Several Techniques for Maintaining Fully Dynamic Graphs

徐寅展 <xuyinzhan@gmail.com>

杭州学军中学

February 11, 2015

Dynamic Graphs

Fully Dynamic Graphs: 在线，有删边和加边操作。
Techniques:

- Sampling
- Clustering and Topology Tree
- Sparsification
- Offline Techniques

Dynamic Graph Connectivity

删边加边，询问两个点连通性。
Partially dynamic（只支持加边）：维护一个并查集，询问的时候检验两点是否在同一个集合里即可。

Level

给每条边分配一个0到$\log n$之间的等级。某条边的等级只会随着时间增加而减少。
令G_i是由等级小于等于i的边构成的子图，F_i为G_i的一个极大生成森林。

Invariant

- G_i的任意联通块最多有2^i个节点。
- $F_0 \subseteq F_1 \subseteq \ldots \subseteq F_{\log n}$，即$F_i = F_{\log n} \cap G_i$，且$F_{\log n}$是以等级为边权的$G_{\log n}$的最小生成森林。

同时，对每一个G_i维护一个邻接矩阵（对其中每个点建立一棵平衡树）。

Connected

询问u和v是否联通。
查询在$F_{\log n}$中u和v是否属于同一棵树。
$O(\log n)$.

Insert

插入一条边$e = (u, v)$。
先将e的等级设置为$\log n$，再插入到$G_{\log n}$中，并更新$F_{\log n}$和$G_{\log n}$的邻接矩阵。
$O(\log n)$.

Delete

将$e = (u, v)$从图中删去。
先将e从邻接矩阵中删去。
若e不在$F_{\log n}$中，则整张图的连通性不变，此次操作结束。否则：

Delete

我们想找到一条e的替换边。注意这条边的等级必然大于等于e。
在找替换边的同时还需要满足上面所述的两条性质。
依次从$level(e)$到$\log n$寻找这条替换边。

Delete

考虑在第i层寻找替换边。

1. 令T_u是包含u的树，T_v是包含v的树。不失一般性地，令$|T_v| \leq |T_u|$.
2. 由于本来T_u和T_v是在同一个联通块，因此$|T_u| + |T_v| \leq 2^i$，所以$|T_v| \leq 2^{i-1}$。而把T_v加到第$i-1$个等级是没有联通块与它相连的，所以可以把T_v移动到第i个等级。
3. 对于任意在第i层的边(x, y)，且x在T_v中：
   - 若y在T_u，则把(x, y)加入到$F_i, F_{i+1}, \ldots, F_{\log n}$中，并结束这个操作。
   - 否则将(x, y)的等级置为$i-1$.

Details

用大多数动态树数据结构都能完成对F的操作。
每一条边最多会往下掉$\log n$层，每次操作复杂度为$\log n$，所以均摊复杂度为$O(\log^2 n)$

Clustering and Topology Tree

以动态最小生成树为例。
加边、删边、修改边权，维护最小生成树

Transformation

经过转化，使每个点的度数小于等于3
将一个的度数为n的点v，拆成$v_0, v_1, \ldots, v_{n-1}$。在$v_i$和$v_{i+1}$之间连边权$-\infty$的边。若原图中有一条边$(u, v)$，则将对应的$u_x, v_y$之间连上原边权的边。
并可以由添加一个新点的方式，将原图转化为联通图。
新图的点数和边数都为$O(m)$，且两图的最小生成树结构是相同的。

Simplification

删边：将一条边边权改为$+\infty$
加边(u,v,x)：u和v拆成的点表里都新加一个点，这两个之间连上$+\infty$的边。这时不会影响MST的形态。最后再把边权改为x。之后只考虑修改边权的操作。

Algorithm 1

- Preprocessing time $O(m)$
- Update time $O(m^{\frac{2}{3}})$
- Space requirement $O(m)$

Topological Partitions

在最小生成树上删除一个边集E'，使得剩下的每个联通块点数都在z与$3z-2$之间。z的大小将会在之后分析。
每一个联通块叫做vertex 块，联通块的集合叫做topological partition of order z。

Topological Partitions

选最小生成树上的一个叶子作为根，进行dfs。
如果一棵子树的大小大于等于z，就把这棵子树作为一个vertex块；否则继续递归。
最后会求出很多在z与$3z-2$之间的集合，以及一个点数可能小于z的集合（根所在的集合）。如果小于z，那么就把这个集合与其相邻的任一集合合并。
这个构造是$O(m)$的。

Topological Partitions

Lemma

由上述过程形成的联通块大小在z到$3z-2$之间。

Proof.

生成树中每个点的度数最多为3，而根的度数为1，因此每个点最多只有两个孩子（二叉树）。
除了最后那个特殊的联通块，别的联通块都最多由一个点和这个点两个孩子的联通块构成，最大为$1+2(z-1)=2z-1$
最后形成的联通块最大为$(2z-1)+(z-1)=3z-2$ □

Maintain the Key Information

初始时找到两两块 V_i, V_j之间的最小边E_{ij}，存下来。
块个数：$O(\frac{m}{z})$
时间复杂度：$O(m + (\frac{m}{z})^2)$

Update Operations

1. 增加非树边权
2. 减小树上边权
3. 减小非树边权
4. 增加树上边权

Operation 1

增加非树边权。
对MST不会有任何影响。

Operation 2

减小树上边权。
不会对MST的形态造成影响，只会修改一条MST的边权，容易维护。

Operation 3

减小一条非树边的权值，可能会把一条树边替换掉。

1. 找到这条树边
2. 替换掉

Find the Edge

路径上最大边
LCT?
暂时先用这个方法，后面会讲一种方便的做法。

Switch the Edge

假设用(u, v)替换(x, y)。
如果x, y不在同一个块，那么块的形态是不会改变的，这时的时间复杂度为$O(z)$。
麻烦的情况是x, y在同一个块。

Split a Cluster

如果x, y在同一个块 V_i，就要把(x, y)删掉，并把V_i分裂成两个块V_{i1}, V_{i2}
分裂时需要维护V_{i1}和V_{i2}和其他所有块之间的最小边。
初始化：$O(\frac{m}{z})$
枚举所有一端在V中的边，更新：$O(z)$。（每个点度数最多为3）。

Merge Two Clusters

最后会把u, v所在的块合并。
这里只要添加上两块之间的边即可。

Consider Sizes of Clusters

当分裂一个块时，两个新的小块大小会小于z。这时要把小的块与相邻的块合并。合并后如果过大，再用dfs的方法分裂成两个块。
Special Case：当分裂后，某个块可能没有相邻的联通块。这时，只要等待把u, v连上即可，再去合并。

Operation 4

增加一条树边的权值，可能会把这个树边删掉，加入另一个非树边。

1. 找到这条非树边
2. 替换掉（与Operation 3大同小异）

Find the Edge

删除树边(x, y)会把所有块分成两个集合。
枚举所有i, j，使得V_i, V_j在分别在不同集合里。找到最小的一条E_{ij}。
若这条E_{ij}小于(x, y)的边权，那么就用这条边替换(x, y)
时间复杂度：$O(z + (\frac{m}{z})^2)$

Complexity

取$z = O(m^{\frac{2}{3}})$，则可以达到每次询问时间复杂度$O(m^{\frac{2}{3}})$，预处理时间复杂度$O(m)$。
空间复杂度为$O((\frac{m}{z})^2 + m) = O(m)$
每当边数增加$O(m^{\frac{2}{3}})$时（加边操作虽然不会影响MST，但实际上还是使边数增加了），都重新选择z并构造整个结构。

Algorithm 2

- Preprocessing time $O(m)$
- Update time $O(\sqrt{m \log m})$
- Space requirement $O(m)$

External Degree

一个块的度数定义为有且只有一端在这个块内的边数。

Simply-Connected Topological Partition

一个simply-connected topological partition与topological partition类似，都由$O(\frac{m}{z})$个大小为$O(z)$的块形成。不同之处在于，一个simply-connected topological partition每个块的度数至多为3.

Simply-Connected Topological Partition

与上面topological partition的构造类似。
选最小生成树上的一个叶子作为根，进行dfs。
如果一棵子树的大小大于等于z，或者一棵子树的度数为3，就把这棵子树作为一个块；否则继续递归。
若最后根所在的块度数不为3且点数小于z，则要把这个块再与之前形成的一个块合并。
这个构造是$O(m)$的。

Simply-Connected Topological Partition

Lemma

由上述过程所形成的联通块，要么节点数在z到$3z-2$之间，要么节点数小于z且度数为3。

Simply-Connected Topological Partition

Proof.

如果一棵子树不独自形成块，说明它的度数小于3且节点数小于z。那么，这种子树对父亲度数的贡献最多为1，节点数贡献最多为$z-1$。
那么对于父亲所形成的块，度数最多为3，节点数最多为$2z-1$，符合条件。
对于最后根所形成的块，若度数最多为2，点数最多为$z-1$，那么合并之后的度数最多为3，节点数最多为$3z-2$。 □

Lemma

由上述过程构造的联通块个数为$O(\frac{m}{z})$。

Proof.
将构造之后的联通块缩为一个点，相邻的联通块之间连一条边。这样形成的图为最大度数为3的树。令度数为x的点有V_x个。这棵树中，度数小于3的点，都包含了原图中至少z个点，因此$V_1 + V_2 = O(\frac{m}{z})$。

$$V_1 + 2V_2 + 3V_3 = 2(m-1)$$
$$V_1 + V_2 + V_3 = m$$

可以得到$V_1 = V_3 + 2$，所以$V_3 < V_1 = O(\frac{m}{z})$，因此$(V_1 + V_2) + V_3 = O(\frac{m}{z}) + O(\frac{m}{z}) = O(\frac{m}{z})$.

$\square$

Add an Edge

加边就可以直接暴力重构被连起来两个块。
如果所有点的度数都为3，那么所有点都单独变为一个小块即可。
否则找到一个度数小于3的点作为根*dfs*构造块。

Delete an Edge

删除一条边(x, y)，可能会使x和y所在的块不合法：度数小于3且大小小于z。
遇到这种情况，只需将它不断与相邻块合并，直到不再处于度数小于3且大小小于z的状态。
若合并之后的块大于$3z-2$，则需要使用simply-connected topological partition的分割方法分离。

Topology Trees

Topology trees是一种对simply-connected topological partition在$z = 2$时的递归结构，即不停地进行simply-connected topological partition，最终得到一个树形结构。
每次进行划分后，在相邻的块之间连边，形成新的树，不断进行划分。

Multi-Level Topological Partition

1. 对于每一个i，第i层的块是原来点集的一个划分。
2. 第0层的块是单个节点。
3. 对于一个在大于0层的块，它是以下两者之一：
    - 由在第$i-1$层的2或3或4个块连接而成，且这个新形成的块度数不超过3。
    - 一个在$i-1$层的块，且它的度数为3。

Topology Tree

一棵树T的topology tree:

1. 内部节点最多只有4个孩子，且所有叶子都在相同深度。
2. 一个在第i层的节点对应在multi-level topological partition中第i层的块。
3. 一个在第$i > 0$层的节点的孩子对应于形成这个块的那些块。

Depth

Lemma

如果n是一棵树T的节点数，那么T对应的*topology tree*的深度为$O(\log n)$。

Depth

Lemma

如果n是一棵树T的节点数，那么T对应的*topology tree*的深度为$O(\log n)$。

Proof.

如果在第i层的点数为n_i。令度数为j的点有V_j个。
$V_1 = V_3 + 2$说明度数为3的节点最多为$\frac{1}{2}n_i$，则n_{i+1}最多为$n_i - \frac{1}{2}(\frac{1}{2}n_i) = \frac{3}{4}n_i$。 $\square$

Generation Time

Lemma

一个*topology tree*的构造复杂度是$O(n)$的。

Proof.

$$O\left(\sum_{i=0}^{\infty} n(\frac{3}{4})^i\right) = O(4n) = O(n)$$

□

Critical Operations

重要的操作是在最小生成树中删除一条边，以及增加一条边以连接两棵生成树。
这两个操作使得我们需要支持合并和分裂topology tree.

Difficulty

修改形态的难点在于topology tree有度数限制。合并两棵树可能会导致块的度数大于3，分裂一棵树可能会导致节点数过少。

Merge

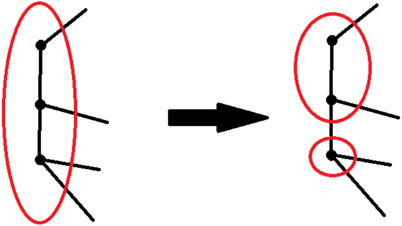
加入一条边之后，会使得某些块的度数从3变为4。令level最小的这样的块为W。

Lemma

W必然能分成两个度数合法的块W'和W''。

Deal with Degree

Proof.
分情况讨论。比如：



Deal with Degree

像这样把W变W', W''之后，将它们的父亲都设置为原W的父亲。这时候，W的父亲可能需要分裂。
分裂只需构造一个W父亲的topological partition，这个可以用上面提到过的算法。
由于之后W的父亲又会变为多个节点，因此会在topology tree中从下往上依次改变。
更新过的块度数不会再有问题。这时再找下一个这样的W进行同样的操作。
每一次改变是$O(1)$的，而层数又是$O(\log n)$ 的，因此处理度数的时间复杂度为$O(\log n)$.

Merge

将高度较低的那棵topology tree的根插入到另一棵的对应高度。
再进行以上分裂操作。
总时间复杂度：$O(\log n)$.

Split

删除一条边，会使包含这条边的块都分裂，这些分裂的块会形成topology tree上的一条路径。
这个分裂操作可以在$O(\log n)$时间完成。

Deal with Degree

分裂之后，节点度数只减不增。因此，非法的情况只能是某个由一个节点组成的块，度数从3变为了2，这样的块必然都在一条到根的路径上。
令level最小的这样的块为W。令W'为跟W在同一层，且与W的lca最低的点，必然存在这样一个W'与W之间有边相连。将W与W'合并，并调整新形成的块，依次往上更新。
更新过的块度数不会再有问题。这时再找下一个这样的W进行同样的操作。
因此时间复杂度为$O(\log m)$。

Procedure

先对于原图的最小生成树求出simply-connected topological partition，再建出这个新图的topology tree。

Heaps

对第一次划分后的每一个块，维护一个topology tree。
块A所对应的topology tree的每一个节点x，代表的是A到x的所有叶子的最小值。
那样Algorithm 1中的$(\frac{m}{z})^2$就被消去了，但更新块的复杂度变为了$O(\frac{m}{z} \cdot \log m)$.
因此选$z = \sqrt{m \log m}$，单次操作的复杂度为$\sqrt{m \log m}$.

Algorithm 3

- Preprocessing time $O(m)$
- Update time $O(\sqrt{m})$
- Space requirement $O(m)$

The 2-Dimensional Topology Tree

令V_α和V_β为在topology tree中同一层的两个节点，那么在2-dimensional topology tree中就有一个标记为$V_\alpha \times V_\beta$的节点，表示有一端在V_α另一端在V_β的那些非树边中边权最小的边。
若一个节点为$V_\alpha \times V_\alpha$且V_α在topology tree中有孩子$V_{\alpha_1}, V_{\alpha_2}, \ldots, V_{\alpha_r}$，那么$V_\alpha \times V_\alpha$有孩子$\{V_{\alpha_i} \times V_{\alpha_j} | 1 \le i \le j \le r\}$.
类似地，$V_\alpha \times V_\beta$有孩子$\{V_{\alpha_i} \times V_{\beta_j} | 1 \le i \le r_\alpha, 1 \le j \le r_\beta\}$.
2-dimensional topology tree的叶子代表的是E_{ij}（上文提到过的i与j两块之间的最小边）。

Maintenance

修改topology tree会相应地影响2-dimensional topology tree。
topology tree上的变化都是从根到某一个节点V_α进行。
在2-dimensional topology tree中，修改的是形如$V_r \times V_\beta$的节点，其中$V_r = V_\alpha$或V_r为V_α的祖先。那么一共修改的点数为level大于等于V_α的点数，而topology tree中的总点数为$O(\frac{m}{z})$.
又如果修改2-dimensional topology tree的某个节点，那么它的父亲也一定会被修改，因此维护2-dimensional topology tree是$O(\frac{m}{z})$的。

Find the Replacement Edge

所有这些数据结构都是为了解决一个最棘手的操作：删除最小生成树上的一条边，并找到它的替换边。
删边之后，topology tree会变为T_α和T_β分别储存着点集V_α和V_β。
不失一般性的，令T_α的深度小于等于T_β，那么只要在2-dimensional topology tree中询问所有形如$V_\alpha \times V_r$的节点即可。
时间复杂度为$O(\frac{m}{z})$

The End

令$z = \sqrt{m}$，则可以得到一个每次更新复杂度为$O(\sqrt{m})$的做法。

Open Question

上文中提到的simply-connected topological partition与普通分块的区别在于每个块度数最多为3。这个性质使它有什么特别用处呢？

Sparsification

Sparsification通过将原图转化为稀疏图，从而使原来的每次更新操作复杂度从$f(n, m)$变为$f(n, O(n))$.
在这里提到三种Sparsification的方法。

1. Basic Sparsification
2. Stable Sparsification
3. Asymmetric Sparsification

Certificate

对于一个图的性质P和一张图G，G的一个certificate是一张图G'，使得G有性质P当且仅当G'有性质P.
G'不需要是G的子图。

Strong Certificate

对于一个图的性质P和一张图G，G的一个strong certificate是一个与G有相同点集的图G'，使得对于任意H，$G \cup H$有性质P当且仅当$G' \cup H$有性质P.
G和H有相同的点集，和不相交的边集。
G'不需要是G的子图。

Property

Lemma

在性质P下，若G'是G的strong certificate，G''是G'的strong certificate，那么G''是G的strong certificate.

Property

Lemma

在性质P下，若G'是G的*strong certificate*，H'是H的*strong certificate*，那么$G' \cup H'$是$G \cup H$的*strong certificate.*

Sparse Certificates

如果有一个常数c，使得对于任意一个n个顶点的图G，都能找到一个点数不超过cn的strong certificate，那么认为这个性质P是sparse certificates 的。

Sparsification Tree

每次将原图的点集尽量平均地分成两部分，并递归划分。这样会形成一个完全二叉树的结构，离根距离为i的节点数有$\frac{n}{2^i}$个。

Sparsification Tree

用类似一个2-dimension topology tree的结构，对于任意两个上述点集划分中同一层的点V_α和V_β，我们在边集划分树中在这一层创造一个点$E_{\alpha\beta}$，它包含了所有一端在V_α和一端在V_β的边。$E_{\alpha\beta}$的父亲为$E_{\gamma\delta}$，其中γ和δ分别是α和β在边集划分树中的父亲。

每个节点$E_{\alpha\beta}$有0或3（$\alpha = \beta$）或4个孩子。

Sparsification Tree

那些之间没有边的点集之间的E是不会被创造的。因此删边的时候可能会把某些E删除，加边的时候可能会增加某些E。
又有每一条边对应于$O(\log n)$个节点，所以这个结构的空间复杂度为$O(m \log n)$。

Lemma

在第i层（从0开始）的$E_{\alpha\beta}$所形成的导出子图的点数最多为$\frac{n}{2^{i-1}}$.

Proof.

V_α和V_β分别最多只有$\frac{n}{2^i}$个节点。 $\square$

Basic Sparsification

用来加速静态图上的算法。

Well Behaved Time Bound

对于一个时间上界$T(n)$，若存在常数$0 < c < 1$，使得$T(\frac{n}{2}) < cT(n)$，就称$T(n)$为well behaved。
任何多项式都是well behaved。
对数和其他增长一些缓慢的函数，都不是well behaved。

Basic Sparsification

先对原图建上述的边集的划分树，这棵树的根对应的就是原图。每次修改一条边，只会影响这棵树中$O(\log n)$ 个点。
对于每个点，求出这个边集的导出子图的sparse certificates。那么对于一个点，只需要把它的所有孩子的sparse certificates并起来再求一次sparse certificates。最后得到根的sparse certificates后，再求出答案。

Time Complexity

若求一张点数为n，边数为m的图的sparse certificates复杂度为$f(n,m)$，且f是well behaved。
求第i层的sparse certificates时，点数为$\frac{n}{2^{i-1}}$，边数为$O(\frac{n}{2^i})$，因此总的时间复杂度为

$$O(\sum_{i=0}^{\infty} f(\frac{n}{2^{i-1}}, O(\frac{n}{2^i}))) = O(f(n, O(n)))$$

这样就能把原来有m的复杂度消除m了。

Sample

动态图连通性。
加边删边，询问两点是否在同一联通块中。
没有加边删边操作，每次更新只要dfs一遍。时间复杂度是$O(n+m)$的。

Sparse Certificates

这里的sparse certificate就是原图的一个极大生成森林，也就是原图中每一个联通块都用一棵生成树表示。
显然strong和sparse。
对原图进行边集的划分。

Operation

修改边集的时候，在边集划分树中从底向顶依次修改。每次将一个点的所有儿子的生成森林中的边都并起来，再在这个并起来的图中求生成森林。

在第i层，求出来的生成森林最大为$\frac{n}{2^{i-1}}$。因此单次修改时间复杂度为

$$O(\sum_{i=0}^{\infty} f(\frac{n}{2^{i-1}}, 4\frac{n}{2^{i-1}})) = O(n)$$

Stable Sparsification

用来加速已经有动态算法的动态图算法。

Further Refinement

令A是一个变量为图G，函数值为G的strong certificates的一个函数。
如果A满足对任意G和任意$e \in G$，$A(G)$与$A(G-e)$只有$O(1)$条边不同，称A为stable的。
例：$A(G) = G$
当然我们需要A也是sparse的。

Stable Sparsification

边集划分树中的每一层，都只有$O(1)$条边改变。
需要注意的是，如果这个$O(1) = s$，那么第i层s条边改变会引起第$i-1$层有s^2条边改变，累计下来会很大。
但是这些边中真正改变的（删除重复或无效的变化）只有$O(1)$，因此每一层都要删除一些没用的操作。

Time Complexity

接下去基本与basic sparsification类似，只是每次更新都是用动态的方法维护那些改变的边罢了。

Sample

Dynamic MSF
这边的A函数即把原图转变为原图的最小生成森林。
为了更好地说明这，下面定义一种唯一的最小生成森林：边权相等的时候按端点的字典序排序。

Property

Lemma

令T为G的一最小生成森林，$e \in T$。 那么要么$T-e$是G的最小生成森林，$T-e+f$为最小生成森林。

Lemma

如果T是G的MSF，T'是G'的MSF，那么$T' \cup T$的MSF是$G \cup G'$的MSF。

Maintain

修改一条边，从底到根依次更新。
求出每个边集集合的MSF，之后合并所有孩子的MSF再求一次。
时间复杂度为

$$O(\sum_{i=0}^{\infty}\sqrt{\frac{n}{2^{i-2}}}) = O(n^{\frac{1}{2}})$$

More Insertion

- 删除时暴力重建
- Asymmetric Sparsification

Different Sparsification Tree

将图分成$O(\frac{m}{n})$个子图，使得所有子图尽可能都正好有n条边，除了一个至多有n条边的小子图。
再将这些子图作为叶节点，构造平衡树，每个内部节点代表的是对应叶子的并集。这棵平衡树就是这里新定义的sparsification tree.

Modification

Insertion 直接在小子图上添加一条边，如果过大，就把它分裂成两个。

Deletion 删除一条边，之后在小子图上拿一条边，补到这个子图中。如果小子图为空，就把另一个子图作为小子图，把本来这个删掉。

在这边不考虑每个子图内部如何维护。单单看影响的子图个数，是$O(\log \frac{m}{n})$.

Pros and Cons

实现比类似2-dimension topology tree的结构更简单，常数更小，空间为$O(m)$.
Certificates不会随深度增加而减小规模，因此总的复杂度会多出一个$O(\log \frac{m}{n})$.

Outline

维护一个可插入的部分动态图结构。
如果只有加边，就只需维护这个部分动态图结构（往往复杂度很优秀），并记录下加了哪些边。
只有当删边的时候才更新sparsification tree，并把在列表中加的边清空。
使得插入边的效率非常高。

Help to Improve Space

用basic和stable sparsification时，只把节点大小分裂到$O(\frac{n^2}{m})$，之后用asymmetric sparsification分。

Property

Lemma

basic和stable sparsification在这个结构中的空间为$O(m)$.

Proof.

basic和stable sparsification的第i层节点个数为$O(4^i)$，第i层所存的certificates的总大小为$O(2^i \times n)$.
在节点大小为$\frac{n^2}{m}$时，$\frac{2^i \times n}{4^i} = \frac{n}{2^i} = \frac{n^2}{m}$.
而

$$\sum_{j=0}^{i}(2^j \times n) = O(2^i \times n) = O(m)$$

□

Property

Lemma

此结构中*asymmetric sparsification*的复杂度是$O(f(n))$的（$f(m)$是*sparsification*之前做法的复杂度。）

Property

Proof.

在大小为$O(\frac{n^2}{m})$开始，点数为$O(\frac{n^2}{m})$，边数为$O(m)$。
按照asymmetric sparsification的复杂度计算，为$f(\frac{n^2}{m})\log\frac{m^2}{n^2}$
但是别忘记f是well behaved，也就是$f(\frac{n}{2}) < cf(n)$，其中$0 < c < 1$.
也就有

$$O(f(\frac{n^2}{m})\log\frac{m^2}{n^2}) = O(f(n)c^{\log\frac{m}{n}}\log\frac{m}{n}) = O(f(n))$$

注意，因为若$m < n$，就不用进行这一系列优化了，所以在上面认为$m \geq n$. □

Details

由于空间复杂度降为了$O(m)$，因此预处理复杂度也降为了$O(m)$。
在加边删边的过程中，$O(\frac{n^2}{m})$的边界可能会变化。这只要在若干次操作之后重建整个数据结构即可。

Dynamic Connectivity Training

Problems

#	Name			
A	Connect and Disconnect	**connect.in / connect.out** 3 s, 256 MB		x6
B	GraphAero	**bridges.in / bridges.out** 2 s, 256 MB		x3
C	Bridges in a Tree	**bridges2.in / bridges2.out** 5 s, 256 MB		x4
D	Bridges: The Final Battle	**bridges3.in / bridges3.out** 2 s, 256 MB		
E	Disconnected Graph	**disconnected.in / disconnected.out** 3 s, 256 MB		x1

Complete problemset

Problem A

维护一张无向图的联通块个数，能支持加边和删边。
$1 \leq N, M \leq 300000.$

Solution

分治

Solution

维护按删除时间为权值的最大生成树。

Problem B

维护一张无向图中桥的个数，能支持加边。

$1 \leq N, M \leq 100000.$

Solution

并查集

Problem C

给定一棵树，每次处理一个这样的询问：
如果增加一组边$(1, p_1), (2, p_2), ..., (K, p_K)$，将会有多少桥。
$1 \leq N, \sum K \leq 100000.$

Solution

虚树。

Problem D

维护一张无向图的桥个数，能支持加边和删边。

$1 \leq N, M \leq 100000.$

Solution

与Problem A相差不多。

Problem E

给定一张图，每次问如果删掉c条边，这张图是否还会联通。
$1 \leq c \leq 4, 1 \leq n \leq 10000, 1 \leq m \leq 100000.$

Solution

一种做法是跟A相同的。

Solution

然后有一种随机化的做法，首先随便找出一棵生成树，然后非树边的权值随机，树边的权值为覆盖它的非树边权值的异或和。这样子生成的图与“每个点的所有邻边异或和为0”是等价的，可以用数学归纳法每次缩掉一个叶子。于是就有一个结果：对于任意生成树都满足上面这个性质。

Solution

考虑一个边集是否是极小的能把原图割开的边集。
若这个边集不能把原图割开，则删去这个边集后原图肯定有一棵生成树。对于这棵生成树的非树边也是随机的。那么从这些非树边中选出若干条边，异或和为0的概率为$\frac{1}{2^w}$.
如果能割开，则异或和必然为0.
一个大小为w的不能隔开的集合判断正确的概率为$\prod_{i=0}^{v-1}(1-\frac{1}{2^{w-i}})$.

THANK YOU!

References

[1] Feigenbaum J, Kannan S. Dynamic graph algorithms[J]. 2000.

[2] Frederickson G N. Data structures for on-line updating of minimum spanning trees, with applications[J]. SIAM Journal on Computing, 1985, 14(4): 781-798.

[3] Eppstein D, Galil Z, Italiano G F, et al. Sparsification—a technique for speeding up dynamic graph algorithms[J]. Journal of the ACM (JACM), 1997, 44(5): 669-696.