Euler-Tour-Tree的应用
Dynamic Connectivity

胥拿云 龙思杉 冼臧越洋 张哲恺

April 30, 2016

Contents

1 功能介绍

在完全动态图问题中, 我们考虑一个图$G(V,E), |V| = N$。 这个图会被加边删边操作更新。我们假设从一个边集为空的图开始，有对当前图的更新以及查询操作。
对于一个完全动态联通问题而言，每次询问的是所给的两个点在当前图中是否联通。更新和查询都是在线的。

于是我们维护这个图的生成森林（对每个联通块维护一棵生成树），如果可以在每次更新的时候用$O(t(N)logN)$的时间维护生成森林，利用接下来介绍的算法，就可以在$O(logN/logt(N))$的时间内回答联通性询问。而$Euler\,tour\,trees$是适合进行这些操作的数据结构，无论是在两棵树间连边，从树上删边，还是检查两个点是否在同一棵树中，它都可以在$O(logN)$时间内完成。

我们用$treap$来维护平衡。假设图中每条边出现两次，对图取一欧拉路径，记下所遇见的点的序列，存入$treap$.如果对树进行$link$或cut操作，对原路径进行最多2次分离和2次连接操作后就可形成新的欧拉路径。而分离和连接操作的时间复杂度为$O(logN)$。

2 原理说明

2.1 简要介绍

对于一个无向图，动态维护任意两点间连通性，一个非常直观的想法是维护这个图的生成树森林。那么询问两点是否连通，就是查询这两个点是否在同一棵生成树中。加边操作就变得十分显然：如果这条边的两端点已经连通，那么只要把这条边加进原图中，生成树森林没有改变。如果这条边两端点并不连通，那么将这两棵生成树合并。

然而删边操作就比较复杂，一种简单的情况是待删边的两端点在不同的生成树中，那么只要在原图中将这条边删去。如果待删边两端点在同一棵生成树中，那么考虑删除这条边之后，原图中不在生成树上的边（即非树边）有可能替代这条边连接两棵树，而使得连通性没有发生改变。我们需要寻找是否存在这样的替代边，自然的想法是检测所有的非树边，然而这样的耗费太大了，所以我们使用一种分层的思想，保证了寻找替代边操作的复杂度。实现方式较为复杂，将在下文算法轮廓中说明。

2.2 算法轮廓

我们的算法将正在维护的图中边的集合分为$O(logN)$层(level)：第0层，第1层，第2层，…，第$logN$层。每一条边都属于且仅属于某一层。 设E_i表示第i层中，边的集合。那么对于每一层，我们维护一个生成树林F_i，F_i是第i以及更高层的一个生成树林: $\bigcup_{j\geq i} E_i$。且$F_i \subseteq Fi-1(i>0)$。因此F_0就是整个图的生成树林。 所以可以说，F_i就是F_0中第i层边的集合。同样可以看出，我们的生成树林中第i层的边将第$(i+1)$或以上层的边形成的连通分量连接起来。 这样一来，可以看出，如果删除了某一条第i层的树上的边，则其对应的交换边所在的层数一定不会高于i。

我们的动态算法，即维护这样一个图G的生成树林F。F中的边将被称为树上边（tree-edges）。同样，如上文所述，将每一条边e与某一层$l(e) \leq L = log_2 N$关联，

即e属于第$l(e)$层。且对于每一层i，如上文定义$Fi : F = F0 \supseteq F1 \supseteq F2 \supseteq ... \supseteq F_L$。并需要保证下列性质成立：

a)对于每一层而言，F_i是该层以及该层以上所有边的一棵尽量大的生成树，即如果存在边(u, v)是非树边，那么u, v一定在$F_{l(u,v)}$中被连接。

b)F_i中最大的连通分量的节点个数，不得超过$\frac{n}{2^i}$，即最大的非空层为$logN$。 在算法的初始时刻，所有的边都位于第0层，这样满足上述的所有条件。那么我们可以较为细致的描述算法如何实现各种操作：

$Insert(e)$：将新加入的边放入第0层：E_0，如果这条边连接了F_0的两个不同的连通分量，则将这条边加入F_0。

$Delete(e)$：如果e是一条非树边，则仅仅在$E_{l(e)}$中将其删去即可。否则若e是一条树上边，那么将其删去，并寻找一条交换边来重新连接被分开的两部分。由于交换边所在的层数一定不大于$l(e)$，因此我们可以调用$Replace(e, l(e))$来寻找交换边。

$Replace(e(v, w), i)$：假设在第i层以上都不存在e的可交换边，那么我们这个过程试图在$level \leq i$的集合中，寻找一条层数尽可能高的交换边。 假设T_v和T_w是在F_i中删除e以后包含点u和v的两个连通分量。不是一般性的假设$|T_v| \leq |T_w|$，则在删除之前，$T = T_v \cup e \cup T_w$是第i层的一棵树，它至少含有$|T_v|$两倍多的节点。由b)性质，T最多有$\frac{n}{2^i}$个节点，因此，T_v最多有$\frac{n}{2^{i+1}}$个节点。所以我们可以将T_v中所有第i层的边移入第$(i+1)$层，而不触犯任何性质：即让T_v成为第$(i+1)$层的一棵树。 这样，我们枚举第i层的所有一端端点在T_v的非树边f，依次判断他们是否可以成为一条交换边。 如果f不能连接T_v和T_w，则根据性质a)，f的两个端点一定都属于T_v，则将其改为第$(i+1)$层的边而不会影响每一条性质的成立。而这次增加，则可以作为检测f需要的时间费用。 如果f连接T_v和T_w，则f就是一条交换边，将其插入树中，我们的检测过程结束。 如果将所有第i层的与T_v连接的非树边检测完毕而仍为找到交换边，则继续调用$Replace(e(v, w), i-1)$，若$i = 0$，则过程结束，我们可以断定整个图中没有交换边。

3 正确性分析

使用上述方法，可以在动态删边加边中维护出当前图的生成树森林，因此查询正确性显然。

删除和插入过程中生成树的正确性，在原理描述中已有涉及。

4 时空复杂度证明

4.1 时间复杂度

4.1.1 insert操作

当一条边插入 F_0时，使用$Euler\,Tour\,Tree$，合并两棵生成树花费是$O(logN)$。但是它的$level$可能会被提升$O(logN)$次，每次提升的花费是$O(logN)$，所以均摊复杂度为$O(log^2N)$.

4.1.2 delete操作

删除一条非树边时，涉及原图的修改$O(1)$，以及$O(logN)$的修改平衡树标记(用于$replace$)。

删除一条树边时，我们需要将F_j（$j \leq l(e)$）中的这条边都删除，需要$O(log^2N)$的时间.

接着调用$Replace$过程：最多调用$O(logN)$次，每次都需要$O(logN)$的时间（除去被平摊为插入的时间代价），这一部分的时间复杂度也是$O(log^2N)$。

4.1.3 query操作

查询两个点是否在一棵生成树中，利用$Euler\,Tour\,Tree$花费$O(logN)$。

4.2 空间复杂度

每个生成树森林需要一棵$Euler\,Tour\,Tree$， $O(N)$空间。这部分$O(NlogN)$空间。

对于原图需要维护一个边表，$O(M)$空间。 因此空间复杂度$O(NlogN+M)$.

5 具体实现方式及细节

- $Insert(e=(v,w))$
  设定e的初始$level$为0， 并更新v,w的邻边列表。
  如果v,w在F_0中不同的联通块，我们把e加入F_0.

- $Delete(e=(v,w))$
  首先把e从v,w的邻边列表中删除。
  - 如果e在F_0中
    - 把e从F_i中删除($i \leq level(e)$)
    - 找到重新连接v和w的替代边
      - 替代边的$level$不可能大于$level(e)$
      - 所以我们从$level(e)$往下找替代边
      - 遍历所有非树边需要时间过多，随机取边检查提高效率
      - 只检查那些连接了由于之前删掉树边后不连接的树的边。
    - 对于$i=level(e),...,0$:
      - 令T_v为包含v的树， T_w为包含w的树
      - 给v,w重新标号使得$|T_v| \leq |T_w|$.
      - $|T_v|+|T_w| \leq 2^i \Rightarrow |T_v| \leq 2^{i-1}$(这保证了我们可以把$T_v$上所有的边移到$level\ i+1$)
      - 对于每条边$e'=(x,y)$满足x在T_v并且$level(e')=i$
        - 如果y在T_w中:对$F_i,F_{i-1},...,F_0$加入(x,y)
        - 否则令$level(e')=i-1$

- $Connected(v, w)$
  用二叉平衡树保存F_0中的节点。对v, w调用$Findroot$函数检查它们是否在同一联通块中。

实现细节:

本算法实现的最大难度在于寻找替代边部分，如何以最高的效率遍历所有一端在T_v 中，且$level$为$l(e)$的边。

1.$mark$

参照论文上的描述，我们对每个$level$的生成树森林，都维护每个点和每条边的状态$markV$和$markE$，状态有on和off。如果一条边状态为on，则表明这条边的$level \geq l(e)$，且存在于原图中，若状态为off则表明这条边不存在或$level < l(e)$。如果一个点的状态为on，则这个点在这个生成树森林中，有$level$为$l(e)$的状态为on的非树边与之相连。

我们对$Euler\,Tour\,Tree$的每个节点维护这两个信息，并且，维护两个信息$markVUnion$和$markEUnion$，表明它的子树中是否存在$markV$为on的点，和子树中是否存在$markE$为on的边。那么我们在找寻目标替代边的时候，就可以通过$mark$和$markUnion$来决定我们要遍历树的哪些位置。记录了$mark$之后，我们也仅需要一个边表就可以维护出每个生成树森林中的树边和非树边。每次加边删边的时候，维护$mark$。

2.$Tree$

定义一个$tree$的类型，区分普通节点和一棵被$split$之后的生成树，方便实现比较两边子树大小和寻找替代边的操作。