# 1

(1) $k^d$，其中 $d$ 为层数.

证明：每个结点都有 $k$ 个子结点，因此第 $d$ 层总结点数为第 $d-1$ 层总结点数的 $k$ 倍，以此类推即得.

(2) $ki+m+1-k.$

证明：计算即得.

(3) $k \nmid (i-1). i+1.$

证明：从根节点的第一个子结点开始，每 $k$ 个结点为一组，最后一个没有右兄弟.

# 2

对于树，按"根—右子—左兄"的顺序进行遍历，保留遍历序列和必要的信息（是否有左兄弟、是否有子结点），再将这个序列看作按"根—左子—右兄"的顺序遍历得到的，还原出树，即为所求. 根据课堂所讲，这是可以做到的.

对于多棵树的森林，加入一个根节点，将其转化为树，作镜面反射，再去掉根节点即可.

这个算法的时空复杂度都是 $O(N)$，其中 $N$ 为森林的总结点数.

# 3

下面采用与课程幻灯片中相同的表示方法.

带右链的先根次序存储：

| **下标** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| rlink | 7 | 5 | 4 | -1 | -1 | 6 | -1 | 10 | 9 | -1 | -1 |
| info | A | B | D | H | I | E | J | C | F | K | G |
| ltag | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

带度数的后根次序存储：

| **下标** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| degree | 0 | 1 | 0 | 2 | 0 | 0 | 3 | 0 | 0 | 2 | 0 |
| info | H | D | I | B | E | J | A | F | K | C | G |

# 4

下面以结点数据的类型为 `int` 为例编写代码，并假定输入数据是合法的.

```
struct DegNode{
    int deg, data;
};

struct TreeNode{
    int sib, son, data;
};

// N is the number of nodes
DegNode arr[N];    // the sequence to be dealt with. arr[0] is an imaginary root
TreeNode res[N];   // result will be stored here

void solve(){
    for (int i = 0; i < N; ++i)
        res[i].data = arr[i].data;

    int h = 0, t = 1, deg;
    res[0].sib = -1;

    int idx = 1;
    while (h < t){
        deg = arr[head].deg;

        if (deg == 0){
            res[h].son = -1;
        }
        else{
            res[h].son = t++;
            for (int i = 1; i < deg; ++i)
                res[t-1].sib = t, ++t;
            res[t-1].sib = -1;
        }

        ++head;
    }
}
```

这个算法很简单，只是模拟层次遍历的执行过程而已，无需赘述其思想.

这个算法的时空复杂度都是 $O(N)$，其中 $N$ 为森林的总结点数.

## 5

初始化 $n$ 个单元素并查集，遍历等式并用并查集合并（以相等作为等价关系），再遍历不等式，若有冲突则无解，否则有解.

并查集不同实现中单次 find 操作的平均时间代价：

| 实现方式 | 最好时间 | 平均时间 | 最坏时间 |
| --- | --- | --- | --- |
| 无优化 | $O(1)$ | $O(\log n)$ | $O(n)$ |
| 按秩合并 | $O(1)$ | $O(\log n)$ | $O(\log n)$ |

| 实现方式 | 最好时间 | 平均时间 | 最坏时间 |
| --- | --- | --- | --- |
| 路径压缩 | $O(1)$ | $O(\alpha(n))$ | $O(\log n)$ |

其中 $\alpha$ 为反阿克曼函数.