数据结构问题选讲——动态图连通性

黄洛天

IIIS, THU

December 23, 2024

碎碎念

选完题目才发现我只有三个小时（
于是就从题目选讲变成选一个题目讲了（
考虑到我没有科研经验，所以可能会比较民科（

Acknowledgments

感谢段然教授在《算法分析与设计》课中的教导。

Problem

边修改

你需要维护一个图有 n 个点，初始图为空。维护一个数据结构需要支持插入一条边，删除一条边，查询两个点是否联通。

点修改

初始有一个图有 n 个点 m 条边，每个点有开和关两种状态。维护一个数据结构需要支持修改一点的开关状态的，查询只保留开点时两个点是否联通。

强制在线。

Related work

边修改:

- Amortized: $O(\log^2 n)$ [1], $O(\frac{\log^2 n}{\log\log n})$[2].
- Amortized Randomized: $O(\log n(\log\log n)^2)$ [3].
- Worst-Case: $O(\sqrt{m})$[4], $O(\sqrt{n})$[5].
- Worst-Case Randomized: $O(\log^5 n)$[6], $O(\log^4 n)$ [7].

Related work

点修改:

- Amortized : $\tilde{O}(m^{\frac{2}{3}})$, with query time $\tilde{O}(m^{\frac{1}{3}})$[8].
- Worst-Case : $\tilde{O}(m^{\frac{4}{5}})$, with query time $\tilde{O}(m^{\frac{1}{5}})$[9].
- Worst-Case Randomized: $\tilde{O}(m^{\frac{3}{4}})$, with query time$\tilde{O}(m^{\frac{1}{4}})$[10].

Preliminaries

在此之前，先看看前置知识吧！
Splay, ET-Tree, Link/Cut-Tree, ST-Ttree.[11]
ST-Tree 是不均摊的 Link/Cut-Tree。
虽说是前置知识，但你把他当作黑盒也完全没有问题。

Basic idea

维护一个生成森林 F。
insert(u, v)： 如果 u, v 不联通，就把两个树连上。
delete(u, v)： 如果 (u, v) 是一条树边，那么就把树边断了。

Basic idea

为什么这个东西不对呢？

delete(u, v) 里删除一条树边之后，树的两部分可能被在此之前插入的其他边连起来。

Holm, Lichtenberg & Thorup’s structure

- 每条边有一个等级 $0 \le l_e \le l_{\max}$。
- $E_i = \{e : l(e) \ge i\}$。
- F_i 是 E_i 的一个生成森林，满足 $F = F_0 \supseteq F_1 \supseteq \cdots \supseteq F_{l_{\max}}$。
- F_i 中每个连通块的大小不超过 $\frac{n}{2^i}$。（这意味着 $l_{\max} = O(\log n)$）。
- 每条边的等级 l 只会增加，不会减少。

Algorithm - insert

令 $l(e) = 0$，也就是把他插入到 E_0 中，若 F_0 中 u, v 不联通则连上，否则什么都不干。

Algorithm - delete

若 $e \notin F_0$，什么都不做即可。

令 $k = \max\{k : e \in F_k\}$，把 $F_0 \sim F_k$ 中的边 e 全部删除。设分裂出来的两个连通块分别是 $T(u), T(v)$。

首先尝试在 E_k 里找边 e'，使得 e' 连接 $T(u)$ 和 $T(v)$。不妨假设 $|T(u)| \le |T(v)|$，则 $|T(u)| \le \frac{n}{2^{k+1}}$，即就算把 $T(u)$ 里的所有 i 级边全都升级为 $i+1$ 级，也仍然满足条件。

Algorithm - delete

考虑枚举至少一个端点在 $T(u)$ 中的 k 级边 e'。

- 如果 $e' = (u', v') \in (T(u), T(u))$，则把 e' 升级为 $k+1$ 级边。
- 如果 $e' = (u', v') \in (T(u), T(v))$，则说明成功找到了 e'，立刻停止检索（否则可能会找到很多此类边，而耽误了复杂度）。

注意到 E_i 中的边不会从 $T(u)$ 连到 $V - T(u) \cup T(v)$，否则 F_i 不会是生成树。

Algorithm - delete

如果我们用 $O(t\log n)$ 的时间找到 t 条边，则我们至少可以升级 $t-1$ 条边。由于可能 $t \ll |T(u)|$，需要维护一棵树中哪些节点有 i 级边相邻，这里需要用 ET-Tree 维护一下。

如果找完了所有边都没出现第二类边，那么可以断言 F_k 中 $T(u)$ 和 $T(v)$ 是连不回来了，此时需要继续考虑 F_{k-1} 中是否可以连回来。沿用刚刚的思路即可。直到 F_0 中都连不会来就彻底连不回来了。

Algorithm - query

查询操作只需要查询一下 F_0 中两点是否在一个连通块里即可。

Running time analysis

一条边只会升级 $O(\log n)$ 次，每次升级会做一次 link 操作。删除一条边会在 $O(\log n)$ 个树上删除，总共使用了 $O(\log n)$ 次 link/cut 操作。

Link/Cut-Tree 完成一个操作均摊 $O(\log n)$ 时间，因此每次修改均摊时间复杂度为 $O(\log^2 n)$。

查询时间 $O(\log n)$。

Running time analysis

一条边只会升级 $O(\log n)$ 次，每次升级会做一次 link 操作。删除一条边会在 $O(\log n)$ 个树上删除，总共使用了 $O(\log n)$ 次 link/cut 操作。

Link/Cut-Tree 完成一个操作均摊 $O(\log n)$ 时间，因此每次修改均摊时间复杂度为 $O(\log^2 n)$。

查询时间 $O(\log n)$。

查询并不需要使用每个 ET-Tree，其实只需要维护 F_0 的 ET-Tree。由于这个 ET-Tree 均摊 $O(1)$ 次修改，因此可以改成 $\log n$ 叉 ET-Tree。其中单次修改 $O(\frac{\log^2 n}{\log\log n})$，查询 $O(\frac{\log n}{\log\log n})$。

不过这里的修改不是瓶颈，一次加边删边代价仍然为 $O(\log^2 n)$。

Cutset problem

有一个图 $G=(V,E)$，其中有一个森林 $F\subseteq E$，每次操作会把一条边从树边变成非树边（对应删边），或者把非树边变成树边（对应加边）。对于每个 $T\in F$ 你需要维护任意一条边 $e\in(T,V-T)$（如果存在）。

保证修改与你返回的边无关。或者可以说修改序列在最开始就决定好了，只是每次只告诉你一个操作。

Cutset data structure

先考虑一个简单情况：$|cut(T, V - T)| = 1$。

对于每个点随机一个长度为 $2\lceil \log n \rceil$ 的二进制串 $w(u)$ 作为权值。令边 (u, v) 的权值 $w(u, v)$ 为 $w(u)$ 和 $w(v)$ 的拼接。令 $xor(u)$ 表示与 u 相连的边的权值的异或和。则

$$\bigoplus_{u \in T} xor(u) = \bigoplus_{(u,v) \in (T, V-T)} w(u, v)$$

这是由于 $w(u, v)$ 在 $xor(u)$ 和 $xor(v)$ 各贡献了一次，就抵消了。因此我们只需要求出左式，找到与之对应边的即可。

Cutset data structure

当 $|cut(T, T-V)|$ 不确定时可能会有以下几种情况。

- T 向外没有连边，此时 $xor(T)=0$。
- T 向外有一条连边 e，此时 $xor(T)=w(e)$。
- T 向外有若干条连边，但我们并不能分辨是哪条连边。

考虑维护 $2\lceil\log n\rceil$ 个上述结构，在第 i 个结构里一条边有 2^{-i} 的概率出现，记找到的边集为 $cut_i(T, V-T)$。存在一个 i 使得第 i 个结构里，我们找到了恰好一条边的概率为常数，可以计算其至少为 $\frac{1}{9}$。

但这并不能满意，因此我们需要维护 $c\log n$ 套结构，每套至少 $\frac{1}{9}$ 的概率找到，则总共有至少 $n^{-0.17c}$ 的概率找到一条边（如果存在）。

Cutset data structure

查询时我们需要分清三类情况，如果每条边的边权是随机二进制串，我们可以用哈希表很简单的判断，但这需要 $O(m)$ 的额外空间（m 为操作数量）。

Cutset data structure

为了规避这个额外空间，我们可以设计一个校验器 (B, W)，其中 B 和 W 都是一个 01 变量，初始为 0。对于每条边 e，我们把他随机地分配一个颜色和一个随机 01 权值 x。若为黑色则 $B \leftarrow B+x$，否则 $W \leftarrow W+x$。若有至少两条边，则其有 $\frac{1}{2}$ 的概率使其有不同的颜色，因此 B 和 W 可以被表示为至少一个随机变量的和，因此分别有 $\frac{1}{2}$ 的概率为 1，总计有 $\frac{1}{8}$ 的概率满足 $B = W = 1$，而只有一条边或没有边时不可能出现这种情况。因此考虑维护 $c \log n$ 组校验器，若存在一组 $(B, W) = (1, 1)$ 则判定为有至少两条边。

Cutset data structure

考虑用 ET-Tree 维护 F，每个节点上维护 s_{ij} 表示在第 i 套结构中以 2^{-j} 为概率采样得到的异或和，B_{ijk}, W_{ijk} 表示第 i 套结构中以 2^{-j} 为概率采样中第 k 个校验器的值，信息合并的代价为 $O(\log^2 n)$。每次查询 $\bigoplus_{u\in T} xor(u)$ 只需要在 ET-Tree 查询信息即可，时间复杂度 $O(\log^3 n)$。

Why cutset data structure does not work?

用 cutset data structure 维护所有非树边，每次删除一条树边时就找到一条非树边拿出来当作树边。

看起来我们几乎做完了这个问题!

但不要忘了，题目假设中的“保证修改与你返回的边无关”。

Why cutset data structure does not work?

用 cutset data structure 维护所有非树边，每次删除一条树边时就找到一条非树边拿出来当作树边。

看起来我们几乎做完了这个问题！

但不要忘了，题目假设中的“保证修改与你返回的边无关”。

实际问题当中，被你选出来的边 e 会成为下一次 link 的对象，这与“保证修改与你返回的边无关”矛盾了！

这会导致什么问题呢？

Why cutset data structure does not work?

让我们举一个极端例子，假设交互库知道你每次找到的新边 (u,v)。那么交互库每次就删你新加的边，这样会导致 T 永远不变，而在第 i 套结构中，会找到 $|cut_j(T,V-T)|=1$ 中的边，然后由于这条边变成树边了，因此删除这条边就会使得 $|cut_j(T,V-T)|$ 变成 0。

由此可见，对于第 i 套系统，有 $\Omega(\frac{1}{c\log n})$ 删除的边都集中在很“关键”的 $|cut_j(T,V-T)|$ 很小的 $cut_j(T,V-T)$！这会很快的导致你的分布不再正确，进而导致 $|cut_j(T,V-T)|$ 会有一个断档，一下降到 0。

Why cutset data structure does not work?

究其本质，其实是因为删边加边方案与自身的随机结果有关了。因此文章的关键点就是如何去掉自己对自己的影响。

Kapron, King & Mountjoy's structure

考虑 Boruvka Algorithm 最小生成树算法，每轮会把若干连通块合并成一个连通块。由此，我们引入 Boruvka 重构树的概念。称 F_i 表示第 i 轮后 Boruvka 找到的森林，则我们有 $F_i \subseteq F_{i+1}$。

对于第 i 版本中的一个树 $T \in F_i$，我们把它视为一个重构树种的节点，其子节点是 F_{i-1} 合并成 T 的若干棵树对应的节点。此节点代表了一个边集合即合并子节点用到的边集合，边集合中的边具有级别 i。

对于一个重构树中的节点 u，若其没有兄弟节点，我们称其是孤立的。对于每个 i 我们都建立一个 cutset data structure DS_i 来维护 F_i。

Kapron, King & Mountjoy's structure

不同于 Boruvka Algorithm 会选择边权最小的点在 F_i 中，本算法对每一层维护一个 cutset data structure，通过第 0 层到第 i 层的 cutset data structure 去决策哪些边会被加入到 F_{i+1}，这样 F_{i+1} 的加边删边操作就和第 $i+1$ 层及以上的随机结果无关了。

Kapron, King & Mountjoy's structure

我们维护的 (F_i) 只需要满足以下五个性质:

- Boruvka 重构树第 0 层是 n 个原图中的节点，即 $F_0 = \emptyset$。
- $F_i \subseteq F_{i+1}$ 且 Boruvka 重构树 $i+1$ 层的节点是 i 层节点由 $i+1$ 级边连接而得。
- 第 i 层的节点 u 如果是孤立点，则 u 一定代表了一个原图的连通块。
- 第 i 层的森林结构与第 i 层及更高层的随机结果无关。
- $F_{\max}$ 构成了 G 的生成森林。

Algorithm - insert

若 u, v 在 $F_{\max}$ 中不连通，则在 $F_i (0 < i \leq \max)$ 中加入 e。反之，什么都不做。

Algorithm - delete

若 u, v 不在 $F_{\max}$ 中，则什么都不做。

否则，令 $d = \min\{i : e \in F_i\}$，并找到包含 e 的节点 u。在 $F_i(d \le i \le \max)$ 中删除 e，这样 u 及 u 的每个祖先都会拆成两部分，这些点可能会违反性质 3。

设 u 分裂成了 x, y，下面只考虑 x 祖先，另一侧是对称的。

Algorithm - delete

我们只考虑维持 x 祖先的性质 (3)，另一侧是类似的。

考虑 x 的层数最小的祖先 A, 满足 A 违反了性质 (3)。设 A 在第 k 层，我们用第 k 层的 cutset data structure 去决策一条边 e 使得连接 A 和 $V-A$，并把 e 加入到 $F_{k+1} \sim F_{\max}$。若加入 e 后成环了，则找到环上等级最大的边 e'，设其等级为 k'，把 e' 从 $F_{k'} \sim F_{\max}$ 中删掉。

完成上述步骤后 A 会符合性质 3，然后继续考虑 x 的不满足性质 3 的祖先即可。

Algorithm - delete

上述过程很抽象，让我们来分类讨论一下他为什么让 A 满足了性质 3。

不妨设 e 连接了第 k 层的节点 A 和节点 B。那连接 e 相当于把 fa_A 和 fa_B 揉成一个点，并把它们祖先揉一起，这样 A 就有了 B 这个兄弟。

Algorithm - delete

如果 A 和 B 不联通，我们不需要额外调整。但如果 A 和 B 原本就会在第 k' 层合并在一起，这样就连出了环。于是我们需要找到环上等级最大的边，即 Boruvka 重构树上 A 和 B 的最近公共祖先 C 包含的边。我们把这条边删掉后会导致 C 的两个包含 A 的儿子 A' 和包含 B 的儿子 B' 合并成同一个，因此可能导致在 $k'-1$ 层出现一个不符合性质 3 的点。不过注意到 $k' > k+1$，因此层数最小的不符合性质 3 的点上移了，这样只用 $O(\log)$ 次调整即可满足性质 3。

Algorithm - query

查询只需要在 $F_{\max}$ 里查询一下两点是否在一个连通块里即可。

Running time analysis

单次调用 cutset data strucure 是 $O(\log^3 n)$ 我们已经分析过。

- insert 最多需要 $O(\log n)$ 次修改，时间复杂度 $O(\log^4 n)$。
- delete 包含 $O(\log n)$ 次调整使 x 的一个祖先符合性质 3。每次调整需要包含 $O(\log n)$ 次修改，时间复杂度 $O(\log^5 n)$。
- query 时间复杂度 $O(\frac{\log n}{\log\log n})$。

需要注意的是，我们无法真正的维护出 Boruvka 重构树，因此我们分裂节点时并不知道儿子们应该怎么分裂。我们需要用 ST-Tree 维护 $F_{\max}$，并且令每条边的边权为其等级，这样环上等级最大的边可以在 ST-Tree 上查询，其余信息可以在每层分别维护的 ET-Tree 上查询。

Improve

我们可以使用“假”的 cutset data structure。每层的 cutset data structure 只维护一套结构，这样可以保证正确率至少为 $\frac{1}{9}$。把前文的性质 3 改为：若 u 没有兄弟节点，则 $DS_i.search(u)$ 一定失败。对于一个连通块，随着层数的加深，期望节点数量会指数级降低。可以简单的用概率方法证明，我们仍然只需要保存 $O(\log n)$ 层。

Chan, Pâtraşcu & Roditty’s structure

下文用 $\tilde{O}$ 记号省略 $\log$ 因子。

本作法类似于 OI 中的定期重构。

令 P 为上次重构时状态为开的点集合。

对于每次操作修改了 u 的状态，如果 $u \in P$ 就把 u 从 P 中移除。无论 u 是否在 Q 中，都向 Q 中加入 u。每当 $|Q| = m^{\frac{2}{3}}$ 时重构，并清空 Q。

对于 P 的导出子图的一个联通分量 C，若 $\sum_{u \in C} \deg(u) > m^{\frac{1}{3}}$ 就称它为一个大连通块，反之称其为一个小连通块。

我们新建一个图 G'，G' 的点集包含 Q 和大连通块，用前文的任意一个数据结构维护 G' 即可在 $\tilde{O}(1)$ 时间内修改边的状态和查询连通性。

Algorithm - Preprocessing

把 Q 中的点当特殊点，我们暴力连出来他们之间的边，这种边共 $O(m)$ 条。

对于每个大连通块建一个点表示这个连通块。若特殊点 u 与这个大连通块有连边，则给 u 和代表大连通块的点连边，这种边共 $O(m^{\frac{4}{3}})$ 条。

对于每个小连通块 C，其最多连向 $m^{\frac{1}{3}}$ 个特殊点。给这些特殊点之间两两连边，这种边最多 $O(m^{\frac{4}{3}})$ 条。

Algorithm - query

若 u, v 均不属于小连通块，则我们已经在 G' 中维护了他们之间的连通性，时间复杂度 $\tilde{O}(1)$。

若 u 和 v 属于同一个小连通块，则联通。

若 u 属于小连通块，我们可以暴力找到与小连通块相连的任意状态为开的特殊点 u'。若不存在则不连通。同理找到 v'，判断 G' 中 u' 和 v' 是否联通即可。

Algorithm - update

考虑一个操作修改了 u 的状态。

若 $u \in Q$，则我们只需要暴力修改所有与 u 有关的 $O(m^{\frac{2}{3}})$ 条边。

若 $u \notin P$，说明 u 原本是暗的，需要把 u 加入 Q，暴力修改与 u 有关的 $O(m^{\frac{2}{3}})$ 条边。

若 $u \in P$，首先我们需要把 u 所在的连通块分裂成若干小连通块，然后重新维护 G'。

Algorithm - update

若 $u \in P$，首先我们需要把 u 所在的连通块分裂成若干连通块，然后重新维护 G'

同样用前文数据结构维护原图 G，暴力把与 u 相连的边全部断掉。设连通块 C 分裂成 $C_1, C_2, \cdots, C_k$，这里按照度数和从大到小排序（魔改一下前文的做法 ET-Tree 即可）。

若 C 为大连通块：

设 $C_1, C_2, \cdots, C_t$ 为大连通块。对于 $C_2, C_3, \cdots, C_t$ 新建一个点表示新的大连通块，暴力向特殊点连边。$C_{t+1}, \cdots, C_k$ 成为小连通块后，暴力新建其对应的边。

Algorithm - update

若 $u \in P$，首先我们需要把 u 所在的连通块分裂成若干连通块，然后重新维护 G'

同样用前文数据结构维护原图 G，暴力把与 u 相连的边全部断掉。设连通块 C 分裂成 $C_1, C_2, \cdots, C_k$，这里按照度数和从大到小排序（魔改一下前文的做法 ET-Tree 即可）。

若 C 为小连通块：

则 $C_1, \cdots, C_k$ 均为小连通块。对于所有 $i \neq j$，若特殊点 u 与 C_i 相邻，特殊点 v 与 C_j 相邻，删去一条 u 到 v 之间的边。

Running time analysis

查询中暴力找与小连通块相连的开的特殊点需要 $\tilde{O}(m^{\frac{1}{3}})$，因为小连通块度数就这么多。其余部分都是 $\tilde{O}(1)$，因此总时间复杂度 $\tilde{O}(m^{\frac{1}{3}})$。

修改中，$u \in Q$ 和 $u \notin P$ 两种情况均只需要修改 $O(m^{\frac{2}{3}})$ 条边的状态，因此时间复杂度是 $O(m^{\frac{3}{4}})$。

Running time analysis

对于第三种情况，我们纵观整个重构周期分析。

由于每个点只会从 P 中移除一次，因此暴力删除其周围的边时间复杂度 $\tilde{O}(m)$。

对与每个分裂出来的大连通块 C_i，需要向特殊点 Q 连不超过 $\deg(C_i)$ 条边。由于 $\deg(C_i) \le \frac{1}{2}\deg C$，因此每个度数最多会贡献 $O(\log n)$ 的贡献，这部分时间复杂度 $\tilde{O}(m)$

Running time analysis

每个小连通块 C 第一次在 P 中出现时会连 $\deg(C)^2$ 条边，其中 $\deg(C) \leq m^{\frac{1}{3}}$。由于 $\sum \deg(C) = m$，因此小连通块的连边数量为 $O(m^{\frac{4}{3}})$。

此后拆散小连通块时，只会删边，因此删除的边的数量也不超过 $O(m^{\frac{4}{3}})$。

对于每 $m^{\frac{2}{3}}$ 个操作，需要 $\tilde{O}(m^{\frac{4}{3}})$ 时间，因此均摊 $\tilde{O}(m^{\frac{2}{3}})$。

Application

Problem

有一个 n 个点 m 条带颜色图 G，颜色数量为 k，一个生成树 T 的权值为边权的众数（若有多个取最小的），求最小众数生成树。

有

一个随机算法可以在 $O(n \cdot \text{poly}(k, \log n))$ 时间内解决上述问题，其错误率不超过 $\frac{1}{n^c}$，其中 $c > 0$ 为任意常数。

Preliminaries

- 枚举答案 ans，则所有颜色为 ans 的边能选一定会选，设选了 t 条。
- 把所有选的边组成的连通块缩点。
- 把问题加强为：颜色为 c 的边选不超过 $f(c)$ 条，判定是否可以组成一个生成树。
- 对于每种颜色只需要保留一个生成树即可，因此 $m = O(nk)$。
- 加强后的问题可以描述为求森林拟阵和颜色拟阵的交。

Preliminaries

令 $M_1=(E,I_1)$ 表示颜色拟阵，其限制了颜色为 c 的边的数量不超过 $f(c)$，$M_2=(E,I_2)$ 为森林拟阵。我们的目的就是找到 $M_1\cap M_2$ 的秩。让我们回顾经典拟阵交算法，我们时刻维护了一个边集 F 满足 $F\in I_1\cap I_2$，每轮尝试增广一条边。考虑二分图 $B=(F,E(G)-F,E(B))$，其中 F 和 $E(G)-F$ 分别为二分图的两部分点集，$E(B)$ 为其中的边集合。

Preliminaries

可以作为起点的集合 $S=\{e\in E(G)-F\mid cur(C(e))<f(C(e))\}$，其中 $cur(c)$ 表示 F 中颜色 c 的边的数量。

可以作为终点的集合 $T=\{e\in E(G)-F\mid F\cup\{e\}$ is a forest$\}$。

对于每个 $e\in E(G)-F, e'\in F$，若 $e'\in P(F,e)$，则从 e 向 e' 连一条边，其中 $P(F,\{x,y\})$ 表示 F 中 x 到 y 的路径中边的集合。

若 $C(e)=C(e')$，则从 e' 向 e 连一条边。

Algorithm Overview

容易发现增广时，能放到到的右侧点集合可以被描述为所有颜色属于某个集合的所有边。

因此考虑用小图 G' 代表二分图 B，令：

- $LE_c = \{e|e \in F, C(e) = c\}, RE_c = \{e|c \in E(G) - F, C(e) = c\}$，
- $E_c = LE_c \cup RE_c$，
- 点 e_c 表示 RE_c，
- $G' = (\{e_c \mid c \in [k]\}, E(G'))$。

对于任意两个颜色 c_1, c_2，若存在边 $(e_1, e_2), (e_2, e_3) \in E(B)$ 且 $C(e_1) = c_1, C(e_3) = c_2$，则把边 (e_{c_1}, e_{c_2}) 加入图 $E(G')$。这等价于存在边 $e_1, e_2(C(e_1) = c_1, C(e_2) = c_2)$，且 $e_2 \in P(F, e_1)$。

Algorithm Overview

计算起点集合 S:
时刻维护 $cur(c)$，每次更新边的时候更新 $cur(c)$，
$S = \{e_c \mid cur(c) < f(c)$ and $c \in [k]\}$ 就是起点集合。

Algorithm Overview

计算终点集合 T:

对于每个颜色 c，维护一个边集合 $DE_c = \{\{x,y\} \mid x \text{ and } y \text{ are disconnected in } F \text{ and } e = \{x,y\} \in E_c\}$。每轮增广会导致 F 中两棵树 T_1, T_2 被连接，不妨设 $|T_1| < |T_2|$，则可以枚举所有 $u \in T_1$ 和 $e = (u,v) \in E$，尝试从 $DE_{C(e)}$ 中删除 e。

Algorithm Overview

建图 G':

令 $w(e)$ 为一个随机二进制向量，$c\lceil\log n\rceil$ 维，其中 c 为常数。对于所有 $e\in F$，令 $val_c(e)=\sum_{e'\in RE_c, e\in P(F,e')} w(e')$。

我们使用 Link/CUt-Tree 维护 F 和 $val_c(e)$，具体的维护方式后续会讨论。

对于每个 c_1, c_2 我们需要查询是否存在 $e\in LE_{c_2}$ 使得 $val_{c_1}(e)\neq 0$。若存在，则说明一定存在边 (c_1, c_2)。反之，有至多 $\frac{1}{n^c}$ 的概率存在边 (c_1, c_2)。

Algorithm Overview

增广并更新 F:

设 $c_1, c_2, \cdots, c_t$ 为一条增广路。对于 $1 \leq i < t$，需要找到 $e \in LE_{c_i}, e' \in RE_{c_{i+1}}$，使得 $e \in P(F, e')$。通过 Link/Cut-Tree 的查询操作，我们可以确定 e。我们会在后续中讨论如何使用 dynamic graph connectivity data structure 找到 e'。

找到所有要从 F 中删除和加入的边之后，我们在 LinkCut-Tree 和 dynamic graph connectivity data structure 中更新。

Algorithm Overview

Link/Cut-Tree 需要支持：

- Link(e) ： 加入边 e。
- Cut_and_Link(e_1, e_2) ： 删除边 e_1， 加入边 e_2， 保证 F 的连通性不变。
- Update(e, k) ： 设 $e = x, y$， 对于 $e' \in P(x, y)$， $val(e') \leftarrow val(e') + k$
- query(c_1, c_2) ： 找到一条边 $e \in LE_{c_2}$ 使得 $val_{c_1}(e) \neq 0$， 或判断不存在。
- Get_val(e, c) ： 返回 $val_c(e)$.

Algorithm Overview

对于 Link/Cut-Tree 上每个节点 u，设其代表的边集合为 $E(u)$。令 $E_c(u) = \{e | e \in E(u) and C(e) = c\}$。对于每个 c_1, c_2，显然我们并不能完整的存储 $val_{c_1}(E_{c_2}(u))$。但我们可以找到 $e_1, e_2 \in E_{c_2}(u)$，使得 $val_{c_1}(e_1) \neq val_{c_1}(e_2)$，并只存储四元组 $(e_1, e_2, val_{c_1}(e_1), val_{c_1}(e_2))$。显然两个四元组可以在 $O(1)$ 时间内合并。对于任意二进制向量 a, b, c 由于 $a \neq b$ 等价于 $a + c \neq b + c$，因此修改后并不会改变 $val_{c_1}(e_1) \neq val_{c_1}(e_2)$ 的性质。若不存在 $val_{c_1}(e_1) \neq val_{c_1}(e_2)$，则我们只存二元组 $(e_1, val_{c_1}(e_1))$ 即可，二元组同样可以和二元组或四元组合并。

Algorithm Overview

对于 Query(c_1, c_2) 操作。如果存在 $val_{c_1}(e_1) \neq val_{c_1}(e_2)$，我们会得到一个四元组 $(e_1, e_2, val_{c_1}(e_1), val_{c_2}(e_2))$。$val_{c_1}(e_1) \neq 0$ 和 $val_{c_1}(e_2) \neq 0$ 至少满足其一，返回那个满足条件的即可。反之，所有 $e \in LE_{c_2}$ 的 $val_{c_1}(e)$ 都相同。此时我们会一个二元组 $(e, val_{c_1}(e))$，若 $val_{c_1}(e) \neq 0$ 则返回 e，否则返回 null。

Algorithm Overview

对于 Link_and_Cut(e_1, e_2) 操作，可能会影响 $P(F, e)$，其中 e 为非树边。我们称操作前的森林为 F，操作后的森林为 F'。

Theorem

设 F' 为 F 删除 e_1 加入 e_2 而得到。对于非树边 e，若 $e_1 \in P(F, e)$，则

$$P(F', e) + e_1 = P(F, e) \triangle P(F', e_1)$$

证明画个图就好了。

综上，我们可以在 $O(k^2 \log n)$ 时间内完成一次 Link,Link_and_Cut，Update，Get_val 操作，在 $O(1)$ 时间内完成一次 Query 操作。

Algorithm Overview

当我们知道 e 后找 e':

用动态图连通性数据结构维护 k 个图 $G_1, G_2, \sim G_k$，其中 $G_i = (V(G), F \cup E_i)$。当修改 F 时，我们需要在 k 个图上同时进行修改。

对于一个查询操作，我们首先删除边 e。此时 G_c 中 x 和 y 一定联通 $(e = \{x, y\})$，因为已知存在一条颜色为 c 的边 e' 可以替换 e。假设 F 被删除 e 后分裂为两个连通块分别为 $T(x)$ 和 $T(y)$，在 G_c 中 x 到 y 的路径上节点为 $v_1 = x, v_2, \cdots, v_t = y$。我们的目的就是找到一个 i 使得 $v_i \in T(x), v_{i+1} \in T(y)$，这样 $e' = v_i, v_{i+1}$ 即符合条件。

Algorithm Overview

我们可以在序列上二分，并时刻保证 $v_l \in T(x), v_r \in T(y)$，每次判定 $v_{mid} \in T(x)$ 还是 $v_{mid} \in T(y)$，来决策递归到左侧还是右侧。我们可以把序列的过程扩展到 Link/Cut-Tree 上，这使得我们只需要找到 G_c 中 x 到 y 的链，并在平衡搜索树上二分即可。每次判定 $u \in T(x)$ 还是 $u \in T(y)$ 需要额外的一个 $O(\frac{\log}{\log\log n})$ 因子。在找到 e' 之后需要连回来 e 来还原图 G_c 的结构。

References I

[1] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. "Poly-logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity". In: STOC '98. 1998, pp. 79–89.

[2] Christian Wulff-Nilsen. "Faster deterministic fully-dynamic graph connectivity". In: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium SODA '13. 2013, pp. 1757–1769.

[3] Shang-En Huang et al. "Fully dynamic connectivity in O(log n(log log n)2) amortized expected time". In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium SODA '17. 2017, pp. 510–520.

References II

[4] Greg N. Frederickson. "Data Structures for On-Line Updating of Minimum Spanning Trees, with Applications". In: SIAM J. Comput. 14.4 (1985), pp. 781–798.

[5] David Eppstein et al. "Sparsification—A Technique for Speeding up Dynamic Graph Algorithms". In: J. ACM 44.5 (1997), pp. 669–696.

[6] Bruce M. Kapron, Valerie King, and Ben Mountjoy. "Dynamic Graph Connectivity in Polylogarithmic Worst Case Time". In: SODA '13. 2013, pp. 1131–1142.

[7] David Gibb et al. "Dynamic graph connectivity with improved worst case update time and sublinear space". In: (2015).

References III

[8] Timothy M. Chan, Mihai Patrascu, and Liam Roditty. "Dynamic Connectivity: Connecting to Networks and Geometry". In: FOCS '08. 2008, pp. 95–104.

[9] Ran Duan. "New Data Structures for Subgraph Connectivity". In: ICALP '10. 2010, pp. 201–212.

[10] Ran Duan and Le Zhang. "Faster Randomized Worst-Case Update Time for Dynamic Subgraph Connectivity". In: WADS '17. 2017, pp. 337–348.

[11] Daniel D. Sleator and Robert Endre Tarjan. "A Data Structure for Dynamic Trees". In: STOC '81. 1981, pp. 114–122.

谢谢大家