UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE)

WINTER DOMAIN CAMP

Date : 27/12/2024

Submitted to:

Faculty name: Er.Rajni Devi

Submitted by:

Name: Akshi Datta

UID: 22BCS15369

Section: IOT-620

DAY 6(27/12/24)

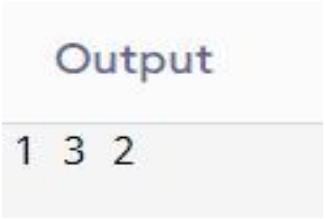
PROBLEM 1

Objective: Given the root of a binary tree, return the inorder traversal of its nodes' values.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
struct TreeNode {
int val;
TreeNode *left, *right;
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
void inorder(TreeNode* root, vector<int>& res) {
if (!root) return;
inorder(root->left, res);
res.push_back(root->val);
inorder(root->right, res);
}
vector<int> inorderTraversal(TreeNode* root) {
vector<int> res;
inorder(root, res);
return res;
}
int main() {
TreeNode* root = new TreeNode(1);
root->right = new TreeNode(2);
root->right->left = new TreeNode(3);
vector<int> result = inorderTraversal(root);
for (int val : result) cout << val << " ";
return 0;
}
```

Output:



PROBLEM 2

Objective: Given the root of a complete binary tree, return the number of the nodes in the tree.

Code:

```
#include <iostream>
#include <cmath>
using namespace std;
struct TreeNode {
int val;
TreeNode *left, *right;
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
int getHeight(TreeNode* root) {
int height = 0;
while (root) {
height++;
root = root->left;
}
return height;
}
int countNodes(TreeNode* root) {
if (!root) return 0;
int leftHeight = getHeight(root->left);
int rightHeight = getHeight(root->right);
if (leftHeight == rightHeight)
return (1 << leftHeight) + countNodes(root->right);
else
return (1 << rightHeight) + countNodes(root->left);
}
int main() {
TreeNode* root = new TreeNode(1);
root->left = new TreeNode(2);
root->right = new TreeNode(3);
root->left->left = new TreeNode(4);
root->left->right = new TreeNode(5);
root->right->left = new TreeNode(6);
cout << countNodes(root);
return 0;
}
```

Output:

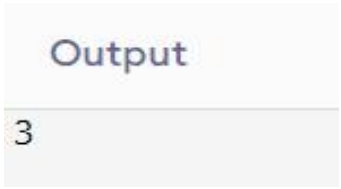
PROBLEM 3

Objective: A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Code:

```
#include <iostream>
#include <algorithm>
using namespace std;
struct TreeNode {
int val;
TreeNode *left, *right;
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
int maxDepth(TreeNode* root) {
if (!root) return 0;
return 1 + max(maxDepth(root->left), maxDepth(root->right));
}
int main() {
TreeNode* root = new TreeNode(3);
root->left = new TreeNode(9);
root->right = new TreeNode(20);
root->right->left = new TreeNode(15);
root->right->right = new TreeNode(7);
cout << maxDepth(root);
return 0;
}
```

Output:



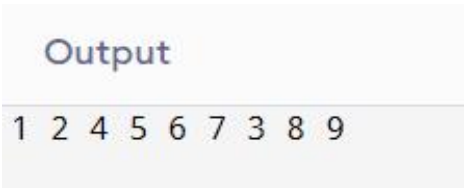
PROBLEM 4

Objective: Given the root of a binary tree, return the preorder traversal of its nodes' values.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
struct TreeNode {
int val;
TreeNode *left, *right;
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
void preorder(TreeNode* root, vector<int>& res) {
if (!root) return;
res.push_back(root->val);
preorder(root->left, res);
preorder(root->right, res);
}
vector<int> preorderTraversal(TreeNode* root) {
vector<int> res;
preorder(root, res);
return res;
}
int main() {
TreeNode* root = new TreeNode(1);
root->left = new TreeNode(2);
root->right = new TreeNode(3);
root->left->left = new TreeNode(4);
root->left->right = new TreeNode(5);
root->right->right = new TreeNode(8);
root->left->right->left = new TreeNode(6);
root->left->right->right = new TreeNode(7);
root->right->right->left = new TreeNode(9);
vector<int> result = preorderTraversal(root);
for (int val : result) cout << val << " ";
return 0;
}
```

Output:



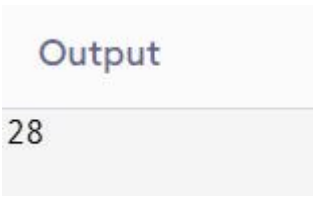
PROBLEM 5

Objective: Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.

Code:

```
#include <iostream>
using namespace std;
struct TreeNode {
int val;
TreeNode *left, *right;
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
int sumOfNodes(TreeNode* root) {
if (!root) return 0;
return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
}
int main() {
TreeNode* root = new TreeNode(5);
root->left = new TreeNode(2);
root->right = new TreeNode(6);
root->left->left = new TreeNode(1);
root->left->right = new TreeNode(3);
root->right->left = new TreeNode(4);
root->right->right = new TreeNode(7);
cout << sumOfNodes(root);
return 0;
}
```

Output:



PROBLEM 6

Objective: **Same Tree….**Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Code:

```
#include <iostream>
using namespace std;
struct TreeNode {
int val;
TreeNode *left, *right;
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
bool isSameTree(TreeNode* p, TreeNode* q) {
if (!p && !q) return true;
if (!p || !q || p->val != q->val) return false;
return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}
int main() {
TreeNode* p = new TreeNode(1);
p->left = new TreeNode(2);
p->right = new TreeNode(3);
TreeNode* q = new TreeNode(1);
q->left = new TreeNode(2);
q->right = new TreeNode(3);
cout << (isSameTree(p, q) ? "true" : "false");
return 0;
}
```

Output:

```
Output
true
```

PROBLEM 7

Objective: Symmetric tree……

Code:

```
#include <iostream>
using namespace std;
struct TreeNode {
int val;
```

```
TreeNode *left, *right;
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
bool isMirror(TreeNode* t1, TreeNode* t2) {
if (!t1 && !t2) return true;
if (!t1 || !t2 || t1->val != t2->val) return false;
return isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);
}
bool isSymmetric(TreeNode* root) {
if (!root) return true;
return isMirror(root->left, root->right);
}
int main() {
TreeNode* root = new TreeNode(1);
root->left = new TreeNode(2);
root->right = new TreeNode(2);
root->left->left = new TreeNode(3);
root->left->right = new TreeNode(4);
root->right->left = new TreeNode(4);
root->right->right = new TreeNode(3);
cout << (isSymmetric(root) ? "true" : "false");
return 0;
}
```

Output:

PROBLEM 8

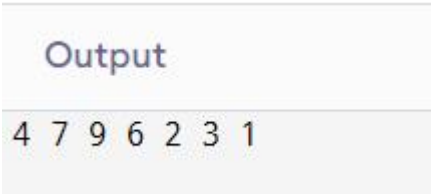
Objective: Invert binary tree

Code:

```
#include <iostream>
using namespace std;
struct TreeNode {
int val;
TreeNode *left, *right;
TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
TreeNode* invertTree(TreeNode* root) {
if (!root) return nullptr;
TreeNode* left = invertTree(root->left);
TreeNode* right = invertTree(root->right);
```

```
root->left = right;
    root->right = left;
    return root;
}

void preorder(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    preorder(root->left);
    preorder(root->right);
}

int main() {
    TreeNode* root = new TreeNode(4);
    root->left = new TreeNode(2);
    root->right = new TreeNode(7);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(9);

    root = invertTree(root);
    preorder(root);
    return 0;
}
```

Output:



PROBLEM 9

Objective: leaf nodes of a binary tree

Code:

```
#include <iostream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
int countLeaves(TreeNode* root) {
    if (!root) return 0;
    if (!root->left && !root->right) return 1;
    return countLeaves(root->left) + countLeaves(root->right);
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);

    cout << countLeaves(root);  // Output: 3
    return 0;
}
```

Output:

PROBLEM 10

Objective: Path sum

Code:

```
#include <iostream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

bool hasPathSum(TreeNode* root, int targetSum) {
    if (!root) return false;
    if (!root->left && !root->right) return root->val == targetSum;
    targetSum -= root->val;
    return hasPathSum(root->left, targetSum) || hasPathSum(root->right, targetSum);
}

int main() {
```

```
    TreeNode* root = new TreeNode(5);
    root->left = new TreeNode(4);
    root->right = new TreeNode(8);
    root->left->left = new TreeNode(11);
    root->right->left = new TreeNode(13);
    root->right->right = new TreeNode(4);
    root->left->left->left = new TreeNode(7);
    root->left->left->right = new TreeNode(2);
    root->right->right->right = new TreeNode(1);

    int targetSum = 22;
    cout << (hasPathSum(root, targetSum) ? "true" : "false");
    return 0;
}
```

Output:

```
Output
true
```

PROBLEM 11

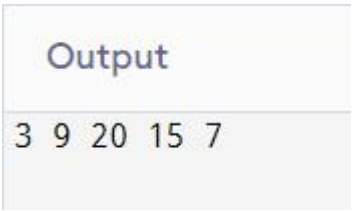
Objective: Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* buildTreeHelper(vector<int>& preorder, unordered_map<int, int>& inorderMap, int&
preIndex, int inStart, int inEnd) {
    if (inStart > inEnd) return nullptr;

    int rootValue = preorder[preIndex++];
    TreeNode* root = new TreeNode(rootValue);
```

```
    int inIndex = inorderMap[rootValue];

    root->left = buildTreeHelper(preorder, inorderMap, preIndex, inStart, inIndex - 1);
    root->right = buildTreeHelper(preorder, inorderMap, preIndex, inIndex + 1, inEnd);

    return root;
}

TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
    unordered_map<int, int> inorderMap;
    for (int i = 0; i < inorder.size(); ++i) {
        inorderMap[inorder[i]] = i;
    }

    int preIndex = 0;
    return buildTreeHelper(preorder, inorderMap, preIndex, 0, inorder.size() - 1);
}

void preorderTraversal(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

int main() {
    vector<int> preorder = {3, 9, 20, 15, 7};
    vector<int> inorder = {9, 3, 15, 20, 7};

    TreeNode* root = buildTree(preorder, inorder);
    preorderTraversal(root);  // Output: 3 9 20 15 7
    return 0;
}
```

Output:



PROBLEM 12

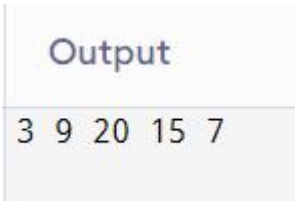
Objective: Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree.

Code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, unordered_map<int, int>& inorderMap, int& postIndex, int inStart, int inEnd) {
    if (inStart > inEnd) return nullptr;

    int rootValue = postorder[postIndex--];
    TreeNode* root = new TreeNode(rootValue);

    int inIndex = inorderMap[rootValue];

    root->right = buildTreeHelper(inorder, postorder, inorderMap, postIndex, inIndex + 1, inEnd);
    root->left = buildTreeHelper(inorder, postorder, inorderMap, postIndex, inStart, inIndex - 1);

    return root;
}

TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
    unordered_map<int, int> inorderMap;
    for (int i = 0; i < inorder.size(); ++i) {
        inorderMap[inorder[i]] = i;
    }

    int postIndex = postorder.size() - 1;
    return buildTreeHelper(inorder, postorder, inorderMap, postIndex, 0, inorder.size() - 1);
}

void preorderTraversal(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}
```

```
int main() {
    vector<int> inorder = {9, 3, 15, 20, 7};
    vector<int> postorder = {9, 15, 7, 20, 3};

    TreeNode* root = buildTree(inorder, postorder);
    preorderTraversal(root);  // Output: 3 9 20 15 7
    return 0;
}
```

Output:



PROBLEM 13

Objective: Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

Code:

```
#include <iostream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
    // Base case: if root is null or if we found p or q
    if (!root || root == p || root == q) {
        return root;
    }

    // Recurse on the left and right subtree
    TreeNode* left = lowestCommonAncestor(root->left, p, q);
    TreeNode* right = lowestCommonAncestor(root->right, p, q);

    // If p and q are found in different subtrees, the current node is the LCA
    if (left && right) {
        return root;
    }

    // Otherwise, return the non-null child (if any)
    return left ? left : right;
```

```
}

void inorderTraversal(TreeNode* root) {
    if (!root) return;
    inorderTraversal(root->left);
    cout << root->val << " ";
    inorderTraversal(root->right);
}

int main() {
    // Constructing the binary tree
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(5);
    root->right = new TreeNode(1);
    root->left->left = new TreeNode(6);
    root->left->right = new TreeNode(2);
    root->right->left = new TreeNode(0);
    root->right->right = new TreeNode(8);
    root->left->right->left = new TreeNode(7);
    root->left->right->right = new TreeNode(4);

    TreeNode* p = root->left;  // Node with value 5
    TreeNode* q = root->right; // Node with value 1

    TreeNode* lca = lowestCommonAncestor(root, p, q);
    cout << "LCA of " << p->val << " and " << q->val << " is: " << lca->val << endl;

    return 0;
}
```

Output:

```
Output
LCA of 5 and 1 is: 3
```

PROBLEM 14

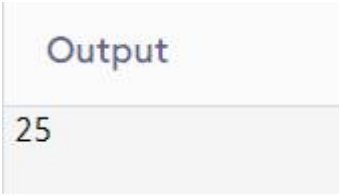
Objective: You are given the root of a binary tree containing digits from 0 to 9 only.Each root-to-leaf path in the tree represents a number.

Code:

```
#include <iostream>
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

int sumNumbers(TreeNode* root, int currentSum) {
    if (!root) return 0;
    currentSum = currentSum * 10 + root->val;
    if (!root->left && !root->right) return currentSum;
    return sumNumbers(root->left, currentSum) + sumNumbers(root->right, currentSum);
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);

    cout << sumNumbers(root, 0) << endl;

    return 0;
}
```

Output:



PROBLEM 15

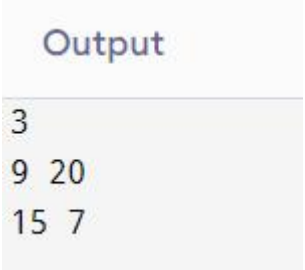
Objective: Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>> result;
    if (!root) return result;

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int levelSize = q.size();
        vector<int> level;

        for (int i = 0; i < levelSize; ++i) {
            TreeNode* node = q.front();
            q.pop();
            level.push_back(node->val);

            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }

        result.push_back(level);
    }

    return result;
}

int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);
    vector<vector<int>> result = levelOrder(root);
    for (const auto& level : result) {
        for (int val : level) {
            cout << val << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:



PROBLEM 16

Objective: Populating Next Right Pointers in Each Node

Code:

```
#include <iostream>
#include <queue>
using namespace std;

struct Node {
   int val;
   Node *left, *right, *next;
   Node(int x) : val(x), left(nullptr), right(nullptr), next(nullptr) {}
};

Node* connect(Node* root) {
   if (!root) return nullptr;

   Node* levelStart = root;

   while (levelStart) {
      Node* current = levelStart;
      levelStart = nullptr;
      Node* prev = nullptr;

      while (current) {
         if (current->left) {
            if (!levelStart) levelStart = current->left;
            if (prev) prev->next = current->left;
            prev = current->left;
         }
         if (current->right) {
            if (!levelStart) levelStart = current->right;
            if (prev) prev->next = current->right;
            prev = current->right;
         }
         current = current->next;
      }
   }

   return root;
}

int main() {
   Node* root = new Node(1);
   root->left = new Node(2);
   root->right = new Node(3);
   root->left->left = new Node(4);
   root->left->right = new Node(5);
```

```
    root->right->left = new Node(6);
    root->right->right = new Node(7);

    connect(root);

    Node* level = root;
    while (level) {
        Node* current = level;
        while (current) {
            cout << current->val << " ";
            current = current->next;
        }
        cout << "# ";
        level = level->left;
    }

    return 0;
}
```

Output:

PROBLEM 17

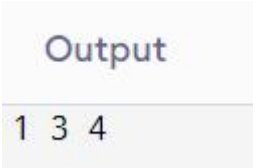
Objective: Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

vector<int> rightSideView(TreeNode* root) {
    vector<int> result;
    if (!root) return result;
```

```
    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int size = q.size();
        for (int i = 0; i < size; ++i) {
            TreeNode* node = q.front();
            q.pop();

            if (i == size - 1) result.push_back(node->val);

            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
    }

    return result;
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->right = new TreeNode(5);
    root->right->right = new TreeNode(4);

    vector<int> result = rightSideView(root);

    for (int val : result) cout << val << " ";

    return 0;
}
```

Output:



PROBLEM 18

Objective: Given the root of a binary tree, return the zigzag level order traversal of its nodes' values.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <deque>
using namespace std;

struct TreeNode {
   int val;
   TreeNode *left, *right;
   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
   vector<vector<int>> result;
   if (!root) return result;

   queue<TreeNode*> q;
   q.push(root);
   bool leftToRight = true;

   while (!q.empty()) {
      int size = q.size();
      vector<int> level(size);

      for (int i = 0; i < size; ++i) {
         TreeNode* node = q.front();
         q.pop();

         int index = (leftToRight) ? i : size - 1 - i;
         level[index] = node->val;

         if (node->left) q.push(node->left);
         if (node->right) q.push(node->right);
      }

      result.push_back(level);
      leftToRight = !leftToRight;
   }

   return result;
}

int main() {
```

```
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);

    vector<vector<int>> result = zigzagLevelOrder(root);

    for (const auto& level : result) {
        for (int val : level) {
            cout << val << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:

```
Output
3
20 9
15 7
```

PROBLEM 19

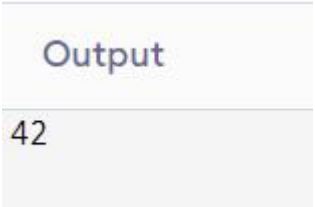
Objective: A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.The path sum of a path is the sum of the node's values in the path.Given the root of a binary tree, return the maximum path sum of any non-empty path.

Code:

```
#include <iostream>
#include <algorithm>
#include <climits>  // Include this header for INT_MIN
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    int maxPathSum(TreeNode* root) {
```

```
        int result = INT_MIN;
        maxPathSumHelper(root, result);
        return result;
    }

private:
    int maxPathSumHelper(TreeNode* node, int& result) {
        if (!node) return 0;

        int left = max(0, maxPathSumHelper(node->left, result));
        int right = max(0, maxPathSumHelper(node->right, result));

        result = max(result, left + right + node->val);

        return max(left, right) + node->val;
    }
};

int main() {
    TreeNode* root = new TreeNode(-10);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);

    Solution sol;
    cout << sol.maxPathSum(root) << endl;

    return 0;
}
```

Output:



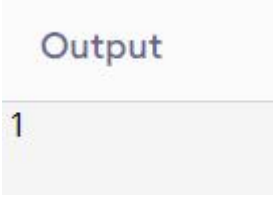
PROBLEM 20

Objective: Given a binary search tree (BST), write a function to find the kth smallest element in the tree.

Code:

```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        inorder(root, k);
        return result;
    }
private:
    int count = 0;
    int result = 0;
    void inorder(TreeNode* node, int k) {
        if (!node) return;
        inorder(node->left, k);
        count++;
        if (count == k) result = node->val;
        inorder(node->right, k);
    }
};
int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(1);
    root->right = new TreeNode(4);
    root->left->right = new TreeNode(2);
    Solution sol;
    cout << sol.kthSmallest(root, 1) << endl;
    return 0;
}
```

Output:



PROBLEM 21

Objective: You are given a tree (i.e. a connected, undirected graph that has no cycles) rooted at node 0 consisting of n nodes numbered from 0 to n - 1. The tree is represented by a 0-indexed array parent of size n, where parent[i] is the parent of node i. Since node 0 is the root, parent[0] == -1.You are also given a string s of length n, where s[i] is the character assigned to the edge between i and parent[i]. s[0] can be ignored.Return the number of pairs of nodes (u, v) such that u < v and the characters assigned to edges on the path from u to v can be rearranged to form a palindrome.A string is a palindrome when it reads the same backwards as forwards.

Code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

class Solution {
public:
   int countPalindromePaths(vector<int>& parent, string& s) {
      int n = parent.size();
      vector<vector<int>> tree(n);
      for (int i = 1; i < n; ++i) {
         tree[parent[i]].push_back(i);
      }
      unordered_map<int, int> count;
      int result = 0;
      dfs(0, tree, s, count, result);
      return result;
   }

private:
   void dfs(int node, vector<vector<int>>& tree, string& s, unordered_map<int, int>& count, int&
result) {
      int mask = 1 << (s[node] - 'a');
      count[mask]++;
      for (int child : tree[node]) {
         dfs(child, tree, s, count, result);
      }
      result += count[mask];
      for (auto& [key, value] : count) {
         if (value > 0) {
            result += value;
         }
      }
      count[mask]--;
   }
};

int main() {
```

```
    vector<int> parent = {-1, 0, 0, 1, 1, 2};
    string s = "acaabc";
    Solution sol;
    cout << sol.countPalindromePaths(parent, s) << endl;

    return 0;
}
```

Output:

PROBLEM 22

Objective: There is an undirected tree with n nodes labeled from 0 to n - 1. You are given the integer n and a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree.You are also given a 0-indexed integer array values of length n, where values[i] is the value associated with the ith node, and an integer k.A valid split of the tree is obtained by removing any set of edges, possibly empty, from the tree such that the resulting components all have values that are divisible by k, where the value of a connected component is the sum of the values of its nodes.Return the maximum number of components in any valid split.

Code:

```
#include <iostream>
#include <vector>
using namespace std;

class Solution {
public:
    int maxComponents(int n, vector<vector<int>>& edges, vector<int>& values, int k) {
        vector<vector<int>> tree(n);
        for (auto& edge : edges) {
            tree[edge[0]].push_back(edge[1]);
            tree[edge[1]].push_back(edge[0]);
        }
        int result = 0;
        dfs(0, -1, tree, values, k, result);
        return result;
    }

private:
    int dfs(int node, int parent, vector<vector<int>>& tree, vector<int>& values, int k, int& result)
{
        int sum = values[node];
```

```
        for (int child : tree[node]) {
            if (child == parent) continue;
            sum += dfs(child, node, tree, values, k, result);
        }
        if (sum % k == 0) {
            result++;
            return 0;
        }
        return sum;
    }
};

int main() {
    Solution sol;
    int n = 5;
    vector<vector<int>> edges = {{0, 2}, {1, 2}, {1, 3}, {2, 4}};
    vector<int> values = {1, 8, 1, 4, 4};
    int k = 6;
    cout << sol.maxComponents(n, edges, values, k) << endl;

    return 0;
}
```

Output: