PQ tree

许臻佳、曾凡高、杨卓林

June 1, 2016

Abstract

*PQ tree*是可以计算出一类排列，使得给定子集出现在排列里的位置是连续的数据结构。它巧妙的将有序树和无序树结合在一起，通过适当的分情况讨论，使原问题可以在$O(nm)$的复杂度内解决。（其中n为集合大小，m为限制子集的数量）。这一问题的解决在图论，生物医学等诸多方面有诸多应用。

Contents

1 原模型

设E为n个元素的集合。给定E的m个子集，求E中元素的所有可能排列，满足对于任意一个子集，子集中的元素在排列中成为连续的一段。 [1]

2 算法梗概

我们用一个树形结构来维护这个集合的可能排列。我们将集合中的具体元素放在叶子节点。很明显，将树遍历一遍即可求出其一种可能排列。

为了方便之后的说明，定义无序树和有序树的概念。

- 无序树: 对于一颗树，它被定义为无序树，当且仅当它的儿子节点可以任意排列。任意排列后得到的树的遍历结果仍然是一组合法的排列。我们把无序树的根节点叫做P节点。
- 有序树: 和无序树相对，如果它的儿子节点已经被固定下来，只能从左到右(或者从右到左)排列，那么称为有序树。我们把有序树的根节点叫做Q节点。特别的，对于一个Q，可能是**可翻转**的(既可以从左到右遍历儿子，也可以从右到左遍历儿子)，也可能是**不可翻转**的(遍历方向唯一)，这使得PQ $tree$本质上除了叶子节点外有三种不同类型的节点。

很容易发现，对于一开始无任何限制子集的情况，PQ $tree$的形态就是一个P节点接着全部的叶子节点。而每加入一个限制，就需要对PQ $tree$进行修改。可以看出，PQ $tree$是一种优秀的允许在线的数据结构，而且具有**可持久化**的潜能。

关于修改过程，为了之后叙述的方便，我们定义全集，空集，子集修改的概念。

- 全集: 对于一颗树，其所管辖的节点全部在这次限制的子集之内。
- 空集: 对于一棵树，其所管辖的节点全部不在这次限制的子集之内。
- 子集: 对于一棵树，其所管辖而节点部分在这次限制的子集之内。

很明显，在某次的限制条件之下，如果一个子树是全集或者空集，那么这次限制不会对这颗子树的结构造成任何的影响。而如果是子集，就需要更进一步的讨论。

为了更好的说明之后的修改讨论，我们定义偏左修改，偏右修改，无限制修改。

- 偏左修改: 对于一颗子树，要求修改完之后，所涉及到的节点必须全部靠左放置。
- 偏右修改: 对于一颗子树，要求修改完之后，所涉及到的节点必须全部靠右放置。
- 无限制修改: 对于一颗子树，修改完之后，无放置要求(仅仅要求涉及到的节点连续)

很明显，对于一次限制，我们要做的也就是对于根节点进行无限制修改。

下面将会分别阐述对于一个P节点或者Q节点进行修改的方法。

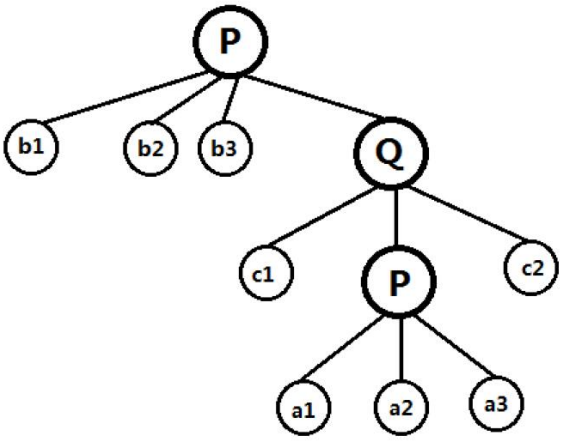
3 结构

每个节点记录：

- **int** id：0表示内部节点，$1 \sim n$表示叶子节点对应的编号
- **int** $size$：子树中叶子节点总个数
- **int** sum：子树中的叶子与选中子集的交集的大小
- **int** PQ_type：0表示P节点，1表示不可翻转的Q节点，2表示可以翻转的Q节点

- **bool** tag_rev：0表示不要翻转，1表示要翻转
- **vector**$\langle$**Node**$*\rangle$ $edge$：边表

整体的结构如下：



(a) 整体的结构

4 具体实现

对于每次修改，通过一次dfs，计算出每个点的$size$, sum。然后进行关于根的**无限制修改**的递归函数的调用。该函数根据所在节点是P节点还是Q节点进行分类修改讨论。

在进一步讨论前先将子节点进行分类：

- 集合a：全集（$sum > 0$ 且$sum = size$）
- 集合b：空集（$sum = 0$）
- 集合c：子集（$sum > 0$,且$sum < size$）

具体讨论如下: [2] [3]

4.1 P_operation

对于该节点的儿子，我们以全集，空集，子集的概念将它们分为三个集合a,b,c.

很容易发现，如果$|c| > 2$，显然不可能有解(必定有一块无法拼合)。而即便$|c| = 2$, 而此时有偏左限制或者偏右限制，那么也无解。

如果$|c| = 2$，那么必定一个偏左限制，一个偏右限制递归下去。否则按照所要求的限制递归下去。

值的注意的是，在无要求的情况下，如果$|a| = 0$，那么c集合应当无限制，否则应该默认有要求的情况下并在a的一侧。(可以默认偏右限制)

接下来我们建立x宏点，来连接所有的a集合的点。同样建立一个y宏点，来连接所有b集合的点。可以发现这两个点都是P类型的点。我们现在考虑把a集合和c集合并起来。这里分三种情况。

- c集合没有元素: 那么就直接返回a集合的宏点。
- c集合有一个元素。那么新的这个点一定是Q类型的点。如果有偏左或者偏右限制，则按照要求用一个Q点将x 宏点和c集合的元素连起来。这个Q点是**不可翻转的**。如果无限制，则用一个Q点将x与c集合的元素并起来，这个Q是**可翻转的**。
- c集合有两个元素。那么新的这个点也是Q类型的点。用这个Q点将x宏点和c集合的元素，按照$c \sim x \sim c$ 的样子并起来。很容易发现这个Q也是**可翻转的**。

最后考虑将上述过程所得的点与b集合的并起来。如果无限制那么直接并起来就好。否则用一个Q按照限制的要求将他们并起来，很明显这个Q点是**不可翻转的**

这样就完成了P点上满足限制的调整和维护。

4.2 Q_operation

若$|c|>2$，显然无解。
显然要求这些节点按照$b\sim bca\sim acb\sim b$排列，否则因为Q节点顺序不能改变导致无解。
如果当前节点是无限制修改：

- 如果$|c|=0$，不作处理。
- 如果$|a|=0$，$|c|=1$，对c节点进行无限制修改。
- 从左到右按顺序进行处理，c（如果有的话，偏右修改），a（如果有的话，无限制修改），c（如果有的话，偏左修改）。

若当前节点是有限制修改，以偏左修改为例：

- 如果$|c|>1$，无解。
- 若儿子节点是按照$a\sim acb\sim b$，对c节点进行偏左修改（如果有的话）。
- 否则判断当前节点是否可以翻转，若不可翻转，无解，若可以翻转，进行翻转，翻转后将该节点标记为不可翻转，转到上一步骤。

偏右修改与偏左类似。

5 复杂度分析

5.1 空间复杂度

首先，叶子节点只有n个；其次，在操作过程中保证每个点至少有两个儿子（将一根链进行进行合并）。
由每个节点出度≥ 2，那么有：

$$\sum_{v\in V} deg^{+}(v)=|E|=|V|-1=2(|V|-n)$$

所以

$$|V|\leq 2n-1$$

因此空间复杂度是$O(n)$的。

5.2 时间复杂度

在修改过程中每个点最多被经过一次，根据空间复杂度是$O(n)$，可知修改的复杂度是$O(n)$。
查询是将PQ $tree$进行一次dfs，依次记录经过的叶子节点，根据空间复杂度是$O(n)$，可知查询的复杂度是$O(n)$。
所以单次操作的时间复杂度都是$O(n)$，如果有m个子集的限制（或者说m个操作），那么总的时间复杂度是$O(nm)$。

6 应用

PQ $tree$看似解决的问题很单一，但其实它在很多地方都有应用 [4]。
在图论方面：弦图的判定、区间图的判定都可以通过转化借助PQ $tree$来进行。
PQ $tree$还可以用来判定平面图，同时PQ $tree$有一个升级版本是PC $tree$ [5]，可以更为简单地进行平面性判定。
除了计算机方向，PQ $tree$在生物医学中也有重要应用，比如在DNA的重组与排序中就要借助PQ $tree$，在这里就不详细介绍了。
总之，PQ $tree$作为一个性能优异的数据结构，有广泛的应用空间，而且还有更多的应用等待着我们去探索。

References

[1] Codeforces Round #150 (Div. 1)http://www.codeforces.com/problemset/problem/243/E

[2] 刘才良 **" 0083-Space Suckers 解题报告 "** *IOI2003*中国国家集训队难题讨论活动

[3] 王鉴浩，张恒捷**"PQ tree"**

[4] Wikipedia：PQ treehttps://en.wikipedia.org/wiki/PQ_tree

[5] 刘剑成，刘研绎，杨定澄**"Planarity Testing"**