**Instituto Tecnológico y de Estudios Superiores de Monterrey**

Tecnológico de Monterrey

# Programación de Estructuras de Datos y Algoritmos Fundamentales

Ivan Reyez Amezcua

Act 3.4 - Actividad Integral De BST (Evidencia Competencia)


Santos Alejandro Arellano Olarte // A01643742
Carlos Iván Armenta Naranjo // A01643070
Arturo Ramos Martínez //A01643269
Adair Virgilio Figueroa Medina //A00572826
Leonardo Mario Alberto Guillén Soria // A00574110
Daniela Rocha Muñoz //A00573664


08 de Noviembre del 2023

| | | |
|---|---|---|
| **push** | Descripción | Inserta en el BST un nuevo nodo, el cual ahora contiene 2 atributos nuevos: node* array[] y int puerto.<br><br>Este método es muy parecido a un push para un BST normal, lo único diferente es que se le agregan 2 procesos:<br><br>1. Un ciclo for antes de iniciar con la inserción normal:<br><br>Esto para controlar aquellos casos en los que el nodo que se intenta insertar, tiene el mismo puerto que alguno de los que hasta el momento tienen más accesos, en este caso solo se aumenta la variable “data” y se llama un “return;” para terminar la función.<br><br>2. Una condición más en el ciclo while de lo que sería la inserción normal de un BST:<br><br>Esto para nuevamente controlar cuando si, y cuando no mandar el nodo con puerto repetido al array de los puertos con más acceso. Esto es necesario, ya que no solo por repetirse el acceso al puerto, se debería intercambiar inmediatamente con alguno de los que se encuentran en el array. |
| | Entrada | int |
| | Salida | nada |

| | | |
|---|---|---|
| | Precondición | Que se haya iniciado una instancia de binarySearchTree |
| | Postcondición | En este caso existirían 3, que el nodo se haya insertado normalmente, que ya haya sido insertado con anterioridad y por tanto solo se le agrega un uno a su atributo “data”, y por último podría ser que ya se haya insertado tantas veces repetidas, que se tenga que agregar al “node* array[]”. |
| | Complejidad | En el mejor de los casos, sería de O(1) cuando se inserta en la raíz, pero dejando de lado ese caso que solo se presenta una vez, los casos promedio serían de O(log n), ya que en principio se seguiría conservando el mismo método que para un BST normal, sin embargo; no estamos balanceando el árbol, por lo que habría posibilidad de tener complejidad de O(n). Y claro, los ciclos extra que agregamos tendrán siempre complejidad de O(k) y k = 5, por qué estamos buscando solo los 5 puertos con más accesos. |
| **main** | Descripción | En esta parte del código lo que hacemos es leer las líneas de Texto del archivo resultante de la actividad anterior y una vez con ellos separamos los puertos y los parseamos para convertirlas en int, de ahí los almacenamos en el |

| | | |
|---|---|---|
| | | BST para después con un If verificar si ninguno se repite más de 1 vez indicar con un mensaje que no hay ninguna ip que tenga más de 1 acceso, de lo contrario se recorre el arreglo con los 5 con más acceso y se imprimen en pantalla. |
| | Entrada | Un archivo de texto llamado rango.txt |
| | Salida | La impresión de los 5 puertos con mas accesos |
| | Precondición | Nada |
| | Postcondición | Nada |
| | Complejidad | La complejidad de leer el archivo es lineal es decir 0(n) pero el push al BST es de complejidad Log(n) y el for que imprime los 5 puertos es constante |

***Investigacion y reflexion individual:***

Investiga y reflexiona sobre la importancia y eficiencia del uso de BST en una situación problema de esta naturaleza. ¿Cómo podrías determinar si una red está infectada o no?

***Santos Arellano:***
La detección de redes infectadas implica un análisis sutil de patrones. Al aprovechar la estructura de un BST, se pueden identificar conexiones inusuales o intentos repetidos, proporcionando una alerta temprana ante posibles actividades maliciosas. Además, el monitoreo de la frecuencia de accesos a puertos específicos se simplifica con un BST, convirtiéndose en un indicador potente de una posible infección.

La evaluación multinivel, lograda a través de un recorrido por niveles del BST, facilita la identificación de comportamientos sospechosos en diferentes estratos de la red. Sin embargo, estos beneficios no están exentos de desafíos. El mantenimiento continuo del árbol es esencial para asegurar su eficiencia en respuesta a la evolución de amenazas y patrones de acceso.

Integrar la potencia de los BST con otras técnicas, como el análisis de tráfico y el aprendizaje automático, fortalece significativamente la capacidad de detección. Además, la gestión adecuada de registros de acceso en conformidad con las regulaciones de privacidad y requisitos legales es un factor crítico a considerar.

***Daniela Rocha:***

Estamos en un mundo donde tus datos y la seguridad de tu red son de máxima importancia. Aquí es donde entran en juego los Árboles de Búsqueda Binaria (BST). No se trata solo de teoría, sino de habilidades prácticas que fortalecerán nuestra preparación para desafíos del mundo real. Los BST marcan la diferencia entre la seguridad y la vulnerabilidad ante ciberamenazas. Como estudiante de Ingeniería en Tecnologías Computacionales, comprender su importancia es muy importante.

Estos árboles tienen la capacidad de agilizar búsquedas y reducir la complejidad temporal, lo cual es invaluable en el contexto de la detección de redes infectadas. Pero, claro, está eficiencia no es inmune a las disrupciones que una infección puede imponer. Una red infectada en tu sistema, a simple vista no la ves, pero está ahí. Altera flujos de datos, patrones de acceso y hasta la estructura misma del árbol. Son cosas tan sutiles que requieren de una observación y comprensión aguda y a detalle. Para identificar una red infectada, observa anomalías en el tráfico de datos, una disminución en el rendimiento, patrones de acceso alterados y cambios en la estructura del árbol. Estos signos, aunque son evidentes, son algo más a profundidad que requiere una investigación a fondo.

La amenaza de redes infectadas es una manifestación de la evolución de la ciberdelincuencia y la búsqueda de vulnerabilidades en sistemas informáticos. Los ataques pueden surgir de diversas fuentes, ya sea a través de malware y virus infiltrados, intrusos malintencionados que explotan vulnerabilidades, o incluso errores de configuración que exponen la red a amenazas.

La defensa efectiva contra redes infectadas son una combinación, desde una estrategia de seguridad multicapa hasta un monitoreo constante de la red, pasando por la actualización y parcheo de sistemas, la educación y concientización de los usuarios y un plan de respuesta a incidentes bien estructurado, cada una de estas estrategias se conectan para proporcionar una defensa completa contra las amenazas cibernéticas.

***Carlos Armenta:***
*En la actualidad existen grandes flujos de datos en la red 6 la seguridad de estos es de suma importancia,  por lo que mantener un monitoreo en la actividad y el número de accesos que se tiene por puerto es importante para garantizar que la privacidad y la confiabilidad de la red siga siendo óptima, es aquí donde nos podemos dar cuenta de que el uso de estructuras de datos como el Binary Seach Tree nos permite mantener un control del número de accesos que se tiene por puerto a la red, como lo hicimos en esta actividad en la cual a través de esta estructura podemos llegar a obtener los puertos de una red con más acceso y esto con una complejidad de Log(n) lo que es muy beneficioso si se piensa a gran escala ya que en una red muy grande la cantidad de puertos y de direcciones Ip puede ser muy extensa y el conteo y la búsqueda de los mismos se puede llegar a complicar, pero con ayuda de un BST, se puede garantizar una búsqueda óptima y un monitoreo que nos ayude a vigilar los accesos de nuestra red y poder detectar un ataque en cuanto antes para una mayor seguridad de red y de manejo de datos.*

***Arturo Ramos Martínez:***

Los usuarios de hoy día exigimos mucho, tanto velocidad como seguridad y funcionalidad impecable, claramente con el nivel de tecnología que ya existe, esto es prácticamente pan de cada día, por eso la importancia de comprender por lo menos con un cierto nivel de abstracción, la manera en que se manejan estos sistemas tan eficientes y seguros. El árbol de búsqueda binario es bien conocido por su gran velocidad de búsqueda, inserción y eliminación, y es que sus complejidades (siempre y cuando el árbol se encuentre balanceado) en estos procesos son de O(log n), la cual es muy eficiente y recomendable a la hora de tratar con casos donde la información de una empresa o de alguna persona se pone en peligro, y que tener un sistema para actuar rápidamente podría ser crucial.

Ahora, para responder a la pregunta: “¿Cómo podrías determinar si una red está infectada o no?” Creo que tomando en cuenta la actividad y su propósito, la respuesta recaería en que probablemente la red está siendo atacada si de pronto se reconocen mucha cantidad de accesos en poco tiempo. Nos podríamos dar cuenta fácilmente que algo raro pasa si conocemos más o menos el número de accesos promedio, y que de un momento a otro, este número incremente desmedidamente.

***Adair Figueroa:***
Un Binary Search Tree o BST es una estructura de datos utilizada comúnmente en la ciencia de la computación para organizar y buscar datos de una manera eficiente. Aunque estas no siempre son la mejor manera para detectar datos por su eficiencia, muchas veces desempeña un papel importante en ciertos aspectos de la detección de amenazas cibernéticas.

Muchas veces se requieren enfoques más avanzados para encontrar amenazas avanzadas, tal como un aprendizaje automático. Pero, un BST es más ideal para buscar datos existentes que para analizar flujos de datos reales.

En conclusión, aunque los BST pueden contribuir a la organización de datos en el ámbito de la ciberseguridad, la detección efectiva de infecciones en una red requiere enfoques más avanzados y diversas estrategias. La combinación de técnicas como el análisis de comportamiento, el aprendizaje automático y la monitorización en tiempo real del tráfico de red es esencial para abordar la complejidad de las amenazas cibernéticas en constante evolución.

***Leonardo Guillén:***
Actualmente, los datos y el manejo de ellos en la red, es muy importante para todos, ya que, se utilizan para todas las actividades que hacemos en nuestro día a día al conectarnos a internet, es por ello que la seguridad de los datos es uno de los aspectos más importantes.

Constantemente se ejecutan ataques a servidores o dispositivos en específico por medio de la red y es necesario darnos cuenta de que conexiones podrían llegar a infectarse. Aquí es donde entran los árboles binarios, ayudándonos a tener una mayor facilidad, agilidad, eficiencia y manejo de los datos, en este caso en específico nos ayudará a identificar redes que sean sospechosas víctimas, esto se da gracias a identificar si cierto puerto tiene una cantidad exagerada de conexiones, esto sería más sospechoso aún si es una red abierta.

Sabemos que el uso de un BST para lograr identificar si una red se encuentra infectada o no, puede ser muy eficiente y útil, al tomar como consideración los números de acceso y puertos de la red.

Es por ello que, en este caso, lo aplicaremos para identificarlas redes. A pesar de esto, para que logre tener eficiencia depende de una correcta implementación y su uso específico. Lo más fundamental es la eficiencia en la búsqueda, así como los análisis de los patrones que se den usualmente en la red, ya que con esto se podrían reconocer comportamientos inusuales.

La forma en la que se tendría identificada alguna sería el número de accesos en un mismo puerto, porque si este es elevado, es muy probable que esté siendo infectado.
Aplicar un reconocimiento de este tipo con árboles binarios es muy bueno, porque estos tiene un manejo de los registros rápido y efectivo, depende de cómo sea aplicado, pero es una de las mejores opciones para un caso como este, en el que se busca identificar el número de accesos en un puerto.

Gracias a esto, una vez se haya identificado, se pueden tomar las medidas necesarias para tener una red segura en todo momento y proteger los datos que se trasladen por ahí.

# *Código:*

**binarySearchTree.h**

```
#include <iostream>
using namespace std;
struct node{
    public:
        int data;
        int puerto;
        node* right;
        node* left;
        node(){
            data = 1;
            right = nullptr;
            left = nullptr;
        }
};

class BinarySearchTree{
public:
   node *raiz;
   int altura;
   node* puertos[5];
   BinarySearchTree();
   void push(int num);
```

```
    void preOrder(node* root);
    void inOrder(node* root);
    void postOrder(node* root);
    void levelByLevel(node* root,int num);
    void visit(BinarySearchTree* arbolito);
    int height(node* root);
};
```

## binarySearchTree.cpp

```
#include "binarySearchTree.h"

BinarySearchTree::BinarySearchTree(){
    altura=0;
    raiz=nullptr;
    puertos[0]=nullptr;
    puertos[1]=nullptr;
    puertos[2]=nullptr;
    puertos[3]=nullptr;
    puertos[4]=nullptr;
}
void BinarySearchTree::push(int num) {
    node* nodo = new node;
    nodo->puerto = num;

    for (int i = 0; i <= 4; i++) {
        if (puertos[i] != nullptr && puertos[i]->puerto == num) {
            puertos[i]->data++;
            return;
        }
    }

    if (raiz == nullptr) {
        raiz = nodo;
        altura += 1;
        puertos[0] = nodo;
    }else {
        node* ptr = raiz;
        while (ptr != nullptr) {
            if (nodo->puerto < ptr->puerto && ptr->left == nullptr) {
                ptr->left = nodo;
                return;
```

```
            } else if (nodo->puerto > ptr->puerto && ptr->right ==
nullptr) {
                ptr->right = nodo;
                return;
            } else if (nodo->puerto == ptr->puerto) {
                ptr->data++;
                for (int i = 0; i <= 4; i++) {
                    if (puertos[i] == nullptr || ptr->data >
puertos[i]->data) {
                        puertos[i] = puertos[i - 1];
                        puertos[i] = ptr;
                        break;
                    }
                }
                return;
            } else if (nodo->puerto < ptr->puerto) {
                ptr = ptr->left;
            } else{
                ptr = ptr->right;
            }
        }
    }
}


void::BinarySearchTree::preOrder(node* root){
    if(root==nullptr){
        return;
    }
    cout<<root->data<<" ";
    cout<<root->puerto<<" ";
    preOrder(root->left);
    preOrder(root->right);
}

void::BinarySearchTree::inOrder(node* root){
    if (root == nullptr){
        return;
    }
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}
```

```
void::BinarySearchTree::postOrder(node* root){
    if (root == nullptr){
        return;
    }
    postOrder(root->left);
    postOrder(root->right);
    cout << root->data << " ";
}

void BinarySearchTree::levelByLevel(node* root, int level) {
    if (root == nullptr) {
        return;
    }

    if (level == 1) {
        cout << root->data << " ";
    }
    else if (level > 1) {
        levelByLevel(root->left, level - 1);
        levelByLevel(root->right, level - 1);
    }
}

void::BinarySearchTree::visit(BinarySearchTree* arbolito){
    int op=0;
    cout<<"Introduce un numero entero del 1 al 4 para decir como
quieres que muestre el BST"<<endl;
    cout<<"1. Preorder"<<endl;
    cout<<"2. Inorder"<<endl;
    cout<<"3. Postorder"<<endl;
    cout<<"4. Level by level"<<endl;
    cin>>op;
    if(op==1){
        cout<<"Opcion seleccionada (1. Preorder)"<<endl;
        arbolito->preOrder(arbolito->raiz);
        cout<<endl;

    }else if(op==2){
        cout<<"Opcion seleccionada (2. Inorder)"<<endl;
        arbolito->inOrder(arbolito->raiz);
        cout<<endl;
```

```
    }else if(op==3){
        cout<<"Opcion seleccionada (3. Postorder)"<<endl;
        arbolito->postOrder(arbolito->raiz);
        cout<<endl;
    }else if(op==4){
        cout<<"Opcion seleccionada (4. Level by level)"<<endl;
        for (int i = 1; i < arbolito->height(arbolito->raiz); i++) {
            cout << "Nivel " << i << ": ";
            arbolito->levelByLevel(arbolito->raiz, i);
            cout << endl;
        }
    }else{
        cout<<"Esa no es una opcion disponible"<<endl;
    }
}

int BinarySearchTree::height(node* root) {
    if (root == nullptr) {
        return 0;
    } else {
        int leftHeight = height(root->left);
        int rightHeight = height(root->right);
        altura = max(leftHeight, rightHeight)+1;
        return altura;
    }
}
```

## main.cpp

```
#include <iostream>
#include <sstream>
#include <fstream>
#include "binarySearchTree.cpp"


string extractIp(string line){
    string mes, dia, hora, ip;
    stringstream extract(line);
    extract >> mes >> dia >> hora >> ip;
    return ip;
}
```

```cpp
int main(){
    ifstream bitacora("../rango.txt");
    string line, ip, puerto;
    BinarySearchTree arbolito;
    while(getline(bitacora, line)){
        ip = extractIp(line);
        size_t dos_puntos = ip.find(':');
        if (dos_puntos != string::npos) {
            puerto = ip.substr(dos_puntos+1);
            ip = ip.substr(0, dos_puntos);
            int puertito = stoi(puerto);
            arbolito.push(puertito);
        }
    }
    if(arbolito.puertos[0]->data == 1){
        cout << "Ningun puerto es accesado más de una vez" << endl;
        return 0;
    }
    for(int i = 0; i <= 4 ;i++){
        cout <<"Puerto: "<<arbolito.puertos[i]->puerto << " accesos:
"<<arbolito.puertos[i]->data<< endl;
    }
    return 0;
}
```