AAA tree

许臻佳、杨卓林、曾凡高

April 29, 2016

Abstract

AAA tree [1] [2]是Link-Cut Tree的加强版，LCT可以在$O(n \log n)$的时间复杂度内处理链修改、链查询，而AAA tree是在此基础上将轻儿子通过平衡树进行维护，从而实现了子树修改、子树查询的功能。所以AAA tree是处理动态树问题的的一种高效、有力的工具。本文将详细介绍AAA tree的结构、操作、标记以及一些实现细节。

Contents

1 结构

在一般的*link-cut tree*的基础上，对于每个节点，将其所有虚儿子(在树上通过虚边相连的子节点)用一颗*splay*(伸展树)来单独维护。下传子树操作的标记时，通过这颗维护虚儿子的*splay*来下传给虚儿子所在的子树。

*splay*的介绍和实现不在本文范围之内，故不再做详细介绍。

2 基本操作

2.1 内层splay

内层*splay*需要做到因虚边实边变化导致*splay*中节点的增删，同时更新内层*splay*维护的一个*AAA*节点的所有虚儿子的*size*总信息和*data*总信息。主要操作有:

- add(x): 将一个新的虚儿子插入*splay*。
- del(x): 将一个虚儿子移出*splay*。
- change(x, delta, tag type): 对*splay*节点放置标记。
- pushdown(x): 标记的下传。
- update(x): 更新这个节点的信息。

  注意的是，这个节点的信息由于本质上是两个儿子以及自己的信息和。因此为了获得真实的信息，我们仍然需要到对应的*AAA*节点处询问来获得当前的信息。

2.2 外层AAA节点

- access(x): 将x与整个树的根节点的路径打通，并且断掉x与其在树上的实儿子之间的实边（如果x有实儿子的话）

  实现过程与一般的*link-cut tree*的*access*类似。但是在进行一般的操作之前，我们必须要将x与整个树的根节点之间路径上的标记全部下传。因为在一般的*access*操作中是先将实儿子断掉再与所在实链的父亲联通。

  将标记下传完毕之后，我们进行一般的*access*操作。在将原有实边断掉之后将该实儿子成为新的虚儿子，将其作为该节点所在实链的代表元插入父节点维护虚儿子的*splay*中；在将虚边变为实边时，原虚儿子变为实儿子，将其从父节点的*splay*中删去。

  需要注意的是我们将一条实链深度最小的节点，而不是在*link-cut tree*中维护该实链的*splay*的根节点，作为该实链的代表元插入在父节点的维护虚儿子的*splay*中，这样可以方便子树修改标记的下传。

- BeRoot(x): 将x节点变为整棵树的根节点。

  实现过程亦和*link-cut tree*类似。先执行$access(x)$，即把x到根节点的路径打通。很容易可以发现，换根可能发生儿子父亲关系的调换。又可以发现，只有x到根的这一条路径上的点的深度关系会发生反转。因此在打通的这条路径形成的*splay*的根上，打上$reverse$标记就可以了。

3 具体操作

3.1 link(x, y)

先将x变为根，再将x的父亲设为y，由于此时x到y的边本质上是虚边，因此x成为y的虚儿子，在y的内层*splay*中加入x所对应的信息。

3.2 cut(x)

先$access(x)$, 将x到根的路径打通。那么在这条路径所对应的$splay$中，x明显是深度最大的节点。因此将x节点$splay$ 到根，此时x没有右子树。于是简单的将其与左子树断开就可以了。

3.3 lca(x, y)

先$access(x)$, 此时x到根的路径全部变成实链。再$access(y)$, 容易发现在虚实交错的地方的节点就是它们的LCA，于是在$access$的过程中记录一下就可以了。

3.4 链操作

先将x暂时换为整棵树的根，再$access(y)$，将y到根的路径打通。显然此时x到y之间的路径恰好由一个$splay$维护起来了。那么在这颗$splay$上打标记或者询问信息就非常简单了。

值得注意的是，需要事先记录原树的根，事后要记得换回来。

3.5 子树操作

首先$access(x)$,这个时候发现，x的子树信息全部由x的虚儿子和它本身的这个点构成。于是在它的虚儿子$splay$上打上标记，再改改这个节点的值就可以了。询问子树的话，将这两部分的信息合并一下便是答案。

4 信息维护

4.1 功能

一下介绍的标记支持的功能是：链/子树改成x，链/子树加x，和容易想到是用两个tag分别维护$cover$、add，但仔细分析可以发现两个tag不可能同时出现，假如现在有$add\ tag$，如果进行$cover$操作，那之前的add操作都没有意义，因此直接将标记改为$cover\ tag$；假如现在有$cover\ tag$，如果进行add操作，那直接在$cover\ tag$基础上加x即可。因此真正实现中只要维护一个tag，同时有一个$tag\ type$来表示tag的类型，0表示没有tag，1表示有$cover\ tag$，2表示有$add\ tag$。

4.2 AAA Node中的信息

每个$AAA\ Node$中有两个$size$，分别是$path\ size$和$subtree\ size$，$path\ size$表示LCT的$splay$中子树的大小，$subtree\ size$表示LCT的$splay$中子树中的每棵子树的大小总和减去$path\ size$(LCT的$splay$中每个点其实代表了一颗子树）。

同理每个$AAA\ Node$还会有两个$data$，两个tag，分别表示$path$,$subtree$的信息。

显然把$path$和$subtree$的信息合并就是整个子树的信息。

4.3 splay Node中的信息

在轻儿子的$splay$中每个点其实代表着一颗子树，因此$size$表示子树中的子树总和(第一个子树表示$splay$子树，第二个子树表示每个点表示的子树)。同理$data$也是这个含义。

4.4 标记下传

对于$AAA\ Node$中$path\ tag$只要只要传给左右儿子即可，$subtree\ tag$除了要传给左右儿子，还要传给自己轻儿子的$splay$(也就是轻儿子$splay$的根)。

对于$splay\ Node$中的tag只要传给左右儿子，但要注意$splay\ Node$一旦被改，要在对应的$AAA\ Node$进行修改。

References

[1] 黄志翱. **" 浅谈动态树的相关问题及简单拓展 "** *2014*国家集训队论文

[2] Claris. **" Claris' Blog：BZOJ3153：Sone1 "** *http://www.cnblogs.com/clrs97/p/4403244.html*