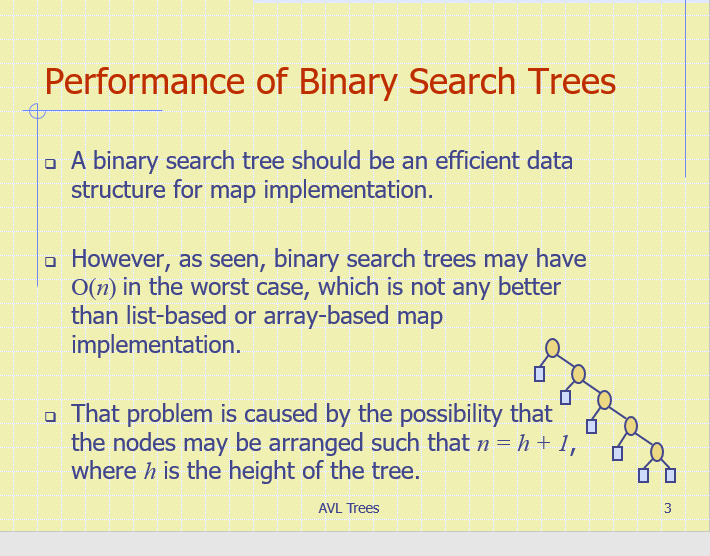
AVL TREES



一个binary search tree 应该是implement map的有效结构

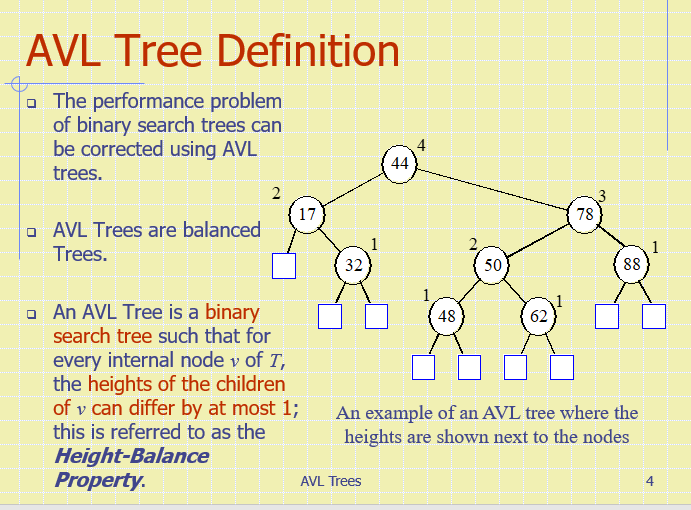
然而，binary search tree最差的情况复杂度是O（n）,并不比list based 或array-based map更好

这个问题是因为node的分布导致的，他们有可能让n之比h大1

比如一条线走到黑

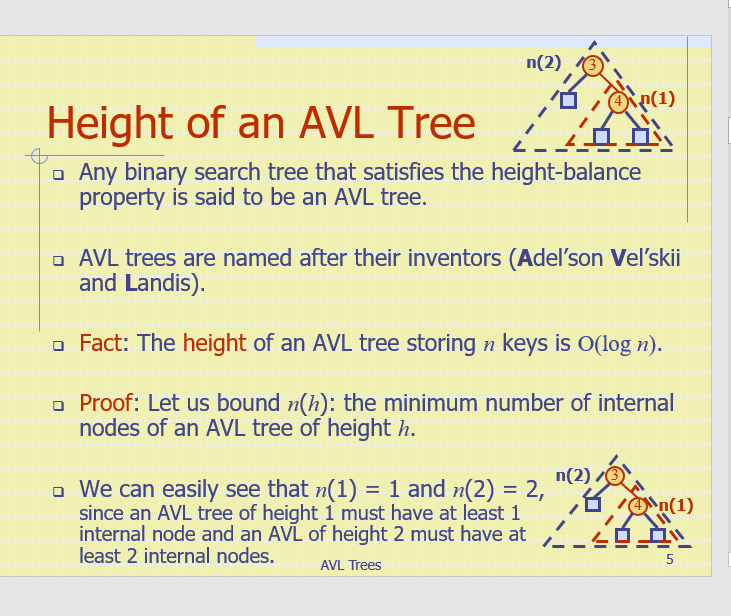
使用AVL Tree 可以规避这个问题

avl tree首先是balanced trees



AVL tree 是binary search tree的一种，每一个对于T的每一个Internal nodev，他们children的高度最多差1，这叫做height-balance property

1. 总共的height 是4，17的height 是2，78的height是3，对于44的孩子，最多差1



AVL tree的height。

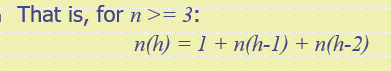
任何满足height-balance property的binary search tree就叫做AVL tree

FACT: 一个存储了n个key的avl tree的height是Ologn

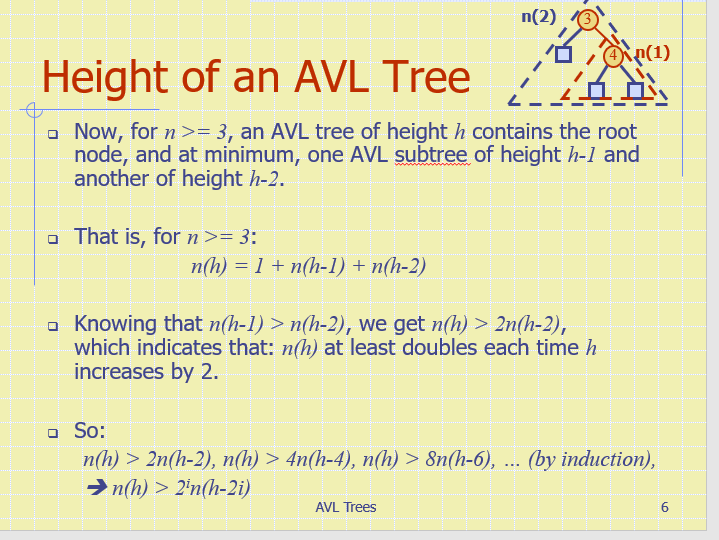
proof：假设nh是高度为h的avl tree所存储的最小数量internal node

n(1)=1,n(2)=2, 因为height 为1 至少有一个internal node，height2 为·2至少有两个internal node

那么当n>=3时，假设AVL tree 的height是h，那么他的子树height至少也是h-1和h-2，（因为是子树，所以少了root那一层，是h-1,另外一个H-2最多也只能比h-1小一层AVL限制）

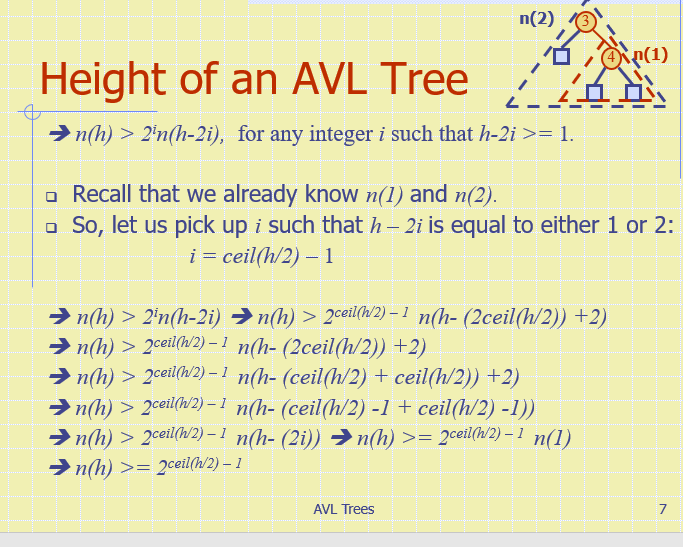


h高度的·internal node=自己本身root+H-1高度的internal+h-2的internal

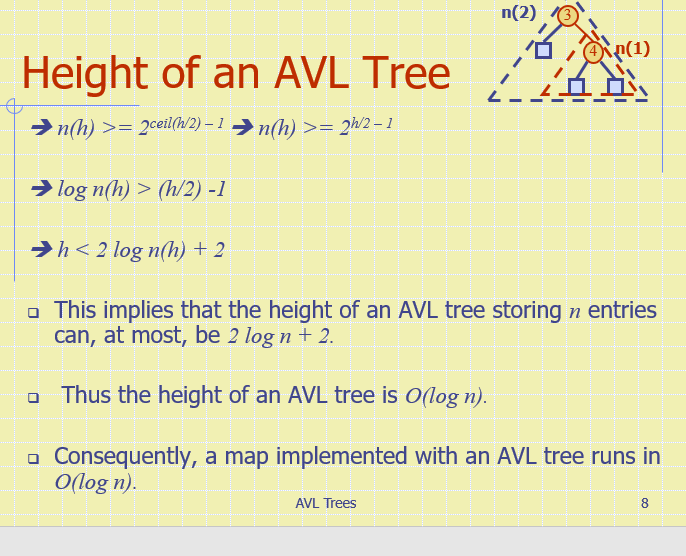


n(h-1)的node数量必然大于nh-2的Node数量，

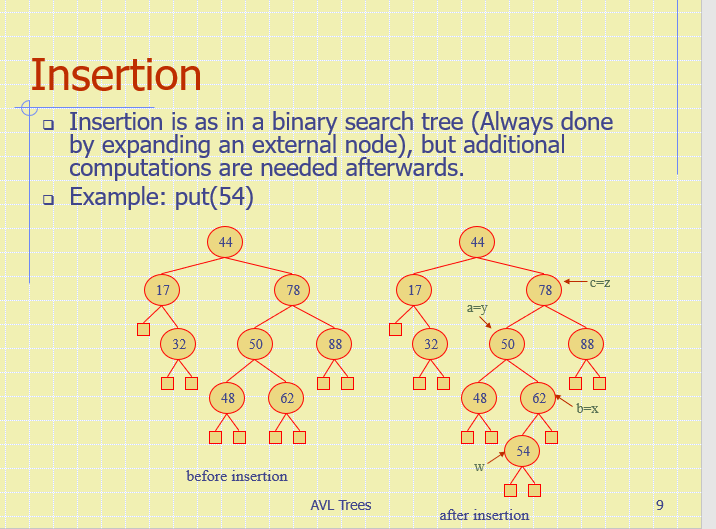
就无限循环

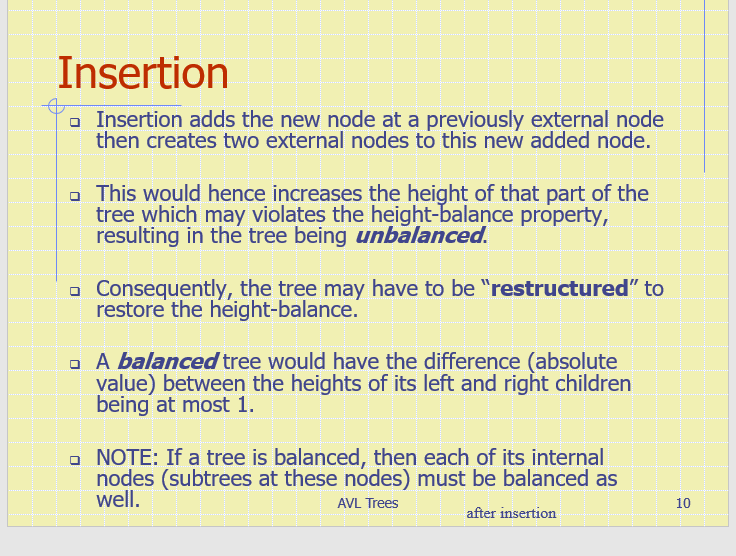


h-2i>=1



insertion, 与binary search tree相同（都是拓展external node），但需要后续计算





insertion在之前的external node上加了一个新Node，这样就创造了两个新external node，这就让tree的某些subtree高度产生变化，有可能违背height-balance property，让这个ttee变得unbalanced<

最终结果是这个tree必须被重新构建一次

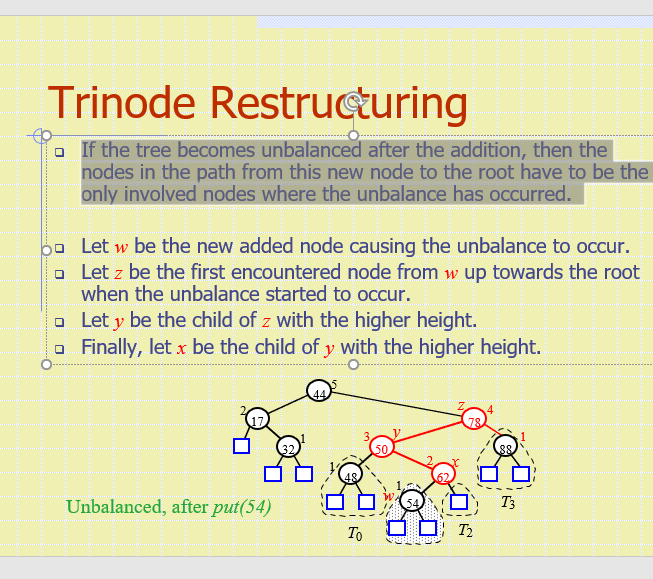
一个balanced tree:subtree高度差必须小于等于1

trinode restructing

如果一个tree在插入后变的unbalanced，那么新插入的node必然是这个tree的unbalanced 部分

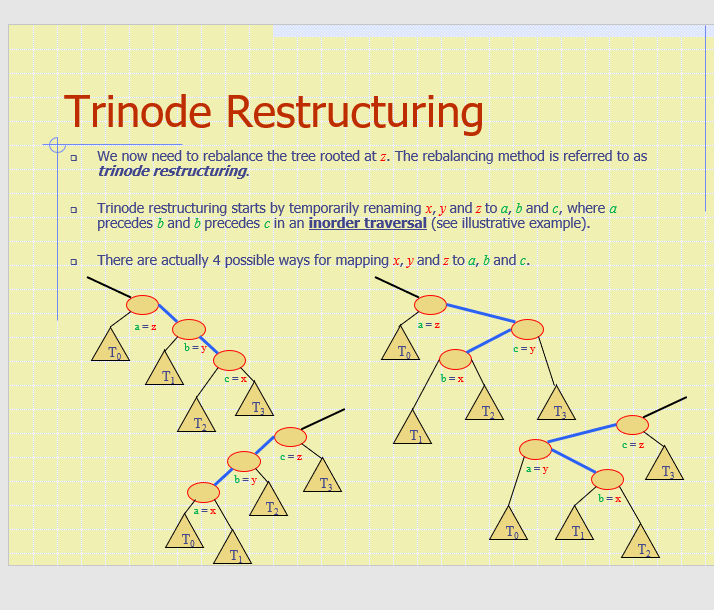
让w作为新节点，z是第一个他从root往下遇到的节点，y是z的子孙，但是y相比别的子孙有着更高的height，

最后，让x作为y的子孙，x有着更高Height

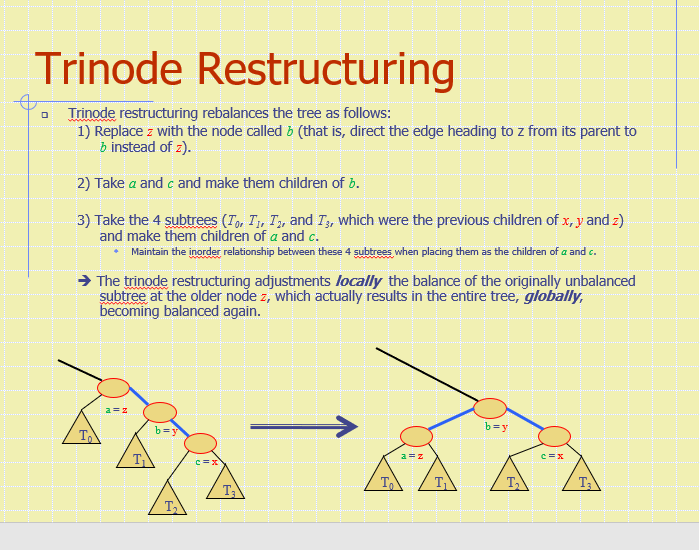


我们现在需要rebalance这个tree， 这个rebalance方法叫做trinode restructuring

trinode restructuring 暂时把ZXY命名成ABC,，顺序是a->b->c， inordertraversal



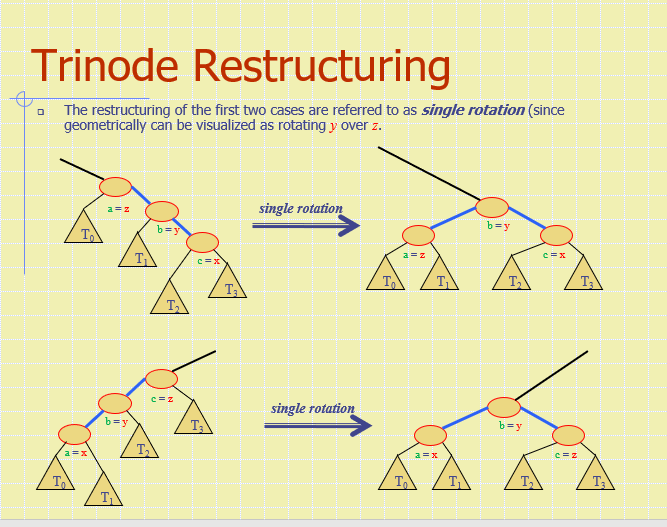
图片是把xyz赋值给abc的四种实例



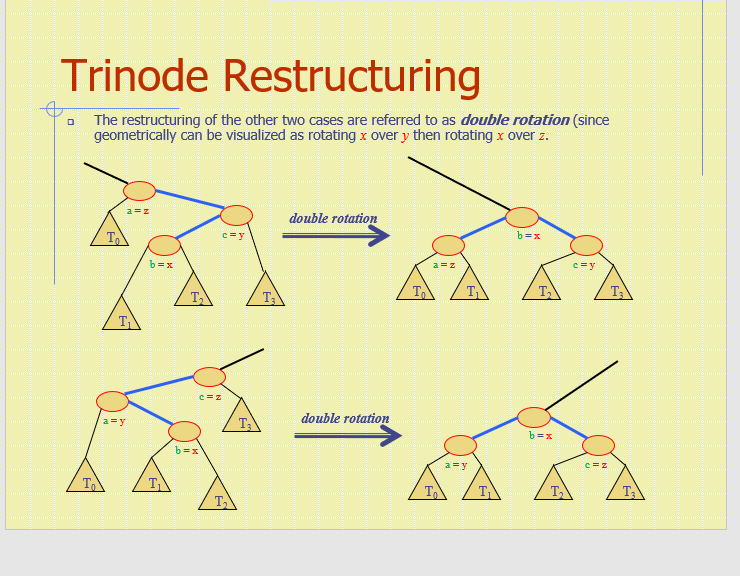
1. 把Z所在的位置用Bnode代替，
2. 让A和C成为Bnode的children
3. 让a bc之前的children, T0,T1,T2,T3,成为a与C的children
4. 这样整体结构就平衡了

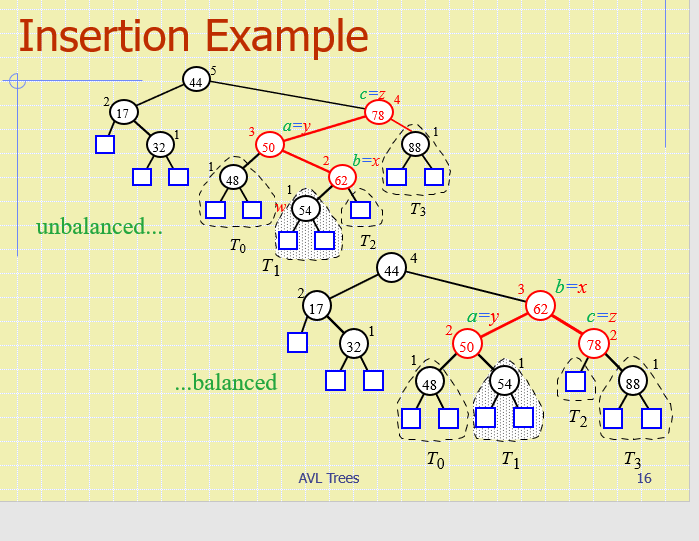
而这四种情况又分为两类

single retation:一条道走到黑的那种，可以看成是把y拉上来，ac下去，



double rotation:本身有转弯，把最底下的b=x提到最上面，然后左a右c



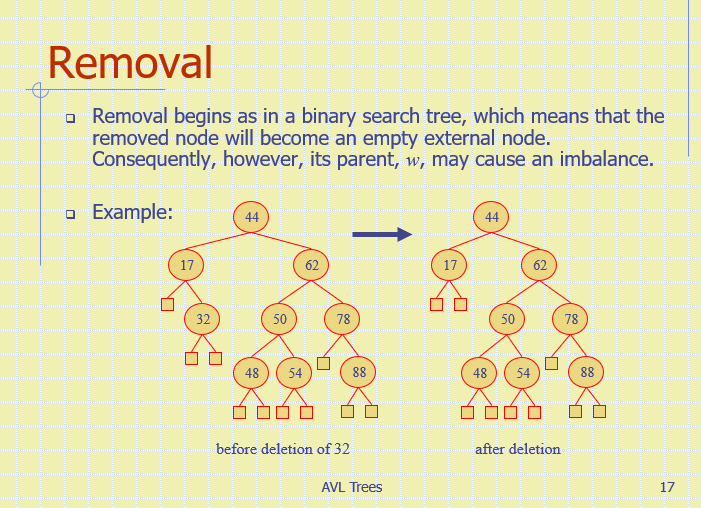
例子，我们要插入54，从上到下，大于44，右，小于78左，大于50右，小于62，左

cab，以inorder的话就是，50，62，78，

因为是double rotation,b往上拉，a左c右

Removal

假设remove 32,会产生 Unbalanced



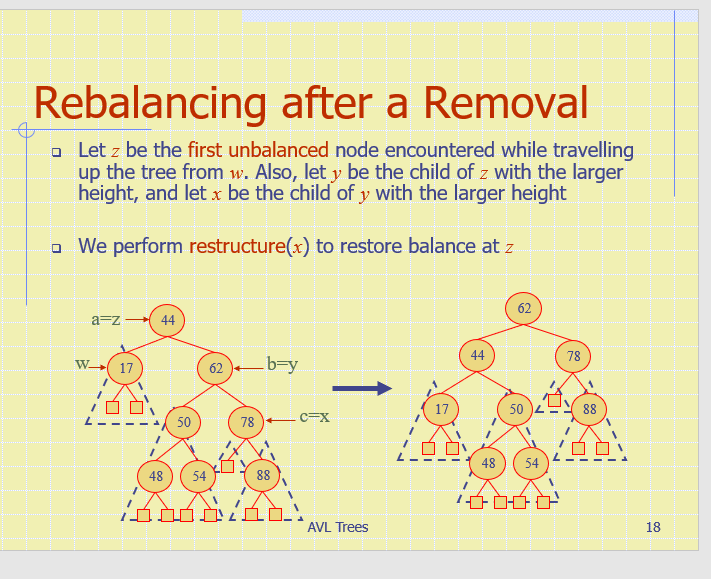
让z作为当从上到下travel时第一个遇到的unbalanced node，

Y作为Z的有着较高height的child

x作为Y·...

然后再restructure...，

设置abc, inorder,instructure，因为是single，所以b往上a往下



注意这里分配T0T1T2,A自己的左边不变，右边换位B原来的左边，C不变