1. 伸展树

Splay Tree，中文叫伸展树，或者分裂树。伸展树(Splay Tree)是一种二叉排序树，它能在 O(log n)内完成插入、查找和删除操作。它的优势在于不需要记录用于平衡树的冗余信息。伸展树由 Daniel Sleator 和 Robert Tarjan 创造。

2. 为什么会有伸展树？

假设想要对一个二叉查找树执行一系列的查找操作，为了使整个查找时间更小，被查频率高的那些节点就应当经常处于靠近树根的位置。于是想到设计一个简单方法，在每次查找节点之后对树进行重构，把被查找的节点搬移到树根，这种自调整形式的二叉查找树就是 splay tree（伸展树），它会沿着从某个被访问节点到树根之间的路径，通过一系列的旋转把这个被访问节点搬移到树根去。

3. 怎样旋转搬移至树根？

伸展树通过一系列的旋转把当前被访问节点搬移到树根，以便下次再次访问该节点时速度极快（直接访问根节点就被命中）。为了将当前被访问节点搬移到树根，我们需要沿着查找路径做自底向上的旋转，直至该节点成为树根为止（伸展树的旋转是成对进行的，伸展操作不单是把当前被访问节点搬移到树根，而且还把查找路径上的每个节点的深度都大致减掉一半。）。（假设当前被访问节点为 X，X 的父亲节点为 P（如果 X 的父亲节点存在），X 的祖父节点为 G（如果 X 的祖父节点存在））

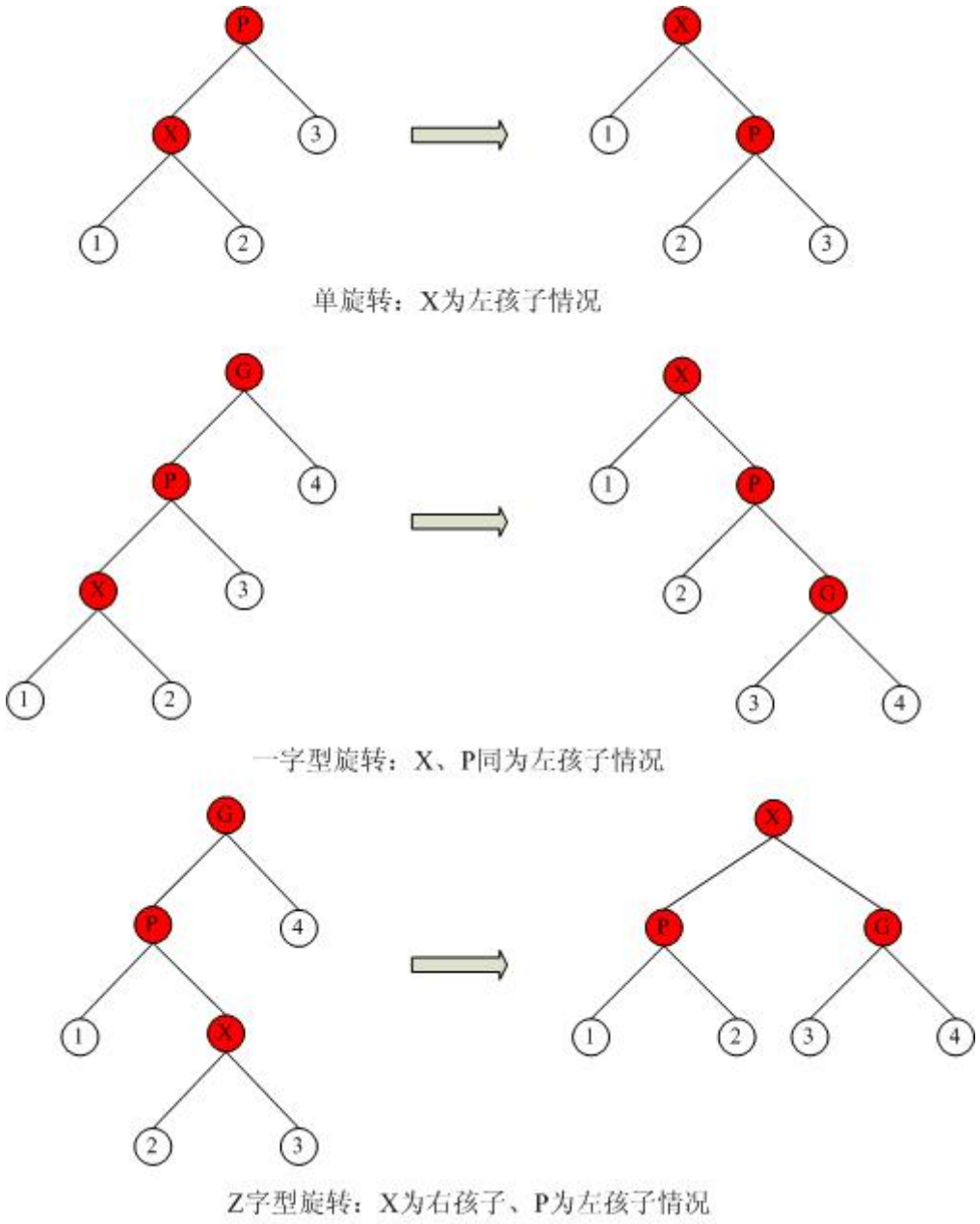
每一旋转步骤都是下列三种情况之一：

第一种情况：如果 P 是树根，则旋转连接 X 和 P 的边（这种情况是最后一步）。

（如果 X 是左儿子就右旋，如果 X 是右儿子就左旋）

第二种情况：如果 P 不是树根，而且 X 和 P 本身都是左孩子或者都是右孩子，则先旋转连接 P 和 G 的边，然后再旋转连接 X 和 P 的边。

第三种情况：如果 P 不是树根，而且 X 是左孩子，P 是右孩子，或者相反，则先旋转连接 X 和 P 的边，再旋转连接 X 和新的 P 的边。



图一

4. 伸展树的自底向上伸展（bottom-up splay）伪码

假设在当前伸展树中的 X 节点处进行伸展，X 的父亲节点为 P(X)（如果 X 的父亲节点存在），X 的祖父节点为 G(X)（如果 X 的祖父节点存在）。

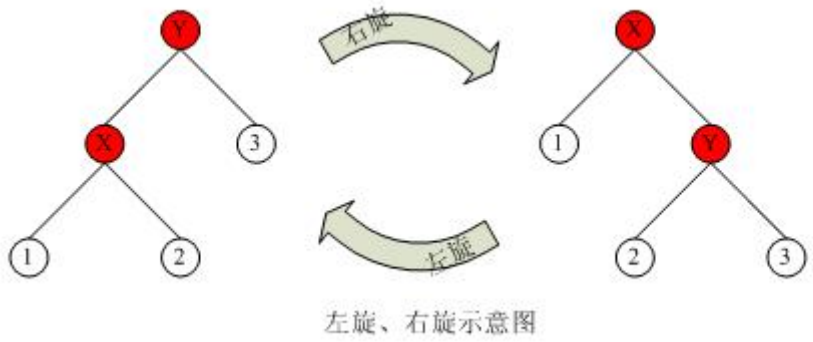
```
FUNC bottom-up-splay
    DO
        IF X 是 P(X)的左孩子节点 THEN
            IF G(X)为空 THEN
                右旋 P(X)
            ELSEIF P(X)是 G(X)的左孩子节点 THEN
                右旋 G(X)
                右旋 P(X)
```

```
            ELSE
                右旋 P(X)
                左旋 P(X)（注意：经过上一次右旋后此处的 P(X)和上一个 P(X)不一样）
            ENDIF
        ELSE X 是 P(X)的右孩子节点 THEN
            IF G(X)为空 THEN
                左旋 P(X)
            ELSEIF P(X)是 G(X)的右孩子节点 THEN
                左旋 G(X)
                左旋 P(X)
            ELSE
                左旋 P(X)
                右旋 P(X) （注意：经过上一次左旋后此处的 P(X)和上一个 P(X)不一样）
            ENDIF
        ENDIF
    WHILE P(X)不为空
ENDFUNC
```

仔细分析各种情况下的旋转，对于 X、P、G 成 z 字型排列的旋转情况（上面所说的三种情况中的第三种），其第二次旋转可以延后进行，这样就可以使得第三种情况和第一种情况统一起来而简化编程，从而得到简化后的伪代码，结果如下所示：

```
FUNC bottom-up-splay
    DO
        IF X 是 P(X)的左孩子节点 THEN
            IF P(X)是 G(X)的左孩子节点 THEN
                右旋 G(X)
            ENDIF
            右旋 P(X)
        ELSE X 是 P(X)的右孩子节点 THEN
            IF P(X)是 G(X)的右孩子节点 THEN
                左旋 G(X)
            ENDIF
            左旋 P(X)
        ENDIF
    WHILE P(X)不为空
ENDFUNC
```

对于伪码中的左旋和右旋操作，之前就已经图示过（虽然没有明说），那就是图一中的第一个（表示的是右旋，左旋是对称的），如果还不是很清楚则请看下图示：



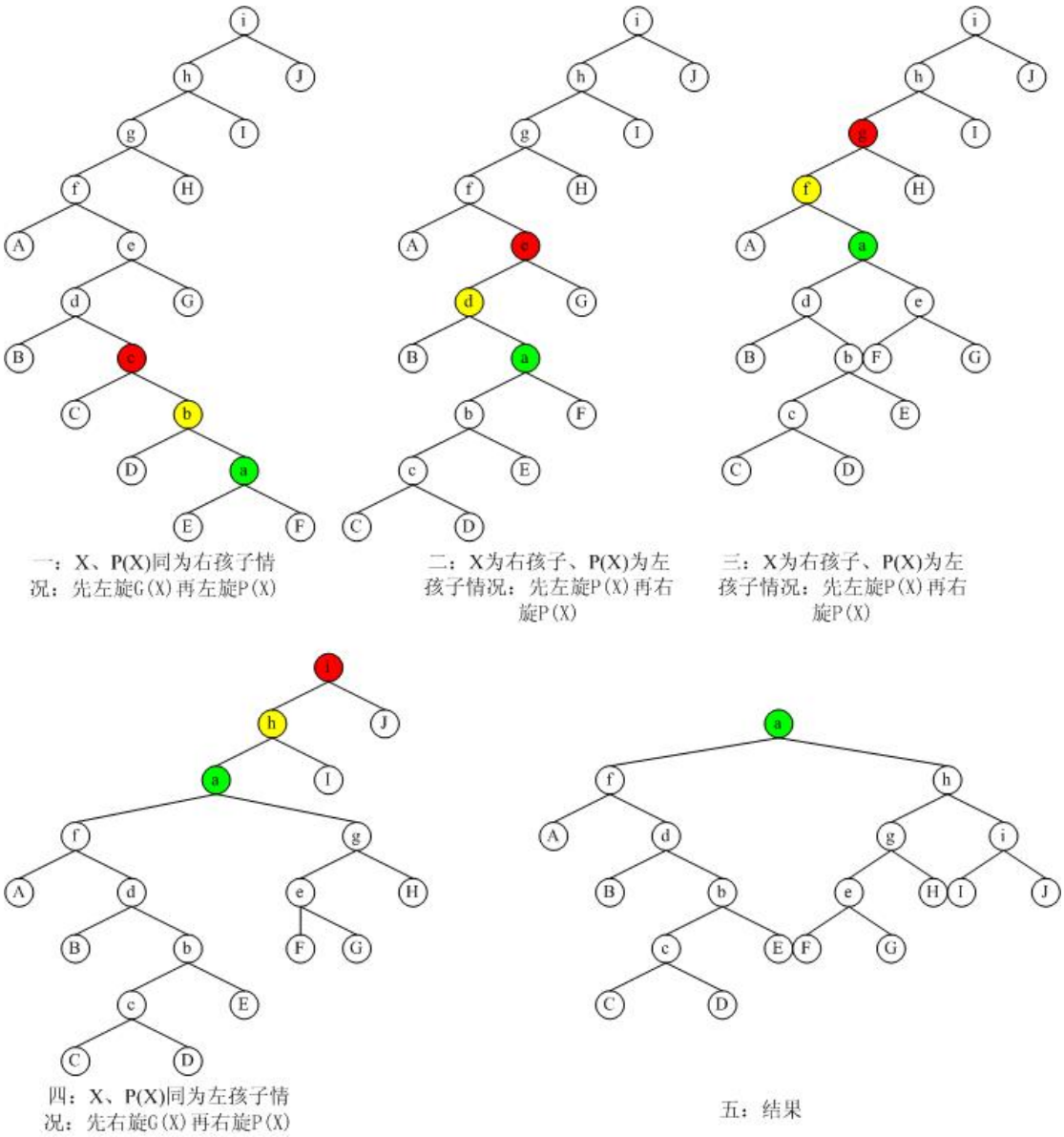
左旋、右旋示意图

图二

另外在单个的旋转过程中，如果 G(X)还有父节点（假设为 O）则先断开 G(X)与 O 之间的连接即只考虑以 G(X)为根节点的子树（假设为 T），旋转完之后再连接节点 X 和节点 O 即 X 占据旋转前 G(X)的位置（注意：旋转后 X 变成了 T 的根节点）。

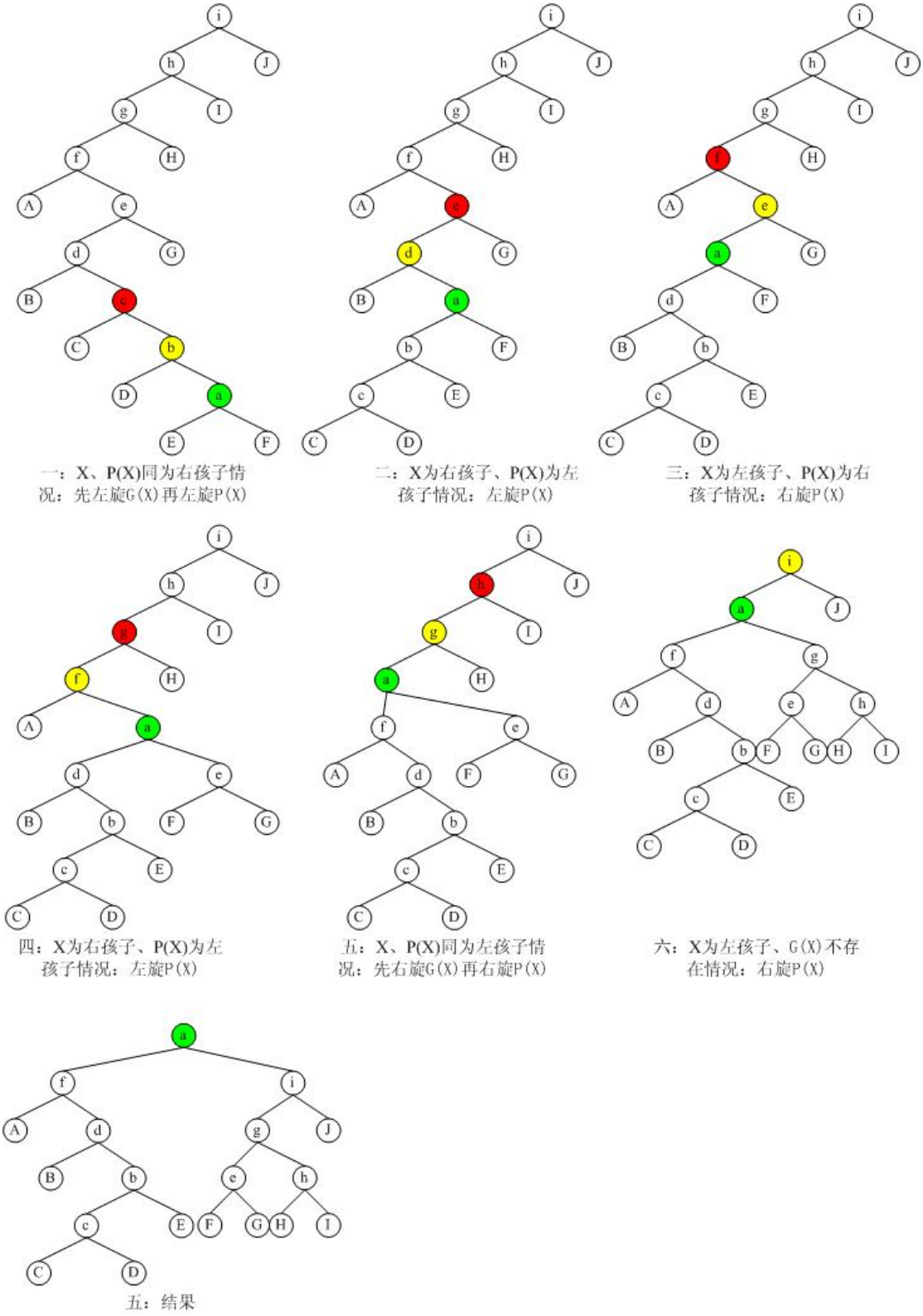
最后，值得一说的是，两种伸展伪码虽然都可以对伸展树进行伸展操作，但是对于同一伸展树在同一节点开始伸展最后得到的伸展树结构不一定完全一致（接下来的实例讲解会看到这一情况），因为第二种伪码实现使得伸展树的旋转不再是成对进行。

伸展树的自底向上伸展实例：
伪码一的旋转过程以及结果



图三

伪码二的旋转过程以及结果



图四

结论：前述的两种伸展伪码对于同一伸展树在同一节点开始伸展最后得到的伸展树结构不一定完全一致。

5. 伸展树的自顶向下伸展（top-down splay）伪码

假设在当前伸展树中的 X 节点处进行伸展，X 的父亲节点为 P(X)（如果 X 的父亲节点存在），X 的祖父节点为 G(X)（如果 X 的祖父节点存在）。

从上一节的讲述，我们知道自底向上伸展的方法是在完全查找完之后再进行的伸展操作（如果找到待查节点 X，就在节点 X 处进行伸展操作；如果未找到待查节点 X 即遇到了空节点，就在最后一个非空节点处进行伸展操作），而自顶向下伸展将在节点查找的过程中就同时进行着伸展操作。

自顶向下伸展操作将伸展树分为三部分：

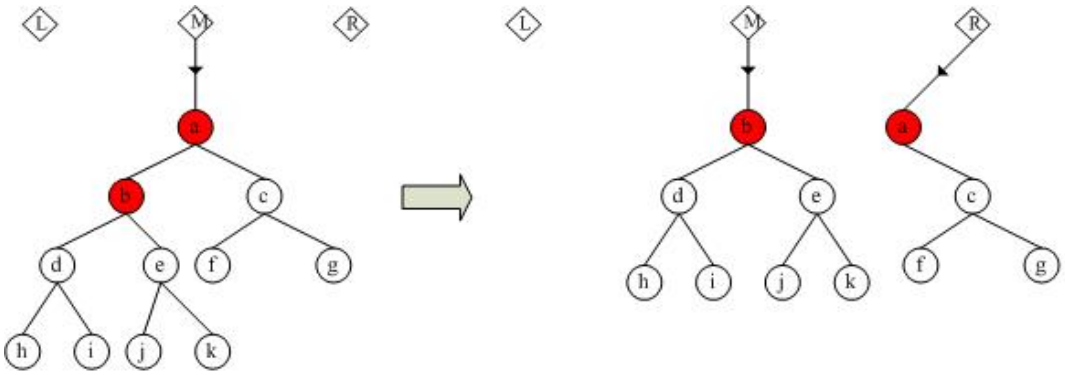
左树：包含所有已经知道比待查节点 X 小的节点。

右树：包含所有已经知道比待查节点 X 大的节点。

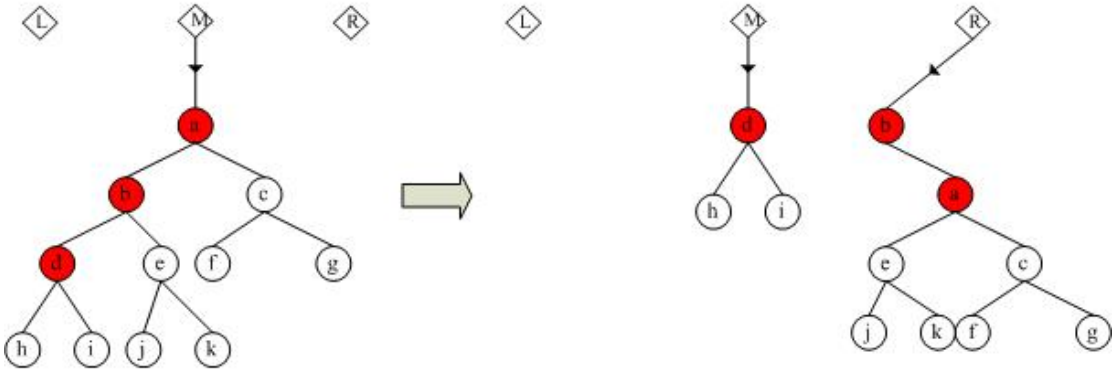
中树：包含所有其它节点。

在中树自根向下进行节点查找（每次向下比较两个节点），根据查找情况将中树中的节点移动（此处的移动是指将节点和中树的连接断开，而将节点连接到左或右树的适当位置。）到左树或右树（如有必要则会先对中树进行旋转再进行节点移动）。

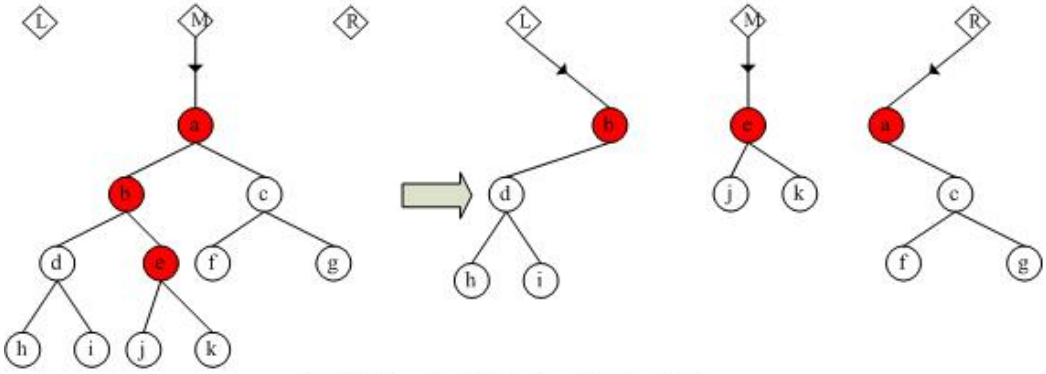
初始状态时，左树和右树都为空，而中树为整个原伸展树。随着查找的进行，左树和右树会因节点的逐渐移入变大，中树会因节点的逐渐移出变小。最后查找结束（找到或遇到空节点）时组合左中右树并是伸展树自顶向下伸展方法的最终结果。



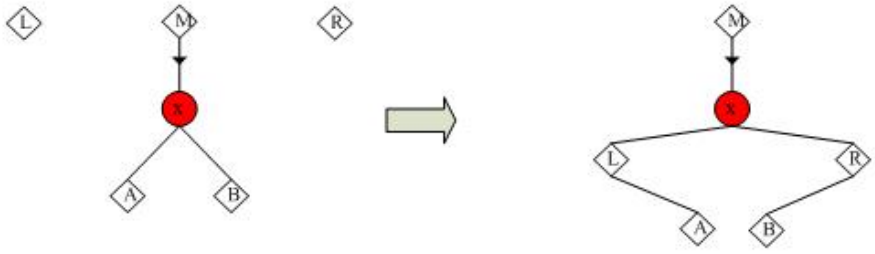
单旋转：待查节点比a小却正好等于b



一字型旋转：待查节点比a和b都小



Z字型旋转：待查节点比a小但比b要大



组合左中右树得到伸展结果

图五（图中所给旋转为 b 的左右子树都存在时而不仅限如此）

（图只画出了待查节点比 a 小的情况，比 a 大的情况可由对称关系推出）
下面给出伸展树的自顶向下伸展的伪码，其中的 L、R 分别表示左树和右树且初始值为空，M 为中树且初值为原伸展树；X 为待查节点，T 为 M 的根节点。

```
FUNC top-down-splay
    DO
        IF X 小于 T THEN
            IF X 等于 T 的左孩子 THEN
                右连接
            ELSEIF X 小于 T 的左孩子 THEN
                右旋
                右连接
            ELSE X 大于 T 的左孩子 THEN
                右连接
                左连接
            ENDIF
        ELSE X 大于 T THEN
            IF X 等于 T 的右孩子 THEN
                左连接
            ELSEIF X 大于 T 的右孩子 THEN
                左旋
                左连接
            ELSE X 小于 T 的右孩子 THEN
                左连接
                右连接
            ENDIF
        ENDIF
    WHILE 找到 X 或遇到空节点
    组合左中右树
ENDFUNC
```

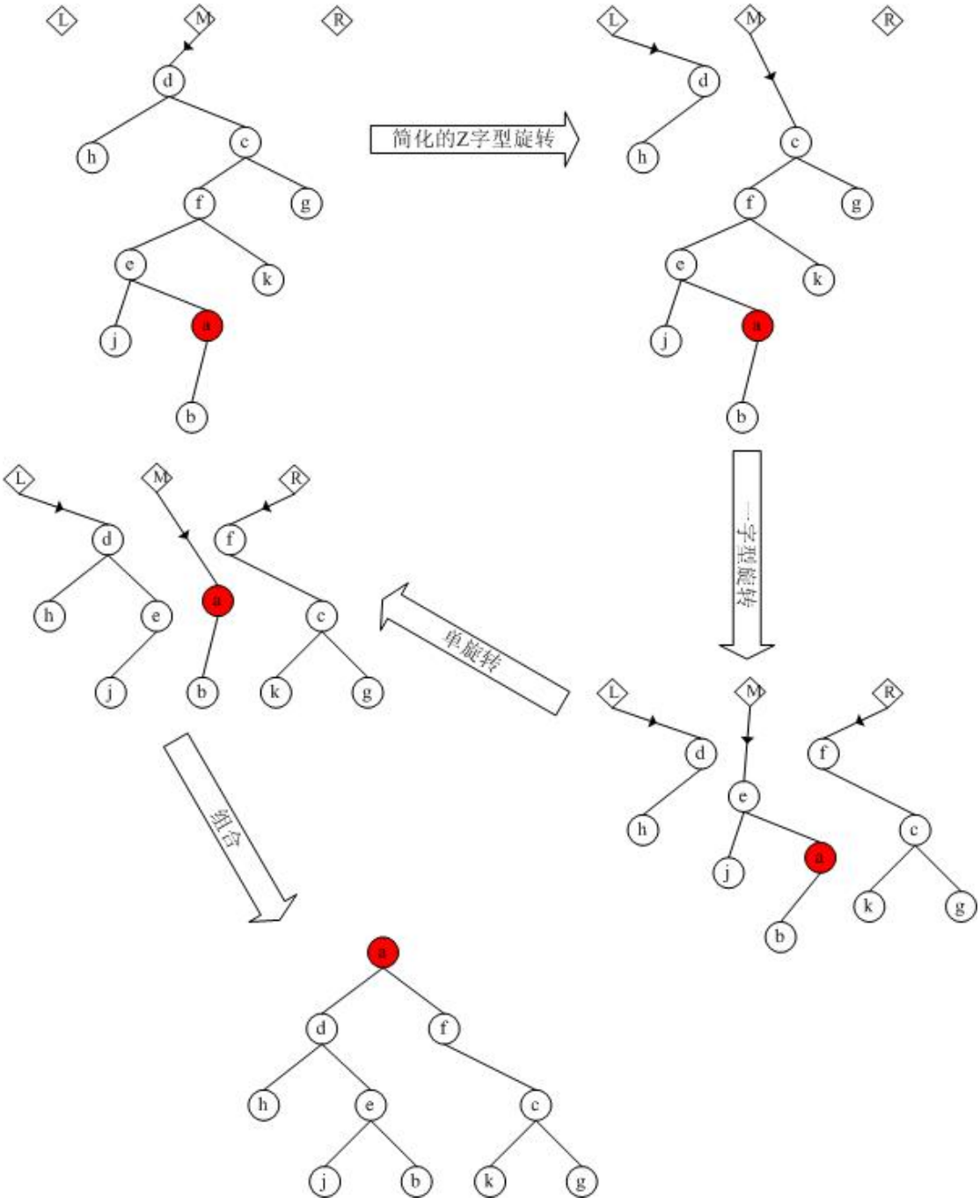
和伸展树的自底向上伸展（bottom-up splay）伪码一样，同样可以简化上一个伪码得到如下形式：

```
FUNC top-down-splay
    DO
        IF X 小于 T THEN
            IF X 小于 T 的左孩子 THEN
                右旋
            ENDIF
            右连接
        ELSE X 大于 T THEN
            IF X 大于 T 的右孩子 THEN
                左旋
            ENDIF
            左连接
```

ENDIF
WHILE 找到 X 或遇到空节点
组合左中右树
ENDFUNC

伪码中的左旋和右旋操作之前已经讲解，对于左连接和右连接可以参看图五中的第一个小图（表示的是右连接，左连接是对称的）。另外，两种伸展伪码对同一伸展树进行伸展后，得到的伸展树结构不一定完全一致。

伸展树的自顶向下伸展实例（以简化的伸展伪码来示例）：



图应该很清晰，应该很好懂。有一点需要说明的是左连接和右连接的新加入进来的节点的连

接点在哪儿，怎么理解。对于左连接，因为新连接进来的所有节点比左树中的所有节点都要大，所以连接点应该在左树的最右边，所以应该顺着左树根节点一直往右孩子节点向下搜索，直到为空节点就是我们要找的连接点。右链接对称处理。

对于伸展树的基础知识讲到这就算结束了，也许没讲得很透彻，但是对于我们分析 lighttpd 源码分析已经足够，如果需要更细致的了解伸展树这一数据结构，可以参考本节后列出的相关资料。

6. lighttpd 源码分析

splay_tree 定义在 splaytree.h 文件内，如下：

```
typedef struct tree_node {
    struct tree_node * left, * right;
    int key;

    int size;     /* maintained to be the number of nodes rooted here */

    void *data;
} splay_tree;
```

其中的 size 元素记录该节点有多少个子孙节点（并把自己也计算在内）。

另外，splaytree.h 内有一个函数申明

```
splay_tree * splaytree_size(splay_tree *t);
```

没有找到定义体，为什么？也许之前有实现体，后来改用宏而作者忘记删除这个函数申明了，因为的确紧接着就有个 splaytree_size 的宏实现（不过我期望有更合理的解释）。

splaytree.h 内另外的几个申明函数在 splaytree.c 中都有定义，其中

splaytree_splay 为伸展树的伸展操作（采用的是自顶向下且简化实现方式），函数在伸展树 t 中查找 key 等于 i 的节点（当然在查找的过程中会进行伸展操作，同时为保证 size 元素的正确要对其进行更新），如果找到则返回该节点（该节点经过伸展后变成根节点），否则返回空。

splaytree_insert 首先在伸展树 t 中查找 key 为 i 的节点，如果找到则直接返回该节点，函数结束；否则新建立一个节点，其 key 为 i，附加数据由参数 data 指针提供，然后将该节点作为新根节点插入伸展树 t 并作为返回值返回。

splaytree_delete 从伸展树 t 中删除 key 为 i 的节点，如果该节点存在的话。对于该函数中的 x->right = t->right;一句，也许不好理解（为什么可以直接就这样赋值了？），其实应看到因为 i 大于 t 的左子树内的所有节点值（因为 i 和 t 的节点值相等，见前面的 if (compare(i, t->key) == 0)判断），因此执行完 splaytree_splay(t->left, i)后，返回的 x 其必没有右孩子节点，所以可以直接赋值。

find_rank 函数功能为查找左子树包含节点数目刚好为 r 的节点。

只要理解了伸展树的基本知识，上面几个函数都不难懂，唯一需要注意的是在对伸展树实现的过程中对结构体元素 size 的处理和使用。

7. 参考

Daniel Sleator, Robert Tarjan. Self-Adjusting Binary Search Trees. Journal of the Association for

Computing Machinery. Vol. 32, No. 3, July 1985, pp. 652-686.。
http://techunix.technion.ac.il/~itai/
http://cs.nyu.edu/algvis/java/SplayTree.html
http://www.ibr.cs.tu-bs.de/courses/ss98/audii/applets/BST/
http://www.ibr.cs.tu-bs.de/courses/ss98/audii/applets/BST/Splay-Algorithm.html
http://www.ibr.cs.tu-bs.de/courses/ss98/audii/applets/BST/SplayTree-Example.html
http://www.link.cs.cmu.edu/link/ftp-site/splaying/
http://www.link.cs.cmu.edu/cgi-bin/splay/splay-cgi.pl/0.1.2.3.4.5.6.7.8.9.10.11/?op=splay&arg=6
Mark Allen Weiss. Data Structures and Algorithm Analysis in C (Second Edition)