

问题概述

LCA的全称是Least Common Ancestors，顾名思义，LCA问题就是求树上两个点的最近公共祖先的问题。由定义，两个点的最近公共祖先是满足这两个点都在其子树中的点里面深度最大的那个，因此有可能这两个点的最近公共祖先就是其中一个。

倍增算法求LCA

求LCA最简单的方法就是先把两个点中深度更深的那个移动到和另外一个深度相同，然后两个一起向自己的父亲移动，直到移动到同一个点，但是这个方法需要花费的时间较长，我们考虑如何优化这一算法。我们知道，任何一个数都可以表示成二进制，所以我们如果能每次向上跳一个2的幂次深度，我们就可以把时间复杂度从 $O(n)$ 优化到 $O(\log n)$ 。

令 $f[i][j]$ 表示点 i 的第 2^j 个祖先是谁，显然 $f[i][0]$ 就是 i 的父亲，而这个信息我们可以递推，显然 $f[i][j] = f[f[i][j-1]][j-1]$ ，即我的第 2^{j-1} 个祖先的第 2^{j-1} 个祖先是我的第 2^j 个祖先，所以我们只要知道了所有的 $f[i][0]$ 就可以推出所有的信息。

在实际求两个点的LCA时，我们依照刚刚的想法，先把深度更深的点向上跳到和深度浅的点同深度，这一过程可以依据 f 数组在 $O(\log n)$ 的时间内完成，然后两个点一起往上跳，时间也是 $O(\log n)$ ，正是因为这一算法的信息是以2的幂次为基础，所以被称为倍增算法。具体代码如下

```
inline void init()
{
    for(int j = 1; j <= deps + 1; ++j)
        for(int i = 1; i <= n; ++i)
            f[i][j] = f[f[i][j - 1]][j - 1];
}

inline int lca(int a, int b)
{
    int i, j;
    if(deep[a] < deep[b])
    {
        int y = a; a = b; b = y;
    }

    for(j = deps; j >= 0; --j)
        if(deep[a] - (1 << j) >= deep[b])
            a = f[a][j];
    if(a == b)
        return a;

    for(j = deps; j >= 0; --j)
        if(f[a][j] != -1 && f[a][j] != f[b][j])
            a = f[a][j], b = f[b][j];
    return f[a][0];
}
```

Tarjan算法求LCA

除了倍增算法，Tarjan爷爷发明的Tarjan算法也可以用来求LCA。

首先我们需要记录下所有的查询，因为用Tarjan求LCA必须要离线才能做，如果形式是在线就无法使用Tarjan。

接下来我们要从根开始遍历所有的点，当一个点的子树被遍历完以后，把它的所有儿子在并查集中与它合并并回溯到它的父亲。当我们遍历到一个点的时候，我们依次查看所有与这个点相关的查询，若另一个点已经被遍历过，那么它在并查集中的父亲就是这一组询问的LCA。这一算法的原理是，我们遍历到一个点时，设这个点是 x 。若我们查询了 (x,y) 且 y 已经被遍历，那么 y 一定已经向上合并了，设 $LCA(x,y)=z$ 。那么由于我们刚刚遍历到 x ，说明 z 肯定还没有被向上合并，而 y 一定已经合并到了 z 。否则说明 x 和 y 在 z 的同一个子树里，那么 z 显然不是 x 和 y 的LCA。

更具体的讲解和代码可以看[这篇博客](#)，其中最后还有一个可以手动控制的算法流程详解，可以帮助理解。

树链剖分求LCA

(留在树链剖分部分再写，到时候给链接)

RMQ算法求LCA

例题——BZOJ1001狼抓兔子

题目大意

解题思路

代码实现

From:
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:
<https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:hotpot:lca&rev=1594882544>

Last update: 2020/07/16 14:55