数据结构第二讲

一些关于树的统计问题

丁尧尧

上海交通大学

February 2, 2018

目录

1. 一些关于树的统计问题
2. DFS 序
   - 概念
   - 应用
3. 最近公共祖先
   - 问题
   - 朴素做法
   - 倍增做法
   - RMQ 做法
   - 后两个问题

关于树的一些统计问题

算法竞赛中，我们经常遇到一些和树有关的统计问题（以点、链、子树作为修改和询问对象）。
这里列出一些基本的问题：

1. 单点修改，子树询问
2. 子树修改，单点询问
3. 子树修改，子树询问
4. 单点修改，链询问
5. 链修改，单点查询
6. 链修改，子树修改，链查询，子树查询

上面最后一个是前五个的更一般的情况（点是特殊的链），我们留到后面解决，今天主要解决前面的问题。要解决前面 5 个问题，我们需要学习一下最近公共祖先和 dfs 序这两个概念。

DFS 序

DFS 序有三种，我们姑且用序列的长度来对它们进行分类：

- n 个点，这是最简单、最常用的一种
- $2*n$ 个点，这个用得不多，但还是有些有用的性质
- $2*n-1$ 个点，这个主要是用于 $O(1)$ 求 lca

n 个点

先看看我们怎么得到:

```
int in[N], out[N], seq[N], idc;
void dfs( int u, int f ) {
    seq[++idc] = u;
    in[u] = idc;
    for( int t = head[u]; t; t = last[t] ) {
        int v = dest[t];
        if( v == f ) continue;
        dfs( v, u );
    }
    out[u] = idc;
}
idc = 0;
dfs( root, root );
```

这种 DFS 序满足：

- 每个节点和 DFS 序中的位置一一对应，即 u 所在位置是 in[u]，位置 i 对应节点是 seq[i]。
- 每棵子树在 DFS 序中是连续的，即 u 节点代表的子树的所有节点都在 [in[u],out[u]] 中。

有了上面的两个性质，子树修改就是 DFS 序上的区间修改，子树询问就是 DFS 序上的区间询问，从而可以用树状数组或线段树解决前三个问题。

2*n 个点

这个又称树对应的一个括号序列。与上面不同的是，我们不光进来的时候将点加到序列中，出去的时候也加一次。
如果我们把进来的时候那次改成左括号，出去的那次看成右括号，那么我们得到的就是一个层层嵌套的括号序列。
`[1,in[u]]` 中那些没有被其有括号匹配的左括号都是从根节点到 `u` 的点对应的左括号。
`[in[u]+1,in[v]]` 中那些匹配剩下的括号个数就是 `u` 到 `v` 的节点数。

2*n-1 个点

DFS 的过程实际上也是一个遍历边的过程，我们把边顺次接起来，那么那些串起来的节点按照顺序排列起来就是这种 DFS 序，有 $2m+1=2n-1$ 个节点。

如果我们用 `in[u]` 表示序列中 `u` 节点第一次出现的位置，那么 `[in[u],in[v]]` 中深度最小的节点就是 `u` 和 `v` 的最近公共祖先（下面讲）。

最近公共祖先

Definition

最近公共祖先 树 T 中，如果 a 在 u 到根的路径上，我们称 a 是 u 的祖先。
如果 a 既是 u 的祖先，又是 v 的祖先，并且其深度是所有满足条件的点中最大的，我们称其为 u、v 的最近公共祖先。

求最近公共祖先是一个很基本的问题，许多其他的算法都需要快速地求出任意两个点的最近公共祖先。

朴素做法

我们可以 dfs 一遍这颗树，把每个点的深度和父亲搞出来，然后每次将 u 和 v 中深度较大的那个点跳到它父亲，直到两个点重合：

```
whie( u != v ) {
    if( dep[u] > dep[v] )
        u = fat[u];
    else
        v = fat[v];
}
return u;
```

但这样最坏是 $O(n)$ 复杂度的。

倍增求 LCA

我们上一讲学习了 ST 表，这次我们类似地定义：

```
int anc[N][P+1];
```

anc[u][p] 表示 u 节点向上“跳”了 2^p 步之后到达的节点（我们可以认为根节点向上跳一步又跳回了自己）。
和 ST 表一样，我们可以用 $O(nlogn)$ 的时间将 anc 数组求出来。
然后算法的大概思路是：

1. 让深度较深的点向上跳，直到两个点的深度相同
2. 让两个点一起跳，跳到它们的 lca

RMQ 做法

上面单次查询是 $O(log(n))$ 的，其实还有查询更块的算法（$O(1)$ 查询）。就是用上面提到的 DFS 序。有了第三种 DFS 序的性质之后，我们可以**以深度为比较关键字**，建立 ST 表，求出 `[in[u],in[v]]` 中深度最小的节点，其即为我们要求的 `lca(u,v)`。

后两个问题

有了 lca 以后，后两个问题我们也能解决了。但是要转换一下。
我们把问题特殊化一下，加入我们只修改或询问从根节点开始的一条链，该怎么做？
对于单点修改，链查询，我们单点修改时，修改整个子树，查询时查询单点。
对于链修改，单点查询，我们链修改时修改单点，查询时查询子树。
让后如果问题满足“相减性”，我们可以用关于 `u`，`v`，`lca(u,v)`，`fa[lca(u,v)]` 这几个点到根节点的链来拼凑出 `u,v` 之间的链。