Instituto Tecnológico y de Estudios Superiores de Monterrey

Programación de estructuras de datos y algoritmos fundamentales (Gpo 602)

Act 3.4 - Actividad Integral de BST

Adela Alejandra Solorio Alcázar A01637205
Luz Patricia Hernández Ramírez A01637277
Marcela Beatriz De La Rosa Barrios A01637239

01/10/2023

Act 3.4 - Actividad Integral de BST (Evidencia Competencia)

Las instrucciones para la evidencia competencia fueron las siguientes:

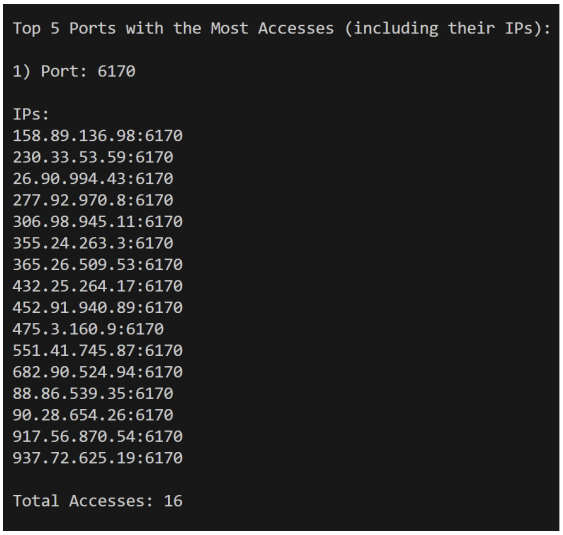
- En equipos de tres personas, hacer una aplicación que: Abra el archivo de entrada que fue el resultado de la actividad integradora anterior (ordenadas.txt) y que agregue todos los accesos por ip, los cuales se deben de almacenar en una estructura tipo BST dónde la llave es el número de accesos y el valor es la dirección IP.
- Encuentran los cinco puertos con más accesos.

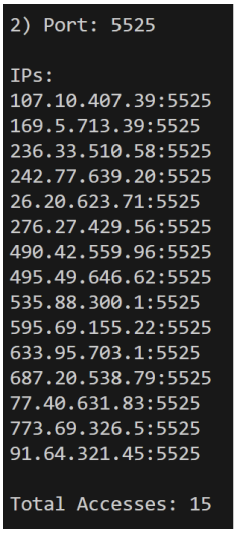
Código implementado:

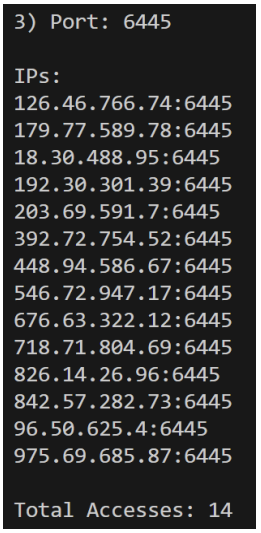
```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <queue>
#include <map>
#include <algorithm>
using namespace std;

struct Node {
  int port;  //key
  vector<string> ipAdd; //value
  int count; //value (number of accesses)
  Node *left, *right;
  Node(int& p) : port(p), count(1),
left(nullptr), right(nullptr) {}
};

class BST {
  private:
      Node *Root;
      void Insert(int&, string&, Node*&);
      Node* Search(Node*, int);

    public:
      //Constructor
      BST(): Root(NULL) {}
      //Destructor
      //~BST(){
        //DeleteBST(Root);
        //cout << "\nDestructor: BST
eliminado\n";
      //}
    void Insert(int &port, string
&ipAddress) {Insert(port, ipAddress,
Root);};
    void PrintNode(Node*);
    void PrintBST();
    Node* Search(int);
    //void DeleteBST(Node*&);
};
//Insert
void BST::Insert(int &port, string
&ipAddress, Node*& currentNode) {
  if(currentNode == NULL){
    currentNode = new Node (port);
    currentNode -> count = 1;
    currentNode ->
ipAdd.push_back(ipAddress);
  } else {
    if(port < currentNode->port){
      Insert(port, ipAddress,
currentNode->left);
    } else if(port > currentNode->port){
      Insert(port, ipAddress,
currentNode->right);
    } else {
      currentNode -> count += 1;
      currentNode ->
ipAdd.push_back(ipAddress);
    }
  }
}
//Print Node
void BST::PrintNode(Node* currentNode) {
  if(currentNode != NULL){
    int i = 0;
    cout << "Port: " << currentNode->port <<
endl;
    cout << "Number of accesses: " <<
currentNode->count << endl;
    cout << "IP Addresses: ";
    while(i < currentNode->ipAdd.size()){
      cout << currentNode->ipAdd[i] << " ";
      i++;
      if(i == currentNode->ipAdd.size()){
        cout << endl;
      }
    }
  } else {
    cout << "El nodo ingresado no existe."
<< endl;
  }
}
//Print BST
void BST::PrintBST() {
  if(Root == NULL){
    cout << "No se encontraron datos"
<<endl;
    return;
  }
  queue<Node*> Q;
  Q.push(Root);
  Node *Aux;
```

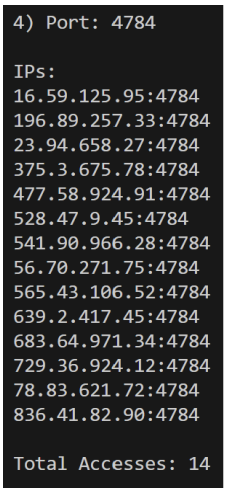
```
  while(!Q.empty()){
    Q.push(NULL);
    Aux = Q.front();
    while(Aux != NULL){
      PrintNode(Aux);
      if(Aux->left != NULL){
        Q.push(Aux->left);
      }
      if(Aux->right != NULL){
        Q.push(Aux->right);
      }
      Q.pop();
      Aux = Q.front();
    }
    Q.pop();
    cout << endl;
  }
}
//Compare count (number of accesses)
bool CompareCount(pair<int, Node*>& a,
pair<int, Node*>& b){
  return a.second->count > b.second->count;
}
//Search
Node* BST::Search(int portNum) {
    return Search(Root, portNum);
}

Node* BST::Search(Node* currentNode, int
portNum) {
    if (currentNode == nullptr ||
currentNode->port == portNum) {
        return currentNode;
    }
    if (portNum < currentNode->port) {
        return Search(currentNode->left,
portNum);
    } else {
        return Search(currentNode->right,
portNum);
    }
}
int main() {
  //Abrir archivo
  ifstream inputFile("ordenadas.txt");
  if (!inputFile.is_open()) {
    cerr << "Error: No se pudo abrir el
archivo." << endl;
    return 1;
  }
  //Crear el árbol
  BST Tree;
  map<int, Node*> portMap;
  string mes, fecha, ipAddress, razon;
  int dia, portNum;
  while (inputFile >> mes >> dia >> fecha >>
ipAddress) {
    getline(inputFile, razon);
    razon = razon.substr(1);
    portNum =
stoi(ipAddress.substr(ipAddress.find(":") +
1));
    Tree.Insert(portNum, ipAddress);

        if (portMap.find(portNum) ==
portMap.end()) {
            // If it doesn't exist, create a
new entry
            portMap[portNum] =
Tree.Search(portNum);
        }
  }
  Tree.PrintBST();
  cout << endl;
  cout << "Top 5 Ports with the Most
Accesses (including their IPs):" << endl;
  //Five ports with most accesses
  vector<pair<int, Node*>>
topPorts(portMap.begin(), portMap.end());
  sort(topPorts.begin(), topPorts.end(),
&CompareCount);
  for (int i = 0; i < 5 && i <
topPorts.size(); ++i) {
    pair<int, Node*> topPort = topPorts[i];
    cout<<"\n"<<i+1<<") ";
    cout << "Port: " << topPort.first <<
endl<<"\nIPs: "<<endl;
    for (const string& ip :
topPort.second->ipAdd) {
      cout << ip << " "<< endl;
    }
    cout << "\nTotal Accesses: " <<
topPort.second->count << endl;
  }
// Summarized output of found ports
cout << "\nTop 5 Ports with the Most
Accesses (Summarized):" << endl;
for (int i = 0; i < 5 && i <
topPorts.size(); ++i) {
    pair<int, Node*> topPort = topPorts[i];
    cout << i + 1 << ") ";
    cout << "Port: " << topPort.first;
    cout << ", Total Accesses: " <<
topPort.second->count << endl;
}
  inputFile.close();
  return 0;
}
```

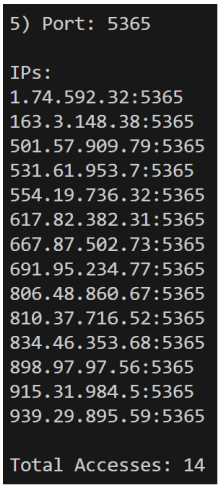
Salida del código:

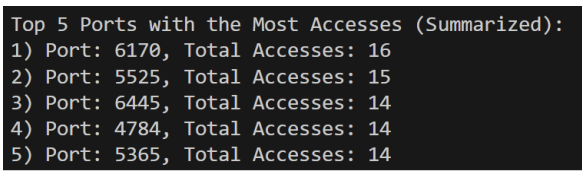












Reflexión individual:

Marcela de la Rosa: Al hacer uso de un Binary Search Tree (BST) para analizar los registros de acceso de red de un archivo de texto se pudieron observar ventajas y limitaciones considerables, es por esto que varios de los puntos más importantes al trabajar con un BST se pudieron observar a través de la implementación de la actividad, donde el almacenamiento de los datos siempre fue ordenado y permitió una búsqueda y recuperación de información más eficiente (como por ejemplo, en este caso se almacenaron las IP y el número de accesos por orden). Además, gracias a su complejidad promedio de O(log n) las búsquedas son mucho más rápidas que una lineal.

En base a esto, la pregunta que se realiza para la reflexión: ¿cómo podrías determinar si una red está infectada o no? Se puede contestar tocando algunos puntos, como el poder identificar los cinco puertos con más accesos, para compararlos con

puertos que normalmente permanecen inactivos, ya que si estos comienzan a activarse más que los que frecuentemente están activos, se puede deducir que hay actividad sospechosa (como un intento de ataque o un virus). Además, se podrían analizar las IP que acceden a la red para detectar si alguna de ellas es desconocida o no tiene sentido que esté activa en el sistema. Por último, se podría hacer algún tipo de investigación para examinar patrones de acceso que no tengan sentido, como el de repente tener muchos accesos desde un solo IP, lo cual puede indicar actividad maliciosa.

Adela Solorio: La resolución de problemas de este tipo por medio de Binary Search Trees es conveniente por la eficiencia que el almacenar los datos ya de manera organizada proporciona, esto permite que el buscar la llave (port) que más se repita (como en este caso) o cualquier otra información sea menos complejo, también, el que la información completa de cada Port pueda estar almacenada en un solo nodo hace mucho más sencillo obtenerla y, por lo tanto, buscarla.

Se podría determinar si una red está infectada o no analizando el tráfico de esa red, esto pudiendo ser con las direcciones IP de los usuarios; “el tráfico normal suele ser muy complejo, diverso y cambiante” (Venosa, 2021), entonces si se tiene una repetición constante o anómala se debe tener alerta y debe ser investigada, esto además de otras diversas técnicas de seguridad que existen y pueden ser implementadas.

Luz Patricia: Entre los diversos usos del árbol de búsqueda binaria (BST) se encuentran las tablas de enrutamiento. Estas tablas siguen el principio de organización jerárquica de la información para determinar las mejores rutas de conexión a la red. Además de ellas, las direcciones IP son esenciales para generar este enrutamiento del tráfico de red. En casos como estos que se manejan datos masivos, la implementación de estas estructuras de datos es de gran ayuda dado que cada elemento consta de múltiples factores que al implementar nodos simplifican la organización de la información. Además, la mayor ventaja que los BST tiene es su eficacia de búsqueda que cuenta con una complejidad de O(nlogn). Esta característica es muy útil en la seguridad de la red porque puede identificar rápidamente el malware que se está propagando por Internet mediante visitas sospechosas a puertos y ayudarlo a reaccionar rápidamente para proteger su sistema.

Referencias

Venosa, P. (2021). Detección de ataques de seguridad en redes usando técnicas de ensembling. SEDICI. Recuperado de http://sedici.unlp.edu.ar/handle/10915/120856