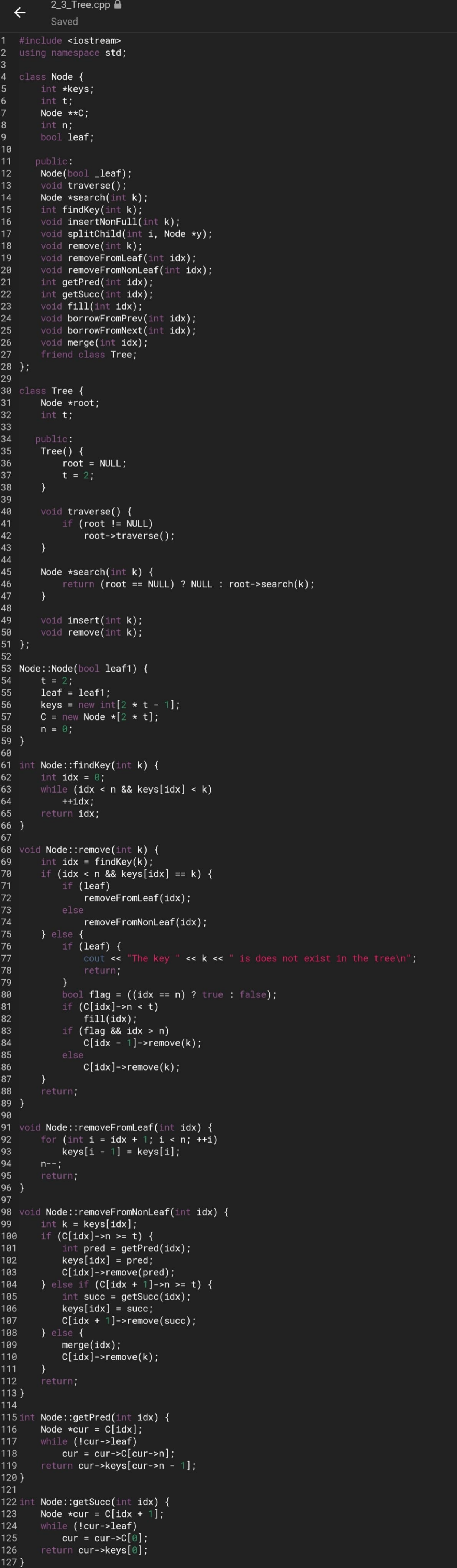
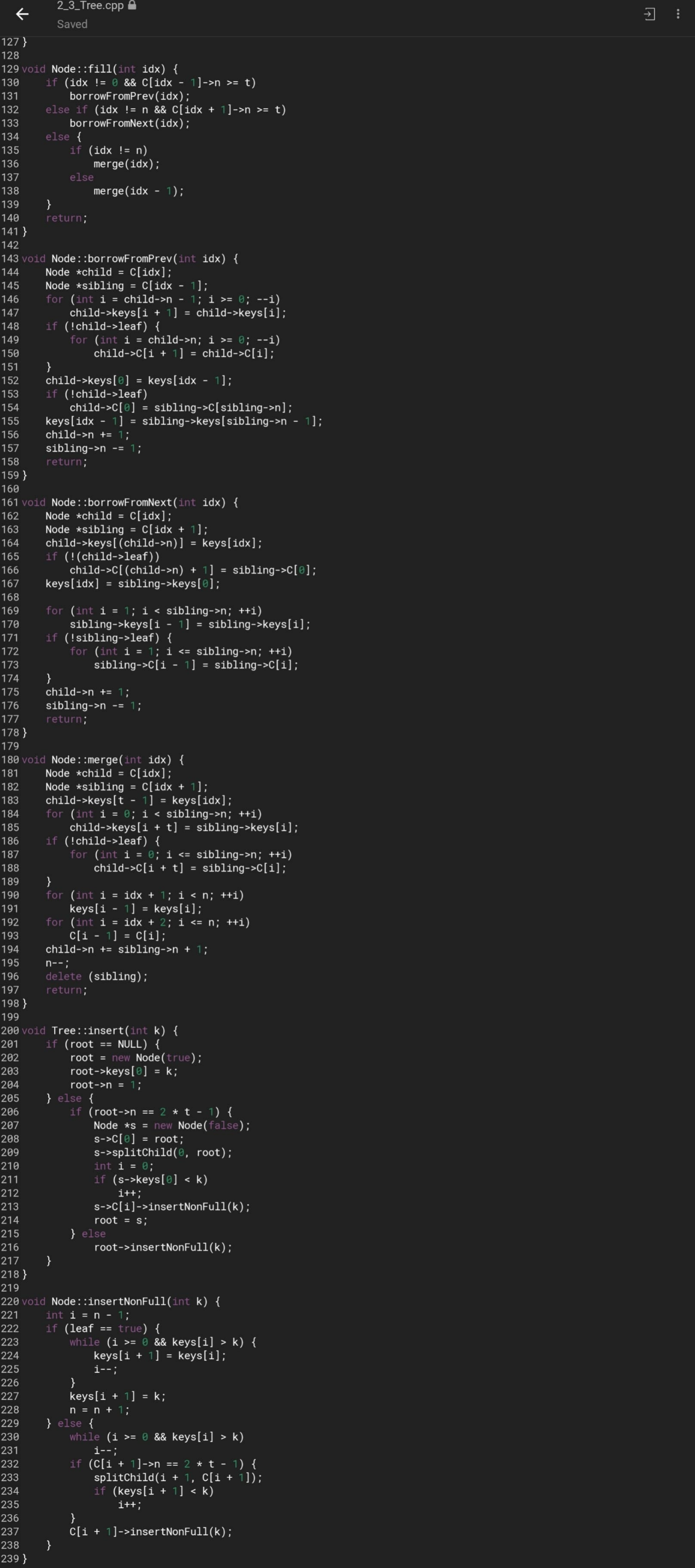
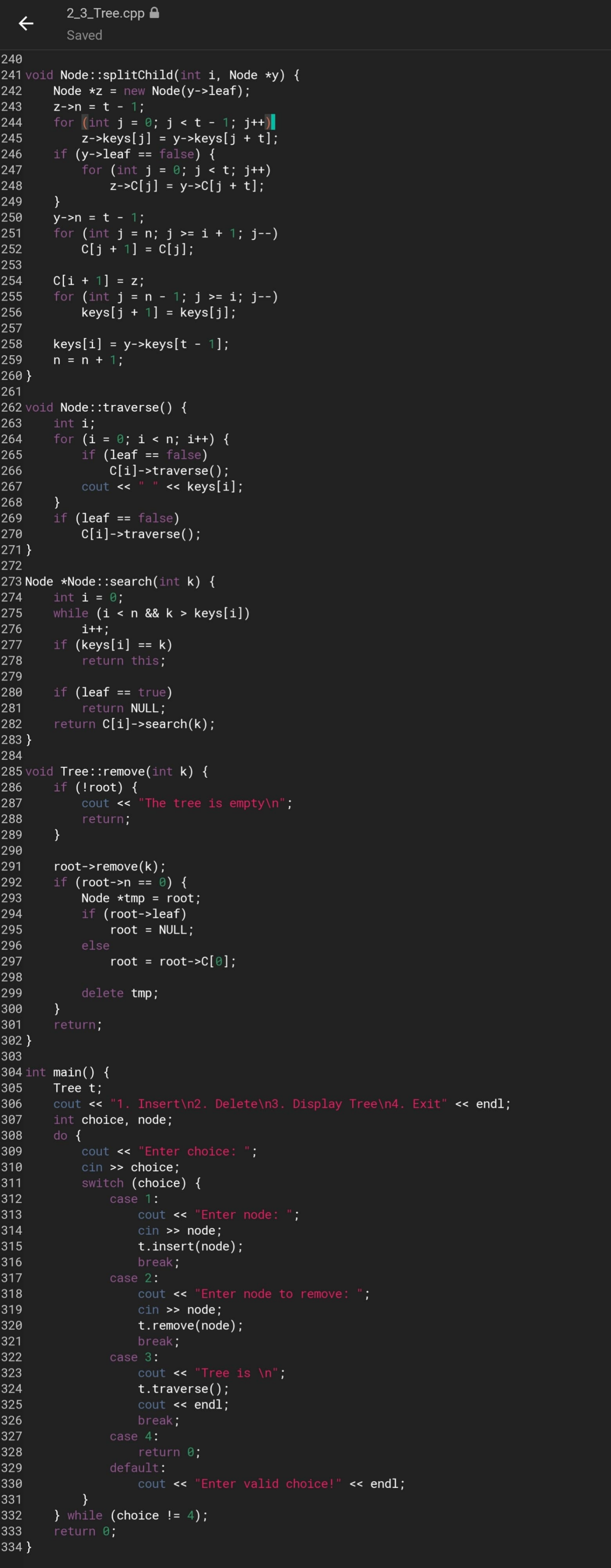


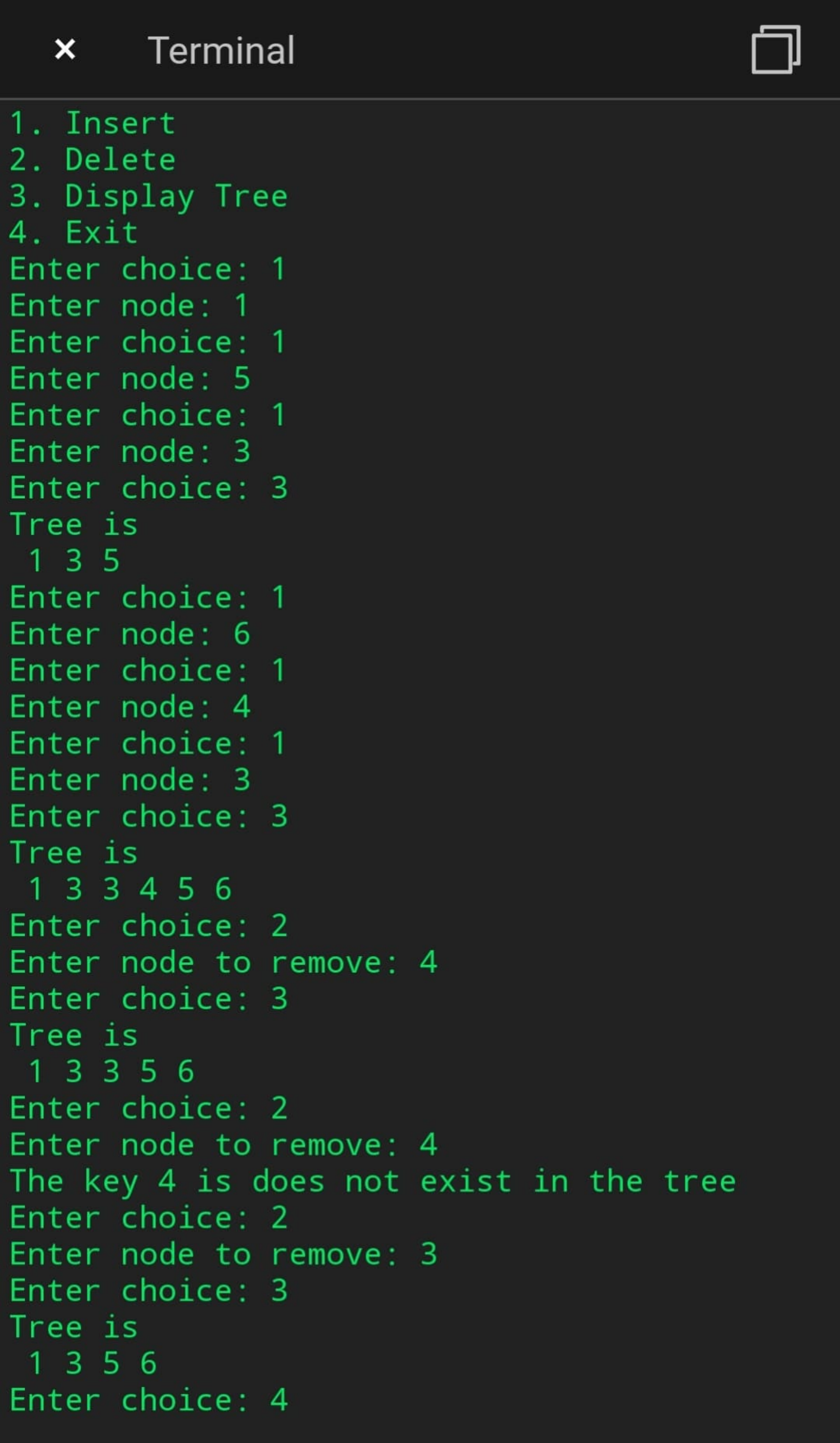
```cpp
#include <iostream>
using namespace std;

class Node {
    int *keys;
    int t;
    Node **C;
    int n;
    bool leaf;

   public:
    Node(bool _leaf);
    void traverse();
    Node *search(int k);
    int findKey(int k);
    void insertNonFull(int k);
    void splitChild(int i, Node *y);
    void remove(int k);
    void removeFromLeaf(int idx);
    void removeFromNonLeaf(int idx);
    int getPred(int idx);
    int getSucc(int idx);
    void fill(int idx);
    void borrowFromPrev(int idx);
    void borrowFromNext(int idx);
    void merge(int idx);
    friend class Tree;
};

class Tree {
    Node *root;
    int t;

   public:
    Tree() {
        root = NULL;
        t = 2;
    }

    void traverse() {
        if (root != NULL)
            root->traverse();
    }

    Node *search(int k) {
        return (root == NULL) ? NULL : root->search(k);
    }

    void insert(int k);
    void remove(int k);
};

Node::Node(bool leaf1) {
    t = 2;
    leaf = leaf1;
    keys = new int[2 * t - 1];
    C = new Node *[2 * t];
    n = 0;
}

int Node::findKey(int k) {
    int idx = 0;
    while (idx < n && keys[idx] < k)
        ++idx;
    return idx;
}

void Node::remove(int k) {
    int idx = findKey(k);
    if (idx < n && keys[idx] == k) {
        if (leaf)
            removeFromLeaf(idx);
        else
            removeFromNonLeaf(idx);
    } else {
        if (leaf) {
            cout << "The key " << k << " is does not exist in the tree\n";
            return;
        }
        bool flag = ((idx == n) ? true : false);
        if (C[idx]->n < t)
            fill(idx);
        if (flag && idx > n)
            C[idx - 1]->remove(k);
        else
            C[idx]->remove(k);
    }
    return;
}

void Node::removeFromLeaf(int idx) {
    for (int i = idx + 1; i < n; ++i)
        keys[i - 1] = keys[i];
    n--;
    return;
}

void Node::removeFromNonLeaf(int idx) {
    int k = keys[idx];
    if (C[idx]->n >= t) {
        int pred = getPred(idx);
        keys[idx] = pred;
        C[idx]->remove(pred);
    } else if (C[idx + 1]->n >= t) {
        int succ = getSucc(idx);
        keys[idx] = succ;
        C[idx + 1]->remove(succ);
    } else {
        merge(idx);
        C[idx]->remove(k);
    }
    return;
}

int Node::getPred(int idx) {
    Node *cur = C[idx];
    while (!cur->leaf)
        cur = cur->C[cur->n];
    return cur->keys[cur->n - 1];
}

int Node::getSucc(int idx) {
    Node *cur = C[idx + 1];
    while (!cur->leaf)
        cur = cur->C[0];
    return cur->keys[0];
}
```



```cpp
}

void Node::fill(int idx) {
    if (idx != 0 && C[idx - 1]->n >= t)
        borrowFromPrev(idx);
    else if (idx != n && C[idx + 1]->n >= t)
        borrowFromNext(idx);
    else {
        if (idx != n)
            merge(idx);
        else
            merge(idx - 1);
    }
    return;
}

void Node::borrowFromPrev(int idx) {
    Node *child = C[idx];
    Node *sibling = C[idx - 1];
    for (int i = child->n - 1; i >= 0; --i)
        child->keys[i + 1] = child->keys[i];
    if (!child->leaf) {
        for (int i = child->n; i >= 0; --i)
            child->C[i + 1] = child->C[i];
    }
    child->keys[0] = keys[idx - 1];
    if (!child->leaf)
        child->C[0] = sibling->C[sibling->n];
    keys[idx - 1] = sibling->keys[sibling->n - 1];
    child->n += 1;
    sibling->n -= 1;
    return;
}

void Node::borrowFromNext(int idx) {
    Node *child = C[idx];
    Node *sibling = C[idx + 1];
    child->keys[(child->n)] = keys[idx];
    if (!(child->leaf))
        child->C[(child->n) + 1] = sibling->C[0];
    keys[idx] = sibling->keys[0];

    for (int i = 1; i < sibling->n; ++i)
        sibling->keys[i - 1] = sibling->keys[i];
    if (!sibling->leaf) {
        for (int i = 1; i <= sibling->n; ++i)
            sibling->C[i - 1] = sibling->C[i];
    }
    child->n += 1;
    sibling->n -= 1;
    return;
}

void Node::merge(int idx) {
    Node *child = C[idx];
    Node *sibling = C[idx + 1];
    child->keys[t - 1] = keys[idx];
    for (int i = 0; i < sibling->n; ++i)
        child->keys[i + t] = sibling->keys[i];
    if (!child->leaf) {
        for (int i = 0; i <= sibling->n; ++i)
            child->C[i + t] = sibling->C[i];
    }
    for (int i = idx + 1; i < n; ++i)
        keys[i - 1] = keys[i];
    for (int i = idx + 2; i <= n; ++i)
        C[i - 1] = C[i];
    child->n += sibling->n + 1;
    n--;
    delete (sibling);
    return;
}

void Tree::insert(int k) {
    if (root == NULL) {
        root = new Node(true);
        root->keys[0] = k;
        root->n = 1;
    } else {
        if (root->n == 2 * t - 1) {
            Node *s = new Node(false);
            s->C[0] = root;
            s->splitChild(0, root);
            int i = 0;
            if (s->keys[0] < k)
                i++;
            s->C[i]->insertNonFull(k);
            root = s;
        } else
            root->insertNonFull(k);
    }
}

void Node::insertNonFull(int k) {
    int i = n - 1;
    if (leaf == true) {
        while (i >= 0 && keys[i] > k) {
            keys[i + 1] = keys[i];
            i--;
        }
        keys[i + 1] = k;
        n = n + 1;
    } else {
        while (i >= 0 && keys[i] > k)
            i--;
        if (C[i + 1]->n == 2 * t - 1) {
            splitChild(i + 1, C[i + 1]);
            if (keys[i + 1] < k)
                i++;
        }
        C[i + 1]->insertNonFull(k);
    }
}
```



```cpp

void Node::splitChild(int i, Node *y) {
    Node *z = new Node(y->leaf);
    z->n = t - 1;
    for (int j = 0; j < t - 1; j++)
        z->keys[j] = y->keys[j + t];
    if (y->leaf == false) {
        for (int j = 0; j < t; j++)
            z->C[j] = y->C[j + t];
    }
    y->n = t - 1;
    for (int j = n; j >= i + 1; j--)
        C[j + 1] = C[j];

    C[i + 1] = z;
    for (int j = n - 1; j >= i; j--)
        keys[j + 1] = keys[j];

    keys[i] = y->keys[t - 1];
    n = n + 1;
}

void Node::traverse() {
    int i;
    for (i = 0; i < n; i++) {
        if (leaf == false)
            C[i]->traverse();
        cout << " " << keys[i];
    }
    if (leaf == false)
        C[i]->traverse();
}

Node *Node::search(int k) {
    int i = 0;
    while (i < n && k > keys[i])
        i++;
    if (keys[i] == k)
        return this;

    if (leaf == true)
        return NULL;
    return C[i]->search(k);
}

void Tree::remove(int k) {
    if (!root) {
        cout << "The tree is empty\n";
        return;
    }

    root->remove(k);
    if (root->n == 0) {
        Node *tmp = root;
        if (root->leaf)
            root = NULL;
        else
            root = root->C[0];

        delete tmp;
    }
    return;
}

int main() {
    Tree t;
    cout << "1. Insert\n2. Delete\n3. Display Tree\n4. Exit" << endl;
    int choice, node;
    do {
        cout << "Enter choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter node: ";
                cin >> node;
                t.insert(node);
                break;
            case 2:
                cout << "Enter node to remove: ";
                cin >> node;
                t.remove(node);
                break;
            case 3:
                cout << "Tree is \n";
                t.traverse();
                cout << endl;
                break;
            case 4:
                return 0;
            default:
                cout << "Enter valid choice!" << endl;
        }
    } while (choice != 4);
    return 0;
}
```



```
1. Insert
2. Delete
3. Display Tree
4. Exit
Enter choice: 1
Enter node: 1
Enter choice: 1
Enter node: 5
Enter choice: 1
Enter node: 3
Enter choice: 3
Tree is
 1 3 5
Enter choice: 1
Enter node: 6
Enter choice: 1
Enter node: 4
Enter choice: 1
Enter node: 3
Enter choice: 3
Tree is
 1 3 3 4 5 6
Enter choice: 2
Enter node to remove: 4
Enter choice: 3
Tree is
 1 3 3 5 6
Enter choice: 2
Enter node to remove: 4
The key 4 is does not exist in the tree
Enter choice: 2
Enter node to remove: 3
Enter choice: 3
Tree is
 1 3 5 6
Enter choice: 4
```