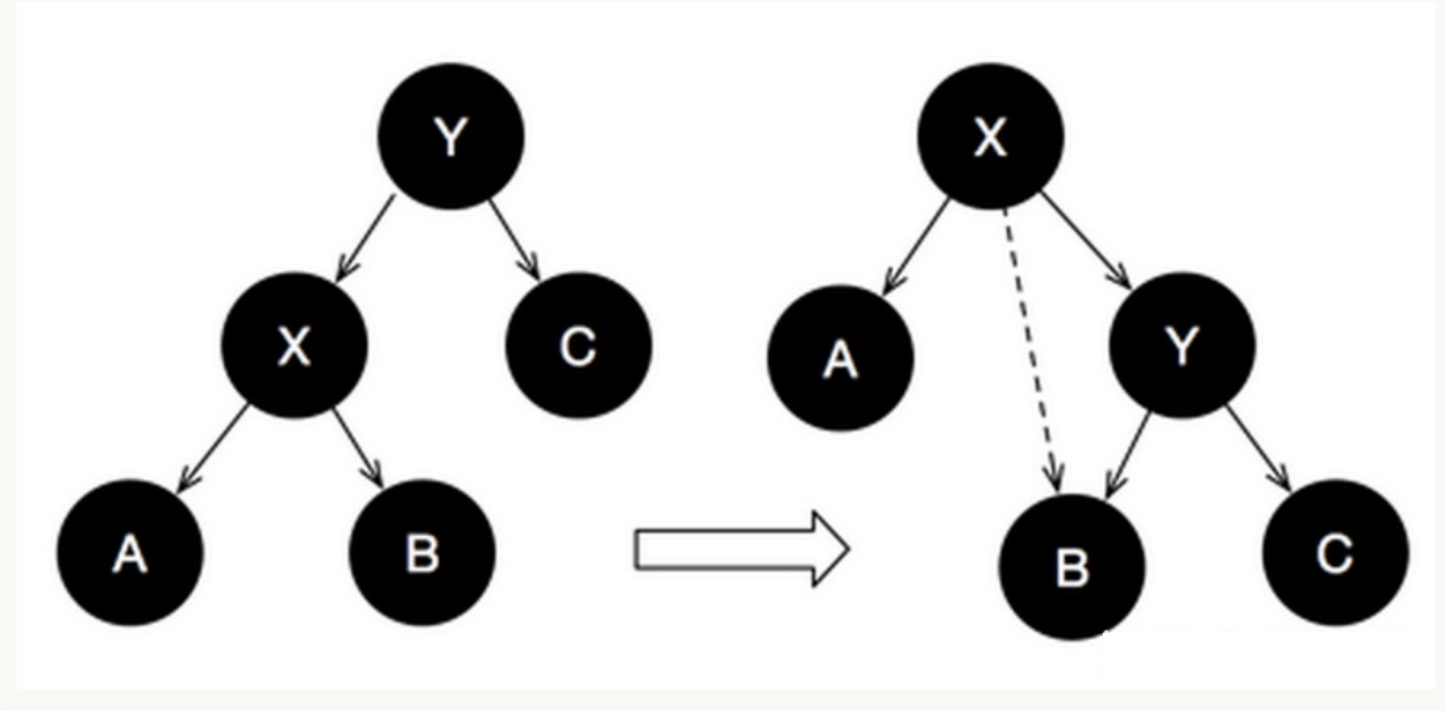
伸展

伸展树的伸展操作其实是用一个或多个旋转组合而成的。旋转的特点是不会破坏树的有序性。伸展树的伸展包含三种旋转：单旋转，一字形旋转和之字形旋转。为了便于解释，我们假设当前节点为X，X的父亲节点为Y，X的祖父节点为Z。

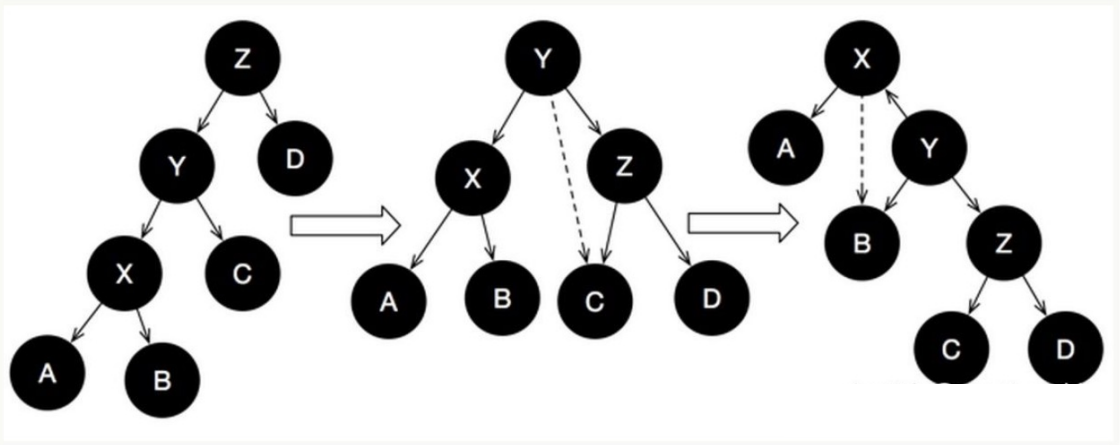
单旋转

又叫Zig或Zag。节点X与Y都是在同一条线上，X是Y的左孩子，Y是根。X要伸展到根的位置，需要做一次右旋（Y是在X的右边，因此要右旋）；如果X是Y的右孩子，那么我们要做一次左旋操作。经过旋转，X成为二叉查找树T的根节点，调整结束。



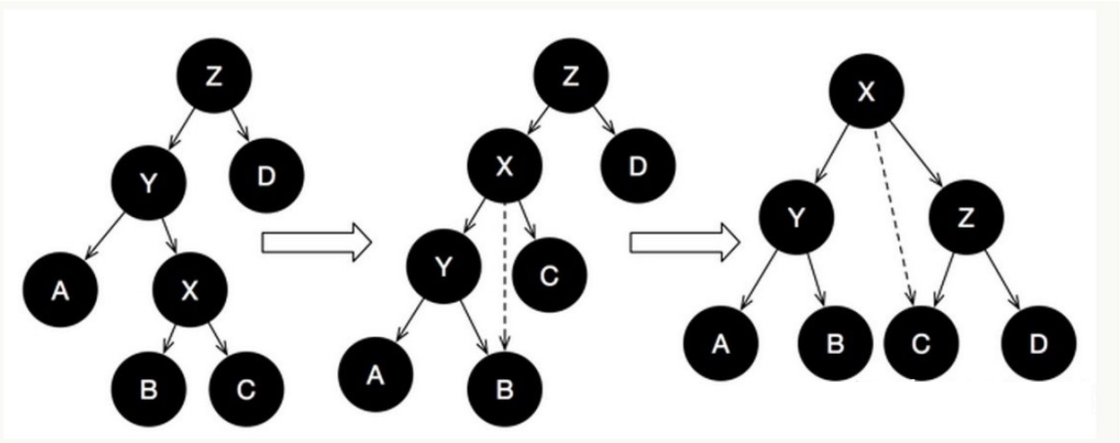
一字型旋转

又叫Zig-Zig或Zag-Zag。节点X的父节点Y不是根，Y的节点Z才是我们想要伸展到的位置，并且X，Y，Z是在同一条线上。这里我们可以先旋转Y的父节点，将Y升上去，间接将X升上去，再将X旋转上去。这相当做两次左旋或右旋操作。



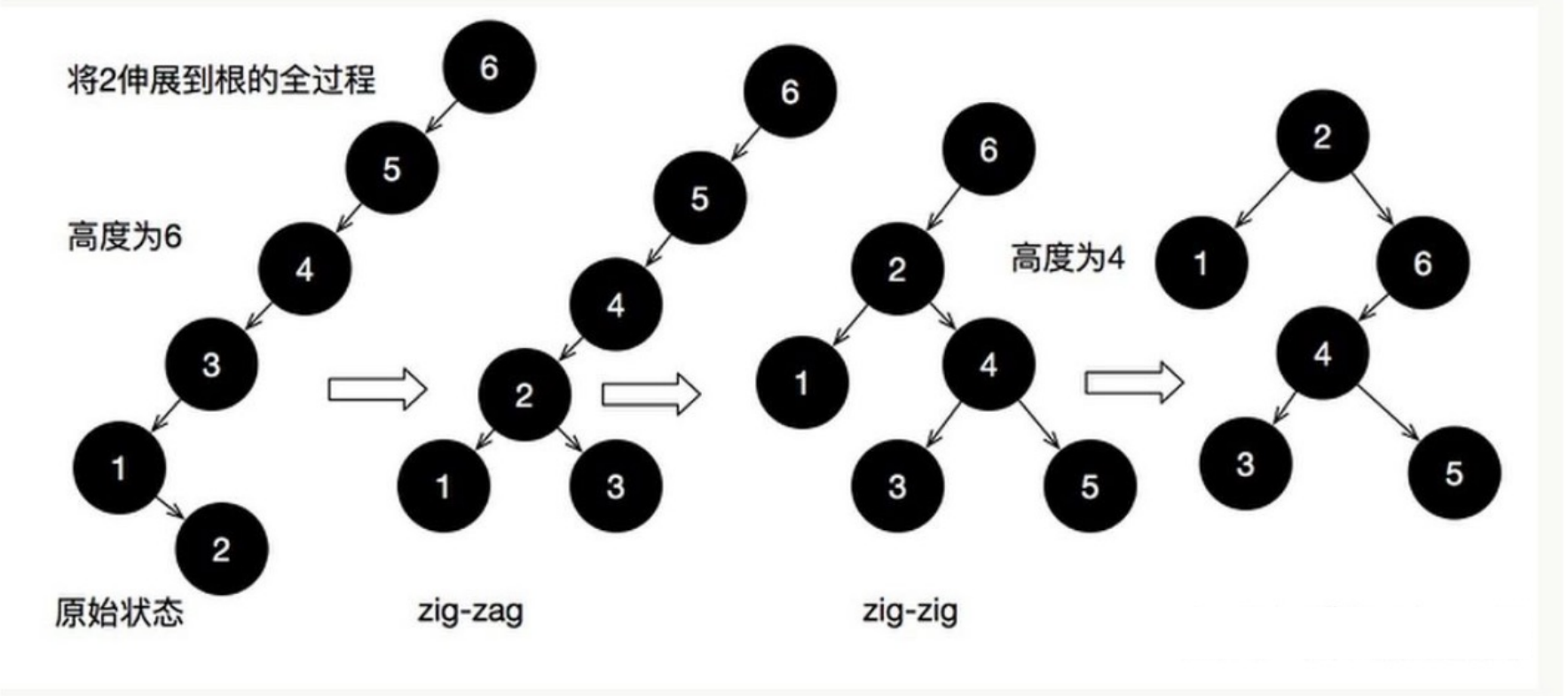
之字形旋转

又叫Zig-Zag或Zag-Zig。X，Y，Z不是在同一条线上，那么我们先将X旋转到Y的位置，然后再将X用另一个方向的旋转到X的位置。

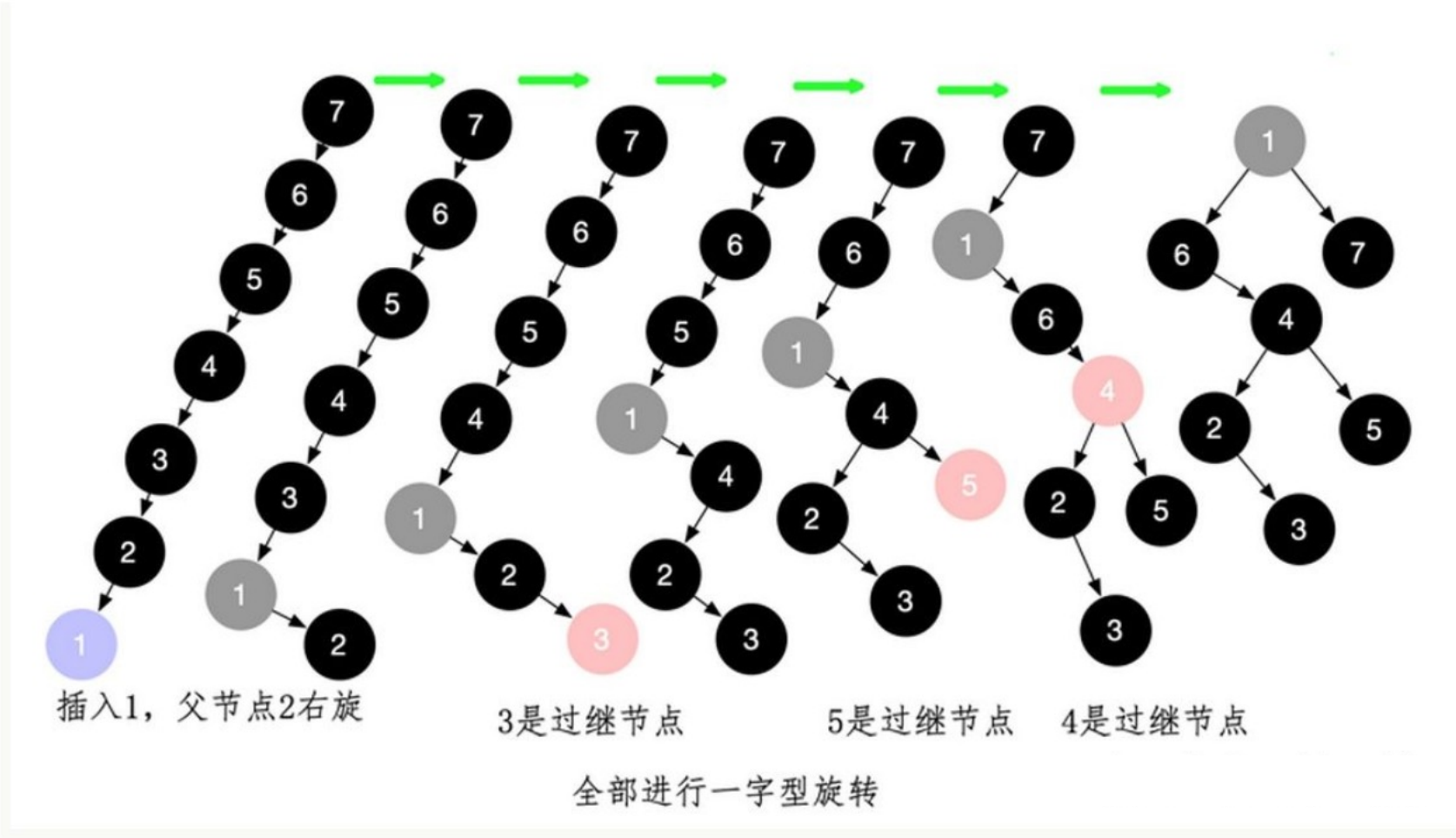


伸展操作带来的收益：

它不仅将访问的节点移动到根处，而且还有把沿途经过的节点也进行挪动，将它们的深度减少为原来的一半左右。



下图插入 节点1 后发生伸展操作的一系列分解图。在对节点1进行访问后（花费N-1个单元的时间），对节点2的访问只花费 N/2 个时间单元而不是 N - 2个时间单元。



```
// data structure that represents a node in the tree
struct Node {
        int data; // holds the key
        Node *parent; // pointer to the parent
        Node *left; // pointer to left child
        Node *right; // pointer to right child
};

// rotate left at node x
void leftRotate(NodePtr x) {
        NodePtr y = x->right;
        x->right = y->left;
        if (y->left != nullptr) {
                y->left->parent = x;
        }
        y->parent = x->parent;
        if (x->parent == nullptr) {
                this->root = y;
        } else if (x == x->parent->left) {
                x->parent->left = y;
        } else {
                x->parent->right = y;
        }
        y->left = x;
        x->parent = y;
}
```

```
// rotate right at node x
void rightRotate(NodePtr x) {
        NodePtr y = x->left;
        x->left = y->right;
        if (y->right != nullptr) {
                y->right->parent = x;
        }
        y->parent = x->parent;
        if (x->parent == nullptr) {
                this->root = y;
        } else if (x == x->parent->right) {
                x->parent->right = y;
        } else {
                x->parent->left = y;
        }
        y->right = x;
        x->parent = y;
}
```

```
// splaying
void splay(NodePtr x) {
        while (x->parent) {
                if (!x->parent->parent) {
                        if (x == x->parent->left) {
                                // zig rotation
                                rightRotate(x->parent);
                        } else {
                                // zag rotation
                                leftRotate(x->parent);
                        }
                } else if (x == x->parent->left && x->parent == x->parent->parent->left) {
                        // zig-zig rotation
                        rightRotate(x->parent->parent);
                        rightRotate(x->parent);
                } else if (x == x->parent->right && x->parent == x->parent->parent->right) {
                        // zag-zag rotation
                        leftRotate(x->parent->parent);
                        leftRotate(x->parent);
                } else if (x == x->parent->right && x->parent == x->parent->parent->left) {
                        // zig-zag rotation
                        leftRotate(x->parent);
                        rightRotate(x->parent);
                } else {
                        // zag-zig rotation
                        rightRotate(x->parent);
                        leftRotate(x->parent);
                }
        }
}
```