Instituto Tecnológico y de Estudios Superiores de Monterrey

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

Act 2.3: Actividad Integral, estructura de datos lineales

Marcela Beatriz De La Rosa Barrios A01637239

01/10/2023

Act 2.3: Actividad Integral, estructura de datos lineales

Investigación

Las listas doblemente enlazadas son estructuras de datos en forma de lista que permite moverse hacia delante y hacia atrás, estas listas pueden actualizar y eliminar elementos por lo que la posición de los elementos es importante. Cada nodo de una lista doblemente enlazada tiene dos enlaces, además de los campos de datos. Un enlace, el derecho, se utiliza para navegar la lista hacia delante. El otro enlace, el izquierdo, se utiliza para navegar la lista hacia atrás.

En cuanto a la realización del código, como necesita administrar registros y realizar operaciones de inserción, búsqueda y eliminación, las listas doblemente enlazadas permiten realizar estas de manera más eficiente por las siguientes razones:

- Se puede realizar la inserción en tiempo constante cuando se conoce la posición, por lo que se puede mantener un orden específico como la organización de la dirección IP.
- Las búsquedas son más eficientes, como por ejemplo en este caso, cómo ya se ordenó la dirección IP, es más rápido buscar un rango específico de registros.
- Puede eliminar registros específicos a tiempo constante mientras que se tenga acceso al nodo que se va a eliminar.
- Permiten acceder al elemento anterior y al posterior de manera automática, cosa que ayuda a comparar el elemento actual con el siguiente para las funciones establecidas.

Sin embargo, hay desventajas al usar este tipo de listas ligadas, las cuales son:

- En comparación con un vector o una lista simple, la lista doblemente ligada requiere más memoria porque contiene un puntero extra (prev) el cual se encuentra presente en cada nodo, lo cual se convierte en una desventaja si se tiene poca memoria.
- Mantener una lista de este tipo, de manera ordenada puede ser complicado ya que se manejan 3 casos para la inserción y eliminación dependiendo de en qué lugar de la lista se debe colocar.

Para la realización de este código se requirió priorizar las siguientes instrucciones:

1. Abrir el archivo de entrada llamado "bitacora.txt para poder leerlo y almacenar los datos que contiene usando una lista doblemente ligada ordenada.
2. Ordenar la información por IP para la realización de las búsquedas.
3. Solicitar al usuario las ips de inicio y fin de búsqueda de información.
4. Desplegar los registros correspondientes a esas IPs.
5. Almacenar en un archivo el resultado del ordenamiento.

Código por partes

`class bitacoraEntrada –>` Para empezar, se declara la Clase "bitacoraEntrada" la cual representa una línea de texto individual con los atributos de mes, día, fecha, ipAdress y razón. Además, se realizaron los apuntadores de la lista doblemente ordenada (que normalmente se representan como una estructura de nodo separada) dentro de la declaración de la clase. Por último se declara el constructor para inicializar los atributos cuando se cree una nueva entrada de bitácora.

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

class bitacoraEntrada {
public:
    string mes;
    int dia;
    string fecha;
    string ipAddress;
    string razon;
    bitacoraEntrada* next;
    bitacoraEntrada* prev;

    bitacoraEntrada(string m, int d, string t, string ip, string r) : mes(m),
dia(d), fecha(t), ipAddress(ip), razon(r), next(nullptr), prev(nullptr) {}
};
//--------------------------------------------------------------------------
```

`class bitacoraLista –>`

La clase "bitacoraLista" representa la lista de entradas de bitácora, por lo que tiene 2 punteros (head y tail) mismos que apuntan a la primera y última entrada de la lista.

`void addOrderedEntry –> Complejidad: O(n)`

Primero se crea un nuevo objeto de "bitacoraEntrada" con la información necesaria. Si la lista está vacía, se le da a head y a tail el valor de la nueva entrada, en caso contrario, se inicia un bucle while que recorre la lista mientras current no sea nulo y la dirección IP de newEntry sea mayor que la dirección IP de current (el cual busca la posición donde insertar newEntry). Esto se maneja en 3 casos, si se inserta al final es porque current se volvió nulo lo cual pone a newEntry al final de la lista y se ajustan los punteros para que este sea el nuevo último elemento (tail), al principio o en medio que es cuando "current" no es igual a "head" y no se ha llegado al final de la lista, por lo que newEntry se inserta entre dos elementos existentes para esto se ajustan los punteros "next" y "prev" y los elementos alrededor del nuevo valor para que sea insertado en el lugar correcto.

`void displayEntries–> Complejidad: O(n)`

Se utiliza esta función para que todas las entradas de la lista del documento de "bitácora" se impriman, para lo que itera a lo largo de la lista empezando por head para imprimir los atributos de cada entrada de la bitácora.

`void filterAndDisplayEntries–> Complejidad: O(n)`

Se declara la variable “current” de tipo puntero de “bitacoraEntrada” para iniciar por el primer elemento en la lista y poder recorrerla desde el principio. Con esto se crea un bucle while que se repite mientras que current no sea nulo y la dirección IP del elemento actual (current) sea menor o igual a endIP. También se verifica si la dirección IP de current es mayor o igual a startIP. Si estas condiciones se cumplen (la dirección IP está dentro del rango) se imprime la información de la entrada de registro actual.

`void saveOrderedDataToFile–> Complejidad: O(n)`

Primero se crea un objeto de flujo de salida (outputFile) para escribir en un archivo, cuyo nombre se provee en el main. Después se crea un puntero de tipo “bitacoraEntrada” que señala al primer del valor de la lista para poder ir recorriéndolo (mientras current no sea nulo). Con esto, se pasa la información del registro para poder mandarlo al archivo por orden y con formato (mes dia fecha ipAddress razón). Luego, el puntero current se actualiza para apuntar al siguiente elemento de la lista para finalmente tener todos los registros ordenados en el otro archivo.

```
class bitacoraLista {
public:
    bitacoraEntrada* head;
    bitacoraEntrada* tail;

    bitacoraLista() : head(nullptr), tail(nullptr) {}

    void addOrderedEntry(string m, int d, string t, string ip, string r) {
        bitacoraEntrada* newEntry = new bitacoraEntrada(m, d, t, ip, r);

        if (!head) {
            head = tail = newEntry;
        } else {
            bitacoraEntrada* current = head;

            // Find the correct position to insert the new entry
            while (current && ip > current->ipAddress) {
                current = current->next;
            }

            if (!current) { // inserta al final
                newEntry->prev = tail;
                tail->next = newEntry;
                tail = newEntry;
            } else if (current == head) { // inserta al principio
                newEntry->next = head;
                head->prev = newEntry;
                head = newEntry;
            } else { // inserta a la mitad
                newEntry->next = current;
                newEntry->prev = current->prev;
                current->prev->next = newEntry;
                current->prev = newEntry;
```

```
            }
        }
    }

    void displayEntries() {
        bitacoraEntrada* current = head;
        while (current != nullptr) {
            cout << "Mes: " << current->mes << endl;
            cout << "Día: " << current->dia << endl;
            cout << "Fecha: " << current->fecha << endl;
            cout << "IP: " << current->ipAddress << endl;
            cout << "Razón: " << current->razon << endl;
            cout << "----------------------" << endl;
            current = current->next;
        }
    }

void filterAndDisplayEntries(string startIP, string endIP) {
        bitacoraEntrada* current = head;

        while (current != nullptr && current->ipAddress <= endIP) {
            if (current->ipAddress >= startIP) {
                cout << "Mes: " << current->mes << endl;
                cout << "Día: " << current->dia << endl;
                cout << "Fecha: " << current->fecha << endl;
                cout << "IP: " << current->ipAddress << endl;
                cout << "Razón: " << current->razon << endl;
                cout << "----------------------" << endl;
            }
            current = current->next;
        }
    }

    void saveOrderedDataToFile(const string& filename) {
        ofstream outputFile(filename);
        bitacoraEntrada* current = head;

        while (current != nullptr) {
            outputFile << current->mes << " " << current->dia << " " <<
current->fecha << " " << current->ipAddress << " " << current->razon << endl;
            current = current->next;
        }

        outputFile.close();
    }

    ~bitacoraLista() {
        bitacoraEntrada* current = head;
        while (current != nullptr) {
            bitacoraEntrada* temp = current;
            current = current->next;
            delete temp;
        }
    }
```

```
};
```

Con lo declarado anteriormente se crea un main para que se puedan llevar a cabo las funciones descritas anteriormente.

```
int main() {
    ifstream inputFile("bitacora.txt");
    if (!inputFile.is_open()) {
        cerr << "Error: No se pudo abrir el archivo." << endl;
        return 1;
    }

    bitacoraLista logList;
    string mes, fecha, ipAddress, razon;
    int dia;

    while (inputFile >> mes >> dia >> fecha >> ipAddress) {
        getline(inputFile, razon);
        razon = razon.substr(1);
        logList.addOrderedEntry(mes, dia, fecha, ipAddress, razon);
    }

    inputFile.close();
    string startIP, endIP;
    cout << "Inicio de la busqueda del IP: ";
    cin >> startIP;
    cout << "Final de la busqueda del IP: ";
    cin >> endIP;

    logList.filterAndDisplayEntries(startIP, endIP);
    logList.saveOrderedDataToFile("ordenadas.txt");

    return 0;
}
```

Funcionamiento del código:

```
Inicio de la busqueda del IP: 1.14.815.77:4402
Final de la busqueda del IP: 1.26.288.72:4768
Mes: Oct
Día: 27
Fecha: 18:48:25
IP: 1.14.815.77:4402
Razón: Failed password for admin
----------------------
Mes: Sep
Día: 19
Fecha: 12:46:04
IP: 1.16.249.84:5139
Razón: Failed password for root
----------------------
Mes: Jul
Día: 26
Fecha: 22:52:23
IP: 1.26.288.72:4768
Razón: Failed password for illegal user guest
----------------------
```

Referencias

Estructura de Datos: Lista Enlazada Doble. (s.f.). Universidad Nacional de Rosario. Recuperado de https://www.fceia.unr.edu.ar/estruc/2005/listendo.htm#:~:text=Es%20un%20tipo%20de%20lista,navegar%20la%20lista%20hacia%20atras.