INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Act 3.4 - Actividad Integral de BST (Evidencia Competencia)

Daniel Esparza Arizpe - A01637076

Diego Alberto Cisneros Fajardo - A01643454

Luis Fernando Li Chen - A00836174

Sadrac Aramburo Enciso - A01643639

Diego Michell Villa Durán - A00836723

Programación de estructuras de datos y algoritmos fundamentales

Grupo 605

Tecnológico de Monterrey

Jueves 11 de noviembre 2023

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>
#include <set>
#include <vector>
using namespace std;

struct LogEntry {
    string date;
    string time;
    string ip;
    string port;
};

struct Node {
    int accesses;
    string port;
    Node* left;
    Node* right;

    Node(int accesses, const string& port) : accesses(accesses),
port(port), left(nullptr), right(nullptr) {}
};

void insert(Node*& root, const string& port) {
    if (!root) {
        root = new Node(1, port);
        return;
    }

    if (port < root->port) {
        insert(root->left, port);
```

```
    } else if (port > root->port) {
        insert(root->right, port);
    } else {
        // Incrementa el cont
        root->accesses++;
    }
}

void processLogFile(const string& filename, Node*& root) {
    ifstream inputFile(filename);
    string line;

    while (getline(inputFile, line)) {
        istringstream iss(line);
        LogEntry entry;
        iss >> entry.date >> entry.time >> entry.ip;

        iss.ignore(256, ':');  // Se salta todo hasta ':'
        iss >> entry.port;

        insert(root, entry.port);  // Incrementa cont accesos
    }

    inputFile.close();
}

void inOrderTraversal(Node* root, multiset<pair<int,
string>>& result) {
    if (root) {
        inOrderTraversal(root->left, result);
        result.insert({root->accesses, root->port});
        inOrderTraversal(root->right, result);
    }
}
```

```
int main() {
    Node* root = nullptr;
    processLogFile("bitacora.txt", root);

    // Busca los 5
    multiset<pair<int, string>> sortedPorts;
    inOrderTraversal(root, sortedPorts);

    // Top 5
    ofstream outputFile("bitacorafixed.txt");
    int count = 0;

    for (auto it = sortedPorts.rbegin(); it != sortedPorts.rend()
&& count < 5; ++it) {
        outputFile << "Port: " << it->second << ", Accesses: " <<
it->first << endl;
        count++;
    }

    delete root;

    outputFile.close();

    return 0;
}
```

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Complejidad

Inserción en el árbol (BST):

La inserción en un árbol de búsqueda binaria puede tener una complejidad de O(h), donde h es la altura del árbol. En el peor escenario, si el árbol no está balanceado, podría degenerar en una lista ligada, y h sería igual al número de nodos en el árbol. Sin embargo, si el árbol está balanceado, la altura sería O(log n), donde n es el número de nodos en el árbol en el peor de los casos.

Recorrido Inorden del Árbol (BST)

Recorrer un árbol de búsqueda binaria mediante un recorrido inorden tiene una complejidad de O(n), donde n es el número de nodos en el árbol.

Lectura del Archivo y Procesamiento

La lectura del archivo y el procesamiento línea por línea tiene una complejidad de O(m), donde m es el número de líneas en el archivo.

Ordenamiento del Conjunto (multiset)

La inserción y el ordenamiento de elementos en un multiset tienen una complejidad de O(log k) por elemento, donde k es el número de elementos en el conjunto.

Casos de prueba

```cpp
int main() {
    Node* root = nullptr;
    processLogFile("bitacora.txt", root);

    // Busca los 5
    multiset<pair<int, string>> sortedPorts;
    inOrderTraversal(root, sortedPorts);

    // Top 5
    ofstream outputFile("bitacorafixed.txt");
    int count = 0;

    for (auto it = sortedPorts.rbegin(); it != sortedPorts.rend() && count < 5; ++it) {
        outputFile << "Port: " << it->second << ", Accesses: " << it->first << endl;
        count++;
    }

    delete root;

    outputFile.close();

    return 0;
}
```

```
Port: 6170, Accesses: 16
Port: 5525, Accesses: 15
Port: 6445, Accesses: 14
Port: 6195, Accesses: 14
Port: 5504, Accesses: 14
```

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Reflexiones

Daniel Esparza Arizpe:

Un BST es un tipo de árbol binario en el que cada nodo tiene, a lo sumo, dos hijos (izquierdo y derecho), y para cada nodo, todos los elementos en su subárbol izquierdo son menores que el nodo, y todos los elementos en su subárbol derecho son mayores. Esto facilita la búsqueda eficiente de elementos en el árbol.

Los Sistemas de Detección de Intrusiones basados en comportamiento BST son muy importantes para detectar amenazas en las direcciones IP de un registro. Su enfoque en el análisis del comportamiento de la red permite la identificación de patrones de tráfico anómalos, la adaptación a nuevas amenazas y la reducción de falsos positivos (Lim, 2013). La rápida detección, la minimización de daños y el análisis contextual demuestran la eficacia de los BST. En este contexto, la determinación de una infección de una red se logra mediante la identificación de anomalías en el tráfico, el monitoreo de la actividad de usuario, el análisis de protocolos y la integración con otras soluciones de seguridad para proporcionar respuestas automatizadas y una visión completa del panorama de amenazas en las direcciones IP del registro.

Sadrac Aramburo Enciso:

El binary search tree (BST) es una estructura de datos que organiza los datos de manera jerárquica, lo que puede ser de gran importancia al procesar grandes cantidades de datos. Importancia de BST:

Los BST son estructuras de datos eficientes para organizar y buscar información de manera rápida. En el contexto de la búsqueda de direcciones IP y puertos, los BST pueden utilizarse para almacenar y ordenar información relevante, permitiendo una búsqueda rápida y eficiente. Dada la gran cantidad de datos que se manejan en redes, la capacidad de búsqueda rápida es esencial para detectar y responder a posibles amenazas de manera oportuna.

Eficiencia en la Búsqueda de IP y Puertos:

La búsqueda de direcciones IP y puertos generalmente implica comparar y verificar si una determinada combinación se encuentra en la red. Los BST ofrecen un tiempo de búsqueda promedio de O(log n), lo que significa que la eficiencia no se degrada significativamente incluso para grandes conjuntos de datos. Esto es crucial cuando se intenta determinar rápidamente si una dirección IP o un puerto específico está asociado con actividades maliciosas.

Cómo Determinar si una Red está Infectada:

Utilizando BST, es posible implementar algoritmos de búsqueda que permitan detectar patrones específicos asociados con infecciones conocidas. Por ejemplo, se pueden construir BST para almacenar direcciones IP y puertos relacionados con malware conocido. La detección de una coincidencia en el BST indicaría la presencia de una amenaza.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Diego Alberto Cisneros Fajardo:

Un binary Search Tree es una estructura de datos que Un BST es eficiente para organizar y buscar datos ordenados, en este caso, los puertos utilizados en la red.

Al organizar los puertos en un árbol binario, puedes realizar búsquedas y actualizaciones de manera rápida, ya que el tiempo promedio de búsqueda en un BST es O(log n).

Para cada nodo en el árbol, todos los nodos en su subárbol izquierdo tienen valores menores, y todos los nodos en su subárbol derecho tienen valores mayores.

Eficiencia en Búsquedas:

Esta propiedad de ordenamiento facilita la búsqueda de un elemento específico en el árbol de manera eficiente. El tiempo promedio de búsqueda en un BST es proporcional al logaritmo del número de nodos en el árbol.

Luis Fernando Li Chen:

El uso de estructuras de datos como los árboles binarios de búsqueda (BST) nos muestra cómo la elección adecuada puede impactar significativamente en la eficiencia y la resolución de problemas. Este código, al implementar un BST para almacenar registros de accesos por IP, ilustra la versatilidad y la eficacia de esta estructura en la gestión de datos. Por lo que, el uso de árboles y la revisión de las IP más utilizadas es esencial en la administración de redes y la ciberseguridad para optimizar el rendimiento de la red, detectar amenazas, garantizar la eficiencia en el uso de recursos y mantener la seguridad de la red en general. En conclusión, nos muestra que en el mundo digital, el conocimiento y el control de las IP son elementos clave para mantener nuestras redes seguras y funcionando sin problemas.

Los BST se utilizan comúnmente en la implementación de estructuras de datos como diccionarios, conjuntos y mapas, ya que proporcionan un acceso eficiente y rápido a los datos ordenados. Además, son útiles en la resolución de problemas que involucran búsquedas, inserciones y eliminaciones

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Diego Michell Villa Durán:

El uso de un BST para analizar registros de acceso a puertos sinceramente ofrece varias ventajas ya que en términos de eficiencia y estructuración de datos. La elección de esta estructura de datos se basa en su capacidad para ordenar y organizar la información de manera eficiente.El uso de un BST en este contexto destaca la importancia de seleccionar estructuras de datos adecuadas para el análisis de registros de acceso. Pero también considero que es igualmente importante considerar otras estrategias de seguridad y abordar la seguridad de la red de manera holística para proteger eficazmente contra amenazas potenciales.

Referencias

Lim, H. (2013). *US20050083937A1 - IP address lookup method using pipeline binary tree, hardware architecture, and recording medium - Google Patents*. https://patents.google.com/patent/US20050083937A1/en