Errata & Updates

• Links on main site have been updated to reflect correct filenames

• An excerpt from “Data Structures Demystified” has been placed on the simon server. You can access it via scp:

• The homework assignment has been updated to clarify the requirements. It has been posted to the website.

• Pop quiz opportunities were listed as 5 @ 2pts each. It should have said between 5 and 10 quizzes @ 2 pts each
  – Remember, these are only bonus points (additive); they are not subtracted from your overall grade

• Attendance & Class Participation will influence your grade where it is borderline between 2 grades
  – Example: On cusp of B- and C, good attendance & participation will produce a B-
Example Pop Quiz Questions

1. What’s the difference between a Stack & a Queue? [2 pts]
2. Give one advantage of an Array (versus a Linked List) [1 pt]
3. Give one advantage of a Linked List (versus an Array) [1 pt]
4. Give one disadvantage of a Hash [1 pt]
Analysis of Algorithms
(continued from last lecture)

• Order of Complexity:
  – Data structures are populated with $n$ objects
  – As $n$ grows, operations on the data structure take longer
  – How much longer?

• Function of “$n$”
  – Linear?
  – Exponential?
  – Quadratic?

• It obviously depends on the algorithm…

• It can depend on the data set:
  – If all members of $n$ are odd, then $O(n^2)$ but
  – If all members of $n$ are even, then $O(n \log n)$
## Rates of Growth

<table>
<thead>
<tr>
<th>$n$</th>
<th>$O(1)$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
<th>$O(N^3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>24</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>4,096</td>
</tr>
<tr>
<td>1,024</td>
<td>1</td>
<td>10</td>
<td>1,024</td>
<td>10,240</td>
<td>1,048,576</td>
<td>1,073,741,824</td>
</tr>
<tr>
<td>1,048,576</td>
<td>1</td>
<td>20</td>
<td>1,048,576</td>
<td>20,971,520</td>
<td>$10^{12}$</td>
<td>$10^{16}$</td>
</tr>
</tbody>
</table>
Rates of Growth (Graphed)

- Linear -- $O(n)$
- Quadratic -- $O(n^2)$
- Cubic -- $O(n^3)$
- Logarithmic -- $O(\log n)$
- Exponential -- $O(2^n)$
- Square root -- $O(\sqrt{n})$

From: http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/index.html
Expected Performance

- Most algorithms have Upper and Lower bounds
- Big-Oh ($O$) is the Upper bound, i.e., worst-expected case
  - $T(n)$ is in $O(f(n))$ IFF:
    - Positive constants $k$, $n_0$
    - $|T(n)| \leq |k(f(n))|$ for all $n \geq n_0$
- Omega ($\Omega$) is the Lower bound, i.e., best-case
  - $T(n)$ in $\Omega(f(n))$
    - Positive constants $k$, $n_0$
    - $|k(f(n))| \leq |T(n)|$ for all $n \geq n_0$
- There is also Theta ($\theta$), “Tight bound”
  - Both Upper and Lower bounds have same performance expectation
  - I.E., $T(n)$ is in $\theta(f(n))$ IFF:
    - $T(n)$ is in $O(f(n))$ AND
    - $T(n)$ in $\Omega(f(n))$
Typical Data Management Operations

1. Insert
2. Delete
3. Search
4. Predecessor
5. Successor
6. Maximum
7. Minimum

- Also: Update (Edit) an element
- The Update action may affect the above operations, e.g. by changing a value which alters the Max

Sometimes called “Dictionary Operations”
Performance Goals

• Growth of n can be a problem
  – Some better at one operation than another
  – Trade-off decisions are normal
  – Combining structures (Hash of Arrays, e.g.) may help

• How do we quickly perform the operations (Better than “n”)?
  – Finding an element is a factor in several of the operations
    • So you can find the element you want to delete
    • Or find if an element exists in your data set
  – Ordering of elements is another factor
    • Knowing which element is Minimum or Maximum
    • Traversing the elements in (reverse/) order
An Array?

• Performance:
  1. Insert
  2. Delete
  3. Search
  4. Predecessor
  5. Successor
  6. Maximum
  7. Minimum
A Linked List?

- Performance:
  1. Insert
  2. Delete
  3. Search
  4. Predecessor
  5. Successor
  6. Maximum
  7. Minimum
A Hash?

- Performance:
  1. Insert
  2. Delete
  3. Search
  4. Predecessor
  5. Successor
  6. Maximum
  7. Minimum
What’s Behind the Problem?

• The operations on these structures are generally $k \times n$
  – i.e., $O(n)$ [where $k$ is constant]
• Consider this situation:
  – What is the asymptotic worst-case running times for each of the seven fundamental dictionary operations when the data structure is implemented as
    • A singly-linked unsorted list,
    • A doubly-linked unsorted list,
    • A singly-linked sorted list, and finally
    • A doubly-linked sorted list.
### Solution Blank

<table>
<thead>
<tr>
<th>Operation</th>
<th>singly sorted</th>
<th>singly unsorted</th>
<th>doubly sorted</th>
<th>doubly unsorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search($L, k$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert($L, x$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete($L, x$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Successor($L, x$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predecessor($L, x$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum($L$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum($L$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Solution

<table>
<thead>
<tr>
<th>Dictionary operation</th>
<th>singly unsorted</th>
<th>double unsorted</th>
<th>singly sorted</th>
<th>doubly sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search($L, k$)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Insert($L, x$)</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Delete($L, x$)</td>
<td>$O(n)*$</td>
<td>$O(1)$</td>
<td>$O(n)*$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Successor($L, x$)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Predecessor($L, x$)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)*$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Minimum($L$)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Maximum($L$)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)*$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
One Common Solution: Trees

• A tree is a special data structure with a root node (which is the “entry point”), which optionally has child and/or leaf nodes. A key feature is that the child and leaf nodes are an ordered representation of the data set.

• Let’s consider what NIST has to say about them:

• How can a tree provide an improvement on $O(n)$ for the 7 operations?
  – Finding element $[i]$ in log n time
  – Finding max, min
Trees Defined

• Definition:
  – A data structure with a special node called the “root” node which serves as the entry point to other nodes in the structure.
  – The root node has no parent
  – The root node may have 0 or more child nodes
  – *The nodes are ordered (have a specific order)*
  – Child nodes with no children of their own are called “leaf” nodes
  – Child nodes with both parent and child nodes are also called “internal” nodes (and represent “subtrees”)

• Structural Elements
  – Parent ID
  – Left Child ID
  – Right Child ID
  – Node data
Trees: Visualized

From NIST http://www.itl.nist.gov/div897/sqg/dads/HTML/tree.html
Trees: Height (Ex. 1)

- Although (c) does not look like a tree, it meets the minimum definition of a tree.

This slide & next from:
Trees: Height (Ex. 2)

This is a valid AVL tree since for each node in the tree the height of the left and right subtrees differs by at most 1. (Notice that for each NULL child, the height of that child’s subtree is -1.)

This is a valid AVL tree since for each node in the tree the height of the left and right subtrees differs by at most 1.

This tree is NOT a valid AVL tree since the height of all nodes’ subtrees do not differ by at most 1. Specifically, the root’s left and right subtrees’ heights differ by 2.

This tree is NOT a valid AVL tree since the height of all nodes’ subtrees do not differ by at most 1. Both node 20 and node 5 violate this property.
Disk Directories As A Tree

From http://claymore.rfmh.org/public/computer_resources/images/dir_structure.png
Traversing A Tree

• Consider the previous Unix Directory tree:
  • Pre-Order Traversal:
    – Work on the node before the children
    – Example: Directory Listings
  • Post-Order Traversal
    – Work on the children before the node
    – Example: Directory Sizes
Trees: Binary Search Tree ("BST")

- Trees improve efficiency when a specific ordering of nodes is adopted
- BST is one of the most widely used
- The basic Tree definition is amended with the following constraints:
  - Each node can have at most 2 children
  - The left subtree values are less than the parent
  - The right subtree values are more than the parent

Binary Trees vs BSTs

- All Binary Trees satisfy the property that each node can have at most 2 children
- The other properties (left subtree < parent, right subtree > parent) apply to a BST
- Binary Trees may have values that do not meet the left, right subtree requirement
- Only (b) is a BST:
An RFID Example of a BST

From http://www.enigmatic-consulting.com/Communications_articles/RFID/Resources/protocol_pix/navigate_binary_tree_class1.gif
BST Traversal Example

• Find (min) and Find (max) are typical operations
• The path followed is what saves us time
  – Keep Left: reach minimum value in the BST
  – Keep Right: reach maximum value in the BST
• What is the Big-Oh for this operation?
Issues with BSTs

• Creating, updating can take order n time
  – Recursive reassignment of pointers may be needed
  – Process can be simple or complex

• Here is an example:
BST Node Removal - continued

- Node to be removed has one child.

In this case, node is cut from the tree and algorithm links single child (with its subtree) directly to the parent of the removed node.

Example. Remove 18 from a BST.
BST Node Removal - Continued

- Removing node (12):
- We will need to rework the tree structure
Replace 12 with 19. Notice, that only values are replaced, not nodes. Now we have two nodes with the same value.

Remove 19 from the left subtree.