Solutions to Homework 2: Search Trees

Solution 1: For part (a) see Fig. 1 (upper left). When key 19 is deleted, the ancestor with key 18 has a balanced factor of $-2$. It is heaviest on its left-left side, which implies that we perform a right single rotation 18. Following this, the root with 13 has a balance factor of $-2$. Since it is heaviest on its left-right side, we do a left-right double rotation at 13, which leads to the final tree in the lower left. Now all the balance factors are good.

![Figure 1: AVL balance factors and deletion.](image)

Solution 2:

(a) To perform the operation \texttt{insert}(3), we first descend the tree and insert 3 as the left child of 4 at level 1 (see Fig. 2). We return to 4 and perform \texttt{skew}(4), which performs a right rotation at 4. We then perform \texttt{split}(3), which performs a left rotation at 3 (an example of an ineffective skew-split) which moves 4 up to level 2. We return up to 6, which performs \texttt{skew}(6). This is followed by split at 4, which does nothing. Next, we return to 2. The skew at 2 does nothing, but \texttt{split}(2) does a left rotation at 2 and moves 4 up to level 3. We return to 8, which first performs \texttt{skew}(8), which does a right rotation at 8, bringing the 4 up to the root. We perform \texttt{split}(4), which does a left rotation at 4 (another ineffective skew-split) and pushes 8 up to level 4, which takes over again as the tree’s root.
(b) The operation delete(5) begins by deleting the node 5 (see Fig. 3). As a result, its parent 6 (at level 2) has a left child of nil (at level 0) and so updateLevel(6) pulls 6 and its right child 8 down to level 1. We now have the cluster of nodes 6, 7, 8, 9, 10, all at the same level. The skew at 6 has no effect, but skew(8) performs a right rotation at 8, which pulls the 7 up as the right child of 6. The skew at 8 has no effect. Next, we perform split(6), which performs a left rotation at 6 and pulls 7 up to level 2. We then do split(8), which performs a left rotation at 8 and pulls 9 up to level 2 as the child of 7. At this point the algorithm continues to return up the search path, but all these nodes satisfy the AA-tree conditions, and no further actions are taken.

Figure 3: AA-tree deletion.
**Solution 3:** The rotation code is virtually identical to the original, except for updating the flags indicating which links are threads. If \( q.right \) is a thread (that is, if \( q \) has no right child), then \( p.left \) is made a thread that points to \( q \). Otherwise, we copy the child link from \( q.right \) to \( p.left \) and we also copy its thread flag. Also, when we set \( q.right \) to point to \( p \), we reset the flag, since \( p \) is the actual child.

```java
Node rotateRight(Node p) {
    Node q = p.left
    if (q.rightIsThread) { // q has no right child
        p.left = q // p.left is a thread to q
        p.leftIsThread = true
    } else { // q has a right child
        p.left = q.right // copy it to p.left (note: p.leftIsThread = false)
    }
    q.right = p // make p the right child of q
    q.rightIsThread = false
    return q
}
```

To establish correctness, observe first that, since rotation does not alter inorder relations, any threads that pointed into \( p \) or \( q \) still do after the rotation (see Fig. 4(a)). Otherwise, consider each of the child links of \( p \) and \( q \). Each of these may be a thread, or may become a thread. Since we know that rotation works when these are standard child links, let’s assume they are threads (see Fig. 4(b)):

- **q.left**: Before the rotation this points to the inorder predecessor of \( q \), and this predecessor is unaffected by the rotation, so it does not need to be updated.

- **p.right**: Before the rotation this points to the inorder successor of \( p \), and this successor is unaffected by the rotation, so it does not need to be updated.

- **q.right/p.left**: If \( q.right \) is a thread, then \( q \)'s right subtree is empty. This means that \( p \) is \( q \)'s inorder successor, and hence \( q \) is \( p \)'s inorder predecessor. After the rotation, \( q.right \) points to \( p \) (as a child link), and \( p.left \) is a thread pointing to \( q \).

**Solution 4:** The solution is relatively simple. The ineffective skew-split combination occurs precisely when both children of node \( p \) are red, that is, they are at the same level as \( p \). To see why,
observe that if the left child is not red, then the skew rotation will not occur, and hence the split (if it occurs) is effective. After the skew, the split rotation will only occur if we have the right-right red situation, and this can only happen if p’s right child was already red, that is, it is at the same level as p. When this happens, the only change to the tree is that p’s level is increased by 1. Thus we have:

```cpp
AANode effectiveSkewSplit(AANode p) {
    if (p.left.level == p.level && p.right.level == p.level) {
        p.level += 1
        return p
    } else {
        return split(skew(p))
    }
}
```

Solution 5:

(a) For \(i \geq 0\), let \(n(i)\) denote the number of nodes at depth \(i\) in an alternating 2-3 tree (where the root is at depth 0). Clearly, \(n(0) = 1\), and for \(i \geq 1\):

\[
    n(i) = \begin{cases} 
    2n(i-1) & \text{if } i \text{ is odd} \\
    3n(i-1) & \text{if } i \text{ is even}
    \end{cases}
\]

By expanding two levels of this recurrence, it is easy to see that for any \(n \geq 2\) (irrespective of \(i\)'s parity) \(n(i) = 6n_{i-2}\). By repeatedly expanding this (or induction, if you prefer), it is easy to see that \(n(2k) = 6^k n(0) = 6^k\). Also, since \(2k + 1\) is odd, we have \(n(2k + 1) = 2 \cdot 6^k\).

Therefore, we have the following general formula for the number of nodes at level \(i\) of the alternating 2-3 tree:

\[
    n(i) = \begin{cases} 
    2 \cdot 6^{(i-1)/2} & \text{if } i \text{ is odd} \\
    6^{i/2} & \text{if } i \text{ is even}
    \end{cases}
\]

This can also be expressed without resorting to cases with the following equivalent formula

\[
    n(i) = 2^{[i/2]} 3^{[i/2]}.
\]

(b) We can use the number of nodes to derive the number of keys. If \(i\) is even, all the nodes are 2-nodes, and since each contains a single key, we have \(k(i) = n(i)\) If \(i\) is odd, all the nodes are 3-nodes, and each contains two keys, so we have \(k(i) = 2n(i)\). In summary, we have

\[
    k(i) = \begin{cases} 
    4 \cdot 6^{(i-1)/2} & \text{if } i \text{ is odd} \\
    6^{i/2} & \text{if } i \text{ is even}
    \end{cases}
\]

(Unfortunately, I can’t think of a cute formula that avoids the cases.)

Solution to Challenge Problem 1:

Clearly \(K(i) = \sum_{j=0}^{i} k(i)\). The messy complication is that we have different terms for the even and odd levels. Since we have a clean formula for each parity (even or odd), our approach is to count the number of keys in the even and odd levels separately and then combine them. Define \(E(m)\) to
be the sum of even depths \( \{0, 2, \ldots, 2m\} \) and \( O(m) \) to be the sum of odd depths \( \{1, 3, \ldots, 2m+1\} \).

(When \( m = -1 \), \( O(m) = 0 \)).

\[
E(m) = k(0) + k(2) + \cdots + k(2m) = \sum_{j=0}^{m} k(2j),
\]

\[
O(m) = k(1) + k(3) + \cdots + k(2m+1) = \sum_{j=0}^{m} k(2j + 1).
\]

We can express \( K(i) \) as a sum of these two components:

\[
K(i) = \begin{cases} 
E \left( \frac{i}{2} \right) + O \left( \frac{i}{2} - 1 \right) & \text{if } i \text{ is even} \\
E \left( \frac{i-1}{2} \right) + O \left( \frac{i-1}{2} \right) & \text{if } i \text{ is odd.}
\end{cases}
\]

For example, \( K(4) = E(2) + O(1) = (k(0) + k(2) + k(4)) + (k(1) + k(3)) \) and \( K(5) = E(2) + O(2) = (k(0) + k(2) + k(4)) + (k(1) + k(3) + k(5)) \).

We can solve for \( E(m) \) and \( O(m) \) by the formula computed for 5(b). We will make use of the geometric series \( \sum_{j=0}^{m} c^j = (c^{m+1} - 1)/(c - 1) \).

\[
E(m) = \sum_{j=0}^{m} k(2j) = \sum_{j=0}^{m} 6^j = \frac{6^{m+1} - 1}{5}.
\]

and

\[
O(m) = \sum_{j=0}^{m} k(2j + 1) = \sum_{j=0}^{m} 4 \cdot 6^j = \frac{4 \cdot 6^{m+1} - 1}{5}.
\]

Putting it all together, if \( i \) is even, we write it as \( i = 2m \), and we have

\[
K(i) = K(2m) = E(m) + O(m - 1) = \frac{6^{m+1} - 1}{5} + \frac{6^{m} - 1}{5} = 2 \cdot 6^{i/2} - 1.
\]

For \( i \) odd, we write it as \( i = 2m + 1 \), and we have

\[
K(i) = K(2m + 1) = E(m) + O(m) = \frac{6^{m+1} - 1}{5} + \frac{6^{m+1} - 1}{5} = 6^{(i+1)/2} - 1.
\]

In summary, after all of this, the formula can be expressed fairly simply:

\[
K(i) = \begin{cases} 
2 \cdot 6^{i/2} - 1 & \text{if } i \text{ is even} \\
6^{\lfloor i/2 \rfloor} - 1 & \text{if } i \text{ is odd.}
\end{cases}
\]

**Solution to Challenge Problem 2:** Recall that \( \Psi = N + 2R + 4B \), where \( N \) is the total number of nodes (both internal nodes and roots), \( R \) is the number of root nodes, and \( B \) is the number of nodes that have exactly one child (so called “bad nodes”).

(a) \( N = 24 \), \( R = 3 \), \( B = 7 \), and so \( \Psi = 24 + 2 \cdot 3 + 4 \cdot 7 = 24 + 6 + 28 = 58 \).
(b) We search three roots, so $T_b = 3$. There is no change to the potential.

(c) We have removed the five copies of node 4, added three new roots (9, 20, 5) but lost one (4), and eliminated one bad node (the root 4). Thus, $T_c = 5$, $\Delta N = -5$, $\Delta R = 3 - 1 = 2$, and $\Delta B = -1$. Therefore, $\Delta \Psi_c = -5 + 2 \cdot 2 + 4 \cdot (-1) = -5$. The amortized cost is $T_c + \Delta \Psi_c = 0$.

(d) The number of new nodes created is $T_d = 2$. We have created two new nodes (8 and 5), we have replaced three roots (5, 8, 20) with one (5), and we have not altered the number of bad nodes. Therefore, $\Delta \Psi_d = 2 + 2 \cdot (-2) + 4 \cdot 0 = -2$. The amortized cost is $T_d + \Delta \Psi_d = 0$.

(e) The number of deleted nodes is $T_e = 7$. We have removed seven nodes (all the nodes at levels 2 and higher), we have replaced two roots (5 and 7) with five roots (5, 8, 20, 7, 14), and we have eliminated four bad nodes (7, 5, 7, and 14). Therefore, $\Delta \Psi_e = -7 + 2 \cdot 3 + 4 \cdot (-4) = -17$. The amortized cost is $T_d + \Delta \Psi_d = -10$.

(f) The total actual costs are $T_b + T_c + T_d + T_e = 3 + 5 + 2 + 7 = 17$ and the total change in potential is $\Delta \Psi_c + \Delta \Psi_d + \Delta \Psi_e = -5 + -2 + -17 = -24$. Therefore, the total amortized cost is $17 - 24 = -7$. 