装订线内

不要答题

注意：姓名、学号和任课教师必须要写

北京大学信息科学技术学院考试试卷

考试科目：数据结构与算法 A　　**姓名：**_____　　**学号：**_____

考试时间：2017 年 11 月 15 日　　**任课教师：**_____

题号	一	二	三	四	五	总分
分数						
阅卷人						

北京大学考场纪律

1、考生进入考场后，按照监考老师安排隔位就座，将学生证放在桌面上。无学生证者不能参加考试；迟到超过 15 分钟不得入场。在考试开始 30 分钟后方可交卷出场。

2、除必要的文具和主考教师允许的工具书、参考书、计算器以外，其它所有物品（包括空白纸张、手机等）不得带入座位，已经带入考场的必须放在监考人员指定的位置，并关闭手机等一切电子设备。

3、考试使用的试题、答卷、草稿纸由监考人员统一发放，考试结束时收回，一律不准带出考场。若有试题印制问题请向监考教师提出，不得向其他考生询问。提前答完试卷，应举手示意请监考人员收卷后方可离开；交卷后不得在考场内逗留或在附近高声交谈。未交卷擅自离开考场，不得重新进入考场答卷。考试结束监考人员宣布收卷时，考生立即停止答卷，在座位上等待监考人员收卷清点后，方可离场。

4、考生要严格遵守考场规则，在规定时间内独立完成答卷。不准旁窥、交头接耳、打暗号，不准携带与考试内容相关的材料参加考试，不准抄袭或者有意让他人抄袭答题内容，不准接传答案或者试卷等。凡有严重违纪或作弊者，一经发现，当场取消其考试资格，并根据《北京大学本科考试工作与学习纪律管理规定》及其他相关规定严肃处理。

5、考生须确认自己填写的个人信息真实、准确，并承担信息填写错误带来的一切责任与后果。

学校倡议所有考生以北京大学学生的荣誉与诚信答卷，共同维护北京大学的学术声誉。

注意事项：

1. 全部题目都在空白答题纸上解答。
2. 试卷对算法设计都有质量要求，请尽量按照试题中的要求来写算法。否则将酌情扣分。
3. 请申明所写算法的基本思想，并在算法段加以恰当的注释。

以下为试题和答题纸，共 5 页，请把答案写在答题纸。

得分

一、 选择与填空（每空 2 分，共 24 分）

评判标准：对应的回答必须完全正确！

1. 下面函数的时间复杂度是（ C ）

```
void recursive(int n, int m, int k){
    if (n <= 0)
        printf("%d, %d\n", m, k);
    else {
        recursive(n-1, m+1, k);
        recursive(n-1, m, k+1);
    }
}
```

A. O(n*m*k)　　B. O(n^2*m^2)

C. O(2^n)　　D. O(n!)

T(n)=2*T(n-1)　T(0)=1 推出 T(n)=2^n

2. 完成在双循环链表结点 p 之后插入 s 的操作为(ACD):

A. p->next->prev=s; s->prev=p; s->next=p->next; p->next=s;

B. p->next->prev=s; p->next=s; s->prev=p; s->next=p->next;

s->next=p->next=s，错误

C. s->next=p->next; s->prev=p; p->next->prev=s; p->next=s;

D. s->prev=p; s->next=p->next; s->prev->next=s; s->next->prev=s;

3. 设栈 S 和队列 Q 初始状态为空，元素 e1，e2，e3，e4，e5，e6 依次通过栈 S，一个元素出栈后即进队列 Q，若 6 个元素出队序列是 e2，e4，e3，e6，e5，e1，则栈 S 的容量至少是 (B)

A.2;　B.3;　C.4;　D.6

输入序列中 i<j<k，若栈的输出序列中 k 出现在 i 和 j 之前，则输出顺序必然是 k, j, i，因为在 k 出栈前 i 和 j 必须被压在栈中。若输入序列中 i<j<k<p，输出序列中 p 出现在 i,j,k 之前，则 p 出栈前 i,j,k 必须被压在栈中。在输出序列中寻找这样的模式（即下降子序列，偏序关系由输入序列中的先后位置决定）即可，比如(e2 e1), (e3 e1), (e4, e3, e1), (e5 e1), (e6,e5,e1)。最长的模式的长度是 3，所以栈中最多存有 3 个元素，容量至少是 3。

反之，如果栈中最多存有 t 个元素，那么这 t 个元素一定是从栈顶向栈底递减的（压栈有明确的先后顺序），并且这 t 个元素弹出后，一定会在输出序列中对应着一个长度为 t 的下降子序列。

寻找最长下降子序列：O(n^2)时间复杂度，O(n)空间复杂度

4. 设循环队列的容量为 40 （序号从 0 到 39），现经过一系列的入队和出队运算后： 1）

front=12，rear=19；2）front=19，rear=12；在这两种情况下，循环队列中的元素个数分别为 __7__和__33__。

牺牲一个存储空间，rear 位置不存数据。初始状态 front=0, rear=0，循环链表为空。
满的判断条件应为：(rear+1)%LENGTH == front 。
空的判断条件为 rear == front。
front=12, rear=19 时，front 到 rear 是有效元素，共有 19-12=7 个。
front=19, rear=12 时，front 到 rear 是有效元素，此时跨过了数组最后一位，也即从 rear 到 front-1 是无效元素，所以元素个数是 40-(19-12)=33。

5. 一个 n 位的字符串共有__(n*(n+1)+2)/2__个子串。

包括空串，长度为 0 的子串有 1 个，长度为 1 的子串有 n 个，长度为 2 的子串有 n-1 个，长度为 3 的子串有 n-2 个……长度为 n-1 的子串有 2 个，长度为 n 的子串有 1 个。总和就是(n*(n+1)+2)/2。

6. 已知二叉树有 n 个结点 (n > 0)，则度为 2 的结点最多有__（n-1)/2__个。(需要下取整)

二叉树中有 n0=n2+1, n=n0+n1+n2=n1+1+2*n2, 要使度数为 2 的节点最多，也即 n2 最大，也即使得 n1 最小，n1 最小是 0。但同时需要保证 n-n1-1 为偶数，当 n 为偶数时 n1 最小只能取 1，所以 n2 最大是(n-1)/2 的下取整。（在完全二叉树中可以取到这个最大值）

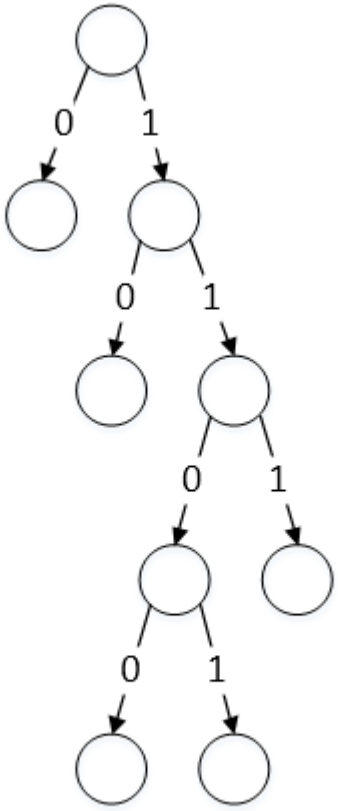
7. 用数组存储一个有 50 个元素的最小值堆。已知堆中没有重复元素。给出最大元素的下标可能的取值范围为 __25__ 到 __49__。

最小堆是完全二叉树且子节点不小于父节点，已知堆中没有重复元素，所以父节点一定小于子节点，即任何内部节点都不可能是最大值。最大值只可能出现在叶节点上，堆用数组存是分层存放的，要求最大元素的下标的可能的取值范围，也就是求叶节点的下标的范围。

最后一个叶节点必然是存在第 49 位（下标从 0 开始），所有只要求出叶节点的个数就可以了。50 个元素的堆，从上到下每层的元素个数分别是 1,2,4,8,16,19。倒数第二层有 19/2（上取整）个内部节点，所以叶节点的个数是 19+16-10=25 个，下标范围是 25~49。

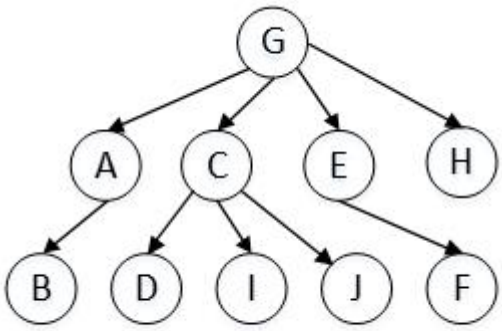
8. 假设用于通信的电文由 5 个字符组成。已知其中一个字符的 Huffman 编码为 1101，则该 Huffman 编码树的平均编码长度为__2.8__。

如下图，总的外部路径长度（不带权）是 1+2+4+4+3=14，所以平均编码长度是 14/5=2.8



9. 假设先根次序遍历某棵树的结点次序为 GABCDIJEFH，后根次序遍历该树的结点次序为 BADIJCFEHG，那么 I 结点的父结点是___C___。

显然根节点是 G，考虑 G 的最右边的子树 T，它的根必然是 H。在先根次序中找到 H，它的子树为空，所以 G 的最右边的子树就是 H。接下来查找 H 的左兄弟 T1，根为 E，其子树为 F；接下来查找 E 的左兄弟 T2，根为 C……依次类推，直到还原整棵树，如下图所示，I 节点的父节点是 C。



10. 一个拥有 144 个结点的完全三叉树，现在从倒数第二层上任取一个结点，该结点为叶结点的概率是_____73/81_____。（请用化简后的分数表示）。

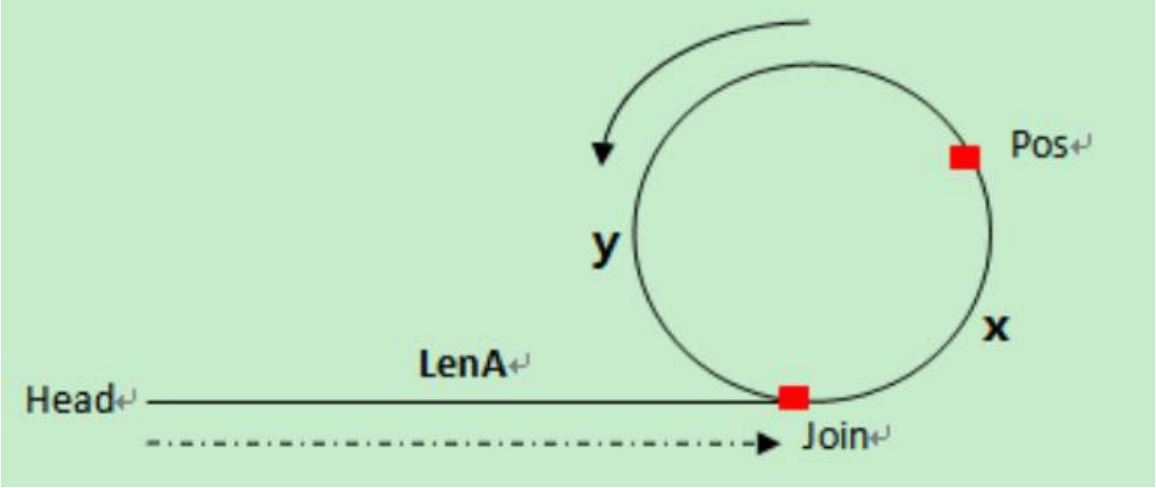
完全三叉树从上到下各层的节点个数是 1,3,9,27,81,23，倒数第二层有 23/3（上取整后为 8）个内部节点，即有 81-8=73 个叶节点。

得分

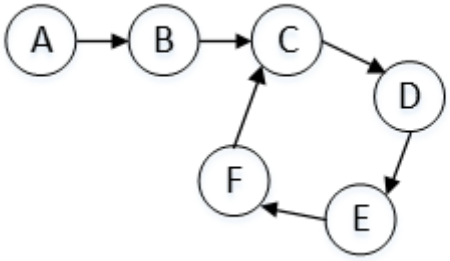
二、 辨析与简答(共 24 分)

1. （6 分）现有一个由单链表和循环单链表构成的特殊链表，单链表的头元素是 head，末尾元素为 tail，head->…->tail 即是一条普通的链表，同时 tail 元素还是另一条循环单链表的头元素。如 A->B->C->D->E->F->G->H->E 就是一个特殊链表，其中 head 为 A，tail 为 E。

现在你只知道 head，但是你想知道这个循环链表中有多少元素，但同时你的剩余空间十分的小，所以你想用尽可能小的空间来解决问题，请问你会怎么做？（依据占用空间给分）
评分标准：空间复杂度 O(n) -> 3 分；O(1)和 O(n)之间-> 5 分; O(1) -> 6 分
（对时间复杂度无要求）
算法错误，0 分；描述有问题，扣 2 分。
题意指的不是求整个链表的长度，而是求环的长度。
（假设链表的总长度是 n=LenA+C， 单链表部分的长度是 LenA，环的长度是 C=x+y）



上面是循环单链表的一个示例，注意单链表中元素只可能有一个 next 指针，所以循环单链表的形状只可能是 0 型或 6 型。（也即不可能有多个环）



几种解法：注意节点向后移动一定会进入循环，进入环后继续走必然是在环里打转。只要找到环中的一个节点，就可以通过在环中再走一圈来确定环的大小。

（1） 朴素：从 head 开始遍历，使用一个链表记录已访问过的节点，然后每次到新的节点就去查找是否已经访问过，若访问过，就找到了 Join 节点（此时总步数就是链表的总长度 n）。 空间占用 O(n)，时间开销与查找节点的索引有关（注意各个节点的指针各不相同），至少是 O(n)。

（2） 快慢指针： 声明两个指针，两个指针的初始值都是链表的头指针，其中一个指针每次前移两个节点，称为快指针，另一个指针每次前移一个节点，称为慢指针，然后让它们两个同时出发，因为链表中存在环，它们最终会第一次相遇（必在环内，设为 Pos 点），然后把其中一个指针移回到链表头的位置，也就是设置为链表头指针，然后让两个指针一个从相遇点出发，一个从链表头节点出发，两个都一次走一步，一直走到两个指针第一次相遇为止（必是 Join 点），这时两个指针都走了 LenA 的长度。空间占用 O(1)，时间复杂度 O(n)。

证明：http://blog.csdn.net/l294265421/article/details/50478818

（3） 倍增：记录从 head 走了 2^0, 2^1, 2^2, 2^3…, 2^k 步的节点，每次只用记录最后一个走了 2^k 步节点，然后在走到 2^(k+1)节点的过程中，若遇到了 2^k 这个节点，那个 2^k 节点（设为 t）就是循环中的节点。空间占用 O(1)，找到环中一个点的时间开销是 O(n)。

但这种方法只能找到环中的一个节点，接下来无法用跟第二种方法类似的手段直接找到 Join 点。如果要求解整个链表的总大小 n，有一种朴素的方法可以用来求解单链表的大小 LenA：每轮迭代中，求出从 head 扫描到环中节点 t 的长度；t=t->next，继续下一轮迭代。最后各轮迭代中求得的最短长度就是 LenA，加上环的长度 C 即

为总长度 n。

思考：事实上任何一种步数的序列，只要保证进入循环后这一轮的步数值超过环的大小，便总能找到环中的一个点。因为事先不知道环的大小，所以采用增量序列，比如 1,2,3,4,5…n 也是可以的。在各种增量序列中，2^k 这种序列有什么好处么？如果是 1,2…n 这种序列，时间开销是 O(n+C*C)；如果是 1,3,9…n 这种序列，时间开销也是 O(n)，但在环内会比 2^k 这种序列更快些。在进入环之前，应该小步走，否则超出单链表的部分是无用功；在进入环之后，应该大步走，尽快超过环的大小以找到相遇点。

上面两个算法均来源于 D Knuth. 1998. The Art of Computer Programming, Vol II: Seminumerical Algorithms. (1998).

2. （6 分）假设以 I 和 O 分别表示入栈和出栈操作，栈的初始和终态均为空，入栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列，称可以操作的序列为合法序列，否则为非法序列。如 IOIIOIOO 合法，IOOIOIIO 和 IIIOIOIO 为非法。请给出一个算法，判断所给操作序列是否合法。

评分标准：功能错误直接零分，按时间复杂度扣分。

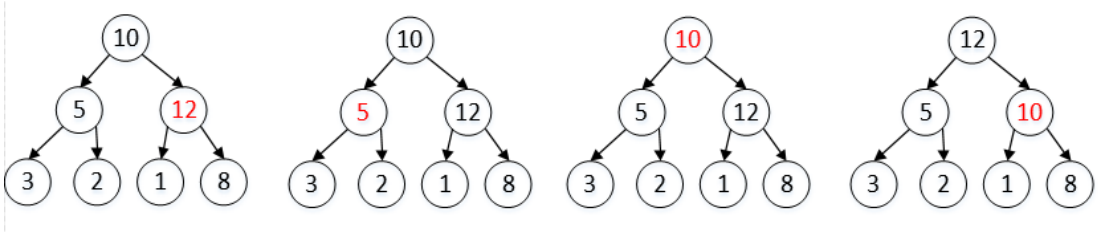
算法思想：从左到右扫描序列，要保证 I 的个数永远大于等于 O 的个数（否则会尝试从空栈中弹出元素，非法！），且最终 I 和 O 的数目相等，O（n）的时间复杂度。

3. （6 分）给定一棵用数组存储的完全二叉树{10, 5, 12, 3, 2, 1, 8}

评分标准：每一小问中，写错一个数则该小问得零分。

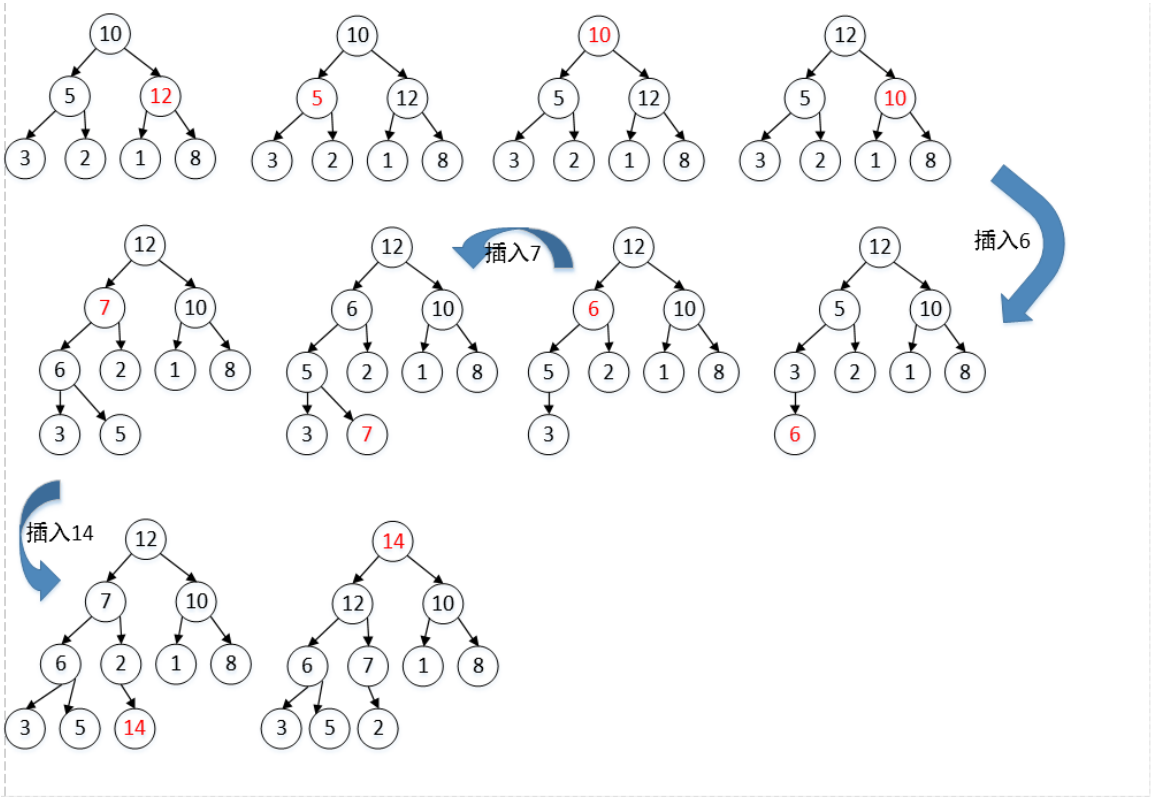
（1）用筛选建堆法将该完全二叉树调整为最大值堆。画出调整后得到的最大值堆，并说明在调整过程中都依次交换了哪些元素。（注：画堆只需要写出层次遍历序列结果即可）

12, 5, 10, 3, 2, 1, 8。 交换 10 和 12



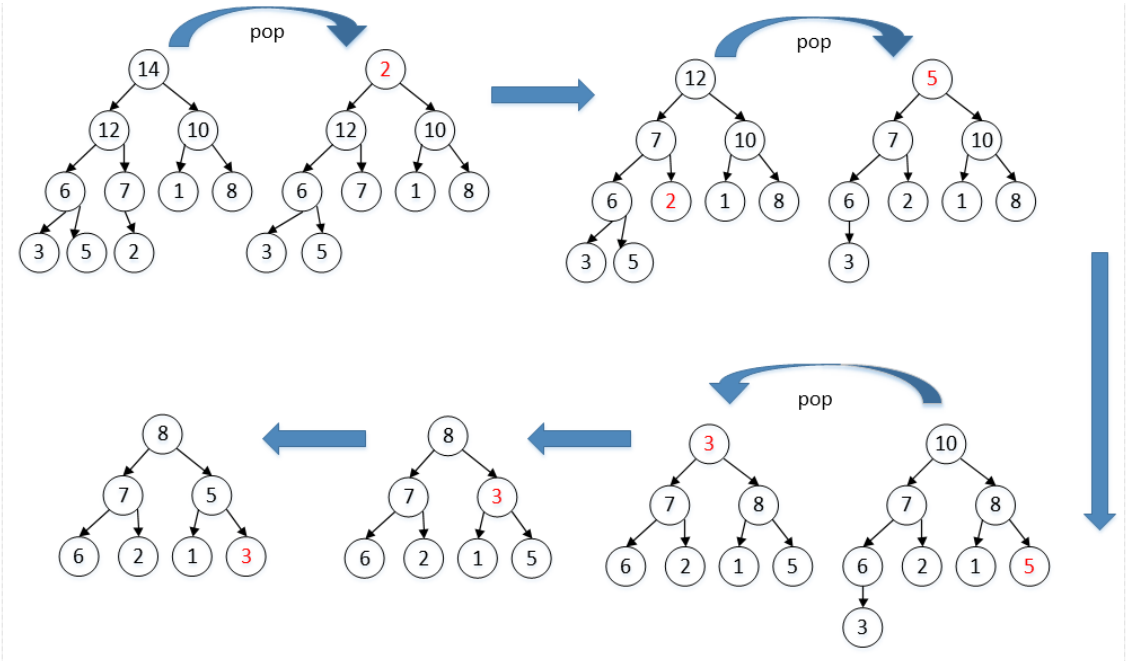
（2）将 6、7、14 按顺序插入（1）得到的最大值堆，堆的空间足够大，画出插入 14 后得到的最大值堆，并说明在插入 14 的过程中都依次交换了哪些元素。

14, 12, 10, 6, 7, 1, 8, 3, 5, 2。 交换 14 和 2，交换 14 和 7，交换 14 和 12



（3）对（2）得到的堆进行 3 次 deleteMax，画出第 3 次 deleteMax 后得到的堆，并说明在第 3 次 deleteMax 的过程中都依次交换了哪些元素。

8, 7, 5, 6, 2, 1, 3。 交换 3 和 8，交换 3 和 5



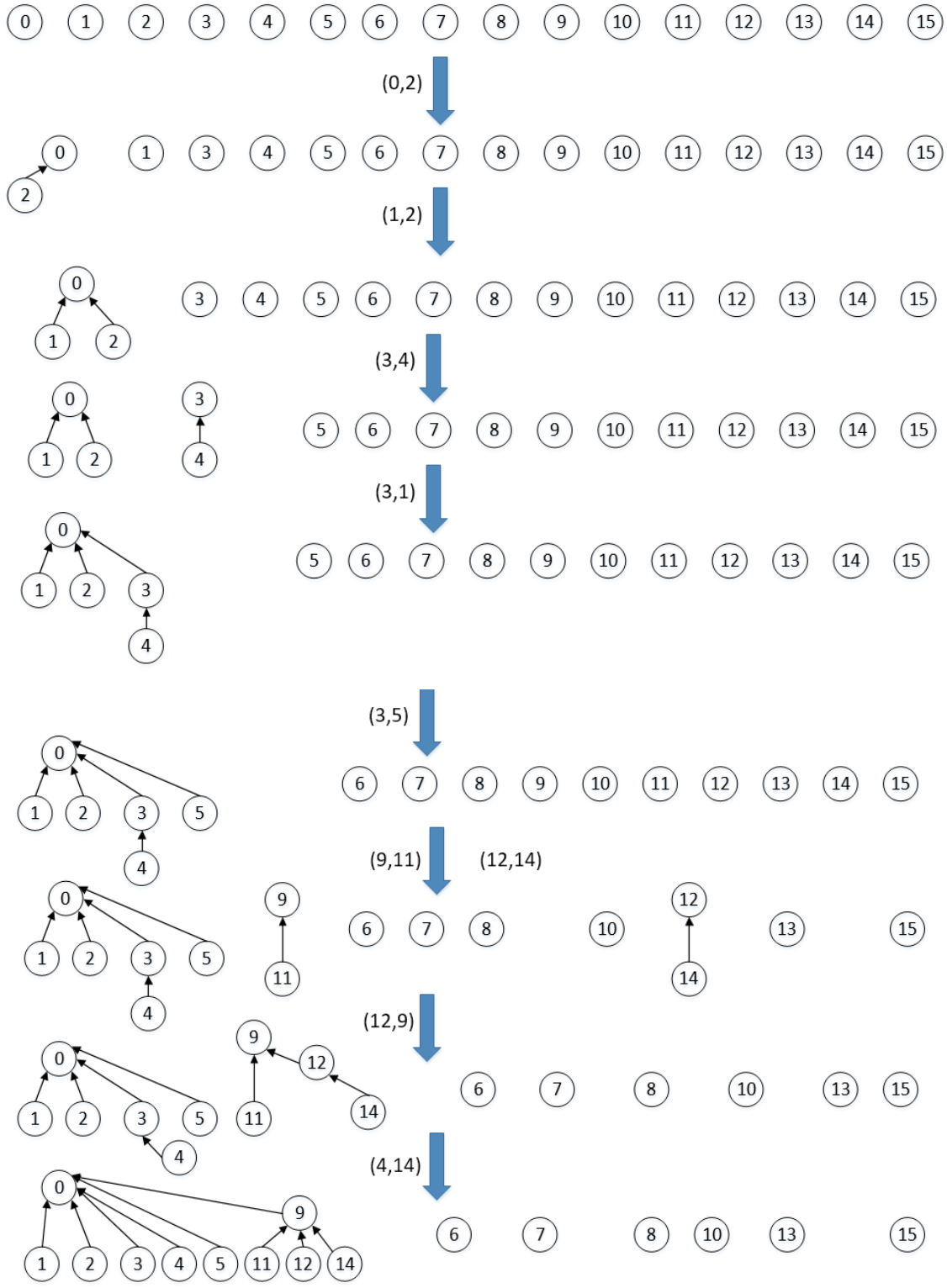
4. （6 分）对下列 15 个等价对进行合并，给出所得等价类树的图示。在初始情况下，集合中的每个元素分别在独立的等价类中。使用重量权衡合并规则，合并时子树结点少的并入结

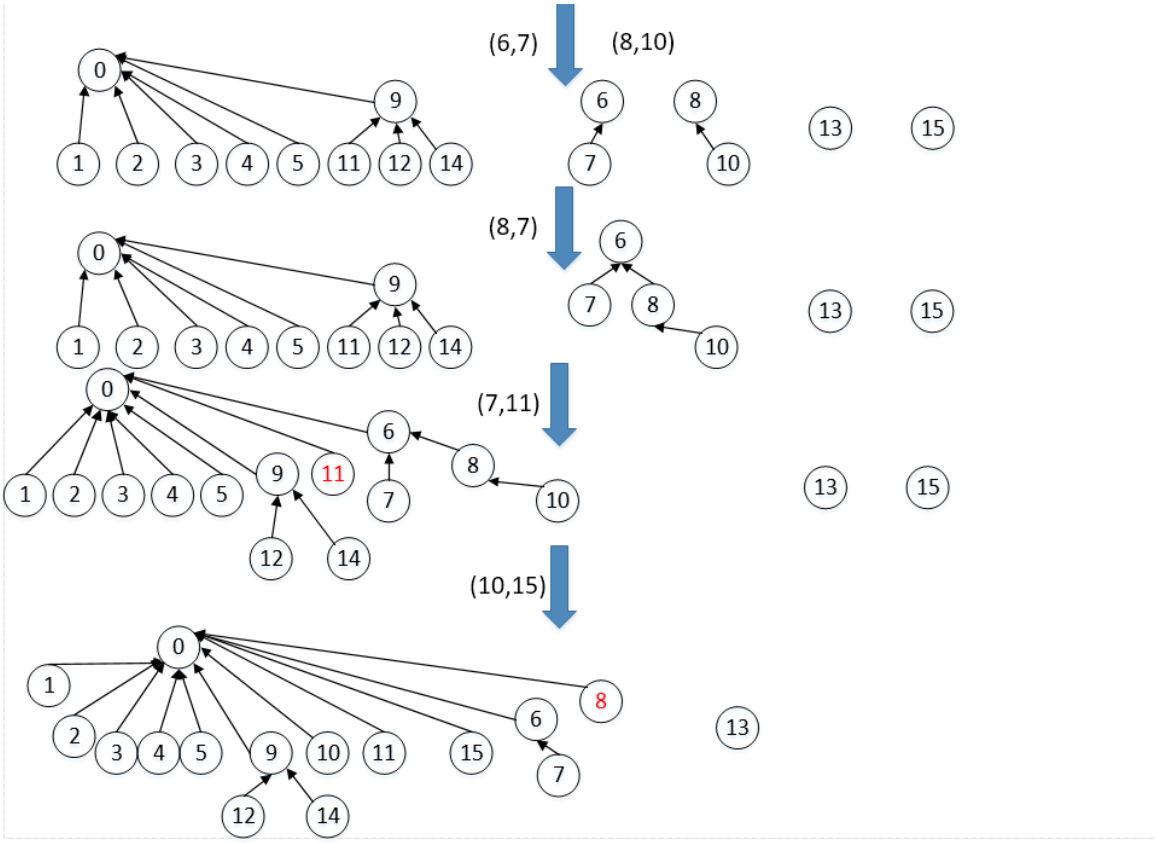
点数多的那棵（多的那个作为新树根，少的那个根作为新根的直接子结点）；若两棵树规模同样大，则把根值较大的并入根值较小（新树根取值小的）。同时，采用路径压缩优化。

(0,2) (1,2) (3,4) (3,1) (3,5) (9,11) (12,14) (12,9) (4,14) (6,7) (8,10) (8,7) (7,11) (10,15) (10,13)

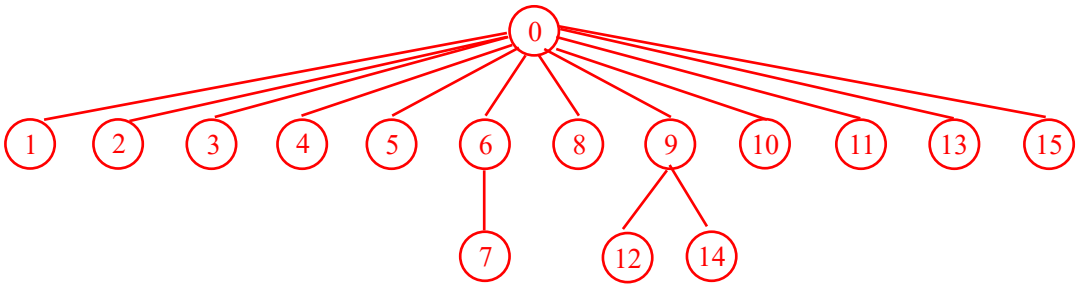
评分标准：最终结果错误，则得零分。

具体过程如下所示：





最后加入(10,13)这条边，得到最终结果如下图



得分

三、 算法填空(每空 3 分，共 24 分)

1. 下面的算法利用一个栈将一给定的序列从小到大排序。长度为n的序列由1, 2, …, n这n个连续的正整数组成。请补全下面的代码段，使其可以判断给定的序列是否可以利用一个栈进行排序，如果可以，输出相应的操作。

例如：序列 4 3 1 2，此序列可以排序，输出：push push push pop push pop pop pop。

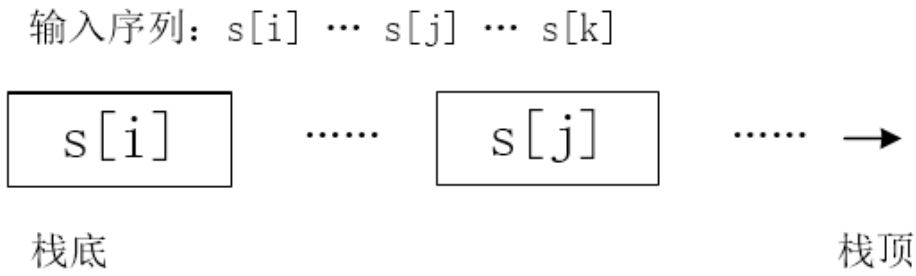
```
template <class T>                          // 栈的元素类型为 T
class Stack {
        public:
                void clear();               // 变为空栈
                void push(const T item);    //   item 入栈
                T pop();                    // 返回栈顶内容并弹出
                T top();                    // 返回栈顶内容但不弹出
```

```
                bool isEmpty();                         // 若栈已空返回真
                bool isFull();                          // 若栈已满返回真
};
Stack<int> s;                                           //  s 为栈

int sequence[maxLen],  len;                             // sequence 为待排序序列， len 为序列长度
bool islegal()  {                                       // 判断序列是否可以排序
    int i, j, k;
    for (i = 0; i < len; i++) {
        for (j = i+1; j < len; j++) {
            for (k = j+1; k < len; k++) {
                if (sequence[k] < sequence[i] && sequence[i] < sequence[j])
                    return false;
            }
        }
    }
    return true;
}
void transform()  {                                     // 输出转换操作
    int cur = 1,  i = 0;
    while  (cur <= len) {
        while (s.isEmpty()  || cur < s.top()) {
            s.push(sequence[i++]);
            cout << "push ";
            if (cur == s.top()) break;
        }
        s.pop();
        cout << "pop ";
        ++cur;
    }
}
```

解析：比如 2 3 1 这个序列就是不可被栈排序的，当 1 被输出后，2 在栈底，3 在栈顶，接下来 2 无法被排序。

必要性：若存在 sequence[k]<sequence[i]<sequence[j]，cur 先增长到 sequence[k]，此时栈的情况如下图。接下来 sequence[k]可以被输出，但当 cur 增长到 sequence[i]时，sequence[i]已无法被排序。



充分性：类似第三章书面作业中栈的合法序列的充分性证明。设 s[j0]=n，那么必有 s[i]<s[k] 对任意 i<j0<k 成立。因此 j0 左边一定是 1~j0-1 的排列，j0 右边一定是 j0 到 n-1 的排列，由归纳法可证。

答：

① sequence[k] < sequence[i] && sequence[i] < sequence[j]

② cur <= len

③ cur < s.top()

cur 是下一次应该输出的数字，只有在每次成功输出元素的时候才会自增，所以比 cur 小的数字都已经被输出了，而 s.top()对应的数字还没有被输出。因为输出结果一定是从小到大的，s.top()没被输出，那么 cur 不可能比 s.top 更大，所以！=应该是对的。

④ s.push(sequence[i++]);

拓展：双栈排序

http://blog.csdn.net/linwh8/article/details/52606751

2. 下面的算法将一个用带右链的先根次序法表示的森林转换为用带度数的后根次序法表示。请利用题目给出的树结点ADT和栈ADT，填充空格，使其成为完整的算法。

```
template<T>          // 数据结构
class RlinkTreeNode {
    int ltag;                    // 左标记
    T info;
    RlinkTreeNode<T> *rlink; // 右链
}
template<T>
class PostTreeNode {
    T info;
    int degree;                  // 度数
}
// 算法描述：递归遍历用带右链的先根次序法表示的森林，同时计算出度数并转成后根次
            序森林；设带右链的先根次序法表示的森林为数组
RlinkTreeNode RlinkTree[n];
// 带度数的后根次序法表示的森林为数组
PostTreeNode PostTree[n];
int count = 0;                       // 计数器
int convert(RlinkTreeNode * node) { // 返回值表示所有右兄弟的数量（包括自己）
```

```
    int tmp;
    if (1 == node->ltag) {                  // 若没有左子结点，将该结点输出到 PostTree
        PostTree[count].info = node->info;   （填空 5）
        PostTree[count].degree = 0; //  度数必然是 0
        count++;
    }
    else {                                  // 有左子结点则压栈
        tmp = convert(node+1);          // 访问第一个子结点，并返回度数
        PostTree[count].info = node->info;
        PostTree[count].degree = tmp;      （填空 6）
        count++;
    }
    if (NULL != node->rlink     （填空 7）) {   // 有右兄弟(，访问并返回右边兄弟的数量+1)
        return convert(node->rlink) + 1;
    }
    else {                                  // 没有右兄弟
        return 1;    （填空 8）
    }
}
```

答案：

填空 5：PostTree[count].info = node->info;

填空 6：PostTree[count].degree = tmp;

填空 7：NULL != node->rlink

填空 8：return 1;

或者，如果没有注意到注释“包括自己”的话

填空 6：PostTree[count].degree = tmp + 1;

填空 8：return 0;

得分

四、 算法设计与实现 (14 分)

1. （8 分） 编写算法伪码，对给定的字符串 str，返回其最长重复子串及其下标位置。如 str=”abcdaccdac”，则子串”cdac”是 str 的最长重复子串，下标为 2。（重复子串允许重叠）

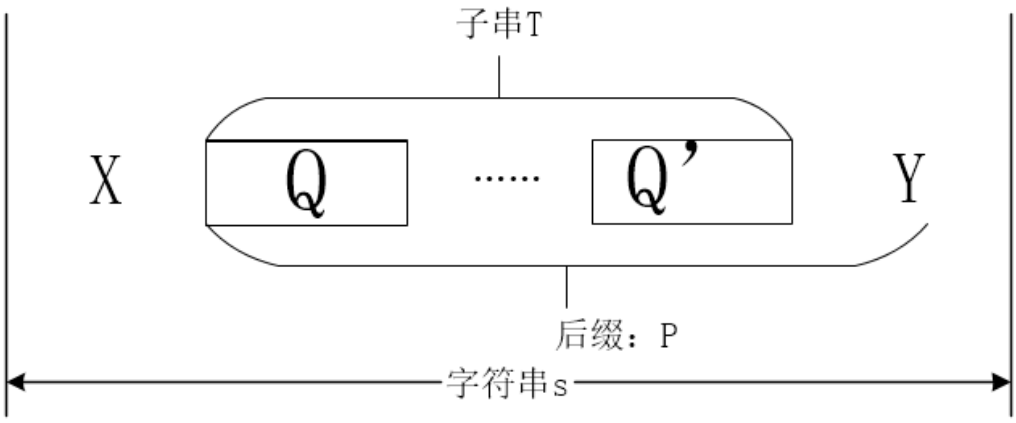
评分标准：按时间复杂度给分，功能错误得零分，不超过 O(n*n)复杂度得满分，O(n*n*logn)复杂度扣 2 分，O(n*n*n)以上复杂度扣 4 分。没有伪码扣 4 分，使用 KMP 方法但没写出 next 数组的求解过程则扣两分。使用 strstr 等判定子串的库函数，按照复杂度不达标扣分。

答案：标准方法：时间复杂度 O(n*n)，空间复杂度 O(n)

每个重复子串一定有一个起始点，每个起始点对应着一个子串 P（实际上是 str 的一个后缀）。子串 P 中从 P 的初始位置开始的最长重复子串 Q 可以用 P 的 next 数组求得，即 Q 的长度等于 next 数组中最大的数值。原因：Q 是 P 中从 P 的起点开始的重复子串，则 P 中必有另一子串 Q’，和 Q 完全一致但坐标不完全相同。那么对于从 P 的起点到 Q’的终点构成的子串 T 而言，T 的前缀和后缀的最长公共部分一定是 Q 和 Q’。

比如 abcdaccdac 这个字符串，cdac 的起始点对应的子串 P 是 cdaccdac，这也恰好就是子串

T，T 的前缀和后缀的最长公共部分就是 cdac。当我们枚举了所有的起始点（对应所有的后缀 P），那么整体的最长重复子串也一定包含在里面。




```
for (index=0;index<len-1;index++)
{
    int *next=new int[len-index+1]; //compute 1 more element
    getNext(str+index,next);//start from str+index
    nextMax = max(next);  //find the maximum value of this next array
    if(nextMax>max)
    {
        max=nextMax;
        maxIndex=index;
    }
}
```

http://dsqiu.iteye.com/blog/1701324

利用后缀数组的方法：-2 分，时间复杂度是 O(n*n*logn)，空间复杂度是 O(n)（后缀数组中对每个后缀只需存储其起始点）

先计算 str 的所有后缀，共有 n 个，对它们排序后（每次比较是 O(n)的，排序的总代价是 O(n*n*lgn)），求出后缀数组相邻元素的最长公共前缀。

比如字符串 aba 的后缀有 a, ba, aba 三个，排序后是 a,aba,ba，对相邻元素求公共前缀得 a 为最长重复子串。

朴素方法：-4 分，时间复杂度 O(n*n*n)，空间复杂度是 O(1)

枚举两个指针 i 和 j，i<j，找出从 i 开始的子串和从 j 开始的子串的最长公共前缀。

后缀数组可以利用倍增法进行优化，达到 O(n*logn*logn)的复杂度。其他算法如果在保证正确的前提下时间复杂度不超过 O(n*n)，也应该给满分。（比如 dc3 或者 trie 树等等）

http://www.cnblogs.com/janmelin/p/6260033.html

http://www.cnblogs.com/jianglangcaijin/p/6035937.html

2. （6 分）给出一个算法，求一棵树的树高。可以写出伪代码，需要相应的注释，或者写出详细算法思路。对时间复杂度无具体要求。

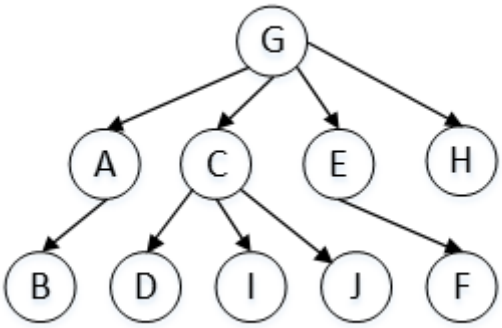
评分标准：功能错误直接零分，伪代码可有可无，按注释给分。

参考答案：

（1） 递归求解：取左子树和右子树深度的最大值，然后加 1 即可。由此递归处理下去，直到一棵树只有一个根结点的时候，可知其深度为 1，这里是递归的边界条件，这时可以返回上一层的递归。 （实际上是以叶节点高度为 0，求根节点的最大高度）

（2） 深度优先遍历：周游树，可以用递归（当前深度作为参数传递），也可用一个栈实现。栈中元素数目的最大值也就是树的高度（也即树的最长路径）

（3） 广度优先遍历：借助一个队列实现，最后也能找到最远的叶节点（实际上是以根节点深度为 0，求叶节点的最大深度）。



得分

五、 分析证明题 (共 14 分)

1. （8 分）二叉树的内部路径长度是指所有节点的深度的和。假设将 N 个互不相同的随机元素插入一棵空的二叉搜索树。请证明得到的二叉搜索树的内部路径长度期望为 O(NlogN)。

评分标准：写对公式则有分，其他情况酌情给分

证明过程：

设 D(N)为 N 个结点的二叉搜索树的内部路径长度，如果根的左子树有 i 个结点，则 D(N)=D(i)+D(N-i-1)+N-1。

如果根的左子树有 i 个节点，则插入序列的第一个元素是 N 各元素中的 i+1 小的元素，由于序列的随机性，根的左子树有 i 个节点(0<=i<=N-1)概率分别为 1/N。

则期望内部路径长度 $D(N) = (\sum_{i=0}^{N-1}(D(i) + D(N-i-1)))/N + N - 1 = (2/N)(\sum_{i=0}^{N-1} D(i)) + N - 1 = O(NlogN)$

类似快速排序的分析，任意元素被选为根（基准）的概率相同，递推公式的具体求解过程如下。注意在给定二叉搜索树的情况下，向左右插入元素的概率不一定相同（与区间大小有关）。

$$D_n = n - 1 + \frac{2}{n} \times \sum_{i=0}^{n-1} D_i, D_1 = 0$$

$$n \times D_n = n \times (n-1) + 2\sum_{i=0}^{n-1} D_i$$

$$(n+1) \times D_{n+1} = (n+1) \times n + 2\sum_{i=0}^{n} D_i$$

$$(n+1) \times D_{n+1} - n \times D_n = 2n + 2D_n$$

$$(n+1)D_{n+1} = (n+2)D_n + 2n$$

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

$$C_n = \frac{D_n}{n+1}, C_1 = 0$$

$$\begin{aligned} C_{n+1} &= C_n + \frac{4}{n+2} - \frac{2}{n+1} \\ &= C_{n-1} + 4(\frac{1}{n+2} + \frac{1}{n+1}) - 2(\frac{1}{n+1} + \frac{1}{n}) \\ &= \dots \\ &= C_1 + 4\sum_{i=3}^{n+2} \frac{1}{i} - 2\sum_{i=2}^{n+1} \frac{1}{i} \end{aligned}$$

$$\begin{aligned} C_n &= 4 \times \sum_{i=1}^{n+1} \frac{1}{i} - 2 \times \sum_{i=1}^{n} \frac{1}{i} - 4 \\ &= 4 \times (\ln n + 1 + c) - 2 \times (\ln n + c) - 4 \\ &= \ln \frac{(n+1)^4}{n^2} + 2c - 4 \\ &= O(\log n) \end{aligned}$$

$$D_n = (n+1) \times C_n = O(n \log n)$$

2. （6 分）证明按先根次序周游树获得的序列与其对应二叉树的前序周游获得的序列相同。

评分标准：视证明严谨性给分，少考虑情况要酌情扣分。

证明过程：

不妨设先根次序周游树得到的序列为 S_1，前序周游二叉树得到的序列为 S_2，则要证明 $S_1 = S_2$，即证明对$\forall(x1,x2) \in S_1$，则一定有$(x1',x2') \in S_2$，其中$x1'$和$x2'$分别是树中结点x1和x2对应二叉树的结点。

（注意：(x1,x2)表示节点 x1 和 x2 之间的偏序关系，当任意一对节点的偏序关系确定后，整个序列也就被确定了。）

若$(x1,x2) \in S_1$，即 x1 出现在 x2 之前，那么分以下三种情况讨论：

（1）若x2在x1的子树中，那么对应二叉树中，$x2'$在$x1'$的左子树中，一定有$(x1',x2') \in$

S_2。

（2）若x2在x1的兄弟树 T（即 T 的根 y 与 x1 有相同的父节点）中，那么对应二叉树中，$x2'$在$x1'$的右子树中，一定有$(x1', x2') \in S_2$。（T 的根 y 一定在 x1 的右边，否则 x2 将出现在 x1 之前）

（3）如果不是以上两种情况（如下图），设树根到 x1 的路径为 p1，树根到 x2 的路径为 p2，则必定存在某个 i>0，使得 p1(j)==p2(j)对任意 j<i 成立且 p1(i)不等于 p2(i)。两条路径从 p1(i)和 p2(i)这里开始分叉，这两个节点互为兄弟且 p1(i)在 p2(i)左边。在把树转换为二叉树时，x1' 在 p1(i)' 的左子树中，以 p2(i)' 为根的子树在 p1(i)' 的右子树中，也即 x2' 在 p1(i)' 的右子树中。二叉树中作前序周游时，左子树的点一定在右子树的点之前被访问，所以 x1' 出现在 x2' 之前，即一定有$(x1', x2') \in S_2$。

所以得证$\forall(x1, x2) \in S_1$，则一定有$(x1', x2') \in S_2$，所以两个序列相同。

这个映射关系必为单射，因为(A,B)和(A,C)不可能对应到同一对顶点。又因为两边的点对数目相同，所以必为满射，因此这个映射关系是双射，满足充分性和必要性的要求。