Trees and Assignment 5

CS187 Data Structures
April 7, 2010
Chris Vigorito
Tree Traversal

- Three types: in-order, pre-order, post-order
- Differences in recursive implementation are just the ordering of 'visit', 'recur left', and 'recur right'
- Location of 'visit' specifies type
- Must be able to carry out all three type (will be on final)
Tree Traversals

• In-order: recurLeft(); visit(); recurRight();

• Pre-order: visit(); recurLeft(); recurRight();

• Post-order: recurLeft(); recurRight(); visit();
In-Order Traversal

recurLeft();
visit();
recurRight();
Pre-Order Traversal

visit();
recurLeft();
recurRight();
Post-Order Traversal

recurLeft();
recurRight();
visit();
Why Use Pre/Post?

- Tree represents algebraic expression
- In-order requires parentheses to disambiguate order of operations
- Pre-order and post-order produce prefix and postfix notations
- Easier to evaluate in a computer; no ambiguities
Why Use Pre/Post?

Infix: A*B+C
Prefix: *A+BC
Postfix: ABC+*
Finding Min and Max

• Minimum:
  • Start at root and keep visiting left child of current node
  • Minimum is first node with no left child

• Maximum:
  • Same procedure but with right child
Finding Min and Max
Storing Trees in Arrays

• Requires $O(2^D)$ space to store ($D=depth$), independent of actual number of nodes
• Deletion - whole subtrees may need to be moved
• Inefficient, but in certain cases might be desirable
• Simple arithmetic expressions for accessing children and parent given a node’s index
Storing Trees in Arrays

- Given a node’s index i
- Left child index: \(2i + 1\)
- Right Child index: \(2i + 2\)
- Parent: \((i - 1) / 2\)  \(\text{ <--integer division}\)
- Must store non-existent nodes as null
Storing Trees in Arrays

```
13
/  \
6   21
/    /
4    17
     /
     25
```

```
<table>
<thead>
<tr>
<th>13</th>
<th>6</th>
<th>21</th>
<th>4</th>
<th>null</th>
<th>17</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
```
Assignment 5

- Non-recursive version of merge sort
- Should be a little faster than recursive version (no method call overhead)
- Comparing/plot running times of 4 sorting algorithms
- Pretty lightweight, most code given