最近公共祖先
—倍增算法

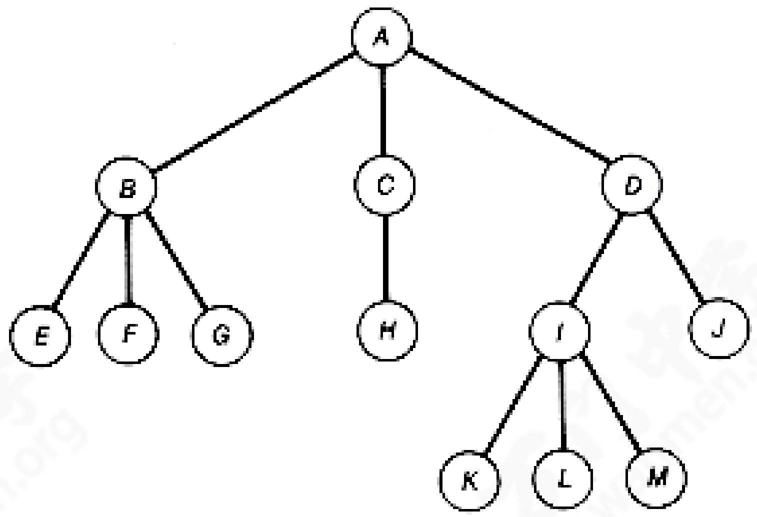
主讲人：邓哲也

大纲

- 树的定义
- LCA的定义
- 暴力算法
- 倍增算法

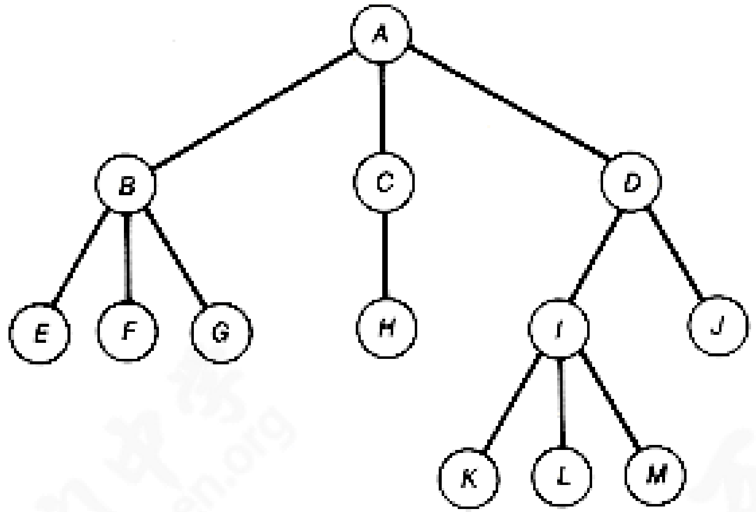
回顾树的定义

- **树**是一种无向图，其中任意两个顶点间存在唯一一条路径。
- 在一棵树中可以指定一个特殊的节点作为**根**。一个有根的树叫做**有根树**。
- 有根树的节点可以根据到根的距离分层，我们可以定义一个点的**深度**为它到根的距离。
- 一条边的两个端点中，靠近根的节点叫做另一个节点的**父节点**。
- 沿着父节点一直到根的所有节点都叫做这个点的**祖先**。



LCA的定义

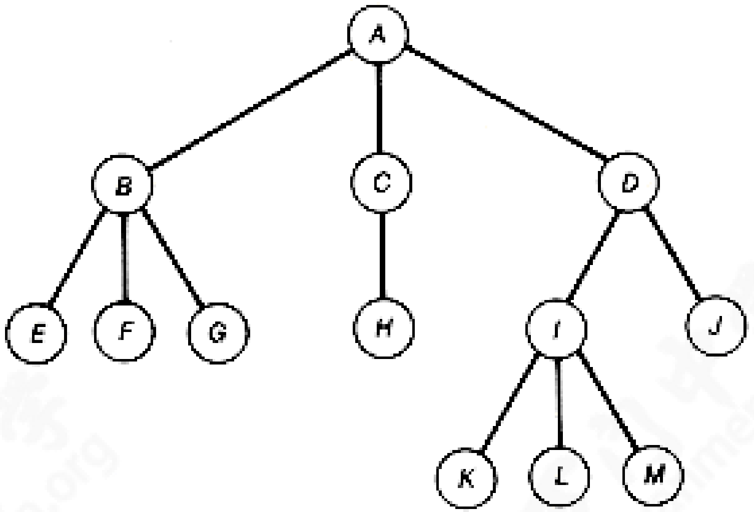
- 定义：给出有根树T中的两个不同的节点u和v，找到一个**离根最远**的节点x，使得x**同时**是u和v的**祖先**，x就是u和v的**最近公共祖先**（Lowest common ancestor）



- 例子：
  - B和M的最近公共祖先是A
  - K和J的最近公共祖先是D
  - C和H的最近公共祖先是C

暴力算法

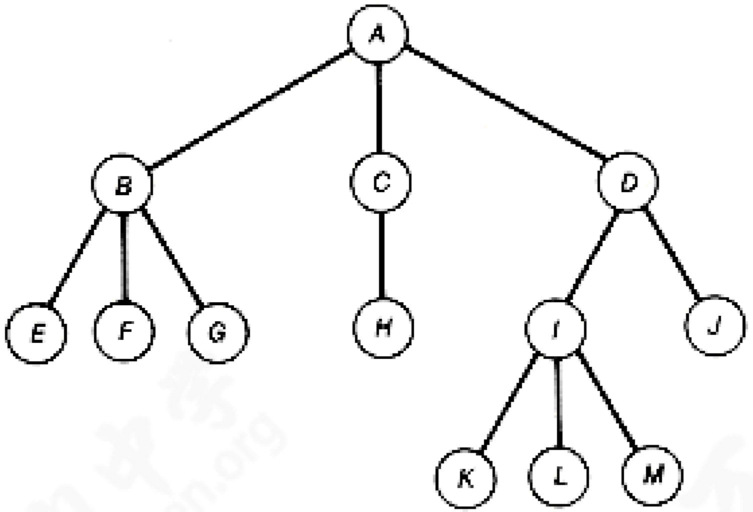
- 简单算法：
  - 先从u往根节点遍历，经过的点都打一个标记
  - 再从v往根节点遍历，路上碰到的第一个打标记的点就是它们的LCA.



- 时间复杂度：一次询问是O(n)的，n为树的节点个数。

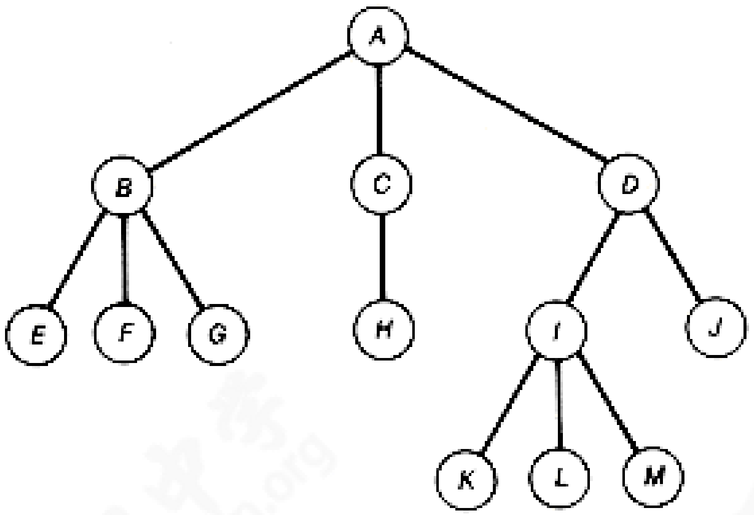
倍增算法

- 我们记fa[u]为u的父节点，即从u向根走1步能到达的节点，对根节点我们记fa[root] = 0.
- 记up[u][k]为从u向根走2^k步到达的节点。
- 显然有
  - up[u][0] = fa[u]
  - up[u][k] = **up[up[u][k - 1]][k - 1]**
- 例如
  - up["K"][0] = "I",
  - up["I"][0] = "D",
  - up["K"][1] = "D".



倍增算法

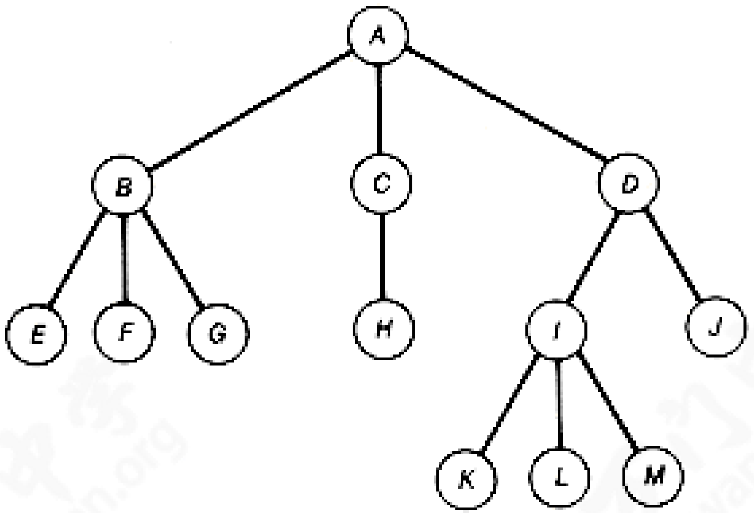
- 那么，有了up[u][k]数组之后我们怎么求LCA呢？
- 若u和v的深度不同，则通过up数组先把他们调整到同一深度
- 我们从大到小枚举k，比较up[u][k]与up[v][k]
  - 若up[u][k] == up[v][k]
    - 说明up[u][k]是u和v的公共祖先
  - 若up[u][k] != up[v][k]
    - 说明u和v还没有走过LCA
    - 令u = up[u][k]，v = up[u][k]
- 可以证明，最后u和v的父亲就是所求的LCA.



倍增算法

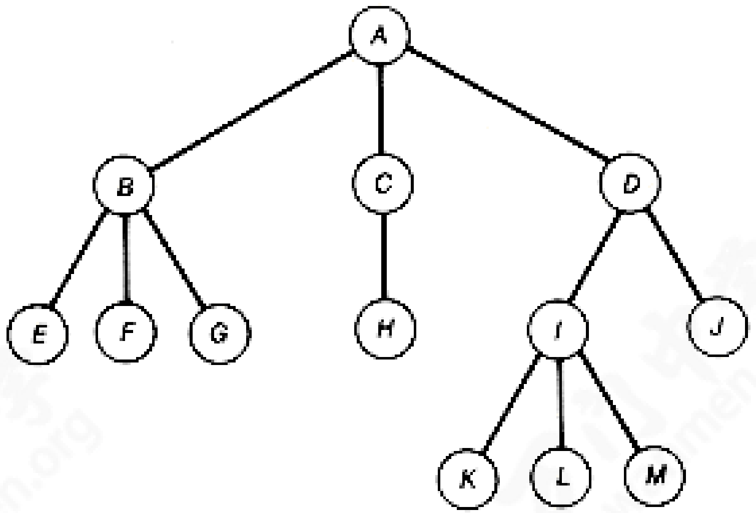
- 代码实现：

```
int lca(int u, int v) {
    if (dep[u] < dep[v])    // 令u是深度大的点
        swap(u, v);
    if (dep[u] != dep[v]) { // 若深度不同，则让u往上爬到和v同一深度
        for (int k = 20; k >= 0; k --)
            if (dep[up[u][k]] >= dep[v])
                u = up[u][k];
    }
    if (u == v)             // 若此时u已经等于v，就可以直接返回了
        return u;
    for (int k = 20; k >= 0; k --)
        if (up[u][k] != up[v][k])   // u和v一起向上爬
            u = up[u][k], v = up[v][k];
    return up[u][0];
}
```



倍增算法

- 时间复杂度分析：
  - 预处理：
    - fa数组：通过dfs或bfs遍历一遍树得到，O(n)
    - up数组：第一层大小为n，第二层大小为log n，故为O(n log n)
  - 每次询问：
    - 两个点向上爬的次数是O(log n)的。
- 于是我们在O(n log n) – O(log n)的时间复杂度下完美的解决了LCA问题，且倍增算法是在线的。
- 后续我们将介绍两个时间复杂度更优秀的在线和离线做法。



下节课再见