PRÁCTICA 2

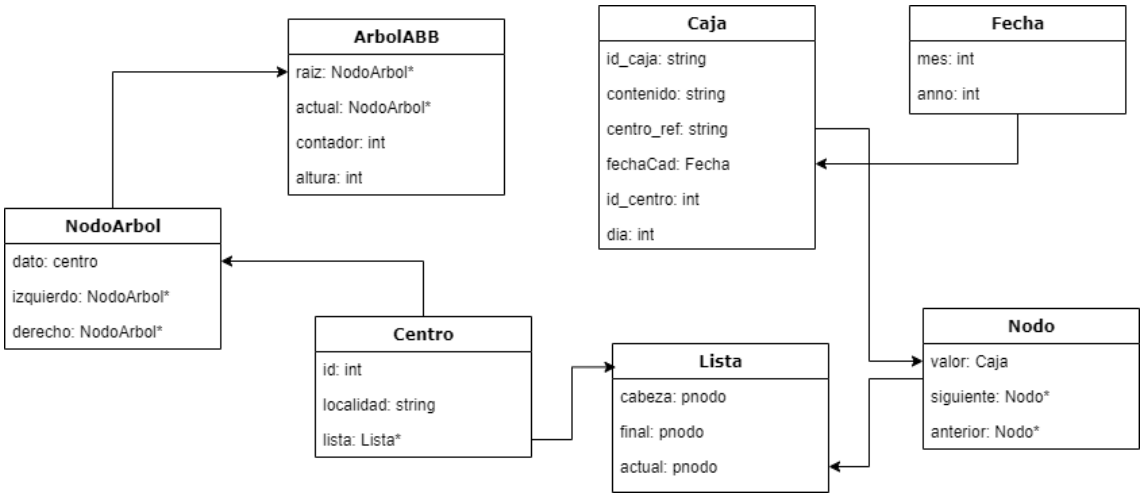
Simulación del funcionamiento de una red de envío de ayuda humanitaria

Universidad de Alcalá
Grado en Ingeniería de Sistemas de Información

Tatiana Huaxuan García Vergara
DNI: 51744543V

Tabla de contenido

1. Dibujo de los TAD´s implementados, estructuras de datos definidas y relación entre ellos.



2. Descripción de los métodos y funciones implementados.

2.1. Métodos para las listas.

Además de los métodos para las listas doblemente enlazadas que ya se vieron en clase, se han implementado las siguientes funciones:

- **contarCajas()** -> se inicia un contador en cero y utiliza un puntero temporal que se inicializa con la cabeza de la lista. Mediante un bucle while, la función recorre la lista nodo por nodo, incrementando el contador en cada iteración. El puntero temporal se desplaza al siguiente nodo en cada paso. Una vez que el puntero alcanza el final de la lista (cuando es NULL), la función devuelve el valor final del contador, que representa la cantidad total de cajas en la lista.

- **contarCajasPais(string pais)** -> funciona parecido a la función **contarCajas()**, pero realiza un conteo de cajas en una lista enlazada cuyos identificadores, comienzan con una cadena específica que se pasa como parámetro (pais).

- **borrarCajaPorID(string id)** -> elimina una caja cuyo identificador coincide con el string id que recibe como parámetro. Se utiliza un par de punteros, temp y anterior, para recorrer la lista. Si encuentra la caja, elimina la caja y dependiendo de la posición de la caja a eliminar, se ajustan los punteros de la lista.

- **ordenarCajasDia()** -> ordena la lista de cajas según el día de llegada de las cajas. Se utilizan dos punteros, ptr1 y ptr2, para comparar y realizar intercambios entre los nodos de la lista. Se emplea el método de burbuja para comparar y ordenar las cajas en la lista según el modo especificado, que consiste en comparar repetidamente pares de elementos adyacentes y realizar intercambios si están en el orden incorrecto. El proceso se repite hasta que no se requieren más intercambios.

- **BuscarCaja(string id) ->** busca una caja por su ID en una lista enlazada, devolviendo la caja correspondiente si se encuentra, o una caja vacía si no. Utiliza un bucle while para recorrer los nodos de la lista y compara el ID de la caja en cada nodo con el ID proporcionado. Si encuentra una coincidencia, devuelve la caja asociada a ese nodo; de lo contrario, devuelve una caja vacía.

2.2. Métodos para los árboles.

Además de los métodos para los árboles que ya se vieron en clase, se han implementado las siguientes funciones:

- **CajasTotales()** -> la función se encarga de calcular y devolver la cantidad total de cajas almacenadas en un árbol de búsqueda binaria (ABB). Primero, inicializa un contador en cero y luego invoca la función auxiliar auxCajasTotales pasando como parámetro el nodo raíz del árbol.

- **auxCajasTotales(NodoArbol *nodo) ->** la función recibe un nodo de tipo NodoArbol como parámetro y cuenta las cajas almacenadas en ese nodo y sus descendientes. La cantidad de cajas en el nodo actual se obtiene llamando al método contarCajas de la lista de cajas asociada al nodo y se suma al contador. Luego, la función se llama recursivamente para los nodos hijos izquierdo y derecho si existen, asegurándose de explorar todo el árbol.

- **CajasTotalPais(string pais) ->** la función se encarga de calcular y devolver la cantidad total de cajas asociadas a un país específico en un árbol de búsqueda binaria (ABB). Inicializa un contador en cero y luego invoca la función auxiliar auxCajasPais pasando como parámetros el nodo raíz del árbol y el nombre del país que se desea buscar.

- **auxCajasPais(NodoArbol *nodo, string pais) ->** la función recibe un nodo de tipo NodoArbol y el nombre del país como parámetros. En cada nodo, la cantidad de cajas asociadas al país especificado se obtiene llamando al método contarCajasPais de la lista de cajas asociada al nodo, y esa cantidad se suma al contador. Luego, la función se llama recursivamente para los nodos hijos izquierdo y derecho si existen, asegurándose de explorar todo el árbol.

- **centroConMasCajasAPais(string pais)** -> devuelve el centro que tiene la mayor cantidad de cajas asociadas a un país específico. Inicia el proceso llamando a la función auxiliar auxCentroConMasCajasAPais, pasando la raíz del árbol, el nombre del país de interés y una referencia a un centro que almacenará la información del centro con la máxima cantidad de cajas.

- **auxCentroConMasCajasAPais(NodoArbol *nodo, string pais, centro& centroMaximo) ->** Para cada nodo, calcula la cantidad de cajas asociadas al país proporcionado mediante la función contarCajasPais. Si esta cantidad supera la previamente registrada en el contador, actualiza el contador y actualiza el centro máximo. Luego, la función se llama recursivamente para los subárboles izquierdo y derecho del nodo actual. Este proceso continúa hasta recorrer todos los nodos del árbol.

- **centroConMasCajasEnTotal() ->** devuelve el centro que tiene la mayor cantidad total de cajas. Similar a la función anterior, esta función también inicia el proceso llamando a la función auxiliar auxCentroConMasCajasEnTotal, pasando la raíz del árbol y una referencia a un centro que almacenará la información del centro con la máxima cantidad total de cajas.

- **auxCentroConMasCajasEnTotal(NodoArbol* nodo, centro& centroMaximo) ->** Para cada nodo, obtiene la cantidad total de cajas. Similar a la función anterior, si esta cantidad supera la previamente registrada en el contador, se actualiza el contador y el centro máximo. Luego, la función se llama recursivamente para los subárboles izquierdo y derecho del nodo actual. Este proceso continúa hasta recorrer todos los nodos del árbol.

- **centroConMenosCajasEnTotal() ->** devuelve el centro que tiene la menor cantidad total de cajas en todos los países. La función inicia el proceso inicializando un contador. Luego, crea un objeto centroMinimo que almacenará la información del centro con la mínima cantidad total de cajas. Posteriormente, se invoca la función auxiliar auxCentroConMenosCajasEnTotal, pasando la raíz del árbol y una referencia al objeto centroMinimo.

- **auxCentroConMenosCajasEnTotal(NodoArbol* nodo, centro& centroMinimo) ->** Para cada nodo, calcula la cantidad total de cajas mediante la función contarCajas(). Si esta cantidad es menor que el valor registrado en el contador, se actualiza el contador y se actualiza el centro mínimo. Luego, la función se llama recursivamente para los subárboles izquierdo y derecho del nodo actual. Este proceso continúa hasta recorrer todos los nodos del árbol.

- **obtenerNodoAleatorio()** -> realiza un recorrido aleatorio en el árbol para seleccionar de manera aleatoria un nodo y devuelve el centro asociado a dicho nodo. Para ello, calcula la altura del árbol y determina un número aleatorio de movimientos limitado por esa altura. A continuación, inicia desde la raíz del árbol y realiza los movimientos aleatorios hacia la izquierda o la derecha en cada iteración del bucle. La función devuelve el centro asociado al nodo final alcanzado después de los movimientos aleatorios.

- **distribuirCajas(Caja caja)** -> distribuye una caja específica en el árbol binario de búsqueda (ABB) de centros. El método toma un objeto Caja como parámetro. La función utiliza un bucle while para recorrer el árbol en busca del centro donde hay que insertar la caja. Dentro del bucle, se compara el ID del centro con el ID_centro de la caja. Si son iguales, se llama al método insertarNodo para agregar la caja a la lista. Si es menor que el ID del centro, la búsqueda continúa en el subárbol izquierdo; de lo contrario, se mueve al subárbol derecho.

- **obtenerNodo(int id)** -> busca un centro con un ID específico en un árbol binario de búsqueda (ABB). Utiliza un bucle while para recorrer el árbol, comparando el ID del centro actual con el ID proporcionado. Si encuentra un centro con el ID deseado, devuelve ese centro. Si el ID proporcionado es menor, la búsqueda continúa en el subárbol izquierdo; de lo contrario, se mueve al subárbol derecho. Si el bucle termina sin encontrar el nodo con el ID específico, la función devuelve un centro vacío.

- **BuscarCajaPorID (string id, NodoArbol *nodo) ->** busca una caja por su ID en un árbol binario de búsqueda (ABB) a partir de un nodo dado. Primero, verifica si el nodo actual es nulo. Si no lo es, la función utiliza el método BuscarCaja de la lista enlazada asociada al nodo para intentar encontrar la caja con el ID proporcionado. Si la caja se encuentra, se imprime la cabecera y se muestra la información de la caja. Si no se encuentra en el nodo actual, la búsqueda continúa en los subárboles izquierdo y derecho llamando recursivamente a la función para estos nodos.

- **ObtenerRaiz()** -> devuelve la raíz.

2.3. Función para recorridos del árbol y para imprimir datos.

- **Mostrar(centro &c)** -> La función toma una referencia de un centro como parámetro e imprime información relevante, como el ID y la localidad del centro. Además, si la lista de cajas asociada al centro está inicializada, muestra el número de cajas utilizando el método contarCajas() de esa lista.

- **MostrarID(centro &c)** -> la función toma como parámetro una referencia de un centro e imprime su id.

- **imprimirDatosCC(int id, ArbolABB &abb) ->** la función imprime los datos contenidos en la lista enlazada asociada al nodo que tiene el mismo valor id en el árbol binario de búsqueda proporcionado.

- **ImprimirCabecera()** -> esta función imprime la cabecera de las cajas.

- **MostrarCaja(Caja caja)** -> recibe una caja como argumento e imprime sus datos.

2.4. Funciones para el menú.

El menú tiene nueve opciones y se han creado funciones para cada opción del menú.

• **introducirCC(ArbolABB &abb):** La función solicita al usuario ingresar un ID para un Centro de Clasificación (CC) y verifica si ya existe en el árbol. Si el ID no está en uso, se solicita la localidad del nuevo centro y se crea una instancia de la clase Lista asociada a él. El centro se inserta en el árbol de búsqueda binaria y se pregunta al usuario si desea introducir otro centro.

• **eliminarCC(ArbolABB &abb):** La función solicita al usuario ingresar el ID del centro que desea eliminar y muestra la lista de IDs existentes. Luego, verifica si existe el centro en el árbol y lo elimina si es el caso. Se informa al usuario sobre el resultado y se pregunta si desea eliminar otro centro.

• **buscarCCPorID(ArbolABB &abb):** La función solicita al usuario ingresar el ID del Centro de Clasificación que desea buscar y muestra la lista de IDs existentes en el árbol. Luego, el usuario ingresa el ID deseado y se muestra información detallada sobre el centro si existe, incluyendo el número de cajas y la cola de cajas. Se pregunta al usuario si desea realizar otra búsqueda.

• BuscarCajaPorID(ArbolABB &abb): La función solicita al usuario introducir el ID de la caja que desea buscar, realiza la búsqueda en el árbol y muestra los datos de la caja si la encuentra. Se pregunta al usuario si desea buscar otra caja.

• borrarCaja(ArbolABB &abb): La función solicita al usuario introducir el ID del Centro de Clasificación al que pertenece la caja que desea eliminar y muestra la lista de IDs existentes. Luego, se procede a eliminar la caja si existe, y se pregunta al usuario si desea eliminar otra caja.

• moverCaja(ArbolABB &abb): La función guía al usuario para mover una caja de un Centro de Clasificación a otro. Se solicita el ID del centro de origen y destino, así como el ID de la caja a mover. Se realiza la operación y se pregunta si desea mover otra caja.

• estadísticas(ArbolABB &abb): La función presenta diversas estadísticas sobre el sistema de ayuda humanitaria, incluyendo el número total de cajas, la distribución por destino y detalles sobre los Centros de Clasificación con mayor y menor cantidad de cajas.

• CrearYDistribuirCajas(ArbolABB &abb, int n): La función crea y distribuye n cajas aleatorias en los Centros de Clasificación presentes en el árbol, mostrando cada caja generada antes de su distribución.

- **ordenarLista(ArbolABB &abb)** -> la función permite al usuario seleccionar un Centro de Clasificación mediante su ID y ordenar la lista de cajas asociada según el día de llegada de la caja . Después de ordenarla, se muestra la lista de cajas y se pregunta al usuario si desea ordenar las cajas de otro Centro de Clasificación.

2.5. Funciones para la creación de cajas.

Las cajas se crean de forma aleatoria, así que hay métodos para crear cada elemento de una caja de forma aleatoria y luego otro método, que junta todos los anteriores para hacer la caja.

- **calcularPosicionAleatoria(int n) ->** calcula un número aleatorio, siendo el máximo el número entero que recibe como argumento.

- **fechaAleatoria() ->** calcula un mes y un año de forma aleatoria, crea una fecha y la devuelve. Los años que se crean están en un rango de 2023 a 2041, para que las fechas de caducidad sean un poco más realistas.

- **idAleatorio() ->** la función utiliza un bucle para generar cuatro dígitos aleatorios y los concatena en una cadena. Luego, elige de manera aleatoria una letra como origen de la lista de letras de la A a la Z. Además, selecciona un destino al azar de la lista de destinos predefinidos (MAR, LIS, GRE). Finalmente, combina el destino, los dígitos y el origen en una cadena y devuelve el ID resultante.

- **contenidoAleatorio() ->** se crea un contenido a partir de crear una posición aleatoria y elegir el elemento que este en esa posición en la lista de posibles contenidos y devuelve ese contenido.

- **cajaAleatoria(string origen) ->** la función utiliza un árbol de búsqueda binaria para obtener un centro de referencia aleatorio para asegurarse que la caja tenga a un centro a donde ir y extrae la localidad y el ID del centro. Genera un ID, contenido y fecha de

caducidad aleatorio mediante sus respectivas funciones. Además, si el contenido es un objeto que no puede caducar, la fecha se convierte en 12/2100. Devuelve la caja que se crea con los elementos anteriores.

2.6. Funciones para la creación de centro.

- **centroAleatorio() ->** la función asigna un ID único al centro, generado aleatoriamente en el rango de 100 a 999. Elige una localidad aleatoria de una lista predefinida de centros. Inicializa una nueva instancia de la clase Lista para el centro. Crea una instancia de la estructura de datos 'centro' y asigna los valores generados a sus atributos. Retorna el centro recién creado con la información aleatoria y la lista inicializada.

3. Explicación del funcionamiento del programa.

El programa simula una red de ayuda humanitaria, utilizando un Árbol de Búsqueda Binaria para gestionar la información de diferentes centros de clasificación.

Inicialmente, se generan aleatoriamente 10 centros y se muestra el estado inicial del árbol. Luego, se crean y distribuyen cajas. Después, se vuelve a mostrar el estado del árbol.

Finalmente, aparece un menú que permite al usuario realizar diversas operaciones, como insertar o eliminar centros, buscar cajas por ID, transferir cajas entre centros, y más. El bucle principal del programa continúa hasta que el usuario elige salir.

4. Problemas encontrados durante el desarrollo de la práctica y solución adoptada.

Se ha tenido que adaptar la función para la creación de cajas, ya que con la función original se podían llegar a crear cajas que fueran de un centro de clasificación que no estuviera en el árbol de binario de búsqueda. Se soluciono esto, haciendo que a la hora de crear la caja aleatoria se seleccionara un nodo aleatorio del árbol y se cogiera de ahí tanto en nombre del centro de referencia como id para que de esta forma coincidiera.

También, a la hora de crear los centros, aunque el id no se repite, la localidad si puede llegar a repetirse. Esto implica que puede haber varios centros de clasificación con diferentes identificadores pero ubicados en la misma localidad.