点 K。

（16） 结点 N 的右指针应该指向后继。

a) 访问了结点 N 之后，应该先序遍历其左右子树。然而结点 N 是叶结点，即以 N 为根的子树的先序遍历已经完成。后续待访问结点应结合 N 与双亲的关系确定。
b) 因为结点 N 是结点 K 的左孩子，完成了对 K 的左子树的遍历后，应遍历 K 的右子树。而 K 的右子树为空，因此，需结合结点 K 与其双亲的关系判定后续访问结点。
c) 结点 K 是结点 G 的右孩子，遍历了 G 的右子树后，需结合结点 G 与其双亲的关系判定后续访问结点。
d) 结点 G 是结点 D 的右孩子，遍历了 D 的右子树后，需结合 D 与其双亲的关系判定后续访问结点。
e) 结点 D 是结点 B 的左孩子，遍历了 B 的左子树后，应访问 B 的右子树树根，即结点 E。

可绘制二叉树验证，先序遍历为：ABDGJMKNEHCFIL。

3. 请给出下图树的后根遍历序列，并画出其对应的二叉树。

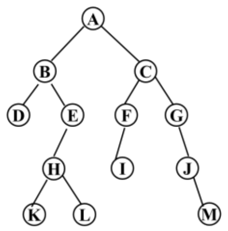
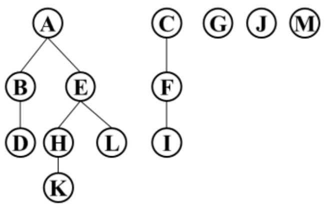



解：对应二叉树如右图所示。

后根遍历序列为：NKLPQEFBGCMHIJDA

注意，树的后根遍历规则为：依次后根遍历各子树，再遍历根。以根为 B 的子树为例，应该先后根遍历以 E 为根的子树，再后根遍历 F，即（E 子树的后根序列）EF。同样规则作用于 E 的子树，得到 KLPQEF。

4. 请画出下面二叉树所对应的森林，并给出其中序遍历序列。

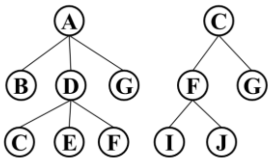



解：对应森林如右图所示。二叉树的中序序列为：DBKHLEAIFCGJM；森林的中序序列为：DBKHLEAIFCGJM。

由二叉树与森林互换规则可知，二叉树及其对应森林的中序遍历序列是相同。注意，森林的中序遍历规则是：中序遍历森林的第 1 棵树的子树森林（对应二叉树的左子树），遍历第 1 棵树的树根（对应二叉树树根），再中序遍历第 1 棵树以外树构成的森林（对应二叉树的右子树）。用以 A 为根的子树为例，序列为：（A 的子树森林的中序遍历序列）A。递归展开为：（（B 的子树森林的中序遍历序列）B（根为 E 的森林的中序遍历序列））A

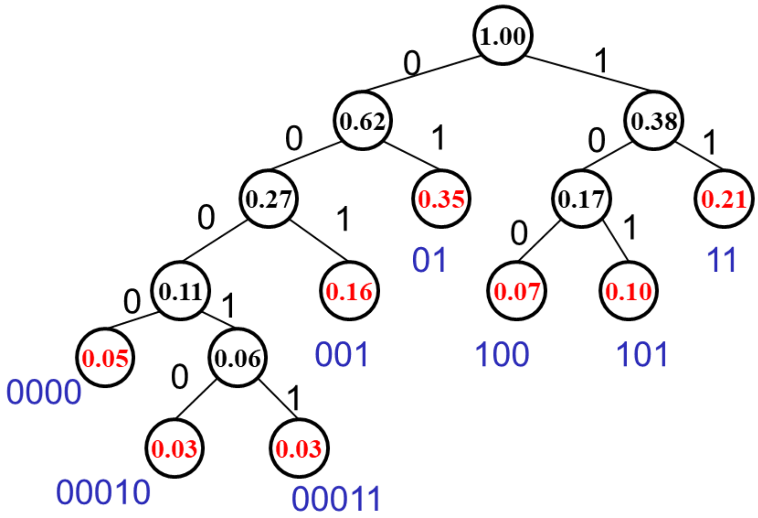
（（D）B（（根为 H 的森林的中序遍历序列）（根为 L 的森林的中序遍历序列）E））A
（（D）B（（KH）（L）E））A
所以，这部份序列为 DBKHLEA

5. 请画出和下列已知序列对应的森林 F。
森林的先序次序遍历序列为：ABDCEFGHIJKL
森林的中序次序遍历序列为：BCEFDGAJKILH
解：



根据森林的遍历规则，逐层递归确定各棵树。
（1） 由先序得知，第 1 棵树的树根为 A。由中序可知，A 左侧结点序列，即 BCEFDG 为 A 的子树森林的中序遍历序列；A 的右侧结点序列，JKILH，为第 1 棵以外的剩余树构成的森林的中序序列。
（2） 分析第 1 棵树的子树森林的遍历序列。由先序可知，子树森林中，第 1 棵树的树根为 B；由中序可知，该树没有子树，而其余子树森林的中序遍历序列为 CEFDG。
（3） 依此类推，逐层展开，即可得森林结构。

6. 假设用于通信的电文仅由 8 个字母组成，字母在电文中的出现频率分别为 0.05，0.16，0.03，0.07，0.35，0.03，0.21，0.10。试为这 8 个字母设计哈夫曼（赫夫曼）编码及定长编码，比较两个方案的优缺点。
解：赫夫曼编码如下图所示



两种编码码字和平均码长度为（对应赫夫曼树即带权路径长度）

方案	平 均码长	0.03	0.03	0.05	0.07	0.10	0.16	0.21	0.35
赫 夫曼	2.61	00010	00011	0000	100	101	001	01	11

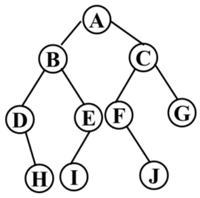
定长	3	000	001	010	011	100	101	110	111

WPL=0.03*5+0.03*5+0.05*4+0.07*3+0.1*3+0.16*3+0.21*2+0.35*2=2.61

固定编码方案编解码简单，但效率低，平均每个字符要用 3 比特表示；赫夫曼编码方案则有较高的效率，平均每个字符要用 2.61 比特表示。显然，消息所包含的字符数越多，效率优势越明显，例如若消息包含 10000 个字符，那么使用赫夫曼编码要少用 3900 比特。

二、算法设计题

1. 已知在二叉树中，*root 为根结点，*p 和*q 为二叉树中两个结点，试编写算法，求它们的最大共同路径。如下图中，若 p 指向 H，q 指向 E，则它们最大共同路径为(A,B)。算法只需给出共同路径上的结点标识即可，在本例中，输出 A、B 即可。



解：本题实质上还是考查遍历。

最简单的想法是通过遍历记录根 root 到节点 p 的路径 path1，本例中为“ABDH”；再通过遍历记录根 root 到节点 q 的路径 path2，本例中为“ABE”；最后找 path1 和 path2 的最长公共路径，也就是说从根开始到首个不相同结点出现前的路径，或者说最长相同子串，本例为“AB”。

现在的问题是如何用尽可能少的代价找到路径和相同子串。

用递归，我们可以很简单地解决这个问题。如果一个结点的左右子树既包括结点 p 又包括结点 q，那么该结点就是 p 和 q 的共同祖先，则该结点必然在路径中。我们可以使用后序遍历来完成这一任务：检查左子树是否包含 p 或 q；检查右子树是否包含 p 或 q；判断当前结点是否为公共祖先，若是，保存入路径。

这个思路需要解决两个问题。

第一， 后序遍历意味着，我们会先找到离 p 和 q 最近的公共祖先，最后找到根。而输出顺序正好相反，先输出根，最后输出最近公共祖先。使用栈来缓存公共祖先序列就可以解决了。

第二， 我们使用递归时，函数返回值只有 1 个，如何用 1 个返回值来表示四种状态？即：不包含 p 且不包含 q，包含 p 不包含 q，包含 q 不包含 p，既包含 p 又包含 q。这也不难，简单编码即可。约定这种情况的返回值依次是 0，2，4，6。那么我们就能根据左右子树及当前结点的状况确定返回值了。

参考代码见 `findMaxComnPath`。本题还有很多种思路，例如通过两次遍历分别找到 p，q 的路径（参考教科书赫夫曼编码算法记录路径），又例如在遍历过程中形成辅助数据结构——树的双亲表示（这样找路劲就非常容易了）……鼓励大家尝试各种不同有趣的算法。

2. 编写算法，不重复地输出以孩子兄弟链表存储的树 T 中所有的边。输出的形式为 (k_1,k_2),…,(k_i,k_j),…,其中 k_i 和 k_j 为树结点中的标识。

解：本题考查对孩子兄弟链表的理解。

但就存储结构而言，孩子兄弟链表表示法，就是用二叉链表来表示树。要把一棵二叉树的所有边不重复的输出，利用任意一种二叉树遍历方法就可以了。那么本题还有何难度呢？这里的误区在于，二叉树表示时，双亲和右子树根形成的边，并不是树中的边。在树的孩子兄弟链表表示法中，右子树根结点是其双亲结点的兄弟。所以问题就转化为如何找到正确双

亲和孩子关系。

在二叉树的遍历算法中，我们很容易找到树根（D）、左子树树根（D->Lc）和右子树树根（D->Rc）。对于树来说，D 和 D->Lc 是边，D 的双亲和 D->Rc 是边，D 和 D->Lc->Rc 是边。因此，两种应对方案。第一种，输出当前结点所有边，第二种，保存当前结点双亲信息。

根据第一种策略，我们只要沿着当前结点左子树的右分支一直走到尽头，就可以把当前结点的所有边都输出了。参见代码 PrintAllEdges。

根据第二种策略，我们在递归中增加一个参数，传递结点双亲信息 Parents 即可。初始时，当前结点为树根，双亲结点为空。在递归过程中，左子树树根结点 D->Lc 的双亲结点即为当前结点 D，右子树树根结点 D->Rc 的双亲结点为当前结点的双亲结点 Parents。为避免重复输出，每次只输出当前结点双亲结点 Parents 和当前结点 D 之间的边。参见代码 PrintAllEdges2。

3. 编写算法，对以孩子链表表示的树计算树的深度。

解：

本题考查对孩子链表的掌握和树的遍历。利用树的遍历，从叶子结点开始，向祖先方向，每次经过双亲结点深度就加 1。也就是说，以双亲结点为根的树的深度，是其所有子树中最深的深度加 1。可以采用后根遍历来实现。

本题还要解决的问题是以孩子链表表示的树的创建。可以考虑较迂回的方法，先以二叉链表为存储结构创建树，再遍历二叉链表将其转换为孩子链表。这个思路的优势在于，我们已经积累了二叉树的创建，以及孩子链表表示法的树的遍历的算法。例如，我们利用本次作业题目 2 的算法，稍加改造，就可以将二叉链表转换为孩子链表。

还可以使用直接方法。读入树的所有“边”的信息，据此构造孩子链表。具体操作为：

- 依次读入树的各条边；
- 每条“边”包含一个双亲结点 p 和一个孩子结点 c；
- 检查孩子链表中，结点 p、c 对应的表结点是否已经存在，若不存在则增加；
- 将结点 c 加入结点 p 的孩子链中。

根据孩子链表计算树的深度的算法参考实现为 CTDepth，根据“边”信息创建树的算法参见 CreateCT。