Announcements - 5/11

- Prog. Assign 3
  - Due Wed of next week

- Sun, May 14, 2-3 pm in CSIC 2117
  - Review session for final

- Mon, May 15, 4-6 pm - Final Exam in IRB 0324
  - Closed-book/Closed-notes
  - 3 cheat sheets (front + back)
  - Coverage = 40% post exam 2
  - 30% exam 1 - exam 2
  - 30% pre exam 1

Big auditorium
Solution 1:

Linear probing

```
0  1  2  3  4  5  6  7  8  9 10 11 12 13
X M L X H C I J F B
```

3 probes

```
0  1  2  3  4  5  6  7  8  9 10 11 12
Y A M L X H C I J F B
```

2 probes

```
0  1  2  3  4  5  6  7  8  9 10 11 12
Y A M L X H Z C I J F B
```

10 probes

Quadratic probing

```
0  1  2  3  4  5  6  7  8  9 10 11 12
F W P L J Q N X
```

4 probes

```
0  1  2  3  4  5  6  7  8  9 10 11 12
F W P L J Q Y N X
```

4 probes

```
0  1  2  3  4  5  6  7  8  9 10 11 12
F W P L J Q Y N X
```

Fails

Figure 1: Hashing with linear and quadratic probing.
Figure 2: Hashing with double hashing.
Solution 2:

```
insert(23)

Figure 3: Solution to Problem 2(a): insert(23).
```
Figure 4: Solution to Problem 2(b): insert(55).

\[ \text{keys} = \#\text{children} - 1 \]
Figure 5: Solution to Problem 2(c): delete(62).
Solution 3:

Text: a b b a b b a b a $

<table>
<thead>
<tr>
<th>Index</th>
<th>Substring ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>abbabb</td>
</tr>
<tr>
<td>1</td>
<td>bbabb</td>
</tr>
<tr>
<td>2</td>
<td>bab</td>
</tr>
<tr>
<td>3</td>
<td>abbaba</td>
</tr>
<tr>
<td>4</td>
<td>bbaba</td>
</tr>
<tr>
<td>5</td>
<td>baba</td>
</tr>
<tr>
<td>6</td>
<td>ab$</td>
</tr>
<tr>
<td>7</td>
<td>b$</td>
</tr>
<tr>
<td>8</td>
<td>a$</td>
</tr>
<tr>
<td>9</td>
<td>$</td>
</tr>
</tbody>
</table>

Index | Substring ID |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ab$</td>
</tr>
<tr>
<td>1</td>
<td>bab</td>
</tr>
<tr>
<td>2</td>
<td>bab$</td>
</tr>
<tr>
<td>3</td>
<td>babbab</td>
</tr>
<tr>
<td>4</td>
<td>bababb</td>
</tr>
<tr>
<td>5</td>
<td>baba</td>
</tr>
<tr>
<td>6</td>
<td>ab$</td>
</tr>
<tr>
<td>7</td>
<td>b$</td>
</tr>
<tr>
<td>8</td>
<td>a$</td>
</tr>
<tr>
<td>9</td>
<td>$</td>
</tr>
</tbody>
</table>

Figure 6: Suffix tree.
Solution 4:

```java
int pcHelper(Node p, String pattern) {
    if (p == null) // fell out of the tree?
        return 0
    else if (pattern.size == 0) // run out of pattern symbols?
        return p.weight // everything in this subtree matches
    else if (p.isExternal) // hit external too soon?
        return 0
    else if (p.key < pattern[0]) // scan for next pattern symbol
        return pcHelper(p.nextSibling, pattern)
    else if (p.key == pattern[0]) // matched this symbol?
        return pcHelper(p.firstChild, pattern.substring(1)) // recurse
    else return 0 // gone too far - no match
}
```
int wcHelper(Node p, String pattern) {
    if (p == null) // fell out of the tree?
        return 0
    else if (p.isExternal) { // external node
        if (pattern.size == 0) // matched last pattern symbol?
            return 1
        else // more pattern symbols remain?
            return 0
    } else { // internal node
        if (pattern.size == 0) // no more pattern symbols?
            return 0
        else if (pattern[0] == '*') // wildcard symbol?
            return wcHelper(p.firstChild, pattern.substring(1)) +
                   wcHelper(p.nextSibling, pattern) // wildcard symbol?
        else if (p.key < pattern[0]) // keep looking for next symbol
            return wcHelper(p.nextSibling, pattern)
        else if (p.key == pattern[0]) // matched this symbol?
            return wcHelper(p.firstChild, pattern.substring(1)) // next symbol
        else return 0 // no match among siblings
    }
}

Figure 7: Trace of recursive calls by wcHelper("*ab") (blue) and return values (red).
Solution to the Challenge Problem: Tortoise and hare - Two pointers, the hare moves twice as fast as the tortoise. A cycle is detected when the hare passes the tortoise.

```java
openOrClosed(Node head) {
    Node tortoise = hare = head // start the race
    while (hare != null && hare.next != null) { // hare hasn't hit end
        hare = hare.next.next // advance hare by 2
        if (tortoise == hare || tortoise.next == hare) // hare passes tortoise?
            return "closed" // ...must be looped
        tortoise = tortoise.next // advance tortoise by 1
    }
    return "open"
}
```

We claim that the running time is $O(n)$, where $n$ is the number of nodes. If the list is open, then we discover when the hare hits the end, after $n/2 = O(n)$ iterations. On the other hand, if the list has a cycle, let $m$ denote the number of nodes in the cycle. After $n - m$ iterations, both the tortoise and the hare have made it into the cycle. Because the hare moves twice as fast, after an additional $m$ iterations, the tortoise will make one full lap around the cycle, but the hare will make two full laps. So, no matter where they started in the cycle, the hare must have passed the tortoise at some point. The total number of iterations before detecting this is at most $n - m$ (to get both into the cycle) plus $2m$ (chasing each other within the cycle), which yields a total running time of

$$(n - m) + 2m = n + m \leq 2n = O(n).$$