CSC190 Notes

Aman Bhargva

January-April 2019

Contents

1 Introduction and Course Information

2 Efficiency and Complexity

There are two things that effect efficiency of a computer algorithm.

- Space Complexity = cost in terms of memory
- Time Complexity = cost in terms of steps

3 Ideal vs. Real Computers

Ideal	Real
Automaton + Workbook	Microprocessor + Memory
Mathematical Algorithms	Programs (Sea of Functions)
Abstract Data Types	Data Structure

`Note: ADTs are Abstract Data Types`

4 List ADT implementation in C

- An array with length L and end index 'end'

5 Stacks and Queues

5.1 Stacks

- First in, last out
- push and pop

5.2 Queues

- First in, first out
- enqueue, dequeue

6 Trees

6.1 Types of Traversal:

6.1.1 Level Order Traversal:

In which you read (1) right to left (2) top to bottom.

6.1.2 Breadth Order Traversal:

In which you put stuff on the same level of the tree at the same indentation level, and the children of nodes are indented underneath it.

6.2 Binary Trees:

6.3 Mathai's Weird Definition for Binary Trees ¡==¿ Normal Trees

The rule: The first child of a node goes as the left child in the binary tree. All of its siblings are added as RIGHT children of that first child. Etc.

6.4 Binary Search Trees:

Think about the binary search algorithm. The rule is simple: Start with the first node. If the thing you want to add is greater than that node, you add it to the RIGHT child tree. Otherwise, you add it to the LEFT child tree. The rule is recursive and super elegant.

6.4.1 Advantages/Disadvantages of Binary Search Trees:

1. Advantage: Will (generally) change search from O(n) to O(log(n)) because you only need to visit every LEVEL instead of every NODE (#levels = log(#nodes))
2. Disadvantage: If it's not a balanced tree, you could end up with a BST that has #levels = #nodes

6.5 Self-Balancing BST's:

The general way we balance binary search trees is with **rotations**

Level of imbalance: calculated as nR (number of levels on the right side of the node) minus nL.

Printing Binary Trees in Order: When you print the left subtree first, then the node, then the right subtree (recursively)

6.5.1 Types of Rotation:

1. Left
2. Right
3. Major Left / Minor Right
4. Major Right / Minor Left

6.5.2 Right Rotation:

The left child becomes the head. The left child's parent (the root) becomes the right child, and the left child of the left child becomes maintains its relationship with the left child of the root. The original left child's RIGHT child becomes the left child of the original root.

When to do this? whenever nL - nR $\geq$ 2...?

6.5.3 Left Rotation:

The right child becomes the head. The right child's parent (the root) becomes the left child, and the right vhild of the right child maintains its relationship with the right child of the original root. The original right child's LEFT child becomes the right child of the original root.

6.5.4 Major Right / Minor Left:

When to do this: Whenever the balance on the root and the balance on a subtree is opposite, and the absolute balance of the root is greater than or equal to 2. Tl;dr: You want the subtrees to have the same SIGN of balance as the root node before you do big switches. Condition for doing compelex switches is: balance (root) ¿=2, sign balance (sub-node) doesn't equal the sign balance (root node)

6.5.5 Summary of Tree Rotations:

1. For easy, child-free lines to the right and left (root balance = +-2, child banace = +-1 with same sign), use the corresponding rotation to make them triangle trees.

2. For bent ones with root balance = +-2 and child balance +-1 of opposite sign), use minor rotation to get back to case 1 and do a major rotation on the root.