1 Trees

1.1 Binary Trees

A binary tree is made up of nodes with left and right children. For these notes, we can assume the structure is implemented via something like the following, very minimal, `Node` class:

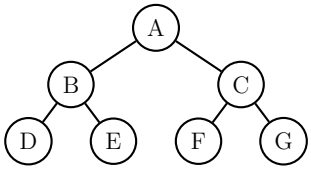
```
class Node:

   def __init__(self, value = None):
      self.left = None
      self.right = None
      self.value = value
```

Binary Tree Traversal

Building a tree:

```
# Creating the tree nodes
root = Node(A)
root.left = Node(B)
root.right = Node(C)
root.left.left = Node(D)
root.left.right = Node(E)
root.right.left = Node(F)
root.left.right = Node(G)
```



Here are three common algorithms for traversing a tree:

- **In-Order**: Traverse left sub-tree, to the root, to the right sub-tree

  Result: D,B,E,A,F,C,G

  Code:

```
# Function to perform inorder traversal
def inorderTraversal(root):
   # Base case: if the current node is null, return
   if root is None:
      return None
   # Recur on the left subtree
   inorderTraversal(root.left)
   # Do whatever you want with the visited nodes...
   print(root.data)
   # Recur on the right subtree
   inorderTraversal(root.right)
```

- **Pre-order**: Traverse from the root, to the left sub-tree, to the right sub-tree

  Result: A,B,D,E,C,F,G

  Code:

```
def preorderTraversal(root):
   if root is None:
      return None
   print(root.data)
   preorderTraversal(root.left)
   preorderTraversal(root.right)
```

- **Post-order**: Traverse from the left sub-tree, to the right sub-tree, to the root

  Result: D,E,B,F,G,C,A

  Code:

```
def postorderTraversal(root):
if root is None:
   return None
postorderTraversal(root.left)
postorderTraversal(root.right)
print(root.data)
```