**SOURCE CODE:-**

class BinaryTreeNode {

constructor(key, value = key, parent = null) {

this.key = key;

this.value = value;

this.parent = parent;

this.left = null;

this.right = null;

}

get isLeaf() {

return this.left === null && this.right === null;

}

get hasChildren() {

return !this.isLeaf;

}

}

class BinaryTree {

constructor(key, value = key) {

this.root = new BinaryTreeNode(key, value);

}

\*inOrderTraversal(node = this.root) {

if (node.left) yield\* this.inOrderTraversal(node.left);

yield node;

if (node.right) yield\* this.inOrderTraversal(node.right);

}

\*postOrderTraversal(node = this.root) {

if (node.left) yield\* this.postOrderTraversal(node.left);

if (node.right) yield\* this.postOrderTraversal(node.right);

yield node;

}

\*preOrderTraversal(node = this.root) {

yield node;

if (node.left) yield\* this.preOrderTraversal(node.left);

if (node.right) yield\* this.preOrderTraversal(node.right);

}

insert(

parentNodeKey,

key,

value = key,

{ left, right } = { left: true, right: true }

) {

for (let node of this.preOrderTraversal()) {

if (node.key === parentNodeKey) {

const canInsertLeft = left && node.left === null;

const canInsertRight = right && node.right === null;

if (!canInsertLeft && !canInsertRight) return false;

if (canInsertLeft) {

node.left = new BinaryTreeNode(key, value, node);

return true;

}

if (canInsertRight) {

node.right = new BinaryTreeNode(key, value, node);

return true;

}

}

}

return false;

}

remove(key) {

for (let node of this.preOrderTraversal()) {

if (node.left.key === key) {

node.left = null;

return true;

}

if (node.right.key === key) {

node.right = null;

return true;

}

}

return false;

}

find(key) {

for (let node of this.preOrderTraversal()) {

if (node.key === key) return node;

}

return undefined;

}

}

const tree = new BinaryTree(1, 'AB');

tree.insert(1, 11, 'AC');

tree.insert(1, 12, 'BC');

tree.insert(12, 121, 'BG', { right: true });

[...tree.preOrderTraversal()].map(x => x.value);

[...tree.inOrderTraversal()].map(x => x.value);

tree.root.value;

tree.root.hasChildren;

tree.find(12).isLeaf;

tree.find(121).isLeaf;

tree.find(121).parent.value;

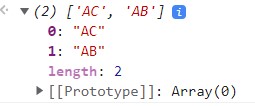
tree.find(12).left;

tree.find(12).right.value;

tree.remove(12);

[...tree.postOrderTraversal()].map(x => x.value);

**OUTPUT:-**

****