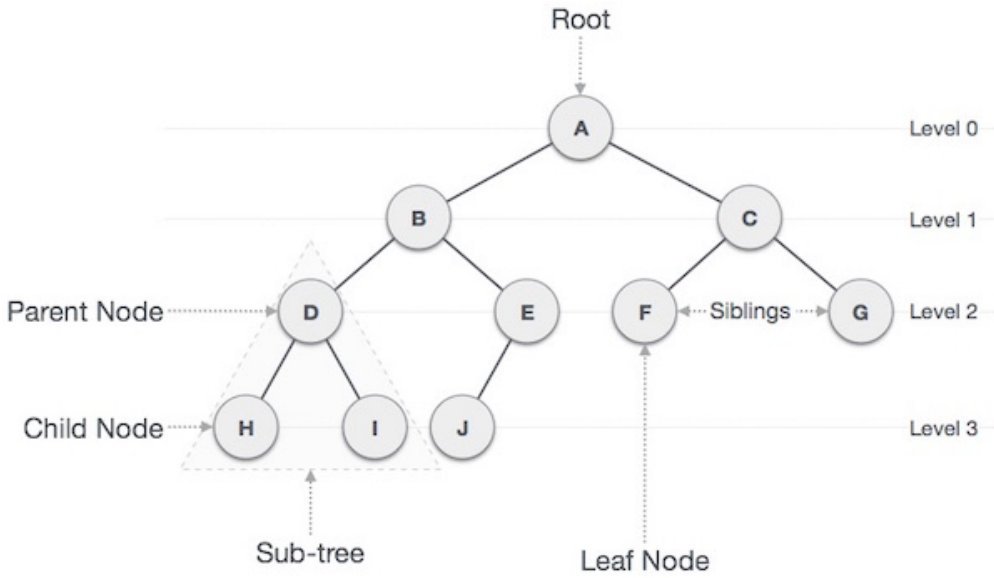
Binary Tree

1. Introduction

Binary Tree is a tree that each node has at most 2 children.

- The two children are typically named `left child` and `right child` .
- The top most node in the tree is the `root node` .
- All nodes have one parent except root node, which has no parent.



https://www.tutorialspoint.com/data_structures_algorithms/images/binary_tree.jpg

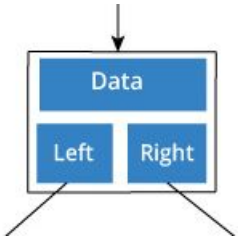
Some Important Terms

- `Levels` : Level of a node represents the generation of a node.
  - If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
- `keys` : Key represents a value of a node based on which a search operation is to be carried out for a node.
- `Traversing` : Traversing means passing through nodes in a specific order.

Node Object

Each **node** in binary tree contains following parts:

- data
- pointer to left child
- pointer to right child



https://cdn.programiz.com/sites/tutorial2program/files/tree-concept.jpg

Exercise 1

Implement a `Node` class which has instance attributes `data`, `left` and `right`.

- Initialize `data`, `left` and `right` in initializer. Both `left` and `right` has default value of `None`.
- Implement `__str__()` method to return string with format `data(left.data,right.data)`

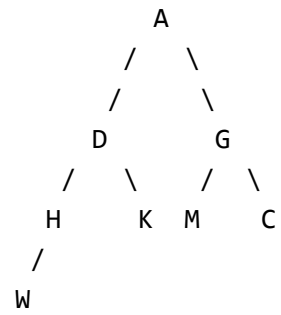
In [45]:

```
class Node:

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

    def __str__(self):
        return '{}({},{})'.format(self.data,
                                  self.left.data if self.left else '',
                                  self.right.data if self.right else '')

if __name__ == '__main__':
    left = Node(5)
    right = Node(15)
    n1 = Node(10, left, right)
    print(n1)
```

```
10(5,15)
```

Exercise 2

Considering following binary tree,



How do you use `Node` class to construct above tree with root node pointed by variable `root`?

In [46]:

```
root = Node('A', Node('D', Node('H',Node('W')),Node('K')), Node('G', Node('M'), Node('C
```

Exercise 3

Implement a recursive function `_print_tree()`, which prints a tree layer by layer from the top.

- It receives a list of nodes as input.
- It prints the current layer of nodes, and continue to print next layer of nodes until it finish printing all nodes.

In [1]:

```
def _print_tree(node_list):
    # Stop recursion if the list is empty
    if len(node_list)==0:
        return
    # define a list to collect nodes in next layer
    next_layer = []
    while node_list:
        node = node_list.pop()
        print(node, end=' ')
        if node.left:
            next_layer.insert(0, node.left)
        if node.right:
            next_layer.insert(0, node.right)
    print()
    _print_tree(next_layer)
```

In [48]:

```
_print_tree([root])
```

```
A(D,G)
D(H,K) G(M,C)
H(W,) K(,) M(,) C(,)
W(,)
```

Exercise 4

Defines a `BinaryTree` class.

- It initialize `root` instance variable with an input parameter. The input parameter has a default value of `None`.
- Implement recursive function `_print_tree()` as an instance method of `BinaryTree`.
- Use `_print_tree()` to implement another instance method `print_tree()`, which prints nodes in each level, starting from root level.

In [52]:

```
class BinaryTree:

    def __init__(self, root=None):
        self.root = root

    def print_tree(self):
        self._print_tree([self.root])

    def _print_tree(self, node_list):
        # Convert node_list to a list if it is not
        if not isinstance(node_list, list):
            node_list = [node_list]
        # Stop recursion if the list is empty
        if not node_list:
            return
        # define a list to collect nodes in next layer
        next_layer = []
        while node_list:
            node = node_list.pop()
            print(node, end=' ')
            if node.left:
                next_layer.insert(0, node.left)
            if node.right:
                next_layer.insert(0, node.right)
        print()
        self._print_tree(next_layer)


if __name__ == '__main__':
    root = Node(27, Node(14, Node(10),Node(19)), Node(35, Node(31), Node(42)))
    tree = BinaryTree(root)
    tree.print_tree()
```

```
27(14,35)
14(10,19) 35(31,42)
10(,) 19(,) 31(,) 42(,)
```

2. Binary Tree Traversals

Traversal is the process of visiting all nodes in a tree in some order.

- While visiting each node, we perform some actions on the node, e.g. print value of the node

There are 3 common orders for traversal, `pre-order` , `post-order` and `in-order` .

Pre-order

In pre-order traversal, we follow the order of `node-left-right` :

- visit a given node first
- visit its left child
- followed by visiting its right child

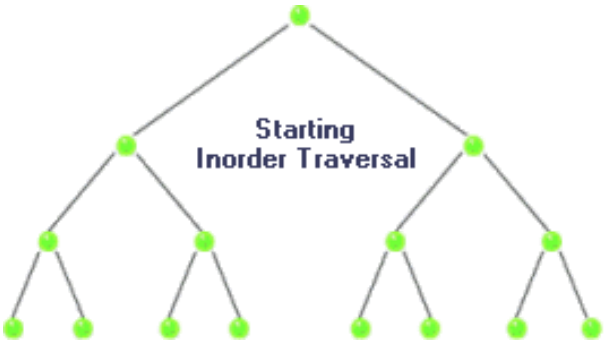
Translate it into recursive function:

```
def _preorder(node):
    if node is not None:
        visitNode(node)
        _preorder(node.left_child)
        _preorder(node.right_child)
```

In-order

In in-order traversal, we follow the order of `left-node-right` :

- visit left child of a given node
- visit the given node
- finally right child of the given node



Translate it into recursive function:

```
def _inorder(node):
    if node is not None:
        _inorder(node.left_child)
        visitNode(node)
        _inorder(node.right_child)
```

Post-order

In post-order traversal, we follow the order of `left-right-node` :

- visit left child of a given node,
- right child of a given node,
- visit given node itself

Translate it into recursive function:

```
def _postorder(node):
    if node is not None:
        _postorder(node.left_child)
        _postorder(node.right_child)
        visitNode(node)
```

3. Traversal Example

Considering the same binary tree in above example,

```
          A
        /   \
       /     \
      D       G
    /   \    /  \
   H     K  M    C
  /
 W
```

Question:

Assume of action of visiting each node is to print the node value, what is the value printed for `pre-order`, `in-order` and `poster-oder` respectively?

```
Pre-order : A D H W K G M C
In-order: W H D K A M G C
Post-order : W H K D M C G A
```

Exercise: Pre-Order

With pre-order traversal, following tree will traverse nodes in such order: A D H W K G M C

Implement a class `BinaryTree2` which inherits from `BinaryTree`.

- Implement its `inorder()` instance method which prints nodes using `in-order` traversal.
- Make use of the recursive function `_inorder()`.

In [54]:

```
class BinaryTree2(BinaryTree):

    def preorder(self):
        self._preorder(self.root)

    def _preorder(self, node=None):
        if node is not None:
            print(node.data, end=' ')
            self._preorder(node.left)
            self._preorder(node.right)
```

In [55]:

```
if __name__ == '__main__':
    root = Node('A', Node('D', Node('H',Node('W')),Node('K')), Node('G', Node('M'), Noc
    t = BinaryTree2(root)
    t.preorder()
```

```
A D H W K G M C
```

Exercise: In-Order

With in-order traversal, following tree will traverse nodes in such order: W H D K A M G C

Implement a class `BinaryTree2` which inherits from `BinaryTree` .

- Implement its `inorder()` instance method which prints nodes using `in-order` traversal.
- Make use of the recursive function `_inorder()` .

In [58]:

```
class BinaryTree2(BinaryTree):

    def inorder(self):
        self._inorder(self.root)

    def _inorder(self, node=None):
        if node is not None:
            self._inorder(node.left)
            print(node.data, end=' ')
            self._inorder(node.right)

```

In [59]:

```
if __name__ == '__main__':
    root = Node('A', Node('D', Node('H',Node('W')),Node('K')), Node('G', Node('M'), Noc
    t = BinaryTree2(root)
    t.inorder()
```

```
W H D K A M G C
```

Exercise: Post-Order

With post-order traversal, following tree will traverse nodes in such order: W H K D M C G A

Implement a class `BinaryTree2` which inherits from `BinaryTree`.

- Implement its `inorder()` method which prints nodes using `in-order` traversal.
- Make use of the recursive function `_inorder()`.

In [60]:

```
class BinaryTree2(BinaryTree):

    def postorder(self):
        self._postorder(self.root)

    def _postorder(self, node=None):
        if node is not None:
            self._postorder(node.left)
            self._postorder(node.right)
            print(node.data, end=' ')

```

```
W H K D M C G A
```

In [61]:

```
if __name__ == '__main__':
    root = Node('A', Node('D', Node('H',Node('W')),Node('K')), Node('G', Node('M'), Noc
    t = BinaryTree2(root)
    t.postorder()
```

```
W H K D M C G A
```

Reference

Traversal in-order, pre-order, post-order

- https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/BinaryTreeTraversal.html (https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/BinaryTreeTraversal.html)
- https://www.programiz.com/dsa/tree-traversal (https://www.programiz.com/dsa/tree-traversal)
- https://www.tutorialspoint.com/data_structures_algorithms/tree_traversal.htm (https://www.tutorialspoint.com/data_structures_algorithms/tree_traversal.htm)
- https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/ (https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/)