浅谈分块在一类在线问题中的应用

宁波市镇海中学 邹逍遥

摘 要

分块是一种适用性高，易于实现且思维难度低的方法，能够用于解决一些信息不能快速合并的题目。本文介绍了几种常见的分块方法，并通过一些例题简单介绍了分块在一些在线维护（或询问）问题中的应用。

引言

在竞赛和实践中，往往会出现一类要求维护序列或树形结构的问题，而且一般带有修改操作。有许许多多的数据结构可以在log级别的复杂度中维护一些比较简单的情况，但是遇到一些比较复杂的情况这些数据结构往往会遇到一些困难，需要使用结构嵌套或挖掘需要维护的信息的深层性质甚至根本无法解决。

在遇到这类题目时分块方法的优势就体现出来了：不需要对问题做过多的分析，一种方法可以维护许许多多不同的要求。虽然分块方法的复杂度往往是根号级别甚至更高，但是分块方法的思维复杂度和代码复杂度往往都远低于复杂度更低的数据结构，在实践中实用性很高。

当然这类题目也有一些优秀的离线算法可以解决，比如莫队算法，cdq分治等等。不过假如遇到强制在线的题目，这些算法就无法发挥了。

本文简单介绍了几种常用的在线维护分块结构或预处理分块结构以便在线回答询问的方法。第1节主要介绍了直接将序列或树分割成许多块进行维护的方法；第2节主要介绍了通过分类将信息分开维护或将询问分类回答的方法；第3节主要介绍了定期重建思想的运用。

以下题目如未特殊说明均要求在线算法。

1 直接分块

1.1 序列分块

序列分块是算法竞赛中最常见也最容易实现的一类分块题目。

基本思想是在序列上每隔$H(N)$个点选择一个关键点，这样任意一段区间询问都能被拆成$O(N/H(N))$个区间和$O(H(N))$个单点，询问时就不需要遍历整个数组，同时在修改的时候也不需要修改整个数组。

1.1.1 最简单的例子

【例1】弹飞绵羊[1]

给你N个数a_i满足$i < a_i \leq N+1$，有N个操作，每次可以

- 修改一个数（修改后仍然满足上述条件）
- 查询从$x = i$开始每次将x变为a_x直到$x = N+1$一共需要几次变换。

这题可以使用Link-cut trees在$O(N \log N)$的时间内解决，不过这和本文关系不大故略去。

首先考虑两种暴力做法：修改直接修改那个位置，查询的时候$O(N)$模拟。或者修改的时候$O(N)$将每个位置的答案算出来，查询的时候$O(1)$回答。

那么我们可以使用分块来平衡两种做法的复杂度：首先将序列每连续的$H(N)$个分成一块，例如当$N = 18, H(N) = 4$时（相同颜色的表示一块）：

那么对于每一块使用第二种暴力算出每个位置的答案，在修改的时候只需要将这一块整个重新计算，时间复杂度$O(H(N))$。在查询的时候使用第一种暴力，由于利用预处理信息每次至少可以跳过一段，时间复杂度$O(N/H(N))$。

注意到只需要在预处理中算出某个数跳出所在的那一块需要多少步和跳出去后落在哪个位置就可以使在查询的时候每次都能从当前点$O(1)$跳出这一段。

由于要平衡修改和询问的复杂度，所以当$H(N) = \sqrt{N}$时最优，这样这题就可以在$O(N\sqrt{N})$的时间内解决了。

[1]题目来源:HNOI2010

1.1.2 一个稍微复杂一点的例子

【例2】Chef and Churu[2]

给定N个数a_i，N个函数f_i，每个函数表示一段区间的a_i的和（即$f_i = \Sigma_{x=l_i}^{r_i} a_x$，$f_i$的值会根据$a_i$的值的改变而改变）。有$N$ 个操作，每个操作可以：

- 修改一个数（a_i）
- 询问$\Sigma_{x=l}^{r} f_x$

注意到维护一个树状数组就可以$O(\log n)$求出某一个f_i的值，所以可以考虑对f_i分块。

和上一题一样先将f_i每隔$H(N)$个分成$N/H(N)$块，对于每一块都预处理出这一块中每个a_i出现了几次，以及初始状态下这一块中f_i的和，时间复杂度$O(N \times H(N))$。

对于每个修改，将每一段的f_i的和更新一下即可（由于预处理出了a_i的出现次数所以这很容易实现），时间复杂度$O(N/H(N))$。

对于每个询问，将询问区间拆分成$O(H(N))$个单点和$O(N/H(N))$个区间，其中每个单点可以在$O(\log N)$的时间内解决（通过维护树状数组即可），每个区间可以在$O(1)$的时间内解决。

不过注意到维护树状数组是$O(\log N)$-$O(\log N)$的，而修改的时候完全可以做到更高的复杂度而不影响整体复杂度。所以可以使用$O(\sqrt{N})$-$O(1)$ 支持单点修改和区间和查询的数据结构维护a_i：

由于只需要查询区间和，满足可减性质，于是可以将区间和转化为前缀和相减。由于改了一个点会影响到的前缀查询是一个后缀，所以单点修改前缀查询可以变成后缀修改单点查询。那么可以把a_i每隔$O(\sqrt{N})$个分为$O(\sqrt{N})$块，每次修改只需要修改$O(\sqrt{N})$块和$O(\sqrt{N})$ 个单点，查询只需要把块内答案和单点答案相加即可。

这样询问和修改就可以做到$O(N/H(N) + H(N))$，当$H(N) = \sqrt{N}$的时候时间复杂度为$O(N\sqrt{N})$，空间复杂度为$O(N\sqrt{N})$。

[2]题目来源:Codechef Nov 14 Challenge

1.1.3 需要维护块的区间信息的情况

【例3】Chef and Problems[3]

给定N个数a_i，有N个询问，每次给出l, r，求满足$a_i = a_j$和$l \le i, j \le r$的$j-i$的最大值。

首先考虑对于每个询问$O(n)$的暴力做法：扫一遍这个区间，并用每个数和这个区间内这个数第一次出现的位置更新答案。使用时间戳进行更新就可以避免每次遍历整个数组而只需要遍历询问区间的长度。

那么可以和上一题一样将a_i每$H(N)$个分成$N/H(N)$块，用暴力算法算出每一块内部的答案。时间复杂度$O(N)$。

但是注意到这题不能像上一题一样直接合并块内信息，还需要计算块的区间内的答案。首先需要预处理出和a_i相同的满足$j > kH(N)$的最小的j，同理算出和a_i相同的满足$j \le kH(N)$的最大的j（时间复杂度$O(N \times N/H(N))$）。这样就可以算出i在第x块，j在第$x \sim y$块时的最大解，记为$calc(x, y)$（时间复杂度$O(N \times H(N))$）。

同时第x块到第y块的答案还包含端点不在第x块或第y块上的情况，所以需要使用一个区间dp来算出这部分的答案：用$f(i, j)$表示第i块到第j块之间的答案。那么$f(i, j) = \max(f(i+1, j), f(i, j-1), calc(i, j))$。

接下来处理询问的时候就方便多了，每个区间都可以分成一个已经计算出答案的区间和不超过$2H(N)$个点。可以扫一遍多出来的点在$O(H(N))$的时间内算出两边端点都在外面的答案。然后就利用预处理信息算出每个外部的点和区间内部的点的最大答案。将三部分答案取最大值即可。时间复杂度$O(H(N))$。

总的时间复杂度为$O(N \times (H(N) + N/H(N)))$-$O(H(N))$，空间$O(N \times N/H(N))$。当$H(N) = \sqrt{N}$的时候时间复杂度为$O(N \times \sqrt{N})$。但是为了节省空间可以把$H(N)$适当开大。

像例3这样的题目不能很方便地加入修改操作，因为每次修改可能造成$O((N/H(N))^2)$个预处理值变动。不过加入修改以后可以通过让$H(N) = O(N^{\frac{2}{3}})$来达到$O(N^{\frac{5}{3}})$的复杂度。

[3]题目来源:Codechef March 15 Challenge

1.2 树上分块

有许多题目在链上分块时很容易实现，但是在树上使用分块就会比较麻烦。本节介绍了一种非常容易实现的对树分块的方法，可以解决一系列的只询问链上信息或祖先关系信息的题目。

1.2.1 解决链上信息询问

【例4】Children Trips[4]

给定一棵N个点的树，每条边长度是1或2。有N个询问，每次给出x,y,z，要求找出尽量少的点$a_1 \cdots a_n$使得$dis(x,a_1), dis(a_1,a_2), \cdots, dis(a_n,y)$ 都不超过z（这里$dis(i,j)$表示树上i点和j点的距离）。

这道题的信息合并非常困难，因为合并不支持结合律，只能从前向后合并过去。但是可以观察到有以下两个性质：

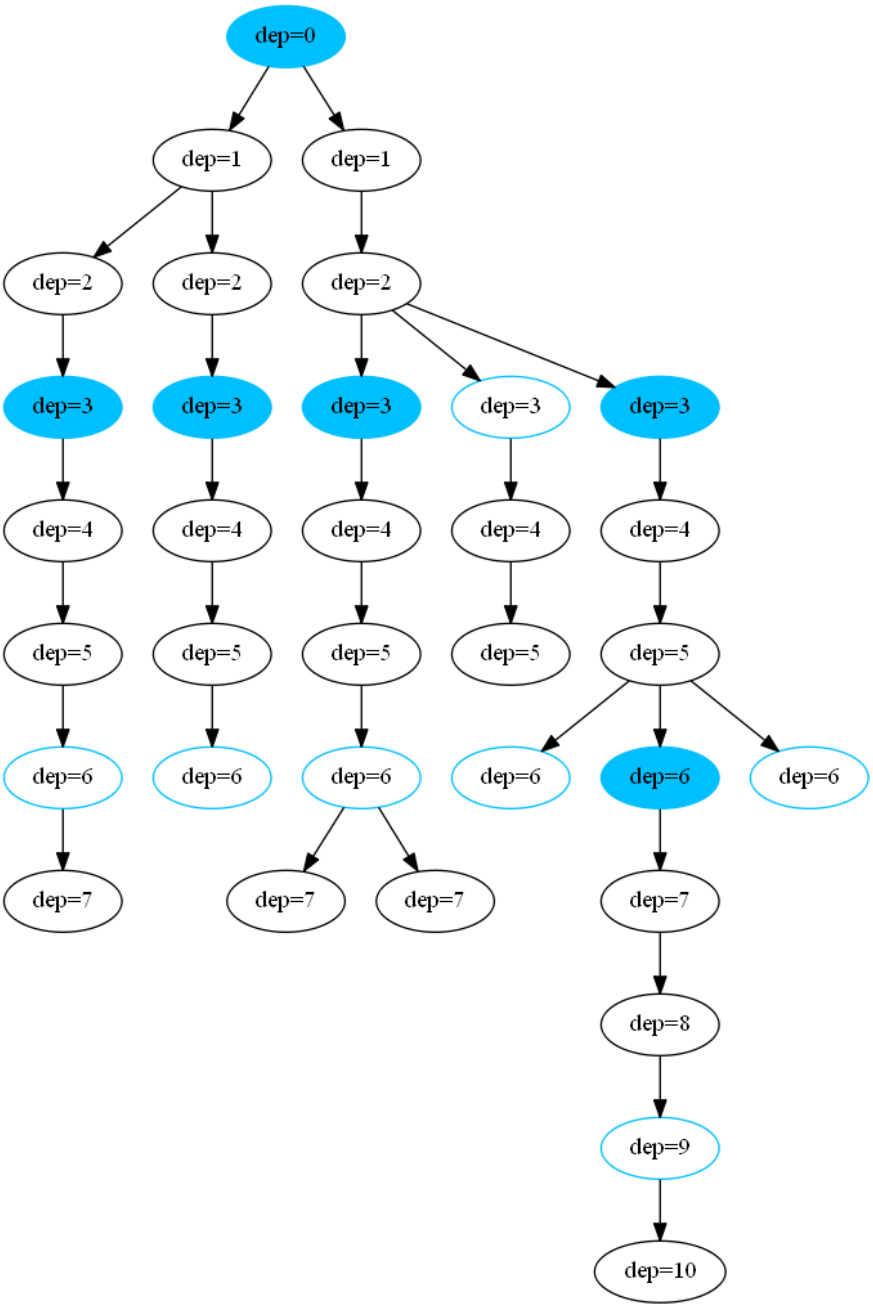
性质1:假如按i递增的顺序选择a_i，每次选a_{i-1}（设x为a_0,y为a_{n+1}）到y路径上最远的那个满足条件的点当a_i，这样选出来的点一定是满足最小的条件的（即每次走的步长尽量大答案不会变差）。

性质2:假如前一些点按i递增选最远的直到a_x，后面一些点按i递减选最远的直到a_x，这样选出来的点也是满足最小的条件的（即两端向内同时走答案不会变差）。

考虑像序列分块一样在树上选出一些关键点。首先求出每个点的深度dep_i（根结点为0）和子树大小$size_i$，并把那些满足$dep_i \mod H(N) = 0$的点选出来。

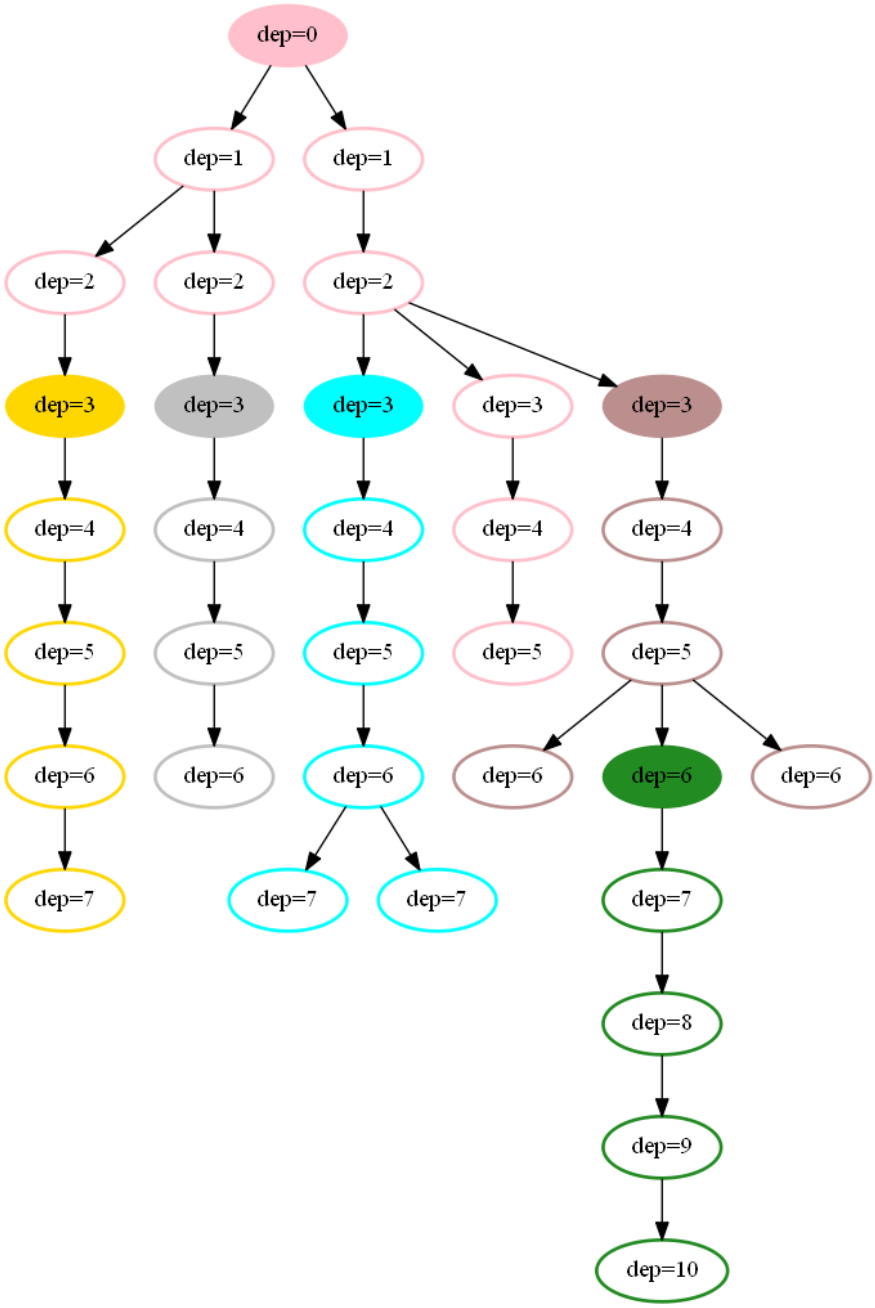
但是注意到这样做可能会选出非常多的点（比如一条长为$H(N)-1$的链，在链的一段接出去$O(N)$个点），所以需要把那些子树大小不超过$H(N)$的点去除。最后选出来的点如下图所示：

[4]题目来源:Codechef Oct 14 Challenge



然后我们把所有的关键点拿出来建成一棵树，称为“关键树”，在关键树上每个节点的父节点是原树上向上走的第一个关键点，每条边对应原树中这两个点之间的链。同时每个关键点代表一个点集，点集包含所有向根走第一个遇到的关键点是它的点。点集分类如下图所示，实心点表示关键点，相同颜色的点

属于同一个集合。



那么我们分析一下这个结构有哪些性质：

性质3:任意两个关键树上的相邻关键点在原树中距离为$H(N)$。

性质4:每个关键点代表的点集的大小不小于$H(N)$。

证明:假如这个关键点在关键树中是叶子，那么它所有原树中的子孙都属于这个点集，由定义显然成立。假如这个关键点在关键树中不是叶子，那么它到它的某一个儿子节点的那条链有$H(N)$个点，而这些点显然从属于它代表的集合。

性质5:最多选出$O(N/H(N))$个关键点。

性质6:每个点至多向上走$2H(N)$步就能遇到一个关键点。

证明:假设走了$2H(N)$步还没有遇到关键点，那么容易看出向上走$2H(N)$步经过的点至少有两个点的深度被$H(N)$整除，那么其中更靠近根的那个点的子树大小显然超过了$H(N)$，所以那个点一定能成为关键点，所以假设不成立。

性质7:任意两个点之间的路径可以被拆分成$O(H(N))$条原树上的边和$O(N/H(N))$条关键树上的边。

证明:由于不可能出现一个点连续向父亲走了$2H(N)$步的情况，所以最多只有$4H(N)$条原树上的边。而关键树上的边总共只有$O(N/H(N))$条，所以不可能多于$O(N/H(N))$条。

假如已经预处理出关键树上相邻两个点之间的答案，询问就很好处理了。

注意到这里只需要维护z为$1 \sim 2H(N)$时的答案即可。因为假如$z \geq 2H(N)$，那么这一段会被直接跳过去所以不需要计算。那么首先需要计算出每个点向上以z为步长跳到的第一个点是什么，跳到最近的关键点需要几步，跳到之后最后一步还剩多长。预处理复杂度是$O(N \times H(N))$。

询问的时候只需要利用性质2，类似倍增求LCA的算法将两个点不停向根移动就可以了。不过注意LCA并不一定是关键点，所以在再跳一步就要高于LCA的情况下要使用暴力把最后的一段跳完。在移动的过程中维护已经移动了几步和最后一步还剩多长。移动一大步的时候首先把最后剩余的长度移动完，然后利用预处理信息一步移动到上一个关键点。时间复杂度$O(N/H(N))$。

当$H(N) = \sqrt{N}$的时候时间复杂度为$O(N\sqrt{N})$，空间复杂度为$O(N\sqrt{N})$。

1.2.2 解决祖先关系询问

【例5】Regions[5]

[5]题目来源:IOI2009

给定一棵N个结点的树，每个结点有一个颜色，每次给出x,y查询有多少对数i,j满足i的颜色是x，j的颜色是y且i是j的祖先。

我们像上一题一样建出一棵关键树，但是注意到这样的关键树并没有一些很容易划分成祖先关系的性质，于是我们将它进行一些扩展。

首先和上一题一样选出“初始”关键点（$dep_i \mod H(N) = 0$且$size_i \geq H(N)$的点），然后把所有满足是两个初始关键点的LCA（最近公共祖先）的点也选出来并加入关键点集合形成最后的关键点集。

性质1:关键点最多增加一倍。

证明:将初始关键点按DFS序排列，按顺序枚举初始关键点，对于每个初始关键点，不停地向根节点走直到走到另一个初始关键点或一个已经访问过的点。假如停下来的时候所在的位置是初始关键点，那么再向上走的情况和那个点向上走的情况完全相同，没有必要再走下去。假如走到的是已经访问过的点，就把这个点加入关键点集合，那么再向上走的情况和第一次访问这个点的点遇到的情况完全相同，也没有必要再走下去。所以每个点至多只会贡献一个新点。

其实这么做就是建出了一棵“虚树”，在实现的时候可以直接加入所有DFS序相邻的两个点的LCA并去重即可。关于虚树的详细讨论可以参考2014年徐寅展的集训队论文。

扩充了关键点集合后关键树的定义仍然和之前相同（每个点的父亲是原树中的最近关键祖先），关键树中的边代表原树中的一条链中的点组成的点集。

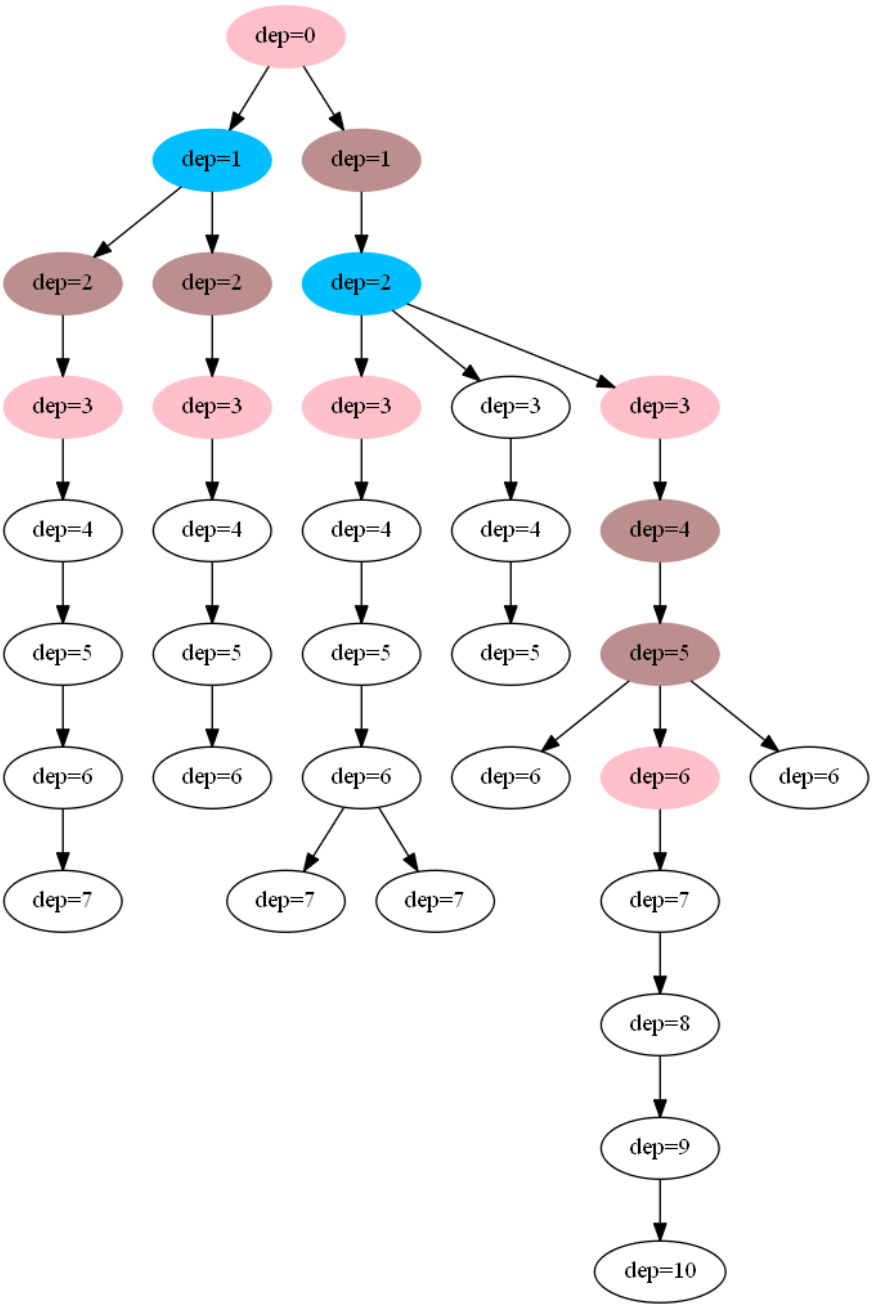
现在这棵树就又满足了更多性质；

性质2:任意两条关键树中的边在原树中代表的路径只会在端点处相交。

性质3:树中任意两点之间的路径都可以被划分为$O(H(N))$条边和$O(N/H(N))$条相邻的关键点之间的路径。

性质4:任意两个相邻的关键点之间的路径长度不超过$H(N)$且关键点数量不超过$2N/H(N)$。

新选出的关键点如下图所示（粉色表示初始关键点，蓝色表示新加入的关键点，棕色表示位于关键树的边上的点）：



那么有了这些性质就容易解决这道题了。

把所有点分成两类：把位于关键树的边上的点（包括关键点，比如上图中的所有有颜色的点）称为“内部点”，把剩下的点称为“外部点”。那么我们把所有需要统计进去的情况分成三类：

- 两个点都是内部点。
- 祖先结点是内部点，子孙结点是外部点，但是祖先结点不在子孙结点向上走到第一个关键点所经过的路径上。
- 祖先结点是内部点，子孙结点是外部点，但是祖先结点在子孙结点向上走到第一个关键点所经过的路径上。

第三种情况很容易解决，由于每个点至多向上走$2H(N)$步就会遇到一个关键点，所以只需要暴力枚举就可以预处理出第三种情况的答案。

在关键树上把每个关键点和它到它父亲结点的路径上的点分成一块（根结点只包含他自己），那么对于每一块都预处理出块内部每种颜色的数量，这个可以在$O(N \times N/H(N))$的时间内解决。

容易证明假如关键点x是关键点y的祖先，那么x所在块中的每一个点都是y所在块中的点的祖先。所以在询问的时候只需要遍历一遍关键树就可以解决第一种情况。

最后考虑第二种情况：预处理出每个外部点向上走遇到的第一个关键点，并将这个外部点归入关键点的集合。那么可以和第一种情况类似地解决：只需要预处理出每个关键点所代表的外部点中每种颜色的出现次数，查询时利用DFS到这个点时这个点到根路径上的x颜色的数量和这个点的集合内的y颜色数量更新答案即可。

当$H(N) = \sqrt{N}$的时候本题的时间复杂度为$O(N\sqrt{N})$，空间复杂度为$O(N\sqrt{N})$。

2 按大小分类讨论

按大小分类讨论的基本思想是将询问（或修改或度数等等）分成大于$H(N)$的和不大于$H(N)$的两类分开解决。也就是将两种$O(size)$（或$O(size \log size$等）和$O(N/size)$的暴力结合在了一起，所以思维复杂度低，而且结合的是两个暴力做法，在考场上容易实现。

2.1 【例4】Children Trips[6]

给定一棵N个点的树，每条边长度是1或2。有N个询问，每次给出x,y,z，要求找出尽量少的点$a_1 \cdots a_n$使得$dis(x,a_1), dis(a_1,a_2), \cdots, dis(a_n,y)$ 都不超过z（这里$dis(i,j)$表示树上i点和j点的距离）。

考虑暴力做法：每次二分下一个a_i的位置，这样是$O(N \log N)$的，因为a_i最多会有$O(N)$个。但是注意到当$z > H(N)$的时候a_i最多只有$O(N/H(N))$ 个可以直接暴力，而$z <= H(N)$的情况只有不超过$H(N)$种，假如把这些z的答案预处理出来，那么问题就解决了。

那么可以对每一个比较小的的z，都预处理出每个点向根走z长度（假如没有距离为z的点那么就选取距离为$z-1$的点）到的是哪个点，并倍增出每个点向上走2^k 步（每步长度为z）到的是那个点。这样需要$O(N \times H(N) \log N)$ 的时间和空间。询问的时候可以像正常的倍增算法一样向上走到再向上走就超过LCA为止，最后多出来的那一段可以暴力在$O(H(N))$的时间内走完。

取$H(N) = \sqrt{N}$就可以在$O(N\sqrt{N}\log N)$的时间和$O(N\sqrt{N}\log N)$的空间下解决这道题。

这是官方给出的本题解法，思维难度比较低，容易想到。但是在树上两点间二分比较麻烦不易实现。使用第一章给出的做法能更容易地解决这道题。

2.2 【例5】Regions[7]

给定一棵N个结点的树，每个结点有一个颜色，每次给出x,y查询有多少对数i,j满足i的颜色是x，j的颜色是y且i是j的祖先。

首先来看一种特殊情况：所有的询问中x都相同。那么可以通过一个简单的DFS来算出每个y的答案，只需要在DFS的过程中记录下当前这个点到根的路径上有多少个颜色为x的结点即可。

[6]**题目来源**:Codechef Oct 14 Challenge
[7]**题目来源**:IOI2009

预处理出DFS序，那么每个子树所代表的点就是左端点为它本身的一个区间。并同时预处理出每种颜色的点代表的区间的左右端点排好序的结果。

对于每个询问，设两种颜色的点分别有A个和B个，那么算出每个点在多少个区间内，然后把答案相加即可。由于已经排好了序，那么可以在$O(A+B)$的时间内将排序结果归并起来，然后扫描一遍就能得到答案。

那么再考虑这样一种针对颜色数较少的情况的暴力：对于每个x都进行一次DFS，可以在$O(NC)$的时间内（C为颜色数）预处理出每个询问的答案，只需要在DFS时记录下当前点到根一共有多少种x颜色即可。类似地固定y时也可以通过扫一遍所有点算出答案，只需要预处理出前缀和就能$O(1)$查询区间了。

考虑将这两种暴力结合起来：对于出现次数超过$H(N)$的颜色，预处理出这些颜色作为其中一种（x或y）时的答案。对于剩下的在询问时使用第一种暴力直接回答。

当$H(N)=\sqrt{N}$时，时间复杂度为$O(N\sqrt{N})$，空间复杂度为$O(N\sqrt{N})$。

当然官方题解中还给出了另外一种分类讨论的方法可以在$O(N\sqrt{N\log N})$的时间和$O(N)$的空间之内解决。这种做法在2014年王悦同的论文中已经提及，这里不再赘述。

3 定期重建

定期重建的思想是将操作分块，每$H(N)$个操作后就重建一次状态，查询的时候根据上一次重建和期间的所有修改计算答案。

注意对操作分块并不需要离线，只需要每隔一定时间重新预处理一遍即可。

3.1 【例2】Chef and Churu[8]

给定N个数a_i，N个函数f_i，每个函数表示一段区间的a_i的和（即$f_i=\Sigma_{x=l_i}^{r_i}a_x$，$f_i$的值会根据$a_i$的值的改变而改变）。有$N$ 个操作，每个操作可以：

- 修改一个数（a_i）
- 询问$\Sigma_{x=l}^{r}f_x$

[8]**题目来源**:Codechef Nov 14 Challenge

先将操作分块，即每$H(N)$次操作重建一次，每次重建的时候$O(N)$算出原数组的前缀和并$O(N)$求出每个f_i的值，复杂度$O(N \times N/H(N))$。

接下来只需要计算每个操作对块内询问的贡献。由于询问是可减的，可以把每个询问拆成前缀询问，然后把每个前缀询问按长度排序。

排完序后就可以按右端点递增扫过来并维护一个支持$O(\sqrt{N})$区间加$O(1)$单点查询的数据结构，这个数据结构在之前提到过：把a_i每隔$O(H(N))$个分为$O(N/H(N))$块，每次修改只需要修改$O(N/H(N))$块和$O(H(N))$个单点，查询只需要把块内答案和单点答案相加即可。然后对于每个询问枚举和它在同一块中并且在它前面的所有$O(H(N))$个修改，由于可以$O(1)$查询某个位置对那个询问的影响，就可以在$O(N \times H(N))$ 的时间内算出这一部分的贡献。

总时间复杂度$O(N\sqrt{N})$，空间复杂度$O(N)$。

当然假如需要强制在线的话可以把排序并扫过来的那个操作改成预处理并记录在一棵可持久化线段树中。这样可以通过增加一个log来完成在线做法，时间复杂度为$O(N\sqrt{N}\log N)$，不过内存也需要$O(N\sqrt{N}\log N)$。

3.2 【例6】支持link,cut的树上第k大问题[9]

给定一个N个点的森林，有N个操作，每次可以加一条边/删一条边（保证处理完后还是森林），可以询问一条链上的第k大。

首先考虑没有link,cut操作的情况。这可以使用可持久化线段树来解决：每个点维护一棵在它的父亲的基础上加上这个点的点权的可持久化线段树，这样在查询的时候每条链可以拆分成两条从某个点开始指向根的路径，利用这两条链的和在可持久化线段树上二分即可。

假如维护了K次操作以前的可持久化线段树，那么可以通过Link-cut trees将现在树上的链拆分成最多$O(K)$条K时刻以前的树上的链。通过这些链的和在可持久化线段树上二分即可。

那么可以每进行$\sqrt{N}$次操作就将可持久化线段树重建，时间复杂度为

[9]经典问题