```
#include <iostream>
#include <algorithm>

struct Node {
    int data;
    Node* left;
    Node* right;
    int height;

    Node(int value) : data(value), left(nullptr), right(nullptr), height(1) {}
};

int height(Node* node) {
    if (node == nullptr) {
        return 0;
    }
    return node->height;
}

int balanceFactor(Node* node) {
    if (node == nullptr) {
        return 0;
    }
    return height(node->left) - height(node->right);
}

void updateHeight(Node* node) {
    node->height = 1 + std::max(height(node->left), height(node->right));
}

Node* rotateRight(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;

    x->right = y;
    y->left = T2;
```

```
    updateHeight(y);
    updateHeight(x);

    return x;
}

Node* rotateLeft(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;

    y->left = x;
    x->right = T2;

    updateHeight(x);
    updateHeight(y);

    return y;
}

Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }

    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    } else {
        return root;
    }

    updateHeight(root);

    int bf = balanceFactor(root);


    if (bf > 1 && value < root->left->data) {
```

```
        return rotateRight(root);
    }
    if (bf < -1 && value > root->right->data) {
        return rotateLeft(root);
    }
    if (bf > 1 && value > root->left->data) {
        root->left = rotateLeft(root->left);
        return rotateRight(root);
    }
    if (bf < -1 && value < root->right->data) {
        root->right = rotateRight(root->right);
        return rotateLeft(root);
    }

    return root;
}

Node* minValueNode(Node* node) {
    Node* current = node;

    while (current->left != nullptr) {
        current = current->left;
    }

    return current;
}

Node* deleteNode(Node* root, int value) {
    if (root == nullptr) {
        return root;
    }

    if (value < root->data) {
        root->left = deleteNode(root->left, value);
    } else if (value > root->data) {
        root->right = deleteNode(root->right, value);
    } else {
        if (root->left == nullptr || root->right == nullptr) {
```

```
        Node* temp = root->left ? root->left : root->right;

        if (temp == nullptr) {
            temp = root;
            root = nullptr;
        } else {
            *root = *temp;
        }

        delete temp;
    } else {
        Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
}

if (root == nullptr) {
    return root;
}

updateHeight(root);

int bf = balanceFactor(root);

// بررسی موارد ناتوانایی
if (bf > 1 && balanceFactor(root->left) >= 0) {
    return rotateRight(root);
}
if (bf > 1 && balanceFactor(root->left) < 0) {
    root->left = rotateLeft(root->left);
    return rotateRight(root);
}
if (bf < -1 && balanceFactor(root->right) <= 0) {
    return rotateLeft(root);
}
if (bf < -1 && balanceFactor(root->right) > 0) {
    root->right = rotateRight(root->right);
```

```
        return rotateLeft(root);
    }

    return root;
}

int main() {
    Node* root = nullptr;



    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = deleteNode(root, 20);

    return 0;
}
```