ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO

FACULTAD DE INFORMATICA Y ELECTRONICA

CARRERA DE SOFTWARE

TEMA

LISTAS DOBLEMENTE ENLAZADAS

ESTUDIANTES **CÓDIGOS**

[redacted]

ASIGNATURA

ESTRUCTURAS DE DATOS

PROFESOR

[redacted]

FECHA DE ENTREGA

09/06/2024

RIOBAMBA - ECUADOR

Introducción

Las estructuras de datos desempeñan un papel fundamental en la informática, proporcionando herramientas y técnicas para organizar y manipular datos de manera eficiente entre estas estructuras, las listas doblemente enlazadas son una opción versátil y poderosa que permite una amplia gama de operaciones, desde la inserción y eliminación dinámica de elementos hasta el recorrido bidireccional de la lista.

En esta investigación, exploraremos en detalle las listas doblemente enlazadas, comenzando por una definición y descripción de sus características principales. Luego, analizaremos la importancia de las listas doblemente enlazadas en el contexto de las estructuras de datos, destacando su flexibilidad y eficiencia en comparación con otras estructuras.

Examinaremos las operaciones básicas que se pueden realizar en una lista doblemente enlazada, como la inserción y eliminación de elementos en diferentes posiciones, así como el recorrido de la lista en ambas direcciones también discutiremos las ventajas y desventajas de las listas doblemente enlazadas en comparación con otras estructuras de datos, como las listas enlazadas simples y los vectores.

Para ilustrar la aplicación práctica de las listas doblemente enlazadas presentaremos un caso de estudio centrado en un sistema de gestión de inventario de una tienda analizaremos el código implementado y explicaremos cómo está vinculado con los conceptos de listas doblemente enlazadas, destacando las operaciones realizadas y las ventajas que ofrece esta estructura en este contexto específico.

Desarrollo

Listas enlazadas

Las listas enlazadas son estructuras de datos lineales compuestas por nodos que están conectados mediante enlaces cada nodo contiene un elemento de datos y un enlace (o puntero) al siguiente nodo en la secuencia esta organización permite almacenar y manipular datos de forma dinámica sin necesidad de una ubicación de memoria contigua.

Listas doblemente enlazadas

Las listas doblemente enlazadas son una variante de las listas enlazadas que poseen un enlace adicional permitiendo la navegación tanto hacia adelante como hacia atrás en la lista cada nodo en una lista doblemente enlazada contiene dos enlaces: uno que apunta al nodo anterior y otro que apunta al siguiente nodo esta estructura bidireccional proporciona mayor flexibilidad y eficiencia en operaciones como inserción, eliminación y búsqueda de elementos.

"Una lista doblemente enlazada es una colección de nodos en la cual cada nodo tiene como mínimo tres campos: un campo de datos, que contiene la información almacenada en el nodo, y los campos de las referencias del nodo de la izquierda y del nodo de la derecha (antecesor y sucesor respectivamente). Las listas doblemente enlazadas pueden ser lineales o circulares. (Hernández Bejarano, 2022)"

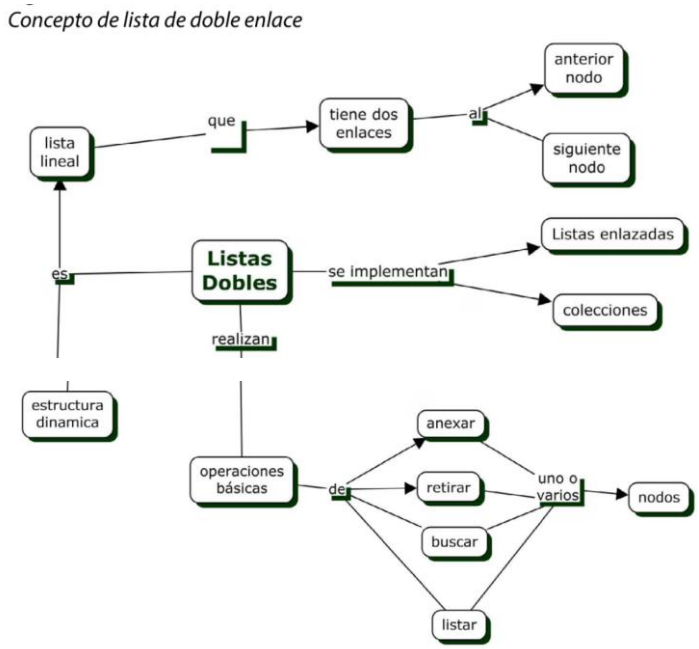

Fig. 1

"Fig. 1(Hernández Bejarano,2022)"

Importancia de las listas doblemente enlazadas en estructuras de datos

Las listas doblemente enlazadas son fundamentales en el ámbito de las estructuras de datos por las siguientes razones:

Flexibilidad en la manipulación de datos: La capacidad de acceder tanto al nodo anterior como al siguiente permite realizar operaciones de inserción y eliminación en cualquier posición de la lista con un rendimiento eficiente.

Eficiencia en el recorrido bidireccional: La estructura bidireccional facilita el recorrido de la lista en ambas direcciones, lo que resulta útil en algoritmos que requieren acceso a los elementos adyacentes.

Gestión eficiente de memoria: Las listas doblemente enlazadas permiten una gestión dinámica de la memoria ya que los nodos pueden ser agregados o eliminados sin necesidad de reorganizar toda la estructura lo que minimiza la fragmentación de la memoria.

Aplicaciones prácticas variadas: Se utilizan en una amplia gama de aplicaciones como sistemas de gestión de inventarios, sistemas de mensajería y redes sociales, sistemas de transporte y logística entre otros donde se requiere una gestión dinámica y eficiente de los datos.

Estructura de una lista doblemente enlazada

Características principales

Las listas doblemente enlazadas presentan las siguientes características distintivas:

Nodos: Cada nodo de la lista contiene un elemento de datos y dos enlaces: uno que apunta al nodo anterior (anterior) y otro que apunta al siguiente nodo (siguiente).

Enlaces: Los enlaces permiten la navegación bidireccional a través de la lista, lo que facilita operaciones como la inserción, eliminación y búsqueda de elementos.

Inicio y fin: La lista tiene dos referencias importantes: el inicio, que señala al primer nodo de la lista, y el fin, que señala al último nodo. Estos puntos de entrada permiten un acceso eficiente a los extremos de la lista.

Componentes: nodos, enlaces, inicio, fin

Ventajas

- *"Pueden recorrerse de dos formas, ya sea para efectuar una operación con cada elemento o para insertar/actualizar y borrar. (González, 2013)"*
- La capacidad de acceso bidireccional facilita la inserción, eliminación y búsqueda de elementos en cualquier posición de la lista.
- La estructura bidireccional permite recorrer la lista tanto hacia adelante como hacia atrás, lo que resulta útil en algoritmos que requieren acceso a elementos adyacentes.
- La gestión dinámica de memoria permite agregar y eliminar nodos sin necesidad de reorganizar toda la estructura, minimizando la fragmentación de la memoria.
- Se utilizan en una variedad de aplicaciones donde se necesita una gestión dinámica y eficiente de datos, como sistemas de gestión de inventarios, sistemas de transporte y logística, sistemas de mensajería, entre otros.

Desventajas

- *"Ocupan más memoria por nodo que una lista simple. (González, 2013)"*
- Cada nodo de la lista requiere almacenar referencias adicionales al nodo anterior y siguiente lo que puede aumentar el consumo de memoria en comparación con otras estructuras de datos.
- La gestión de los enlaces adicionales puede añadir complejidad al código y aumentar la posibilidad de errores de programación especialmente en implementaciones más complejas.
- La manipulación de los punteros en una lista doblemente enlazada puede ser más compleja que en otras estructuras de datos lo que puede dificultar la comprensión y la depuración del código.

Operaciones básicas

Las operaciones fundamentales que se pueden realizar en una lista doblemente enlazada incluyen inserción y eliminación de elementos estas operaciones se pueden realizar en diferentes puntos de la lista, tales como el inicio, el final o una posición específica.

Inserción: "*Esta operación consiste en visitar cada uno de los nodos que forman la lista. Para recorrer todos los elementos de la lista, se inicia con el primero, tomando el valor del campo de la referencia para avanzar al segundo nodo, el campo referencia de este nodo tiene la dirección del tercer nodo y así sucesivamente. La implementación de este método es similar al tipo de lista con enlace sencillo. (Hernández Bejarano, 2022)"*

Al inicio de la lista

Esta operación implica la inserción de un nuevo elemento en la primera posición de la lista. Requiere ajustar los enlaces del nuevo nodo y del nodo anterior al inicio. (Fig .2)

```
void ListaDoble::insertarAlInicio(int dato) {
    NodoDoble* nuevo = new NodoDoble(dato);
    if (esVacia()) {
        inicio = nuevo;
        fin = nuevo;
    } else {
        nuevo->establecerSiguiente(inicio);
        inicio->establecerAnterior(nuevo);
        inicio = nuevo;
    }
}
```

Fig. 2

Al final de la lista

Consiste en agregar un nuevo elemento al final de la lista se actualizan los enlaces del nuevo nodo y del nodo final para mantener la integridad de la estructura. (Fig. 3)

```
6
7  void ListaDoble::insertarAlFinal(int dato) {
8      NodoDoble* nuevo = new NodoDoble(dato);
9      if (esVacia()) {
0          inicio = nuevo;
1          fin = nuevo;
2      } else {
3          fin->establecerSiguiente(nuevo);
4          nuevo->establecerAnterior(fin);
5          fin = nuevo;
6      }
7  }
```

Fig. 3

En una posición específica

Permite insertar un elemento en cualquier posición deseada de la lista implica el desplazamiento de los nodos existentes y la actualización de los enlaces pertinentes para acomodar el nuevo nodo. (Fig .4)

```
void ListaDoble::insertarEnPosicion(int dato, int posicion) {
    if (posicion < 0) {
        cout << "Posición inválida." << endl;
        return;
    }
    if (posicion == 0) {
        insertarAlInicio(dato);
        return;
    }
    NodoDoble* actual = inicio;
    int contador = 0;
    while (actual != NULL && contador < posicion) {
        actual = actual->obtenerSiguiente();
        contador++;
    }
    if (actual == NULL) {
        cout << "Posición fuera de rango." << endl;
        return;
    }
    NodoDoble* nuevo = new NodoDoble(dato);
    nuevo->establecerAnterior(actual->obtenerAnterior());
    nuevo->establecerSiguiente(actual);
    actual->obtenerAnterior()->establecerSiguiente(nuevo);
    actual->establecerAnterior(nuevo);
}
```

Fig. 4

Eliminación: *"Esta operación consiste en agregar un nuevo nodo a la lista doblemente enlazada que puede ser al inicio, al final o en una posición determinada de la lista. (Hernández Bejarano, 2022)"*

Al inicio de la lista

Esta operación implica la eliminación del primer elemento de la lista se ajustan los enlaces del nuevo primer nodo y se libera la memoria del nodo eliminado. (Fig .5)

```
bool ListaDoble::eliminarAlInicio() {
    if (esVacia()) {
        cout << "La lista está vacía." << endl;
        return false;
    }

    NodoDoble* temp = inicio;
    inicio = inicio->obtenerSiguiente();
    if (inicio != NULL) {
        inicio->establecerAnterior(NULL);
    } else {
        fin = NULL;
    }
    delete temp;
    return true;
}
```

Fig. 5

Al final de la lista

Consiste en la eliminación del último elemento de la lista se actualizan los enlaces del nuevo último nodo y se libera la memoria del nodo eliminado. (Fig .6)

```
bool ListaDoble::eliminarAlFinal() {
    if (esVacia()) {
        cout << "La lista está vacía." << endl;
        return false;
    }

    NodoDoble* temp = fin;
    fin = fin->obtenerAnterior();
    if (fin != NULL) {
        fin->establecerSiguiente(NULL);
    } else {
        inicio = NULL;
    }
    delete temp;
    return true;
}
```

Fig. 6

En una posición específica

Permite eliminar un elemento de cualquier posición deseada de la lista implica el ajuste de los enlaces de los nodos adyacentes al nodo eliminado y la liberación de la memoria del nodo eliminado. (Fig .7)

```
bool ListaDoble::eliminarEnPosicion(int posicion) {
    if (esVacia()) {
        cout << "La lista está vacía." << endl;
        return false;
    }
    if (posicion < 0) {
        cout << "Posición inválida." << endl;
        return false;
    }
    if (posicion == 0) {
        return eliminarAlInicio();
    }

    NodoDoble* actual = inicio;
    int contador = 0;
    while (actual != NULL && contador < posicion) {
        actual = actual->obtenerSiguiente();
        contador++;
    }

    if (actual == NULL) {
        cout << "Posición fuera de rango." << endl;
        return false;
    }

    actual->obtenerAnterior()->establecerSiguiente(actual->obtenerSiguiente());
    if (actual->obtenerSiguiente() != NULL) {
        actual->obtenerSiguiente()->establecerAnterior(actual->obtenerAnterior());
    } else {
        fin = actual->obtenerAnterior();
    }
    delete actual;
    return true;
}
```

Fig. 7

Recorrido de la lista

Recorrido hacia adelante

El recorrido hacia adelante en una lista doblemente enlazada se realiza comenzando desde el primer nodo y avanzando secuencialmente hasta el último nodo en cada paso se accede al dato almacenado en el nodo actual y luego se utiliza el puntero al siguiente nodo para moverse al próximo elemento en la secuencia este tipo de recorrido permite procesar los elementos en el orden en que fueron insertados en la lista lo cual es útil para operaciones como la impresión de la lista, la búsqueda de elementos desde el inicio y la aplicación de funciones acumulativas a lo largo de todos los elementos. (Fig .8)

```
void ListaDoble::imprimirAdelante() {
    NodoDoble* actual = inicio;
    while (actual != NULL) {
        cout << actual->obtenerDato() << " ";
        actual = actual->obtenerSiguiente();
    }
    cout << endl;
}
```

Fig. 8

Recorrido hacia atrás

El recorrido hacia atrás en una lista doblemente enlazada implica iniciar desde el último nodo de la lista y moverse de manera retroactiva hasta el primer nodo en cada paso, se accede al dato del nodo actual y luego se utiliza el puntero al nodo anterior para retroceder al elemento previo en la secuencia este método de recorrido es útil para imprimir los elementos en orden inverso, buscar elementos comenzando desde el final de la lista y realizar operaciones que requieren procesar los elementos desde el más reciente al más antiguo proporcionando una flexibilidad adicional en el manejo de la lista. (Fig. 9)

```
void ListaDoble::imprimirAtras() {
    NodoDoble* actual = fin;
    while (actual != NULL) {
        cout << actual->obtenerDato() << " ";
        actual = actual->obtenerAnterior();
    }
    cout << endl;
}
```

Fig. 9

"Se desarrolla una aplicación donde se incorporan las operaciones básicas de una lista doblemente enlazada, para lo cual se realiza el correspondiente modelamiento y construcción del diagrama de clases, que presenta de forma estática el modelo del sistema; además cada una de estas clases es codificada en el lenguaje de programación Java Fig. 10. (Hernández Bejarano, 2022)"

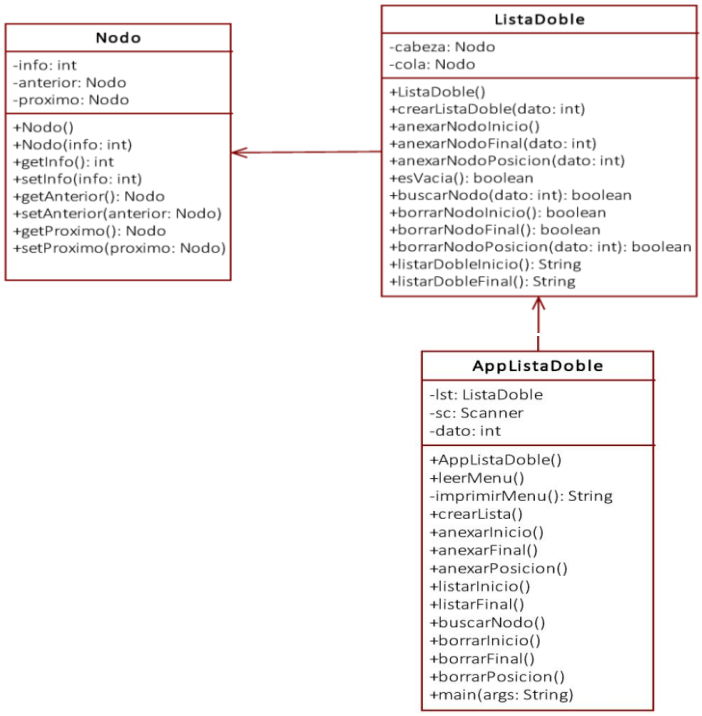

Fig. 10

Aplicaciones y Casos de Uso

Gestión de Inventarios

En la gestión de inventarios las listas doblemente enlazadas son esenciales para mantener un registro preciso y actualizado de los productos almacenados cada nodo en la lista representa un artículo del inventario, almacenando información como el nombre del producto, la cantidad disponible y otros detalles relevantes. La estructura de la lista permite la inserción rápida de nuevos productos y la eliminación eficiente de productos obsoletos o vendidos. Además la capacidad de recorrer la lista en ambos sentidos facilita la auditoría y el control de existencias, permitiendo a los gerentes verificar y actualizar fácilmente el inventario desde cualquier punto de la lista.

Sistemas de Gestión de Archivos

En los sistemas de gestión de archivos las listas doblemente enlazadas se utilizan para organizar y navegar por la estructura de directorios y archivos cada nodo puede representar un archivo o un directorio, y los enlaces dobles permiten moverse fácilmente hacia adelante y hacia atrás dentro de la jerarquía del sistema de archivos esta estructura es particularmente útil para implementar funcionalidades como la navegación por el historial de directorios, la manipulación de listas de archivos recientemente abiertos y la gestión de operaciones de copia y movimiento de archivos la flexibilidad de las listas doblemente enlazadas mejora significativamente la eficiencia y la experiencia del usuario en la interacción con el sistema de archivos.

Aplicaciones de Mensajería y Redes Sociales

En aplicaciones de mensajería y redes sociales las listas doblemente enlazadas son cruciales para gestionar el historial de conversaciones y publicaciones cada nodo en la lista puede representar un mensaje, una publicación o una interacción del

usuario los enlaces dobles permiten a los usuarios desplazarse fácilmente hacia adelante y hacia atrás a través de sus mensajes o publicaciones, proporcionando una navegación fluida y eficiente. Además esta estructura facilita la inserción y eliminación rápida de mensajes lo cual es esencial para la actualización en tiempo real de las conversaciones y las publicaciones en las redes sociales, mejorando la interacción y la experiencia del usuario.

Sistemas de Transporte y Logística

En los sistemas de transporte y logística las listas doblemente enlazadas se utilizan para gestionar rutas de entrega y seguimiento de vehículos cada nodo puede representar un punto de entrega una parada o un vehículo en tránsito los enlaces dobles permiten una planificación y optimización eficientes de las rutas permitiendo a los operadores ajustar dinámicamente las rutas en respuesta a cambios en la demanda condiciones de tráfico u otros factores imprevistos. La capacidad de recorrer la lista en ambos sentidos es fundamental para reordenar las rutas y optimizar los tiempos de entrega, asegurando una logística eficiente y reduciendo costos operativos.

Otros Casos de Uso en la Vida Real

Las listas doblemente enlazadas tienen una amplia variedad de aplicaciones en otros contextos de la vida real en los editores de texto se utilizan para gestionar la lista de operaciones de deshacer y rehacer, permitiendo a los usuarios revertir y repetir acciones fácilmente en los videojuegos, pueden rastrear objetos, personajes y eventos dentro del mundo del juego, facilitando movimientos fluidos y actualizaciones en tiempo real. En aplicaciones científicas y de investigación, las listas doblemente enlazadas son útiles para manejar grandes conjuntos de datos, permitiendo la inserción, eliminación y acceso eficiente a los datos su estructura flexible y eficiente las hace ideales para cualquier aplicación que requiera una manipulación dinámica y rápida de elementos en una lista.

Comparación con otras estructuras de datos

Listas Doblemente Enlazadas: Las listas doblemente enlazadas son estructuras de datos donde cada nodo contiene un enlace al nodo siguiente y otro al nodo anterior esto permite una navegación bidireccional a través de la lista facilitando tanto la inserción como la eliminación de elementos en cualquier posición.

- **Ventajas:** Permiten moverse hacia adelante y hacia atrás lo cual es útil en aplicaciones que requieren acceso en ambas direcciones y las operaciones de inserción y eliminación pueden realizarse en cualquier punto de la lista de manera eficiente sin necesidad de mover otros elementos.
- **Desventajas:** Cada nodo necesita almacenar dos punteros (anterior y siguiente) incrementando el uso de memoria y también son más complejas de implementar y gestionar en comparación con las listas enlazadas simples.

Listas Enlazadas Simples

En las listas enlazadas simples cada nodo contiene un enlace únicamente al nodo siguiente. Esta estructura permite la inserción y eliminación de elementos pero solo en una dirección (hacia adelante). La navegación inversa requiere una mayor complejidad debido a la ausencia de enlaces hacia atrás.

Listas Enlazadas Simples

- **Ventajas:** Cada nodo solo necesita almacenar un puntero al siguiente nodo son eficientes para la inserción y eliminación de elementos en el inicio de la lista.
- **Desventajas:** Solo permiten moverse hacia adelante lo que puede ser una limitación en algunas aplicaciones también la eliminación de nodos intermedios requiere acceso al nodo anterior complicando la operación.

Vectores (Arrays)

Los vectores, o arrays, son estructuras de datos lineales donde los elementos se almacenan en ubicaciones de memoria contiguas permiten un acceso rápido a cualquier elemento mediante un índice, aunque la inserción y eliminación de elementos pueden resultar costosas si implican el desplazamiento de una gran cantidad de elementos.

- **Ventajas:** Permiten el acceso directo a cualquier elemento mediante un índice lo cual es muy rápido y son fáciles de implementar y utilizar.
- **Desventajas:** La inserción y eliminación de elementos pueden requerir mover muchos elementos, lo cual es ineficiente también hay los arrays tienen un tamaño fijo por lo que si se necesita más espacio, se debe crear un nuevo array de mayor tamaño y copiar los elementos lo cual es costoso en términos de tiempo.

Comparación General

- **Flexibilidad**: Las listas doblemente enlazadas ofrecen mayor flexibilidad en comparación con las listas enlazadas simples y los vectores debido a su capacidad de navegación bidireccional.
- **Eficiencia**: Mientras que las listas doblemente enlazadas y las listas enlazadas simples son más eficientes en la inserción y eliminación de elementos los vectores son más eficientes en el acceso aleatorio.
- **Memoria**: Las listas doblemente enlazadas utilizan más memoria debido al almacenamiento de dos punteros por nodo en comparación con las listas enlazadas simples y los vectores.

Cada estructura de datos tiene sus propias ventajas y desventajas y la elección de una sobre otra depende del caso de uso específico y de los requisitos de la aplicación.

Implementación práctica

Caso: Sistema de Gestión de Inventario de una Tienda

En el contexto de una tienda el manejo eficiente del inventario es fundamental para garantizar un flujo de productos ordenado y una gestión adecuada de los recursos. Para ello, se requiere un sistema que permita agregar, eliminar y visualizar los productos en el inventario de manera efectiva.

```
// Caso: Sistema de gestión de inventario de una tienda utilizando listas doblemente enlazadas

#include <iostream>
#include "ListaDoble.h"
using namespace std;
void mostrarMenu() {
    cout << "Seleccione una opción:" << endl;
    cout << "1. Agregar producto al final del inventario" << endl;
    cout << "2. Agregar producto al inicio del inventario" << endl;
    cout << "3. Agregar producto en una posición específica" << endl;
    cout << "4. Eliminar primer producto del inventario" << endl;
    cout << "5. Eliminar último producto del inventario" << endl;
    cout << "6. Eliminar producto en una posición específica" << endl;
    cout << "7. Mostrar inventario de productos" << endl;
    cout << "8. Mostrar inventario de productos en orden inverso" << endl;
    cout << "9. Salir" << endl;
    cout << "Ingrese su opción: ";
}
```

```
int main() {
    ListaDoble inventario;
    int opcion;
    do {
        mostrarMenu();
        cin >> opcion;
        switch (opcion) {
            case 1: {
                int codigo;
                cout << "Ingrese el código del producto: ";
                cin >> codigo;
                inventario.insertarAlFinal(codigo);
                break;
            }
            case 2: {
                int codigo;
                cout << "Ingrese el código del producto: ";
                cin >> codigo;
                inventario.insertarAlInicio(codigo);
                break;
            }
            case 3: {
                int codigo, posicion;
                cout << "Ingrese el código del producto: ";
                cin >> codigo;
                cout << "Ingrese la posición en la que desea insertar el producto: ";
                cin >> posicion;
                inventario.insertarEnPosicion(codigo, posicion);
                break;
            }
            case 4: {
                inventario.eliminarAlInicio();
                break;
            }
            case 5: {
                inventario.eliminarAlFinal();
                break;
            }
            case 6: {
                int posicion;
                cout << "Ingrese la posición del producto que desea eliminar: ";
                cin >> posicion;
                inventario.eliminarEnPosicion(posicion);
                break;
            }
            case 7: {
                cout << "Inventario de productos:" << endl;
                inventario.imprimirAdelante();
                break;
            }
            case 8: {
                cout << "Inventario de productos (en orden inverso):" << endl;
                inventario.imprimirAtras();
                break;
            }
            case 9: {
                cout << "Saliendo del sistema de gestión de inventario..." << endl;
                break;
            }
            default:
                cout << "Opción inválida. Inténtelo de nuevo." << endl;
        }
    } while (opcion != 9);
    return 0;
}
```

Explicación:

El código implementa un sistema de gestión de inventario para una tienda utilizando listas doblemente enlazadas en C++ la estructura de datos principal para representar el inventario es una lista doblemente enlazada implementada en la clase ListaDoble que contiene nodos doblemente enlazados representando los productos en el inventario.

Vinculación con Listas Doblemente Enlazadas:

Inserción y Eliminación: Las operaciones de inserción y eliminación están diseñadas para aprovechar las características de las listas doblemente enlazadas al agregar o eliminar un producto se actualizan adecuadamente los enlaces entre los nodos para mantener la integridad de la lista.

Recorrido de la Lista: El recorrido del inventario hacia adelante y hacia atrás se realiza utilizando los enlaces de los nodos esto permite una visualización eficiente de los productos en el inventario en orden normal y en orden inverso.

Conclusiones

Las listas doblemente enlazadas son una estructura de datos versátil y poderosa que encuentra aplicación en una variedad de escenarios del mundo real su capacidad para permitir el acceso tanto hacia adelante como hacia atrás en la lista, junto con la posibilidad de inserción y eliminación eficientes en cualquier posición las hace ideales para situaciones donde se requiere flexibilidad en la manipulación de datos.

En el ámbito de la gestión de inventarios de tiendas las listas doblemente enlazadas ofrecen una solución eficaz para organizar y mantener un registro de los productos disponibles. Permiten agregar nuevos productos al inventario, eliminar elementos obsoletos y realizar un seguimiento de los cambios en el stock de manera dinámica.

Además, las listas doblemente enlazadas también encuentran aplicación en otros contextos, como en sistemas de gestión de archivos donde se pueden utilizar para mantener un registro de los archivos almacenados y permitir operaciones como inserción, eliminación y búsqueda de archivos de manera eficiente.

Aunque las listas doblemente enlazadas ofrecen muchas ventajas, también presentan algunas limitaciones por ejemplo el consumo de memoria es mayor en comparación con estructuras de datos más simples como los arrays debido a la necesidad de almacenar punteros adicionales para los enlaces entre nodos además las operaciones de inserción y eliminación pueden ser más complejas y requerir un manejo cuidadoso de los punteros para evitar problemas de pérdida de memoria o corrupción de datos.

Bibliografía:

Hernández Bejarano, M. y Baquero Rey, L. E. (2022). Estructuras de datos: fundamentación práctica: (1 ed.). Madrid, RA-MA Editorial. Recuperado de https://elibro.net/es/ereader/espoch/230581?page=215.

González, A. H. (2013). Listas dobles y listas múltiples. https://sedici.unlp.edu.ar/bitstream/handle/10915/33977/Documento_completo.pdf?sequence=3

Manuel, L. M. V. Unidad II. Estructuras de Datos Lineales-Listas. http://ri.uaemex.mx/bitstream/handle/20.500.11799/34465/secme-18391.pdf?sequence=1

Moreno Madrona, N. Estructuras de datos dinámicas. Listas enlazadas. https://core.ac.uk/download/pdf/235859868.pdf

Hernández Bejarano, M. y Baquero Rey, L. E. (2022). Estructuras de datos: fundamentación práctica: (1 ed.). Madrid, RA-MA Editorial. Recuperado de https://elibro.net/es/ereader/espoch/230581?page=215.

Malik, D. S. (2013). Estructuras de datos con C++: (2 ed.). México, D.F, Mexico: Cengage Learning. Recuperado de https://elibro.net/es/ereader/espoch/39995?page=349.

Garrido Carrillo, A. y Valdivia Joaquín, F. (2006). Abstracción y estructura de datos en C++: (ed.). Las Rozas (Madrid), Delta Publicaciones. Recuperado de https://elibro.net/es/ereader/espoch/167034?page=215.

Moisset, D. (s/f-a). Estructuras dinámicas en C++: Listas genéricas doblemente encadenadas. Tutorialesprogramacionya.com. Recuperado el 8 de junio de 2024, de https://www.tutorialesprogramacionya.com/cmasmasya/detalleconcepto.php?punto=42&codigo=173&inicio=30

Moisset, D. (s/f-b). Estructuras dinámicas en C: Listas genéricas doblemente encadenadas. Tutorialesprogramacionya.com. Recuperado el 8 de junio de 2024, de https://www.tutorialesprogramacionya.com/cya/detalleconcepto.php?punto=46&codigo=46&inicio=45

Pozo, S. (2001). C++ con clase. https://conclase.net/c/edd/cap5