# XJOI进阶讲义（六）

## 树上分治算法

Mr_Spade

2021.10

# 概述

树上分治算法包括了点分治/边分治，dsu on tree等常见算法，也是NOI系列赛事的高频考点（WC2018、CTSC2018和NOI2018就各有一道树上分治题，可谓出现频率非常高，不过其中有一些特殊的原因）。作为常用知识点，这些算法的基本用法大家一定已经非常熟悉。

不过，这些算法的一些通用的扩展和技巧、不同树上分治算法的适用范围（某些题能够用哪些算法、不能用哪些算法，背后的原因又是什么）以及应该如何在高难度题目中应用这些算法则不一定为大家所熟悉。本讲就以这三个方面为重点。

# Problem 0

给定一棵 $n$ 个节点的树，初始时刻每个点单独形成一个集合。你要支持一个操作：

选定两个集合$S,T$，求出$S$和$T$的点对中，最远点对的距离，再将$S,T$合并为一个集合。

$n \leq 10^6$，边有非负边权

# Problem 0

这一题是为了引出一个经典结论：若边权非负，令$diam(S,T)$表示$S$和$T$的点对中，最远点对组成的集合，那么：

$$diam(S,T) \subseteq diam(S,S) \cup diam(T,T)$$

证明需要讨论，见手写部分。

如果我们对每个现有集合，维护其最远点对，那么通过预处理lca，回答询问是$O(1)$的。我们还需要考虑维护$diam(S \cup T, S \cup T)$。考虑$S \cup T$的点对要么为$S$的点对，要么为$T$的点对，要么跨越$S$和$T$，于是我们仍然可以$O(1)$维护。

于是瓶颈在于预处理lca，复杂度$O(n\log n)$。

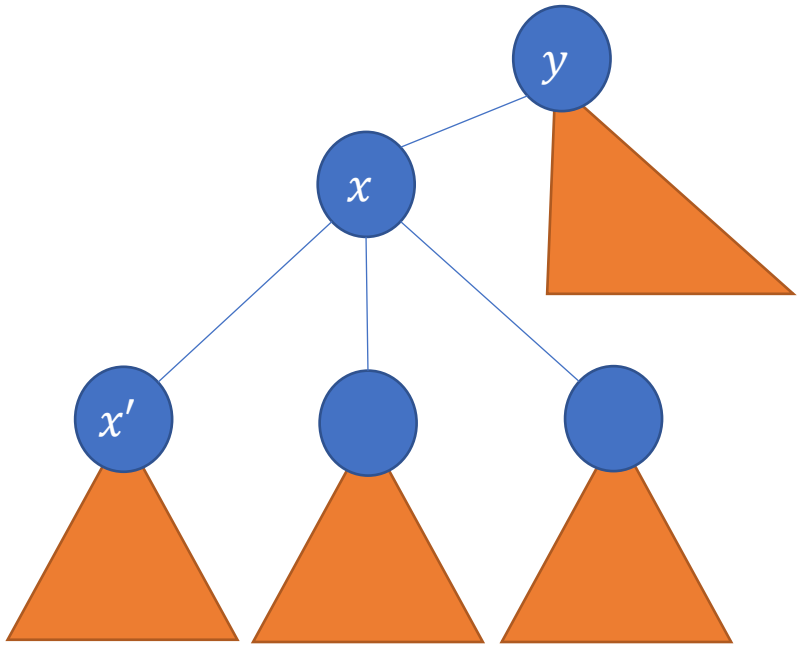
# 点分治（一般）

树的重心的所有子树大小不超过总大小的一半，利用这一点可以进行分治。

大家已经非常熟悉，就不展开讲了。

# 边分治（一般）

在节点度数均不超过$d$时，存在一条中心边将树分为两个不超过$\frac{d-1}{d}n$的子树，利用这一点可以进行分治。




证明：随意选取一条边$(x,y)$，假设$x$一侧的子树大小超过$\frac{d-1}{d}n$。则由于$x$的子树的大小之和$\geq \frac{d-1}{d}n$，且至多有$d-1$个子树，于是最大的那一棵（假设根为$x'$）的大小$\geq \frac{1}{d}n$。选取边$(x,x')$，则$x$那一侧的子树大小不超过$\frac{d-1}{d}n$。即使$x'$一侧的子树大小超过$\frac{d-1}{d}n$，也必然是单调下降的。反复这个过程总能得到中心边。

# 边分治（一般）

当度数的上界可能很大时，通过添加虚点虚边（最经典的是“左孩子右兄弟”的树转二叉树添加法），可以将原树$T$改造为一棵度数$\leq 3$的树$T'$，并且满足：对于$T$中的点$x,y$，$T'$上$x$到$y$的路径所包含的边就是$T$上$x$到$y$的路径所包含的边加上若干条虚边。因此，对于多数统计边信息的题目，这样改造树以后即可利用边分治。

然而，如果考虑$T'$上$x$到$y$的路径所包含的点，就会发现常用的改造方法往往会丢失lca。有时，这一点可以通过再增加一些点解决（但非常繁琐），有时这一点则是致命的。因此，对于统计点信息的题目，不建议使用边分治解决（下面会有具体的例子）。

# Problem 1

给定两棵树，求一对点使得它们在两棵树上的距离之和最大。

$n \leq 10^6$，边有非负边权

# Problem 1

距离只和边的信息有关，我们可以尝试改造树然后使用边分治。

边分治第一棵树，对于一条中心边，跨过它的两点距离可以表示为$d_x + d_y + w$，其中$d_i$表示$i$到对应子树的根的距离，$w$表示中心边的长度。

从而跨过中心边的点对的两树距离可表为$d_x + d_y + w + dis_2(x, y)$，只需要知道$d_x + d_y + dis_2(x, y)$的最大值。

这类似于求最远点对，但带上了两个点权。不过容易发现，只要点权非负，我们可以将现在的点想象为“与实际点连了一条长度为$d_i$的边的新点”，从而Problem 0的结论依然成立。（事实上，还可以证明即使点权能够为负，该性质仍成立，见手写部分）

# Problem 1

假设中心边两端的点集为$S,T$，根据之前的结论，可以分别用合并求出$S$和$T$内部的最远点对，再$O(1)$求出跨越$S$和$T$的最远点对。

由于有$O(n\log n)$次合并，故此时使用预处理lca可以降低复杂度为$O(n\log n)$。

# Problem 2

给定两棵树，求一对点使得它们在两棵树上的距离之和最大。

$n \leq 10^6$，边有可以为负的边权

# Problem 2

边权为负，无法再运用之前的结论。但第一层的边分治依然是有效的，现在仍然要求解$d_x + d_y + dis_2(x, y)$的最大值。

由于点集是静态的，可以建立$S \cup T$的虚树，直接在虚树上进行树形dp，就能求出最远点对。

在边权非负的情况下不采用这个方法的原因是建立虚树比较麻烦。

同一层内，求解lca的复杂度可以做到线性。建立虚树需要的对dfs序排序可以用归并解决。于是总复杂度仍为$O(n \log n)$。

# Problem 3 「WC2018」通道

给定三棵树，求一对点使得它们在三棵树上的距离之和最大。

$n \leq 10^5$，边有非负边权

# Problem 3

边权非负但有三棵树，不难想到把前两题的做法进行结合。

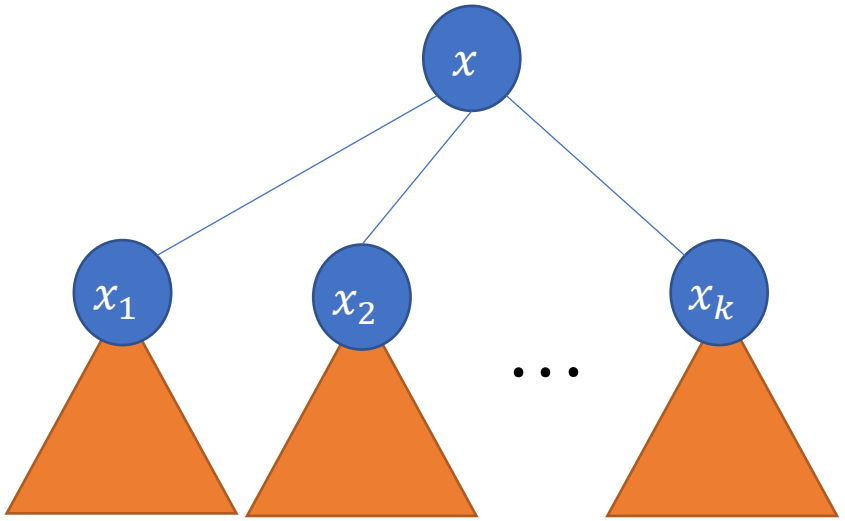
边分治第一棵树，转化为求解$d_x + d_y + dis_2(x,y) + dis_3(x,y)$的最大值。对第二棵树建立$S \cup T$的虚树，每次树形dp的合并过程还需要求解两个点集之间带点权的最远点对。

使用Problem 0的技巧保留最远点对，即可$O(1)$合并。复杂度仍然为$O(n \log n)$。

此外，也可以对第二棵树的虚树继续使用边分治，在第三棵树的虚树上树形dp求最远点对。这样不需要用到边权非负的性质，但复杂度为$O(n\log^2 n)$。

# 点分治（合并优化）

边分治的优势在于每次只会划分为两个部分，因此只需要互相统计贡献。而点分治划分为多个部分，想要统计两两之间的贡献，有时需要使用删除的技巧，有时则难以下手。




$$\sum_{i<j} f(x_i, x_j) = \frac{1}{2}\left[f\left(\sum_i x_i, \sum_i x_i\right) - \sum_i f(x_i, x_i)\right]$$

# 点分治（合并优化）

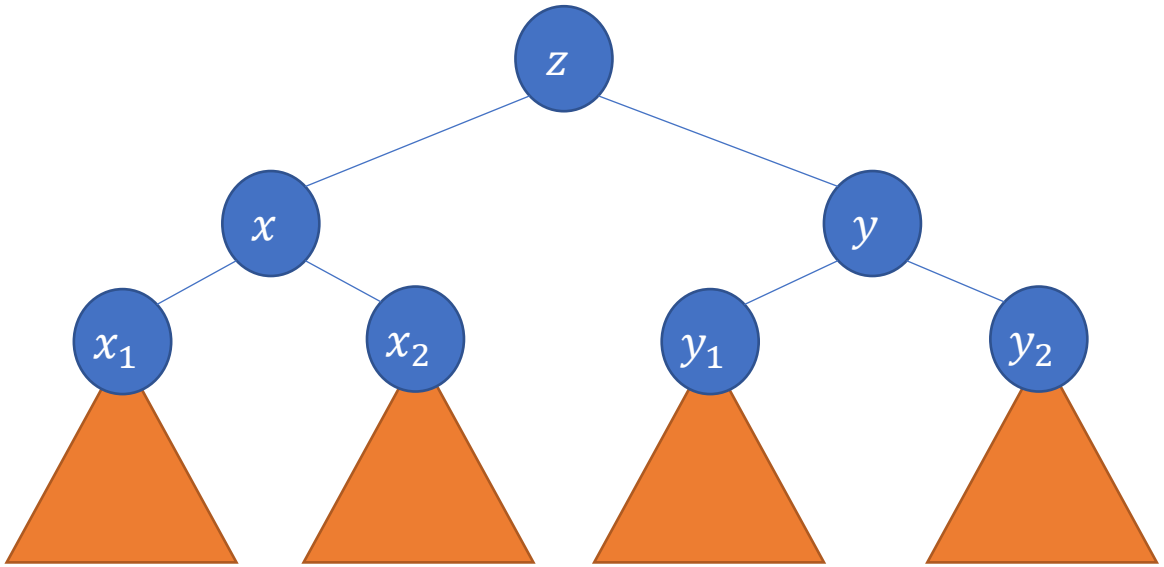
回顾Problem 1，最值的不可减性使得我们无法使用删除的技巧。不过，由于边权非负时合并非常快，可以改为依次枚举所有子树，统计它和已有子树的贡献，再将该子树合并到已有子树中。

不过这对于Problem 2就行不通了：想要用虚树解决问题，就意味着我们只能整体解决，不能动态添加。点分治似乎行不通了。

有什么方法可以使得点分治能和边分治媲美吗？

# 点分治（合并优化）

有一个技巧是：每次统计两棵最小的子树之间的贡献，再将它们合并，不断重复。如果使用了这个技巧，那么有一个重要的结论：每个集合在被合并两次后，大小至少变为两倍。




证明：如图，假设$x$的大小较小，则$x$先于$y$生成。$x_1, x_2$扩张两倍的结论是显然的，对于$y$来说，既然合并的是$y_1, y_2$而不是$x$，说明了$y_1, y_2$的大小不超过$x$，从而扩张两倍的结论也成立。

# 点分治（合并优化）

那么每个点在这一层至多被合并$O(\log n)$次，假设合并的复杂度和两个集合的总点数呈线性，该层复杂度就是$O(n\log n)$，从而总复杂度$O(n\log^2 n)$？比边分治多出一个log?

假设该层总点数为$n$，点分治后划分为点数$a_1, a_2, \cdots, a_k$的部分。那么对任意$a_i$，其至多被合并$\log\frac{n}{a_i}$次。所以，该层的复杂度还可以被估计为：

$$\sum_{i=1}^{k} a_i \log\frac{n}{a_i} = \sum_{i=1}^{k} a_i \log n - \sum_{i=1}^{k} a_i \log a_i = n\log n - \sum_{i=1}^{k} a_i \log a_i$$

则总复杂度显然为$O(n\log n)$。

# 点分治（合并优化）

综上所述，点分治通过合并优化可以做到和边分治一样两两合并的方式，并且复杂度相同。

张哲宇同学曾得到的结论是，点分治的适用范围是边分治的超集。不过实际中边分治思考更为方便，因此不妨先使用边分治思考问题。再根据实际情况选择合适的算法。

由于边分治通常需要重构树，因此合并优化的点分治代码难度实际上相对容易。

# 点分治（合并优化）

现在回顾Problem 2，有了合并优化的技巧，点分治每次也只合并两个点集。于是可以直接类比边分治的做法。

类似地，Problem 3使用点分治当然同样可行。

不过，如果把题目修改一下：距离被定义为路径上的点权和，那么边分治的做法就行不通了——无法统计lca的贡献，解决方法有，但既不通用也很繁琐。然而，点分治就能几乎原封不动解决这个问题。从这里可以看出点分治的优越性。

# Problem 4 「CTSC2018」暴力写挂

给出两棵树$T, T'$，求点对$x, y$使得下面的式子最大化：

$$deep_x + deep_y - deep_{lca(x,y)} - deep'_{lca'(x,y)}$$

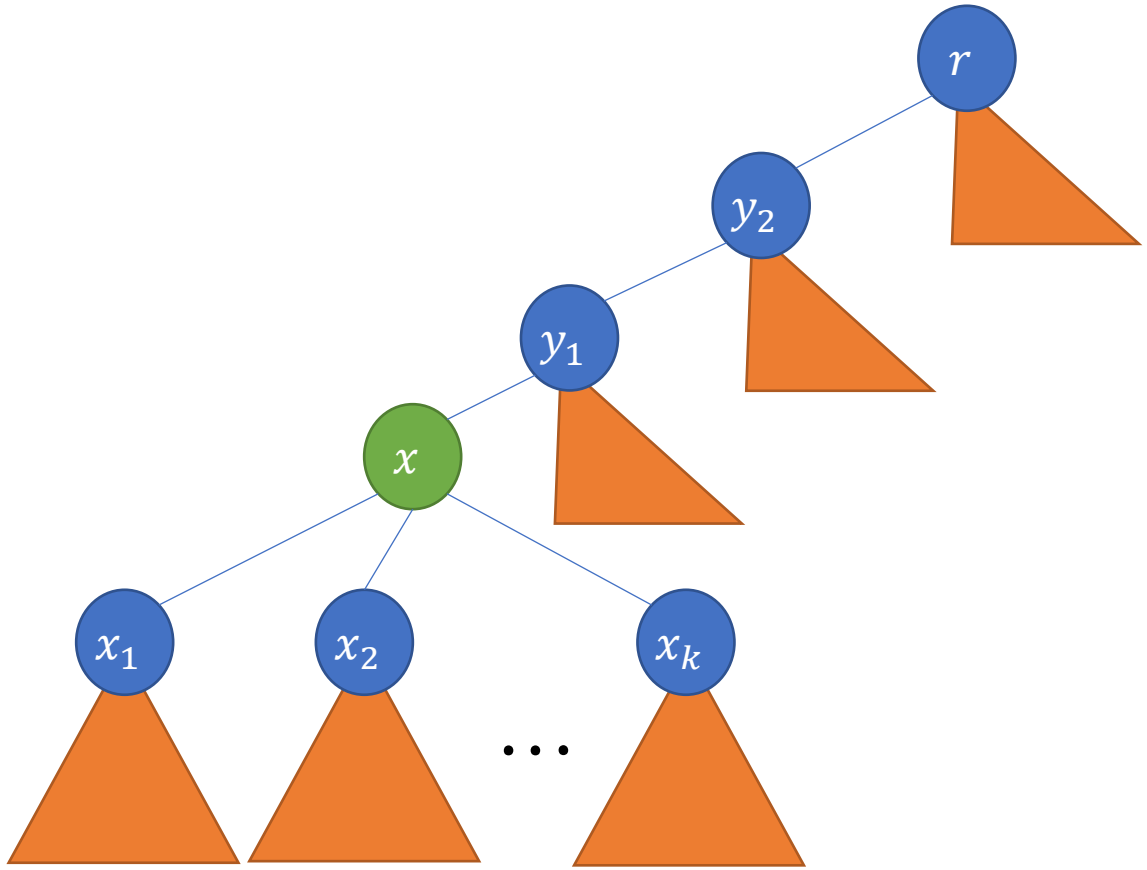
$n \leq 10^5$，边有可以为负的边权

# 有根点分治/边分治

从题目描述可以发现，答案在根不同的情况下是不同的。所以，这一题我们要处理有根树，而之前的点分治/边分治都是针对无根树的。能否将它们应用到有根树上？

# 有根点分治/边分治

以点分治为例，在有根的情况下分治结构如下：




其中$r$表示该连通块的根，$x$是当前的重心。可以发现，除了$r$所在的子树外，其余子树地位是相等的，依然可以正常合并，并且它们有共同的lca是$x$。而$r$所在的子树则还需分类为$y_1, y_2, \ldots, r$几个部分，每个部分和其它子树的lca为对应的根。

# 有根点分治/边分治

不过，将每个部分分别暴力统计贡献复杂度显然是不对的。通常，想要应用有根点分治/边分治，就要结合题目的具体性质，可以利用lca与另一个点无关的性质，将lca的影响隐藏到系数中；或者依赖于题目性质简单，将$r$所在部分进行动态的单点查询。

在有的题目中，有根的关键并不在于lca，这时，可能还有其它处理方法。之后的题目会展示这一点。

# Problem 4

知道了有根点分治/边分治的概念以后，就可以来做这道题了。为简化思考过程，以使用有根边分治为例。

对于跨越了中心边的点对$x, y$，假设$y$处于根所在的连通块，且处在局部根为$y_k$的部分。则题中所要求的量转化为：

$$deep_x + deep_y - deep_{y_k} - deep'_{lca'(x,y)}$$

其中$y_k$与$x$无关，于是可以把这部分贡献直接计算到$y$上。现在所求的量为带点权的第二棵树上的lca深度的最小值。仍然建立虚树进行树形dp。复杂度$O(n \log n)$。

可以看出，动态添加的做法不太可行。

# Problem 5 「THUSC2018」史莱姆之友

给定一棵有根树，每个节点上有一个形如$x - a_i$的多项式。求每个节点到根的路径的多项式乘积的和。

$n \leq 10^5$

# Problem 5

这比较显然是有根树，那么就要用有根点分治。（由于此题只要点到根的路径，因此边分治也是可以的）

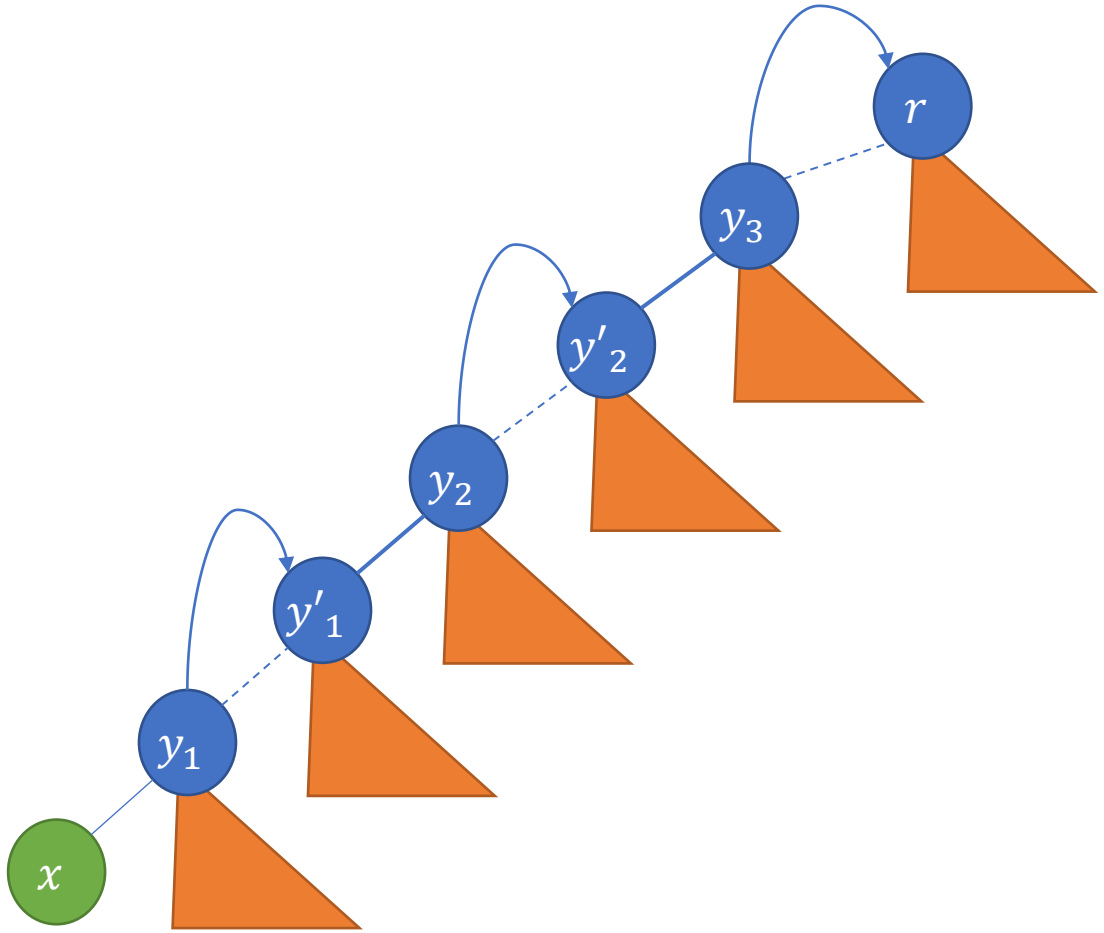
点分治后，先将各个部分递归的原问题解决。对于包含根的子树，我们已经计算了最后的答案。我们假设重心到根的多项式乘积是$f$，那么显然“包含根的子树的答案$+f*(\sum$其它子树的答案$+1)$”就是总的答案。

由于答案的次数不超过子树大小，我们可以先直接将其它子树的答案相加，再加一。

如果使用$O(n\log^2 n)$的分治FFT计算$f$，那么总复杂度就是$O(n\log^3 n)$。

# Problem 5

不过，利用点分治的一个技巧，这一点可以被优化：




如图，$x$是分治重心，$y_1$是$x$的父亲。假设$y_1$在作为重心的时候，连通块的根是某个祖先$y'_1$，那么$y_1$到$y'_1$段的多项式乘积已经被计算过，假设该块内有$l$个点。

而$y'_1$的父亲$y_2$一定也有作为重心时的根$y'_2$，并计算了对应多项式，且由于$y_2$在$y_1$的上层，点数至少加倍，至少有$2l$个点。

如此往复，总能走到$r$，于是除$x - a_x$以外，$f$的剩余部分等于以上多项式的乘积，后一个多项式的次数上界总是前一个的两倍。与倍增类似，计算乘积是$O(n \log n)$的。从而总复杂度优化到了$O(n\log^2 n)$。

# 利用已知信息降低点分治/边分治复杂度

这个技巧一般化地来讲，就是加速计算重心到根的某个信息。由于递归的部分已经分段计算好了这条链的部分的信息，并且段的长度呈倍增关系，有时合并这些信息，会比重新直接计算更快，从而优化了复杂度。

# dsu on tree

比较科学的名字应该是树上启发式合并。

也是一种大家非常熟悉的方法，有时如果是利用线段树等优秀的数据结构维护连通块信息，且贡献适合整体对整体计算，则启发式合并可以被优化为线段树合并。

dsu on tree有两种常见写法，一种是完全启发式合并，另一种是先树链剖分选出重儿子，再枚举轻儿子暴力合并。前一种写法通常简短好写，而后一种写法中启发式合并不会拆分父亲所在的部分，可能在一些题中有着更优良的表现。

# Problem 6

有一棵$n$个点的树和给定的数$k$，每个点有一个权值$a_i$。对每一棵子树，求该子树内的满足$a_i \operatorname{xor} a_j = k$的点对$i, j$的数目。

$n \leq 10^5$

# Problem 6

简单的dsu on tree模板题。用map维护子树内权值的可重集，则对于当前节点，可直接继承重儿子的map，再将自身的权值加入其中。随后暴力枚举所有轻儿子的权值加入其中。每次加入点时，还需要顺带统计能和它配对的点数。

复杂度显然为$O(n\log^2 n)$，可以注意到，这一题就是无法使用线段树合并优化的例子。

# Problem 6

还可以试试有根点分治来解决问题。不妨先对$x$统计所有lca为$x$的有效点对，再求子树和。

一次合并的过程中，除根所在连通块外的连通块显然可以按常规方法合并，并将贡献统计给重心。对于根所在的连通块，根据局部根划分为若干部分并动态统计，将贡献加入对应的局部根。同样可以$O(n\log^2 n)$解决问题。

点分治和dsu on tree能解决的问题似乎有所重合。那么，两种算法能解决的问题是否是相同的呢？或者，一种算法能解决的问题是另一种算法的子集？

某种程度上，正是因为对这些问题了解得不够清晰，我们才在面对树上问题时不能很好分析问题性质，从而找到适合的解法。

# dsu on tree

我们先来试试用dsu on tree解决在点分治中讲过的问题。看看如果不能解决问题，那么原因是什么。

Problem 1是容易的。我们首先将$dis_1(x,y)+dis_2(x,y)$转化为$deep_x+deep_y-2deep_{lca(x,y)}+dis_2(x,y)$。每当进行启发式合并时，lca总是确定的，于是剩下的问题依旧转化为带点权的树上最远点对问题。使用Problem 0的结论即可$O(1)$合并，甚至不需要启发式。复杂度瓶颈在求lca，为$O(n\log n)$。

Problem 2，我们不再能使用Problem 0的结论。我们尝试回到保留所有的点，使用启发式合并，即动态添加每一个轻儿子子树的点，在维护的某点集中查找最远点对。

# dsu on tree

维护一个点集并支持查询到某个点的最远点，这是一个不太可做的问题。于是我们的尝试就只能到此为止了。

回顾一下我们是怎么用点分治解决这个问题的。在点分治中，我们无需担心“因为遍历了重儿子的点导致复杂度错误”的问题，遍历每个点依然是可以接受的。这一限制的放松使得我们能将整体看作一个静态的问题，用虚树上的树形dp解决它。而启发式合并中无法这样遍历重儿子的点，只能以轻儿子的角度统计贡献，从而只能使用动态加入并查询的方式。那么，对于动态远远比静态复杂的查询，dsu on tree自然就遇到了巨大的阻碍。

基于同样的理由，启发式合并无法解决Problem 3，Problem 4。

# dsu on tree

Problem 5如何？

我们看看暴力枚举轻儿子这一步。假设已经得到了当前的根在只有重子树情况下的答案以及每个轻子树的答案。由于轻子树的答案次数不超过轻子树的大小，我们可以支持暴力将其乘以根的一次多项式，再加入根的答案中。这是不是意味着dsu on tree可以做这道题？

似乎有哪里不对。我们甚至没有用FFT？

# dsu on tree

重新思考一下，我们不仅需要保证可以只枚举轻儿子，还需要保证重儿子的子树可以快速加上一个根。

而这个问题中，加一个根意味着一次多项式乘法，是无法$O(1)$计算的，因此在复杂度上我们还是加上了重儿子的子树大小，从而无法用dsu on tree解决这个问题。

那么，dsu on tree能解决的问题是不是点分治的子集呢？

# Problem 7 「清华集训2016」Alice和Bob又在玩游戏

Alice和Bob在有根树森林上玩游戏。轮到一个人时，ta要选择一个未被选中过的点，将该点到根的路径上的点都变为选中过的。如果所有点都被选中过，那么ta就输了。Alice先手，请问她有无必胜策略？

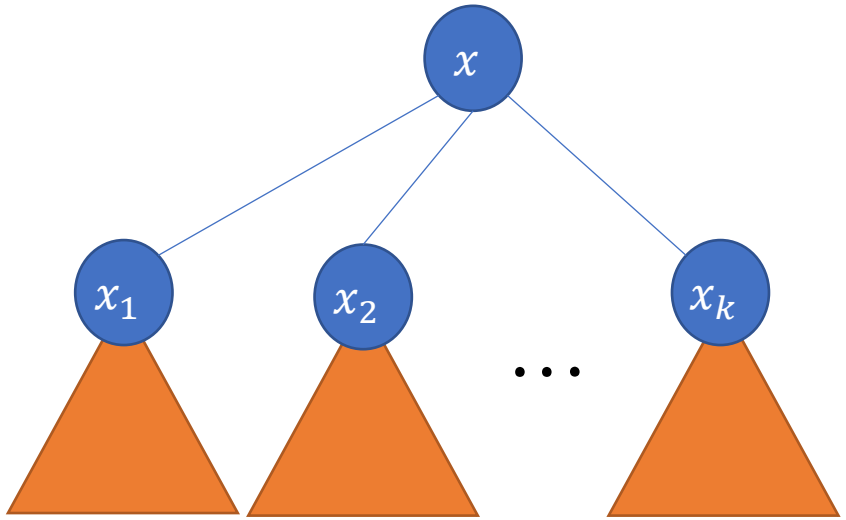
$n \leq 10^5$，$n$是森林的总点数。

# Problem 7

这是一个多局共同进行的游戏。在这种情形下，使用SG函数一般会方便许多。这样，我们就只需要独立计算每棵树的SG值。

一棵树的SG值，就是删除每个点到根的路径上的点后剩下的森林的SG值的$mex$。那么可以考虑能否计算删除每个点后局面的SG值。

# Problem 7

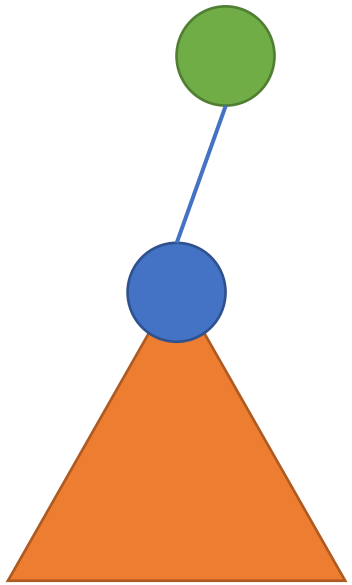



如果删除点$x$，答案就是所有子树的SG值的异或和，这提示我们对每棵子树求解答案。

如果删除的不是$x$，不妨设删除$x_1$子树内的点$z$，那SG值就是在$x_1$子树内删除$z$得到的SG值，再异或上$x_2, \ldots, x_k$的SG值。

# Problem 7

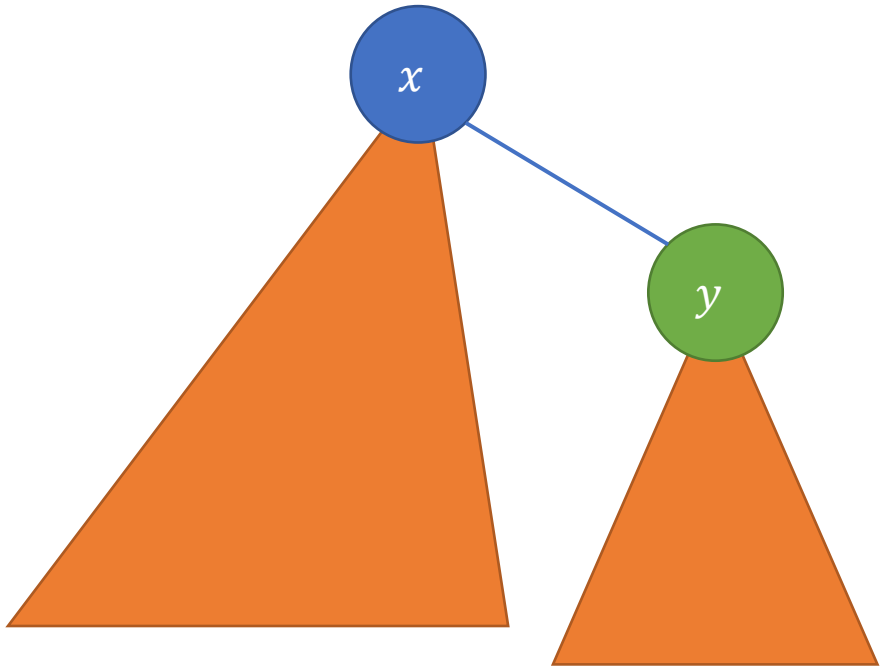
这提示我们尝试用启发式合并对每个点求出“以该点为根的子树删除子树每个点后的SG值”。



首先是第一步，如何在重儿子的子树上加上根。删去重儿子子树内的点，由于根也被删除因此SG值不变。删去根，则答案就是整棵重子树的SG值。

当然，我们还需要维护这之后整棵树的SG值。如果利用线段树/trie树存储“以该点为根的子树删除子树每个点后的SG值”，那么$mex$是很好求的。

# Problem 7



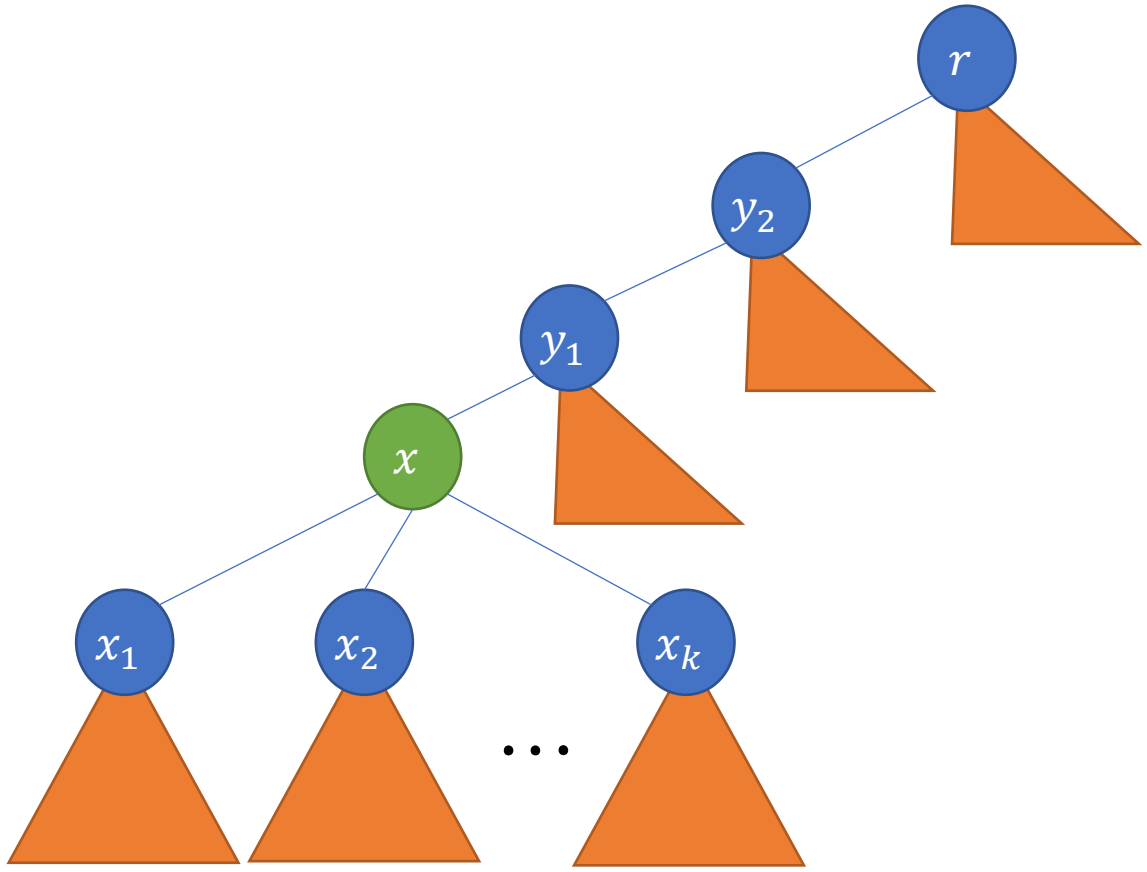

再来考虑将轻儿子合并。如果删除$x$子树内的点，那么SG值在原本的基础上异或$y$的SG值。为了支持整体异或的操作，用trie树维护SG值。

如果删除$y$子树内的点，那么SG值在原本的基础上异或“在$x$的子树重删除$x$之后的SG值”，同样用trie树打异或标记。

之后，进行trie树合并或启发式合并就可以解决问题了。

# Problem 7

点分治能否做这道题？




显然，我们需要用有根点分治。不过此时，在确定重心$x$以后，先对各个橙色子树求解同样的问题（即上方的子树拆为多个部分递归求解）。

首先，我们可以仿照启发式合并的做法，在$x_1$的基础上添加$x$作为新根。此后，照搬启发式合并的方法加入$x_2, x_3, \dots, x_k$，那么$x$子树的部分就算完成。

之后，我们顺次枚举$y_1, y_2, \dots, r$，将其橙色子树（包括自身）与$x$所在的部分合并。通过类比启发式合并的做法，依然可以只枚举$y_i$橙色子树内的点，就将两个点集合并，并更新$y_i$的信息；如果是利用线段树合并，显然也可以同样操作。完成后，我们就更新了该部分所有点的信息。并且，该部分所有的点在这一层只被枚举恰好一次（如果是用启发式合并）。而每个点只会出现在$O(\log n)$层中，因此被合并的次数与dsu on tree相同，也就有相同的复杂度。

# 重启点分治

事实上，容易看出上面的方法是通用的。只要dsu on tree对某道题适用，那么必然可以将合并的方法应用到点分治上。

另外还有一个小要点，如果我们要对每个子树统计原问题的答案，那点分后的连通块可能并不包含某一点的所有子树的点。不过，可以证明每个点最多只会作为一次$y_i$，且此时必定能够包含原子树的所有点，证明见手写部分。因此，我们可以放心执行线段树合并而不必担心需要保留历史版本。

# 重启点分治

考虑一下此时点分治大显神通的本质原因是什么。这是因为如果一道题是可以使用dsu on tree做的，就意味着需要求解的信息支持动态的添加查询。在动态添加查询可行之后，我们就可以放心地将根所在的子树根据lca分为若干部分，对每个部分以动态方式添加并计算贡献；而不是如Problem 4一样，需要仔细考虑lca的影响后进行静态统计，从而使得解题能力受到一定的限制。

# 重启点分治

综上所述，在点分治经过一些扩展后，其能解决的问题是边分治、dsu on tree的超集。因此，从解题能力来看，点分治具有绝对的优势。

但这并不意味着另外两种算法没有意义。边分治的思维情况更为简单，因此常可以作为思考的工具；而dsu on tree代码轻量化，方便好写，可以被认为是贡献支持动态添加查询的情况下，点分治的一个精巧替代品（类比树状数组与线段树的关系）。

如果不确定一道特定的题目是否适合使用dsu on tree，不妨直接从点分治的角度考虑问题。在得出正解后，再思考贡献是否支持动态添加查询——就像得到线段树解法后再思考能否改为树状数组一样。

# 作业题

- 「ZJOI2019」语言
- 「清华集训2016」汽水
- Codeforces 741D

对于以上问题，如果使用dsu on tree解决，试试思考如何使用点分治解决；如果使用点分治解决，试试思考能否用dsu on tree解决，如果不能，那么静态统计的贡献为何，导致了转化为动态添加查询时遇到了困难。

# 代码

复杂的树上问题的代码通常比较长，使得题目像是一道码农题。

但既然这样的题目出现在比赛中（而且是近年的比赛），说明这样的代码能力是我们需要掌握的。况且，题目的主要难点依然在于思维。

大家目前的代码能力可能稍有不足，想要写出本讲一些题目的正确代码，可能需要调试很长时间。但希望大家在调试完成后，不要马上放弃此题，而是回想一下究竟在代码的哪个部分遇到了困难。在一段时间后，重新尝试写一写该题，并尽力在比赛时限内完成。这样可以很好锻炼大家写代码时的思维以及对复杂代码的全局掌控能力。