伸展树的
基本操作与应用

芜湖一中 杨思雨

引言

- **二叉查找树(Binary Search Tree)** 可以被用来表示有序集合、建立索引或优先队列等。
- 最坏情况下，作用于二叉查找树上的基本操作的时间复杂度，可能达到O(n)。
- 某些二叉查找树的变形，基本操作在最坏情况下性能依然很好，如红黑树、AVL树等。

伸展树

- 伸展树(Splay Tree)是二叉查找树的改进。
- 对伸展树的操作的平摊复杂度是$O(\log_2 n)$。
- 伸展树的空间要求、编程难度非常低。

伸展树

- 伸展树与二叉查找树一样，也具有**有序性**。

  即伸展树中的每一个节点x都满足：该节点左子树中的每一个元素都小于x，而其右子树中的每一个元素都大于x。

- 伸展树可以**自我调整**，这就要依靠

伸展操作Splay(x,S)

伸展操作Splay(x,S)

- 伸展操作Splay(x,S)是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调整至树的根部的操作。
- 在旋转的过程中，要分三种情况分别处理：

1)Zig 或 Zag

2)Zig-Zig 或 Zag-Zag

3)Zig-Zag 或 Zag-Zig

伸展操作Splay(x,S) 情况1

- Zig或Zag操作：

  节点x的父节点y是根节点。

伸展操作Splay(x,S) 情况2

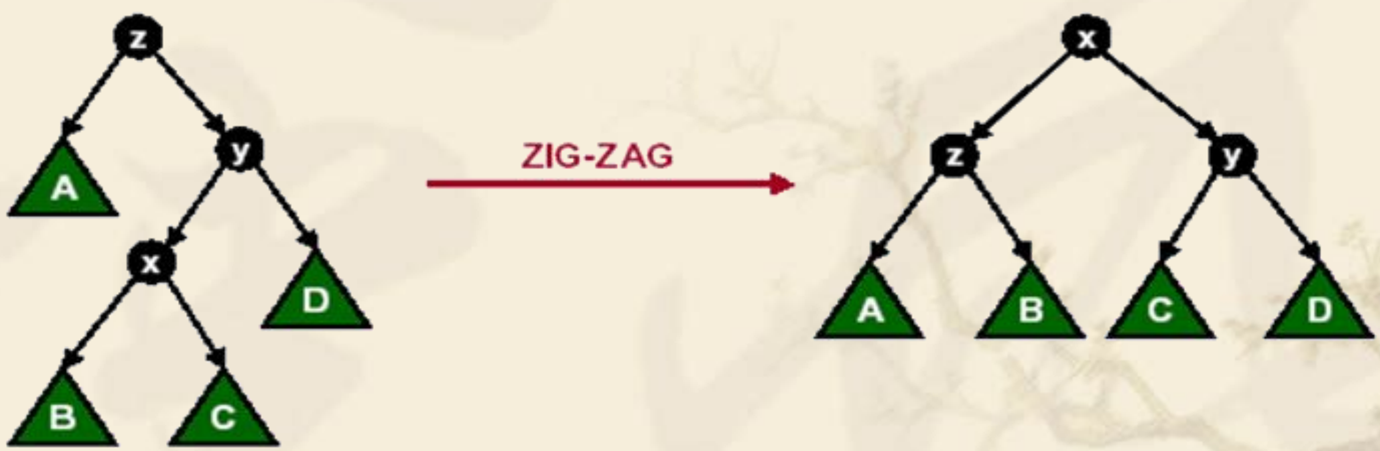
- Zig-Zig或Zag-Zag操作：

  节点x的父节点y不是根节点，且x与y同时是各自父节点的左孩子或者同时是各自父节点的右孩子。

伸展操作Splay(x,S) 情况3

- Zig-Zag或Zag-Zig操作：

  节点x的父节点y不是根节点，x与y中一个是其父节点的左孩子而另一个是其父节点的右孩子。



伸展操作举例

Splay(1,S)

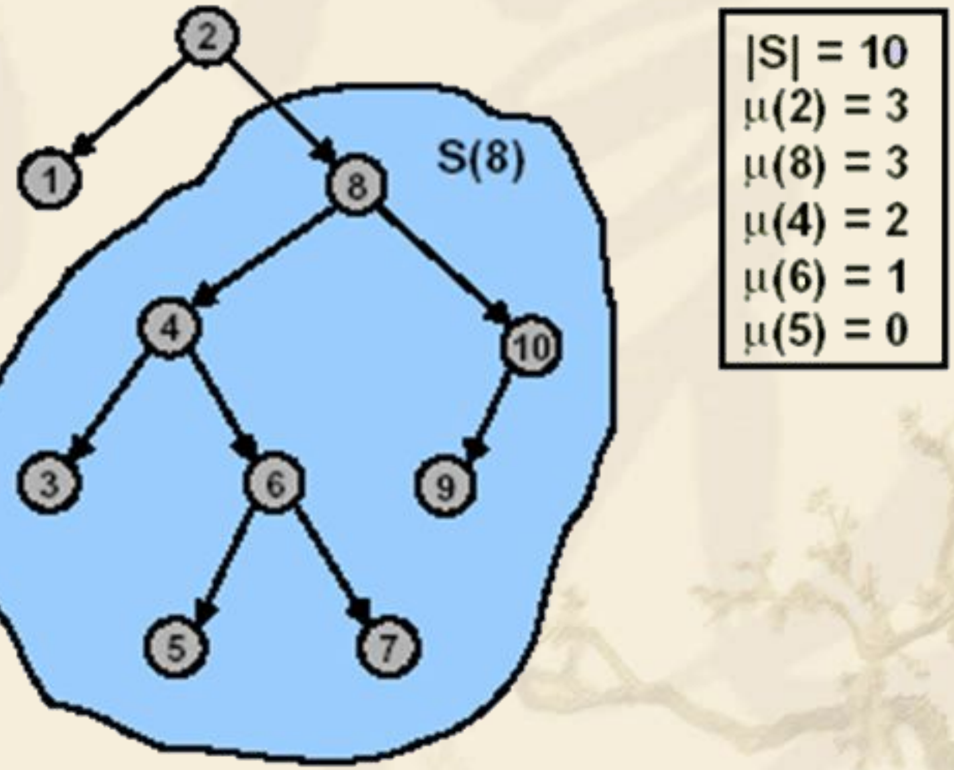
伸展树的基本操作

- 查找
- 插入
- 删除
- 求最大值
- 求最小值
- 求前趋
- 求后继
- 合并
- 分离

伸展操作是基础！

时间复杂度分析

- S(x)表示以x为根的子树
- |S|表示树S的节点个数
- 令 $\mu(S) = [\log_2|S|]$
  ([ ]表示取下整)
- $\mu(x) = \mu(S(x))$



时间复杂度分析—会计方法

对某个节点的访问和旋转

- 用1元钱表示一个**单位时间代价**。

- **伸展树不变量：**在任意时刻，伸展树中的任意节点x都至少有 $\mu(x)$元的存款。

时间复杂度分析

- 在Splay调整过程中，费用将会用在以下两个方面：

  (1)为使用的时间付费。也就是每一次单位时间的操作，要支付1元钱。

  (2)当伸展树的形状调整时，需要加入一些钱或者重新分配原来树中每个节点的存款，以保持不变量继续成立。

时间复杂度分析

- 用μ(x)和μ'(x)分别表示在进行一次旋转操作前后节点x处的存款。
- 分三种情况分析旋转操作的花费：
  (1)Zig 或 Zag
  (2)Zig-Zig 或 Zag-Zag
  (3)Zig-Zag 或 Zag-Zig

时间复杂度分析 情况1

- Zig或Zag

- 为了保持伸展树不变量继续成立，需要花费：

$$
\begin{aligned}
\mu'(x)+\mu'(y)-\mu(x)-\mu(y) &= \mu'(y)-\mu(x) && \Leftarrow \boxed{\mu(y)=\mu'(x)} \\
&\le \mu'(x)-\mu(x) \\
&\le 3(\mu'(x)-\mu(x)) \\
&= 3(\mu(S)-\mu(x)) && \Leftarrow \boxed{\mu'(x)=\mu(S)}
\end{aligned}
$$

此外我们花费另外1元钱用来支付访问、旋转的基本操作。
所以，一次Zig或Zag操作的花费至多为3(μ (S)- μ (x))+1

时间复杂度分析 情况2

- Zig-Zig或Zag-Zag

- 为了保持不变量，需要花费：

$$
\begin{aligned}
\mu'(x)+\mu'(y)+\mu'(z)-\mu(x)-\mu(y)-\mu(z) &= \mu'(y)+\mu'(z)-\mu(x)-\mu(y) \\
&= (\mu'(y)-\mu(x))+(\mu'(z)-\mu(y)) \\
&\le (\mu'(x)-\mu(x))+(\mu'(x)-\mu(x)) \\
&= 2(\mu'(x)-\mu(x))
\end{aligned}
$$

与情况1一样，也需要花费另外的1元钱来支付单位时间的操作。

时间复杂度分析 情况2

- Zig-Zig或Zag-Zag

- 当 $\mu'(x) < \mu(x)$ 时，显然 $2(\mu'(x) - \mu(x)) + 1 \le 3(\mu'(x) - \mu(x))$
  也就是进行Zig-Zig操作的花费不超过 $3(\mu'(x) - \mu(x))$
- 当 $\mu'(x) = \mu(x)$ 时，可以证明：
  $$\mu'(x) + \mu'(y) + \mu'(z) < \mu(x) + \mu(y) + \mu(z)$$
  也就是说我们不需要任何花费保持伸展树不变量，并且可以得到退回来的钱，并用其中的1元支付单位操作的费用。

- 一次Zig-Zig或Zag-Zag的花费至多为 $3(\mu'(x) - \mu(x))$

时间复杂度分析 情况3

- Zig-Zag或Zag-Zig

  与情况2相似，可以证明

  一次Zig-Zag或Zag-Zig操作的花费至多为3 (μ′(x) - μ (x))

时间复杂度分析

Zig	$3(\mu(S)-\mu(x))+1$
Zig-Zig	$3(\mu'(x)-\mu(x))$
Zig-Zag	$3(\mu'(x)-\mu(x))$
⋮	⋮
+	+
Splay(x,S)	$3(\mu(S)-\mu(x))+1$
基本操作	$O(\log_2 n)$

伸展树的应用　例题描述

- 营业额统计Turnover (HNTSC-02)

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$a_i \le 1,000,000$。
最后结果$T \le 2^{31}$。

伸展树的应用　初步分析

- 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。

- 顺序查找前面输入的数

  时间复杂度$O(n^2)$，不能在时限内出解

- 用线段树记录输入的数

  空间要求很大，需要一个大数组

- 红黑树或平衡二叉树

  编程太复杂，调试不方便

伸展树的应用　算法描述

- 这题中，涉及到对于有序集的三种操作：
  插入、求前趋、求后继

- 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。

伸展树的应用　解题小结

- 使用伸展树算法解决本题，时间复杂度为$O(n\log_2 n)$，空间要求不大，编程和调试也都非常容易。

下面的表格对几种算法做出了比较：

	顺序查找	线段树	AVL树	伸展树
时间复杂度	$O(n^2)$	$O(n\log_2 a)$	$O(n\log_2 n)$	$O(n\log_2 n)$
空间复杂度	$O(n)$	$O(a)$	$O(n)$	$O(n)$
编程复杂度	很简单	较简单	较复杂	较简单

总结

- 伸展树有以下三个优点：

1) 时间复杂度低，伸展树的各种基本操作的平摊复杂度都是$O(\log_2 n)$的。

2) 空间要求不高，伸展树不需要记录任何信息以保持树的平衡。

3) 算法简单。编程容易，调试方便。

总结

- 在信息学竞赛中，我们不能一味的追求算法有很高的时间效率，而应该合理的选择算法，找到时间复杂度、空间复杂度、编程复杂度三者之间的平衡点

多做比较

灵活应用

反复琢磨

Thank you all!

谢谢

希望能和大家多交流

E-mail:

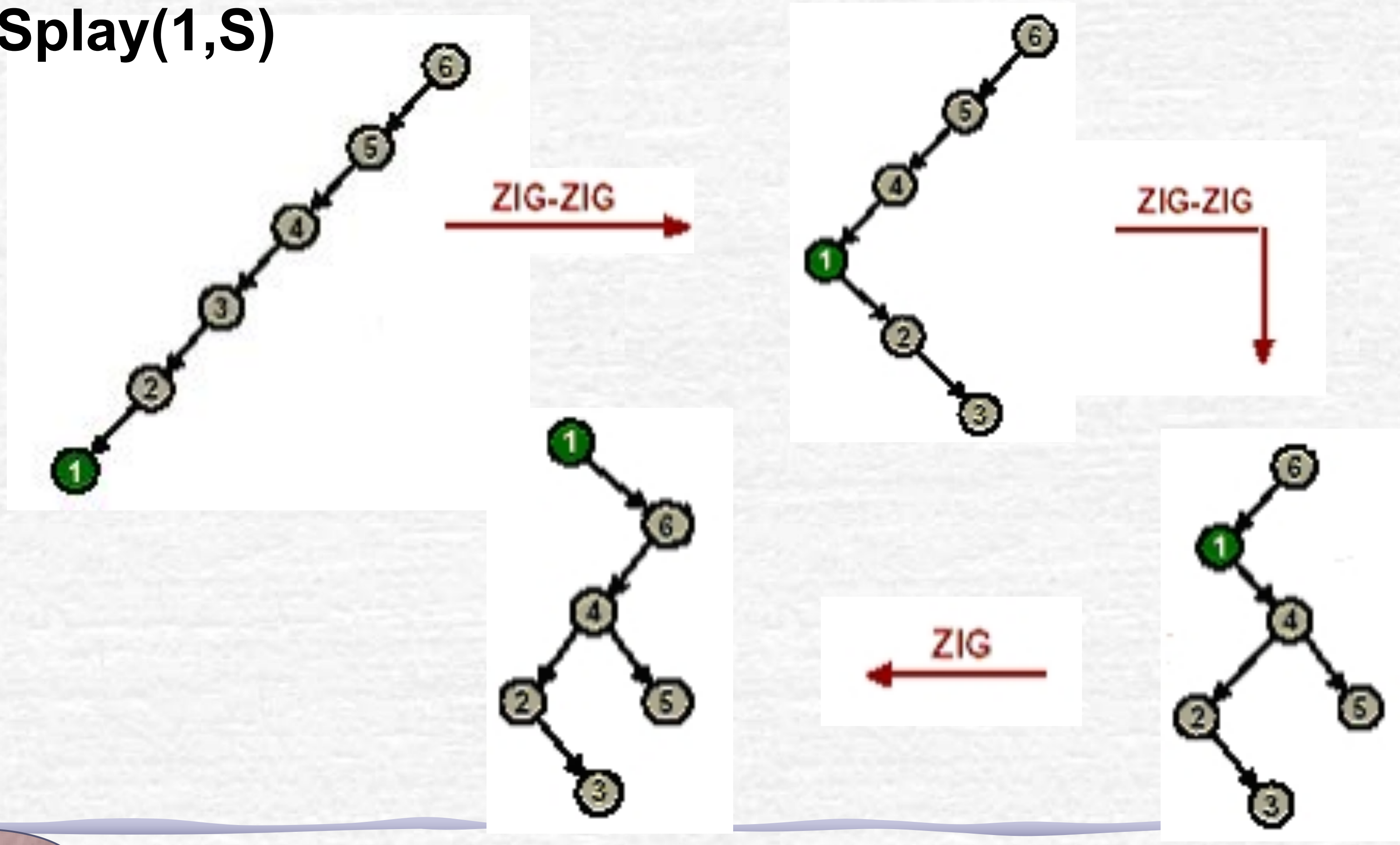
DeadFishYSY@163.com

一些解释

- 伸展操作 例1
- 伸展操作 例2
- 合并操作
- 分离操作
- 情况2时间效率
- 平摊复杂度
- Splay Tree vs AVL Tree

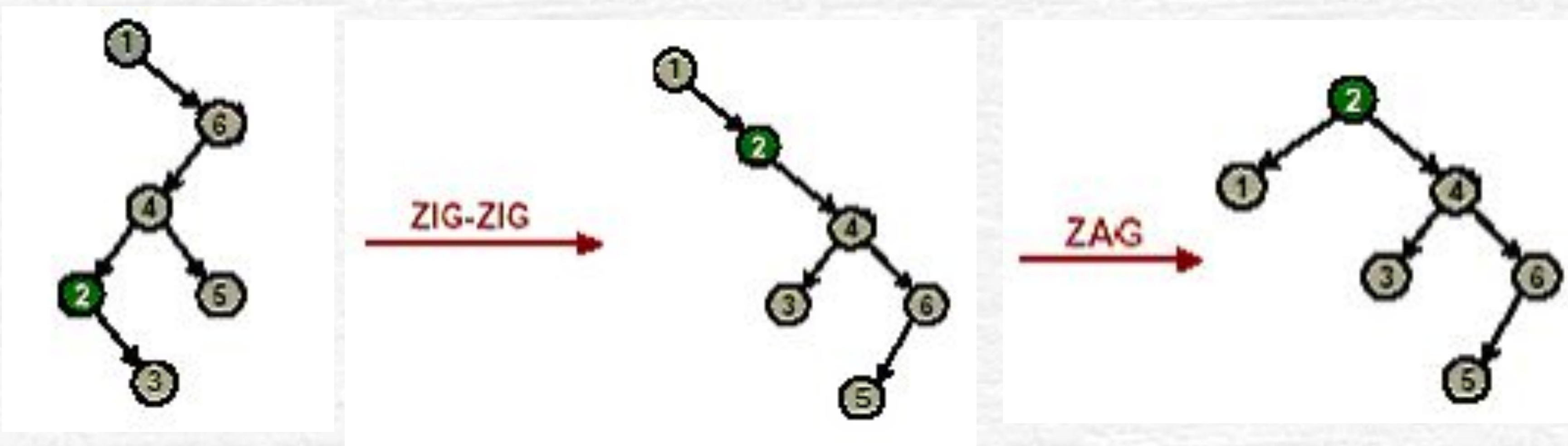
伸展操作 例1

Splay(1,S)



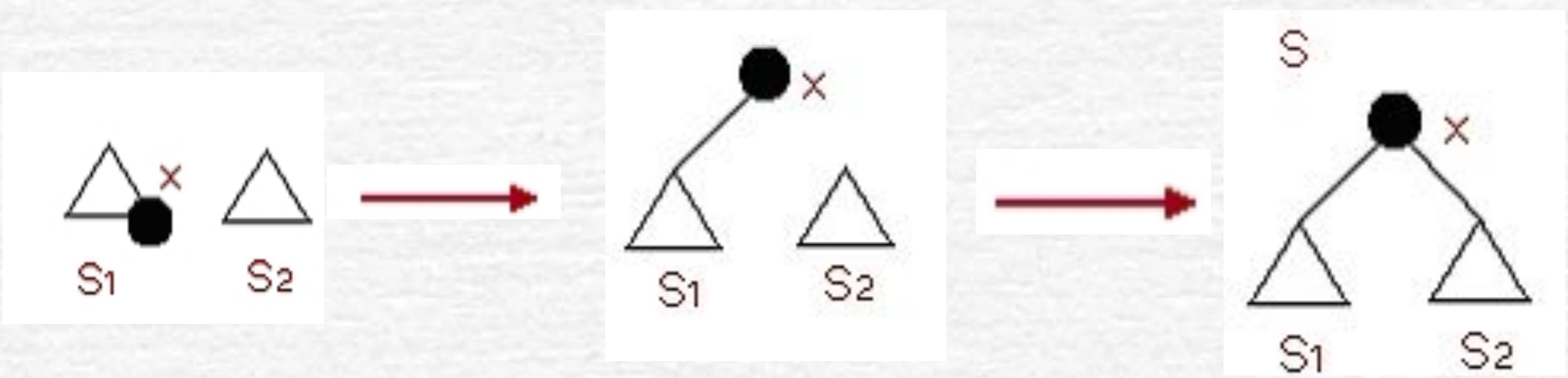
伸展操作 例2

Splay(2,S)



合并操作

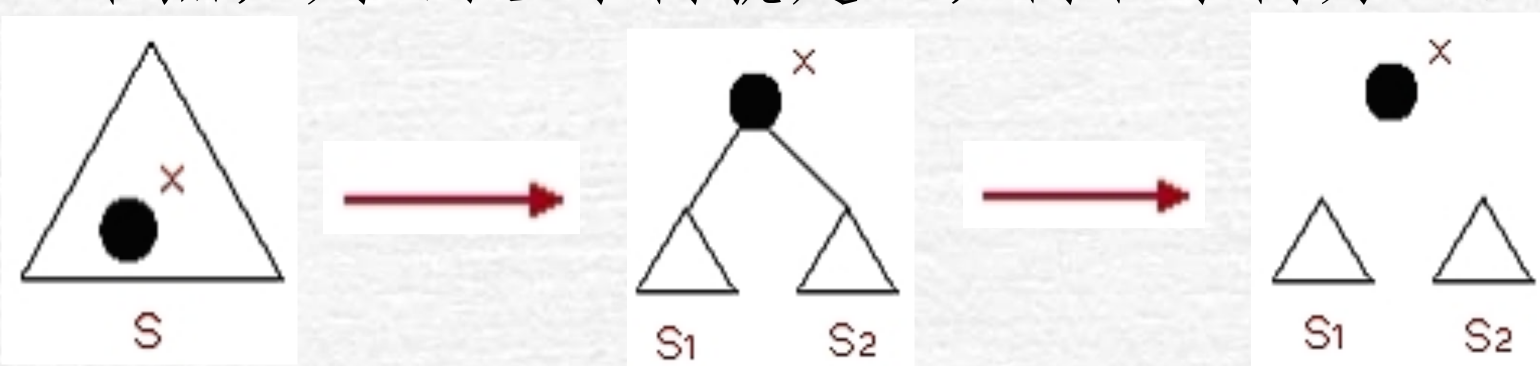
- Join(S1,S2)：将两个伸展树S1与S2合并。其中S1的所有元素都小于S2的所有元素。
- 首先，找到伸展树S1中的最大元素x，再通过Splay(x,S1)将x调整为S1的根。然后将S2作为x节点的右子树。这样，就得到了新伸展树S。



分离操作

- Split(x,S)：以x为界，将伸展树S分离为两棵伸展树S1和S2，其中S1中所有元素都小于x，S2中的所有元素都大于x。

首先执行Find(x,S)，将元素x调整为伸展树的根节点，则x的左子树就是S1，而右子树为S2。



情况2的时间效率分析

- 花费不超过2 (μ'(x) - μ(x)) +1
- 当 μ'(x) < μ(x) 时，显然不超过3 (μ'(x) - μ(x))
- 当 μ'(x) = μ(x) 时，可证明

  μ'(x) + μ'(y) + μ'(z) < μ(x) + μ(y) + μ(z)

图中， μ(x) = μ'(x) = μ(z)。

显然 μ(x) = μ(y) = μ(z)。并且可以得出 μ(x) = μ'(z) = μ(z)。

令a = 1 + |A| + |B|，b = 1 + |C| + |D| 。那么

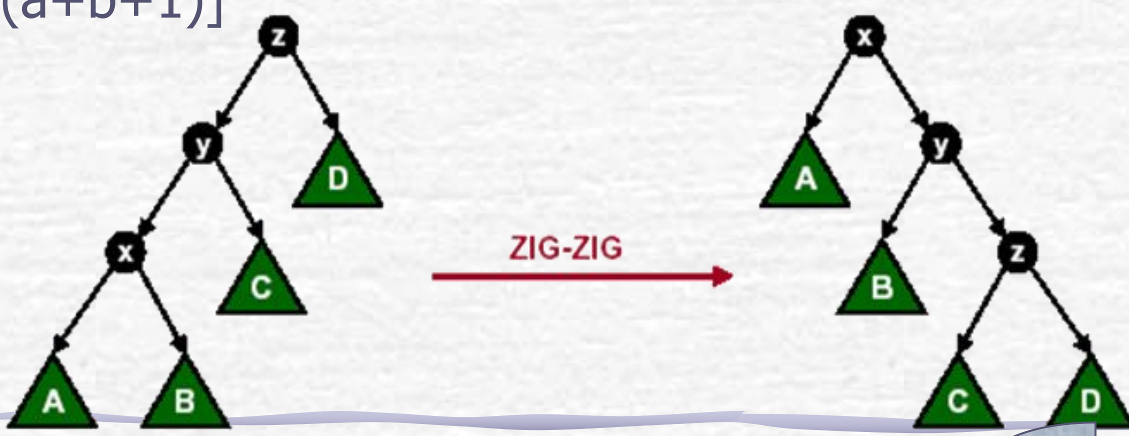
[log a] = [log b] = [log (a+b+1)]

不妨设b≥a，则有

[log (a+b+1)] ≥ [log (2a)]

= 1+[log a]

> [log a]



关于时间复杂度

- 伸展树操作中，
  被访问越多的节点越接近根节点。

- 一个结论：
  对于伸展树的一系列n个操作，其中关于节点x的有m个，那么这m条操作的平摊复杂度为$O(\log_2(n/m))$。

伸展树 Vs 平衡二叉树

- 不需要记录其他信息
- 编程复杂度低
- 合并、分离操作时间复杂度$O(\log_2 n)$

- 要记录平衡因子
- 编程较复杂
- 不支持直接的合并操作和分离操作

Thank you