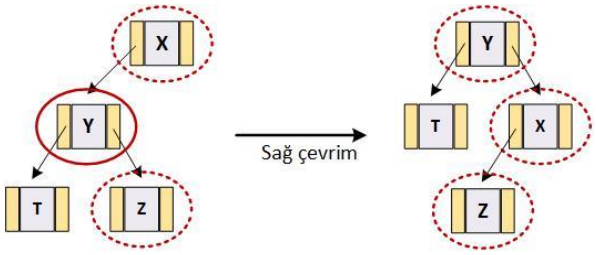
SPLAY AĞAÇLARI

Splay ağaçları son işlem gören düğümü kök düğüme getirmeyi amaçlayan özel ikili ağaçlardır. En son işlem gören düğümün sık işlem görebileceğini varsayarak ulaşılması kolay olsun diye üst sıralara taşır. Splay ağaçları için de ikili arama ağaçlarında kullanılan benzer düğüm yapısı kullanılır. Bu düğüm yapısı aşağıda verilmiştir.

```
static class node
{
    int key;
    node left, right;
};
```

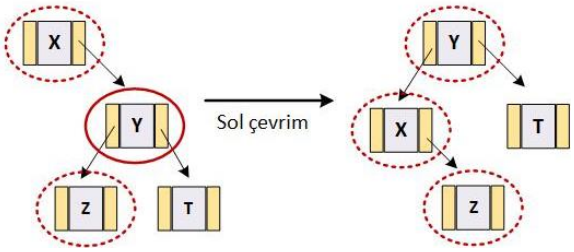
Splay ağaçlarında AVL ağaçlarında olduğu gibi sağ ve sol çevrimler gerçekleştirilir. Şekil 1'de sağ çevrim işlemi gösterilmiştir. Hemen yanında ise bu işlemi gerçekleştirmek için gereken JAVA kodu verilmiştir.



Şekil 1. Sağ çevrim işlemi

```
static node rightRotate(node x)
{
    node y = x.left;
    x.left = y.right;
    y.right = x;
    return y;
}
```

Şekil 2'de benzer olarak sol çevrim işlemi gösterilmektedir. Ve JAVA kodu hemen altında verilmiştir.



Şekil 2. Sol çevrim işlemi

```
static node leftRotate(node x)
{
    node y = x.right;
    x.right = y.left;
    y.left = x;
    return y;
}
```

Videoda açıklanan splay ağacının rekürsif çağrımı ise aşağıdaki kod bloğu ile verilmiştir.

```
static node splay(node root, int key)
{
    // Base cases: root is null or
    // key is present at root
    if (root == null || root.key == key)
        return root;

    // Key lies in left subtree
    if (root.key > key)
    {
        // Key is not in tree, we are done
        if (root.left == null) return root;

        // Zig-Zig (Left Left)
        if (root.left.key > key)
        {
            // First recursively bring the
            // key as root of left-left
            root.left.left = splay(root.left.left, key);

            // Do first rotation for root,
            // second rotation is done after else
            root = rightRotate(root);
        }
        else if (root.left.key < key) // Zig-Zag (Left Right)
        {
            // First recursively bring
            // the key as root of left-right
            root.left.right = splay(root.left.right, key);

            // Do first rotation for root.left
            if (root.left.right != null)
                root.left = leftRotate(root.left);
        }

        // Do second rotation for root
        return (root.left == null) ?
                               root : rightRotate(root);
    }
    else // Key lies in right subtree
    {
        // Key is not in tree, we are done
        if (root.right == null) return root;

        // Zag-Zig (Right Left)
        if (root.right.key > key)
        {
            // Bring the key as root of right-left
            root.right.left = splay(root.right.left, key);

            // Do first rotation for root.right
            if (root.right.left != null)
                root.right = rightRotate(root.right);
        }
        else if (root.right.key < key)// Zag-Zag (Right Right)
        {
            // Bring the key as root of
            // right-right and do first rotation
            root.right.right = splay(root.right.right, key);
            root = leftRotate(root);
        }

        // Do second rotation for root
        return (root.right == null) ?
                                root : leftRotate(root);
    }
}
```