ACT 3.4

Elías Uriel Velázquez Rojas _ A01639716

Actividad 3.4

Esta actividad fue un código hecho en equipo lo primero que realizamos fue el TreeNode.h que son los nodos del árbol

```
#pragma once
#include <iostream>
using namespace std;
struct bst {
    string dirIP = ""; //Valor de IP en string
    int accesos = 0; //Valor llave, numero de accesos de cada IP
};

class TreeNode
{
public:
    TreeNode(bst);
    void insert(bst);
    TreeNode* leftNode;
    bst data;
    TreeNode* rightNode;
};
```

Se crea una estructura bst para meter los valores de la IP en un tipo string y los accesos para cada ip, después creamos la clase TreeNode que nos va a representar los nodos del árbol, en publico esta su constructor y otra función para insertar los valores, los atributos un apuntador izquierda y derecha y un data de estructura bst.

```
#include "TreeNode.h"
TreeNode::TreeNode(bst nodeData)
{
    data = nodeData;
    leftNode = rightNode = NULL;
}

void TreeNode::insert(bst insertValue)
{
    if (insertValue.accesos < data.accesos) {
        if (leftNode == NULL)
            leftNode = new TreeNode(insertValue);
        else
            leftNode->insert(insertValue);
    }
    else if (insertValue.accesos > data.accesos) {
        if (rightNode == NULL)
            rightNode = new TreeNode(insertValue);
        else
            rightNode->insert(insertValue);
    }
    else if (insertValue.accesos == data.accesos && insertValue.accesos > 1) {
        if (rightNode == NULL)
            rightNode = new TreeNode(insertValue);
        else
            rightNode->insert(insertValue);
    }
}
```

El constructor se asigna el data el valor recibido y los apuntadores leftNode y rightNode se declaran como NULL, después la otra función es para insertar el valor recibido, primero se verifica si los accesos del valor recibido es menor a los accesos de data, después de esto se comprueba si el nodo izquierda esta vació si es asi se crea un nuevo apuntador con el valor recibido si la primera verificación no es verdadera se hace lo mismo pero con el apuntador hacia la derecha, la siguiente comprobación es la misma que la anterior pero se le agrega la verificación que el acceso del valor recibido mayor a uno.

```
#pragma once
#include "TreeNode.h"
class Tree
{
public:
    Tree();
    void insertNode(bst );
    void inorderTraversal();
private:
    TreeNode* root;
    void inorderHelper(TreeNode* ,int &);
};
```

Este nuevo programa es el Tree.h en publico se crea un constructor, dos funciones uno para insertar el nodo, y otro para imprimir llamado inorderTraversal, y en privado ser crea un apuntador la raíz, y inorderHelper.

```
Tree::Tree() {
    root = nullptr;
}

void Tree::insertNode(bst data) {
    if (root == NULL) {
        root = new TreeNode(data);
    }
    else {
        root->insert(data);
    }
}
```

```
void Tree::inorderTraversal()
{
    int limite = 5;
    cout << "ip con mayores accesos " << endl;
    inorderHelper(root, limite);
    cout << endl;
}
void Tree::inorderHelper(TreeNode* node,int &limit)
{
    if (node != NULL && limit > 0)
    {
        inorderHelper(node->rightNode, limit);
        if (limit > 0) {
            limit--;
            cout << " Ip: " << node->data.dirIP<<"Accesos: "<< node->data.accesos<< endl;
        }
        if (limit > 0) {
            inorderHelper(node->leftNode, limit);
        }
    }
}
```

Primero el constructor, se declara la raíz como nulo para después la función de insertar nodo, se recibe un valor con la estructura bst, primero se hace una verificación para ver si la raíz esta vacía, si es así se crea un nuevo apuntador con el valor recibido si esto no sucedía y se llamaba a la función insert del TreeNode, para insertar este valor.

La función InorderTraversal, esta función se manda a llamar la función inorderHelper que ayuda para imprimir lo que hace esta función es recibir un apuntador node y un valor entero llamado como limit referenciado, primero se hace la verificación si el nodo es diferente a nulo y el limite era mayor a 0, y si esta se cumple se llama recursivamente esta función pero el nodo se recorre hacia la derecha luego si el limite es mayor a 0 se imprime la dirección y los accesos de la ip , para después si también es mayor a 0 se llama recursivamente y se recorre hacia la izquierda ahora.

Luego nuevo código 3.4 cpp.

```
int ip2int(string ip) { //Regresa primer valor entero de IP
    int rst = 0;
    int a, b, c, d;
    char t = '.';
    stringstream ss(ip);
    ss >> a >> t >> b >> t >> c >> t >> d;
    rst = (a);
    return rst;
}

int ip2int2(string ip) { //Regresa segundo valor entero de IP
    int rst = 0;
    int a, b, c, d;
    char t = '.';
    stringstream ss(ip);
    ss >> a >> t >> b >> t >> c >> t >> d;
    rst = (b);
    return rst;
}
```

```
int ip2int3(string ip) { //Regresa tercer valor entero de IP
    int rst = 0;
    int a, b, c, d;
    char t = '.';
    stringstream ss(ip);
    ss >> a >> t >> b >> t >> c >> t >> d;
    rst = (c);
    return rst;
}

int ip2int4(string ip) { //Regresa cuarto valor entero de IP
    int rst = 0;
    int a, b, c, d;
    char t = '.';
    stringstream ss(ip);
    ss >> a >> t >> b >> t >> c >> t >> d;
    rst = (d);
    return rst;
}
```

Estas cuatro funciones son iguales, pero hacen algo diferente cada una de ellas regresa los valores , la 1 pues el primero y asi sucesivamente hasta llegar al 4, para esto se usa la librería stringstream.

```
void llenarArbol(string filename) { //Llena el arbol
    string linea;
    char delimitador = ' ';
    char delimitador2 = '\13';
    Tree tree;
    int dirIpN = 0, dirIpP = 0;
    int dirIpN2 = 0, dirIpP2 = 0;
    int dirIpN3 = 0, dirIpP3 = 0;
    int dirIpN4 = 0, dirIpP4 = 0;
    int n = 0;
    fstream archivo;
    archivo.open(filename);

    if (archivo.fail()) {
        cout << "Error al abrir archivo" << endl;
    }
```

Para terminar tenemos la función llenar árbol, que recibe un string llamado filename, lo que hace esta función es llenar el árbol con los datos del txt, primero genera un string que será la línea y dos delimitador, obviamente el árbol, y las direcciones de las ip se generan dos una con terminación N y otra con terminación p, luego se abre el archivo si este falla se menciona error al abrir archivo y no se corre el código

```
else {
    // Leemos todas las líneas
    while (getline(archivo, linea)) {
        stringstream stream(linea);  // Convertir la cadena a un stream
        string mes, dia, segundos, dirIP, razon, tiempo;
        // Extraer todos los valores de esa fila
        getline(stream, mes, delimitador);
        getline(stream, dia, delimitador);
        getline(stream, tiempo, delimitador);
        getline(stream, dirIP, delimitador);
        getline(stream, razon, delimitador2);
```

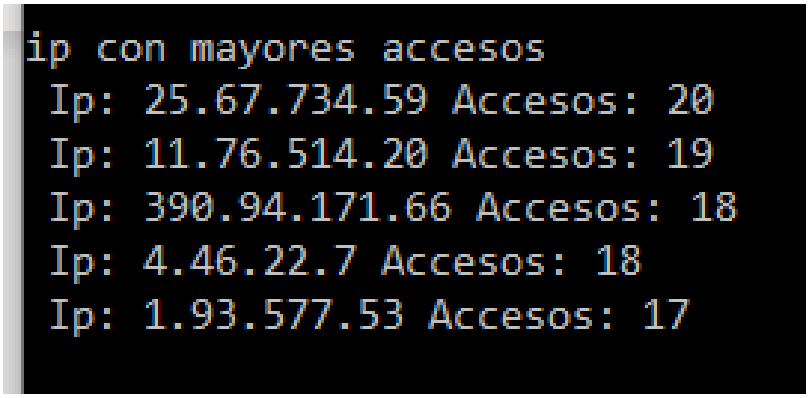
```
        //Se verifica si las IPs se repiten
        dirIpN = ip2int(dirIP);
        dirIpN2 = ip2int2(dirIP);
        dirIpN3 = ip2int3(dirIP);
        dirIpN4 = ip2int4(dirIP);

        if (dirIpP == 0 && dirIpP2 == 0 && dirIpP3 == 0 && dirIpP4 == 0) {
            dirIpP = dirIpN;
            dirIpP2 = dirIpN2;
            dirIpP3 = dirIpN3;
            dirIpP4 = dirIpN4;
        }
        if (dirIpN != dirIpP || dirIpN2 != dirIpP2 || dirIpN3 != dirIpP3 || dirIpN4 != dirIpP4 || archivo.eof()) {
            if (archivo.eof() && dirIpN == dirIpP && dirIpN2 == dirIpP2 && dirIpN3 == dirIpP3 && dirIpN4 == dirIpP4) {
                n++;
            }
            bst ipA;
            ipA.accesos = n;
            ipA.dirIP = to_string(dirIpP) + "." + to_string(dirIpP2) + "." + to_string(dirIpP3) + "." + to_string(dirIpP4) + " ";
            tree.insertNode(ipA);
            n = 0;
        }
        n++;
        dirIpP = dirIpN;
        dirIpP2 = dirIpN2;
        dirIpP3 = dirIpN3;
        dirIpP4 = dirIpN4;
    }

    }
    archivo.close();
    tree.inorderTraversal();
}
```

Luego se use las librerías stringstream y se declaran las variables en string, y se usa getline con los valores y los delimitadores, luego se verifica las IP se repiten, con las funciones que implementamos al principio, después se verifica si las IP son iguales a 0, se le asigna a las terminaciones P las terminación con N que se verificaron si no se repitieron, y luego una verificaciones que si es diferentes y luego se hace otra verificación si esto sucede se aumenta n, se crea con la estructura bst un ipA, que a sus accesos se les asigna el valor de N, y a las direcciones se asigna los valores de dirIp con terminación P, con la ayuda de to_string, luego al árbol se le inserta este nodo creado para después se le asigna a n el valor de 0 luego asignar dirIp con terminación P los de dirIp con terminación N. Al terminar todo esto, se cierra el archivo, y se imprime el árbol con la ayuda se inorderTraversal, para imprimir.

```
int main()
{
    Tree miArbol;
    string filename;
    filename = "archivo nuevo.txt";
    llenarArbol(filename);// Crear lista
    return 0;
}
```

En el main se crea un árbol llamado miArbol, un string llamado filename y se carga el archivo nuevo y se guarda en filename, se aplica la función llenarArbol y se retorna 0

CASOS PRUEBA 1:



Caso prueba 2: