CSE 373

APRIL 17TH – TREE BALANCE AND AVL
ASSORTED MINUTIAE

• HW3 due Wednesday
  • Double check submissions
  • Use binary search for SADict
• Midterm text Friday
  • Review in Class on Wednesday
• Testing Advice
  • Empty and New are different edge cases
  • HW1 regrade
TODAY’S LECTURE

• Tree traversals
  • Memory Allocation
  • Traversal ordering

• Tree Balance
  • Improving on worst case time for trees
REVIEW

• Breadth First Search
  • Enqueue the root
  • While the queue has elements
    • Dequeue
    • Process
    • Enqueue children
  • How much memory does this take?
SEARCH MEMORY USE

• When does the queue have the most elements?
SEARCH MEMORY USE

• At the widest point in the traversal
  • How many elements is this?
SEARCH MEMORY USE

• Breadth First Search
  • In a perfect tree (where every row is complete) of size $n$, how many elements are in the last row?
    • $\text{ceiling}(N/2)$, this is important to know!
    • $O(n)$ memory usage!
SEARCH MEMORY USE

• What about depth first search?
  • When does the stack have the most elements on it?
• When does the stack have the most elements?
  • When it’s at the bottom
SEARCH MEMORY USE

• How many elements are in the stack in this worst case?
  • The height of the tree, \( O(n) \) if the tree is one-sided, but \( O(\log n) \) if the tree is balanced
  • We will discuss balance later
  • Classic exam question! Consider memory AND execution times
REVIEW

• Depth First Search
  • Iterative and Recursive options
  • Consider the recursive approach we discussed in class
REVIEW

• Ordering
  • What is the difference between these three implementations
    • Process; DFS(left); DFS(right)
    • DFS(left); Process; DFS(right)
    • DFS(left); DFS(right); Process
  • How does this impact the final output?
REVIEW

• Ordering
  • Three traversal types
    • Pre-order
    • In-order
    • Post-order
• Instruction (Parse) trees
PREORDER TRAVERSAL

Output:

Stack:

Output:
**PREORDER TRAVERSAL**

Add the root to the stack

<table>
<thead>
<tr>
<th>Stack:</th>
<th>+</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**PREORDER TRAVERSAL**

Process the node and then add children (right then left)

<table>
<thead>
<tr>
<th>Stack:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>+</td>
</tr>
</tbody>
</table>
**PREORDER TRAVERSAL**

Process the node and then add children (right then left)

**Stack:**
- 4
- 2
- 6
- 5
- 9
- 1
- 3
- 6

**Output:**
+X
Preorder Traversal

Process the node and then add children (right then left)

<table>
<thead>
<tr>
<th>Stack:</th>
<th>4</th>
<th>2</th>
<th>-</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>+X+</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PREORDER TRAVERSAL

Process the node and then add children (right then left)

<table>
<thead>
<tr>
<th>Stack:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>
PREORDER TRAVERSAL

Process the node and then add children (right then left)

<table>
<thead>
<tr>
<th>Stack:</th>
<th>-</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>+X+42</td>
<td></td>
</tr>
</tbody>
</table>
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack: 6 | 5 | +
Output: +X+42-
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack: 5 | +
Output: +X+42-6
**PREORDER TRAVERSAL**

Process the node and then add children (right then left)

**Stack:**
- 

**Output:**
+X+42-65
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack:  X | /
Output: +X+42-65+
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack: 9 | 1 | /
Output: +X+42-65+X
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack: 1 /
Output: +X+42-65+X9
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack: \(/\)
Output: \(+X+42-65+X91\)
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack: 3 | 6
Output: +X+42-65+X91/
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack: 6
Output: +X+42-65+X91/3
PREORDER TRAVERSAL

Process the node and then add children (right then left)

Stack: 

Output: +X+42-65+X91/36
What does this evaluate to?

Stack: 

Output: +X+42-65+X91/36
What does this evaluate to?

Stack:

Output: $+X+42-65+X\frac{91}{36}$
PREORDER TRAVERSAL

• Knowing the rule of preorder, is that string ambiguous?
  • +X+42-65+X91/36

• Given that preorder traversal is DFS with ordering:
  • Process, Left, Right

• What string results from postorder?
  • Left Right Process?
POSTORDER TRAVERSAL

```
+     +
|     |
X-----X
/ 
+ - / 
| | |
4 2 5
```

4 + 2 + 6 
= 12 

9 * 3 = 27 

12 + 27 = 39
POSTORDER TRAVERSAL

• Pre-order
  • \(+X+42-65+X91/36\)

• Post-order
  • \(42+65-X91X36/++\)
POSTORDER TRAVERSAL

• Pre-order (Polish Notation)
  • +X+42-65+X91/36

• Post-order (Reverse Polish Notation)
  • 42+65-X91X36/++
POSTORDER TRAVERSAL

- **Pre-order** (Polish Notation)
  - +X+42-65+X91/36
- **Post-order** (Reverse Polish Notation)
  - 42+65-X91X36/++
- These are unambiguous strings
**POSTORDER TRAVERSAL**

- **Pre-order** (Polish Notation)
  - $+X+42-65+X91/36$
- **Post-order** (Reverse Polish Notation)
  - $42+65-X91X36/++$
- These are unambiguous strings
- What about the final ordering?
  - Left, Process, Right?
IN-ORDER TRAVERSAL
IN-ORDER TRAVERSAL

• In-order
  • $4 + 2 \times 6 - 5 + 9 \times 1 + 3/6$

• What is the problem here?
IN-ORDER TRAVERSAL

```
(4 +
  2 -
  6 /
)(9 +
  1 3
)(5 X
  6)
```

Diagram:
![In-Order Traversal Diagram](https://via.placeholder.com/150)
TRAVERSALS

• In-order
  • $4 + 2 \times 6 - 5 + 9 \times 1 + \frac{3}{6}$

• What is the problem here?
  • There are multiple trees!

• In order returns the left-to-right sorted order
  • In-order traversal of a BST is sorted result
IN-ORDER TRAVERSAL

4 + 2 * 6 - 5 + 9 * 1 + 3 / 6
TRAVERSALS

• Pre-order and post-order are unambiguous, why?
  • They can only represent one tree because we can distinguish parents from leaves
  • Parents are operators and leaves are numbers
  • If they are all numbers, the multiple trees represent the multiple ways of storing the data
BALANCE AND HEIGHT

• If the same data can be represented multiple ways, what is best?
BALANCE AND HEIGHT
BALANCE AND HEIGHT

• Height is key for how fast functions on our tree are!
  • If we can structure the same data two different ways, we want to choose the better one.
  • Balanced is better for BSTs
  • Can we enforce balance?
BALANCE AND HEIGHT

• Balance
  • How can we define balance?
  • $\text{Abs}(\text{height(left)} - \text{height(right)})$
  • If the heights of the left and right trees are balanced, the tree is balanced.
  • Anything wrong with this?
BALANCE AND HEIGHT
BALANCE AND HEIGHT

• Not enough for the root to be balanced!
• All nodes must be balanced!
• Ideally, our “balance” property will say:
  • For all nodes in the tree, height(left) = height(right)
  • What is the problem with this?
  • Not always enforceable!
BALANCE AND HEIGHT

• Consider adding an element to a tree.
  • When the tree is empty, it is balanced
• We add one element
  • Height(left) = height(right) = 0
• Add another element
  • Oh no! There is no way to enforce balance!
BALANCE AND HEIGHT

• New property
  • If \( \text{Abs}(\text{height}(\text{left}) - \text{height}(\text{right})) \) is balance
  • We can only enforce if this is \( \leq 1 \)
  • That is, the height left and right subtrees can differ by at most one
  • Still must preserve this for every node!

• This is the AVL property

• AVL Trees are Binary Search Trees that have the AVL property
  • They have worst case \( O(\log n) \) find!
NEXT CLASS

• AVL Trees
  • Prove that they have $O(\log n)$ height
  • Come up with implementations for insert and delete
  • Want to get $O(1)$ time for these, ideally