#include <iostream>

#include <string>

using namespace std;

class Node

{

public:

    string word;

    int frequency;

    Node \*left;

    Node \*right;

    int height;

};

Node \*createNode(string word, int frequency)

{

    Node \*newNode = new Node();

    newNode->word = word;

    newNode->frequency = frequency;

    newNode->left = newNode->right = nullptr;

    newNode->height = 1;

    return newNode;

}

int getHeight(Node \*node)

{

    if (node == nullptr)

        return 0;

    return node->height;

}

int getBalance(Node \*node)

{

    if (node == nullptr)

        return 0;

    return getHeight(node->left) - getHeight(node->right);

}

Node \*rightRotate(Node \*y)

{

    Node \*x = y->left;

    Node \*T2 = x->right;

    x->right = y;

    y->left = T2;

    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;

    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;

    return x;

}

Node \*leftRotate(Node \*x)

{

    Node \*y = x->right;

    Node \*T2 = y->left;

    y->left = x;

    x->right = T2;

    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;

    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;

    return y;

}

Node \*insert(Node \*node, string word, int frequency)

{

    if (node == nullptr)

        return createNode(word, frequency);

    if (word < node->word)

        node->left = insert(node->left, word, frequency);

    else if (word > node->word)

        node->right = insert(node->right, word, frequency);

    node->frequency += frequency;

    node->height = 1 + max(getHeight(node->left), getHeight(node->right));

    int balance = getBalance(node);

    if (balance > 1 && word < node->left->word)

        return rightRotate(node);

    if (balance < -1 && word > node->right->word)

        return leftRotate(node);

    if (balance > 1 && word > node->left->word)

    {

        node->left = leftRotate(node->left);

        return rightRotate(node);

    }

    if (balance < -1 && word < node->right->word)

    {

        node->right = rightRotate(node->right);

        return leftRotate(node);

    }

    return node;

}

void searchPrefix(Node \*node, string prefix)

{

    if (node == nullptr)

        return;

    if (node->word.substr(0, prefix.length()) == prefix)

        cout << node->word << " (Frequency: " << node->frequency << ")" << endl;

    if (prefix < node->word)

        searchPrefix(node->left, prefix);

    else if (prefix > node->word)

        searchPrefix(node->right, prefix);

    else

    {

        searchPrefix(node->left, prefix);

        searchPrefix(node->right, prefix);

    }

}

int main()

{

    Node \*root = nullptr;

    root = insert(root, "car", 50);

    root = insert(root, "cat", 30);

    root = insert(root, "calendar", 20);

    string prefix = "ca";

    cout << "Suggestions for prefix \"" << prefix << "\":" << endl;

    searchPrefix(root, prefix);

    return 0;

}

