Solutions to Homework 2: Search Trees

Disclaimer: These solutions have not been carefully checked. If anything seems to be fishy, please check with me.

Solution 1:

(a) The original tree and the corresponding AA tree are both shown in Fig. 1.

(b) The intermediate steps are shown in Fig. 2, and the final result is shown in the middle of the lower row.

Solution 2: First, we update the child links in the standard manner. Next, we update the parent pointers (only \( p \), \( q \), and \( p \)'s new left child are affected). After the links have been updated, we update the min, max, and size by looking at these values for our children.

```java
BinaryNode rotateRight(BinaryNode p) {
    BinaryNode q = p.left; // update child links
    p.left = q.right;
    q.right = p;
}
```
q.parent = p.parent;  // update parent links
p.parent = q;
if (p.left != null) p.left.parent = p;
setMinMaxSize(p);  // update sizes
setMinMaxSize(q);
return q;
}

void setMinMaxSize(BinaryNode p) {
    p.min = (p.left == null ? p.key : p.left.min);
    p.max = (p.right == null ? p.key : p.right.max);
    p.size = 1 + (p.left == null ? 0 : p.left.size) + (p.right == null ? 0 : p.right.size);
}

Solution 3: The (elegant) solution involves just a few adjustments to the template code. We first apply a standard descent to find the exposed node, sets the priority to \(-\infty\) (actually `Integer.MIN_VALUE`), and then walk back up the search path to the root. As we return from a call to expose, the exposed node has replaced the child. Thus, if we apply `expose` to the left subtree, a single right rotation suffices to move it to the current node. We then return this value. (The right side is symmetrical.) An example is shown in Fig. 3.

```java
TreapNode expose(Key x, TreapNode p) {
    if (p == null) { // error - key not in tree
        throw Exception("Key not found");
    }
    else if (x < p.key) { // x is smaller - search left
        p.left = expose(x, p.left);
        return rotateRight(p); // rotate the exposed node up
    }
    else if (x > p.key) { // x is larger - search right
        p.right = expose(x, p.right);
        return rotateLeft(p); // rotate the exposed node up
    }
    else { // found it
        p.priority = Integer.MIN_VALUE; // set priority to -infinity
        return p;
    }
}
```

Figure 3: Example of the solution to Problem 3.