Emiliano Cabrera - A01025453
Programacion de estructuras de datos y algoritmos fundamentales
11 de octubre del 2021

Reflexión

Preguntas

```
PS C:\Users\GlowingMan\Documents\vscode\c++\sem3\act2_2> .\a.exe 10
1.- Que direccion ip estas usando?
IP: 192.168.29.10
2.- Cual fue la direccion IP de la ultima conexion que recibio esta computadora? Es interna o externa?
IP: 192.168.29.91       Interna
3.- Cuantas conexiones entrantes tiene esta computadora?
Conexiones entrantes: 2
4.- Cuantas conexiones salientes tiene esta computadora?
Conexiones salientes: 380
5.- Tiene esta computadora 3 conexiones seguidas a un mismo sitio web?
Si hay una conexion seguida a 195.86.236.187
```

1.- 192.168.29.10
2.- 192.168.29.91, Interna
3.- Tiene 2 entrantes
4.- Tiene 380 salientes
5.- Hay una conexión repetida a la IP: 195.86.236.187

Uso de estructuras lineales

El uso de estructuras lineales permite ordenar los datos de manera utilizando algún criterio específico, en este caso de manera cronológica. Esto permite obtener los datos más significativos sin la necesidad de iterar a lo largo de una lista indexada. El único problema es que la imposibilidad de iteración hace una búsqueda o un conteo más largos y necesita el uso de una estructura de datos auxiliar.

Para el ADT, dados los requerimientos propuestos, se usaron queues y stacks para acceder a los elementos pedidos: el último en el caso de las conexiones entrantes y el primero en el caso de las conexiones salientes. También se usaron vectores como estructuras puramente auxiliares para facilitar transferencias o registros, esto por su naturaleza de ser una estructura ordenada mediante un índice.

Complejidades de algoritmos

Los algoritmos para ingresar los datos al ADT de CompConnections son de complejidad O(n). El primero busca el nombre que le corresponde al ip ingresado entre los distintos registros, pasando por *n* objetos. El segundo los mete al campo correcto dependiendo si son conexiones entrantes o salientes, pasando por todos los *n* registros correspondientes.

```
for(int i = 0; i < data.size() - 1; i++){
    if(data[i].sourceIp_a == ip){
        name = data[i].sourceName_a;
        break;
    }
}

CompConnections connect(ip,name);


for(int i = 0; i < data.size() - 1; i++){
    if(data[i].sourceIp_a == ip){
        connect.addToOutgoingConnections(data[i].destinationIp_a,data[i].destinationPort_a,data[i].destinationName_a);
    }
    else if(data[i].destinationIp_a == ip){
        connect.addToIncomingConnections(data[i].sourceIp_a,data[i].sourcePort_a,data[i].sourceName_a);
        last_ip = data[i].sourceIp_a;
    }
}
```

La impresión de la ip usada tiene complejidad de O(1), es un *cout* común y corriente.

```
// ---
std::cout << "1.- Que direccion ip estas usando?" << std::endl;
std::cout << "IP: " << ip << std::endl;
// ---
```

Para verificar el tipo de conexión de la última ip entrante se utiliza la función *std::string::find()* que permite verificar la existencia de un substring dentro de otro string. Esto tiene una complejidad de O(1).

```
// ---
std::cout << "2.- Cual fue la direccion IP de la ultima conexion que recibio esta computadora? Es interna o externa?" << std::endl;
std::cout << "IP: " << last_ip;

if(last_ip.find(ip_base) != std::string::npos){
    std::cout << "\tInterna" << std::endl;
}
else{
    std::cout << "\tExterna" << std::endl;
}

// ---
```

Todos los algoritmos que se usan al checar la cantidad de conexiones entrantes y salientes son de complejidad O(n). Se saca un elemento de la queue/stack y se ingresa a una variable

auxiliar de tipo Connection, se elimina el objeto de la estructura y se ingresa la variable auxiliar a otra estructura auxiliar de orden opuesto: en el caso de un stack, se ingresa a una queue auxiliar y viceversa. Se incrementa un contador y al finalizar el conteo se vuelven a ingresar a las estructuras correspondientes en el ADT.

```
// ---
std::cout << "3.- Cuantas conexiones entrantes tiene esta computadora?" << std::endl;

std::queue<Connection> in_aux_q;
counter = 0;

while(!connect.inConnections.empty()){
    aux_c = connect.inConnections.top();
    connect.inConnections.pop();
    counter++;
    in_aux_q.push(aux_c);
}

std::cout << "Conexiones entrantes: " << counter << std::endl;

while(!in_aux_q.empty()){
    connect.addToIncomingConnections(in_aux_q.front().ip, in_aux_q.front().port, in_aux_q.front().host);
    in_aux_q.pop();
}
// ---
```

```
// ---
std::cout << "4.- Cuantas conexiones salientes tiene esta computadora?" << std::endl;

std::stack<Connection> out_aux_s;
counter = 0;

while(!connect.outConnections.empty()){
    aux_c = connect.outConnections.front();
    connect.outConnections.pop();
    counter++;
    out_aux_s.push(aux_c);
    out_connections.push_back(aux_c);
}

std::cout << "Conexiones salientes: " << counter << std::endl;

while(!out_aux_s.empty()){
    connect.addToOutgoingConnections(out_aux_s.top().ip, out_aux_s.top().port, out_aux_s.top().host);
    out_aux_s.pop();
}
// ---
```

Finalmente, para revisar las conexiones continuas se recorre un vector previamente creado, verificando que los elementos en la posición *i* e *i+1* sean iguales, y que los elementos en la posición *i+1* e *i+2* sean iguales. No es necesario verificar que los elementos en las posiciones *i* e

i+2 sean iguales, ya que una propiedad lógica permite asumir que si A=B y B=C, entonces A=C. La complejidad temporal de este proceso es de O(n).

```
// ---
std::cout << "5.- Tiene esta computadora 3 conexiones seguidas a un mismo sitio web?" << std::endl;

bool r_flag; // true = si hay 3 repetidas, false = no hay 3 repetidas
std::string r_ip;

for(int i = 0; i < out_connections.size() - 3; i++){
    if(out_connections[i].ip == out_connections[i + 1].ip && out_connections[i + 1].ip == out_connections[i + 2].ip){
        r_flag = true;
        r_ip = out_connections[i].ip;
        break;
    }
    else{
        r_flag = false;
    }
}

if(r_flag == true){
        std::cout << "Si hay una conexion seguida a " << r_ip << std::endl;
    }
    else if(r_flag == false){
        std::cout << "No hay una conexion repetida" << std::endl;
    }
// ---
```