# I. Binary Tree

## Ex1:

#include <iostream>

struct Node

{

    int data;

    Node \*left;

    Node \*right;

    Node(int value) : data(value), left(nullptr), right(nullptr) {}

};

Node \*createBinaryTree(int arr[], int n, int index)

{

    if (index >= n)

    {

        return nullptr;

    }

    Node \*root = new Node(arr[index]);

    root->left = createBinaryTree(arr, n, index \* 2 + 1);

    root->right = createBinaryTree(arr, n, index \* 2 + 2);

    return root;

}

double averageNodeValue(Node \*root, int &count, int &sum)

{

    if (root == nullptr)

    {

        return 0;

    }

    sum += root->data;

    count++;

    averageNodeValue(root->left, count, sum);

    averageNodeValue(root->right, count, sum);

    return (double)sum / count;

}

double averagePositiveNodeValue(Node \*root, int &count, int &sum)

{

    if (root == nullptr)

    {

        return 0;

    }

    if (root->data > 0)

    {

        sum += root->data;

        count++;

    }

    averagePositiveNodeValue(root->left, count, sum);

    averagePositiveNodeValue(root->right, count, sum);

    return (double)sum / count;

}

double averageNegativeNodeValue(Node \*root, int &count, int &sum)

{

    if (root == nullptr)

    {

        return 0;

    }

    if (root->data < 0)

    {

        sum += root->data;

        count++;

    }

    averageNegativeNodeValue(root->left, count, sum);

    averageNegativeNodeValue(root->right, count, sum);

    return (double)sum / count;

}

double ratioPosToNeg(Node \*root, int &sumPos, int &sumNeg)

{

    if (root == nullptr)

    {

        return 0;

    }

    if (root->data > 0)

    {

        sumPos += root->data;

    }

    if (root->data < 0)

    {

        sumNeg += root->data;

    }

    ratioPosToNeg(root->left, sumPos, sumNeg);

    ratioPosToNeg(root->right, sumPos, sumNeg);

    return (sumNeg != 0) ? (double)sumPos / sumNeg : 0;

}

int main()

{

    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, -12, -13, 14};

    int n = sizeof(arr) / sizeof(arr[0]);

    Node \*root = createBinaryTree(arr, n, 0);

    int count = 0, sum = 0;

    double avgAll = averageNodeValue(root, count, sum);

    std::cout << "Average value of all nodes in the tree: " << avgAll << std::endl;

    int posCount = 0, posSum = 0;

    double avgPos = averagePositiveNodeValue(root, posCount, posSum);

    std::cout << "Average value of positive nodes in the tree: " << avgPos << std::endl;

    int negCount = 0, negSum = 0;

    double avgNeg = averageNegativeNodeValue(root, negCount, negSum);

    std::cout << "Average value of negative nodes in the tree: " << avgNeg << std::endl;

    int sumPos = 0, sumNeg = 0;

    double ratio = ratioPosToNeg(root, sumPos, sumNeg);

    std::cout << "Ratio of positive sum to negative sum: " << ratio << std::endl;

    return 0;

}

|  |  |
| --- | --- |
| Input | Output |
| [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, -12, -13, -14] | 3.92857 |
| 6.66667 |
| -12.5 |
| -3.2 |

## Ex2:

#include <iostream>

struct Node

{

    int data;

    Node \*left;

    Node \*right;

    Node(int value) : data(value), left(nullptr), right(nullptr) {}

};

void modifyNode(Node \*root)

{

    if (root == nullptr)

    {

        return;

    }

    Node \*temp = root->left;

    root->left = root->right;

    root->right = temp;

    temp = root->left->left;

    root->left->left = root->left->right;

    root->left->right = temp;

}

Node \*createBinaryTree()

{

    Node \*root = new Node(10);

    root->left = new Node(5);

    root->right = new Node(15);

    root->left->left = new Node(3);

    root->left->right = new Node(9);

    root->right->left = new Node(12);

    root->right->right = new Node(18);

    root->right->right->left = new Node(20);

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

    return root;

}

void printTree(Node \*root)

{

    if (root == nullptr)

    {

        return;

    }

    std::cout << root->data << " ";

    printTree(root->left);

    printTree(root->right);

}

int main()

{

    Node \*root = createBinaryTree();

    std::cout << "Original tree: ";

    printTree(root);

    std::cout << std::endl;

    modifyNode(root);

    std::cout << "Modified tree: ";

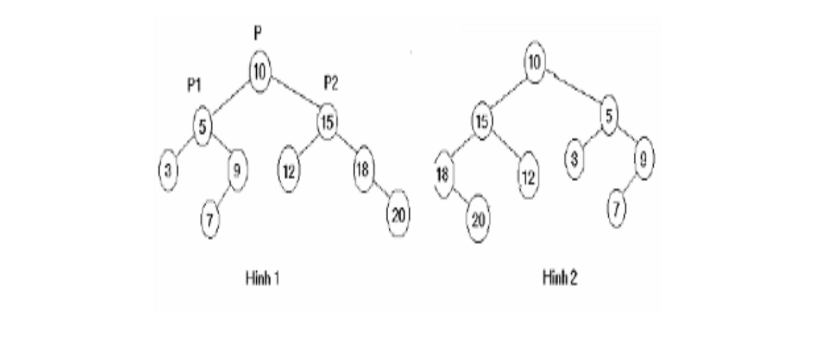
    printTree(root);

    std::cout << std::endl;

    return 0;

}

|  |  |
| --- | --- |
| Input | Output |
| Original tree: 10 5 3 9 7 15 12 18 20 | Modified tree: 10 15 18 20 12 5 3 9 7 |



## Ex3:

#include <iostream>

struct Node

{

    int data;

    Node \*left;

    Node \*right;

    Node(int value) : data(value), left(nullptr), right(nullptr) {}

};

Node \*insertNode(Node \*root, int value)

{

    if (root == nullptr)

    {

        return new Node(value);

    }

    if (value < root->data)

    {

        root->left = insertNode(root->left, value);

    }

    else if (value > root->data)

    {

        root->right = insertNode(root->right, value);

    }

    return root;

}

Node \*createBinaryTree(int arr[], int n)

{

    Node \*root = nullptr;

    for (int i = 0; i < n; ++i)

    {

        root = insertNode(root, arr[i]);

    }

    return root;

}

Node \*findMin(Node \*root)

{

    while (root != nullptr && root->left != nullptr)

    {

        root = root->left;

    }

    return root;

}

Node \*findMax(Node \*root)

{

    while (root != nullptr && root->right != nullptr)

    {

        root = root->right;

    }

    return root;

}

int main()

{

    int arr[] = {10, 5, 3, 9, 7, 15, 12, 18, 20};

    int n = sizeof(arr) / sizeof(arr[0]);

    Node \*root = createBinaryTree(arr, n);

    Node \*minNode = findMin(root);

    Node \*maxNode = findMax(root);

    if (minNode != nullptr)

    {

        std::cout << "Minimum value in the tree: " << minNode->data << std::endl;

    }

    else

    {

        std::cout << "Tree is empty." << std::endl;

    }

    if (maxNode != nullptr)

    {

        std::cout << "Maximum value in the tree: " << maxNode->data << std::endl;

    }

    else

    {

        std::cout << "Tree is empty." << std::endl;

    }

    return 0;

}

|  |  |
| --- | --- |
| Input | Output |
| 10 5 3 9 7 15 12 18 20 | Minimum value in the tree: 3  Maximum value in the tree: 20 |