树上数据结构问题的一些技巧

宁波市镇海中学 梁晏成

2018 年 7 月 1 日

Overview

常见转化方法

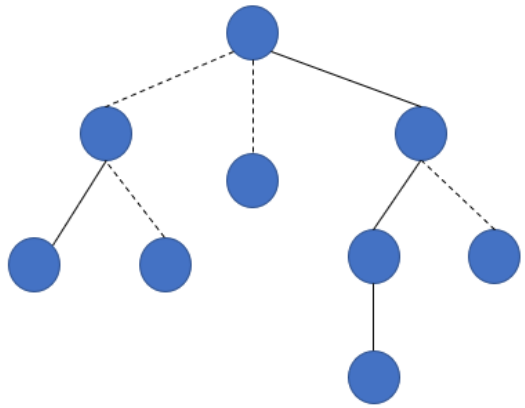
- dfs序：按照dfs时的入栈顺序形成一个序列
- 括号序列：dfs时，某个节点入栈时加入左括号，出栈时加入有括号
- 欧拉序列：dfs时，某个节点入栈时将其加入序列，出栈时将父亲加入序列

树链剖分

一般的树链剖分，指的都是重链剖分

定义一个节点的重儿子为该节点所有儿子中，子树内节点最多的节点。

某个点和重儿子之间的边为重边，和其余儿子的边为轻边。重边首尾相连形成重链。



dfs序与树链剖分

对于任意一个节点，它到根的路径上最多只有$O(\log N)$条轻边，这也意味着只有$O(\log N)$条重链。

在dfs时优先遍历重儿子，那么对于树上的一条链，可以将其对应到dfs序中$O(\log N)$个区间，这样可以将许多树上的问题转化为序列上的问题。

结合dfs序中同一子树内节点连续的性质可以解决更多问题。

NOI2015 软件包管理器

给定一棵$N(N \leq 100000)$的树，要求支持以下操作：

- 统计某个点到根的路径上权值为0的点的个数，然后将路径上所有点的权值变成1
- 统计某个点子树内权值为1的点的个数，然后将子树内所有点的权值变成0

NOI2015 软件包管理器

给定一棵$N(N \leq 100000)$的树，要求支持以下操作：

- 统计某个点到根的路径上权值为0的点的个数，然后将路径上所有点的权值变成1
- 统计某个点子树内权值为1的点的个数，然后将子树内所有点的权值变成0

直接利用树链剖分和dfs序将其转化为区间赋值和区间求和，复杂度$O(N \log^2 N)$

括号序列与树上莫队

如果我们认为一对匹配的括号（即同一个点对应的左括号和右括号）可以互相抵消，那么对于一对点x, y 满足$L_x \leq L_y$，考虑：

- 如果x是y的祖先，那么x到y的链与括号序列中区间$[L_x, L_y]$对应
- 如果x不是y的祖先，那么x到y的链除lca的部分与括号序列中区间$[R_x, L_y]$对应

可以用括号序列将树上莫队转化为普通莫队。

欧拉序列与lca

在欧拉序列中，观察点x与点y之间深度最小的点，即这两个点的lca

可以将lca问题转化为rmq问题

差分思想

对于一对点x,y，设它们的lca为z，那么x到y的链可以由x,y,z,fa[z]到根的链表示

如果两条相同的边上的信息可以抵消（如链上所有边异或的值），那么可以直接由x,y到根的链表示

链上第K大

链上第K大

对每个节点，维护一棵可持久化权值线段树，每次在父亲的线段树的基础上加入这个节点的权值。

询问时利用x,y,lca,fa[lca]的可持久化权值线段树，在线段树上二分。

时间复杂度$O((N+M)\log N)$

单点、链、子树的转化

在某些情况下，我们需要改变修改和查询的对象来减小维护的难度

- 单点修改链上查询⇔子树修改单点查询
- 单点修改子树查询⇔链上修改单点查询
- 链上修改子树查询⇒单点修改子树查询（利用dep数组）

点-边

一些与“链的相交”有关的问题，我们可以在点上赋正权，边上赋负权的方式简化问题

点-边

一些与“链的相交”有关的问题，我们可以在点上赋正权，边上赋负权的方式简化问题

例如这样一个问题：

- 插入一条链
- 给定一条链，问有多少条链与这条链相交

我们只需要在插入时，给链上的点+1，链上的边-1，询问时就是一个链上求和。

提取关键点

我们可以在一棵树中取不超过$\sqrt{N}$个关键点，保证每个点到最近的祖先的距离$\leq \sqrt{N}$

具体地，我们自底向上标记关键点。如果当前点子树内到它最远的点的距离$\geq \sqrt{N}$，那么就将当前的点标记为关键点。

HDU 6271 Master of Connected Component

- 给定两棵有N个节点的树，每个节点上有一对数(x,y)，表示图G中的一条边
- 对于每一个x，求出两棵树中x到根路径上所有边在图G中构成的子图的连通块个数
- 多组数据，$N \leq 10000$

HDU 6271 Master of Connected Component

可以利用dfs序+莫队转化为$M = N\sqrt{N}$的动态最大生成树问题，常数和代码复杂度都较大。

HDU 6271 Master of Connected Component

可以利用dfs序+莫队转化为$M = N\sqrt{N}$的动态最大生成树问题，常数和代码复杂度都较大。

考虑在第一棵树中提取关键点，将关于x的询问挂到第一棵树中x的最近关键点祖先上。之后对于某一个关键点p统一处理询问。首先将p到根节点的路径上的边都加入并查集，然后维护一个可撤销并查集。

HDU 6271 Master of Connected Component

可以利用dfs序+莫队转化为$M = N\sqrt{N}$的动态最大生成树问题，常数和代码复杂度都较大。

考虑在第一棵树中提取关键点，将关于x的询问挂到第一棵树中x的最近关键点祖先上。之后对于某一个关键点p统一处理询问。首先将p到根节点的路径上的边都加入并查集，然后维护一个可撤销并查集。

上述两个算法的时间复杂度都是$O(N\sqrt{N}\log N)$

bzoj1086 王室联邦

- 给定一棵N个点的树和常数B，将树分块，满足每一块的大小在$[B, 3B]$之间
- 集合S构成一个块当且仅当存在一个点x，满足S中的任意一个点到x的路径只经过$S \cup \{x\}$中的点

bzoj1086 王室联邦

- 给定一棵N个点的树和常数B，将树分块，满足每一块的大小在$[B, 3B]$之间
- 集合S构成一个块当且仅当存在一个点x，满足S中的任意一个点到x的路径只经过$S \cup \{x\}$中的点

直接dfs，将某个点的儿子依次合并，超过B就作为一个新的块

如果完成了对树的分块，常常可以类比序列的分块简化问题

启发式合并

启发式合并即合并两个集合时按照一定顺序（通常是将较小的集合的元素一个个插入较大的集合）合并的一种合并方式，常见的数据结构有并查集、平衡树、堆、字典树等

具体地，如果单次合并的复杂度为$O(B)$，总共有M个信息，那么总的复杂度为$O(BM\log M)$

树的特殊结构，决定了常常可以使用启发式合并优化信息合并的速度

51nod 1743 雪之国度

- 给定一个N个点，M条带权边的图，Q次询问
- 每次询问给出两个点x, y，求最小的权值v，满足：
- 存在两条边不相交的路径，满足两条路径上的边权值都$\leq v$
- $N, Q \leq 100000, M \leq 500000$

51nod 1743 雪之国度

- 给定一个N个点，M条带权边的图，Q次询问
- 每次询问给出两个点x, y，求最小的权值v，满足：
- 存在两条边不相交的路径，满足两条路径上的边权值都$\leq v$
- $N, Q \leq 100000, M \leq 500000$

首先求出最小生成树，然后从小到大枚举非树边，将对应路径上的点缩在一起。将询问挂到对应的两个点上，对每个点维护一个并查集，合并点时通过并查集判断是否有同一询问的两个点被缩到一起

时间复杂度不超过$O(M \log M + (N + Q \log Q) \log N)$

清华集训2016 汽水

- 给定一棵N点带权树，常数k
- 求一条路径使得边权平均值与k的差距最小
- $N \leq 100000$

清华集训2016 汽水

- 给定一棵N点带权树，常数k
- 求一条路径使得边权平均值与k的差距最小
- $N \leq 100000$

二分答案W转化为判断问题。考虑一条边i的权值为v，构造$A_i = v - k + W, B_i = v - k - W$，变为判断是否存在一条路径满足$\sum A \geq 0, \sum B \leq 0$

启发式合并，对每个点以$\sum A$为关键字维护所有以它为根的路径的$\sum B$的最小值。合并时查询$\sum A \geq$某个值的$\sum B$的最小值

使用splay维护的时间复杂度为$O(N \log N \log W)$，注意splay的启发式合并复杂度为$O(N \log N)$

JLOI2015 城池攻占

- 给定一棵N个点的树，每个节点有一个值h_i，两个参数$a_i, v_i \geq 0$
- 有M个骑士，初始攻击力x_i，从某个节点出发向根节点前进
- 每到达一个节点，如果当前攻击力小于该节点h_i值，那么这个骑士牺牲
- 否则，如果$a_i = 0$攻击力加上v_i，如果$a_i = 1$攻击力乘上v_i
- 求每个骑士牺牲的城市

JLOI2015 城池攻占

- 给定一棵N个点的树，每个节点有一个值h_i，两个参数$a_i, v_i \geq 0$
- 有M个骑士，初始攻击力x_i，从某个节点出发向根节点前进
- 每到达一个节点，如果当前攻击力小于该节点h_i值，那么这个骑士牺牲
- 否则，如果$a_i = 0$攻击力加上v_i，如果$a_i = 1$攻击力乘上v_i
- 求每个骑士牺牲的城市

注意到如果两个骑士经过了同一节点，那么攻击力相对大小不会改变；因此对每个点维护一个可并小根堆，打标记维护即可。

时间复杂度$O(M \log^2 M)$

直径的一个性质

令$F(S)$表示集合S中最远的两个点构成的集合，那么对同一棵树中的集合S, T，$F(S \cup T) \subseteq F(S) \cup F(T)$

直径的一个性质

令$F(S)$表示集合S中最远的两个点构成的集合，那么对同一棵树中的集合S, T，$F(S \cup T) \subseteq F(S) \cup F(T)$

例题：51nod 树上的最远点对：

给定一棵树，多次询问a, b, c, d，求$\max\limits_{a \le i \le b, c \le j \le d} dist(i, j)$

直径的一个性质

令$F(S)$表示集合S中最远的两个点构成的集合，那么对同一棵树中的集合S, T，$F(S \cup T) \subseteq F(S) \cup F(T)$

例题：51nod 树上的最远点对：

给定一棵树，多次询问a, b, c, d，求$\max\limits_{a \le i \le b, c \le j \le d} dist(i, j)$

用线段树维护区间最远点对，利用上面的性质进行合并即可。

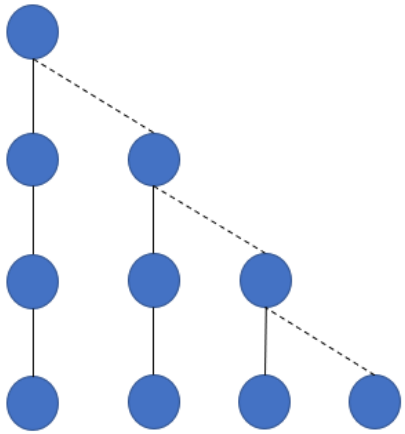
长链剖分

在重链剖分中，将重儿子的定义变为子树内叶子深度最大的儿子，得到“长链剖分”

长链剖分

在重链剖分中，将重儿子的定义变为子树内叶子深度最大的儿子，得到“长链剖分”

注意长链剖分后，一个点到根的轻边个数可以是$O(\sqrt{N})$级别的，如图：



k-th ancestor

利用长链剖分，我们可以在$O(N \log N)$的预处理后，$O(1)$询问一个点x的第k个祖先。

k-th ancestor

利用长链剖分，我们可以在$O(N \log N)$的预处理后，$O(1)$询问一个点x的第k个祖先。

对于某一条长度为l的重链以及链头x，记录x的第$1, \ldots, l$个祖先以及这条重链从上到下的l个点。这一部分是$O(N)$的，需要一些技巧分配空间

接下来预处理$f_{i,j}$表示i的第2^j个祖先，复杂度$O(N \log N)$，同时要预处理$1..N$的二进制最高位

询问时，考虑r为$\leq k$的最大的2的幂。令y为x的第r的祖先，z为y所在的重链的链头，链长为l，显然$l \geq r$。因此x的第k个祖先在预处理时已经记录了，可以直接查询

定长最长链

给定一棵树，求长度为L的边权和最大的链

定长最长链

给定一棵树，求长度为L的边权和最大的链

对点x，设重链长为l，维护$f_{x,1..l}$表示以x为根长度为$1..l$的边权和最大的链。每次从重儿子地方继承得到f，然后和轻儿子依次合并。

时间复杂度为$O(N)$

待修树直径

利用树链剖分每个点到根的路径上轻边个数为$O(\log N)$的性质，可以维护一些动态规划，例如：

- 多次操作，每次修改一条边的边权（可以为负），输出修改后的直径长度

待修树直径

利用树链剖分每个点到根的路径上轻边个数为$O(\log N)$的性质，可以维护一些动态规划，例如：

- 多次操作，每次修改一条边的边权（可以为负），输出修改后的直径长度

考虑dp，令f_x, g_x表示以x为根的最长链和x子树内的直径长度，令y为当前转移的儿子，用$(f_y + 1, \max(f_x + f_y + 1, g_y))$更新$(f_x, g_x)$

考虑优先转移轻儿子，最后转移重儿子。令f', g'表示转移完轻儿子的结果，那么每次修改只会改变$O(\log N)$个f', g'，可以暴力处理；重链上的转移可以维护类似$a_i = \max(a_{i+1} + A, B)$的标记$A, B$，使用线段树进行合并

总复杂度为$O(M \log^2 N)$，常数较大

link-cut-tree

动态树问题，现在已经兴起成为了OI中的重要题型。而lct，则是解决动态树问题的首选

lct的核心思想是access的均摊$O(N \log N)$复杂度

SDOI2017 树点染色

- 给定一棵N个点的树，接下来进行M次操作，每次从下面三种中选一个：
  - 选择一个点x，将x到根的路径所有点涂上一种新的颜色
  - 询问x, y路径上有几种颜色
  - 在x的子树内选择一个点y，满足y到根的路径上颜色数量最多
- $N, M \leq 100000$

SDOI2017 树点染色

- 给定一棵N个点的树，接下来进行M次操作，每次从下面三种中选一个：
  - 选择一个点x，将x到根的路径所有点涂上一种新的颜色
  - 询问x, y路径上有几种颜色
  - 在x的子树内选择一个点y，满足y到根的路径上颜色数量最多
- $N, M \leq 100000$

注意到我们只需要维护一个集合S，S包含所有和父亲的颜色不一样的点，就能完成询问

由于第一种操作类似lct的access，利用lct的access的均摊分析可以发现S中点的变化总次数是$O(N \log N)$的，这样再维护一棵答案线段树即可

总复杂度$O(N \log^2 N)$

维护MST

利用lct可以维护只有插入的MST

为了方便，可以对每条边新建一个点，将$x-y$变成$x-z-y$，然后将边权保存在新建点上

NOI2014 魔法森林

给定一个N个点，M条边的图，每条边有两个值a_i, b_i，求一条从1到N的路径使得$\max a_i + \max b_i$ 最小

NOI2014 魔法森林

给定一个N个点，M条边的图，每条边有两个值a_i, b_i，求一条从1到N的路径使得$\max a_i + \max b_i$ 最小

假设每条边只有一个参数，那么就是一个经典的最小生成树

于是我们考虑从小到大枚举$\max a_i$，维护关于b_i的最小生成树

时间复杂度$O(M \log N)$

维护图的连通性

如果允许离线，那么可以利用lct维护图的连通性的有关信息

根据贪心的思想，我们希望保留尽量晚被删除的边。于是可以考虑维护以删除时间为权值的最大生成森林，并利用lct加速插入和删除的过程

维护图的连通性

如果允许离线，那么可以利用lct维护图的连通性的有关信息

根据贪心的思想，我们希望保留尽量晚被删除的边。于是可以考虑维护以删除时间为权值的最大生成森林，并利用lct加速插入和删除的过程

如果一个图G存在补图G'，可以考虑同时维护图G和G'的两个生成森林T和T'，在G中的删边相当于在G'中加边，这样可以解决lct难以实现删边的问题

点分治

树的重心：一般情况下，如果一个点满足它作为根时最大子树的大小最小，我们就将这个点称为这棵树的重心

点分治是将当前子树的重心作为分治中心的一种分治方法，作用与序列分治类似，常常用来优化dp或加快合并速度

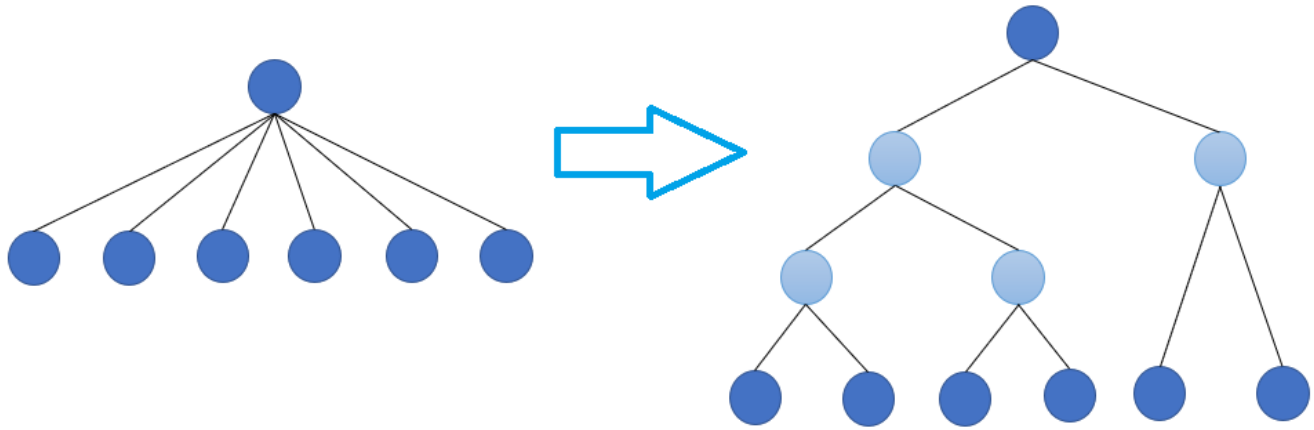
可以发现在点分治结构中，一个点与一个以它为重心的子树对应。如果将当前重心与所有子树的重心相连，得到的树称为“点分树”或者“重心树”。点分树的高度为$O(\log N)$，修改一个点时，将会修改点分树上它到根路径上所有点对应的子树的信息。

边分治

在当前子树中选择一条边作为分治中心的分治方法，称为边分治。一般而言我们希望分治后最大的子树尽量小

边分治的优点在于分治后只有两个子问题，但是往往需要二叉化来避免退化

二叉化：



实际上，二叉化之后的点分治往往也能达到边分治的效果

树上k远点

与“距离”有关的问题，往往可以使用点分治加以解决

我们首先二分答案W，问题转化为求与x距离$\leq W$的点的个数。我们在点分治时，对每个点，记录其对应的子树内所有点到它的距离并排序，这样询问时二分查找即可。注意要减去同一子树内的点

单词询问时间复杂度$O(\log^2 N \log W)$

带权距离和

树上所有点到点x的带权距离和定义为$\sum_{i=1}^{N} w_i dist(x,i)$

带权距离和

树上所有点到点x的带权距离和定义为$\sum_{i=1}^{N} w_i dist(x,i)$

方法一：点分治

对每个点，维护它对应的子树内所有点到它的带权距离和。修改时沿着点分树修改，修改和查询都是$O(\log N)$

带权距离和

树上所有点到点x的带权距离和定义为$\sum_{i=1}^{N} w_i dist(x,i)$

方法一：点分治

对每个点，维护它对应的子树内所有点到它的带权距离和。修改时沿着点分树修改，修改和查询都是$O(\log N)$

方法二：树链剖分

$$\sum_{i=1}^{N} w_i dist(x,i) = \sum_{i=1}^{N} w_i(d_i + d_x) - 2\sum_{i=1}^{N} w_i d_{lca(i,x)}$$

因此，对每个点将其到根的路径链上$-w_i$，然后询问x到根的路径的点权和，修改和询问都是$O(\log^2 N)$

这两种方法都可持久化

带修带权重心

树的带权重心指的是满足$\sum_{i=1}^{N} w_i dist(x,i)$最小的$x$，修改为单点修改

带修带权重心

树的带权重心指的是满足$\sum_{i=1}^{N} w_i dist(x,i)$最小的$x$，修改为单点修改

重心有一个重要的性质：对于一条边$x-y$，如果x侧子树权值和$>y$侧权值和，那么重心一定在x侧

值得一提的是这个结论对于$dist^k(x,i), k \geq 1$都是成立的，一般情况下需要求出$\sum w_i dist^{k-1}(x,i)$

带修带权重心

树的带权重心指的是满足$\sum_{i=1}^{N} w_i dist(x,i)$最小的$x$，修改为单点修改

重心有一个重要的性质：对于一条边$x-y$，如果x侧子树权值和$>y$侧权值和，那么重心一定在x侧

值得一提的是这个结论对于$dist^k(x,i), k \geq 1$都是成立的，一般情况下需要求出$\sum w_i dist^{k-1}(x,i)$

利用这个结论我们可以快速求出树的带权重心

带修带权重心

第一种方法是点（边）分治，在二叉化之后，我们枚举重心（即不带权重心）的所有儿子就能得到带权重心所处的子树，分治即可，单次询问$O(\log N)$，但需要维护所有点对应的子树的权值和

修改时沿着点分树将所有点对应的子树的权值和进行修改

如果修改不是单点修改，那么就需要用一个额外的结构来维护子树和，时间复杂度变为$O(\log^2 N)$

第二种方法是利用树链剖分，我们可以利用重链上二分+维护所有轻儿子子树权值和最大值的方法，做到修改和查询都是$O(\log^2 N)$的复杂度

动态点分治

WC2014紫荆花之恋

带插入，每次插入后询问有多少对i,j，满足：$dist(i,j) \leq r_i + r_j$，强制在线

动态点分治

WC2014紫荆花之恋

带插入，每次插入后询问有多少对i,j，满足：$dist(i,j) \leq r_i + r_j$，强制在线

假设我们有了点分治的结构，只需要对每个点x用平衡树维护对应子树所有点的$dist(i,x) - r_i$，插入点y时用$r_y - dist(x,y)$去查询即可

由于树的形态会改变，需要维护点分治的结构，可以利用替罪羊的思想暴力重构

时间复杂度$O(N \log^3 N)$

线段树合并

对两棵线段树（一般为动态开点线段树），合并方法为：

- 令根节点为x, y
- 如果其中一颗线段树为空（$x = 0$或$y = 0$）,返回另外一棵
- 否则递归合并x,y的左子树，递归合并x,y的右子树，最后合并信息

容易证明线段树合并的时间复杂度不超过总的空间复杂度

子树内众数

对每个节点用一棵线段树维护子树内每个数的出现次数

时间复杂度$O(N \log N)$

hihoCoder挑战赛13 树堆

一棵带权树被称作树堆当且仅当每个点点的权值都≥它所有儿子的权值

给定一棵N个点的带权树，可以不断进行操作：选择一个非根节点x，将x所有儿子连到x的父亲上

对每个节点x求出若干次操作后，满足以x为根的子树形成一个树堆的前提下，以x为根的子树的最大可能大小，

$N \leq 100000$

hihoCoder挑战赛13 树堆

一棵带权树被称作树堆当且仅当每个点点的权值都≥它所有儿子的权值

给定一棵N个点的带权树，可以不断进行操作：选择一个非根节点x，将x所有儿子连到x的父亲上

对每个节点x求出若干次操作后，满足以x为根的子树形成一个树堆的前提下，以x为根的子树的最大可能大小，

$N \leq 100000$

对每个节点x维护一棵线段树，线段树下标k保存如果x及x的某个祖先的权值为k，树堆的大小的最大值

这样每次将x的儿子线段树全部合并后，只需给区间$[w_x, +\infty)$区间+1

ZJOI2018 历史

有N个城市构成一棵有根数，每个城市有一个国家和一个值a_i表示这个国家崛起的次数。一个国家x崛起时将会夺取x到根的路径上所有城市的控制权，并和原先控制这个城市的国家战斗。一次崛起的灾难度定义为这个国家崛起一共和多少不同的国家战斗

一共有M次单点修改，你需要给出修改后所有的崛起顺序中灾难度之和的最大值

$N, M \leq 400000$

ZJOI2018 历史

考虑求出所有点的贡献之和，假设一个点的儿子的子树$\sum a_i$分别是$b_1, b_2, ..., b_k$，令$S = a_i + \sum b_j$，那么这个点的贡献为$\min(2(S - \max(\max\{b_j\}, a_i)), S - 1)$

考虑如果一个点x的儿子y满足$b_y \geq a_x$且$2b_y \geq S$那么就将y作为x的重儿子，x和y之间为重边。那么显然一个点到根的路径上轻边的数量为$O(\log \sum a_i)$，利用lct维护这个轻重链剖分即可

时间复杂度为$O(N \log \sum a_i)$

2018多省省队联测

秘密袭击

给定一棵N个点的树和两个数k, W，求所有连通块的第k大的值的和对64123取模的值

$N, W \leq 1333$

2018多省省队联测

秘密袭击

给定一棵N个点的树和两个数k, W，求所有连通块的第k大的值的和对64123取模的值

$N, W \leq 1333$

考虑枚举x，统计有多少个连通块含有$\geq k$个权值$\geq x$的点。

利用线段树合并，线段树的每个节点保存一个背包，合并两个点即合并两个背包，注意标记永久化。同时为了最后统计方便，需要额外围护一个背包表示子树的背包的和

这样可以做到$O(N^3 \log W)$的复杂度

注意到背包的总容量为N，也就是对应的多项式次数为N，因此只需要保留多项式的N个点值，单次合并的复杂度降低为$O(N)$，因此总复杂度为$(N^2 \log W)$