#include <iostream>

#include <locale.h>

#include <cstdlib>

#include <windows.h>

using namespace std;

struct arbol

{

    int dato;

    arbol \*izq;

    arbol \*der;

    int factorEquilibrio; *// Factor de equilibrio*

    arbol(int *val*) : dato(val), izq(NULL), der(NULL), factorEquilibrio(0) {}

};

arbol \*raiz = NULL, \*rec = NULL, \*nuevo = NULL;

arbol insertar(arbol \*&*raiz*, int *valor*);

void imprimirArbol(arbol \**raiz*, int *espacios* = 0);

int altura(arbol \**nodo*);

int factorEquilibrio(arbol \**nodo*);

void actualizarFactoresEquilibrio(arbol \**nodo*);

bool esAVL(arbol \**nodo*);

bool existe(arbol \**raiz*, int *valor*); *// Declaración de la función existe*

void rotacion(arbol \*&*rec*, arbol \*&*raiz*); *// Declaración de la función de rotacion*

void rotarIzquierda(arbol \*&*rec*); *// Declaración de la función de rotación izquierda*

void rotarDerecha(arbol \*&*rec*); *// Declaración de la función de rotación derecha*

void balancearHastaAVL(arbol \*&*raiz*); *// Declaración de la función de balanceo*

int main()

{

    int NumVeces;

    char opcion;

    srand(time(0)); *// Inicializa la semilla para números aleatorios*

    do

    {

        system("cls");

        do

        {

            cout << "Cuantas veces desea insertar un dato: ";

            cin >> NumVeces;

            if (NumVeces <= 0)

            {

                cout << "El numero de veces a insertar debe ser mayor a 0" << endl;

            }

        } while (NumVeces <= 0);

        int opcion1;

        do

        {

            cout << "De qué manera desea insertar los datos? " << endl;

            cout << "1. Manualmente" << endl;

            cout << "2. Aleatoriamente" << endl;

            cout << "Ingrese su opcion: ";

            cin >> opcion1;

        } while (opcion1 != 1 && opcion1 != 2);

        switch (opcion1)

        {

        case 1:

            do

            {

                int dato;

                cout << "Ingrese el dato " << (NumVeces) << ": ";

                cin >> dato;

                insertar(raiz, dato);

                cout << "Se insertó el dato: " << dato << endl;

                NumVeces--;

            } while (NumVeces > 0);

            cout << endl

                 << endl;

            break;

        case 2:

            while (NumVeces > 0)

            {

                int dato;

                do

                {

                    dato = rand() % 100 + 1;

                } while (existe(raiz, dato));

                insertar(raiz, dato);

                cout << "Se insertó el dato: " << dato << endl;

                NumVeces--;

            }

            cout << endl

                 << endl;

            break;

        default:

            break;

        }

*// Siempre actualizar factores antes de imprimir*

        actualizarFactoresEquilibrio(raiz);

        cout << " Arbol resultante: " << endl;

        imprimirArbol(raiz, 0);

*// Comprobar si es AVL*

        if (esAVL(raiz))

        {

            cout << "\nEl árbol ES AVL (todos los factores de equilibrio están entre -1 y 1)" << endl;

        }

        else

        {

            cout << "\nEl árbol NO es AVL (hay factores de equilibrio fuera de [-1, 1])" << endl;

        }

        system("pause");

        cout << endl

             << endl;

        cout << "Realizando rotaciones para balancear el árbol..." << endl;

*// Balancear el árbol hasta que sea AVL*

        balancearHastaAVL(raiz);

        cout << "Arbol después de las rotaciones: " << endl;

        imprimirArbol(raiz, 0);

*// Comprobar nuevamente si es AVL después de las rotaciones*

        if (esAVL(raiz))

        {

            cout << "\nEl árbol ES AVL después de las rotaciones." << endl;

        }

        else

        {

            cout << "\nEl árbol NO es AVL después de las rotaciones." << endl;

        }

        cout << endl

             << endl;

        system("pause");

        cout << endl

             << endl;

        cout << "Desea ingresar mas datos? (s/n): ";

        cin >> opcion;

        if (opcion == 'n' or opcion == 'N')

        {

            cout << endl

                 << endl

                 << "Saliendo...";

            Sleep(1000);

        }

    } while (opcion == 'S' or opcion == 's');

    return 0;

}

arbol insertar(arbol \*&*raiz*, int *valor*)

{

    if (raiz == NULL)

    {

        raiz = new arbol(valor);

    }

    else

    {

        if (valor < raiz->dato)

        {

            insertar(raiz->izq, valor);

        }

        else if (valor > raiz->dato)

        {

            insertar(raiz->der, valor);

        }

        else *// valor == raiz->dato*

        {

*// Si el valor ya existe, generar uno nuevo y volver a intentar*

            int nuevoValor = valor;

            do

            {

                nuevoValor = rand() % 100 + 1;

            } while (nuevoValor == valor);

            insertar(raiz, nuevoValor);

        }

    }

    return \*raiz;

}

void imprimirArbol(arbol \**raiz*, int *espacios*)

{

    if (raiz == NULL)

    {

        return;

    }

*// Aumentamos los espacios para la indentaci��n*

    imprimirArbol(raiz->der, espacios + 4);

*// Mostramos el valor con espacios para dar forma y el factor de equilibrio*

    cout << string(espacios, ' ') << raiz->dato << " (F:" << raiz->factorEquilibrio << ")" << endl

         << endl;

    imprimirArbol(raiz->izq, espacios + 4);

}

*// Calcula la altura de un nodo*

int altura(arbol \**nodo*)

{

    if (nodo == NULL)

        return 0;

    int altIzq = altura(nodo->izq);

    int altDer = altura(nodo->der);

    return 1 + max(altIzq, altDer);

}

*// Calcula el factor de equilibrio de un nodo*

int factorEquilibrio(arbol \**nodo*)

{

    if (nodo == NULL)

        return 0;

    return altura(nodo->der) - altura(nodo->izq);

}

*// Actualiza el factor de equilibrio de todos los nodos del árbol*

void actualizarFactoresEquilibrio(arbol \**nodo*)

{

    if (nodo == NULL)

        return;

    nodo->factorEquilibrio = factorEquilibrio(nodo);

    actualizarFactoresEquilibrio(nodo->izq);

    actualizarFactoresEquilibrio(nodo->der);

}

*// Comprueba si el árbol es AVL (todos los factores de equilibrio en [-1, 0, 1])*

bool esAVL(arbol \**nodo*)

{

    if (nodo == NULL)

        return true;

    if (nodo->factorEquilibrio < -1 || nodo->factorEquilibrio > 1)

        return false;

    return esAVL(nodo->izq) && esAVL(nodo->der);

}

*// Busca si un valor ya existe en el árbol*

bool existe(arbol \**raiz*, int *valor*)

{

    if (raiz == NULL)

        return false;

    if (raiz->dato == valor)

        return true;

    if (valor < raiz->dato)

        return existe(raiz->izq, valor);

    else

        return existe(raiz->der, valor);

}

*// Función para identificar el tipo de rotación necesaria en un nodo*

void rotacion(arbol \*&*rec*, arbol \*&*raiz*)

{

    if (rec == NULL)

    {

        return;

    }

    actualizarFactoresEquilibrio(raiz); *// Asegurarse de que los factores de equilibrio estén actualizados*

*// Recursivamente para los hijos*

    rotacion(rec->izq, raiz);

    rotacion(rec->der, raiz);

*// Caso I Izquierda-Izquierda*

    if (rec->factorEquilibrio < -1 && rec->izq && rec->izq->factorEquilibrio <= 0)

    {

        cout << "Rotación Izquierda-Izquierda en nodo: " << rec->dato << endl;

        rotarDerecha(rec); *// Agregar rotación a la derecha*

        cout << "CASO I: ARBOL RESULTANTE DESPUES DE ROTACION" << endl;

        imprimirArbol(raiz, 0); *// Imprimir el árbol después de la rotación*

        cout << endl;

        cout << "-----------------------------------------------------------------------" << endl;

        actualizarFactoresEquilibrio(raiz); *// Actualiza factores de equilibrio después de rotar*

    }

*// Caso II Derecha-Derecha*

    else if (rec->factorEquilibrio > 1 && rec->der && rec->der->factorEquilibrio > 0)

    {

        cout << "Rotación Derecha-Derecha en nodo: " << rec->dato << endl;

        rotarIzquierda(rec); *// Agregar rotación a la izquierda*

        cout << "CASO II: ARBOL RESULTANTE DESPUES DE ROTACION" << endl;

        imprimirArbol(raiz, 0); *// Imprimir el árbol después de la rotación*

        cout << endl;

        cout << "-----------------------------------------------------------------------" << endl;

        actualizarFactoresEquilibrio(raiz); *// Actualiza factores de equilibrio después de rotar*

    }

*// Caso III Izquierda-Derecha*

    else if (rec->factorEquilibrio < -1 && rec->izq && rec->izq->factorEquilibrio > 0)

    {

        cout << "Rotación Izquierda-Derecha en nodo: " << rec->dato << endl;

        rotarIzquierda(rec->izq); *// Primero rota a la izquierda el hijo izquierdo*

        rotarDerecha(rec); *// Luego rota a la derecha el nodo actual*

        cout << "CASO III: ARBOL RESULTANTE DESPUES DE ROTACION" << endl;

        imprimirArbol(raiz, 0); *// Imprimir el árbol después de la rotación*

        cout << endl;

        cout << "-----------------------------------------------------------------------" << endl;

        actualizarFactoresEquilibrio(raiz); *// Actualiza factores de equilibrio*

    }

*// Caso IV Derecha-Izquierda*

    else if (rec->factorEquilibrio > 1 && rec->der && rec->der->factorEquilibrio <= 0)

    {

        cout << "Rotación Derecha-Izquierda en nodo: " << rec->dato << endl;

        rotarDerecha(rec->der); *// Primero rota a la derecha el hijo derecho*

        rotarIzquierda(rec); *// Luego rota a la izquierda el nodo actual*

        cout << "CASO IV: ARBOL RESULTANTE DESPUES DE ROTACION" << endl;

        imprimirArbol(raiz, 0); *// Imprimir el árbol después de la rotación*

        cout << endl;

        cout << "-----------------------------------------------------------------------" << endl;

        actualizarFactoresEquilibrio(raiz); *// Actualiza factores de equilibrio*

    }

    return;

}

void rotarIzquierda(arbol \*&*rec*)

{

    if (rec == NULL || rec->der == NULL)

        return;

    arbol \*nuevoPadre = rec->der;

    rec->der = nuevoPadre->izq;

    nuevoPadre->izq = rec;

*// Actualiza factores de equilibrio solo en los nodos afectados*

    rec->factorEquilibrio = factorEquilibrio(rec);

    nuevoPadre->factorEquilibrio = factorEquilibrio(nuevoPadre);

    rec = nuevoPadre;

}

*// Rotación simple a la derecha*

void rotarDerecha(arbol \*&*rec*)

{

    if (rec == NULL || rec->izq == NULL)

        return;

    arbol \*nuevoPadre = rec->izq;

    rec->izq = nuevoPadre->der;

    nuevoPadre->der = rec;

*// Actualiza factores de equilibrio solo en los nodos afectados*

    rec->factorEquilibrio = factorEquilibrio(rec);

    nuevoPadre->factorEquilibrio = factorEquilibrio(nuevoPadre);

    rec = nuevoPadre;

}

void balancearHastaAVL(arbol \*&*raiz*)

{

    do

    {

        rotacion(raiz, raiz); *// Aplica rotaciones necesarias en todo el árbol*

        actualizarFactoresEquilibrio(raiz); *// Actualiza factores después de rotar*

    } while (!esAVL(raiz)); *// Repite hasta que el árbol sea AVL*

}