HNFMS

平衡树

授课人：长沙市一中 周祖松

营业额统计(turnover.cpp）

【问题描述】账本上记录了公司成立以来每天的营业额。

当最小波动值越大时，就说明营业情况越不稳定。而分析整个公司的从成立到现在营业情况是否稳定，只需要把每一天的最小波动值加起来就可以了。你的任务就是编写一个程序计算这一个值。第一天的最小波动值为第一天的营业额。（ n<=32767，营业额a<= 1000000）

- 输入
- 6
- 5 1 2 5 4 6
- 输出
- 12
- 结果说明：
- 5+|1-5|+|2-1|+|5-5|+|4-5|+|6-5|=5+4+1+0+1+1=12

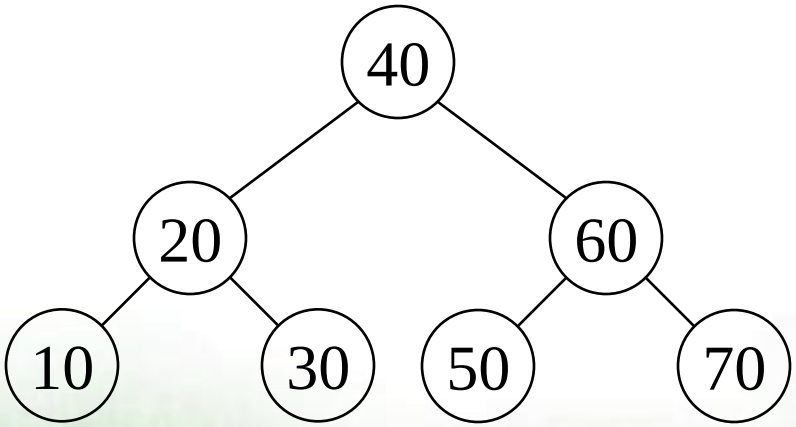
链表(反向处理,跳跃表)

线段树

平衡树

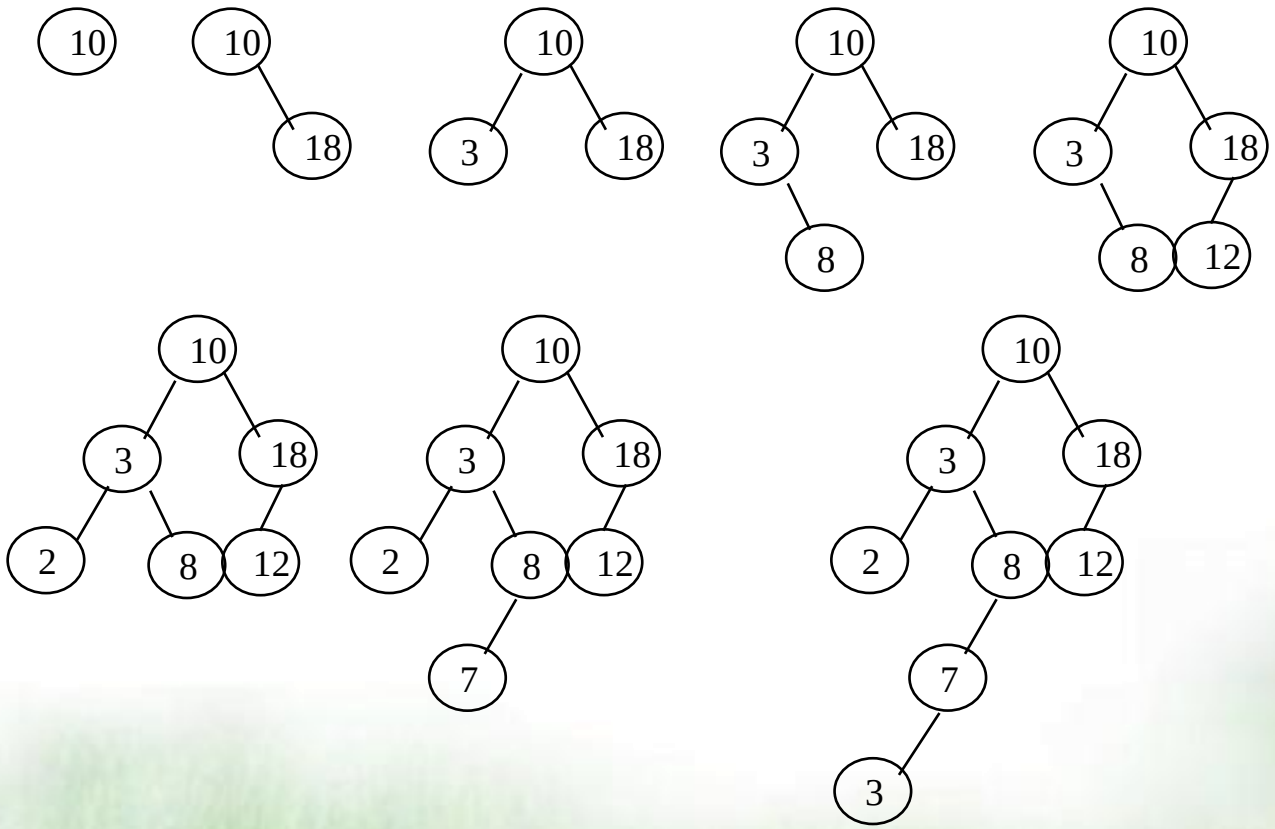
– 二叉查找树

- 定义：二叉排序树或是一棵空树，或是具有下列性质的二叉树：
  - 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值
  - 若它的右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值
  - 它的左、右子树也分别为二叉排序树



– 插入算法

例 {10， 18， 3， 8， 12， 2， 7， 3}



- 二叉排序树的删除

要删除二叉排序树中的p结点，分三种情况：

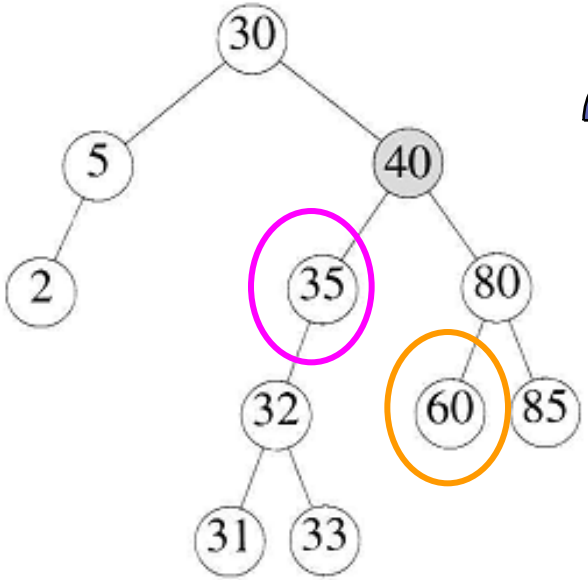
– (1) p为叶子结点

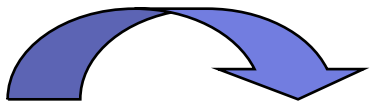
– (2) p只有左子树或右子树

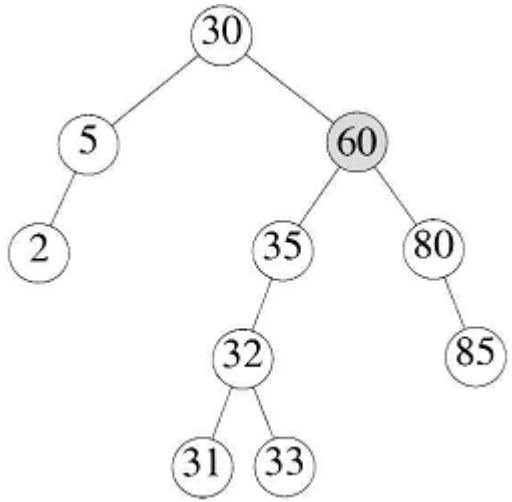
– (3)p左、右子树均非空

情况3-删除示例

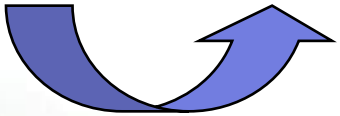
- 删除关键值为40的节点

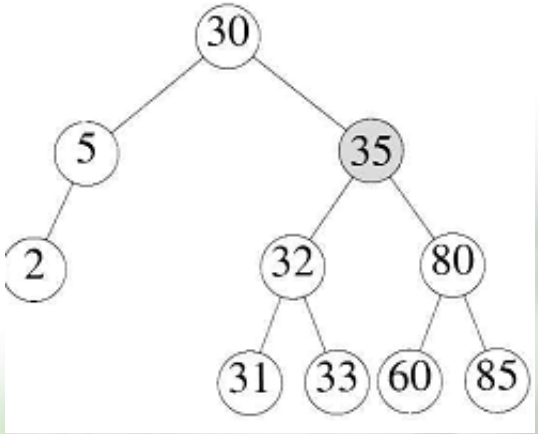






右子树中的最小元素

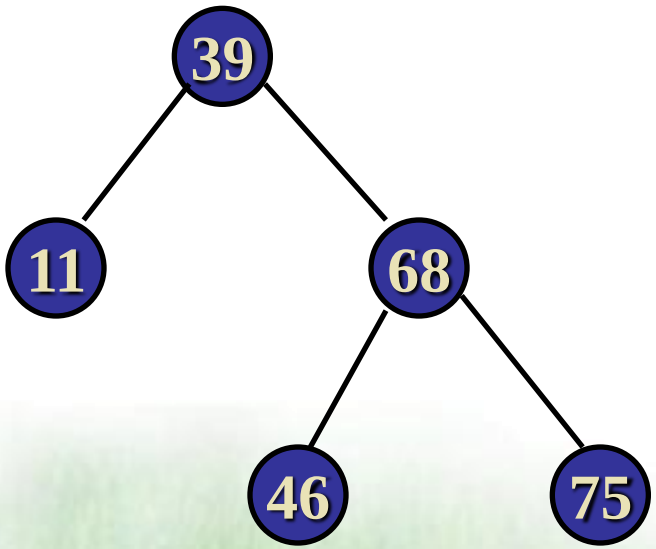


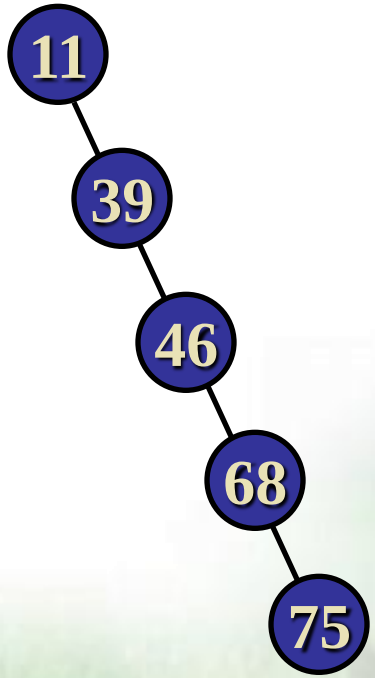


左子树中的最大元素

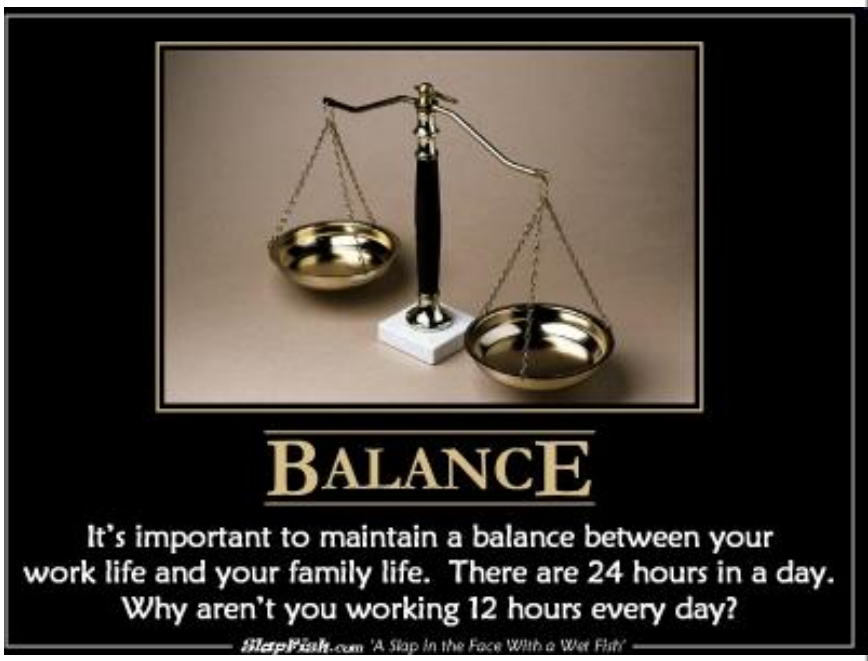
BST的高度

- 含有n个结点的二叉搜索树不是唯一的，从而树的高度就不一定相同。
- 一棵n元素的二叉搜索树的高度可以与n一样大。





平衡树



- 0、暴力平衡(替罪羊树)
- 1、高度平衡(avl)
- 2、重量平衡（treap）
- 3、自动平衡（红黑树，伸展树）

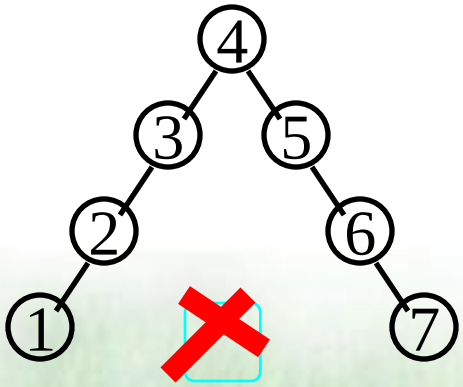
Adelson-Velskii-Landis (AVL) Trees (1962)

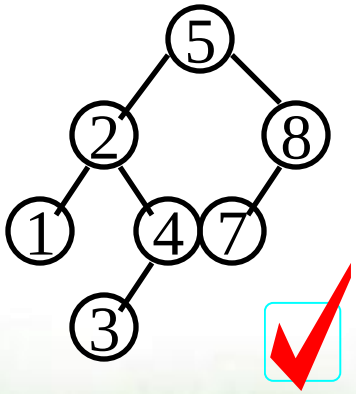
【Definition】 An empty binary tree is height balanced. If T is a nonempty binary tree with T_L and T_R as its left and right subtrees, then T is height balanced iff

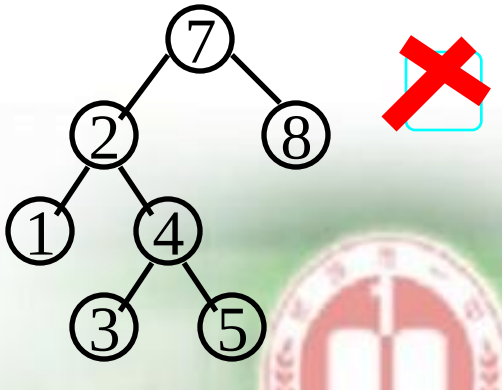
(1) T_L and T_R are height balanced, and

(2) $|h_L - h_R| \leq 1$ where h_L and h_R are the heights of T_L and T_R, respectively.

【Definition】 The balance factor $BF(\text{node}) = h_L - h_R$. In an AVL tree, $BF(\text{node}) = -1, 0, \text{or } 1$.

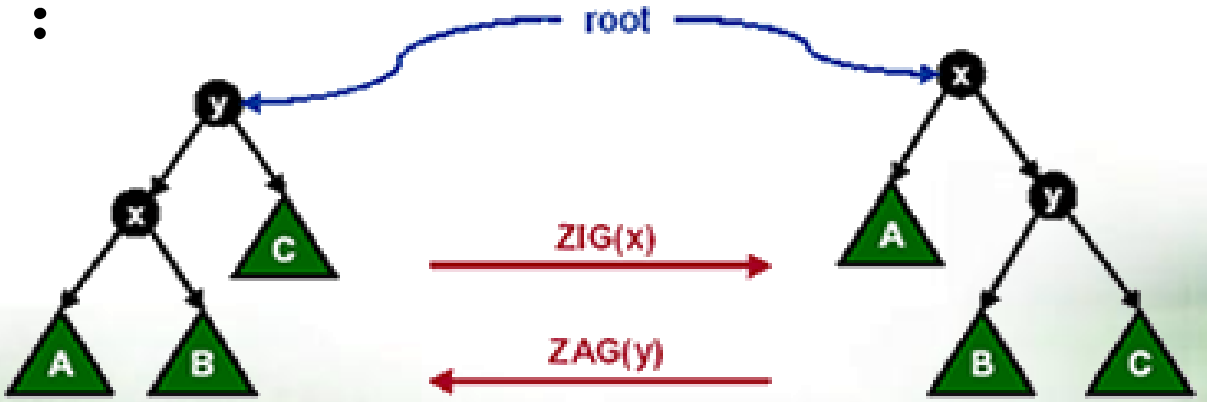




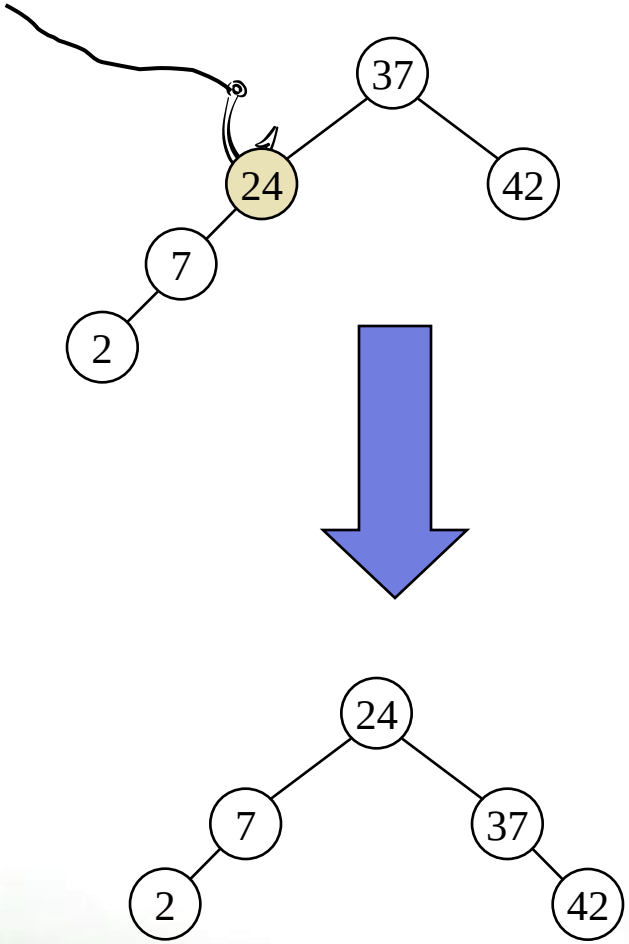


二叉排序树

- 旋转操作是二叉排序树的众多变种的一个共同的理论基础
- 下图是右旋操作示意图(反之就是左旋)：

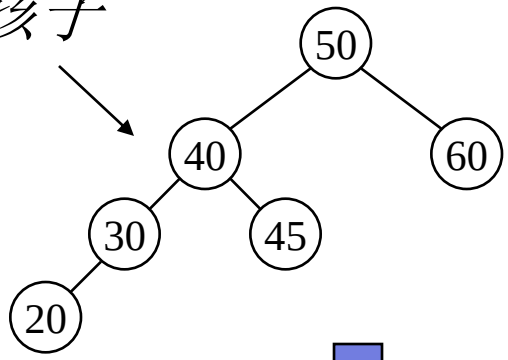


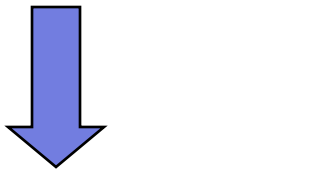
一个不平衡的BST

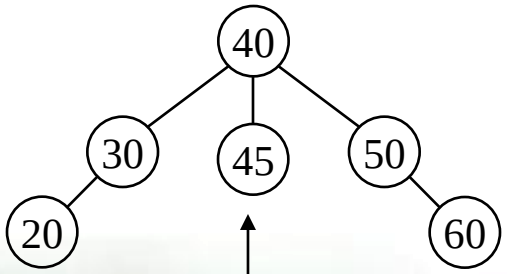


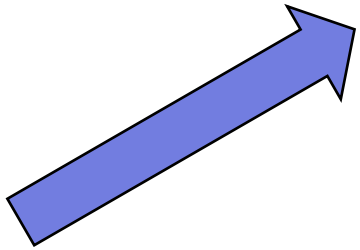
两个孩子的情况

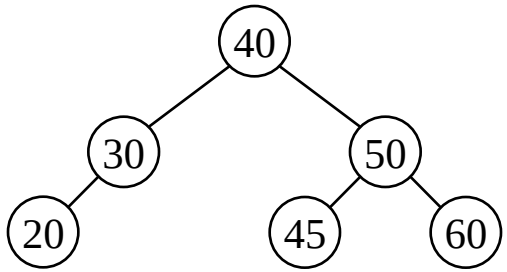
S有两个孩子







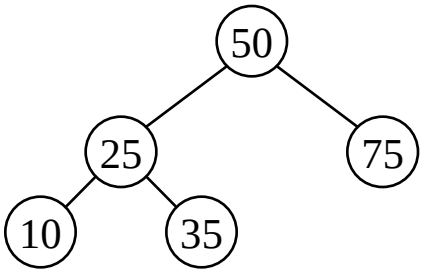




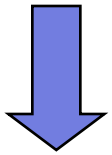
把它放哪里?

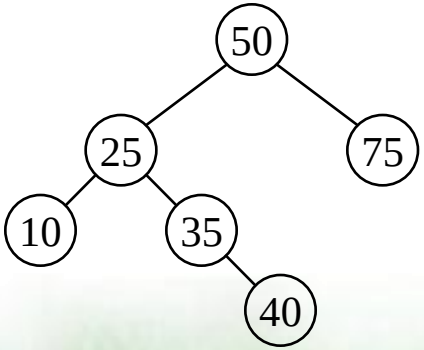

```
void rotate(int x)
{   int y=fa[x],g=fa[y],c=child[y][1]==x;
    child[y][c]=child[x][c^1]; fa[child[y][c]]=y;
    child[x][c^1]=g; fa[y]=x;
    fa[x]=g;
    if(g)
        child[g][child[g][1]==y]=x ;
}
```

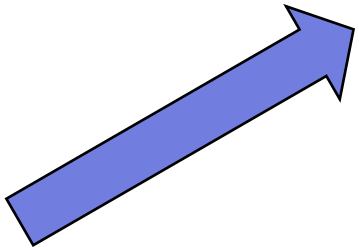
一个单旋??

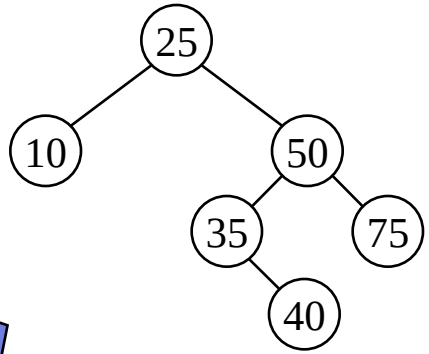


Insert 40



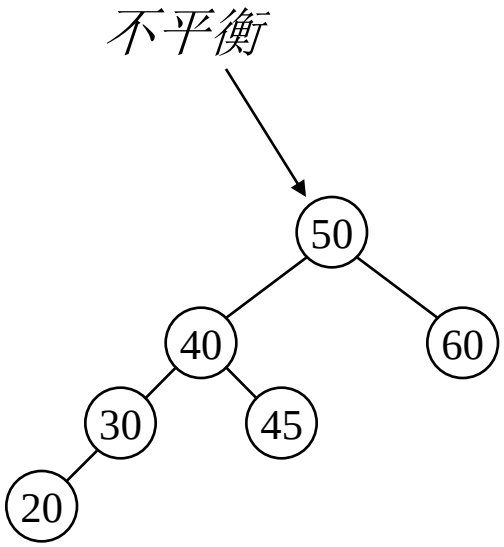


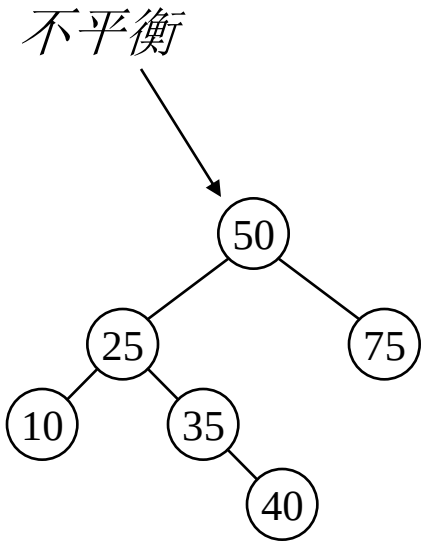




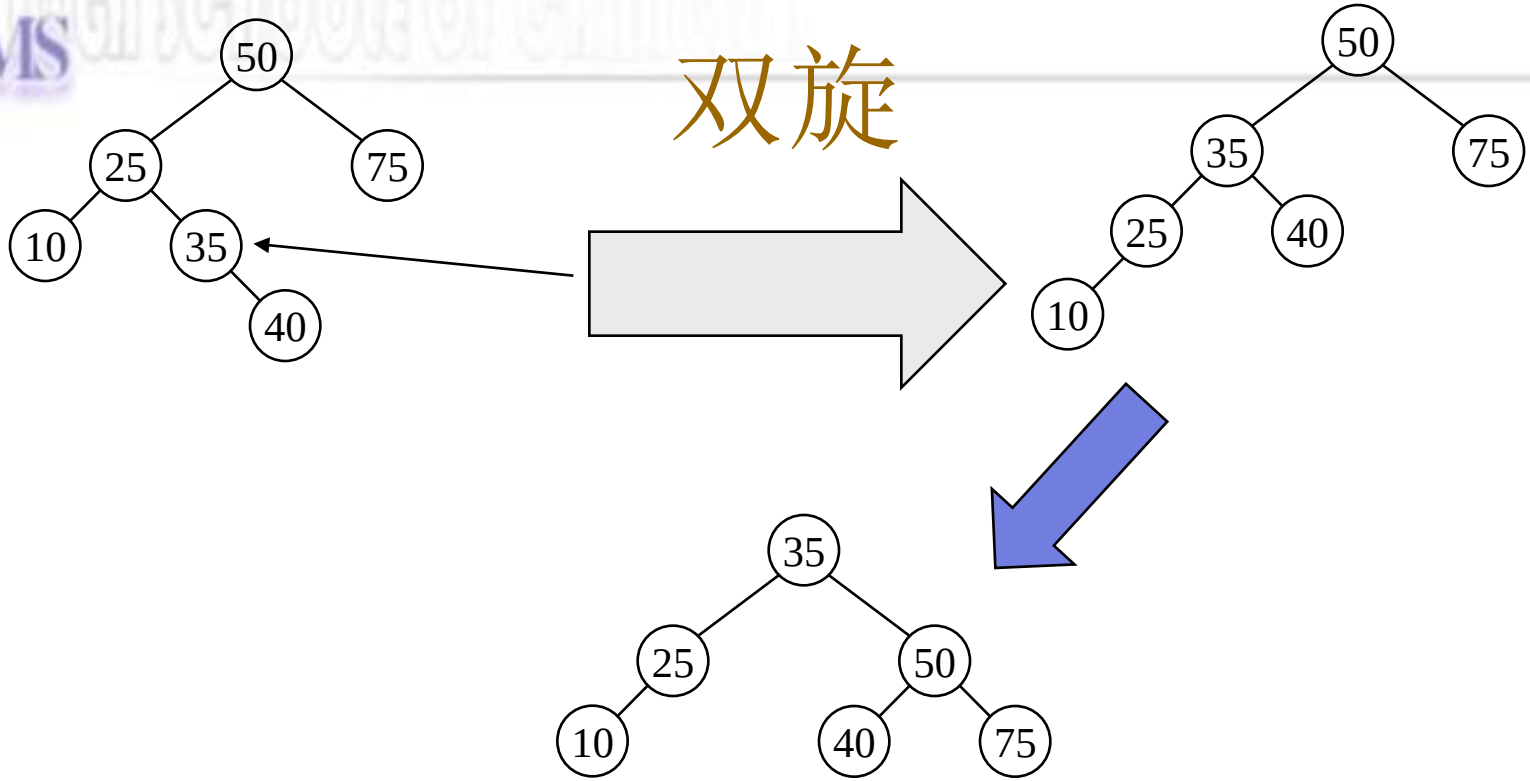
还是不平衡!!

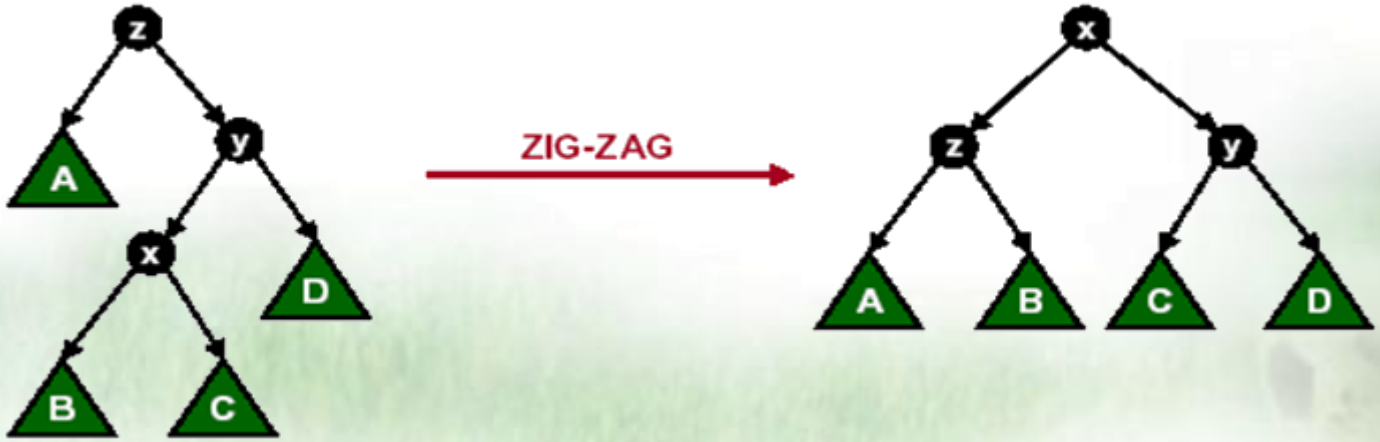
它们的差别是什么?



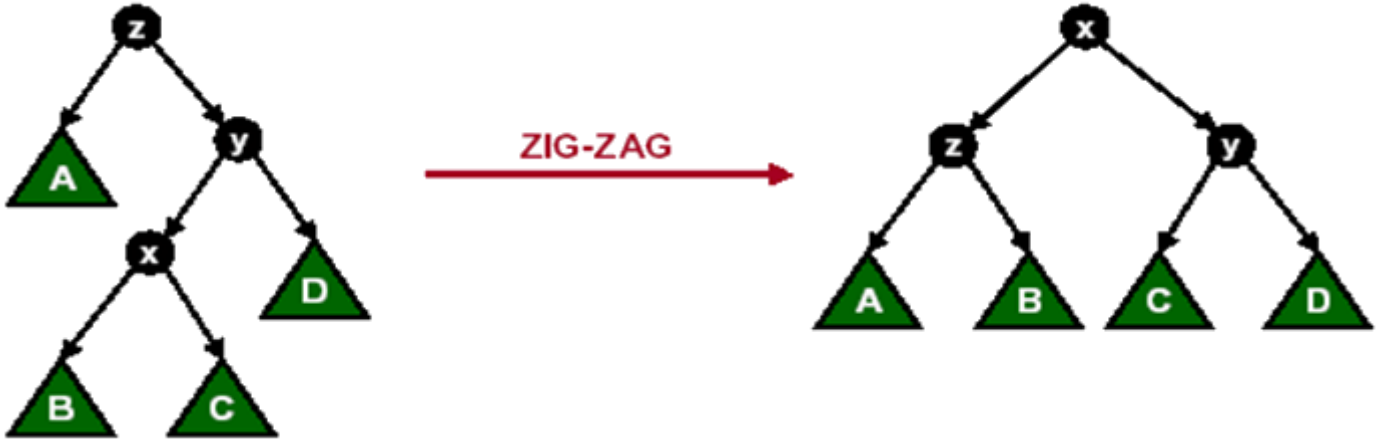


双旋


50
25
75
10
35
40
50
35
75
25
40
10
35
25
50
10
40
75

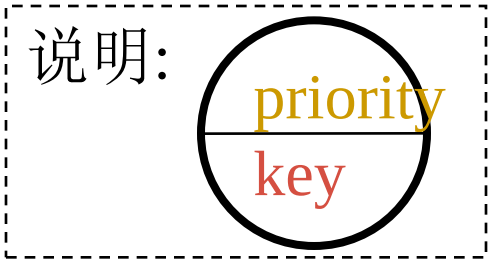

z
A
y
x
D
B
C
ZIG-ZAG
x
z
y
A
B
C
D

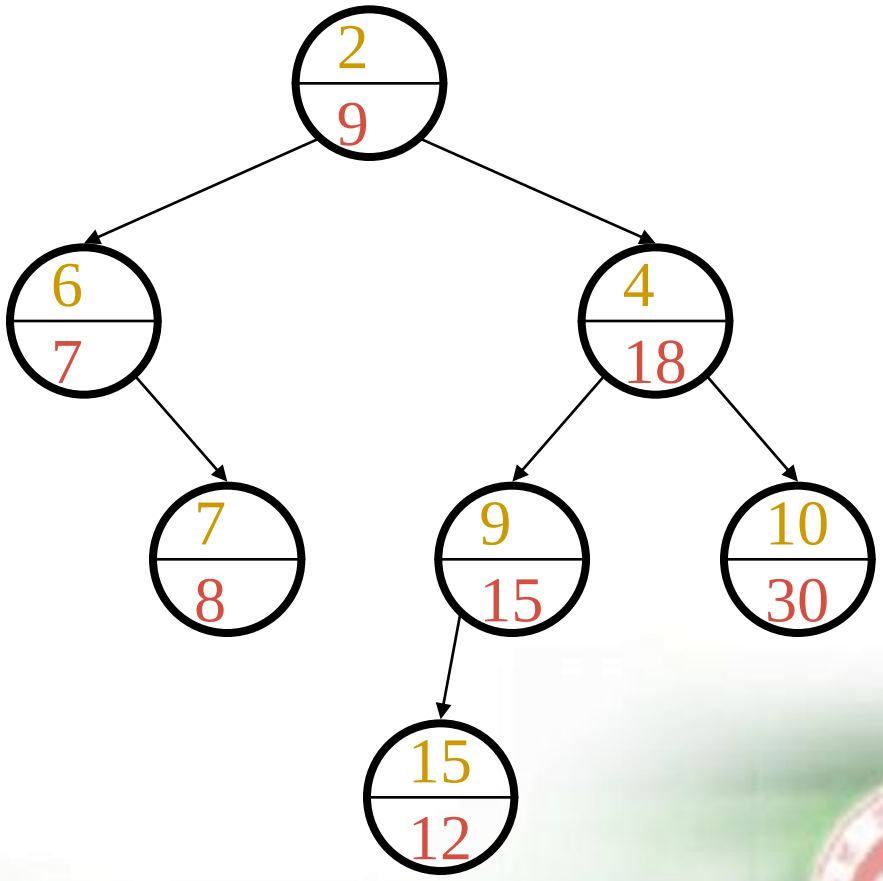
双旋



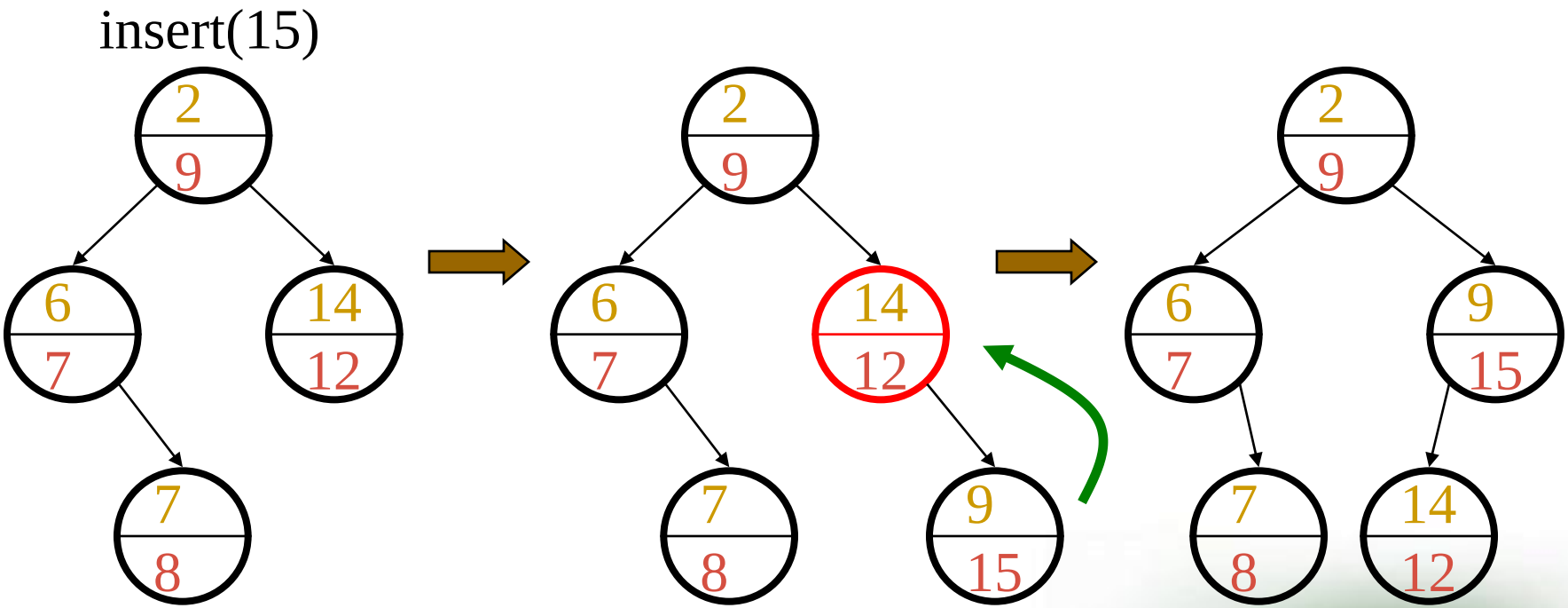

```
void double-rotate(int x)
{   rotate(x);
    rotate(x);
}
```

Treap





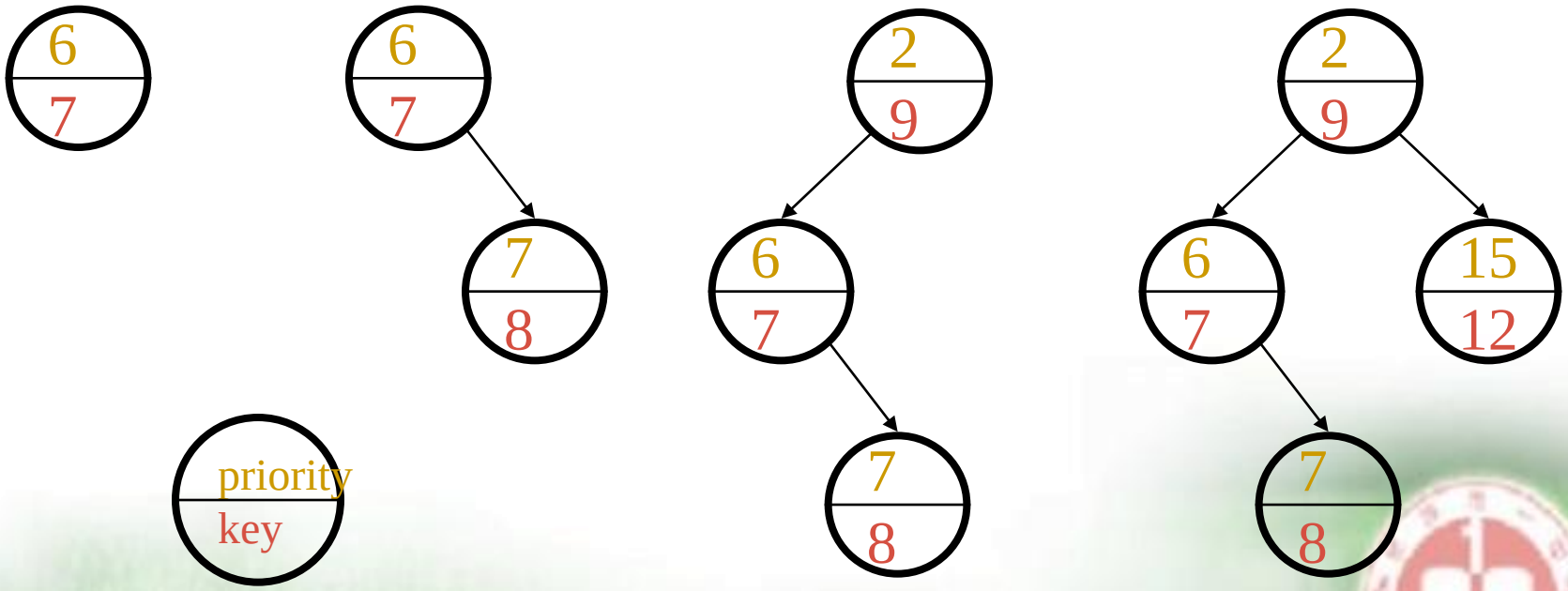
Treap Insert



Tree + Heap

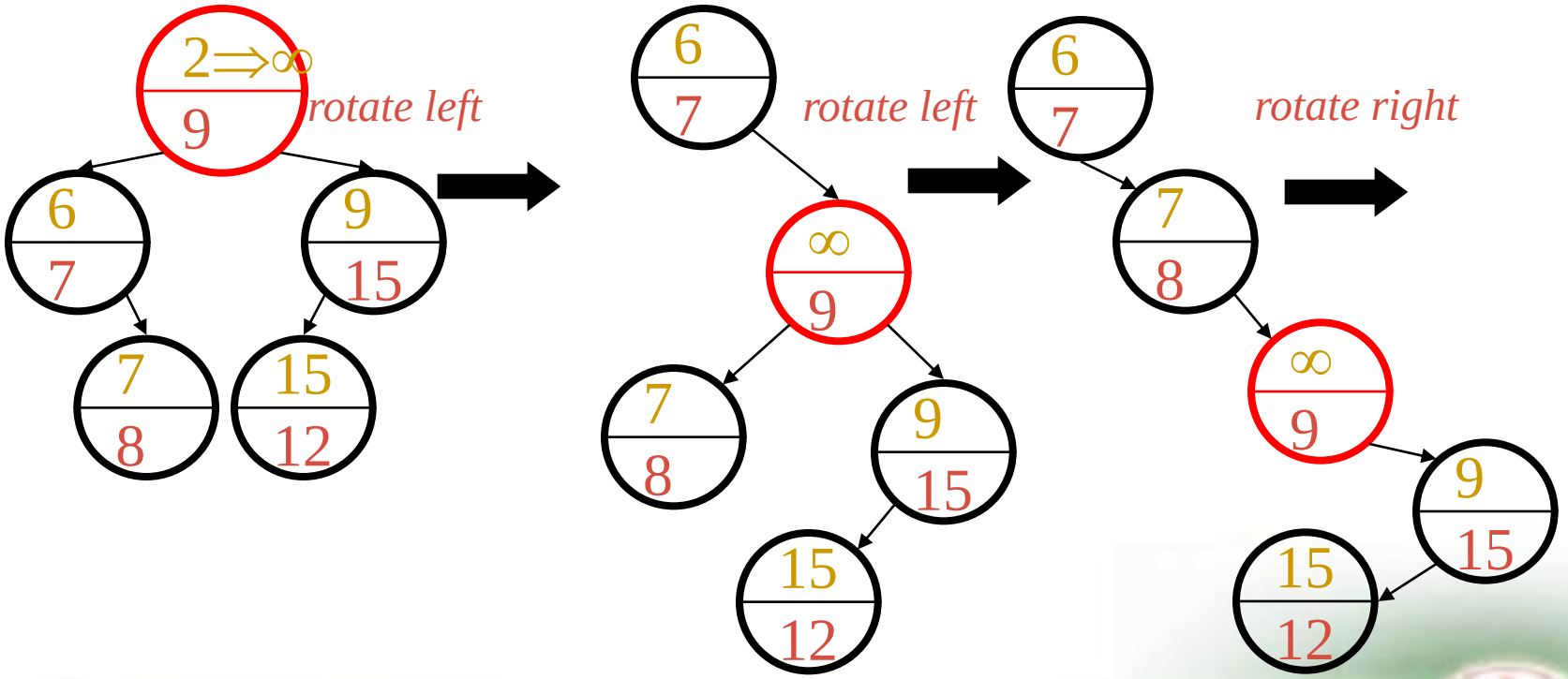
按顺序插入,结果会是什么样子?

insert(7) insert(8) insert(9) insert(12)

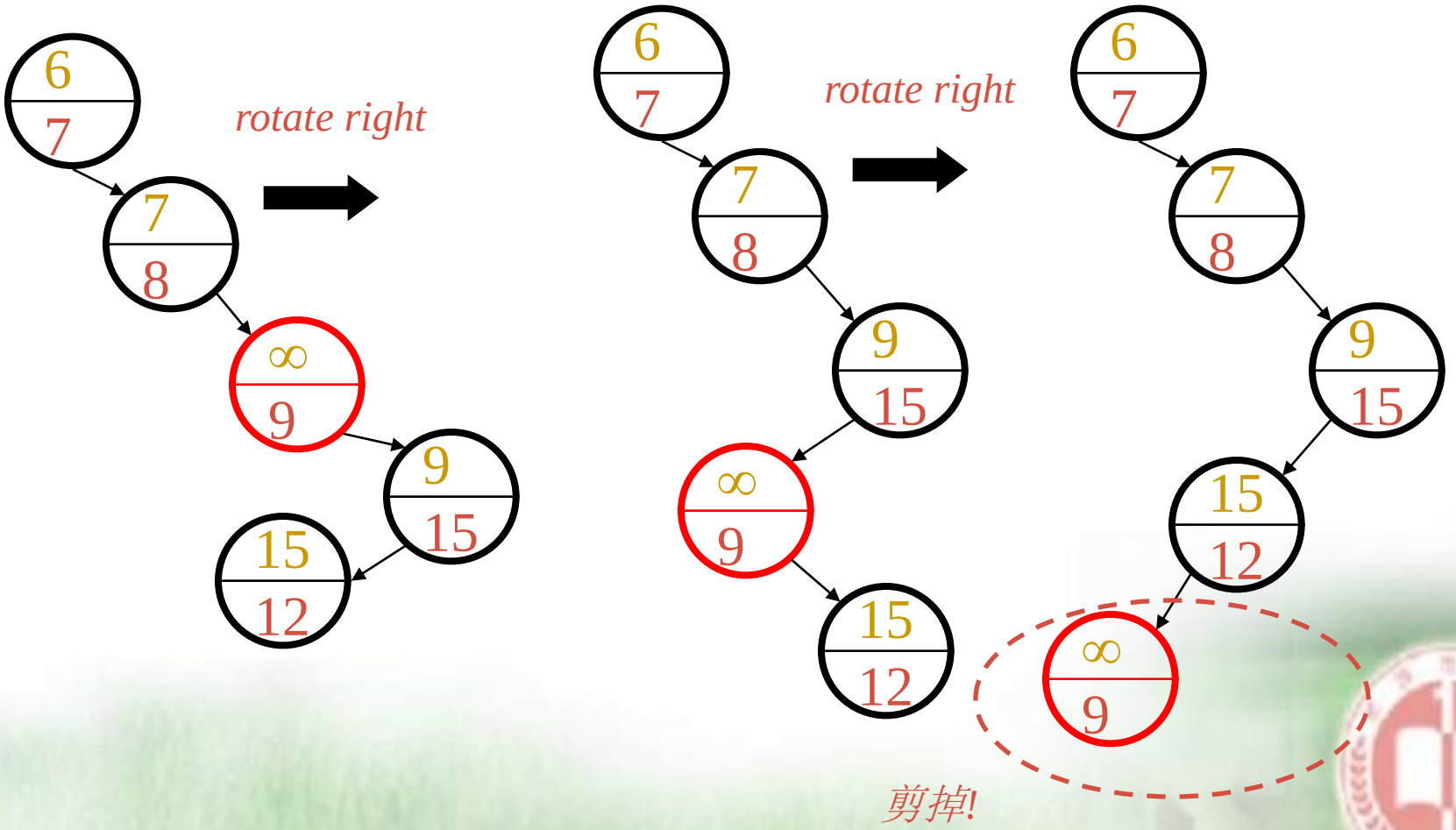


Treap Delete

delete(9)

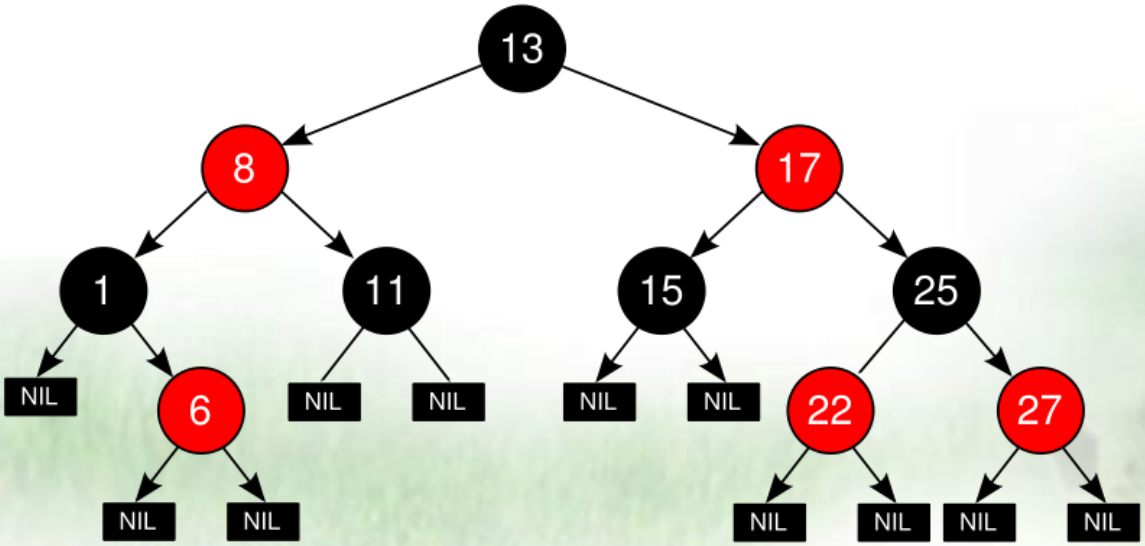


Treap Delete, cont.

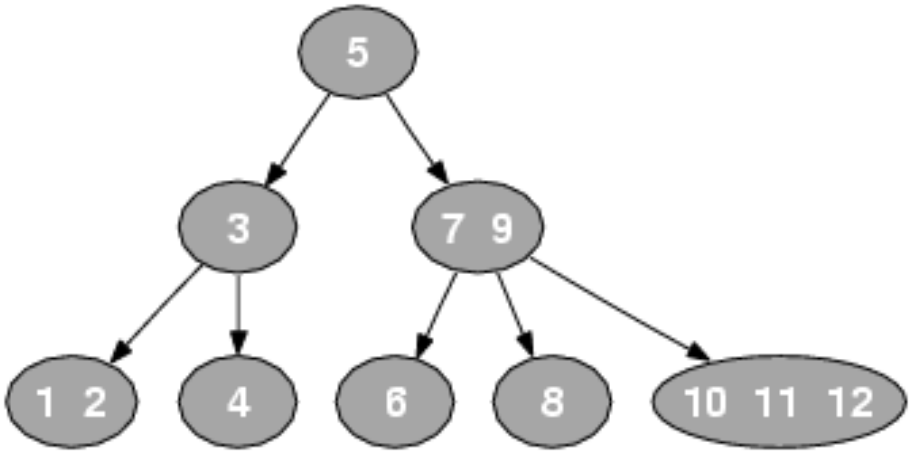


红 黑 树（1972年Rudolf Bayer发明）

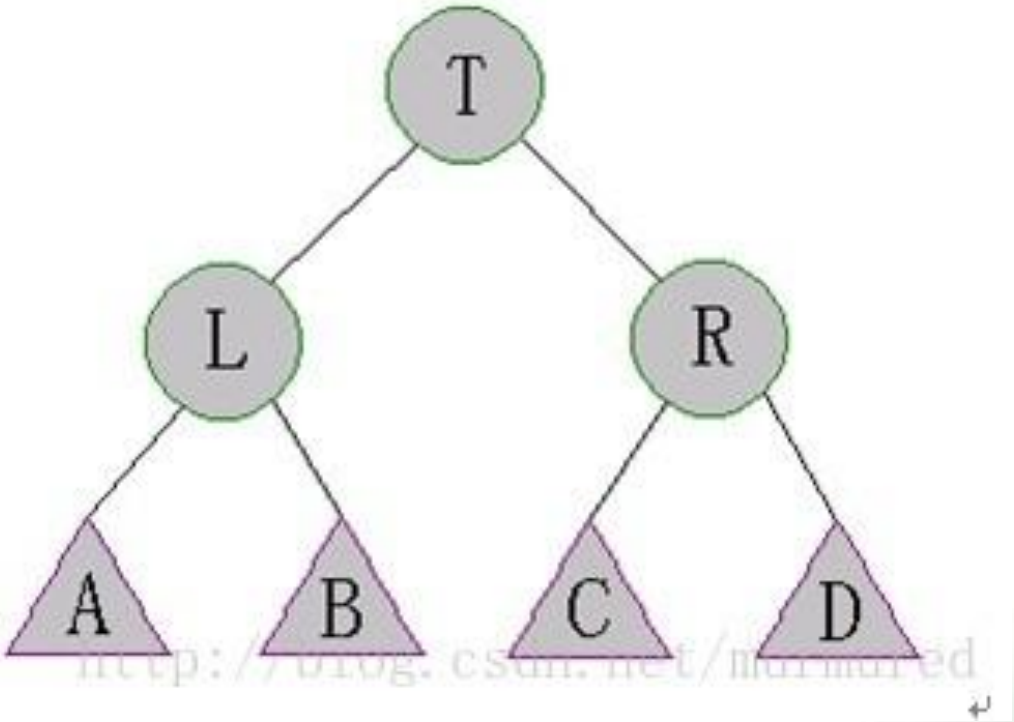
性质1. 节点是红色或黑色。
性质2. 根是黑色。
性质3. 所有叶子都是黑色（包括NIL）。
性质4. 每个红色节点的两个子节点都是黑色。
性质5. 从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。



2-3-4树

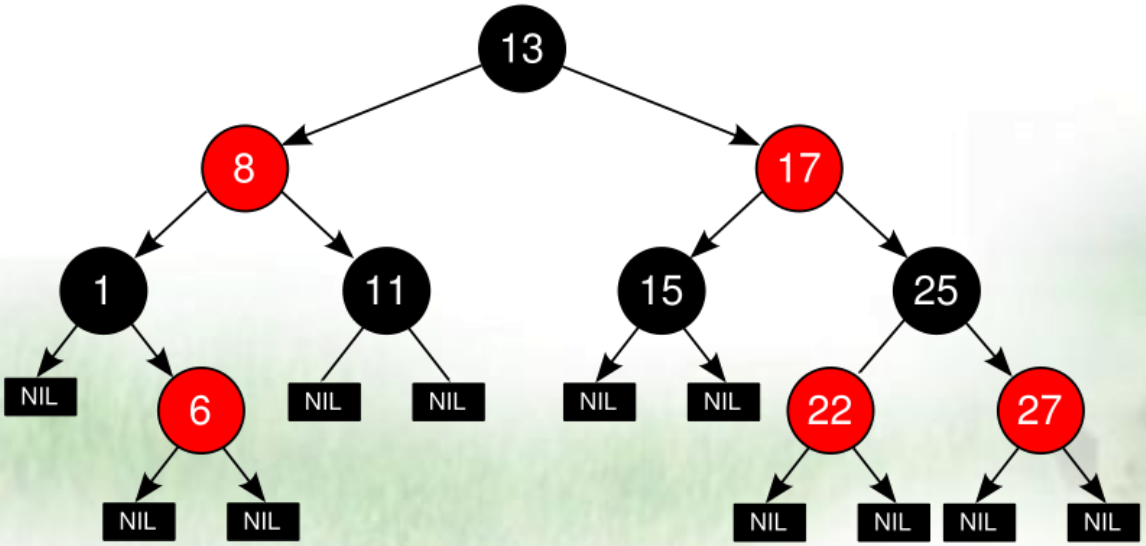


Size Balanced Tree(SBT树)

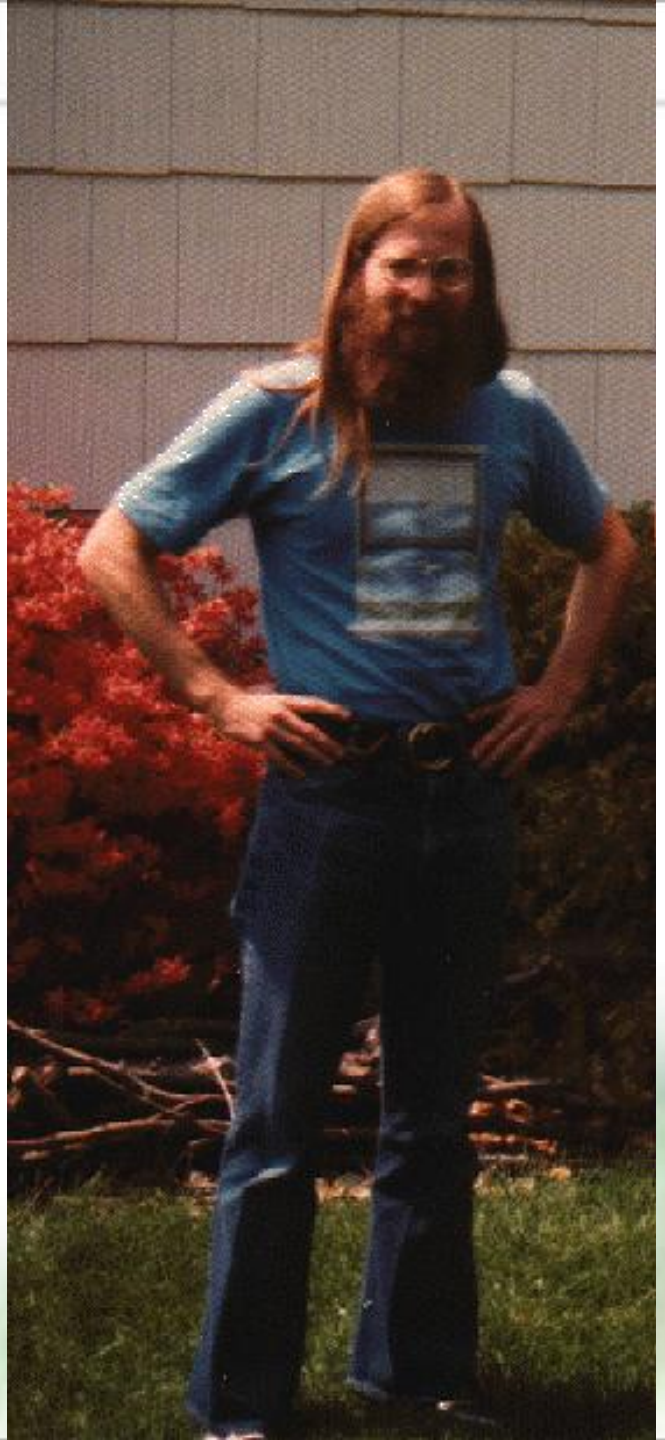


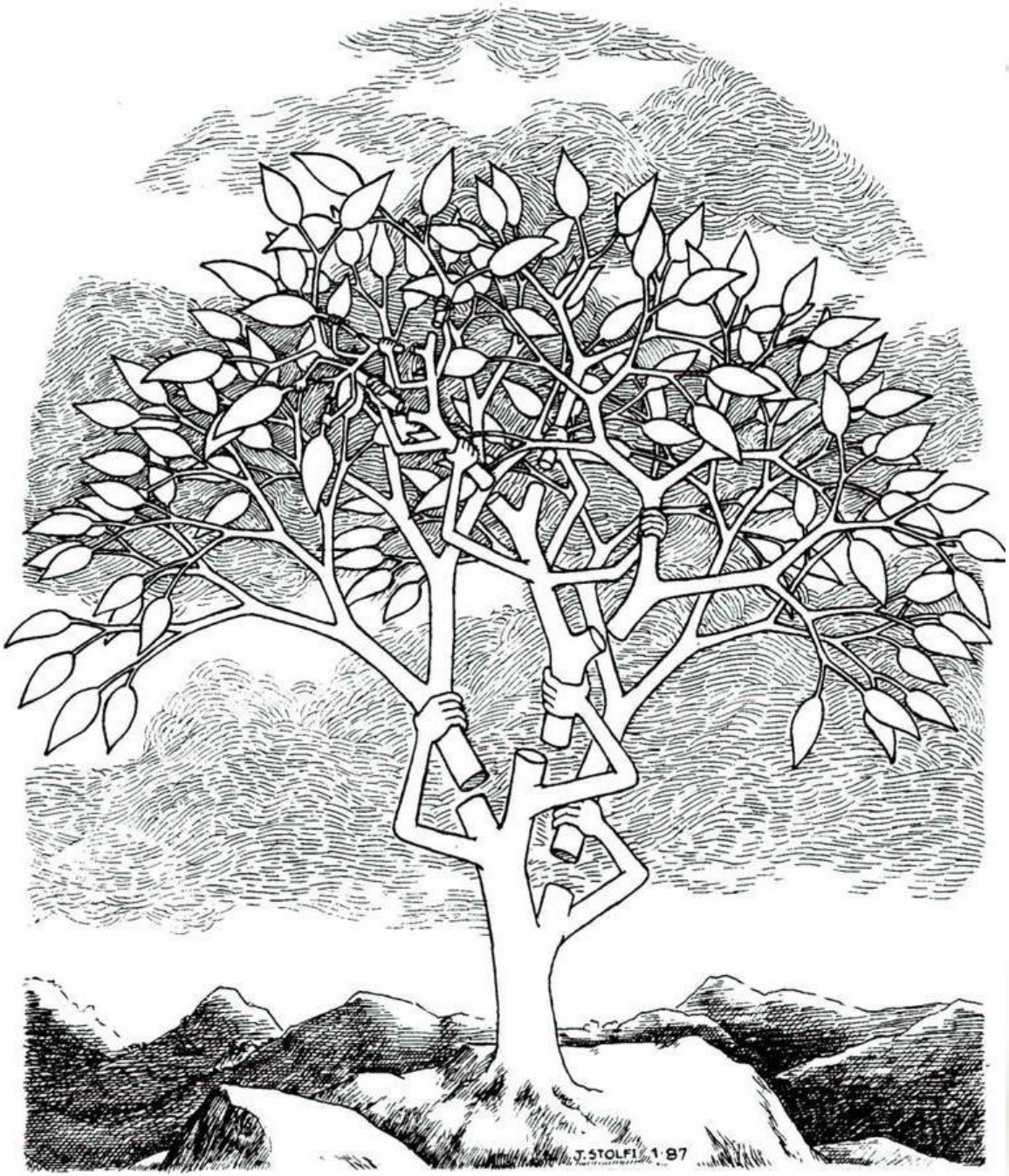
红 黑 树

性质1. 节点是红色或黑色。
性质2. 根是黑色。
性质3. 所有叶子都是黑色（包括NIL）。
性质4. 每个红色节点的两个子节点都是黑色。
性质5. 从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。



Sleator and *Tarjan* (1985)



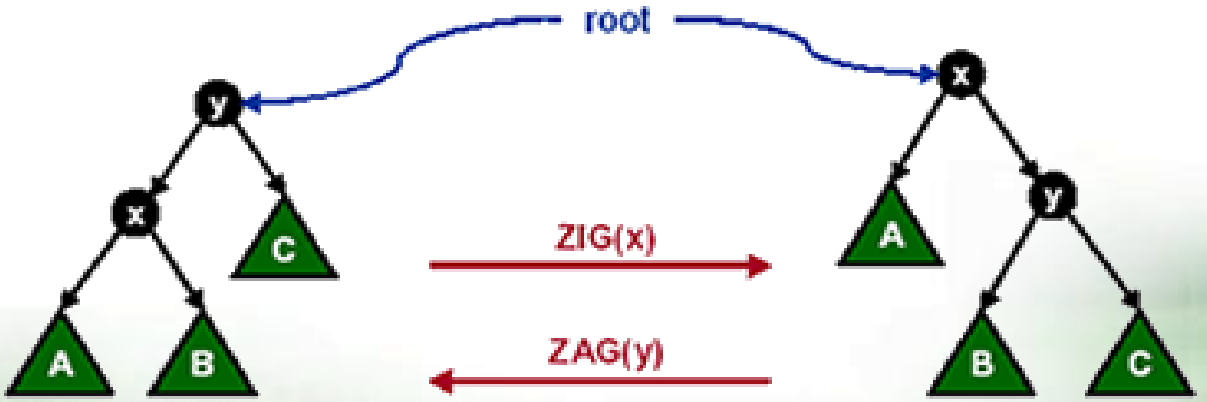

J. STOLFI 1·87

Self-Organization in Biological Systems

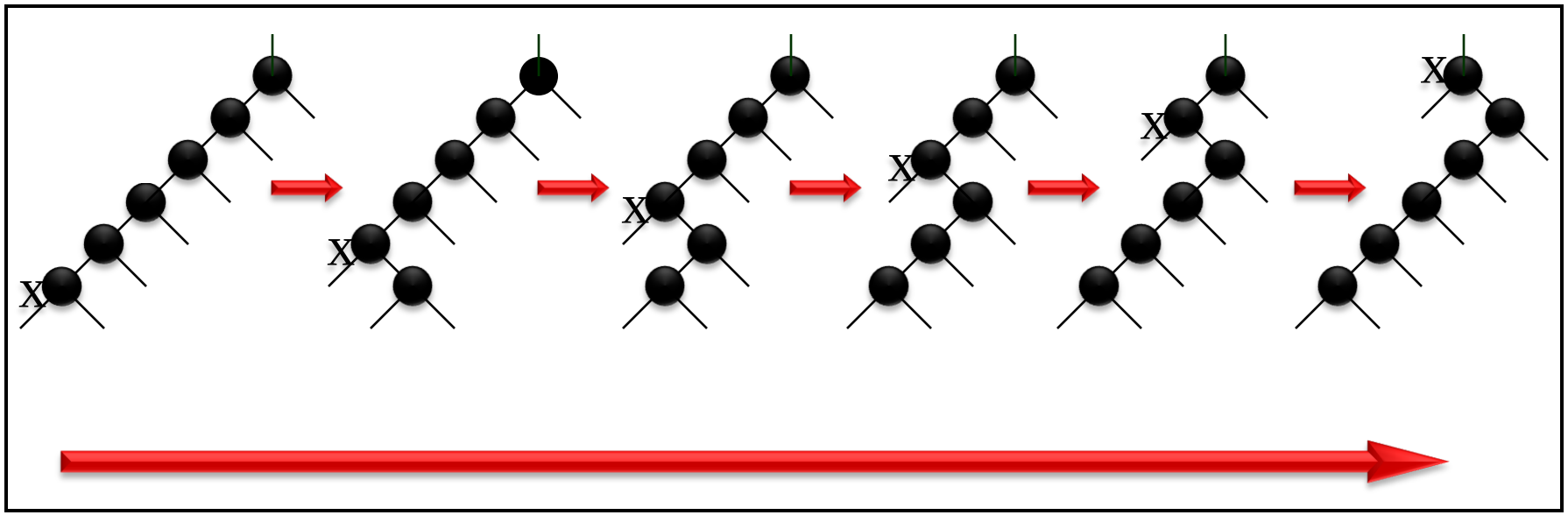
Scott Camazine Jean-Louis Deneubourg Nigel R. Franks
James Sneyd Guy Theraulaz Eric Bonabeau

PRINCETON STUDIES IN COMPLEXITY

单旋操作

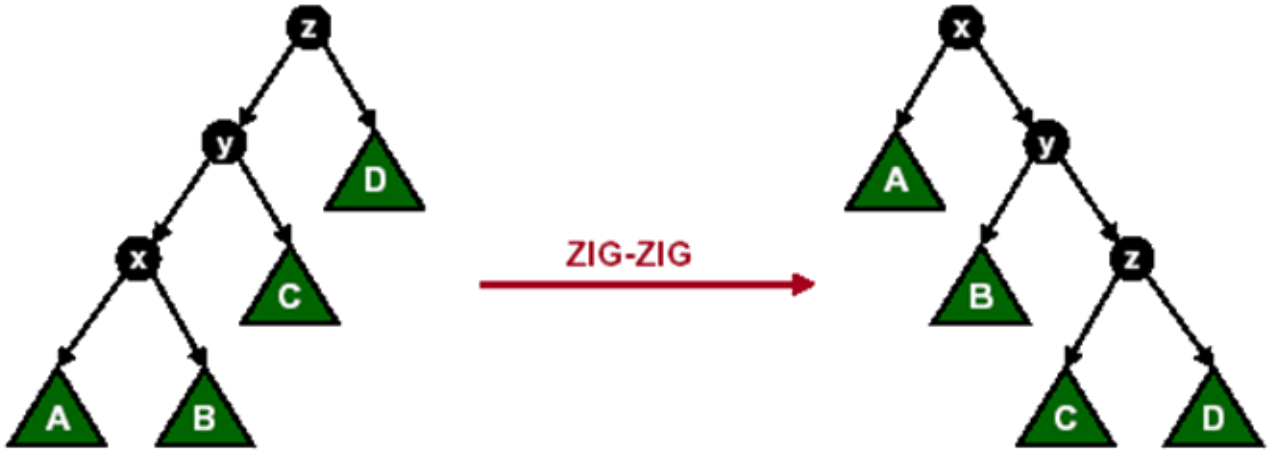


naivesplay

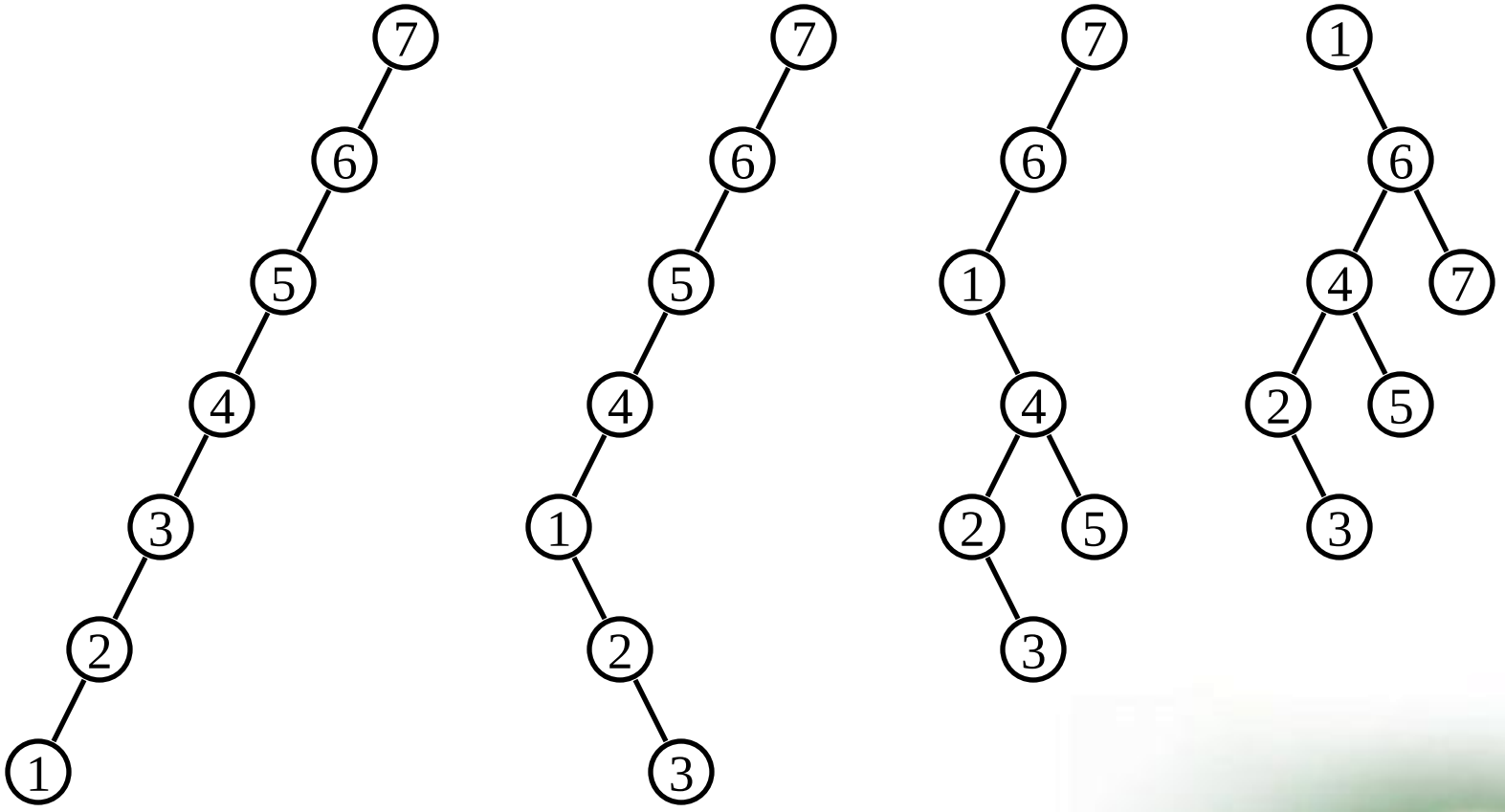



```
void naivesplay(int x)
{    for(int y;y=fa[x];rotate(x));
      root=x;
}
```

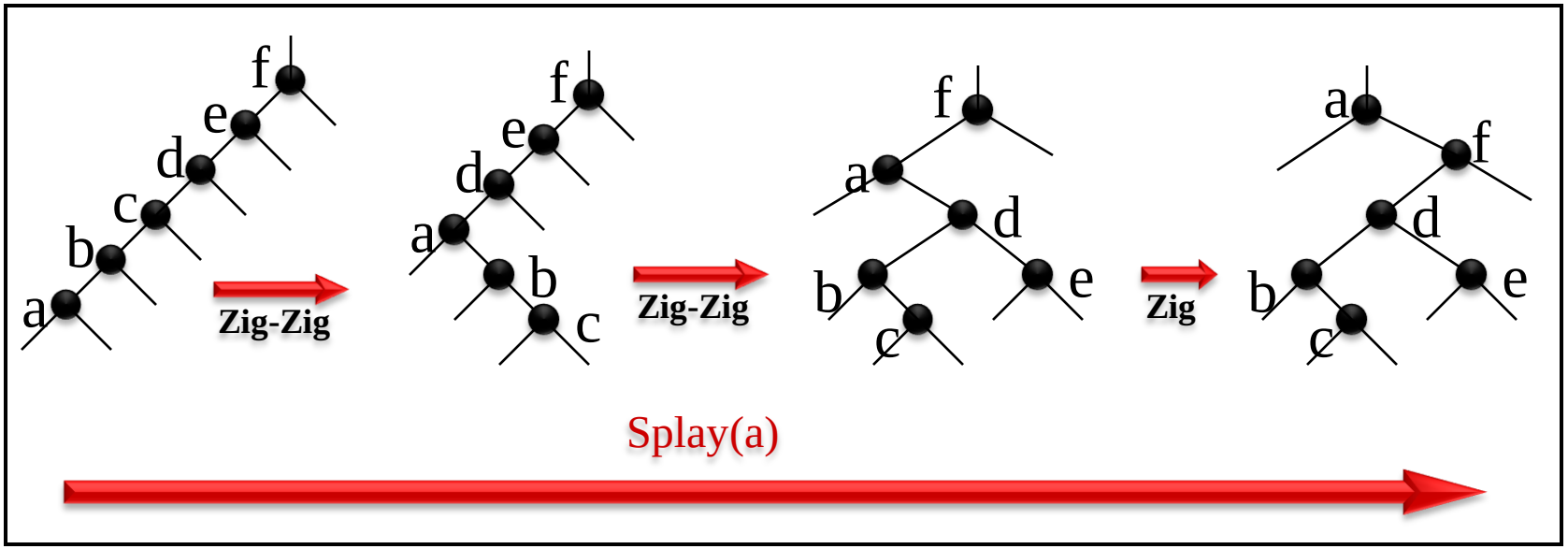
双旋操作



Insert: 1, 2, 3, 4, 5, 6, 7 Find: 1



splay

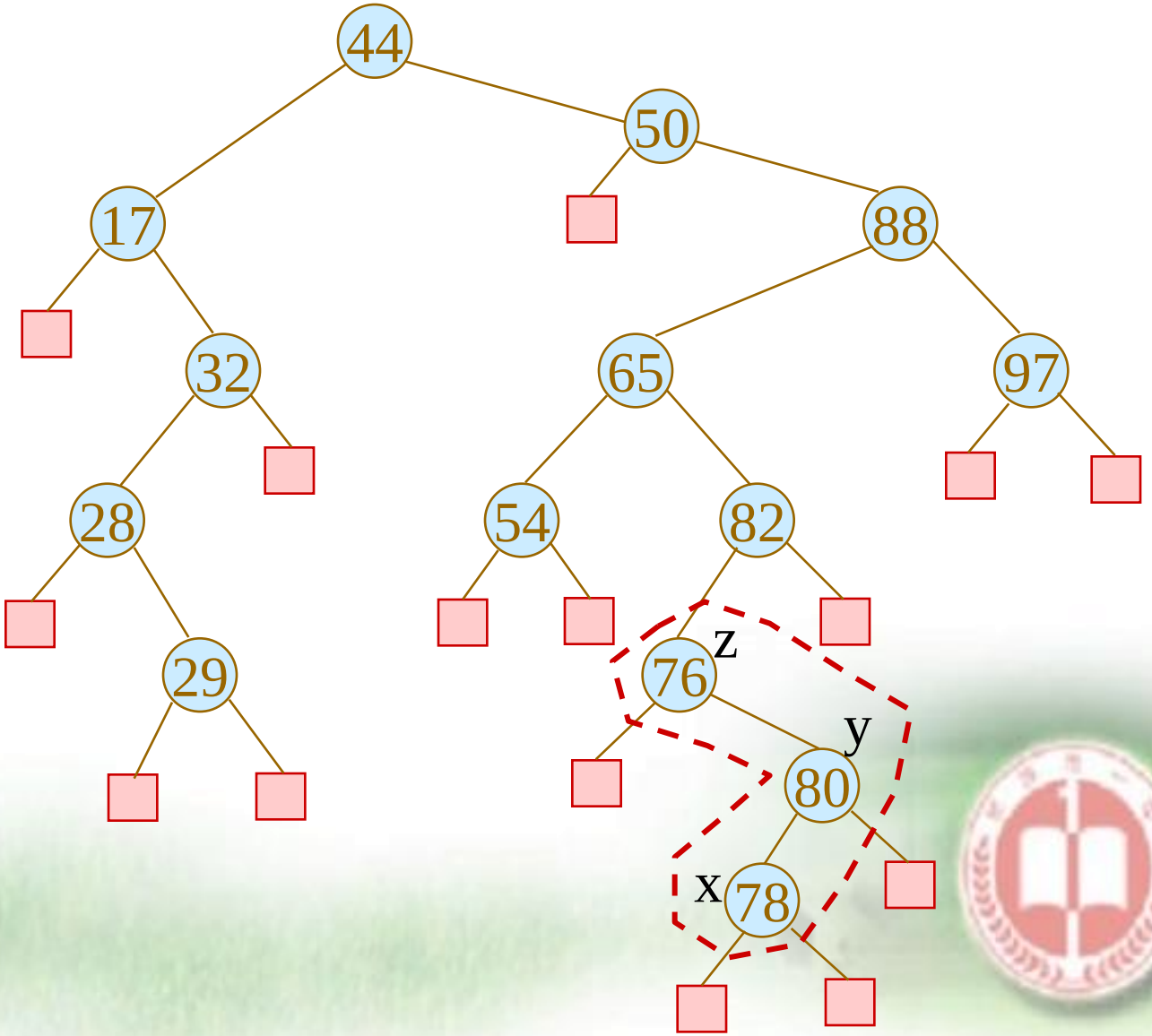



```
void splay(int x)
{  for(int y;y=fa[x];rotate(x))
       if(fa[y])
           rotate((x==child[y][1]))==(y==child[fa[y]][1]))?y:x);
   root=x;
}
```

Complete Example

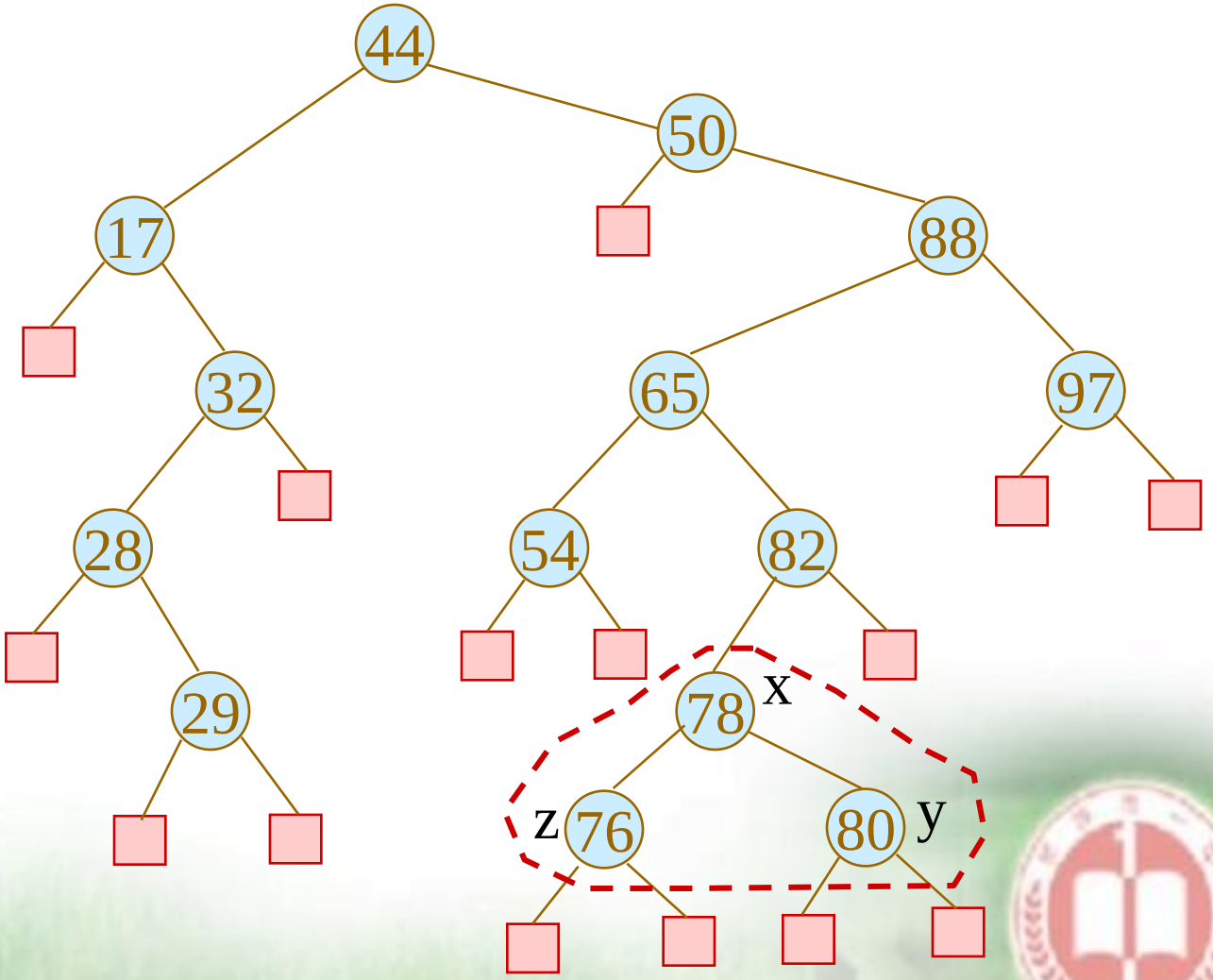
Splay(78)

zig-zag



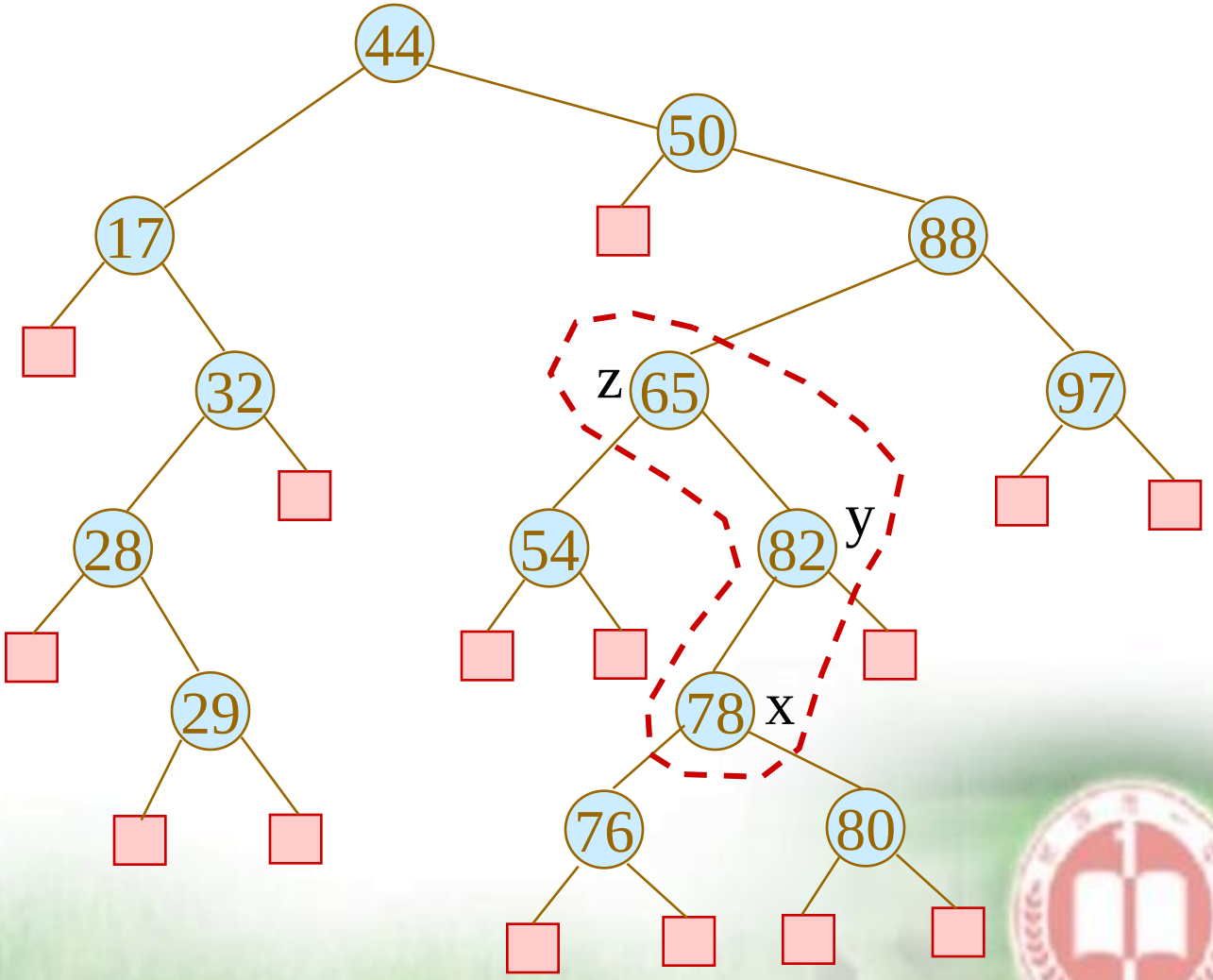
Splay(78)

zig-zag



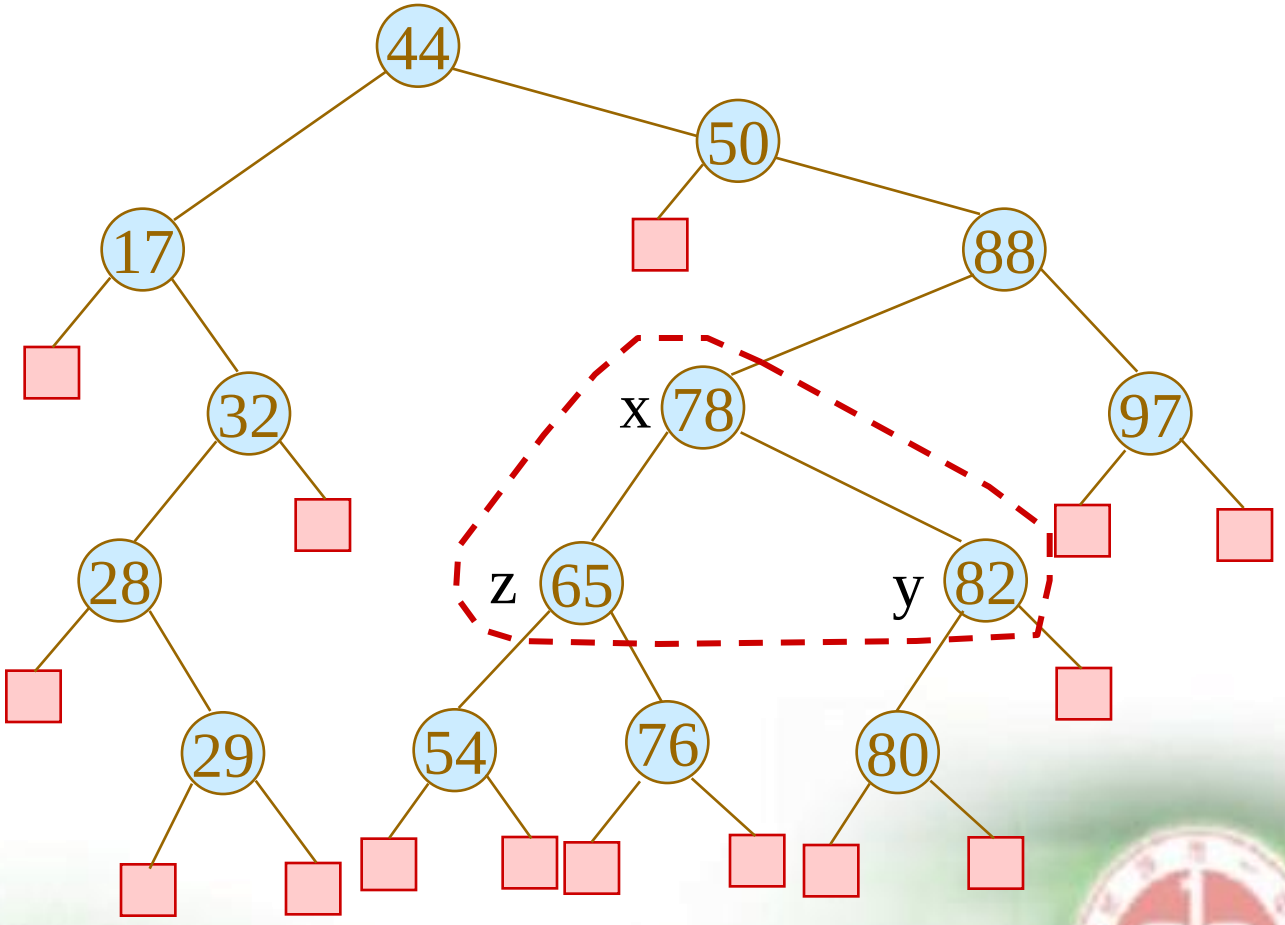
Splay(78)

zig-zag



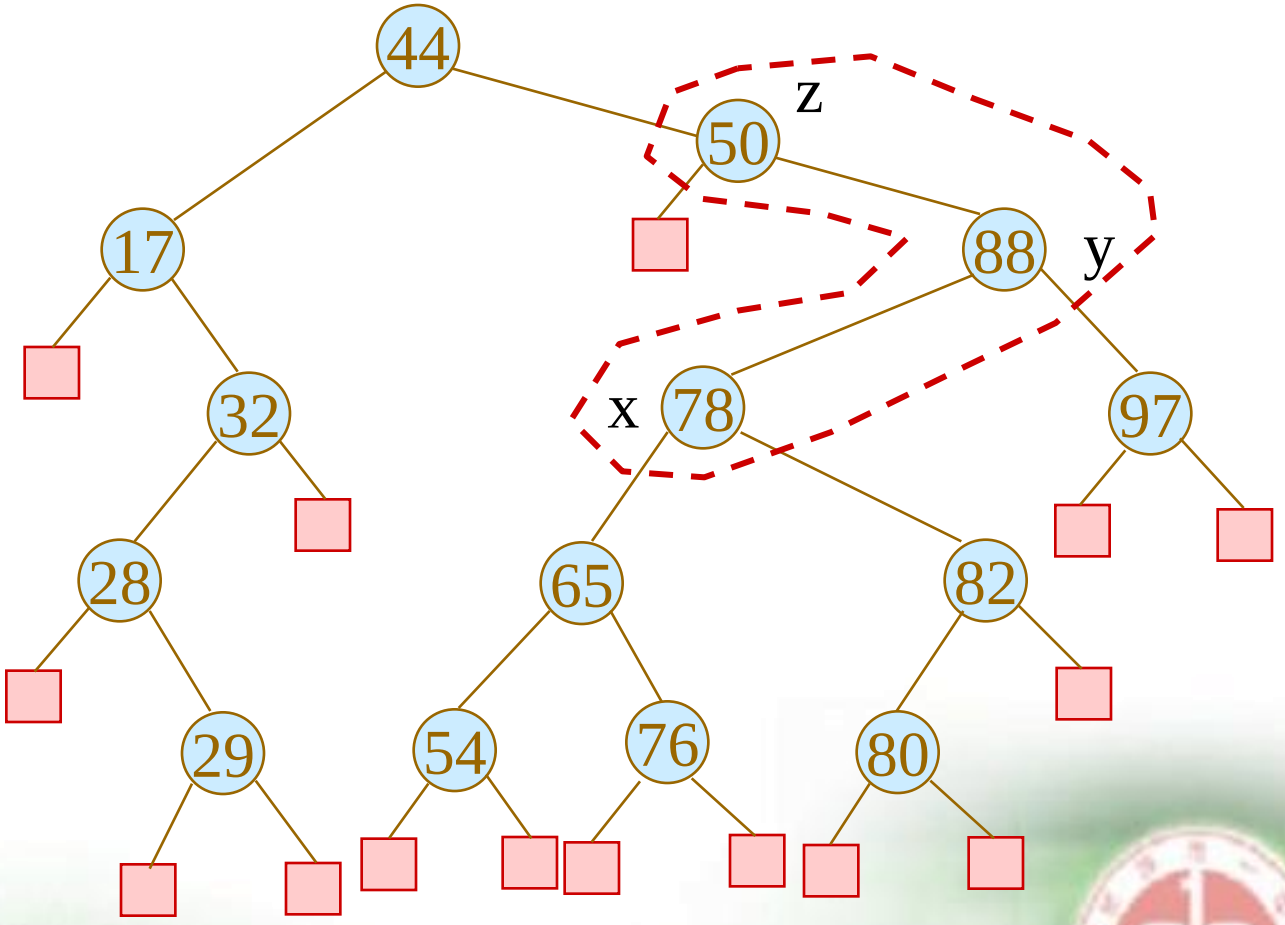
Splay(78)

zig-zag



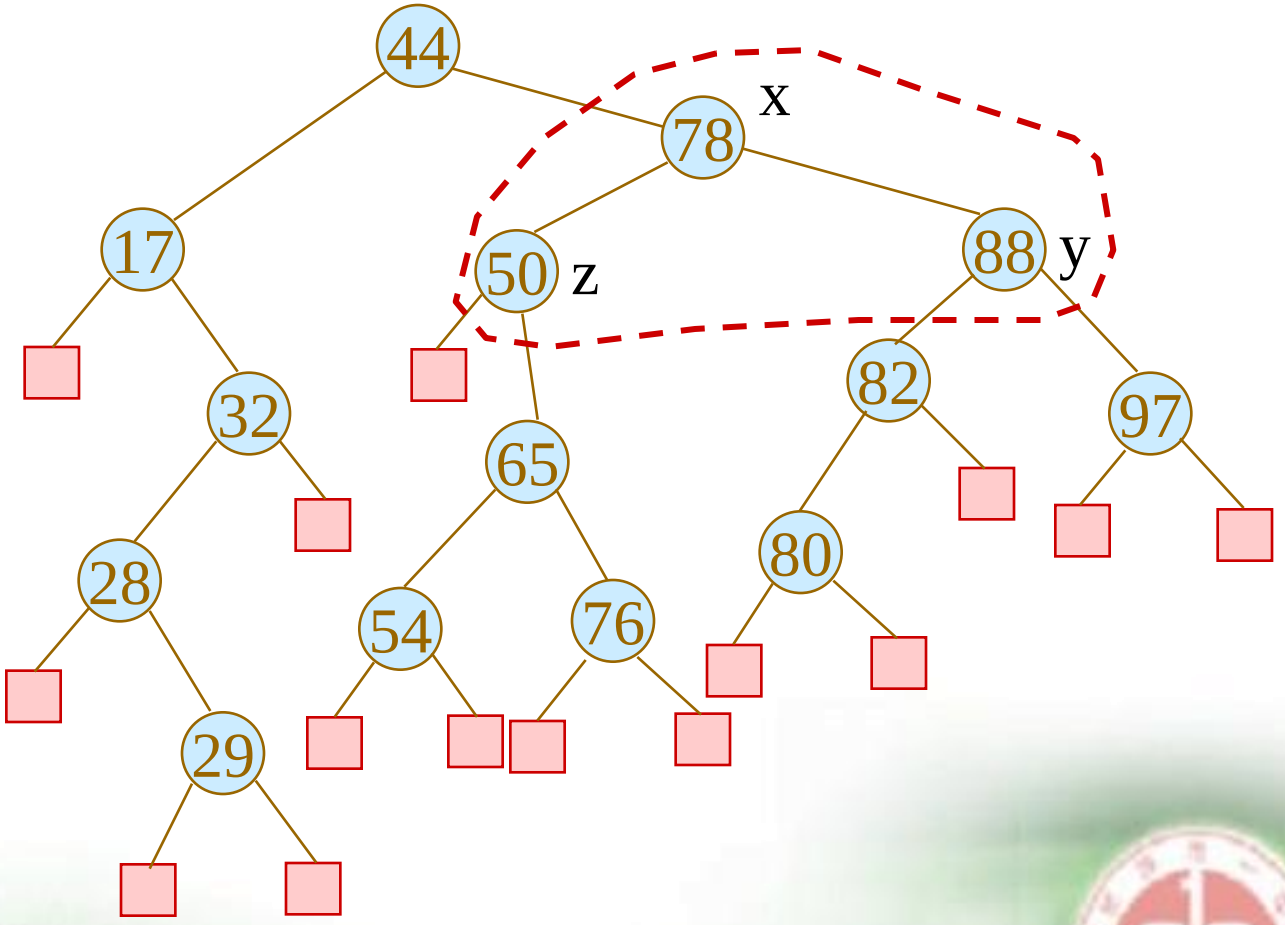
Splay(78)

zig-zag



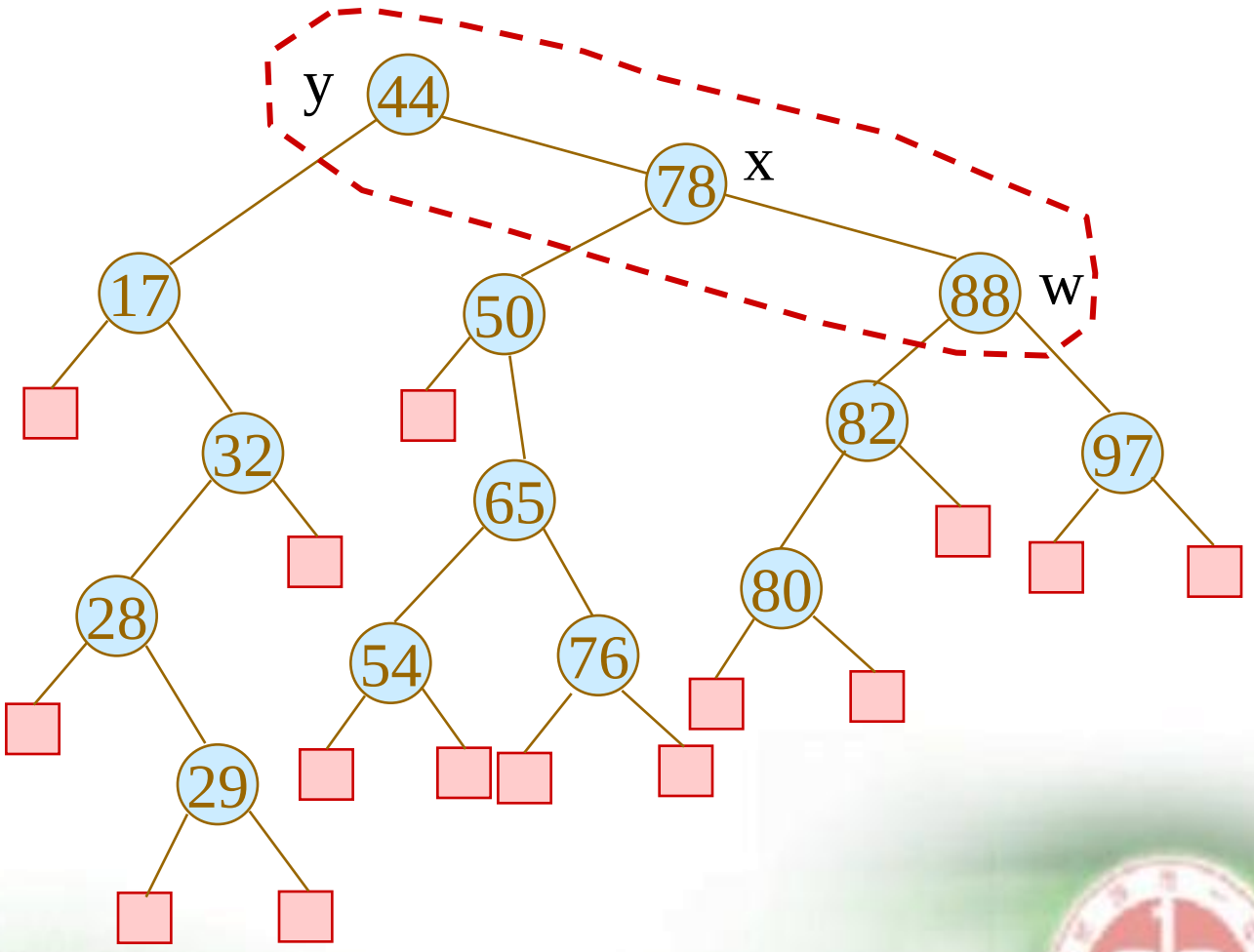
Splay(78)

zig-zag



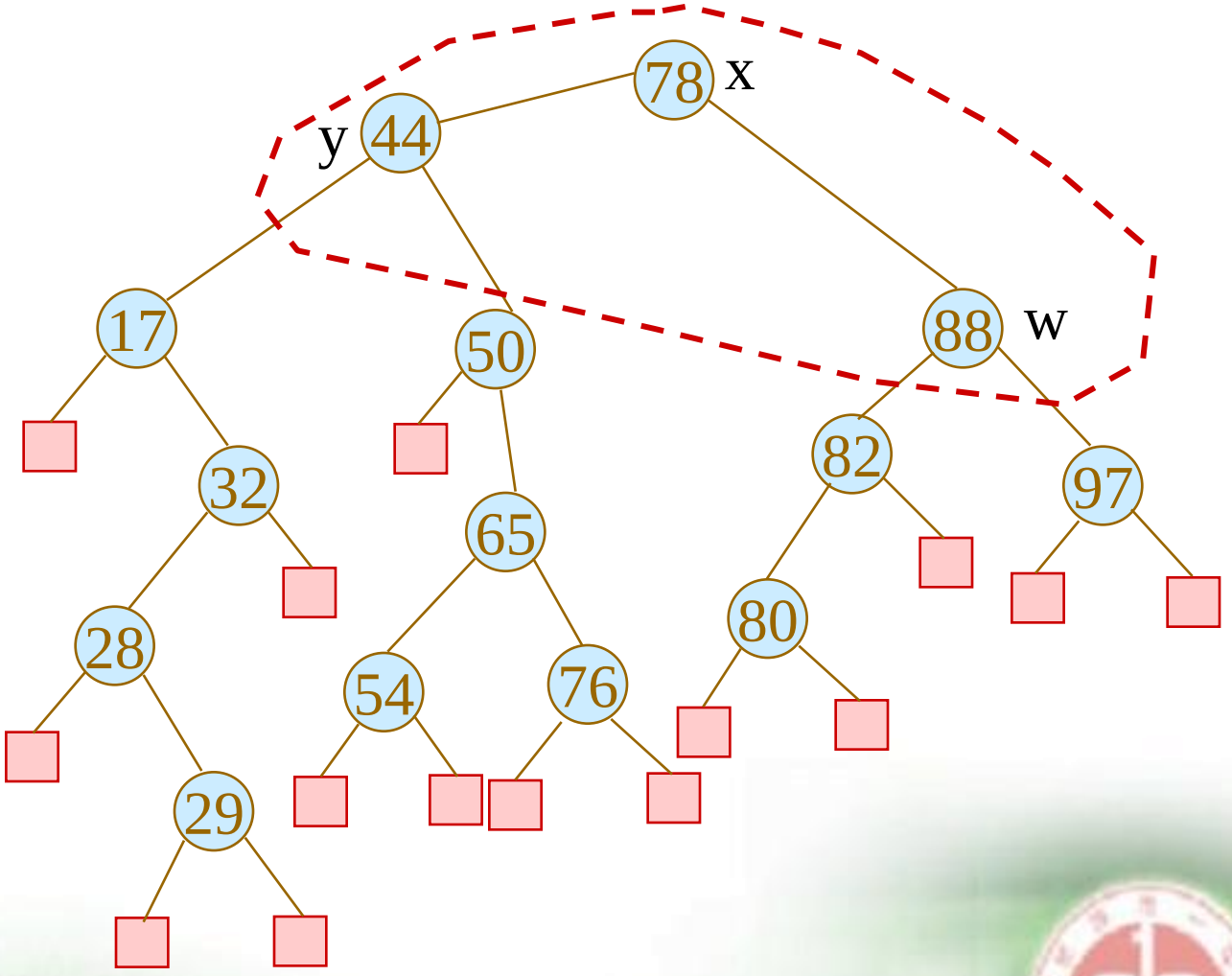
Splay(78)

zig



Splay(78)

zig



1：删除第 K 个节点

2：在第 k 个节点后面插入一个数

3：将区间[l,r]翻转

4：将区间[l,r]每个节点加上一个数

5：查询区间[l,r]的最小值

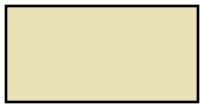
```
 void insert(int v){
1       int y,x=root;
2       if(!x){root=1,k[1]=v;return ;     }
3       while(1){
4               y=ch[x][k[x]<v];
5               if(!y){       y=++cnt;   a[y]=x;
6                       k[y]=v;     ch[x][k[x]<v]=y;
7                       break;
8                       }
9               x=y;
10              }
11      splay(y);
}
```

```
void delet(int x){
     splay(x);
    if(lc(x)){
                if(ch[x][1])
                {       int l=pre(0),r=pre(1);
                        splay(r);splay(l);
                        if(ch[r][0]==x)ch[r][0]=0,fa[x]=0;
                        else ch[l][1]=r,fa[r]=l;
                }
                else root=ch[x][0],fa[ch[x][0]]=0,ch[x][0]=0;
        }
        else
if(ch[x][1])root=ch[x][1],fa[ch[x][1]]=0,ch[x][1]=0;
        else root=0;
}
```

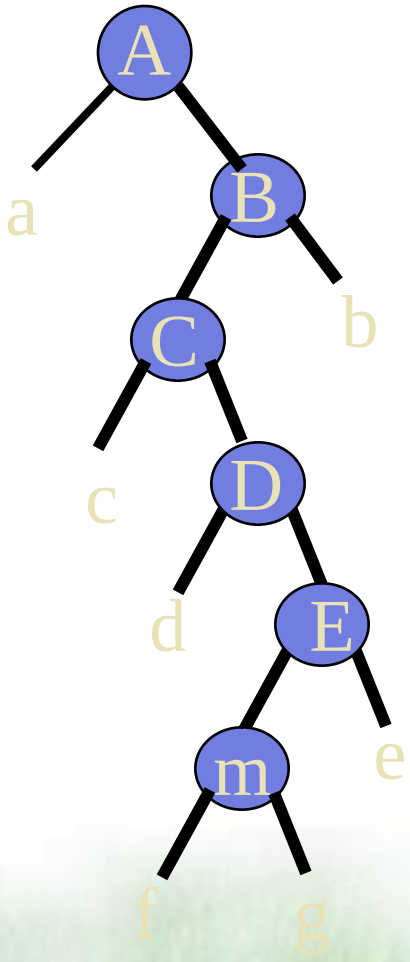
```
int pre(int x){
        int tmp=ch[root][x];
        while(ch[tmp][!x])tmp=ch[tmp][!x];
        return tmp;
}
```

Split A Binary Search Tree

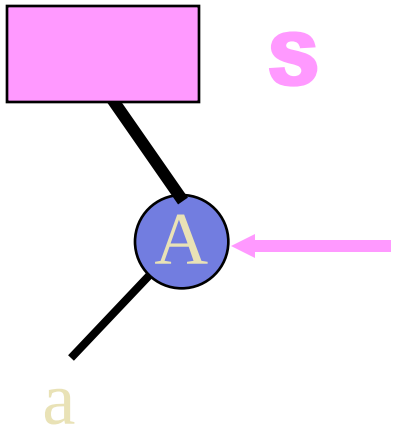
S

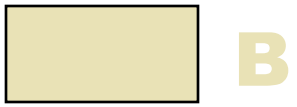


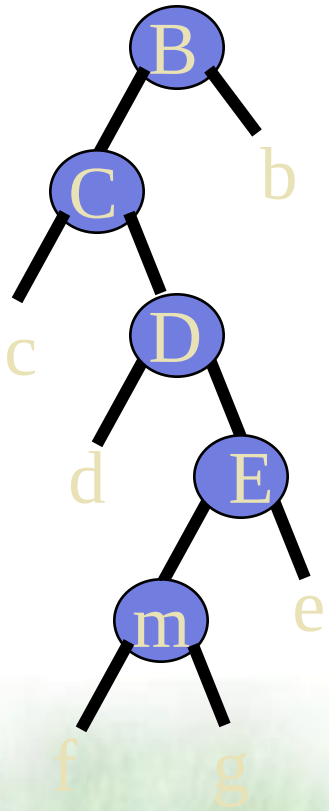
B



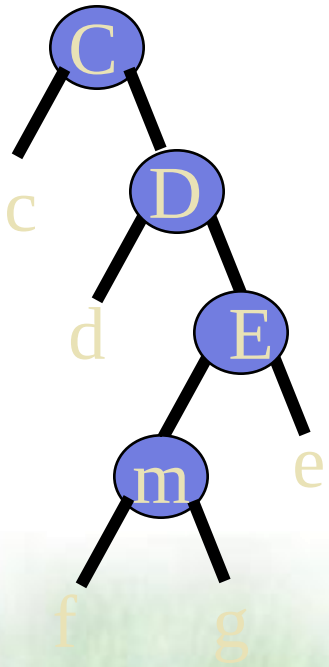
Split A Binary Search Tree



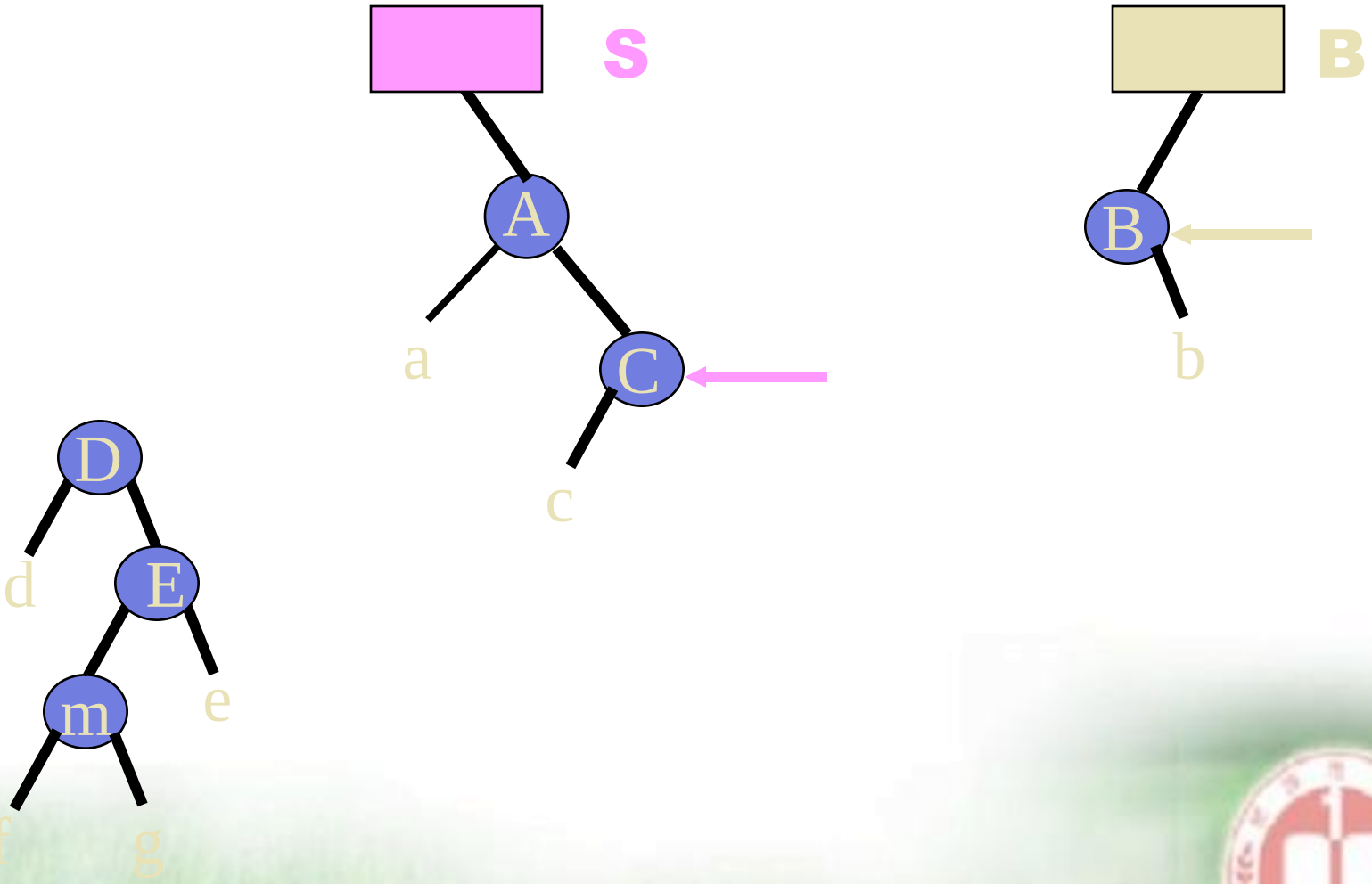




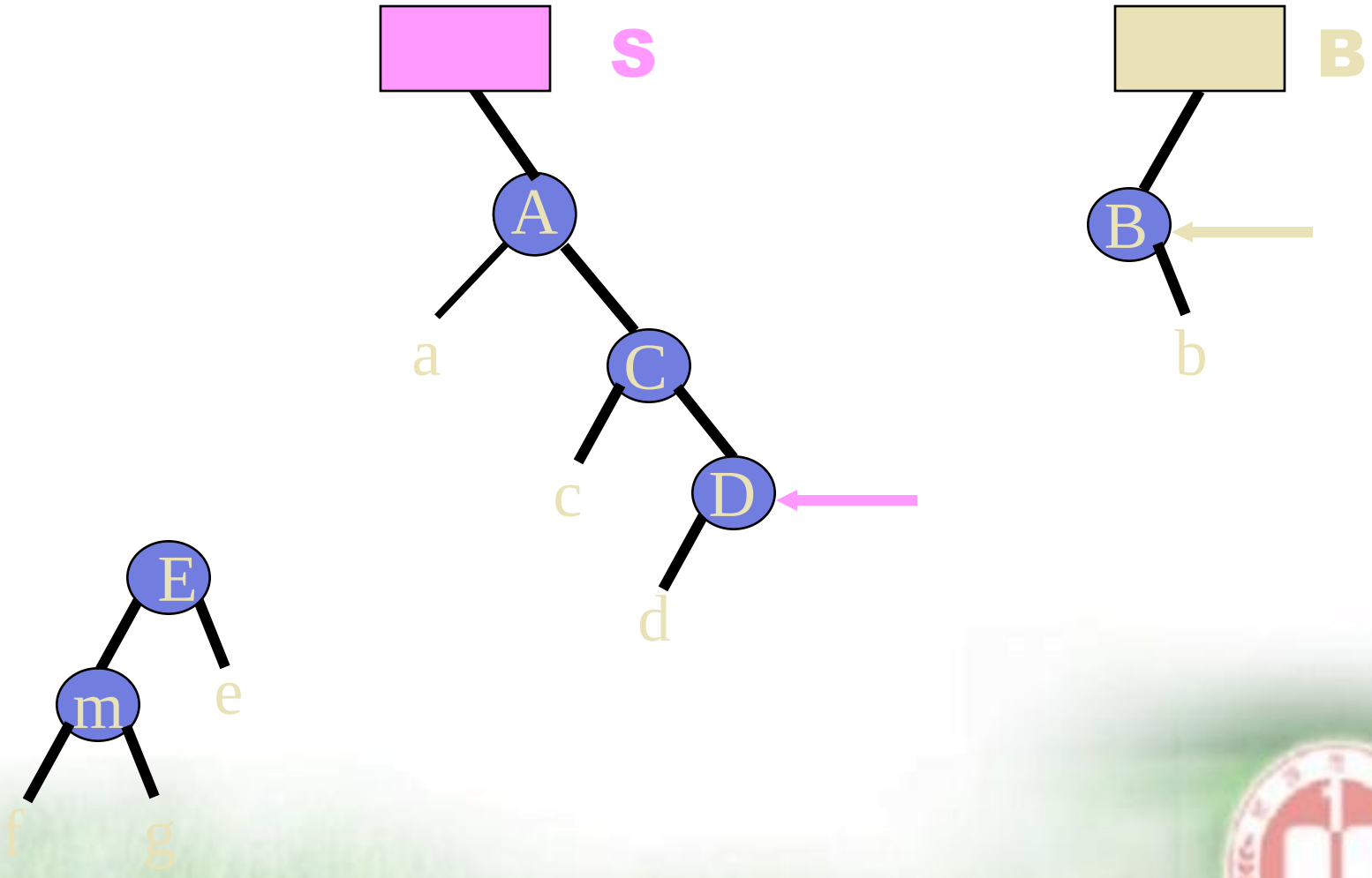
Split A Binary Search Tree



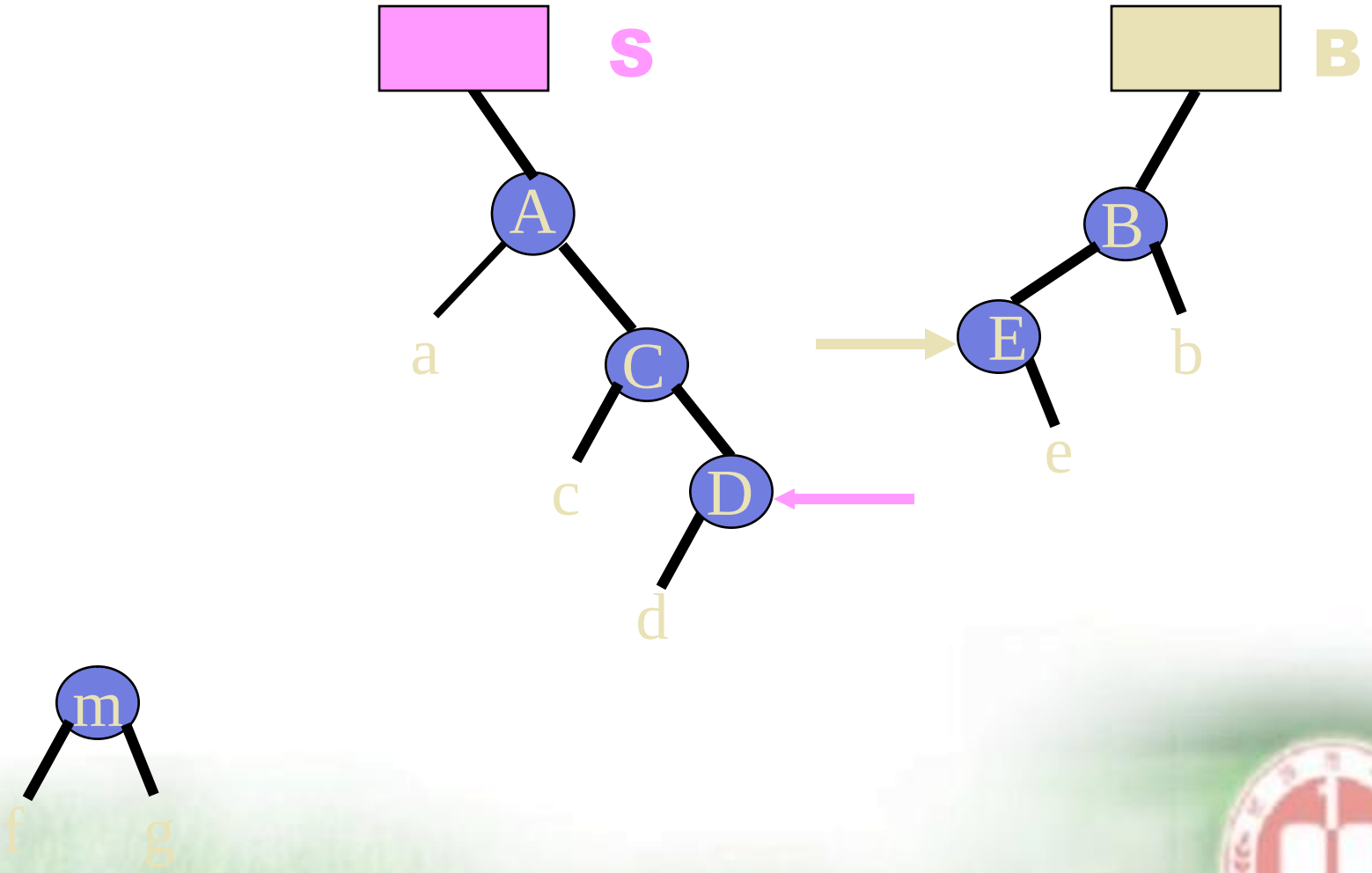
Split A Binary Search Tree



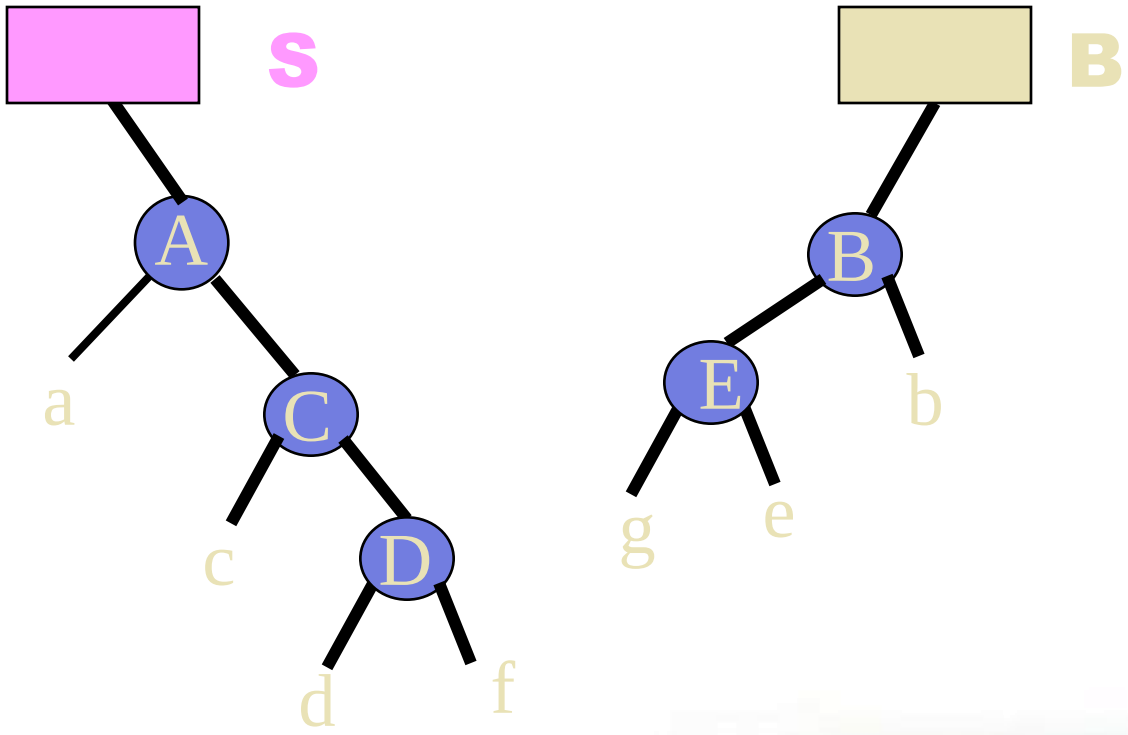
Split A Binary Search Tree



Split A Binary Search Tree



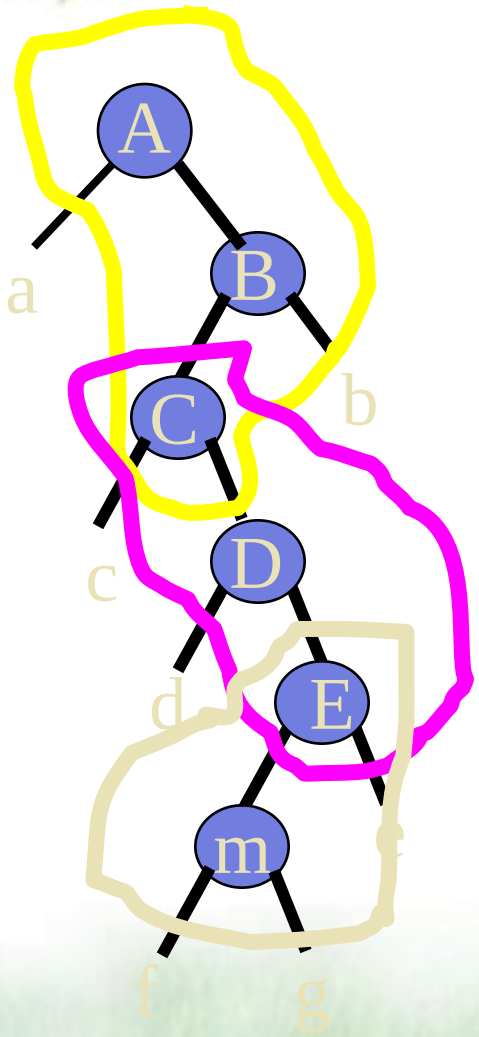
Split A Binary Search Tree



m

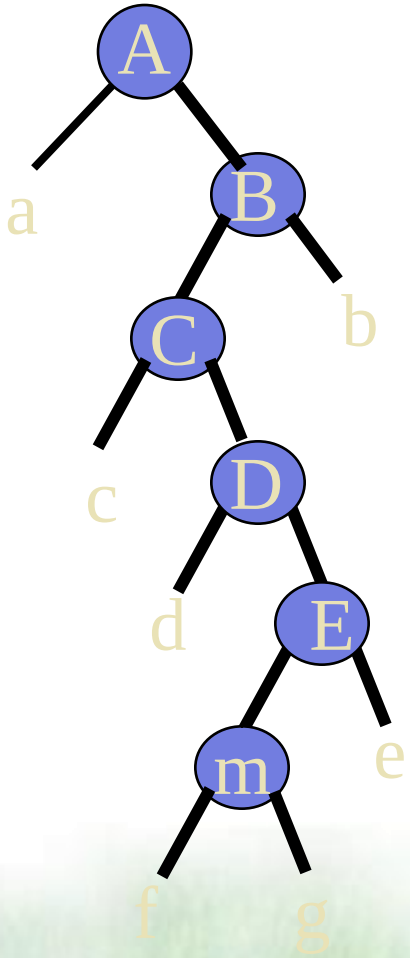
- Let m be the splay node.

Two-Level Moves



- Let m be the splay node.
  - RL move from A to C.
  - RR move from C to E.
  - L move from E to m.

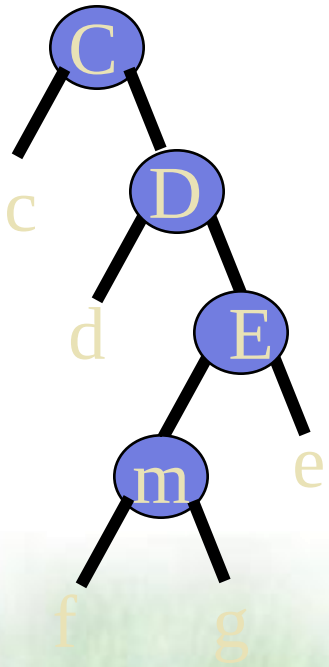
RL Move



S

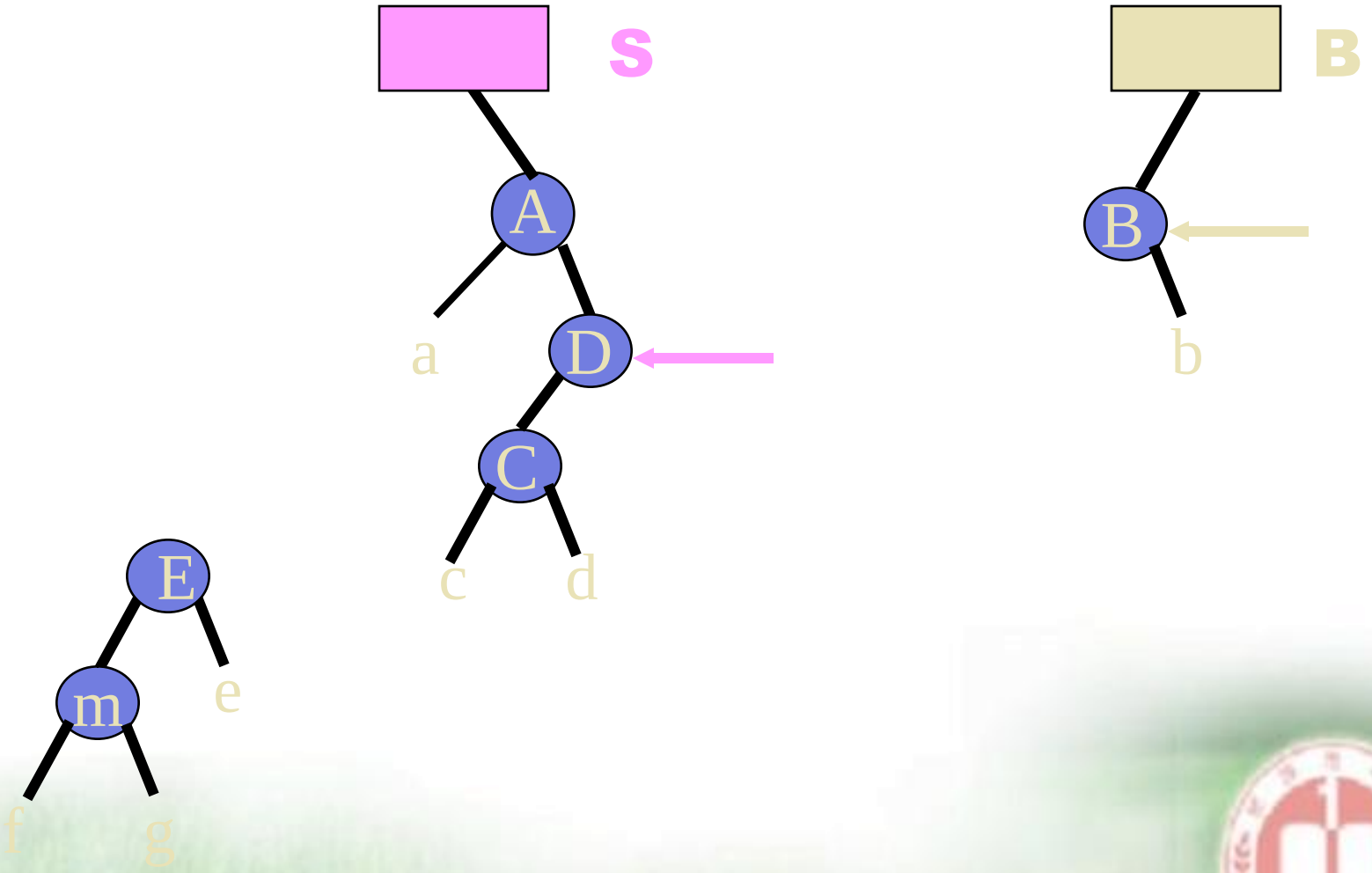
B

RL Move

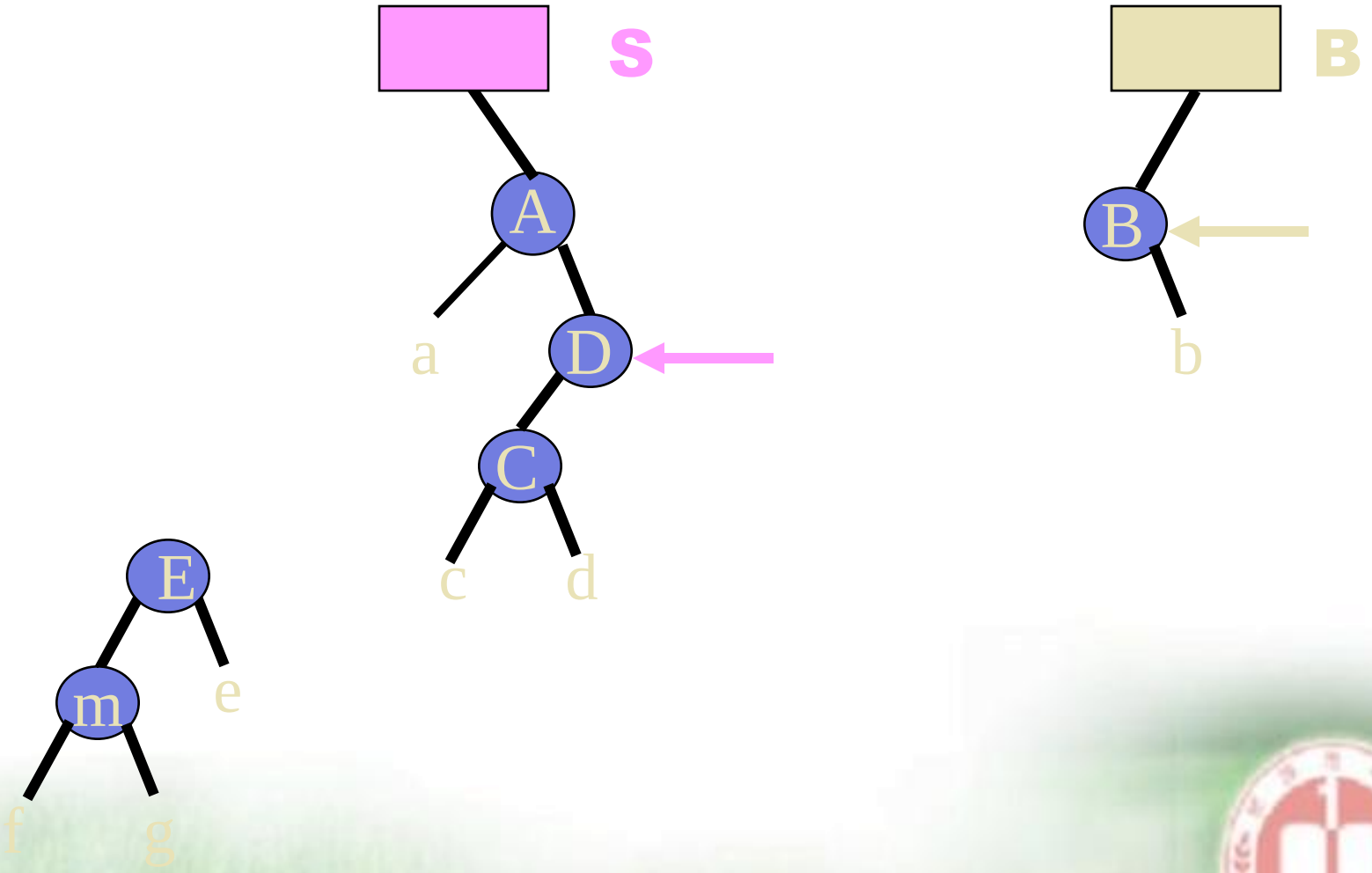


RR Move

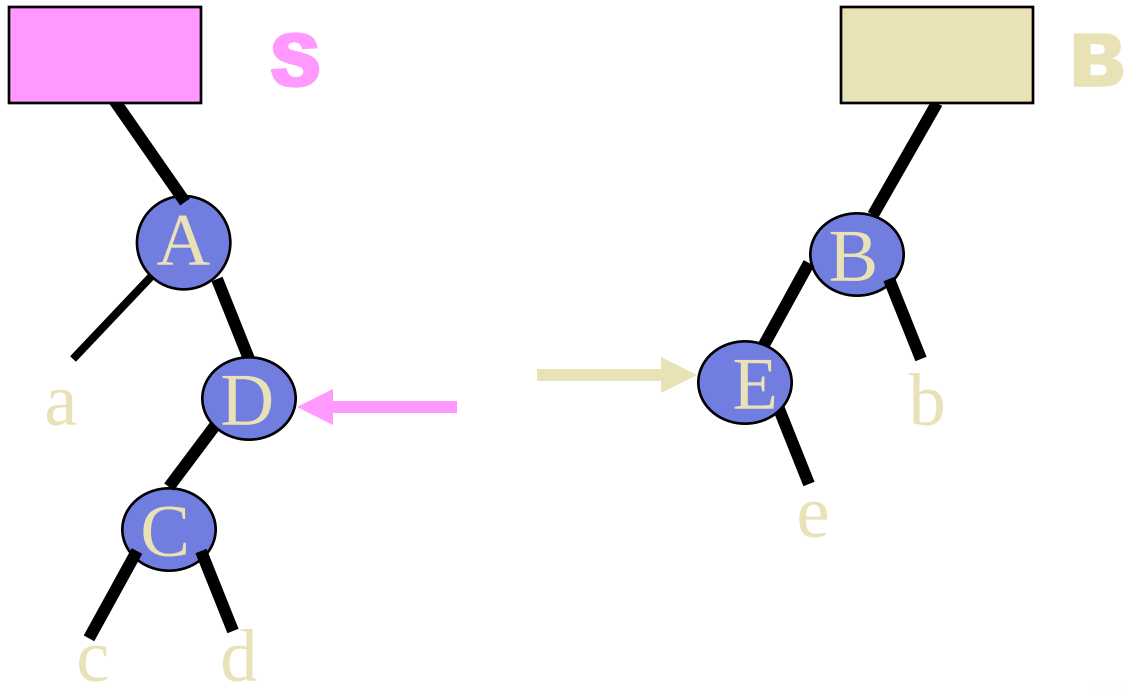
RR Move

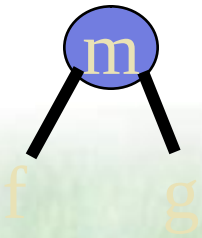


L Move

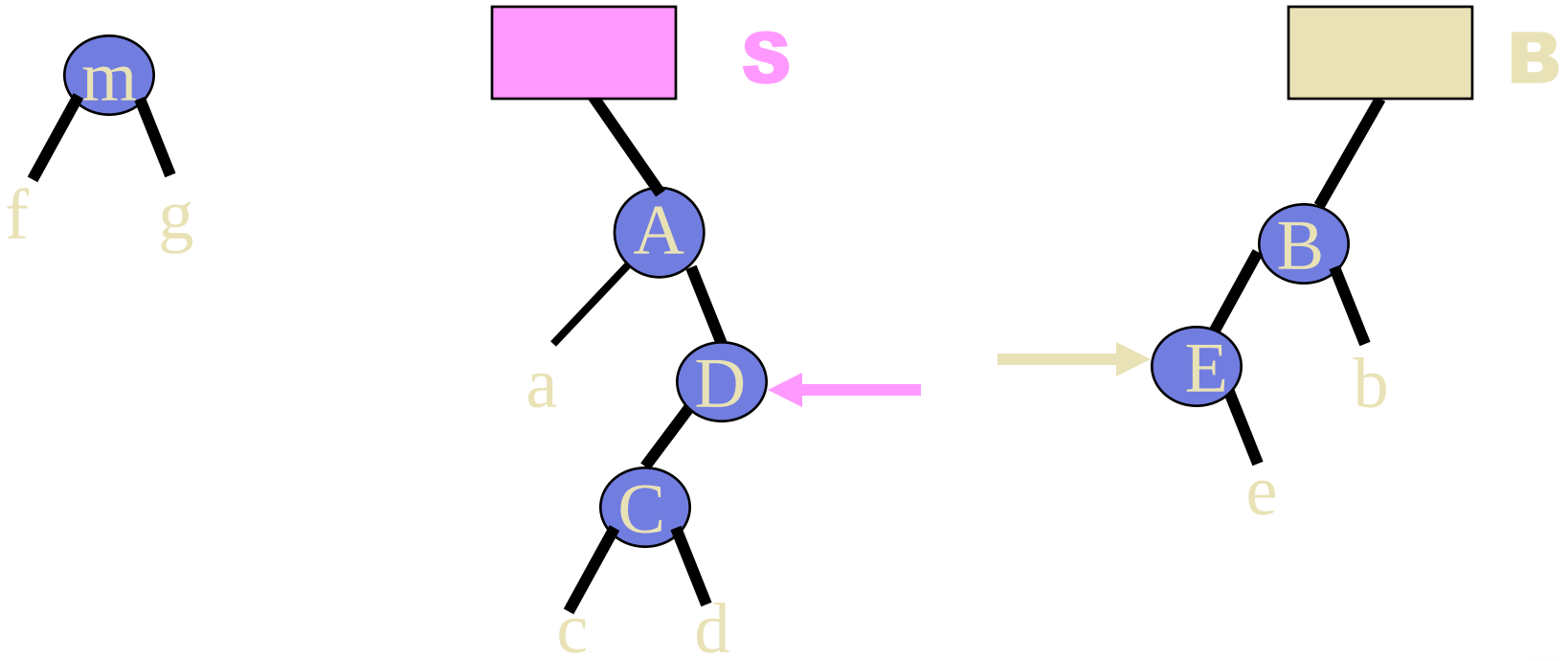


L Move

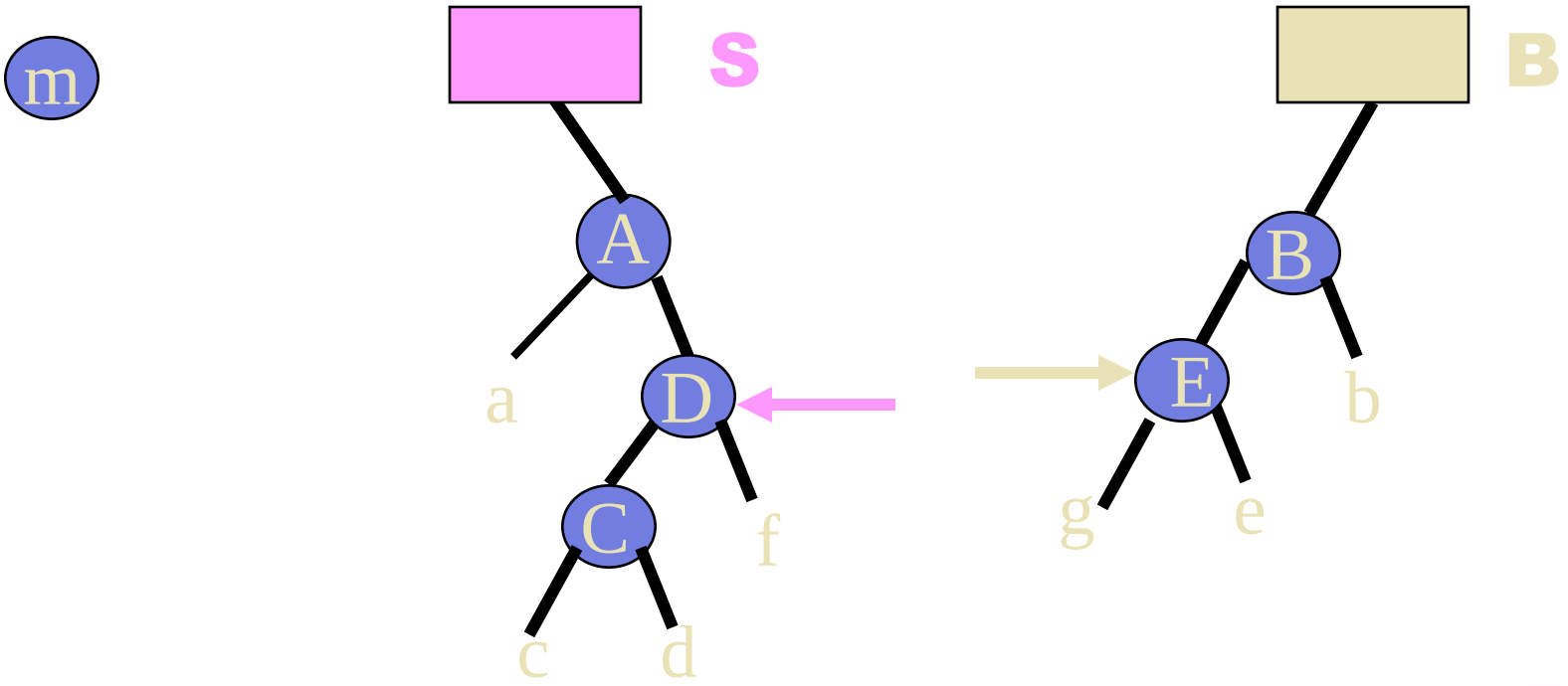




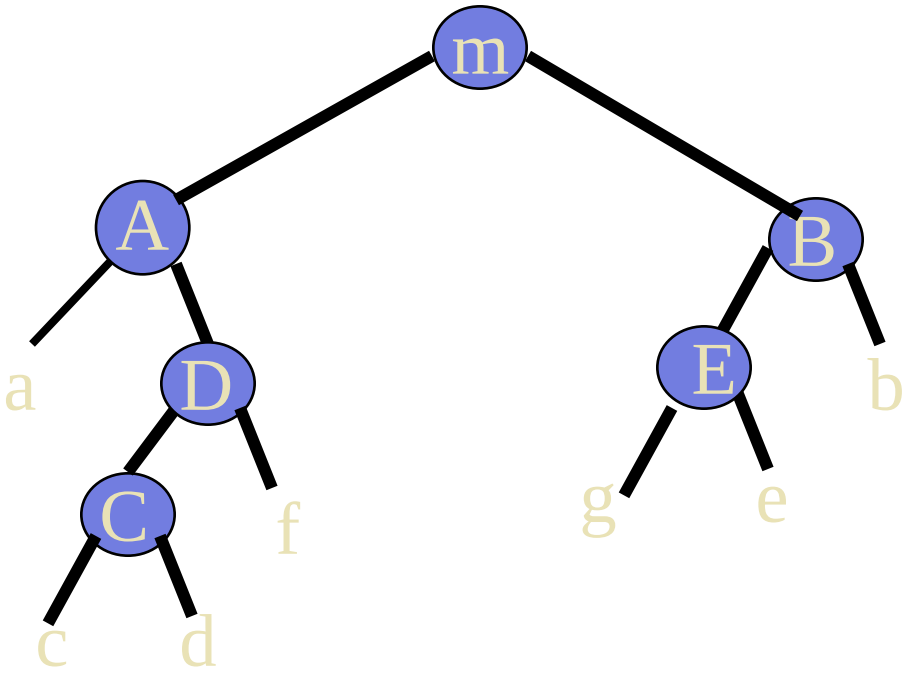
合并



合并



合并



练习题

- BZOJ3224、 BZOJ3223、（模板题）
- BZOJ1507、BZOJ1503
- BZOJ1014、 BZOJ3506、 BZOJ1251、BZOJ1500、