Splay Tree

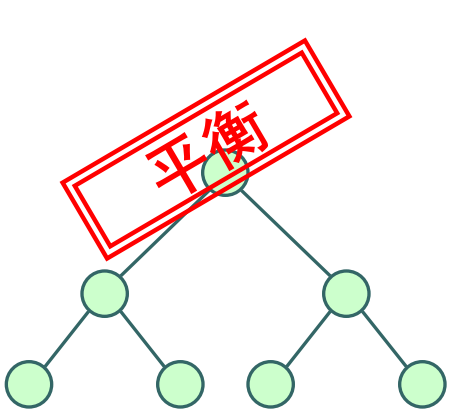
failedbamboo@gmail.com

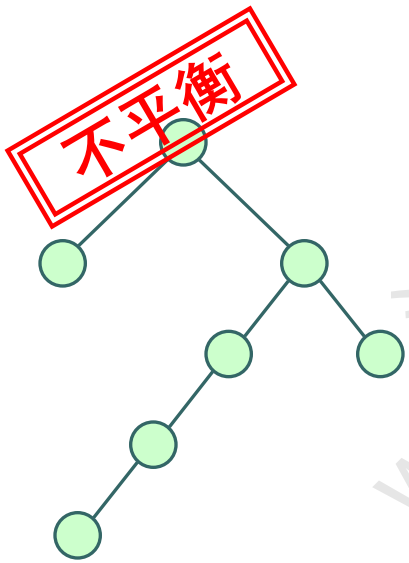
基本BST

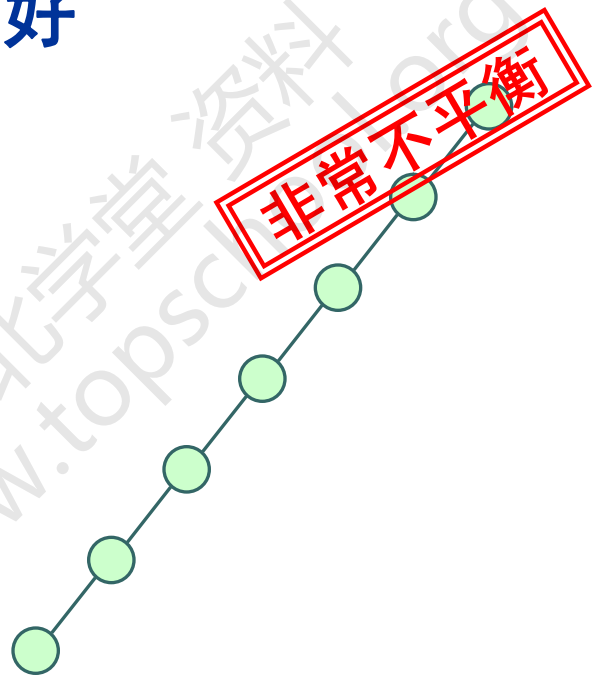
- **BST(Binary Search Tree)定义:**
  - 二叉树
  - 左子树 < 根 < 右子树
  - 左子树和右子树均为BST (递归定义)
- **基本操作实现:**
  - 查找: 从根往下走 O(h)
  - 插入: 查找失败后直接插入 O(h)
  - 删除: 查找后分情况讨论 O(h)

为何要平衡?

- 树越平衡,操作效率越高　操作复杂度O(h)
- 什么是平衡?
  - 固定节点数目,树的高度越小越好







平衡树的种类

- AVL Tree
  - 基本思想:
    - 对树的形状进行限制, 使得满足该限制的树一定是平衡的
  - 定义:
    - 每个结点的左右子树高度差不超过1(空树高度为-1)的排序二叉树称为AVL树
  - 操作:
    - 单旋,双旋

平衡树的种类

- **Splay Tree**
  - **基本思想:**
    - **不严格限制树的形状，而是假设访问有局部性，让数据在访问后不久再次访问时变快(将结点提到根)**
  - **操作:**
    - **伸展操作(Spaly) (左旋、右旋)**

平衡树的种类

- **Treap**
  - **Treap = Tree + Heap**
    - **每个点有两个键值(key , priority)**
    - **Treap关于key是一棵二叉排序树**
    - **Treap关于priority是一个堆(满足堆性质,但是不一定是完全二叉树)**
  - **插入时随机设置优先级，期望树高为O(logN)**

平衡树的种类

- 跳跃表(SkipList)
- 红黑树(RB Tree)

SplayTree 原理

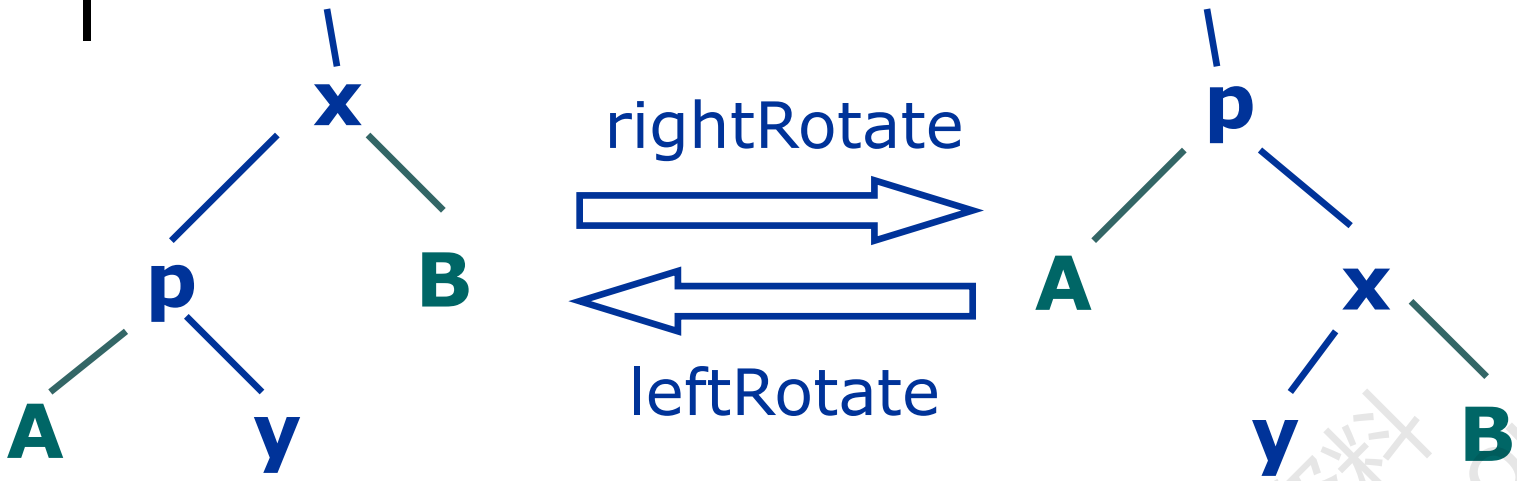
- 不严格限制树的形状，而是假设访问有局部性，让数据在访问后不久再次访问时变快(将节点提到根)
- 平摊操作复杂度O(logN)
- 提到根 → 伸展操作(Splay)

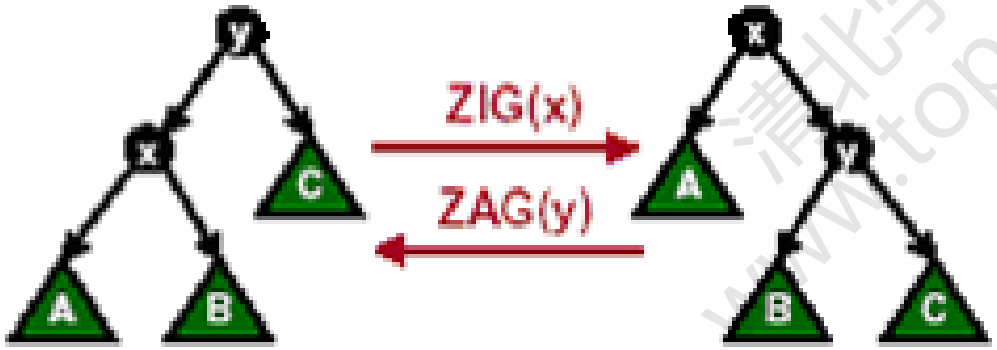
伸展操作(Splay)

- 伸展操作Splay(x,S)
  - 在保持伸展树有序性的前提下，通过一系列旋转操作将伸展树S中的元素x调整至树的根部

- 基本操作:
  - 左旋 (leftRotate / Zag)
  - 右旋 (rightRotate/ Zig)

左旋右旋(Zig / Zag)



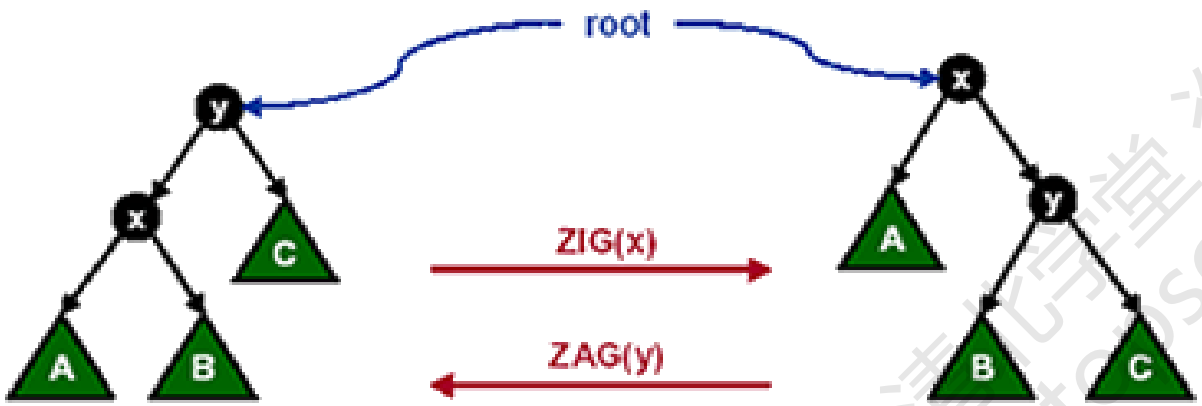


伸展操作实现

- 在伸展操作Splay(x,S)过程中:
  - 设 y = Father(x) , z = Father(y)
- 分情况讨论:
  - (A)　y是根结点: Zig 或 Zag
  - (B)　y不是根结点，且x和y同是各自父亲的左儿子(或者右儿子)，则Zig-Zig 或者 Zag-Zag
  - (C)　y不是根结点，且x与y中一个是左孩子一个是右孩子: Zig-Zag或Zag-Zig

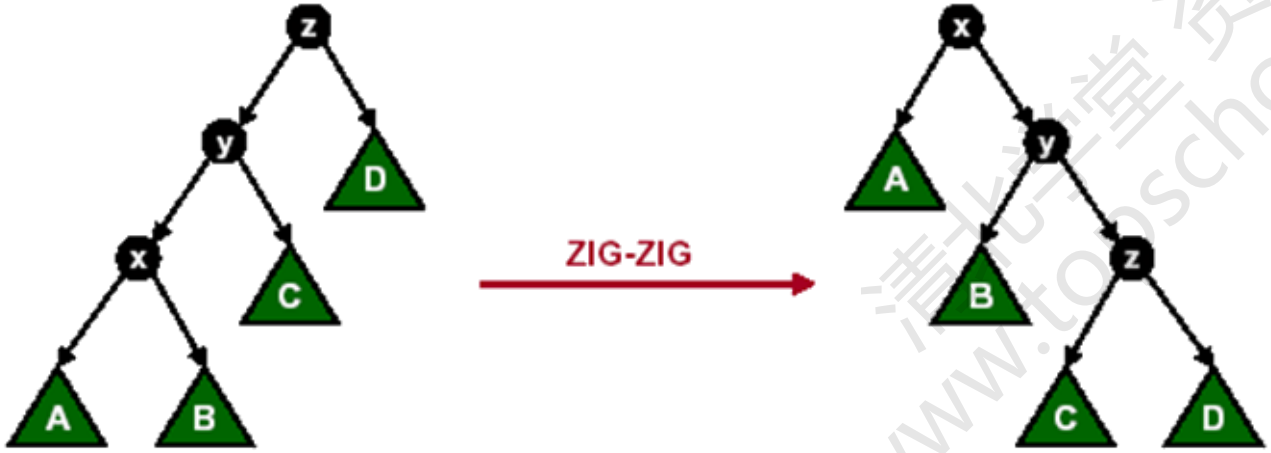
情况(A) Zig或Zag

- y是根节点
  - Zig(x) 或 Zag(x)



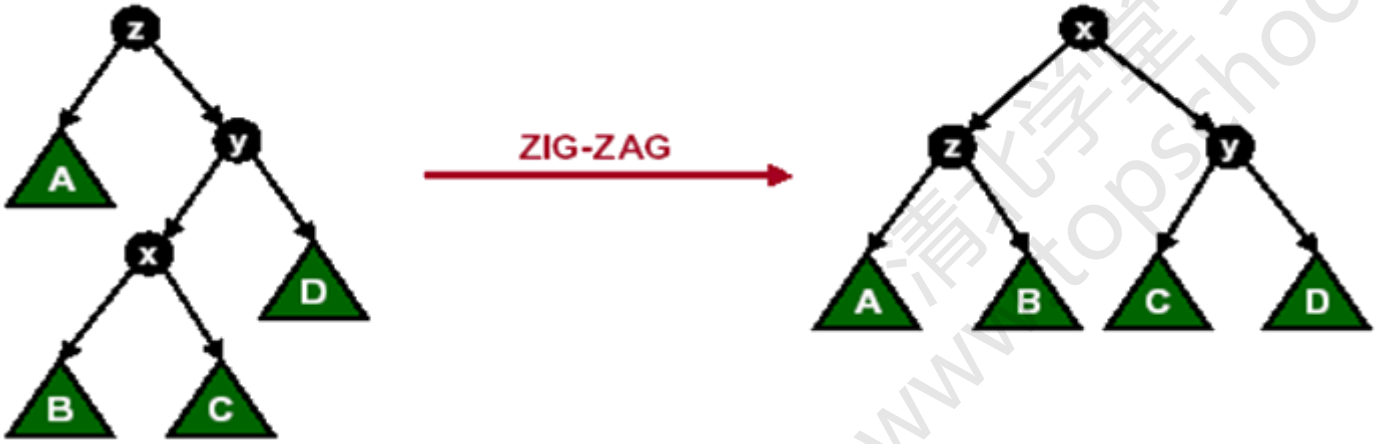
情况(B)　Zig-Zig或Zag-Zag

- y不是根结点，且x和y同是各自父亲的左儿子(或者右儿子)
  - Zig(y) + Zig(x) 或 Zag(y) + Zag(x)



情况(C) Zig-Zag或Zag-Zig

- y不是根结点，且x与y中一个是左孩子一个是右孩子
  - Zig(x) + Zag(x) 或 Zag(x) + Zig(x)



伸展操作举例

- 详见动画演示

SplayTree的一些基本操作

- 查找:
  - 从根节点往下走,完成后进行一次伸展操作
- 插入:
  - 查找该键值,失败后直接插入,并且对该结点进行一次伸展操作
- 删除:
  - 查找该节点,将其提到根(伸展),并合并左右子树

SplayTree的一些基本操作

- 寻找最小值:
  - 从根开始不断往左走,最后进行一次Splay操作
- 寻找最大值:
  - 从根开始不断往右走,最后进行一次Splay操作
- 求前驱:
  - 将当前结点提到根(Splay),前驱就是左子树的最大值
- 求后继
  - 将当前结点提到根(Splay),后继就是右子树的最小值

SplayTree的一些基本操作

- 合并两棵SplayTree A 和 B且满足A中任意一个数都小于B中任意一个数。
  - 若A为空,则返回B
  - 若A非空,将A中最大值提到根(Splay) ,显然这个根不会有右儿子。将B树作为其右子树

Splay操作是基础!

[例一]营业额统计

- 分析公司的营业情况是一项相当复杂的工作。经济管理学上定义了一种最小波动值来衡量营业情况：
  - 每天的最小波动值=
  - min { | 该天以前某一天的营业额-该天的营业额 | }
  - 第一天的最小波动值为第一天的营业额。
- 现在给出公司成立以来每天的营业额，编写一个程序计算公司成立以来每天的最小波动值的总和。
- 数据范围：天数$N \le 32767$,每天的营业额$a_i \le 10^9$。
- 最后结果$T \le 2^{31}$。

[例一]营业额统计

- 本题的关键是要每次读入一个数,并且在前面输入的数中找到一个与该数相差最小的一个。

- 每次顺序查找前面输入的数,找出最小差值
  - 时间复杂度$O(N^2)$
- 用线段树记录输入的数
  - 空间需求过大

[例一]营业额统计

- 每次读入一个数p，将p插入伸展树S，同时p也被调整到伸展树的根节点。
- 求出p点左子树中的最大值和右子树中的最小值，这两个数就分别是有序集中p的前趋和后继。
- 进而求得最小差值，加入最后结果T。
- 时间复杂度 O(NlogN)

[例二] GameZ游戏排名系统

- 排名系统通常要应付三种请求：
  - 上传一条新的得分记录
  - 查询某个玩家的当前排名
  - 返回某个区段内的排名记录。

- 当某个玩家上传自己最新的得分记录时，他原有的得分记录会被删除。为了减轻服务器负担，在返回某个区段内的排名记录时，最多返回10条记录。

[例二] GameZ游戏排名系统

- 输入第一行是一个整数n表示请求总数目。接下来n行每行包含了一个请求。请求的具体格式如下：
  - **+Name Score** 上传最新得分记录。Name表示玩家名字，由大写英文字母组成，不超过10个字符。Score为最多8位的正整数。
  - **?Name** 查询玩家排名。该玩家的得分记录必定已经在前面上传。
  - **?Index** 返回自第Index名开始的最多10名玩家名字。Index必定合法，即不小于1，也不大于当前有记录的玩家总数。

[例二] GameZ游戏排名系统

- 使用Splay
- 所需要的操作:
  - 查找、插入、删除、求后继
- 每个操作平摊复杂度: O(logN)

部分代码实现

```
TSplayNode = record
        left , right , father     : Integer   ;
        key                       : KeyType ;
End;

Tree :
     Array[0..LimitSize] of TSplayNode;
```

部分代码实现

- **rightRotate(x)**
  - y := tree[x].father;
  - z := tree[x].right;

  - If(tree[tree[y].father].left = y)
  - then tree[tree[y].father].left := x;
  - else tree[tree[y].father].right := x;
  - tree[x].father := tree[y].father;
  - tree[x].right := y;
  - tree[y].father := x;
  - tree[y].left := z;
  - tree[z].father := y;
  - renewInfo(y); renewInfo(x);