CS 008
Lecture notes
3/28/24

1 Outline

- Trees
- Binary Trees
- Git

2 Trees

Trees are a series of nodes where one node is the root node to the tree. Every other node apart from the root node will have a parent node. All nodes are connected in some way and the connections between the nodes go in one direction.

Definitions:

- **Parent**: A node that points to another node.
- **Child**: A node that is pointed to by a parent node.
- **Leaf**: These are nodes that do not point to another node.
- **Height & Depth**: These are the height and depth of the tree which represent the amount of nodes followed to reach the bottom for the height or a certain node for the depth. The root node is not included in the height and depth of a tree.
- **Subtree**: A portion of the tree can be also be a tree (ignoring the parent nodes of the root of the subtree to create the subtree).

2.1 Taxonomy

There are many different types of trees that exist. Some of them include:

- Binary Trees
- N-ary trees
- Heaps

3 Binary Trees

A binary tree is a tree where the nodes in the tree would have 2 or fewer child nodes. There are two adjectives that can be used to describe a binary tree.

3.1 Full binary trees

A full binary tree is a binary tree where every leaf is at the same path and every non-leaf has two children.

3.2 Complete binary trees

All leaves have the same or atleast within one depth from each other in a complete binary tree. All non-leaves of a complete binary tree (with the exception of maybe one non-leaf) have two children. Any level of the tree that is not full has nodes that are arranged in the left most spots. With this, a complete binary tree would have a right and a left child node for parent nodes.

3.3 Implementation

A binary tree can be implemented as either an array or a linked list of nodes. The array implementation of the binary tree is simply just an ordered array of the nodes by number (so if we had a tree where the root is considered 1 and the children are 2 and 3, the array representation would be 1, 2, 3).

Figure 1:

/ Tree example

For the array implementation, the time complexity of getting a node or getting a root of the tree would be O(1). With the linked list implementation, looking for a specific node in the linked list implementation would be O(n). Looking for the value of the root is still O(1) even in the linked list implemenation.

Traversing the tree to look for a value is of the time complexity $O(\log_2(n))$, since the most amount of pointers is actually related to the depth of the value in the tree. Traversal of a tree is not linear but it is also not a constant time since the time complexity is still dependant on the amount of nodes. However, since the amount of nodes grows exponentially larger than the worst case number of operations.

3.4 Traversals

A traversal is bidirectional, it is the idea of following the pointers in the nodes. There are three algorithms for traversing a tree consisting of:

- Pre-order
- In-order
- Post-order