**Name: Drishti Bhatia** **Roll Number: 07**
**SYMCA** **Block Chain Lab**

# Practical No: 04

## Aim: Install Geth Ethereum Node (Show installation steps also). Create a Genesis block and a private chain. Use geth commands to create user accounts, mine, transact etc.

**Theory:**

A parent node has two child nodes: the left child and right child. Hashing, routing datafor network traffic, data compression, preparing binary heaps, and binary search trees are some of the applications that use a binary tree.

❖ **Basic Operations:**

Following are the basic operations of a tree –

- **Create Root:** we just create a Node class and assign a value to the node. This becomes a tree with only a root node.
- **Search:** Searches an element in a tree.
- **Insert:** (Inserts an element in a tree.) To insert into a tree, we use the same node class created above and add an insert class to it. The insert class compares the value of the nodeto the parent node and decides to add it as a left node or a right node. Finally, the PrintTreeclass is used to print the tree.
- **Traversing a Tree:** The tree can be traversed by deciding on a sequence to visit each node.As we can clearly see we can start at a node then visit the left sub-tree first and right sub-tree next. We can also visit the right sub-tree first and left sub-tree next. Accordingly, thereare different names for these tree traversal methods. Traversal is a process to visit all the nodes of a tree and may print their values too. Because all nodes are connected via edges,we always start from the root node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree:
  1. **Pre-order Traversal:** In this traversal method, the root node is visited first, then theleft sub-tree and finally the right sub-tree. We use the Node class to create placeholders for the root node as well as the left and right nodes. Then, we create an insert function to add data to the tree. Finally, the Pre-order traversal logic is implemented by creating an empty list and adding the root node first followed by theleft node. At last, the right node is added to complete the Pre-order traversal.
  2. **In-order Traversal:** In this traversal method, the left sub-tree is visited first, then theroot and later the right sub-tree. We should always remember that every node may represent a subtree itself. We use the Node class to create placeholders for the root node as well as the left and right nodes. Then, we create an insert function to add datato the tree. Finally, the In-order traversal logic is implemented by creating an empty list and adding the left node first followed by the root or parent

node. At last, the leftnode is added to complete the In-order traversal.

3. **Post-order Traversal:** In this traversal method, the root node is visited last, hence thename. First, we traverse the left sub-tree, then the right sub-tree and finally the root node. We use the Node class to create place holders for the root node as well as the left and right nodes. Then, we create an insert function to add data to the tree. Finally,the Post-order traversal logic is implemented by creating an empty list and adding theleft node first followed by the right node. At last, the root or parent node is added to complete the post-order traversal.

**Delete** – Given a binary tree, delete a node from it by making sure that tree shrinks from the bottom. This is different from BST deletion. Here we do not have any order among elements, so we replace it with the last element.

**Algorithm:**

1. Starting at the root, find the deepest and rightmost node in the binary tree and node which we want to delete.
2. Replace the deepest rightmost node's data with the node to be deleted.
3. Then delete the deepest rightmost node.

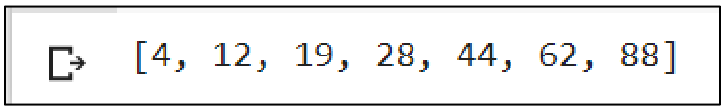
**Code:**

```
class Node:
   def __init__(self, data):
      self.left = None
      self.right = None
      self.data = data
   def insert(self, data):
# Compare the new value with the parent node
      if self.data:
         if data < self.data:
            if self.left is None:
               self.left = Node(data)
            else:
               self.left.insert(data)
         elif data > self.data:
               if self.right is None:
                  self.right = Node(data)
               else:
                  self.right.insert(data)
      else:
         self.data = data


   # Print the tree
   def PrintTree(self):
      if self.left:
         self.left.PrintTree()
      print( self.data),
      if self.right:
         self.right.PrintTree()

# Inorder traversal
# Left -> Root -> Right
   def inorderTraversal(self, root):
      res = []
      if root:
         res = self.inorderTraversal(root.left)
         res.append(root.data)
         res = res + self.inorderTraversal(root.right)
      return res
root = Node(88)
root.insert(62)
root.insert(44)
root.insert(28)
root.insert(19)
root.insert(12)
root.insert(4)
print(root.inorderTraversal(root))
```

**Output:**



```
[4, 12, 19, 28, 44, 62, 88]
```

**Conclusion:** We have successfully we have successfully implemented Binary Tree and showed all operations(Insert, Delete, Traversals, Display).