Announcements - 3/7

- **Midterm 1** - This Thu Mar 9 in class
  - Closed-book / Closed-notes
  - "cheat sheet" - front + back

- **Practice Problems**
  - On handouts page
We first traverse the search path and insert 5 as the left child of 8 (see Fig. 1). As we return along the recursion path, we update heights and compute balance factors. On reaching 4, we see that the balance factor is +2. Since it is right-left heavy, we do an RL double rotation at 4, which brings 8 up and 4 and 10 are its children. We continue up to the root.

![AVL-tree insertion diagram]

**Figure 1:** AVL-tree insertion.
(b) We first locate 10 and delete this node (see Fig. 2). We then return along the recursion path, and update the heights and recompute balance factors.

Figure 2: AVL-tree deletion.
Figure 3: AA-tree insertion.
Figure 4: AA-tree deletion.
Theorem: For any $h \geq 0$, if the nodes of $T_h$ are labeled according to their position in an inorder traversal of the tree (starting with 1), then the labels along the leftmost chain of tree (from leaf to root) generate the Fibonacci sequence

$$\langle F(2), F(3), F(4), F(5), \ldots, F(h + 2) \rangle,$$

where $F(h)$ denotes the $h$th Fibonacci number.

Proof: The proof is by induction on $h$. The basis cases ($h = 0$ and $h = 1$) are trivial, since they leftmost chain form the sequences $\langle 1 \rangle$ and $\langle 1, 2 \rangle$, respectively.

To prove the induction step, let us assume that $h \geq 2$, and let us assume the induction hypothesis that for any $h' < h$, the leftmost chain of the tree $T_{h'}$ yields the Fibonacci sequence $\langle F(2), F(3), F(4), \ldots, F(h' + 2) \rangle$, and we will use this to prove the theorem for $h$ itself.

By definition, the left subtree of $T_h$ has $T_{h-1}$ as its left subtree. In an inorder traversal of $T_h$, the nodes of the left subtree will be labeled first. Since $h - 1 < h$, we may apply the induction hypothesis, which implies that the leftmost chain of $T_{h-1}$ forms the Fibonacci sequence $\langle F(2), F(3), F(4), \ldots, F(h + 1) \rangle$. This subtree has height $h - 1$. Therefore, by the result proven in class, the number of nodes in this left subtree is $F((h-1)+3)-1 = F(h+2)-1$. The very next node in the inorder traversal is the root, which must then be labeled with the next number, that is, $F(h+2)$. Appending this to the end of the previous sequence, we obtain the final sequence $\langle F(2), F(3), F(4), \ldots, F(h + 1), F(h + 2) \rangle$, as desired.
Solution 4: Before presenting the answer, it is useful to have a utility function that determines which child a node \( p \) is (either 0, 1, or 2). If \( p \) has no parent (that is, it is the root) then this function returns \(-1\).

```java
int whichChild(Node p) { // which child is p?
    Node par = p.parent
    if (par == null) return -1 // p is not a child of anyone
    else if (p == par.child[0]) return 0 // it must be one of three
    else if (p == par.child[1]) return 1
    else return 2
}
```

(a) We invoke the `whichChild` function, and return the previous child \( i - 1 \), provided that \( i \) is 1 or larger.

```java
Node leftSib(Node p) { // get p’s left sibling
    int i = whichChild(p)
    if (i < 1) return null // root or leftmost child
    return p.parent.child[i-1] // return previous child
}
```

(b) We invoke the `whichChild` function, and return the next child \( i + 1 \), provided that \( i \) is neither \(-1\) nor the last child.

```java
Node rightSib(Node p) { // get p’s right sibling
    int i = whichChild(p)
    if (i < 0 || i >= p.parent.nChild-1) // root or last child
        return null
    else return p.parent.child[i+1] // return next child
}
```
nCh = 3
(c) We use the above functions to determine which child $p$ is and which are its two siblings. We first attempt a merge with the left sibling and otherwise we attempt a merge with the right sibling.

```java
Node merge(Node p) {
    int i = whichChild(p);
    Node par = p.parent;
    Node ls = leftSib(p);
    Node rs = rightSib(p);
    if (ls != null && ls.nChild == 2) {  // can merge with left?
        Key x = p.parent.key[i];   // key just before p
        return new Node(par, ls.child[0], ls.key[0], ls.child[1], x, p.child[0]);
    } else if (rs != null && rs.nChild == 2) {  // can merge with right?
        Key x = p.parent.key[i];   // key just after p
        return new Node(par, p.child[0], x, rs.child[0], rs.key[0], rs.child[1]);
    } else {
        return null;
    }
}
```
Solution 5: Throughout this problem, we let $k = \sqrt{n}$.

(a) The final column $\langle 4, \ldots, 16 \rangle$ forms a right chain of nodes. Following this, each successive column adds a new left child to each of the nodes from the previous column. See Fig. 5.

(b) Starting at the root, the longest path comes by following a path of length $k - 1$ to the rightmost node of the tree (corresponding to the node $n^2$) followed by a path of length $k - 1$ to its leftmost descendant. Thus, the tree’s height is $2(k - 1) = 2\sqrt{n} - 2$.

(c) The number of nodes per level increases linearly with each level from 1 up to $\sqrt{n}$, and then it descends linearly back to 1. Since the first level is zero, we have

$$d(i) = \begin{cases} 
  i + 1 & \text{if } 0 \leq i \leq \sqrt{n} - 1, \\
  2\sqrt{n} - (i + 1) & \text{if } \sqrt{n} \leq i \leq 2\sqrt{n} - 2.
\end{cases}$$

If you want to avoid the cases, there is an elegant way to express this using absolute values as $d(i) = \sqrt{n} - |i - \sqrt{n} + 1|$.
(d) The total insertion time $T(n)$ is the sum of the product $(i+1)d(i)$ for $i$ ranging from zero up to the height of the tree. We can split the sum between the increasing and decreasing parts, call then $I(n)$ and $D(n)$. Recalling that $k = \sqrt{n}$, we can express the increasing part as

$$I(n) = 1 \cdot 1 + 2 \cdot 2 + \ldots + k \cdot k = \sum_{i=1}^{k} i^2 = k^2 - \sum_{i=1}^{k-1} i^2.$$

(We’ll see below why I separated out the final term.) The decreasing part can be expressed as follows:

$$k = \sqrt{n} \quad D(n) = (k + 1)(k - 1) + (k + 2)(k - 2) + \ldots (2k - 1)(1).$$

Observe that this is a series of terms of the form $(k+a)(k-a)$, which can written as $(k^2 - a^2)$. Thus, we have

$$D(n) = (k^2 - 1) + (k^2 - 4) + \ldots (k^2 - (k - 1)^2) = \sum_{i=1}^{k-1} (k^2 - i^2) = \sum_{i=1}^{k-1} k^2 - \sum_{i=1}^{k-1} i^2.$$

To get the total cost, we take $I(n) + D(n)$. Observe that $\sum_{i=1}^{k-1} i^2$ in $I(n)$ and $D(n)$ cancel each other out, so we obtain

$$T(n) = I(n) + D(n) = k^2 + \sum_{i=1}^{k-1} k^2 = k^2 + (k - 1)k^2 = k \cdot k^2 = k^3.$$

Wow! All that work just to get $k^3$. So, in terms of $n$, the final cost is $T(n) = n^{3/2}$. Since there are $n$ insertions, we can express this more meaningfully as the amortized cost, which is $T(n)/n = \sqrt{n}$.

\[ \log n \leq \sqrt{n} \leq n \]
Exam:
Coverage - Up to AA-trees
  - In-class / Homework / Project / Practice probs

Structure:
① Do x to data structure (mult. parts)
   → Final result
   → Intermediate (partial credit)
② Short answer + 3
   → Be quick → Guess
   → Don't get bogged down

- Pseudo code
- Analysis → Like Sd ? → Amortization ?
- Proof → Induction

Write something
Thumbs up/down