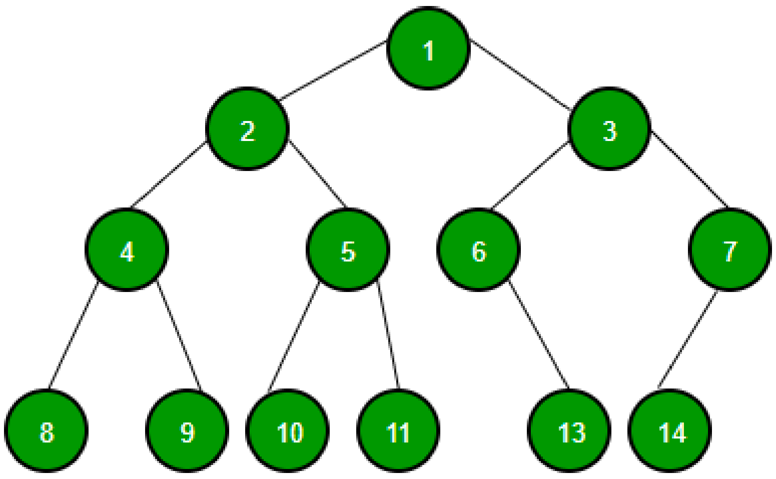
CS 260
Binary Trees

Binary Trees

Binary trees are the next logical step after linked lists. While a linked list is a one-dimensional structure, a binary tree is two dimensional. Instead of having links, as in a linked list, trees are normally considered to have nodes. Instead of a previous and a next, they have parents and children, as shown in the following example. In a binary tree, a node can have zero, one, or two children.



In this image the **root** (the single node in the top level with no parent) is labeled with the number 1. It has two children, the node labeled 2 is its **left child** and the node labeled 3 is its **right child**. Node 1 is the **parent** of nodes 2 and 3. The connection between two nodes is called an **edge**.

The nodes 2 and 3 are both **inner** nodes since they have children and are not the root. Nodes 8 through 14 are **leaf** nodes since they do not have children.

The nodes 1, 2, 4, and 8 make up a **path** from the root to a leaf. This path is of **length** 4. Since the longest paths in the tree are of length 4, this tree has a **height** of 4.

The node 4 and its children 8 and 9 can be considered a **subtree** with three nodes. The node 4 is the root of this subtree and its height is two. Including the node 2, with its children 4 and 5 and their children 8 through 11, creates a subtree of height 3 with node 2 as its root.

The image above image is an example of a **binary** tree since a node has at most two children.

This course will consider three different types of binary trees: binary parse trees, binary search trees, and heaps. More information on each of them can be found in other documents.

Implementation

Nodes are similar to links in that they can be implemented as either structs or objects. They typically contain a value and references or pointers to their parent and left and right children. Normally there is a constructor (init) method that creates a node by passing in its value and initializing parent and child references to nullptr/None/null. The implementation can either use getters and setters for the variables in the node or the variables can be public and with direct access. In the examples in this course, the Node is implemented as a class with setters and getters as shown here.

```
    // definition of Node
class Node
    private:
        int value   // values can be any type of data
        Node left
        Node right
        Node parent // optional depending on use

    public:
          // constructor to build new node
        Node(int value)
                    this.left = this.right = this.parent = nullptr
           this.value = value

          // setters
        void setLeft(Node left)
          this.left = left
        void setRight(Node right)
          this.right = right
        void setParent(Node parent)
          this.parent = parent

          // getters
        Node getLeft()
          return this.left
        Node getRight()
          return this.right
        Node getParent()
          return this.parent
        int getValue()
          return this.value
```

The tree itself is represented by a class that has a reference of type Node named root. This is initialized to nullptr/None/null when creating the tree.

```
    // definition of binary Tree
class Tree
    private:
        Node root

    public:
          // constructor and destructor
        Tree()
          this.root = nullptr
        ~Tree() // delete all nodes in the tree

           // methods depend on use of tree
```