浅谈一类树上路径相关问题

杭州学军中学 周镇东

本文对一类树上路径相关问题做了总结，并罗列了许多相关例题。

在前言中，笔者大致描述了这类问题的特征。

在第二节中，介绍了解决此类问题常用的三种树上算法，以及这些算法起到的部分作用。

在第三节中，笔者给出了一道例题，并依次采用了第二节中提到的三种算法解决了这个问题。

在第四节中，笔者罗列了近年来的 NOI、WC、CTSC 等重要比赛的许多真题，并应用第二节提到的算法解决问题。

1 前言

近年来，一类树上路径相关的问题经常出现在各大比赛中，这类问题常具有以下特点：

- 给定一棵或多棵树
- 可能给定一些点对或一些路径
- 以某种给定的方式，求树上某些路径或点对的信息最值或求和

解决这类问题也存在一些常用的算法，例如树分治、虚树、线段树合并。

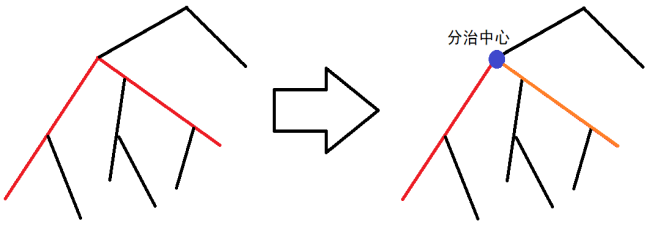
在本文中，将结合具体例题，总结出处理这些问题的常用方法。

如果没有特殊说明，在接下来的问题中，我们总是认为树的规模和询问次数（如果存在多次询问）的大小为 $O(n)$。

2 经典树上算法与其效果

2.1 树分治

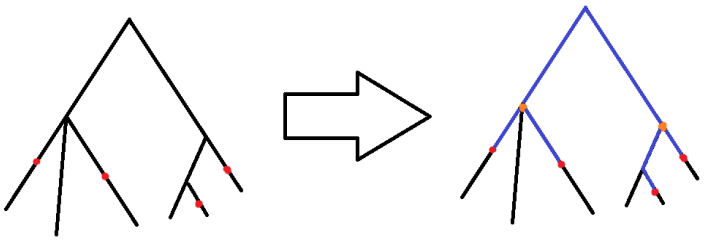
对于一棵树，以及树上的一些链，如果我们要计算的信息是可合并的，那么我们可以通过点分治或边分治将任意链信息拆分成两个从某个分治中心出发的前缀链信息。



因为这么做的代价往往只是在单次处理的时间复杂度的基础上乘上 $O(\log n)$，所以对于很多问题，我们都可以先考虑通过树分治尝试将问题简化，得出 $O(n \cdot poly(\log n))$ 的算法之后再考虑优化。

2.2 虚树

对于一个点集，保留原树上所有两侧均有该点集中的点的边后，删除孤立点，不断收缩所有二度点之后得到的树结构即虚树。因为该点集对应的虚树大小不超过点集大小的两倍，并包含了该点集中所有点，且有树结构，所以对于零散的点集，我们往往可以对其建虚树后再应用一般的树上算法。



有些时候，点集并不是直接给出的。例如，下面列举了两种虚树的使用方法：

- 有两棵树，对第一棵树做树分治，每次得到的点集对应到第二棵树上建虚树。

- 有一棵树及 $O(n)$ 条给定的路径，枚举路径最近公共祖先，并对同一最近公共祖先的所有路径端点建虚树。

在后面的例题中，这两种方法都会被用到。

2.3 线段树合并

对于一棵规模为 $O(n)$ 的树，若每个点初始有一个键值和一些对应的信息。若我们要不断去除某个叶子，并将该叶子的所有信息在不改变键值集的情况下以某种方式合并至父亲，并获取键值在某一个范围内的元素的信息或本次发生合并的点对信息，就可以使用线段树合并的方式实现。

如果按照树上深度优先遍历退出节点的顺序进行合并，线段树合并还能起到固定最近公共祖先的作用。这意味着，当我们依次把某个节点的所有子树信息合并到这个节点上时，每次合并时两个集合之间的点对的最近公共祖先都是这个节点，而线段树合并时还可以得出合并的两个集合中的点对的某些信息，这使得在某些情况下，我们可以把树上路径拆成三个点——两个端点、最近公共祖先来考虑。

若合并信息的复杂度为 $O(1)$，线段树合并的总时间复杂度一般仅为 $O(n\log n)$，而且应用灵活、容易实现，是非常实用的一种工具。

例如，在 NOI2020 中，“命运”[1] 这一题就考察了线段树合并的应用。

3 典型例题

Maximum and Minimum [2]

3.1 题目大意

对于一个有边权无向连通图 G 和两个节点 x,y，我们定义 $f(G,x,y)$ 为图 G 中点 x 与点 y 间所有路径的权重的最小值。其中，一条路径的权值为路径上所有边权的最大值。

给定两个无向图 G_1,G_2，求 $\sum_{i=1}^{n}\sum_{j=i+1}^{n} f(G_1,i,j)f(G_2,i,j)$。

3.2 解题方法

首先，根据 f 函数的定义，我们可以只保留 G_1,G_2 最小生成树上的边，于是问题转化到了两棵树上。

[1] 供参考的题面链接：https://uoj.ac/problem/559

[2] https://www.codechef.com/problems/MXMN

3.2.1 树分治

容易发现，当 G 为一棵树时，$f(G,x,y)$ 就是求 x 至 y 的路径边权最大值，最大值是可合并的信息，所以我们可以先用边分治尝试一下。

对第一棵树进行边分治后，在第一棵树上，得到了由分治中心切割分治区域得出的两部分点集，我们尝试在这一次划分的过程中处理完所有横跨这两个点集的点对。

我们设分治中心到点 x 的前缀边权 max 为 w_x，那么横跨两个点集的点对 (x,y) 给答案的贡献即可被改写为 $\max(w_x,w_y)f(G_2,x,y)$。

3.2.2 虚树

我们将点集取出，并在第二棵树上建出虚树。

接下来考虑按照边权从小到大枚举虚树上的边，每次处理在第二棵树上所有以当前边为最大值的点对，即，按照权值顺序依次加入边，每次将边所在两侧集合合并。

按照加边顺序，我们可以得到一棵 Kruskal 重构树。

3.2.3 线段树合并

现在，有一个树形结构——Kruskal 重构树，且每个点有一个额外的权值——点 x 的权值为 w_x。由于每次合并时的 $f(G_2,x,y)$ 被固定了，我们只需要对于所有满足以下条件的点对 (x,y)，求出 $\max(w_x,w_y)$ 之和即可：

- x 和 y 处于分治中心两侧
- x 和 y 在本次合并之前处于不同集合

于是，对于每个集合，我们可以以权值为下标，维护区间元素个数和权值和。这样，我们在合并两个线段树的对应节点时，因为权值较大的处于右侧子树，所以只需要把左侧的元素个数乘上右侧权值和加入贡献即可。

3.2.4 时间复杂度

对于一个大小为 s 的点集的单次求解的时间复杂度为 $O(s\log s)$，由于我们在这以外还使用了边分治，所以总时间复杂度为 $O(n\log^2 n)$。[3]

[3]本题有树上启发式合并解法，但是与本文主题无关，故不作赘述

3.3 拓展

3.3.1 问题一

- 给定两棵树 T_1, T_2，定义 $g(T, x, y)$ 表示路径边权最大值与路径边权积的乘积。求 $\sum_{i=1}^{n}\sum_{j=1}^{n} g(T_1, i, j)f(T_2, i, j)$。

解法显而易见。在边分治后，设点 x 到边分中心的边权乘积为 s_x，原问题线段树维护了以节点的 w 值为下标的区间 w 值之和，现将其改为维护以节点的 w 值为下标的区间 $w \cdot s$ 值之和即可。

3.3.2 问题二

- 给定两棵树 T_1, T_2，求 $\sum_{i=1}^{n}\sum_{j=1}^{n} g(T_1, i, j)g(T_2, i, j)$。

由于第二棵树的路径边权乘积也被计入，所以 Kruskal 重构树和线段树合并的方式不再适用。

我们可以再次在虚树上使用边分治，于是该问题可以被化简成一个简单的可以使用线段树等数据结构解决的问题。

总时间复杂度为 $O(n \log^3 n)$。

虽然这个时间复杂度不如拓展问题一解法的时间复杂度，但是我们至少得出了一个 $O(n \cdot poly(\log n))$ 的算法。

3.4 小结

在这个问题中，我们多次运用了边分治化简题目，每一次边分治，都将一棵树以及边权信息转化成了每个点的一个权值。注意到，在原问题中，边分治后问题转化成一棵树和一些权值；在拓展问题二中，两次边分治之后，两棵树被转化成了每个点的若干个权值。

这说明，我们往往可以通过一次或多次点分治来很好的达到化简题目的效果。

4 更多应用

4.1 例 1

通道[4]

[4]来自 WC2018，参考题面链接：https://uoj.ac/problem/347

4.1.1 题目大意

给定三棵有非负边权的树，定义一棵树上两点距离为两点间最短路径边权和。问一对点在三棵树上的距离和的最大值为多少。

4.1.2 算法一

首先考虑在第一棵树上边分治，对题目进行简化。在第二棵树上建出虚树之后再次边分治简化题目。然后在第三棵树上再建虚树、再次边分治。

于是每个点获得了一个权值，代表它到三个分治中心的距离之和，而且我们知道每个点在三棵树上边分治时在哪一侧。

我们枚举其中一个点分别位于三次分治的哪一侧，另一个点得分别位于三次分治的另一侧，这种情况下的最优解即两边可能的点的权值最大值的乘积。

至此，我们通过不断边分治，非常快速且容易地得到了一个时间复杂度为 $O(n\log^3 n)$ 的算法。

4.1.3 算法二

考虑不做第三次分治，并对第三棵树做一些修改。

具体地，假设点 x 在前两棵树中与分治中心的距离之和为 d_x，那么在第三棵树上，对于每个点 x，我们新建一个点 x'，并在 x 与 x' 之间连一条权值为 d_x 的边。

于是问题被转化成求“所有在前两棵树中均处于分治中心两侧的点对”在第三棵树上的最远距离。

因为两个点集的最远点对之一的两端点一定分别处于这两个点集各自的最远点对中，所以我们可以轻松解决这部分求最远距离的问题。

因此，时间复杂度被降到了 $O(n\log^2 n)$。

4.1.4 算法三

考虑在算法二的基础上，不对第二棵树进行树分治。

在对第一棵树的分治过程中，我们将点集内的点分成了两种，这两种点分别位于分治中心两侧。

设 $d_k(x)$ 表示节点 x 在第 k 棵树中的深度，设 $dis_k(x,y)$ 表示点对 (x,y) 在第 k 棵树上的距离，设 $D(x)$ 表示节点 x 在第一棵树中与当前分治中心的距离。

设点对 (x,y) 在第二棵树上的最近公共祖先为 p，则 x 与 y 在三棵树中的距离之和可以写为 $D(x)+D(y)+d_2(x)+d_2(y)-2d_2(p)+dis_3(x,y)$，稍加整理得到

$$(D(x)+d_2(x))+(D(y)+d_2(y))+dis_3(x,y)-2d_2(p),$$

也就是说，如果固定了 p，剩下的问题仍然可以转化为在第三棵树上的最远点对问题。

注意到点集的最远点对是支持快速合并的，也就是说我们只需要在第二棵树中进行一次深度优先遍历，并不断将子树的点集向上合并，维护点集在第三棵树上的最远点对，计算合并时产生的新的不同种类点之间的最远点对。以这种方式合并时得到的新点对在第二棵树上的最近公共祖先是已知的，所以我们可以根据上式更新答案。

由于单次求点对距离的时间复杂度可以通过预处理降至 $O(1)$，且虚树复杂度瓶颈——按照 dfs 序排序部分可以在边分治处理子问题结束之后由子问题的结果向上使用归并的方式快速得到，所以我们得到了一个时间复杂度为 $O(n\log n)$ 的算法。

4.1.5 小结

在解决这个问题的过程中，我们先通过多次边分治的方式，非常容易地得到了 $O(n\cdot poly(\log n))$ 的算法，在此基础上，再逐步优化，最终得到了优秀的解法。

这再次告诉我们遇到这类问题时，先尝试树分治往往能轻松地简化题目，并具有较强的启发意义。

4.2 例题 2

暴力写挂[5]

4.2.1 题目大意

给定两棵 n 个节点、边有权的树，定义 $depth_k(x)$ 为第 k 棵树上点 1 到点 x 的距离，定义 $LCA_k(x,y)$ 表示第 k 棵树上点 x 和点 y 的最近公共祖先，定义点对 (x,y) 间的“距离”为

$$depth_1(x)+depth_1(y)-depth_1(LCA_1(x,y))-depth_2(LCA_2(x,y))$$

求“距离”值最大的点对的“距离”值。

4.2.2 简单转化

定义 $dis_k(x,y)$ 表示点对 (x,y) 在第 k 棵树上的距离。

则原“距离”式可以转化为

$$\frac{1}{2}dis_1(x,y)+\frac{1}{2}depth_1(x)+\frac{1}{2}depth_1(y)-depth_2(LCA_2(x,y))$$

如果边权均非负，那么最远点对信息可以合并，我们可以采用类似于通道这一题中的方法来解决。

[5]来自 CSTC2018，参考题面链接：https://uoj.ac/problem/400

但是这里边权可能为负数。

接下来，这题有两个思路：

- 根据之前的经验，我们先对第一棵树进行边分治。
- 在第二棵树上，最近公共祖先似乎占据了重要的位置，这让我们不禁想起了线段树合并。

4.2.3 算法一

首先，我们对第一棵树进行边分治，假设第一棵树上节点 x 到当前分治中心的距离为 $D(x)$，则原式可以进一步转化为

$$\frac{1}{2}(depth_1(x)+D(x)+depth_1(y)+D(y))-depth_2(LCA_2(x,y))$$

直接在第二棵树上建出虚树，然后在树上 dfs，并记录子树内节点 $depth_1(x)+D(x)$ 的最大值，在合并子树的时候更新答案即可。

时间复杂度瓶颈在于建虚树前的按照 dfs 序排序这一部分，而这一部分可以用在通道一题中使用的归并方式快速解决。

于是我们轻松地得到了一个 $O(n\log n)$ 的算法。

4.2.4 算法二

我们考虑在第二棵树上进行深度优先遍历，并不断合并子树信息。

注意到，边分树也是一个深度为 $O(\log n)$ 的二叉树，和线段树非常相似。如果把边分树看做线段树，我们也可以在上面进行类似的合并操作——我们对第一棵树建立边分树，并记录每个分治中心到对应范围内每个点的距离。由于单次合并前第二棵树上的最近公共祖先已经确定，我们只需要在边分树结构里记录每一个边分中心每一侧节点的 $depth_1(x)+D(x)$ 的最大值，在合并的时候更新答案即可。

时间复杂度仍然是 $O(n\log n)$。

4.3 例 3

情报中心[6]

[6]来自 NOI2018，参考题面链接：https://uoj.ac/problem/397

4.3.1 题目大意

给定一棵 n 个节点的树，边有权。

给定 m 条路径，每条路径有一个权重。

对于所有至少有一条边相交的路径对，求它们路径并所有边权之和减去这两条路径各自的权值得到的结果的最大值。

或者判定没有合法的路径对。

边权和路径权值非负。

4.3.2 对边分治效果的分析

由于边分治的过程中，“有交路径对”这个关键限制并没有得到化简，故这题中不适合在第一步进行边分治。

4.3.3 算法一

我们称一条路径的 LCA 为该路径两端点的最近公共祖先。

按照路径相交的方式，我们将相交路径对分成以下两类讨论：

- 两条路径的 LCA 不同
- 两条路径的 LCA 相同

设第 i 条路径的两端点为 x_i, y_i，其 LCA 为 p_i，权值为 v_i。

设点 x 和点 y 的最近公共祖先为 $LCA(x, y)$。

设原树上两点 (x, y) 间距离为 $dis(x, y)$，一个点 x 的深度为 d_x。

考虑第 i 条路径和第 j 条路径，如果 $p_i \neq p_j$，则两条路径的交必然是一条其中一端点是另一点祖先的链，我们不妨设 x_i, x_j 均位于这条链底端节点的子树中，则这两条路径的结果为

$$(dis(x_i, y_i) - v_i) + (dis(x_j, y_j) - v_j) - d_{LCA(x_i,x_j)} + \max(d_{p_i}, d_{p_j})$$

若我们把 $(dis(x_i, y_i) - v_i)$ 看做路径 i 的新权值，把 d_{p_i} 看做路径 i 的键值，再设法固定 $LCA(x_i, x_j)$，就可以发现线段树合并能解决这个问题。

如果 $p_i = p_j$，我们不妨先枚举 LCA，并对同一 LCA 的所有路径端点建虚树，将问题等价转化成了 $p_i = p_j = 1$ 的版本。

由于这两条路径有交，所以节点 1 必然存在一个子树，满足这两条路径恰好各有一个端点在这个子树内，且其他的端点都在这个子树外，我们不妨设在这个子树内的点分别为 x_i, x_j。

则这两条路径的权值为

$$\frac{1}{2}((dis(x_i,y_i)-2v_i+d_{x_i})+(dis(x_j,y_j)-2v_j+d_{x_j})-2d_{LCA(x_i,x_j)}+dis(y_i,y_j))$$

我们可以进行树形 DP，一侧用于固定 $LCA(x_i,x_j)$，另一侧维护与 y_i,y_j 相关的最远点对。

至此，这两部分都已经解决，时间复杂度为 $O(n+m\log n)$。

4.3.4 小结

在第一种情况中，我们发现了线段树合并的经典形式。

在第二种情况中，我们采用了枚举 LCA 并建虚树的方式有效地化简了题目。这种方法非常常用，例如对于 ZJOI2019 中的语言[7] 一题，也可以使用类似的思路。

5 总结

通过上述例题，可以体会到树分治和虚树在简化树上问题时起到的巨大作用，而线段树合并往往解决了问题的最后一步。

在处理这些问题时，我们往往可以先尝试使用树分治或者虚树简化问题，然后尝试将问题转化成线段树合并或者其他算法可以解决的问题。

这些树上算法运用灵活，效果显著，可以处理很多问题。在本文中，笔者挖开了这类问题的冰山一角，希望借此帮助大家熟练掌握此类问题，也希望有更多更加巧妙的相关的问题出现。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队高闻远教练的指导。

感谢徐先友老师对我的培养与指导。

感谢周航锐同学为本文审稿。

感谢父母对我的理解与支持。

参考文献

[1] 张哲宇. 浅谈树上分治算法. IOI2019 中国国家候选队论文集, 2019

[7] https://uoj.ac/problem/470

[2] 陈俊锟, 吕欣. 《通道》试题讲评. 2018

[3] 陈俊锟, 吕欣. 《暴力写挂》试题讲评. 2018

[4] 陈俊锟, 吕欣, 杨景钦, 王逸松. 《情报中心》试题讲评. 2018