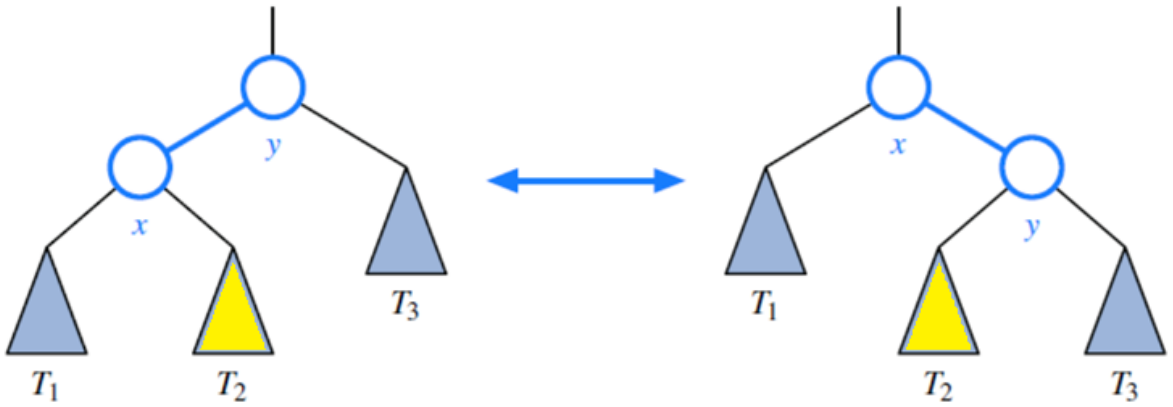
CSE214 Data Structures

Balanced Trees

YoungMin Kwon

Balanced Search Trees

- Rotation operation: *rotate(x)*



  - Keys in T1 are less than x
  - Keys in T2 are in between x and y
  - Keys in T3 are greater than y

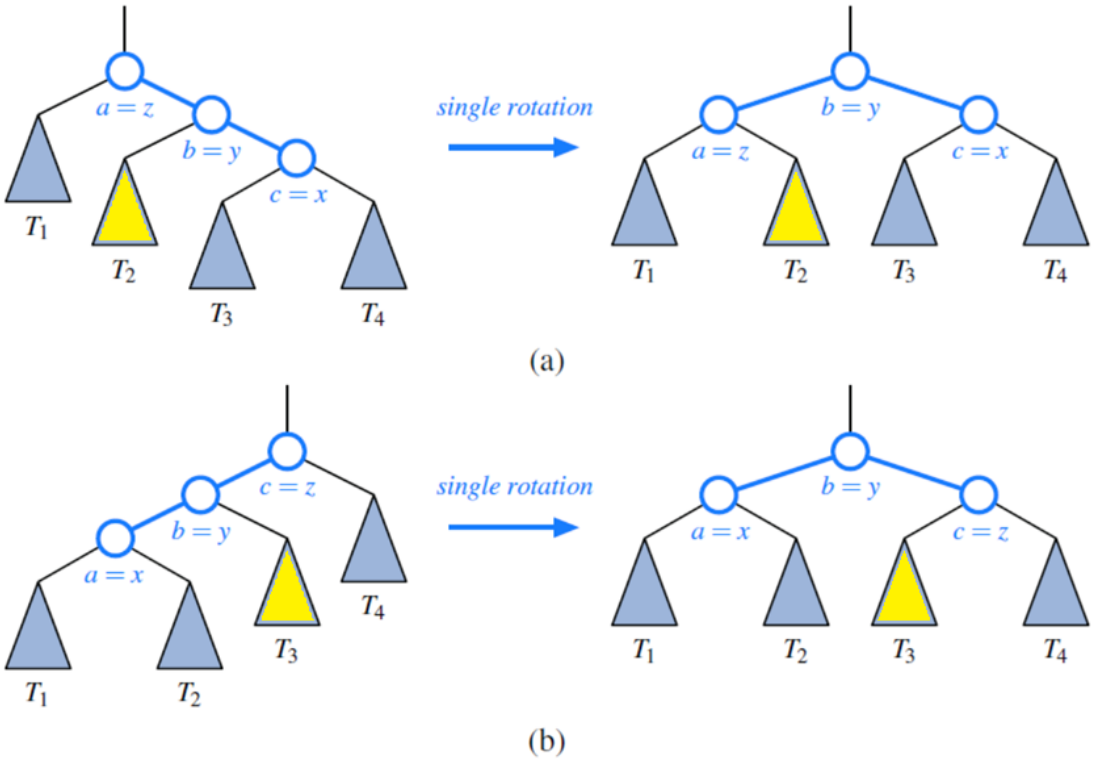
Rotation Operation

```
protected void rotate(Node<E> node) {
    Node<E> x = node;
    Node<E> y = x.parent;   //assumed to be exist
    Node<E> z = y.parent;   //may be null

    //TODO: implement this method
    // - make z the parent of x or make x the root if z is null
    // - make y the parent of x.left or x.right
    // - make x the parent of y
}
```

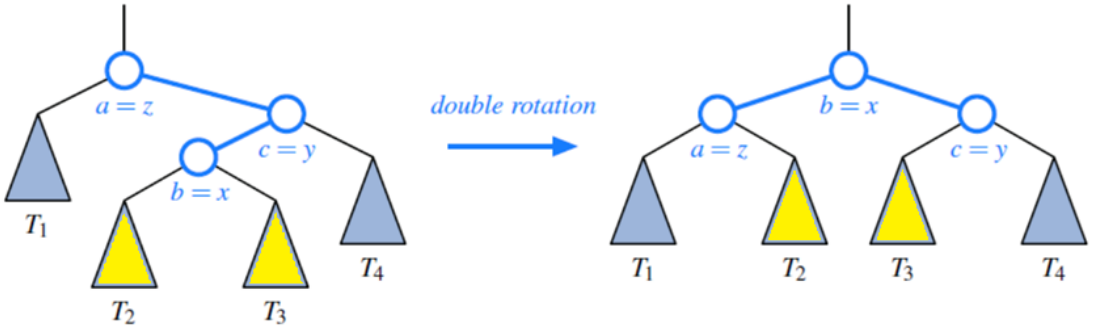
Balanced Search Trees

- Trinode restructuring: *restructure(x)*
  - Case 1: y and x are on the same side
  - *restructure(x) = rotate(y)*

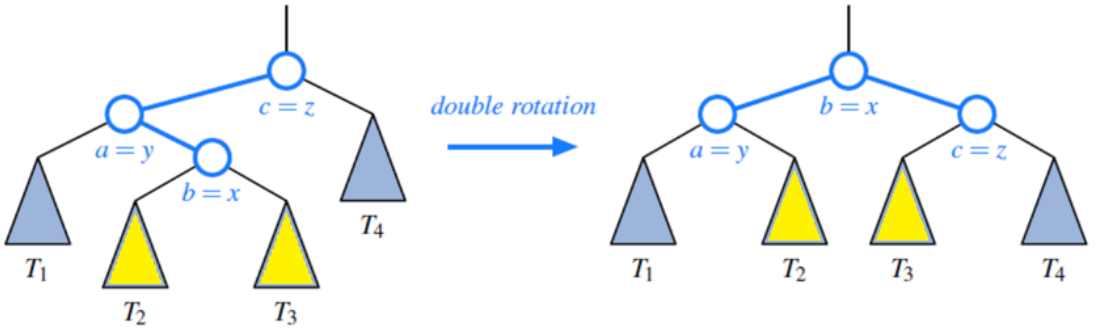


Balanced Search Trees

- Trinode restructuring: *restructure(x)*
  - Case 2: y and x are on different sides
  - *restructure(x) = rotate(x) and rotate(x)*



(c)



(d)

Restructure Operation

```
protected Node<E> restructure(Node<E> node) {
    Node<E> x = node;
    Node<E> y = x.parent;
    Node<E> z = y.parent;

    //TODO: implement this method
    // - case 1: single rotation on y (x, y, z are on the same side)
    // - case 2: double rotation on x (x, y, z are on different sides)
    // - return the new middle node
}
```

Test Result

```
public static <E extends Comparable<E>> void test(E[] arr) {
    BSTRestructure<E> tree = new BSTRestructure<E>();
    for(E e : arr)
        tree.add(e);
    tree.preorder();
    System.out.println();

    tree.restructure(tree.root.left.left.left);
    tree.preorder();
    System.out.println();

    tree.restructure(tree.root.right.right.left);
    tree.preorder();
    System.out.println();
}
public static void main(String[] args) {
    Integer[] arr = {5, 3, 6, 2, 8, 1, 9, 4, 7};
    test(arr);
}
```

```
5, 3, 2, 1, 4, 6, 8, 7, 9,
5, 2, 1, 3, 4, 6, 8, 7, 9,
5, 2, 1, 3, 4, 7, 6, 8, 9,
```