Study.com

Trees in Data Structures: Methods & Examples

Did you know there is a more efficient way to store data in your code and reduce search times by half? Dig into the world of trees, an abstract data type that will make this possible.

What is a Tree?

Trees are abstract data structures, used to manage data in a hierarchical way, making data retrieving much more efficient than other data structure methods. A tree is a collection of **nodes** that are linearly connected and don't have any cyclic relations as shown in Figure 1. These nodes contain information and are usually arranged in a **key-value** structure. The key serves as an identifier for the data, and value is the actual data that you want to store.

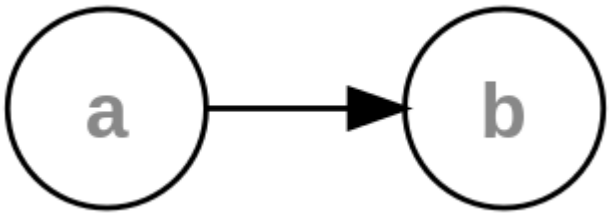

Figure 1: Basic Definition of a Tree

Tree Terminology

When working with trees, there are some terms that are worth becoming familiar with:

- **Root:** The initial node of the tree, where all the operations start.
- **Node:** Any single item in the tree, usually a key-value item.
- **Child:** A node directly connected to another node when moving away from the root.
- **Parent:** The converse notion of a child.

Types of Trees

There are different types of trees that you can work with. The most common type of tree is a **binary tree**. This type of tree is so named because each parent node can only have two children. Other types of trees consist of a parent node being able to have more than 2 children.

The deciding factor of which tree type to use is **performance**. Since trees are data structures, performance is measured in terms of inserting and retrieving data. In this case, the binary tree is the most efficient when it comes to these operations (see Figure 2), and therefore the most used.

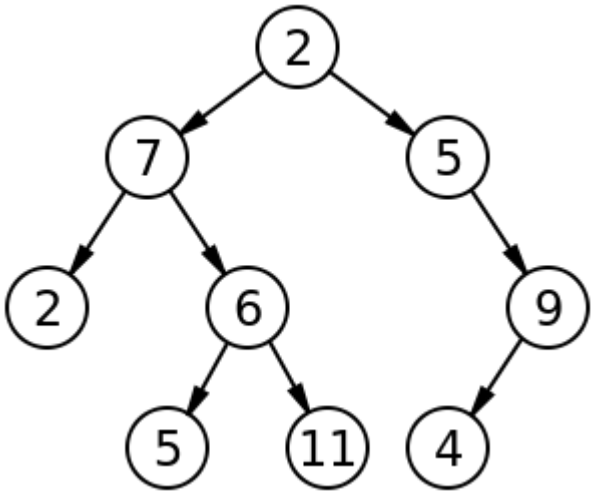

Figure 2: Standard Binary Tree

Common Operations

There are multiple operations that can be performed on trees. For the purposes of this lesson, we will focus on data insertion and search. Other methods that won't be covered include enumerating items and editing originally stored information.

Insert

To start inserting data into the tree, let's assume that you are starting an empty tree and you have an array of numbers (5,3,7,6,2,1,8,4). To start, the initial number will serve as the root for the tree. In this particular case, each number is the key and the value, so the example can be simplified.

Once the tree is created, insertion becomes easy. The question you ask is: Is the next number more or less than the number in the root? That will determine if the number will be set as a left or right child:

```
1  public Node insertNode (Node node, int data){
2   if (node == null){
3    node = new Node(data);
4    return node;
5   }
6   if (data <= node.data){
7    node.leftChild = insertNode(node.leftChild, data);
8   } else if (data > node.data){
9    node.rightChild = insertNode(node.rightChild, data);
10  }
11  return node;
12 }
```

You then repeat the same process with each number. If the child node has data stored already, then that child node becomes the parent node until you have the complete tree as shown in Figure 3.

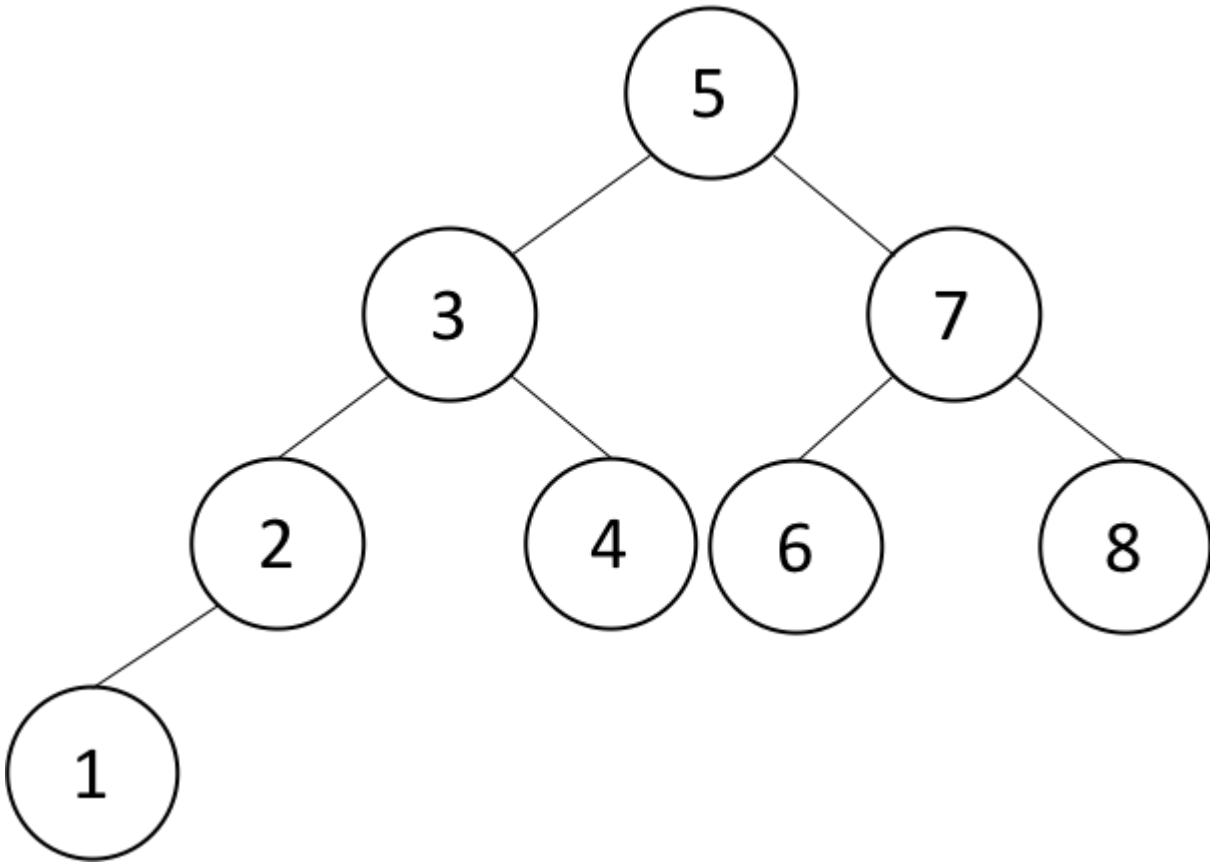

Figure 3: Binary tree after all items were inserted.

Trees in Data Structures: Methods & Examples Related Study Materials