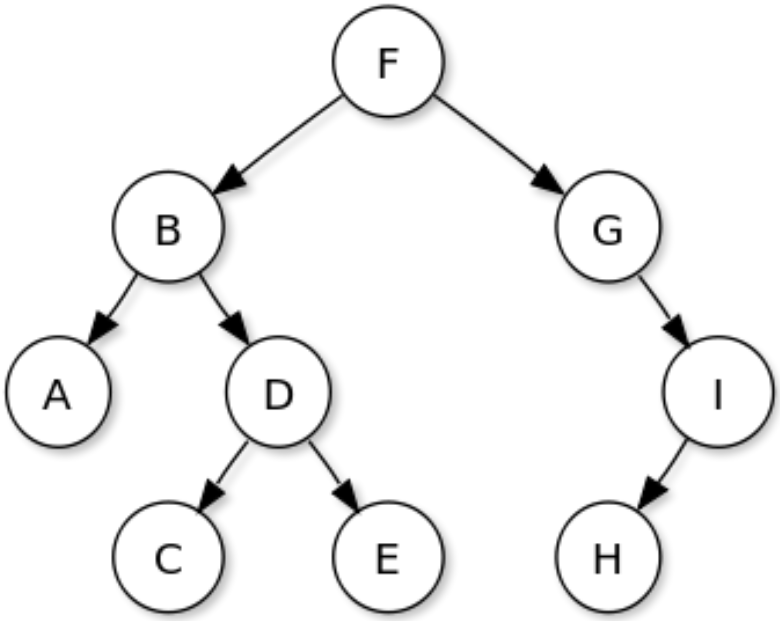
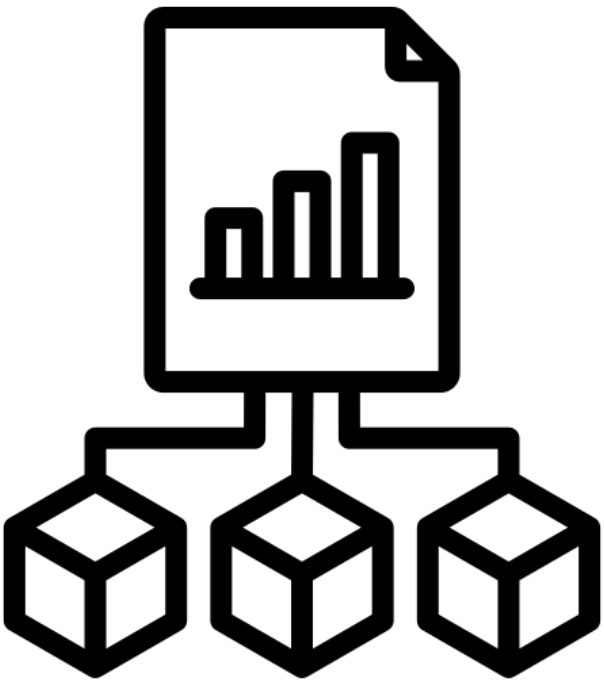
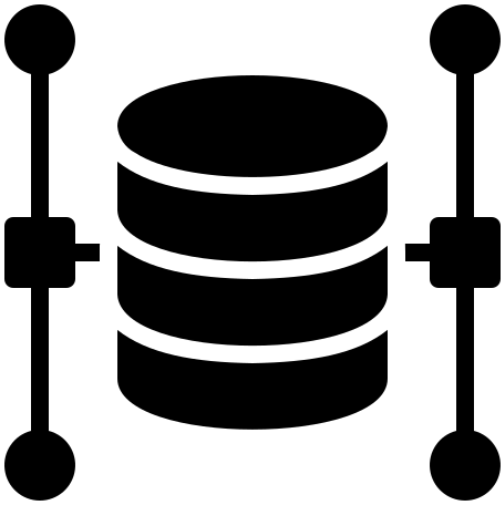
Adrian Anchuela Villoldo
17 June 2023

Estructuras de datos.

Prueba de Evaluación Extraordinaria.







Resumen ejecutivo

Este proyecto se ha centrado en el desarrollo de un sistema de gestión para una empresa de logística. El objetivo principal era diseñar e implementar un software eficiente que facilitara la organización de los paquetes en distintos almacenes, la identificación de estos y su transferencia de un almacén a otro. Para ello, se ha creado un software basado en estructuras de datos como listas doblemente enlazadas y árboles binarios de búsqueda (ABB) para manejar eficientemente los paquetes y las ubicaciones de los almacenes.

El resultado es un sistema robusto que permite realizar operaciones como la adición de nuevos paquetes, la eliminación de paquetes existentes, la transferencia de paquetes entre almacenes y la visualización de los detalles de los paquetes. Asimismo, el software también permite la visualización de los almacenes y su contenido, permitiendo a los usuarios entender claramente la distribución de los paquetes.

En términos de eficiencia, la implementación de las estructuras de datos ha proporcionado un rendimiento notable en términos de la velocidad de búsqueda y manipulación de los datos. Este proyecto proporciona una solución sólida para el manejo y la organización de paquetes en una empresa de logística, facilitando una gestión más eficaz y estructurada.

Introducción

Antecedentes y contexto del proyecto:

En el creciente mundo del comercio electrónico y las operaciones logísticas, es esencial contar con un sistema eficiente de manejo de paquetes. Tradicionalmente, las empresas de logística se han basado en métodos manuales para administrar sus paquetes y almacenes, lo que puede llevar a errores humanos y limitar la eficiencia general de las operaciones. Con el advenimiento de las tecnologías digitales y la creciente necesidad de eficiencia en las operaciones, se ha hecho evidente la necesidad de un sistema de gestión de paquetes informatizado y automatizado.

Declaración del problema o necesidad que aborda el proyecto:

Las empresas de logística enfrentan retos en términos de seguimiento y manejo eficiente de paquetes, así como en el manejo de la información sobre sus almacenes. Existe la necesidad de un sistema que pueda facilitar la organización de los paquetes, proporcionar una fácil identificación de los mismos, y permitir su transferencia de un almacén a otro de manera eficiente. El sistema también debería ser capaz de proporcionar una visión clara de los almacenes y su contenido para una mejor gestión y planificación.

Objetivos y alcance del proyecto:

El objetivo principal del proyecto era diseñar y desarrollar un sistema de gestión de paquetes que facilite las operaciones de las empresas de logística. Los objetivos específicos eran:

1. Crear un sistema que permita agregar, eliminar y transferir paquetes.
2. Desarrollar un sistema que permita a los usuarios visualizar los detalles de los paquetes y los almacenes.
3. Implementar estructuras de datos eficientes para manejar los paquetes y las ubicaciones de los almacenes.

El alcance del proyecto abarcó el desarrollo del software y la implementación de las estructuras de datos. No se consideraron aspectos como la interfaz de usuario y la integración con otros sistemas existentes.

Esquema

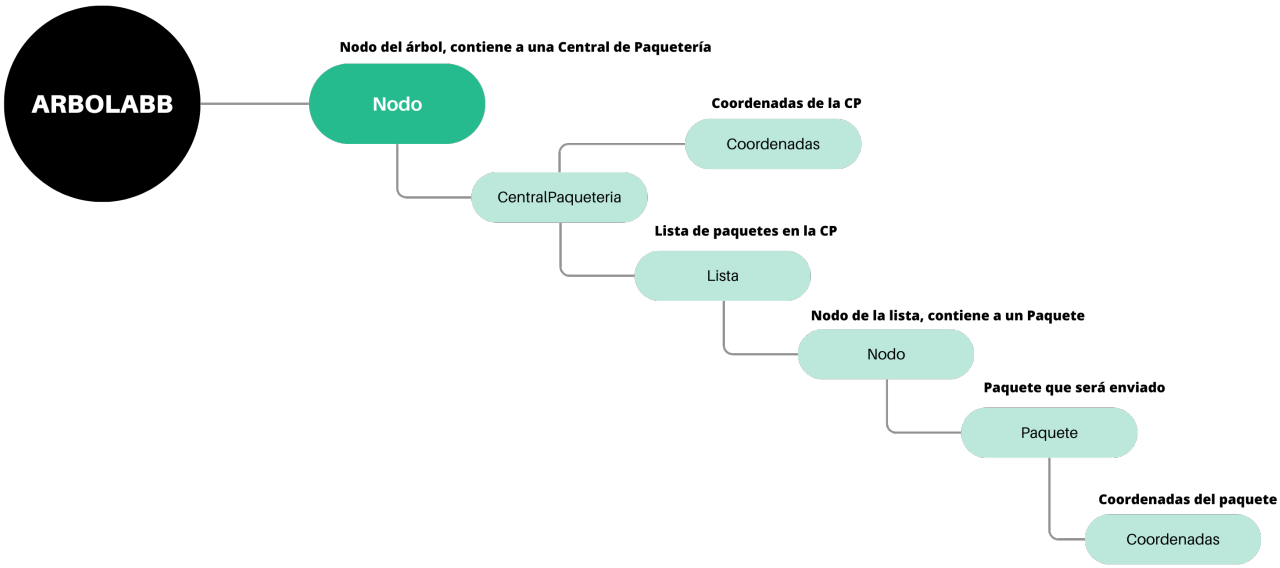
1. **Coordenadas**: Representa las coordenadas de un lugar específico en formato grados, minutos y segundos. Se utiliza para representar la localización de un Paquete y de una Central de Paquetería.
2. **Paquete**: Representa un paquete que debe ser entregado. Contiene un ID, las coordenadas del destinatario, el nombre del destinatario y el número del Centro de Paquetería (CP) al que está asignado.
3. **Nodo**: Es un nodo en una lista doblemente enlazada que se utiliza para representar los paquetes en un Centro de Paquetería. Contiene un paquete (Paquete), así como los nodos anterior y siguiente en la lista (nodo *derecho, *izquierdo)
4. **Lista**: Es una lista doblemente enlazada que se utiliza para representar los paquetes en un Centro de Paquetería. Contiene un puntero al nodo actual, así como al primero y al último nodo de la lista.
5. **CPInfo** e **InformeCP**: Son estructuras de datos que almacenan información sobre un Centro de Paquetería, incluyendo el número de paquetes en el centro y el porcentaje de paquetes en el centro con respecto al total de paquetes en el Centro de Atención al Emprendedor (CAE).
6. **CentralPaqueteria**: Es una estructura de datos que representa un Centro de Paquetería. Contiene un número que identifica al centro, una cadena que representa la localidad del centro y una lista de los paquetes en el centro (Lista Lista).
7. **Nodo**: Es un nodo en un árbol binario que se utiliza para representar los Centros de Paquetería en el sistema. Contiene un Centro de Paquetería (CentralPaqueteria CP) así como punteros al nodo hijo izquierdo y derecho.
8. **ArborlABB**: Es un árbol binario que se utiliza para representar los Centros de Paquetería en el sistema. Contiene un puntero al nodo raíz del árbol, así como al nodo actual.

La relación entre estas estructuras y clases es que una CentralPaqueteria contiene una lista de nodo, cada uno con un Paquete, y cada Paquete contiene Coordenadas. A su vez, estos Centros de Paquetería se almacenan en el ArbolABB mediante nodos del tipo nodo. Las estructuras CPInfo e InformeCP se utilizan para recoger y presentar información de los Centros de Paquetería.

Las relaciones entre estas estructuras de datos son las siguientes:

- Una Central de Paquetería, representada por la estructura centralpaqueteria, contiene una lista de paquetes, implementada mediante la clase lista, que a su vez está compuesta por nodos de la clase nodo, cada uno de los cuales contiene un paquete.
- El arbolABB está compuesto por nodos de la clase nodo y cada nodo tiene una centralpaquetria.

Por lo tanto, el ArbolABB representa la totalidad de las Centrales de Paquetería y sus paquetes organizados de manera eficiente.



Métodos

Métodos de clase ArbolABB

1. **insertarNodo(CentralPaqueteria cp)**: Esta función se utiliza para insertar un nodo con un Centro de Paquetería en el árbol binario. Se comprueba si el número de CP ya existe en el árbol, en cuyo caso se muestra un mensaje de error. Si no, se crea un nuevo nodo y se inserta en el árbol en la ubicación apropiada según su número de CP.
2. **borrarNodo(const int dat)**: Esta función borra un nodo especificado en el árbol. Si el nodo a borrar es una hoja (no tiene hijos), simplemente se borra. Si no es una hoja, se busca un sucesor en el subárbol derecho o un predecesor en el subárbol izquierdo y se intercambian sus valores. Después de eso, el nodo que se intercambió se borra.
3. **postOrden(anodo nodo)**: Esta función imprime los nodos del árbol en postorden. Recorre primero el subárbol izquierdo, luego el subárbol derecho y finalmente visita el nodo actual.
4. **preOrden(anodo nodo)**: Esta función imprime los nodos del árbol en preorden. Visita primero el nodo actual, luego recorre el subárbol izquierdo y finalmente el subárbol derecho.
5. **buscarPaqueteEnCP(anodo nodo, string id, bool t)**: Esta función busca un paquete por su identificador en el árbol. Recorre todo el árbol hasta que encuentra el paquete o hasta que ha visitado todos los nodos.
6. **getAlturaArbol(const int dat)**: Esta función retorna la altura de un nodo específico en el árbol. La altura de un nodo es la longitud del camino más largo desde el nodo hasta una hoja.
7. **numeroNodos()**: Esta función devuelve el número total de nodos en el árbol.
8. **auxContador(nodo *nodo)**: Función auxiliar para la función numeroNodos(). Recorre todos los nodos del árbol e incrementa el contador.
9. **alturaArbol()**: Esta función devuelve la altura del árbol completo. La altura de un árbol es la longitud del camino más largo desde la raíz hasta una hoja.

10. **auxAltura(nodo *nodo, int a)**: Función auxiliar para la función alturaArbol(). Recorre todos los nodos del árbol e incrementa la altura si el nodo actual es una hoja y su altura es mayor que la altura actual.
11. **buscarNodoB(int numero)**: Esta función busca un nodo por su número de CP. Retorna verdadero si encuentra el nodo y falso si no.
12. **sacarlista (int numero)**: Esta función imprime la lista de paquetes en un nodo CP específico.
13. **buscarNodo(int numero)**: Esta función busca un nodo por su número de CP y retorna el nodo si lo encuentra o NULL si no.
14. **pasarCAEaCPdada(int numero, string id, Lista& lis)**: Esta función mueve un paquete de la lista de un Centro de Atención a Clientes a un CP específico.
15. **borrarListaNodo(int numero, string id)**: Esta función borra un paquete específico de la lista de un nodo CP específico.
16. **asignarCP(Paquete pa)**: Este método asigna un código postal (CP) a un paquete si no tiene uno ya. Utiliza la latitud y longitud del paquete para determinar la localidad y luego asigna el CP correspondiente. Si el paquete ya tiene un CP, muestra un mensaje de error.
17. **insertarListaNodo(int numero, Paquete pa)**: Este método inserta un paquete en la lista de paquetes de un nodo (CP) específico. Utiliza la función buscarNodo para encontrar el nodo correcto y luego inserta el paquete en la lista de ese nodo.
18. **asignarCPdada (Paquete pa, int numero)**: Este método asigna un CP específico a un paquete si no tiene uno ya. Utiliza la función buscarNodo para encontrar el nodo correspondiente al CP y luego asigna el CP al paquete.
19. **mudarPaquete(int numeroCP1, int numeroCP2, string id)**: Este método mueve un paquete de un CP a otro. Primero busca el paquete en el CP de origen, lo elimina de allí y luego lo inserta en el CP de destino.
20. **mostrarPaquetesEnCP(int numCP)**: Este método muestra todos los paquetes en un CP específico, ordenados por ID.
21. **recorrerArbolYMostrarCPs(anodo nodo)**: Este método recorre el árbol y muestra todos los CPs y sus localidades.
22. **preOrden(anodo nodo, int totalPaquetesCAE)**: Este método realiza un recorrido en preorden del árbol y muestra información sobre cada CP, incluyendo el número

de CP, la localidad, el número de paquetes en la pila y el porcentaje de paquetes frente al total de la CAE.

23. **informarCPMasMenosPaquetes(anodo nodo, CPInfo& masPaquetes, CPInfo& menosPaquetes, int totalCAE)**: Este método recorre el árbol y encuentra el CP con más y menos paquetes.
24. **buscarPaquetePorID(anodo nodo, string id)**: Este método busca un paquete específico por su ID en todo el árbol.
25. **eliminarPaqueteCPDada2(anodo nodo, string id, int numero, bool& t)**: Este método busca un CP específico en el árbol y, si lo encuentra, elimina el paquete con el ID proporcionado de ese CP.

Métodos de Listas

1. **Lista::~Lista()**: Esta es la función destructora de la clase Lista. Se encarga de liberar la memoria ocupada por la lista, eliminando todos los nodos uno por uno.
2. **Lista::esSiguiente()**: Este método avanza el puntero actual al siguiente nodo en la lista.
3. **Lista::esAnterior()**: Este método retrocede el puntero actual al nodo anterior en la lista.
4. **Lista::esPrimero()**: Este método establece el puntero actual en el primer nodo de la lista.
5. **Lista::esUltimo()**: Este método establece el puntero actual en el último nodo de la lista.
6. **Lista::esActual()**: Este método verifica si el puntero actual es NULL, retornando un valor booleano.
7. **Lista::insertarNodo(int v,char c,Paquete p)**: Este método inserta un nuevo nodo en la lista, ya sea al principio o al final, en función del segundo argumento que toma ('p' para principio, 'f' para final).
8. **Lista::borrarNodo(string ID)**: Este método busca un nodo en la lista que tenga el ID especificado y lo elimina.
9. **Lista::pasarCAEaCP()**: Este método elimina el primer nodo de la lista y devuelve su paquete, es decir, mueve un paquete del CAE al CP.

10. **Lista::buscarPaquete(string numero)**: Este método busca un paquete en la lista por su ID y retorna true si se encuentra, false en caso contrario.
11. **Lista::mostrarPaquete(string id)**: Este método busca un paquete en la lista por su ID y retorna el paquete si se encuentra.
12. **Lista::recorrerLista(int numero)**: Este método recorre la lista y muestra la información de cada nodo en el orden especificado (ascendente o descendente).
13. **Lista::lenLista()**: Este método retorna la longitud de la lista.
14. **Lista::ordenarPorID()**: Este método ordena los nodos de la lista en orden ascendente de ID.
15. **Lista::obtenerTotalPaquetes()**: Este método cuenta y retorna el total de paquetes en la lista.
16. **CentralPaqueteria::totalPaquetes()**: Este método retorna el total de paquetes en la lista de la Central de Paquetería.
17. **Lista::buscarPaquetePorID(string id)**: Este método busca un paquete en la lista por su ID y muestra su información si se encuentra.
18. **Lista::imprimirOrdenado()**: Este método imprime los paquetes en la lista, ordenados en orden descendente por ID.

Métodos globales

1. **generador_id(int& secuencia):** Este método genera un identificador único para un paquete. Primero genera dos dígitos aleatorios y una letra aleatoria. Luego combina estos elementos con una secuencia de cuatro dígitos (la cual se incrementa con cada llamada a la función) para formar un ID único. El ID resultante tiene una estructura similar a "00A0001", "23B0002", etc.
2. **letraDNI(int dni):** Esta función calcula la letra que corresponde a un número de DNI basándose en el método oficial utilizado en España. Para ello, toma el número de DNI, calcula el módulo 23 y usa ese resultado como índice para seleccionar una letra de la cadena de letras "TRWAGMYFPDXBNJZSQVHLCKE".
3. **generador_dni():** Este método genera un número de DNI completo (números y letra). Primero, genera un número aleatorio de 8 cifras. Luego calcula la letra correspondiente utilizando la función letraDNI() y concatena el número y la letra para formar el DNI completo.

4. **generador_latitud():** Esta función genera una latitud aleatoria. Se define una estructura de Coordenadas que contiene grados, minutos y segundos. Los grados son siempre 40, mientras que los minutos son un número aleatorio entre 6 y 35. Los segundos siempre son 0. La función retorna estas coordenadas.
5. **generador_longitud():** Similar a generador_latitud(), esta función genera una longitud aleatoria. La estructura de Coordenadas es definida con grados como -3, minutos como un número aleatorio entre 5 y 35, y segundos como 30. Estas coordenadas se retornan.

En conjunto, estos métodos se usan para generar información aleatoria necesaria para el sistema de seguimiento de paquetes. El sistema se puede personalizar cambiando los valores y rangos en estas funciones según sea necesario.

Funcionamiento

El programa simula el proceso de una empresa de logística que tiene varias Centrales de Paquetería (CP) y un Centro de Almacenamiento de Envíos (CAE). Aquí tienes una explicación más detallada del funcionamiento:

Inicialización del programa: Al inicio, se definen una serie de constantes que describen los parámetros operativos de la simulación, tales como el número de centrales de paquetería (N1), el número de paquetes generados aleatoriamente (N2), el número de paquetes enviados a las centrales en cada paso de ejecución (N3), y el número total de paquetes enviados a las centrales al finalizar el periodo de tiempo evaluado (N4).

Creación de las Centrales de Paquetería y los Paquetes: Inicializa una lista (CAE) y un árbol de búsqueda binaria (ArbolABB a). Luego, en un bucle, se crean N1 CPs y se insertan en el árbol binario. En otro bucle, se generan N2 paquetes aleatorios que se insertan en la lista de CAE.

Distribución inicial de paquetes: Entra en un bucle en el que, en cada paso, se seleccionan N3 paquetes del CAE y se asignan a una CP de manera aleatoria. Esto se repite hasta que se hayan asignado N4 paquetes a las CPs.

Interfaz del menú interactivo: Después de esta fase inicial, el programa entra en un bucle de menú interactivo. El usuario puede elegir entre diversas operaciones, que incluyen:

- **Opción 1 - Insertar una nueva Central de Paquetería (CP):** El usuario proporciona los detalles de una nueva CP, incluyendo su ID. El programa crea un nuevo nodo con estos detalles y lo inserta en el árbol de búsqueda binaria.
- **Opción 2 - Borrar una CP existente:** El usuario proporciona el ID de una CP existente. El programa busca este nodo en el árbol y, si lo encuentra, lo elimina.
- **Opción 3 - Mostrar los paquetes de una CP específica en orden ascendente por ID:** El usuario proporciona el ID de una CP. El programa busca este nodo en el árbol y, si lo encuentra, muestra todos los paquetes en esa CP en orden ascendente por ID.
- **Opción 4 - Recorrer y mostrar todos los nodos del árbol en pre-orden:** El programa recorre todos los nodos del árbol en pre-orden (es decir, visitando primero la raíz, luego el subárbol izquierdo y finalmente el subárbol derecho) y los muestra.

- **Opción 5 - Informar cuál es la CP con más y menos paquetes:** El programa recorre todas las CPs en el árbol, contando los paquetes en cada una. Informa cuál CP tiene el mayor y el menor número de paquetes.
- **Opción 6 - Buscar un paquete específico por su ID:** El usuario proporciona el ID de un paquete. El programa busca este paquete tanto en el CAE como en todas las CPs.
- **Opción 7 - Extraer un paquete específico de una CP específica:** El usuario proporciona el ID de un paquete y el ID de una CP. El programa busca el paquete en la CP especificada y, si lo encuentra, lo extrae (lo elimina de la CP).
- **Opción 8 - Mover un paquete del CAE a una CP específica:** El usuario proporciona el ID de un paquete y el ID de una CP. El programa busca el paquete en el CAE y, si lo encuentra, lo mueve a la CP especificada.
- **Opción 9 - Mover un paquete de una CP a otra:** El usuario proporciona el ID de un paquete y los ID de dos CPs. El programa busca el paquete en la primera CP y, si lo encuentra, lo mueve a la segunda CP.
- **Opción 10 - Mostrar todos los paquetes en el CAE en orden descendente por ID:** El programa muestra todos los paquetes en el CAE en orden descendente por ID.
- **Opción 11 - Continuar con la distribución de paquetes desde el CAE a las CPs:** El programa selecciona una cantidad de paquetes del CAE y los asigna de manera aleatoria a las CPs, hasta que se haya asignado un número predefinido de paquetes.
- **Opción 0 - Salir del programa:** Termina la ejecución del programa.

Cada una de estas opciones realiza diferentes operaciones utilizando los métodos de las clases ArbolABB y Lista.

Por ejemplo, la opción 1 permite al usuario ingresar manualmente los detalles de una nueva CP, que se agrega al árbol binario. Por otro lado, la opción 6 realiza una búsqueda de un paquete en particular, ya sea en el CAE o en todas las CPs, utilizando su ID.

Cierre del programa: El programa se mantendrá en el bucle del menú hasta que el usuario decida salir seleccionando la opción 0. Esto terminará la ejecución del programa.

De esta manera, el programa simula las operaciones de gestión de paquetes en una red de centrales de paquetería, permitiendo al usuario interactuar con el sistema y ver cómo los paquetes son distribuidos y gestionados.

Problemas y dudas

1. **Problema:** Desafíos al implementar y gestionar un árbol de búsqueda binaria para manejar las Centrales de Paquetería (CP).
   Solución: Me di cuenta de la utilidad de los árboles de búsqueda binaria en este contexto y aprendí a usarlos eficazmente. Desarrollé habilidades para insertar nuevos nodos, eliminar nodos existentes y recorrer el árbol de manera eficiente.

2. **Problema:** Dificultades para implementar la funcionalidad de los paquetes en las CPs y en el Centro de Acopio y Envío (CAE).
   Solución: Estudié diversas estructuras de datos y comprendí que las pilas y colas serían particularmente útiles para manejar los paquetes en el CAE y las CPs. Implementé esta funcionalidad con éxito.

3. **Problema:** Desafíos para organizar y presentar la información en un menú interactivo.
   Solución: Diseñé un menú estructurado con diversas opciones que permiten la interacción eficiente con las CPs, el CAE y los paquetes. También implementé con éxito cada opción del menú.

4. **Problema:** Incertidumbre sobre cómo manejar la distribución de paquetes desde el CAE a las CPs.
   Solución: Desarrollé una solución efectiva en la que un número predefinido de paquetes se selecciona del CAE y se distribuye de manera aleatoria a las CPs cada vez que se selecciona la opción correspondiente en el menú.

5. **Problema:** Dificultades para buscar un paquete en todas las CPs.
   Solución: Comprendí que podría realizar una búsqueda en profundidad a través de todos los nodos del árbol (CPs) para encontrar un paquete específico y logré implementar con éxito esta funcionalidad.

A lo largo de esta práctica, superé cada uno de estos desafíos de manera exitosa, fortaleciendo mis habilidades y conocimientos en la gestión de estructuras de datos complejas y el desarrollo de interfaces de usuario interactivas.