Tecnológico de Monterrey

Campus:

Santa Fe

Materia:

Programación de estructuras de datos y algoritmos fundamentales

Nombre de la actividad:

Act 1.3 - Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales (Evidencia Competencia)

Nombre y Matrícula:

Emiliano Deyta Illescas A01785881
Darío Cuauhtémoc Peña Mariano A01785420

Profesor:

Vicente Cubells

Grupo:

601

Fecha:

23/09/2024

- **¿Cuántos registros tiene tu archivo?**

Para calcular el número de registros usé la clase que modela cada Entrada,en la que a partir de una iteración por cada renglón que hay en el csv se va agregando a un vector de Entradas, por último hice un método que recibe un vector y únicamente regresa el tamaño del vector que recibe.

El archivo cuenta con 36761 registros.

```
PS G:\Tec\Tec\programacion\c++\proyecto2 3\proyecto2\output> & .\'main.exe'
Hay un total de 36761 registros
El segundo día 11-8-2020 hubo 3267 registros
```

Código:

```cpp
template <class T>
inline std::vector<T> EntradaService<T>::ObtenerDB(const std::string &nombreArchivo)
{

    std::ifstream archivo(nombreArchivo);
    std::vector<T> db;
    if (!archivo.is_open())
    {
        std::cerr << "Error al abrir" << std::endl;
        return db;
    }
    std::string linea;
    while (std::getline(archivo, linea))
    {

        std::stringstream ss(linea); // Crear un stringstream de la línea
        std::string valor;
        std::string fecha, hora, ipOrigen, puertoOrigenSt, nombreOrigen, ipDestino, puertoDestinoSt, nombreDestino;

        std::getline(ss, fecha, ',');
        std::getline(ss, hora, ',');
        std::getline(ss, ipOrigen, ',');
        std::getline(ss, puertoOrigenSt, ',');
        std::getline(ss, nombreOrigen, ',');
        std::getline(ss, ipDestino, ',');
        std::getline(ss, puertoDestinoSt, ',');
        std::getline(ss, nombreDestino, ',');
        // Conversión de los puertos a enteros
        int puertoOrigen = (puertoOrigenSt == "-" ? 0 : std::stoi(puertoOrigenSt));
        int puertoDestino = (puertoDestinoSt == "-" ? 0 : std::stoi(puertoDestinoSt));

        // Crear objeto Entrada con los valores leídos y agregarlo a la base de datos
        db.emplace_back(Entrada(fecha, hora, ipOrigen, puertoOrigen, nombreOrigen, ipDestino, puertoDestino, nombreDestino));
    }

    archivo.close();
    return db;
}

template <class T>
inline int EntradaService<T>::getSize(const std::vector<T> &vec)
{
    return vec.size();
}
```

- **¿Cuántos récords hay del segundo día registrado? ¿Qué día es este?**

Para obtener el número por día, hice un método que recibe una fecha, con el fin de hacer búsqueda por día, después de eso se hace una búsqueda del vector de Entradas, y cuando el atributo “día” coincide con el día que se está buscando se agrega a un vector temporal de Entradas, ya con esto regreso el tamaño de ese vector temporal.

Existen 3267 en el segundo día, 11-8-2020.

```
Hay un total de 36761 registros
El segundo dia 11-8-2020 hubo 3267 registros
```

Código:

```cpp
template <class T>
inline int EntradaService<T>::getSizeByDay(const std::vector<Entrada> &vec, std::string day)
{
    std::vector<Entrada> resultados;
    int resultado = 0;
    for (size_t i = 0; i < vec.size(); i++)
    {
        if (vec[i].fecha == day)
        {
            resultado++;
        }
    }

    return resultado;
}
```

- **¿Alguna de las computadoras pertenece a Jeffrey, Betty, Katherine, Scott, Benjamin, Samuel o Raymond? ¿A quiénes?**

En el archivo “equipo2.csv” no se encontró a ninguna persona con los nombres previamente mencionados.

```
Ingresa el nombre a buscar: Betty
Betty no tiene una computadora en la red.
```

Código:

```cpp
template <class T>
inline bool EntradaService<T>::buscarComputadoras(const std::vector<Entrada>& bitacora, const std::string& c)
{
    std::string nombreBuscado = c;
    std::transform(nombreBuscado.begin(), nombreBuscado.end(), nombreBuscado.begin(), tolower);

    for (size_t i = 0; i < bitacora.size(); ++i)
    {
        std::string nombreOrigenMinusculas = bitacora[i].nombreOrigen;
        std::transform(nombreOrigenMinusculas.begin(), nombreOrigenMinusculas.end(), nombreOrigenMinusculas.begin(), tolower);

        if (nombreOrigenMinusculas == nombreBuscado + ".reto.com")
        {
            return true;
        }
    }
    return false;
}
```

- **¿Cuál es la dirección de la red interna de la compañía?**

La dirección interna es: 192.168.86.X

```
La dirección de la red interna es: 192.168.86.X
```

Código:

```cpp
template <class T>
inline std::string EntradaService<T>::internalNetworkAddress(const std::vector<Entrada> &vec)
{
    std::map<std::string, int> prefixCounts;

    for (std::vector<Entrada>::const_iterator it = vec.begin(); it != vec.end(); ++it)
    {
        std::string ip = it->ipOrigen;
        size_t lastDot = ip.find_last_of('.');
        if (lastDot != std::string::npos)
        {
            std::string prefix = ip.substr(0, lastDot + 1);
            prefixCounts[prefix]++;
        }
    }
```

- **¿Alguna computadora se llama server.reto.com?**

En el archivo “equipo2.csv” no se encontró ninguna computadora con el nombre “server.reto.com”

```
Ingresa el nombre a buscar: reto
reto no tiene una computadora en la red.
```

Código:

```cpp
template <class T>
inline bool EntradaService<T>::buscarComputadoras(const std::vector<Entrada>& bitacora, const std::string& c)
{
    std::string nombreBuscado = c;
    std::transform(nombreBuscado.begin(), nombreBuscado.end(), nombreBuscado.begin(), tolower);

    for (size_t i = 0; i < bitacora.size(); ++i)
    {
        std::string nombreOrigenMinusculas = bitacora[i].nombreOrigen;
        std::transform(nombreOrigenMinusculas.begin(), nombreOrigenMinusculas.end(), nombreOrigenMinusculas.begin(), tolower);

        if (nombreOrigenMinusculas == nombreBuscado + ".reto.com")
        {
            return true;
        }
    }
    return false;
}
```

- **¿Qué servicio de correo electrónico utilizan (algunas ideas: Gmail, Hotmail, Outlook, Protonmail)?**

Se utilizó Gmail, Outlook/Hotmail, Steam Community, Best Buy, Expedia, Techradar, entre otros.

```
Servicios de correo electrónico utilizados: Best Buy, Expedia, Gmail, Outlook/Hotmail, Steam, Tech Radar,
```

Código:

```cpp
template <class T>
inline std::set<std::string> EntradaService<T>::emailServicesUsed(const std::vector<Entrada> &vec)
{
    std::set<std::string> servicios;
    for (std::vector<Entrada>::const_iterator it = vec.begin(); it != vec.end(); ++it)
    {
        // Buscar patrones comunes en nombres de destino que sugieran servicios de correo
        if (it->nombreDestino.find("gmail") != std::string::npos)
        {
            servicios.insert("Gmail");
        }
        else if (it->nombreDestino.find("microsoft") != std::string::npos)
        {
            servicios.insert("Outlook/Hotmail");
        }
        else if (it->nombreDestino.find("steamcommunity") != std::string::npos)
        {
            servicios.insert("Steam");
        }
        else if (it->nombreDestino.find("techradar") != std::string::npos)
        {
            servicios.insert("Tech Radar");
        }
        else if (it->nombreDestino.find("bestbuy") != std::string::npos)
        {
            servicios.insert("Best Buy");
        }
        else if (it->nombreDestino.find("expedia") != std::string::npos)
        {
            servicios.insert("Expedia");
        }
    }
    return servicios;
}
```

- **Considerando solamente los puertos destino ¿Qué puertos abajo del 1000 se están usando? Lista los puertos e investiga qué aplicación/servicio lo utiliza generalmente.**

```
Puertos destino debajo de 1000 en uso:
53
67
80
135
443
465
965
993
```

Código:

```cpp
template <class T>
inline std::set<int> EntradaService<T>::usedPortsBelow1000(const std::vector<Entrada> &vec)
{
    std::set<int> puertos;
    for (std::vector<Entrada>::const_iterator it = vec.begin(); it != vec.end(); ++it)
    {
        if (it->puertoDestino > 0 && it->puertoDestino < 1000)
        {
            puertos.insert(it->puertoDestino);
        }
    }
    return puertos;
}
```

Puerto	Servicio/Aplicación	Descripción
53	DNS (Domain Name System)	Traduce nombres de dominio a direcciones IP y viceversa, esencial para la navegación web.
67	DHCP (Dynamic Host Configuration Protocol)	Asigna automáticamente direcciones IP a dispositivos en una red.
80	HTTP (Hypertext Transfer Protocol)	Protocolo principal para la transferencia de datos en la web (páginas web sin cifrar).
135	DCOM/RPC (Distributed Component Object Model/Remote Procedure Call)	Permite la comunicación entre componentes de software en diferentes computadoras.
443	HTTPS (Hypertext Transfer Protocol Secure)	Versión segura de HTTP, utiliza cifrado para proteger la comunicación web.
465	SMTP sobre SSL (Simple Mail Transfer Protocol over SSL)	Protocolo para enviar correos electrónicos, versión segura con cifrado SSL.
965	POP3 sobre SSL (Post Office Protocol 3 over SSL)	Protocolo para recibir correos electrónicos, versión segura con cifrado SSL.
993	IMAP sobre SSL (Internet Message Access Protocol over SSL)	Protocolo para acceder a correos electrónicos en el servidor, versión segura con cifrado SSL.

Investigación y reflexión individual: Emiliano Deyta Illescas

El código utiliza principalmente búsquedas lineales, destaca la relevancia de algoritmos de ordenamiento y búsqueda más eficientes en el manejo de grandes volúmenes de datos y la realización de consultas complejas.

El ordenamiento permite un acceso más rápido y eficiente a la información relevante, la búsqueda binaria reduce significativamente el tiempo de búsqueda en comparación con la búsqueda lineal. Los algoritmos eficientes garantizan un buen rendimiento incluso con grandes cantidades de datos. El ordenamiento previo facilita la implementación de análisis más complejos, como la identificación de patrones o anomalías.

La búsqueda lineal con complejidad temporal O(n), es adecuada para conjuntos de datos pequeños o búsquedas poco frecuentes.Mientras que la búsqueda binaria de complejidad temporal O(log n), mucho más eficiente para grandes volúmenes de datos, pero requiere ordenamiento previo. En cuanto a los algoritmos de ordenamiento la elección depende del tamaño y características de los datos. Algoritmos como Mergesort y Quicksort ofrecen un buen equilibrio entre eficiencia y facilidad de implementación (O(n log n)).

Los algoritmos de ordenamiento y búsqueda son pilares fundamentales en el análisis de datos, y su elección adecuada puede tener un impacto significativo en el rendimiento, la escalabilidad y la capacidad de análisis de una aplicación. En el contexto del análisis de registros de red, el uso estratégico de estos algoritmos puede mejorar la eficiencia y permitir análisis más avanzados, especialmente cuando se trabaja con grandes volúmenes de datos o se realizan consultas complejas.

Es esencial considerar cuidadosamente los requisitos específicos de la aplicación, las características de los datos y el equilibrio entre eficiencia y complejidad de implementación al seleccionar los algoritmos de ordenamiento y búsqueda más apropiados. La implementación cuidadosa de estos algoritmos puede marcar la diferencia entre un sistema de análisis de registros de red eficiente, escalable y capaz de realizar análisis sofisticados, y uno que se vuelva lento e inmanejable a medida que crece el volumen de datos y la complejidad de las consultas. La inversión en la elección y optimización de estos algoritmos se traduce en una mayor capacidad para extraer información valiosa de los registros de red y tomar decisiones informadas basadas en datos.

Investigación y reflexión individual: Darío Cuauhtémoc Peña Mariano

Los algoritmos de búsqueda y ordenamiento son herramientas en la informática los cuales hacen el resultado de una búsqueda mucho más eficiente dependiendo del caso de uso en el que se está aplicando, es importante conocer los algoritmos debido a que desde el momento de planeación del código que vas hacer, puedes crearlo en base al algoritmo de búsqueda y ordenamiento con el fin de que el código sea lo más eficiente posible.

Un ejemplo de uso donde se utilizan algoritmos de ordenamiento y búsqueda diferentes a los tradicionales son en las bases de datos, en donde debido a que se debe de pensar en hacer búsquedas rápidas y con poco consumo de memoria se hacen el uso de algoritmos más eficientes para estos casos como el uso del algoritmo Merge Sort, Quick Sort, Índices B-Tree entre otros.

Para esta actividad en mi parte al principio pensé en usar algoritmos diferentes al de la iteración, sin embargo, al momento de analizar el contexto y darme cuenta que de todos modos necesitaba hacer un vector donde pasara todas las entradas de un archivo al mi programa, decidí usar la iteración, ya que el mejor y peor de los casos era el mismo, debía de integrar todos los elementos del archivo sin saltarme ninguno.

Aportaciones individuales:

Darío Cuauhtémoc Peña Mariano:
-Creación de ADT personalizada para las entradas
-Lectura de archivo CSV
-Almacenaje de datos en vector
-Cálculo del total de entradas en archivo csv mediante método
-Cálculo del total de entradas por día mediante método

Emiliano Deyta Illescas:
-Búsqueda de usuarios en la red
-Obtención de red interna
-Búsqueda de servicios de correo electrónico
-Busqueda de puertos de destino

Referencias:

- *Algoritmos de Búsqueda*. (n.d.). http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro9/algoritmos_de_bsqueda.html
- Ponce, J. (2021, February 21). *Algoritmos de búsqueda - Jahaziel Ponce*. Jahaziel Ponce. https://jahazielponce.com/algoritmos-de-busqueda/
- GeeksforGeeks. (s.f.). **Sorting Algorithms**. Recuperado el 23 de septiembre de 2024, de https://www.geeksforgeeks.org/sorting-algorithms/
- Khan Academy. (s.f.). **Algoritmos**. Recuperado el 23 de septiembre de 2024, de https://es.khanacademy.org/computing/computer-science/algorithms
- MIT OpenCourseWare. (s.f.). **Introduction to Algorithms**. Recuperado el 23 de septiembre de 2024, de https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-spring-2020/
- Tutorials Point. (s.f.). **Data Structures & Algorithms**. Recuperado el 23 de septiembre de 2024, de https://www.tutorialspoint.com/data_structures_algorithms/index.htm
- Universidad de San Francisco, Departamento de Ciencias de la Computación. (s.f.). **Sorting Algorithm Animations**. Recuperado el 23 de septiembre de 2024, de https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
- Oracle. (s.f.). **The Java Tutorials: Collections - Algorithms**. Recuperado el 23 de septiembre de 2024, de https://docs.oracle.com/javase/tutorial/collections/algorithms/index.html