浅谈一类树分块的构建算法及其应用

杭州第二中学 周欣

摘 要

树分块算法是序列分块算法在树结构上的自然推广，具有广泛的应用场合。本文介绍了树分块的一类实现方式，并通过一些例题介绍了其应用。

引言

序列分块算法是一种朴素但有效的算法，通过与其它方法相结合，能够处理很多只靠传统树形数据结构不能处理的复杂问题，具有简洁高效的特点。

但是当我们试图将简洁高效的序列分块算法推广并用于处理树上问题的时候，由于树结构本身所具有的复杂性，无论是将树划分为若干个块的过程，还是成功划分之后借助分块结构处理问题的过程，相对于序列而言都增加了不少难度。

这也导致近年来 OI 中很多可以应用树分块算法解决的问题，为大家所熟知的解法都是借助一些通用手段[1]来将问题简化，从而规避对复杂的树结构的直接处理。总之，近年来树分块算法在 OI 中的应用比较局限。

对此，笔者希望通过本文，介绍如何将序列分块算法推广到树结构上，以及树分块算法的一些应用。

本文的第一节会对树分块算法本身进行介绍，第二节讨论树分块算法在一些复杂度根号问题上的应用，第三节讨论非传统块大小的树分块在一些其它问题上的应用。

1 树分块算法简介

可以用于树分块构建的方法很多，本节中仅仅介绍一种基于 top cluster 理论的，适用面较广的做法。

[1]比如沿时间轴定期重构

1.1 top cluster 概念的介绍

一个树簇（cluster）是树上的一个连通子图，有至多两个点和全树的其他位置连接。

这两个结点被称做界点（boundary Node）。

不是界点的点被称做内点（internal node）。

这两个界点之间的路径被称做簇路径（cluster path）。

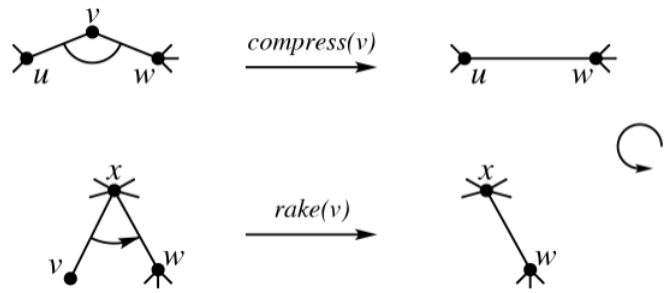


图 1

如上图图 1，簇是可合并的，并且簇的合并有两种方式。

如图所示，第一种合并方式被命名为 compress，要求对一个恰与两个簇相邻的点 v 执行，效果为将与其相邻的两个簇 (u,v) 和 (v,w) 合并为一个新簇 (u,w)。

第二种合并方式被命名为 rake，要求对一个要求对一个恰与一个簇相邻的点 v 执行，记与其相邻的簇的另一个端点为 x，与 x 相邻的除了 v 的另一个簇端点为 w，则效果为将 (v,x) 合并至 (w,x) 中。

无论是第一种还是第二种合并，执行完毕后点 v 都不再作为任何一个簇端点出现。显然，通过不断执行合并操作可以将整棵树收缩为单个簇。

簇的合并过程构成一个二叉树的结构（如图 2），我们称之为 top tree。

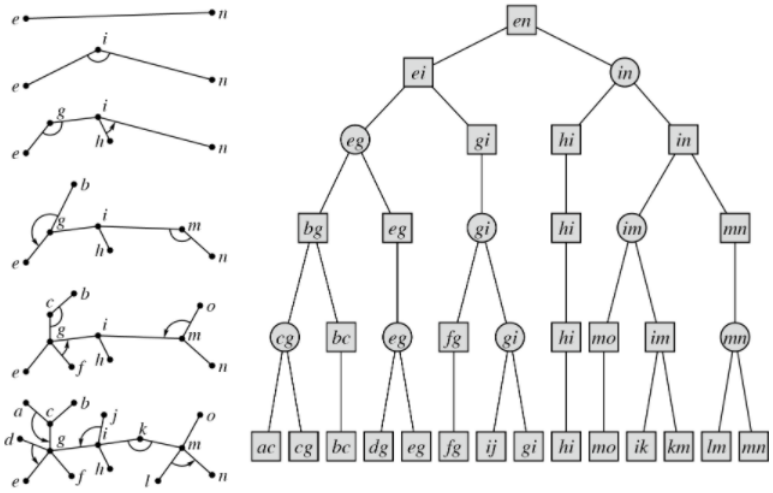


图 2

存在构造算法使得得到的 top tree 深度为 $O(\log n)$，关于 top tree 的更多其它内容，由于与主题无关，这里不展开详细描述。[2]。

1.2 树分块的基本结构

序列分块可以看作是一种只有两层节点的特殊树形数据结构。当我们将序列分块推广到树结构上的时候，自然的想法是，寻找一个将树表示为多层树形结构的基底数据结构，然后将节点强行拍扁成两层。

而上一节中 top tree 的概念，恰好为我们提供了这样一种基底数据结构。

于是，我们可以借用 top 簇的概念，精确地描述树分块构造算法所应当解决的问题：**对于一棵给定的 n 个点的树和一个块大小 B，要求将原树划分为 $O(\frac{n}{B})$ 个不交簇的并，并且每个簇的大小均为 $O(B)$。**

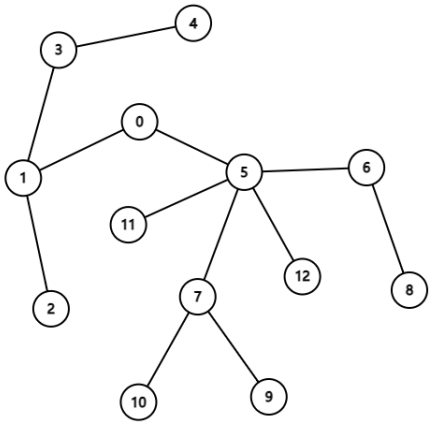


图 3

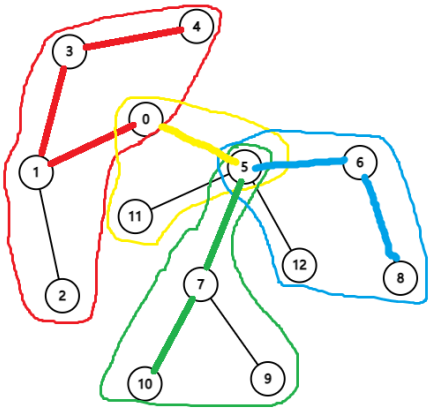


图 4

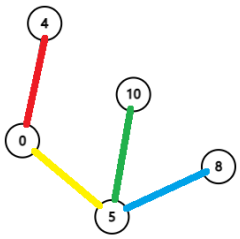


图 5

如图所示，图 3 为原树。图 4 为对原树的一组划分，每个簇都用彩线圈了起来，簇路径用彩线描了一下。图 5 为将每个簇看作一条边后，所有簇的界点所构成的新树，之后我们称之为**收缩树**。

关于这组划分的选取，在引入 top tree 的概念后，一个自然的想法是，先将原树建成一棵深度 $O(\log n)$ 的 top tree，再在 top tree 的二叉树结构上截取大小 $O(B)$ 的子树。但是由于

[2]可以在参考文献 [1] 中了解更多

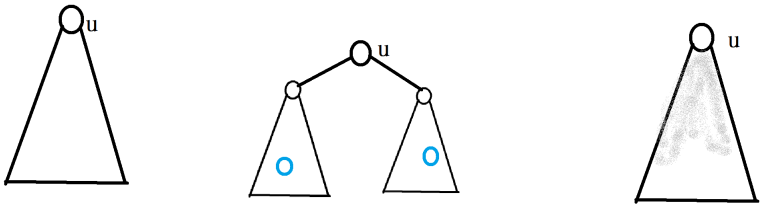
top tree 静态构建算法的复杂性，及其衍生的截取子树的复杂性，这并不是一个好的构造算法。下一小节中我们将介绍一个更简单方便的针对性构造算法。

1.3 一种比较易于实现的静态构造算法

首先，我们任选一个点为根，从该点开始运行 dfs 算法。dfs 过程中，我们需要确定原树的哪些点出现在最终的收缩树中；进而确定收缩树上的每条边包含原树的哪些边，或者说原树中的每条边，属于最终的收缩树的哪个簇。为此，dfs 的过程中我们需要维护一个栈用以存储暂时还未归类的边。

1.3.1 弹栈的过程

当前点 u 要结束 dfs 的时候，有一些栈中的边可能需要弹出来。具体的，出现如下三种情况时需要将栈中的边弹出来进行处理。



情况1 情况2 情况3

图 6

1. u 是全树的根。方便起见我们强制根节点出现在最终的收缩树中。
2. 存在 u 的至少两个不同子树，里面有已确定的簇的界点（图中蓝点）。由于一个簇至多拥有两个界点，所以此时点 u 不能为内点。
3. 栈中剩余边的数量（图中灰色部分）大于设定的阈值 B。

至此，我们先分析一下弹栈的三种情况发生次数。情况三只会发生严格不超过 $\frac{n}{B}$ 次，而三种情况中涉及到的点，所构成的树结构中，所有不是根的非叶子节点儿子数量一定大于 1，且叶子节点数量一定不超过情况三发生次数。从而三种情况的总发生次数一定不超过 $2 \cdot \frac{n}{B}$。

1.3.2 子树与边的划分

下面要解决的问题是，如何合适地将 u 的不同子树分割为不同的簇，来满足最初的要求。

此时问题可以抽象为，给定一个数 m 以及两个长为 m 的序列 $a_1,\cdots,a_m$ 和 $b_1,\cdots,b_m$。其中 m 是点 u 的儿子数量，

a_i 是点 u 第 i 个儿子的子树中，尚未归类的边的数量。如果情况 3 均被正确地处理，则 a_i 恒不大于阈值 B。

b_i 为一个布尔值，表示点 u 的第 i 个儿子的子树中是否存在已确定的界点。

我们要做的是，将序列切割为若干子段，使得每个子段中至多有一个 b_i 取值为真，且子段内的 a_i 和不超过 B。

实际上，每次贪心截取序列的一段极长合法前缀，将其归为同一个簇，所得到分割方案，就是满足算法所要求的复杂度的。

一个小细节是，如果截取的这个子段里 b_i 全为 0，这样得到的簇会只有一个界点，如果想让最后得到的结构符合最初的严谨定义，需要额外地任意选取另一个界点。不过实践中有时可以忽略选取额外界点的步骤。

1.3.3 复杂度证明

首先，这样得到的每个簇大小显然不超过 B。

接着，我们将前面提到的切割序列前缀，按照终止条件分为三类：

1. a_i 和将大于 B。
2. 将出现多于一个的 b_i 取值为真。
3. 这次截取完毕后序列将为空。

我们从前往后贪心地将第一类情况与其邻居配对，则每对的 a_i 和一定大于 B，从而对数不超过 $\frac{n}{B}$，从而归于这一部分的簇数量不超过 $2\cdot\frac{n}{B}$.

之后我们仅需考虑第二类情况和第三类情况的次数，而这两者的出现，均与一次弹栈相对应，故而不超过 $4\cdot\frac{n}{B}$。

于是，总的簇数量就不超过 $6\cdot\frac{n}{B}$。由此，我们的算法成功地完成了最初的要求：对于一棵给定的 n 个点的树和一个块大小 B，要求将原树划分为 $O(\frac{n}{B})$ 个不交簇的并，并且每个簇的大小均为 $O(B)$。

1.4 在动态树问题上的推广

本小节主要讨论如何在支持加边和删边的动态树问题上维护树分块的结构。

1.4.1 一种均摊复杂度的实现

每次执行加边或删边的时候，我们暴力将涉及到的一个或两个簇划分为 $O(1)$ 子簇，以便操作涉及到的点成为界点，这是容易的。之后再将处理一下操作边所构成的簇即可。

这样，一次操作后，每个簇的大小都仍然没有超过限制，并且簇的数量仅仅增加了 $O(1)$。对此，我们每 $O(\sqrt{n})$ 次操作后重构整棵树的分块结构，即可保证每个时刻的簇数量为 $O(\sqrt{n})$。

这样，我们就以单次均摊 $O(\sqrt{n})$ 的复杂度动态维护了树分块结构。这里假定我们需要维护的簇大小为 $O(\sqrt{n})$，具体实现的时候可以根据需要调整重构参数。

不过，在一些特殊场合，需要保证单次操作的最坏复杂度，均摊的算法不总是奏效。后文中我们将介绍一种可以保证单次最坏复杂度的算法。

1.4.2 伪簇的概念

在之前的讨论中，我们已经看到了簇概念的局限性。界点数量的限制导致我们无法保证每个块的大小下界，也为我们分析问题增加了困难。所以这里引入了伪簇的概念。

伪簇，定义为树上的一个连通边集。沿用前文中簇的定义，我们称伪簇中与外界相连的点为界点。但是和簇不同的是，伪簇不要求界点数量至多为 2，可以没有上限。

不过伪簇的结构并不利于信息维护，实践中我们可以保留界点及其虚树中的点，将单个伪簇建成点数至多为界点数量两倍的收缩树。由此，对于原树的一个伪簇划分，我们可以建出对应的簇划分。相应的，想要动态维护簇划分，我们只需以正确复杂度维护伪簇划分。伪簇的概念仅仅是为我们动态维护簇划分提供便利。

引理 1.4.1. *对于一棵树，以及一组分为 a 个伪簇的良好伪簇划分，界点数量不超过 a。这里一组伪簇划分定义为良好的，当且仅当每个界点同时属于至少两个伪簇。*

证明. 我们将伪簇记作圆点，界点记作方点。对于每个伪簇对应的圆点，我们向其所包含的界点对应的方点连边，这样可以得到一棵树。将任意一个点提根，则每个方点都有至少一个圆儿子（否则与“每个界点同时属于至少两个伪簇”矛盾）。从而方点数量不超过圆点数量，引理得证。 □

推论 1.4.1. *对于一棵树，以及一组分为 a 个伪簇的良好伪簇划分，其对应的簇划分界点数量不超过 $2a$。*

在下文的应用场合中，单个伪簇的大小总是有上界的。所以对于一个伪簇，将其建为收缩树的过程，只需在线性时间内完成即可，而且对每个簇的大小上下界没有额外要求。这样的构建是平凡的。

1.4.3 伪簇的合并与分裂

由于对于界点数量没有要求，所以对于两个有公共界点的伪簇来说，合并他们是平凡的，不像簇的合并受到诸多限制。

在合并过程中，为了保证单个伪簇的大小上界，我们需要进行分裂操作。实际上我们有如下引理：

引理 1.4.2. *对于一个大小为 n 的伪簇，存在算法将其划分为两个伪簇，使得较小的那个大小不小于 $\frac{n}{3}$。*

证明. 我们将该伪簇的重心提根，记 m 为重心最大子树的大小。则 $m \le \frac{1}{2}n$。

若 $m \ge \frac{1}{3}n$，则将重心最大子树分为一个伪簇，其它子树分为另一个即可。

否则我们按任意顺序将重心的子树加入第一个伪簇。当第一个伪簇的大小第一次大于等于 $\frac{1}{3}n$ 的时候，由于 m 的限制，其大小一定也不超过 $\frac{2}{3}n$。

此时我们终止加入进程，至此引理得证。 □

1.4.4 树分块的动态维护

对于给定的森林和阈值 B，我们可以维护森林的一组伪簇划分，并且每个伪簇均满足如下两种条件之一：

1. 每个伪簇的大小均在 $[0.5B, 2B]$ 之间。
2. 该伪簇所属连通块仅有一个伪簇。

具体的，对于一组合法的伪簇划分，如果我们要进行加边操作，如前所述，合并是平凡的，唯一问题是新生成的伪簇大小可能不合法。如果不合法，新生成的伪簇大小一定在 $(2B, 4B]$ 之间，我们只需对该伪簇执行至多 2 次引理 2.3.2 中提到的分裂算法，即可将其分为多个大小合法的子伪簇。

如果我们要进行删边操作，切割伪簇是平凡的，唯一的问题仍然是新生成的伪簇大小可能不合法。对此，我们暴力枚举新生成的伪簇的邻居伪簇并将其合并，如果合并后的新伪簇大小过大，像加边时一样进行切分即可。

这样，每次只会合并和分裂 $O(1)$ 个伪簇。由之前提到的伪簇划分与簇划分的对应关系与转化方式，我们得到的算法可以以 $O(B)$ 的复杂度动态维护树分块的结构。

2 树分块算法在一些根号复杂度问题上的应用

在当前的 OI 竞赛中，使用序列分块算法的题目常常具有 $O(n\sqrt{n}poly(\log n))$ 的复杂度。当序列分块推广到树上的时候，自然也会想到以 $O(\sqrt{n}poly(\log n))$ 作为块大小来处理问题。本节将介绍几道以 $O(\sqrt{n}poly(\log n))$ 为块大小的例题来帮助读者熟悉树分块的应用方式。

例题 1. 带 *link*，*cut* 操作的树上第 k 小值相关信息查询[3]

要求维护一个 n 个点的有根森林，每个点都有一个点权，要求支持 q 次操作，每次操作形如

1. 加入一条边
2. 删除一条边
3. 给定常数 c，对一条链上的所有点，将其点权加等于 c。
4. 对于指定的根和给定的常数 c，对某点子树中的所有点，将其点权加等于 c。
5. 给定常数 c，对一条链上的所有点，求出其上点权不超过 c 的点的数量
6. 对于指定的根和给定的常数 c，对一个子树内的所有点，求出点权不超过 c 的点的数量
7. 给定常数 k，对一条链上的所有点，求出这些点权的第 k 小值
8. 对于指定的根和给定的常数 k，对一个子树内的所有点，求出这些点权的第 k 小值

这里认为 n, q 同阶。

关于树结构的维护，如前所述，我们可以以单次 $O(B)$ 的复杂度动态维护块大小为 B 的树分块结构，后面将会分析 B 的最优取值。

关于点权信息的维护，在每个簇中，我们维护簇路径上的点权构成的有序数组，以及所有簇内点的点权构成的有序数组，还有修改操作带来的加法标记。

执行修改操作（同时包括点权修改和树形态修改）时。对于端点所处的簇，我们暴力重构的有序数组。为了避免复杂度增加 log 因子，我们可以使用归并来代替重新排序。这一步复杂度 $O(B)$。对于其它簇，只需修改加法标记。对于涉及到的界点，由于数量有上界，可以暴力处理。这一步复杂度 $O(\frac{n}{B})$。

执行查询操作时，对于端点所处的簇和界点，我们可以暴力提取出涉及到的点的点权。这样，问题就变成了，在多个有序数组中计算常数 c 的排名，或者寻找第 k 小值。这两个子问题均可做到 $O(\frac{n}{B}\log n)$ 的复杂度。对于前者，在每个数组中二分查找即可。至于后者的做法，则可以在参考文献 [2] 中查阅。这一步复杂度为 $O(B+\frac{n}{B}\log n)$。

综上，取 $B = O(\sqrt{n\log n})$ 时本解法有最优复杂度 $O(n\sqrt{n\log n})$)。

实际上本题的弱化版已多次在算法竞赛中出现。

Rujia Liu loves Wario Land![4] 即为本题去掉操作 2,4,6,7,8 的版本，并且操作 3 有额外的特殊性质。

Gty 的超级妹子树[5] 即为本题去掉操作 4,5,7,8 的版本，并且操作 1,3 有额外的特殊性质。

[3]题目来源：经典问题

[4]题目可以在 https://vjudge.net/problem/UVA-11998 查看

[5]题目可以在 https://www.luogu.com.cn/problem/P2166 查看

带 Link、Cut 树上路径 k 小值[6] 即为本题去掉操作 5,6 的版本，并且出题人给出的参考算法复杂度为 $O(n\sqrt{n}\log n)$，不够优秀。

May Holidays[7] 即为本题去掉操作 1,2,4,5,7,8 的版本，并且修改操作 3 中保证常数 $c = 1$，询问操作 4,5 中保证常数 $c = 0$。不过由于修改的特殊性质，本题可以做到更优复杂度，在下一个例题中我们将进行一些探讨。

例题 2. *Simple Tree*[8]

给定一棵 n 个点的有根树，点有点权。

现在有 q 次操作，操作有 3 种：

1. 给定 x, y, w，将 x 到 y 的路径上的点点权加上 w (其中 $w \in 1, -1$)

2. 给定 x, y，询问在 x 到 y 的路径上有多少个点点权 > 0

3. 给定 x，询问在 x 的子树里的点有多少个点点权 > 0

实际上本题即为 *May Holidays* 一题增加操作 5 的版本，并且要求复杂度 $O(n\sqrt{n})$。这里认为 n, q 同阶。

我们沿用上一个例题的做法，不过还需要一些改动。

由于询问操作的特殊性，我们对于每个有序数组额外维护一个指针指向第一个大于 0 的位置。修改的时候只需移动指针即可。

由于单次移动指针需要做到 $O(1)$ 复杂度，而重复元素过多会影响暴力移动指针的复杂度。对此，我们为每个簇的有序数组中的每个元素额外维护一个指针，表示它加减 1 后指向的新元素位置。

这样，每次修改操作时我们只需 $O(1)$ 维护有序数组中第一个大于 0 的位置，即可省去查询时的二分查找，取 $B = O(\sqrt{n})$ 即可做到 $O(n\sqrt{n})$ 的复杂度，与出题人的官方做法[9]具有相同复杂度。

例题 3. *[Ynoi2009] rpdq*[10] 给定一棵 n 个点的树和 q 次询问。每次询问给出 l, r，要求回答 $\sum_{i=l}^{r}\sum_{j=i+1}^{r} dis(i, j)$，其中 $dis(a, b)$ 表示编号为 a 和编号为 b 的点在树上的距离。要求复杂度 $O(n\sqrt{n})$，这里认为 n, q 同阶。

我们将 1 号点提根，则 $dis(a, b) = dep_a + dep_b - 2dep_{lca(a,b)}$，这样问题可以转为计算 $\sum_{i=l}^{r}\sum_{j=i+1}^{r} dep_{lca(i,j)}$。

一个自然的想法是，运用莫队算法计算上式的值，将问题转为动态维护一个集合，支持

[6]题目可以在 https://immortalco.blog.uoj.ac/blog/670 查看

[7]题目可以在 https://codeforces.com/contest/966/problem/E 查看

[8]题目可以在 https://uoj.ac/problem/435 查看

[9]可以在 这里 查看

[10]题目可以在 https://www.luogu.com.cn/problem/P6778 查看

1. 加入一个元素

2. 删除一个元素

3. 维护所有元素两两之间 lca 的深度和

这是经典问题，加入/删除元素时我们将该点到根的链上所有点的权都加/减 1，再求出该点到根的链上所有点的权和，即可更新答案。

链加链求和可以通过树链剖分来维护，这样我们就得到了一个 $O(n\sqrt{n}\log^2 n)$ 的做法。使用全局平衡二叉树[11]可以将复杂度优化至 $O(n\sqrt{n}\log n)$，不过仍然不足以通过此题，还需要一些优化。

借助莫队二次离线的方法[12]，我们可以将链加的次数减少到 $O(n)$ 次，但是询问链和的次数仍然是 $O(n\sqrt{n})$。

询问和修改次数的不平衡为我们提供了一个切入口。序列问题中我们常常使用序列分块来平衡复杂度，类似的，这里我们可以使用树分块。

具体的，修改一个点后，我们可以在收缩树上暴力修改点权，并进行一次树上前缀和，这样就正确更新了所有界点的答案。同时，对于修改点所属的簇内部，我们也要进行一次树上前缀和，以便更新簇内点之间的贡献。由于原树边和收缩树的边都带权，实现的时候要注意细节。

这样，单次修改复杂度即为 $O(B+\frac{n}{B})$，查询复杂度显然可以做到 $O(1)$。由此，我们以 $O(n\sqrt{n})$ 复杂度解决了此题。

例题 4. *[Ynoi2018] 駄作*[13]

给定一棵 n 个点的树和 q 次询问。

每次询问给出参数 p_0, d_0, p_1, d_1，记 $S_i = \{u | dis(u, p_i) \le d_i\}$，其中 $i \in \{0, 1\}$。

要求回答 $\sum_{a\in S_0}\sum_{b\in S_1} dis(a,b)$，其中 $dis(a,b)$ 表示编号为 a 和编号为 b 的点在树上的距离。

要求复杂度 $O(n\sqrt{n})$，这里认为 n, q 同阶。

本题处理起来比较复杂，我们先从一些相对简单的子问题入手。

子问题 1：所有询问满足 $S_0 \cap S_1 = \varnothing$

解法则存在一个点 c，使得对于 $\forall a \in S_0, b \in S_1$，$a, b$ 间的路径经过 c，故而只需统计 S_0, S_1 的点数以及到 c 的距离和。

子问题 2：$q = 1$

解法仿照上一道例题，我们将 $dis(a,b)$ 表示为 $dep_a + dep_b - 2dep_{lca(a,b)}$。

[11] 可以在参考文献 [3] 中查阅

[12] 可以在参考文献 [2] 中查阅

[13] 题目可以在 https://www.luogu.com.cn/problem/P5399 查看

这样我们只需求求两两点间 lca 的深度和。将所有 $a \in S_0$ 到根的路径上边权加上 1，求所有 $b \in S_1$ 到根的路径的边权和。时间复杂度 $O(n)$ 。

子问题 3：所有询问中的 p_0, p_1 均相等，但是 d_0, d_1 可能不同

解法枚举 d_0 的取值，类似情况 2 处理，处理出所有可能的询问的答案。之后还需要进行一次二维前缀和，时间复杂度 $O(n^2)$。

在解决完子问题后，我们会到原题。设定阈值 B 后，我们使用 2.3 中提到的构建算法求出原树的一组簇划分。

则原树的一个邻域[14] 可以拆成每个簇中的邻域，且除了中心所在的簇，其余簇的邻域均以簇的界点为中心。

每次询问时，对于不同块中的邻域之间的距离和，可以在收缩树上进行树形 DP 统计，类似于情况 1。每次询问需要 $O(\frac{n}{B})$ 的时间。

对于同个块内两个以界点为中心的邻域间的距离和，每次询问在每个块中产生至多 1 个询问，可以最后离线计算，类似于情况 3。每个块需要 $O(B^2)$ 时间。

对于同个块内的两个邻域，如果至少一个邻域的中心不在界点上的情况，每次询问至多出现 2 次，类似于情况 2 处理。每次询问需要 $O(B)$ 时间。

这样总复杂度为 $O(nB + \frac{n^2}{B})$，取 $B = \sqrt{n}$ 时有最优复杂度 $O(n\sqrt{n})$。

点评 本题所涉及的信息询问结构非常复杂，但最后借助树分块对于树的结构分解，我们还是较好地解决了本题，体现了基于 top cluster 剖分的树分块算法的优势。特别是子问题 3 中，如果不是簇划分分强制了界点数量至多为 2，那么进行前缀和运算的维数可能就不只 2 了，这也体现了伪簇划分在信息维护上的缺点。

例题 5. *[Ynoi2014] 等这场战争结束之后*[15]

给定一张 n 个点的图，每个点有点权，最开始没有边。之后需要进行 q 次操作，每次形如：

1. 添加一条 x 与 y 之间的双向边。
2. 回到第 x 次操作后的状态（注意这里的 x 可以是 0，即回到初始状态）。
3. 查询 x 所在联通块中点权第 y 小的值。

这里认为 n, q 同阶。

本题做法比较多，这里仅介绍一种常数比较优秀的 $O(n\sqrt{n})$ 做法。

首先我们可以使用离线算法，对于所有询问按照一定顺序建成操作树。

[14]对于点集 S，如果存在点 p 和距离 d 使得 $S = \{x | dis(x, p) \le d\}$，则称 S 为原树的一个邻域

[15]题目可以在 https://www.luogu.com.cn/problem/P5064 查看

然后在操作树上 dfs，同时维护并查集用于判断每次加边操作是否有效（x 和 y 是否属于同一个连通块）。由于需要支持回溯操作，所以并查集需要按秩合并，这一步复杂度 $O(n\log n)$。

通过离散化的方法可以将点权值域压缩为 $O(n)$。

我们设定阈值 $B=\sqrt{n}$，将值域切成 $\sqrt{n}$ 块，并对每组询问求出答案所属的值域块。具体的，我们需要对每个值域块都在操作树上进行一次 dfs，同时维护每个点所属连通块中，属于该值域块的点数。这一步复杂度为 $O(n\sqrt{n})$。

求出答案所属值域块后，我们还需要进一步求出答案的精确值。对此，我们以 $B=\sqrt{n}$ 为阈值将操作树进行分块。对于每个界点，我们可以 $O(n)$ 处理出该时刻整张图的连通信息，再以该界点为基础回答相邻簇的询问。

具体的，对于相邻簇的所有内点，其到界点的路径上至多只有 $O(\sqrt{n})$ 次加边操作，我们将这些加边操作涉及到的点取出来执行 bfs，就可以以 $O(\sqrt{n})$ 复杂度得到该内点处的每个点所属的连通块，之后再从小到大枚举值域块中的元素计算答案即可。

最后时间复杂度为 $O(n\sqrt{n})$，空间复杂度为 $O(n)$，常数较小。

点评 先前大家对定期重构的认识大多停留在序列或时间轴的线性结构上。这里借助树分块的技术将定期重构推广到了树上，从而得到了一个复杂度和常数都比之前算法更优秀的做法。这种推广的思路值得借鉴。

3 树分块算法在一些其它问题上的应用

序列分块的块大小并不总是 $O(\sqrt{n}poly(\log n))$，在一些特殊问题上特殊的块大小常常能出奇制胜，树分块也是。这里将通过一些例题来进行介绍。

例题 6. 线性树上并查集[16]

树上并查集是并查集的一个特殊情况：给定一棵树，每次操作形如将一个节点合并到父亲，每次询问形如查询一个点已合并的祖先。这里要求复杂度线性。

朴素的并查集算法[17]复杂度会带上 $\alpha(n)$ 的因子，不能满足我们的需求。

首先，我们以 $B=\lfloor\log n\rfloor$ 为阈值将原树分块。

对于每个簇内部，我们沿 dfn 序递推求出每个点到簇界点的点集，以一个 B 位的整数形式进行存储。为了后文处理方便，这里需要保证点按深度顺序递增存储。

修改的时候，对于每个簇我们维护已被合并至父亲的点集，同样以一个 B 位的整数来存储。对于收缩树，由于点数只有 $O(\frac{n}{B})$，我们可以直接维护并查集。这部分复杂度为 $O(\frac{n}{B}\alpha(n)+q)=O(n+q)$。

[16]题目来源：经典问题

[17]可以在参考文献 [4] 的第 21 章查阅

查询的时候，如果当前点尚未与所处簇的界点合并，则只需求出该点簇内祖先与簇内未合并点的交集中深度最大的点。这可以通过整数的按位与和 hightbit 查询来解决。由于所有涉及到的整数都不超过 $2^B = O(n)$，所以 highbit 可以提前预处理。如果当前点已与所处簇的界点合并，则在收缩树的并查集上查询即可。

最后总复杂度为 $O(n+q)$，达到了线性的目标。

例题 7. 「十二省联考 *2019*」希望[18] *给定一棵 n 个点的树，每条边长度均为 1。求有多少个连通块满足，存在一个点，使得该连通块内的所有点到该点距离均不超过 L。复杂度要求 $O(n)$。*

记全集 $A = \{1, 2, \cdots, n\}$。定义 $f(x, S)$ 为一个数组，其中 x 是原树中的一个点，S 是一个点集，$f(x,S)_i$ 定义为 $|\{T|\text{T 是连通点集}, x \in T, T \subset S, \forall y \in T \text{有 dis(y,x)} \le i\}|$。

通过简单的容斥技巧，可以将问题转为对于所有有序二元组 (x,y)，其中 x 和 y 在原树中相邻，计算 $f(x, A \setminus y)_{L-1}$。

对此，我们以 L 为阈值将原树分块，对于第 i 个簇，我们记簇的内点集合为 I_i，通过长链剖分优化树上 DP，可以在 $O(L)$ 的复杂度内对簇的两个界点 b_0, b_1 求出 $f(b_0, I_i \cup \{b_0, b_1\})$ 和 $f(b_1, I_i \cup \{b_0, b_1\})$。由于与本文主题无关，长链剖分优化 DP 的具体细节就不在这里详细展开了。

通过收缩树上的简单 DP，我们可以对簇 i 的两个界点 b_0, b_1，求出 $f(b_0, T \setminus I_i)$ 和 $f(b_1, T \setminus I_i)$。

具体实现的时候，建议按照第一节中提到的簇的两种合并方式，rake 和 compress，相应地封装 f 数组的合并方式。

这里涉及到的关于 f 数组的运算均可在 $O(L)$ 的复杂度内完成，而总运算次数为 $O(\frac{n}{L})$，所以这部分总复杂度为 $O(n)$。

最后再通过一些简单的计算即可求出每个有序二元组的权值，进而求得答案。最后总复杂度 $O(n)$。

点评 本题的官方做法是做两遍长链剖分，一遍自底向上，一遍自顶向下，细节繁多且需要计算模意义下的乘法逆元。相比较而言本做法理解起来细节量更少，而且无需求逆，天然可推广至模数为合数的情况。

总结

在目前国内的 OI 竞赛中，出现过的树分块相关问题数量偏少，这与之前已有的树分块算法结构繁复、适用面窄有关。对此本文第一节引入 top cluster 的概念，介绍了一种新的对

[18]题目可以在 https://loj.ac/p/3053 查看

树分块的思路，并针对静态问题和动态问题分别提供了一种构造维护树分块形态的算法，希望能够增加树分块算法的普及程度。

先前 OI 中的树分块相关问题多是借助通用化的手段，用序列问题的手法来处理。对此本文第二节通过数个例题展示了这种树分块算法在一些相关的问题上的优越性。其中例题 1 和例题 2 来源于对之前一系列类似问题的总结归纳。相较于先前依赖弱化问题具体性质的解法而言，这里给出的算法通用性更强，流程上也更加清晰。例题 3,4,5 则展示了在一些其它问题中这类树分块算法的应用。

本文第三节则列举了两个使用了非常规块大小的例题，警示我们有些时候除了分块之后的流程，分块大小的选取本身也是重要的，要具体问题具体分析。不过非常规大小树分块在当前 OI 竞赛中的出现频率还非常低，其应用还有待进一步发掘。

希望本文能起到抛砖引玉的作用，对 OI 中相关的树上的问题产生更多的启发。

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。

感谢父母对我的培养和教育。

感谢学校的栽培，李建老师和施琴儿老师的教导和同学们的帮助。

感谢李欣隆学长、蔡承泽学长与我交流讨论、给我启发。

感谢胡杨同学、吴与伦同学为本文审稿。

感谢其它所有本文所参考过的资料的提供者。

感谢各位百忙之中抽出宝贵的时间来阅读本文。

参考文献

[1] 赵雨扬.《Top tree 相关东西的理论、用法和实现》, 2019, https://negiizhao.blog.uoj.ac/blog/4912

[2] 李欣隆, 蔡承泽. 《信息学竞赛中的抽象数据结构》. 2021.

[3] 张哲宇. 《浅谈树上分治算法》，IOI2019 中国国家队候选队员论文集。

[4] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein. 算法导论 [M]. 第 3 版. 机械工业出版社, 2009.