Tecnológico de Monterrey

Campus:

Santa Fe

Materia:

Programación de estructuras de datos y algoritmos fundamentales

Nombre de la actividad:

Act 2.2 - Actividad Integral estructura de datos lineales (Evidencia Competencia)

Nombre y Matrícula:

Emiliano Deyta Illescas A01785881
Darío Cuauhtémoc Peña Mariano A01785420

Profesor:

Vicente Cubells

Grupo:

601

Fecha:

10/10/2024

1. **¿Qué dirección ip estás usando?**

En este programa estamos ocupando la dirección 192.168.86.79

```
Ingresa un número del 1 al 150 para crear tu dirección ip interna:
79


_____________________________

La base de registros tiene un total de 36761 registros, la ip de la computadora es 192.168.86.79
```

Código:

```
template <typename T>
ConexionesCompudatora<T>::ConexionesCompudatora(std::string ip, std::string nombre){
    std::string valoresUsuario;
    ipComputadora = ip;
    ip.pop_back();
    std::cout << "Ingresa un número del 1 al 150 para crear tu dirección ip interna: " << std::endl;

    std::cin >> valoresUsuario;

     ipComputadora = ip;
     ipComputadora += valoresUsuario;
    nombreComputadora = nombre;
}
```

2. **¿Cuál fue la dirección IP de la última conexión que recibió esta computadora? ¿Es interna o externa?**

La última conexión que recibió la computadora fue desde la IP: 192.168.86.22 y es una conexión interna.

```
La última conexión que recibió la computadora fue desde la IP: 192.168.86.22 y es una conexión interna.
```

Código:

```
template <typename T>
void ConexionesCompudatora<T>::obtenerUltimaConexionEntrante()
{
    if (conexionesEntrantes.empty())
    {
        std::cout << "No hay conexiones entrantes para esta computadora." << std::endl;
        return;
    }

    T ultimaConexion = conexionesEntrantes.back();
    std::string ip = ultimaConexion.ipOrigen;

    std::string tipoConexion;
    if (ip.substr(0, 10) == "192.168.86")
    {
        tipoConexion = "interna";
    }
    else
    {
        tipoConexion = "externa";
    }
    std::cout << "La última conexión que recibió la computadora fue desde la IP: " << ip << " y es una conexión " << tipoConexion << "." << std::endl;
}
```

3. **¿Cuántas conexiones entrantes tiene esta computadora?**

Cuenta con 2 conexiones entrantes.

```
Hay un total de conexiones entrantes: 2
```

Código:

```cpp
template <typename T>
inline void ConexionesCompudatora<T>::rellenarRegistros(std::vector<T> &db)
{
    for (size_t i = 0; i < db.size(); i++)
    {
        if (ipComputadora ==db[i].ipDestino)
        {
            conexionesEntrantes.push_back(db[i]);
        }
        if (ipComputadora ==db[i].ipOrigen)
        {
            conexionesSalientes.push_back(db[i]);

        }



    }
    std::cout << "" << std::endl;
    std::cout << "_____" << std::endl;
    std::cout << "" << std::endl;
    std::cout << "La base de registros tiene un total de " << db.size()<< " registros, la ip de la computadora es "<< ipComputadora<<  std::endl;
    std::cout << "Hay un total de conexiones entrantes: " << conexionesEntrantes.size()<< std::endl;
    std::cout << "Hay un total de conexiones salientes: " << conexionesSalientes.size()<< std::endl;
    std::cout << "" << std::endl;
    std::cout << "_____" << std::endl;
    std::cout << "" << std::endl;

}
```

4. **¿Cuántas conexiones salientes tiene esta computadora?**

Cuenta con 382 conexiones salientes

```
Hay un total de conexiones salientes: 382
```

Código:

```cpp
template <typename T>
inline void ConexionesCompudatora<T>::rellenarRegistros(std::vector<T> &db)
{
    for (size_t i = 0; i < db.size(); i++)
    {
        if (ipComputadora ==db[i].ipDestino)
        {
            conexionesEntrantes.push_back(db[i]);
        }
        if (ipComputadora ==db[i].ipOrigen)
        {
            conexionesSalientes.push_back(db[i]);

        }



    }
    std::cout << "" << std::endl;
    std::cout << "_____" << std::endl;
    std::cout << "" << std::endl;
    std::cout << "La base de registros tiene un total de " << db.size()<< " registros, la ip de la computadora es "<< ipComputadora<<  std::endl;
    std::cout << "Hay un total de conexiones entrantes: " << conexionesEntrantes.size()<< std::endl;
    std::cout << "Hay un total de conexiones salientes: " << conexionesSalientes.size()<< std::endl;
    std::cout << "" << std::endl;
    std::cout << "_____" << std::endl;
    std::cout << "" << std::endl;

}
```

5. **¿Tiene esta computadora 3 conexiones seguidas a un mismo sitio web?**

Si, tiene más de 3 conexiones consecutivas a un sitio web, tiene 8 conexiones consecutivas a el sitio web gmail.com

```
La computadora ha realizado 8 conexiones consecutivas al mismo sitio web: gmail.com
```

Código:

```cpp
template <typename T>
void ConexionesCompudatora<T>::comprobarRegistrosConsecutivos()
{
    int contadorConsecutivos = 1;
    int maxConsecutivos = 0;
    std::string sitioWebMaxConsecutivos;

    if (conexionesSalientes.size() < 3)
    {
        std::cout << "No hay suficientes conexiones salientes para comprobar secuencias de tres o más conexiones." << std::endl;
        return;
    }

    for (size_t i = 1; i < conexionesSalientes.size(); i++)
    {
        if (conexionesSalientes[i].nombreDestino == conexionesSalientes[i - 1].nombreDestino)
        {
            contadorConsecutivos++;

            if (contadorConsecutivos > maxConsecutivos)
            {
                maxConsecutivos = contadorConsecutivos;
                sitioWebMaxConsecutivos = conexionesSalientes[i].nombreDestino;
            }
        }
        else
        {
            contadorConsecutivos = 1;
        }
    }

    if (maxConsecutivos >= 3)
    {
        std::cout << "La computadora ha realizado " << maxConsecutivos << " conexiones consecutivas al mismo sitio web: " << sitioWebMaxConsecutivos << std::endl;
    } else
    {
        std::cout << "No se encontraron secuencias de tres o más conexiones consecutivas al mismo sitio web." << std::endl;
    }
}
```

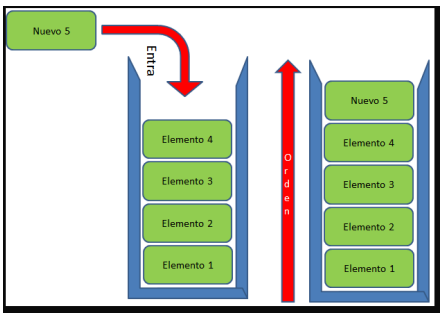
Investigación y reflexión Darío Cuauhtémoc Peña Mariano A01785420

Tanto en la informática como en otras ramas o casos de la vida real existe la necesidad de ordenar objetos y poder accederlos, cada forma de ordenamiento sirve en mayor o en menor medida para solucionar una problemática en especifica.

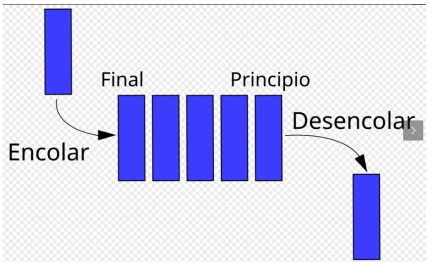
En la informática debido a la necesidad de almacenamiento y acceso a la información se han creado varias estructuras, un tipo de estas estructuras son las lineales, hay muchas estructuras, sin embargo, la diferencia entre ellas es como relacionan los objetos dentro de ellas, cada estructura está optimizada para una búsqueda en específico, dentro de estas estructuras las más utilizadas son:

Listas: Es una colección de objetos ordenados en donde se puede acceder a ellos mediante el índice.

Pilas: Esta estructura funciona alrededor de el principio en el que el último en entrar es el primero en salir.



Filas: Esta estructura es parecida a la fila, sin embargo, el principio que sigue es el primero en entrar es el primero en salir.



Las estructuras de datos son fundamentales en un programa informático, conocer cómo funciona internamente aún más, esto debido a que al saber cómo funciona, nos permite que desde del momento de planeación de un código podamos optimizarlo para ser compatible con una estructura específica, esto con el fin de que la manera en cómo se ordena y procesa la información sea la más veloz posible.

Referencias:

Listas enlazadas. (2020, September 4). Lic. En Sistemas De Información. https://licsistemas2020.wordpress.com/listas-enlazadas/

Oblancarte. (2014, August 6). *Pila (Stack) - Oscar Blancarte - Software Architecture*. Oscar Blancarte - Software Architecture. https://www.oscarblancarteblog.com/2014/08/06/estructura-de-datos-pila-stack/

colaboradores de Wikipedia. (2022, October 19). *Cola (informática)*. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Cola_(inform%C3%A1tica)

Investigación y reflexión individual: Emiliano Deyta Illescas A01785881

Las estructuras de datos lineales, como las listas enlazadas, las colas y las pilas, son fundamentales en el mundo de la programación por su habilidad para manejar datos de manera eficiente. Estas estructuras son especialmente valiosas porque permiten una gestión flexible de la memoria, algo crucial en aplicaciones donde el volumen de datos varía significativamente. A diferencia de los arrays estáticos, estas estructuras pueden ajustar su tamaño durante la ejecución del programa, lo que ayuda a optimizar los recursos y evitar el desperdicio.

Además de la gestión de memoria, las estructuras de datos lineales simplifican operaciones como la inserción y eliminación de elementos. En una lista enlazada, por ejemplo, añadir o eliminar un elemento es tan simple como modificar algunos enlaces, sin importar el tamaño de la lista. Esta simplicidad y eficiencia operativa es vital en entornos donde el rendimiento es crítico, como en los sistemas operativos o algoritmos complejos de procesamiento de datos.

Otro punto a favor de las estructuras lineales es que mantienen un orden específico de los datos, lo que es indispensable en muchas aplicaciones donde el orden de procesamiento afecta los resultados. Esta característica reduce la complejidad del código y minimiza el riesgo de errores, facilitando la implementación de algoritmos que dependen del orden de los datos.

En el ámbito de las situaciones problema de esta naturaleza, las estructuras de datos lineales se utilizan para resolver problemas que requieren eficiencia tanto en tiempo como en espacio. Por ejemplo, las colas son esenciales en la gestión de procesos en los sistemas operativos, donde los procesos se organizan para su ejecución según prioridades. Las pilas son igualmente importantes en aplicaciones que involucran navegaciones recursivas o el manejo de llamadas a funciones, almacenando cada llamada hasta que se completa su ejecución.

Estas estructuras también son cruciales para la optimización de algoritmos, especialmente aquellos que requieren un acceso rápido y secuencial a los datos. Los algoritmos de búsqueda y ordenación, por ejemplo, se benefician enormemente de las propiedades de las estructuras de datos lineales, permitiendo técnicas más eficientes que mejoran el rendimiento del sistema.

Las estructuras de datos lineales no solo ayudan a optimizar el uso de recursos sino que también permiten construir aplicaciones más robustas y eficientes. Por estas razones, son herramientas indispensables en la programación. Para aquellos interesados en profundizar más en este tema, libros como “Data Structures and Algorithms in Java” de Michael T. Goodrich y Roberto Tamassia, y “Introduction to Algorithms” de Thomas H. Cormen y otros, ofrecen una excelente base teórica y

práctica sobre cómo estas estructuras se aplican en problemas reales de programación.

Referencias:

University of California San Diego. (s.f.). *Data Structures*. Coursera. Recuperado de https://www.coursera.org/learn/data-structures

GeeksforGeeks. (s.f.). *Introduction to Data Structures*. Recuperado de https://www.geeksforgeeks.org/data-structures/