Examen Arboles 01/08/22

Construye una función que, dado un Árbol Binario, devuelva true si es un AVL y false en el caso contrario.

*Nota: definir tipos de datos y prototipos de las operaciones de los TADs usados.

Solución:

```cpp
#include "abin.hpp"
#include "abin_E-S.h"
#include <iostream>

using namespace std;

typedef int tElto;
const tElto fin = -1;
bool EsAbbRec(Abin<tElto>& A, Abin<tElto>::nodo n);
bool EsAvlRec(Abin<tElto>& A, Abin<tElto>::nodo n);
int altura(Abin<tElto>& A,Abin<tElto>::nodo n);

//----------------|| Aqui comienza el examen ||----------------

/*
    EsAvl -> Recibe un arbol binario, y devuelve true si es un Arbol Binario de Busqueda Equilibrado (AVL) o false si no
*/
bool EsAvl(Abin<tElto>& A)
{
    bool flag = EsAbbRec(A,A.raiz());   //un AVL debe ser un ABB
    flag = flag && EsAvlRec(A,A.raiz());    //y cumplir la condicion de estar equilibrado :)
    return flag;
}

/*
    EsAbbRec -> Recibe un arbol binario y un nodo de este, y devuelve true si es un Arbol Binario de Busqueda (ABB) o false si no (Version auxiliar recursiva)
*/
bool EsAbbRec(Abin<tElto>& A, Abin<tElto>::nodo n)
{
    if(A.hijoIzqdo(n) != Abin<tElto>::NODO_NULO && A.hijoDrcho(n) != Abin<tElto>::NODO_NULO)
    {
        return A.elemento(n) > A.elemento(A.hijoIzqdo(n)) && A.elemento(n) < A.elemento(A.hijoDrcho(n)) && EsAbbRec(A,A.hijoIzqdo(n)) && EsAbbRec(A,A.hijoDrcho(n));
    }

    if(A.hijoIzqdo(n) != Abin<tElto>::NODO_NULO && A.hijoDrcho(n) == Abin<tElto>::NODO_NULO)
    {
        return A.elemento(n) > A.elemento(A.hijoIzqdo(n)) && EsAbbRec(A,A.hijoIzqdo(n));
    }

    if(A.hijoIzqdo(n) == Abin<tElto>::NODO_NULO && A.hijoDrcho(n) != Abin<tElto>::NODO_NULO)
    {
        return A.elemento(n) < A.elemento(A.hijoDrcho(n)) && EsAbbRec(A,A.hijoDrcho(n));
    }

    if(A.hijoIzqdo(n) == Abin<tElto>::NODO_NULO && A.hijoDrcho(n) == Abin<tElto>::NODO_NULO)
    {
        return true;
    }
}

/*
    EsAvlRec -> Recibe un arbol binario y un nodo de este, y devuelve true si es un Arbol Binario de Busqueda Equilibrado (AVL) o false si no (version recursiva)
*/
bool EsAvlRec(Abin<tElto>& A, Abin<tElto>::nodo n)
{
    if(A.hijoIzqdo(n) != Abin<tElto>::NODO_NULO && A.hijoDrcho(n) != Abin<tElto>::NODO_NULO)
    {
        return abs(altura(A,A.hijoDrcho(n)) - altura(A,A.hijoIzqdo(n))) <= 1 && EsAvlRec(A,A.hijoIzqdo(n)) && EsAvlRec(A,A.hijoDrcho(n));
    }

    if(A.hijoIzqdo(n) != Abin<tElto>::NODO_NULO && A.hijoDrcho(n) == Abin<tElto>::NODO_NULO)
    {
        return abs(altura(A,A.hijoDrcho(n)) - altura(A,A.hijoIzqdo(n))) <= 1 && EsAvlRec(A,A.hijoIzqdo(n));
    }

    if(A.hijoIzqdo(n) == Abin<tElto>::NODO_NULO && A.hijoDrcho(n) != Abin<tElto>::NODO_NULO)
    {
        return abs(altura(A,A.hijoDrcho(n)) - altura(A,A.hijoIzqdo(n))) <= 1 && EsAvlRec(A,A.hijoDrcho(n));
    }

    if(A.hijoIzqdo(n) == Abin<tElto>::NODO_NULO && A.hijoDrcho(n) == Abin<tElto>::NODO_NULO)
    {
        return true;
    }
}


/*
    altura -> Recibe un arbol binario y un nodo de este, y devuelve la altura de dicho nodo en el arbol
*/
int altura(Abin<tElto>& A,Abin<tElto>::nodo n)
{
    if(n == Abin<tElto>::NODO_NULO)
    {
        return 0;
    }
    else
    {
        return 1 + max(altura(A,A.hijoIzqdo(n)),altura(A,A.hijoDrcho(n)));
    }
}

```

```cpp
/*
    DEFINICIONES:
    - A.raiz() -> Devuelve el nodo raiz del arbol (en caso de no tener devolveria NODO_NULO)
    - A.hijoIzqdo(n) -> Devuelve el nodo hijo izquierdo del nodo n del arbol (en caso de no tener devolveria NODO_NULO)
    - A.hijoDrcho(n) -> Devuelve el nodo hijo derecho del nodo n del arbol (en caso de no tener devolveria NODO_NULO)
    - A.elemento(n) -> Devuelve el elemento que posee el nodo n del arbol (Debe ser un nodo del arbol)
*/

//-----------------|| Aqui termina el examen ||-----------------

//Main con el objetivo de probar el programa y comprobar su funcionamiento
int main()
{
    Abin<tElto> A;

    cout << "*** Lectura del arbol binario A ***\n";

    rellenarAbin(A, fin); // Desde std::cin

    bool flag = EsAvl(A);

    if(flag)
    {
        cout << "El arbol es AVL" << endl;
    }
    else
    {
        cout << "El arbol no es AVL" << endl;
    }

    return 0;
}
```