Práctica Extraordinaria Estructuras de Datos

María Alonso Arribas 03222997F
Laura Herranz Vázquez 03226822Z

1- Introducción

Esta práctica tiene como objetivo la simulación del funcionamiento de un punto de control de aduanas en un aeropuerto, mediante la implementación y gestión de varios Tipos Abstractos de Datos (TAD). Un TAD es una estructura que define un conjunto de datos y las operaciones que pueden realizarse sobre ellos, independientemente de su implementación interna.

El sistema simula la llegada de pasajeros, la asignación a distintos boxes de control según prioridades definidas, y el registro de información relevante para análisis posteriores. Esta simulación permite estudiar el comportamiento del sistema y validar estructuras de datos dinámicas y eficientes para gestión en tiempo real.

Las estructuras empleadas para gestionar a los pasajeros son estructuras como pilas, colas, listas y árboles binarios de búsqueda.

2- TAD’S creados

En esta práctica se han diseñado e implementado varios Tipos Abstractos de Datos. A continuación se detallan:

- Pasajero: este TAD modela la unidad fundamental del sistema, haciendo referencia a cada persona que debe ser gestionada en el control de aduanas. Permite abstraer el flujo individual de un viajero desde su llegada al aeropuerto hasta su salida por uno de los puntos de control.
- Box: representa una unidad de atención dentro del sistema. Cada box atiende a los pasajeros uno a uno e incorpora su propia cola de espera personalizada por prioridad. Su existencia es dinámica, ya que los boxes son creados o eliminados automáticamente durante la ejecución dependiendo de la carga del sistema, osea, de la cantidad de pasajeros atendidos en un mismo box.
- Cola: se trata de la cola que acompaña a cada box. Gestiona el orden de espera de los pasajeros según criterios de prioridad del país de destino. Aunque conceptualmente es una cola, se ha personalizado para insertar en posición ordenada: aquellos con prioridad más alta (número más bajo) se atiende primero.
- Lista: es una lista dinámica que almacena todos los boxes activos en el sistema. Permite gestionar el número de boxes creados y eliminados dinámicamente, así como acceder directamente a cualquiera de ellos según su posición.

- Pila: contiene todos los pasajeros pendientes de llegar al aeropuerto, organizados por su minuto de llegada. El uso de una pila simula la carga inicial del sistema, ya que los pasajeros están ordenados de tal forma que el primero en salir es el que llega antes.
- ListaPasajeros: esta estructura representa una colección de pasajeros ya atendidos con destino a un país concreto. Es utilizada internamente por cada nodo del árbol binario de búsqueda (ABB).
- ABB (Árbol Binario de Búsqueda): implementa un árbol binario ordenado por nombre de país de destino. Permite acceder rápidamente a todos los pasajeros que viajan a un país, calcular estadísticas por país, realizar recorridos inorden y preorden, y buscar pasajeros de forma eficiente.
- Aeropuerto: clase controladora que centraliza la gestión de todos los elementos anteriores. Contiene la pila de pasajeros, la lista de boxes, el minuto actual y todas las funciones clave de simulación.

3- Definición de las operaciones del TAD

TAD Pasajero

Atributos principales**:**

- `int ID`: identificador único del pasajero, lo diferencia del resto.
- `int minLlegada`: minuto exacto en el que el pasajero llega al aeropuerto.
- `int duracionControl`: tiempo que debe permanecer en el box durante el control.
- `string pais`: país de origen o procedencia del pasajero.
- `int prioridad`: Nivel de urgencia asignado según criterios externos.
- `int IDBox`: Identificador del box en el que fue atendido (si ya ha pasado por uno).
- `int tiempoTotal`: Minuto en el que el pasajero sale del sistema (finaliza el control).

Operaciones más relevantes:

Operación	**Parámetros**	**Retorno**	**Descripción**

Pasajero(int, int, int, string, int)	ID, minLlegada, duracionControl, pais, prioridad	Constructor	Construye un pasajero con datos específicos.
getID() const	-	int	Devuelve el identificador único del pasajero.
getMinLlegada()	-	int	Devuelve el minuto de llegada al aeropuerto.
getDuracionControl() const	-	int	Devuelve el tiempo que durará el control de aduanas.
getPais() const	-	string	Devuelve el país de procedencia.
getPrioridad()	-	int	Devuelve la prioridad asignada al pasajero.
getTiempoTotal()	-	int	Devuelve el minuto en que el pasajero finaliza el control.
setID(int)	id	void	Modifica el identificador del pasajero.
setMinLlegada(int)	minuto	void	Modifica el minuto de llegada.

setDuracionControl(int)	duracion	void	Modifica la duración del control aduanero.
setPais(string)	país	void	Modifica el país de procedencia.
setPrioridad(int)	prioridad	void	Modifica la prioridad del pasajero.
setIDBox(int)	IDBox	void	Asigna el ID del box en el que es atendido.
setTiempoTotal(int)	tiempo	void	Establece el minuto en que finalizó el control.
calcularTiempoEstancia() const	-	int	Calcula la diferencia entre el tiempo de salida y llegada, representando la estancia total.

TAD Box:

Atributos principales**:**

- `int IDBox`: identificador único del box dentro de la lista.
- `Pasajero pasajeroActual`: contiene los datos del pasajero que está siendo atendido en ese momento.
- `Cola colaEsperaBox`: cola local del box para gestionar pasajeros en espera.
- `bool ocupado`: indica si el box está ocupado o no.
- `int tiempoRestante`: número de minutos que le quedan al pasajero actual para terminar el control

Operaciones:

Operación	Parámetros	Retorno	Descripción
Box()	-	Constructor	Inicializa un box vacío con ID predeterminado (-1).
Box(int id)	id	Constructor	Inicializa un box con un ID específico.
getIDBox() const	-	int	Devuelve el identificador único del box.
getPasajeroActual() const	-	Pasajero	Devuelve el pasajero que está siendo atendido actualmente.
getTiempoRestante() const	-	int	Devuelve el tiempo restante para atender al pasajero actual.
colaEsperaVacia() const	-	bool	Indica si la cola de espera está vacía.
asignarPasajero(const Pasajero&)	pasajero	void	Asigna un pasajero al box y establece su tiempo de control.
liberarBox()	-	void	Libera el box tras finalizar el control del pasajero.

actualizarTiempo(int)	minutos	void	Reduce el tiempo restante en la atención según minutos dados.
estaLibre() const	-	bool	Indica si el box está disponible para atender.
agregarPasajeroAEspera(Pasajero)	pasajero	void	Añade un pasajero a la cola de espera según prioridad.
obtenerPrimerPasajeroEnEspera()	-	Pasajero	Devuelve el primer pasajero en la cola de espera.
mostrarInfo() const	-	void	Muestra información detallada del estado del box.

TAD Cola:

Atributos principales:

- `NodoCola* primero`: puntero al primer nodo de la cola, es decir, al pasajero que será atendido próximamente.
- `NodoCola* ultimo`: puntero al último nodo insertado, por donde llegan los nuevos pasajeros.
- `int longitud`: número de elementos actuales en la cola, útil para estadísticas y decisiones de asignación.

Operaciones:

Operación	**Parámetros**	**Retorno**	**Descripción**

inicio() const	-	Pasajero	Devuelve el primer Pasajero de la cola.
fin() const	-	Pasajero	Devuelve el últimoPasajero de la cola.
encolar(Pasajero elemento, int prioridad)	pasajero, prioridad	void	Añade un Pasajero teniendo en cuenta su prioridad para determinar su posición.
desencolar()	-	Pasajero	Elimina el primer pasajero y lo devuelve.
get_longitud() const	-	int	Devuelve la longitud de la cola.
es_vacia() const	-	bool	Devuelve true si la cola está vacía.
mostarCola()	pasajero	void	Muestra los elementos de la cola o si está vacía.

TAD Lista:

Atributos principales:

- `Box valor`: valor de tipo Box utilizado internamente.
- `pNodoLista cabeza`: puntero al primer nodo de la lista enlazada, que representa el inicio de la secuencia de boxes.
- `pNodoLista cola`: puntero al último nodo de la lista, que facilita inserciones al final.

Operaciones:

Operación	Parámetros	Retorno	Descripción
Lista()	-	Constructor	Inicializa una lista vacía con cabeza y cola apuntando a nullptr.
insertarInicio(Box)	Box	void	Inserta un nuevo nodo al inicio.
insertarFinal(Box)	Box	void	Inserta un nuevo nodo al final.
insertarEnposicion(Box, int)	-Box, int	void	Inserta un nuevo nodo en la posición especificadal.
eliminar(Box)	Box	void	Elimina un nodo con un valor en específico.
buscar(int)	int	Box*	Busca y devuelve un puntero al Box

			con ID igual a id, o nullptr si no se encuentra.
obtenerEnPosicion(int)	int	Box&	Devuelve el Box en la posición especificada.
actualizarTiempo(int)	minutos	void	Reduce el tiempo restante en la atención según minutos dados.
longitud()	-	int	Devuelve l longitud de la lista.
inicio()	pasajero	Box	Devuelve el primer Box de la lista.
fin()	-	Box	Devuelve el último Box de la lista.
mostrar()	-	void	Muestra todos los elementos de la lista.

TAD Pila:

Atributos principales:

- `pNodoPila cima`: puntero al nodo superior de la pila. Este nodo contiene un `Pasajero` y un enlace al siguiente nodo en la pila, estructurando así una pila enlazada.

Operaciones:

Operación	Parámetros	Retorno	Descripción
Pila()	-	Constructor	Inicializa una pila.
apilar(Pasajero v)	Pasajero	void	Inserta un nuevo pasajero en la cima respetando el orden de llegada.
desapilar()	-	void	Elimina el pasajero en la cima.
esVacia()	-	bool	Mira si la pila está vacía.
mostrar()	-	Pasajero*	Muestra el pasajero en la cima

mostrarTodos()	-	void	Muestra todos los pasajeros en la pila.
longitud()	-	int	Devuelve la longitud de la pila.

TAD ListaPasajeros:

Atributos principales:

- `Pasajero* listaPasajeros`: puntero a un arreglo dinámico de objetos `Pasajero`. actúa como contenedor principal.
- `int capacidad`: indica la capacidad máxima actual del arreglo.
- `int cantidad`: representa el número actual de pasajeros almacenados.
- *(Privado)* `void redimensionar(int nuevaCapacidad)`: procedimiento interno para expandir o reducir la capacidad del arreglo si es necesario.

Operaciones:

Operación	**Parámetros**	**Retorno**	**Descripción**
ListaPasajeros(int)	int	Constructor	Inicializa la lista con una capacidad de 10 pasajeros.
insertarPasajero(Pasajero&)	Pasajero	void	Inserta un nuevo pasajero en en la lista.

eliminarPasajero(int id)	int	void	Elimina el pasajero en la lista en base al id dado.
buscarPasajeroPorID (int id)	int	Pasajero*	Devuelve el pasajero especificado dependiendo del id.
buscarPasajeroPorIndice(int index)	int	Pasajero*	Devuelve el pasajero especificado dependiendo de la posición en la lista.
esVacia()	-	bool	Mira si está vacía.
contarPasajeros()	-	int	Cuenta los pasajeros.
mostrarLista()	-	void	imprime la lista entera.

TAD ABB:

Atributos principales:

- `NodoABB* raiz`: puntero al nodo raíz del árbol binario de búsqueda. Representa el punto de entrada al ABB que organiza los pasajeros por país. Desde aquí se accede al subárbol izquierdo y derecho para realizar recorridos o búsquedas.

Operaciones:

Operación	**Parámetros**	**Retorno**	**Descripción**
verInOrden(NodoABB *arb)	NodoABB*	void	Muestra el árbol en orden alfabético (inorden).
verInOrden()	-	void	Muestra el ABB completo en inorden desde la raíz.
insertar(string nombre)	String	void	Inserta un país en el ABB, manteniendo el orden.

insertar(string nombre, NodoABB *nodo)	string, Nodo ABB*	void	Inserta recursivamente un país en la posición adecuada
buscar(string pais, NodoABB *nodo)	string, Nodo ABB*	NodoABB*	Busca y retorna el nodo de un país en el ABB.
buscar(string pais)	string	NodoABB*	Busca y retorna el nodo de un país en el ABB.
mostrarPasajerosPorPais(string pais)	string	void	Muestra los pasajeros registrados para un país específico.
calcularTiempoMedioPorPais(NodoABB *nodo)	Nodo ABB*	void	Realiza el cálculo desde un nodo dado (recursivamente).

agregarPasajeroManual()	-	void	Agrega un pasajero ingresado manualmente al país correspondiente.
buscarPasajero(int id)	int	Pasajero*	Busca un pasajero por ID en todo el árbol.
buscarPasajero(int id, NodoABB *nodo)	int, NodoABB*	Pasajero*	Busca recursivamente un pasajero por ID desde un nodo.

TAD Aeropuerto:

Atributos principales:

- `ABB abbPasajeros`: árbol binario de búsqueda que almacena los pasajeros agrupados por país para estadísticas y consultas.
- `Pila pilaPasajeros`: pila de pasajeros que aún no han llegado al aeropuerto, ordenados por minuto de llegada.
- `Lista listaBoxes`: lista dinámica que contiene todos los boxes actualmente activos y en funcionamiento.

- `int minuto_actual`: variable global que lleva el control del tiempo dentro de la simulación.
- `int intervalo_llegadas`: intervalo (en minutos) entre la llegada programada de nuevos pasajeros desde la pila.

Operaciones:

Operación	**Parámetros**	**Retorno**	**Descripción**
verInOrden(NodoABB *arb)	Nodo ABB*	void	Muestra el árbol en orden alfabético (inorden).
verInOrden()	-	void	Muestra el ABB completo en inorden desde la raíz.
insertar(string nombre)	String	void	Inserta un país en el ABB, manteniendo el orden.

insertar(string nombre, NodoABB *nodo)	string, Nodo ABB*	void	Inserta recursivamente un país en la posición adecuada
buscar(string pais, NodoABB *nodo)	string, Nodo ABB*	NodoABB*	Busca y retorna el nodo de un país en el ABB.
buscar(string pais)	string	NodoABB*	Busca y retorna el nodo de un país en el ABB.
mostrarPasajerosPorPais(string pais)	string	void	Muestra los pasajeros registrados para un país específico.
calcularTiempoMedioPorPais(NodoABB *nodo)	Nodo ABB*	void	Realiza el cálculo desde un nodo dado (recursivamente).

agregarPasajeroManual()	-	void	Agrega un pasajero ingresado manualmente al país correspondiente.
buscarPasajero(int id)	int	Pasajero*	Busca un pasajero por ID en todo el árbol.
buscarPasajero(int id, NodoABB *nodo)	int, NodoABB*	Pasajero*	Busca recursivamente un pasajero por ID desde un nodo.

4- Diseño de la relación entre las clases de los TAD implementados

Respecto a la clase Pasajero esta es la unidad fundamental que contiene toda la información necesaria para simular su recorrido por el aeropuerto. Los pasajeros serán gestionados principalmente en estructuras como Pila (para llegada), Cola (en espera en cada Box), y en nodos de ABB (para estadísticas por país) y cada pasajero guarda un IDBox para indicar en qué box fue atendido, estableciendo la relación con la estructura Box donde.

Dicho eso, la clase Box utiliza el TAD Pasajero para representar tanto al pasajero que está siendo atendido como a aquellos en espera. Además, internamente maneja una Cola que ordena a los pasajeros en espera por prioridad, facilitando una atención eficiente y ordenada. Por último, forma parte de una estructura mayor, como la Lista dinámica de boxes que gestiona la cantidad y disponibilidad de boxes activos en el aeropuerto.

Por otro lado, la clase el ABB guarda en sus nodos el País correspondiente, campo por el que se ordena, y los Pasajeros que van a ese país.

por último, tenemos el aeropuerto que se encarga de implementar todas las funciones necesarias para realizar la simulación de aduanas y proporcionar las opciones especificadas en el menú, usando todas las clases del sistema.

5- Explicación de los métodos más destacados

Pasajero:

Constructor por defecto: Inicializa un pasajero con valores neutros para asegurar que los objetos empiecen en un estado conocido y válido.
Constructor con parámetros: Permite crear objetos Pasajero con todos sus datos, facilitando la inserción directa de pasajeros con sus características.
Métodos get/set: Facilitan la encapsulación de los datos, permitiendo modificar o consultar los atributos con control.
calcularTiempoEstancia(): Método útil para obtener estadísticas, pues mide el tiempo total que un pasajero permanece en el sistema desde su llegada hasta que finaliza el control.

Box:

asignarPasajero: Permite asignar a un pasajero a un box solo si está libre, configurando el tiempo restante de control para la atención.
liberarBox: Finaliza la atención del pasajero actual y, en caso de que haya pasajeros esperando, asigna automáticamente al siguiente en la cola.
actualizarTiempo: Simula el paso del tiempo en la atención, disminuyendo el tiempo restante y libera el box cuando finaliza.
agregarPasajeroAEspera: Inserta pasajeros en la cola de espera del box, respetando la prioridad definida.
mostrarInfo: Función auxiliar para depuración y monitorización del estado del box durante la simulación.

ABB:

Constructores (ABB() y variantes): Permiten crear un árbol vacío o con una raíz y subárboles ya definidos.
insertar(string nombre): Inserta un país en el ABB respetando el orden alfabético. Si el nodo ya existe, puede gestionar nuevas inserciones en su lista de pasajeros.
verInOrden(): Muestra el contenido del ABB en orden alfabético (recorrido inorden), lo cual facilita la visualización ordenada de los países y sus pasajeros.
buscar(string pais): Localiza y devuelve el nodo del árbol que contiene la lista de pasajeros del país indicado.
mostrarPasajerosPorPais(string pais): Muestra los pasajeros correspondientes a un país, accediendo a su nodo en el árbol.
calcularTiempoMedioPorPais(): Recorre el ABB y calcula el tiempo promedio de estancia para los pasajeros de cada país, útil para generar estadísticas por destino.

agregarPasajeroManual(): Permite ingresar los datos de un pasajero desde la consola, buscar o crear el nodo del país correspondiente, y agregarlo a la lista del país.

buscarPasajero(int id): Recorre todas las listas del ABB para encontrar y mostrar un pasajero por su ID, sin importar el país.

Aeropuerto:

Constructor Aeropuerto(): Inicializa todos los boxes y el árbol binario (ABB) de países. Deja listo el sistema para comenzar a recibir pasajeros, gestionarlos y atenderlos.

simularPasoTiempo(): Ejecuta una iteración de simulación del sistema. Llama a actualizarTiempo() en todos los boxes, lo que simula el avance de tiempo y permite procesar pasajeros de forma dinámica.

agregarPasajero(Pasajero p): Determina a qué box se puede asignar un pasajero según su prioridad. Si un box está libre, lo asigna directamente; si no, lo coloca en la cola de espera correspondiente. También inserta al pasajero en el ABB según su país.
mostrarEstadoBoxes(): Recorre todos los boxes del aeropuerto y muestra su estado actual, incluyendo el pasajero atendido, el tiempo restante y los pasajeros en espera. Útil para monitorear visualmente la situación.
mostrarPasajerosPorPais(string pais): Interactúa con el ABB para mostrar todos los pasajeros registrados del país especificado, permitiendo segmentar la información.
calcularTiempoMedioPorPais(): Coordina con el ABB para calcular y mostrar el tiempo medio de permanencia de los pasajeros por país. Facilita el análisis de eficiencia y comportamiento por nacionalidad.

buscarPasajero(int id): Busca un pasajero por su identificador único en todas las listas del ABB. Proporciona acceso rápido a los datos de un pasajero específico sin importar su país de origen.

agregarPasajeroManual(): Permite ingresar manualmente un nuevo pasajero desde consola, incluyendo sus datos y el país de procedencia. El sistema lo asigna adecuadamente al ABB y a un box o cola, según disponibilidad.

6- Solución adoptada: descripción de las dificultades encontradas

Durante el desarrollo del sistema de simulación del aeropuerto, se implementaron dos modos principales de control: uno por avance de minutos (`simularPasoTiempo`) y otro que ejecuta el proceso completo (`simularControlCompleto`). Sin embargo, a pesar de múltiples intentos de depuración y ajustes en la lógica, ninguna de las dos simulaciones logró funcionar correctamente.

En ambos casos, los boxes se creaban y aceptaban pasajeros en sus colas, pero nunca llegaban a atenderlos. Es decir, los boxes permanecían libres, acumulando pasajeros en espera sin que se produjera la asignación ni el avance del proceso de atención.

7- Explicación del comportamiento del programa

El programa simula el funcionamiento de un sistema de control de pasajeros en un aeropuerto, organizando su atención mediante boxes de control y gestionando sus datos a través de un árbol binario de búsqueda (ABB). Está diseñado para representar una situación realista de atención a pasajeros, priorización por urgencia, espera en colas, y generación de estadísticas útiles para la administración del aeropuerto.

1. Registro y gestión de pasajeros

Los pasajeros son introducidos al sistema manualmente por el usuario al seleccionar dicha función del menú. Cada pasajero tiene atributos como nombre, país, prioridad, hora de llegada y duración estimada del control. Esta información se encapsula en objetos de tipo Pasajero.

Tras acabar el control, el pasajero es automáticamente asociado a un país dentro de un ABB. Este árbol permite una gestión ordenada y eficiente de los pasajeros según su nacionalidad, facilitando búsquedas, reportes y estadísticas por país.

2. Asignación a boxes y atención

Los pasajeros son asignados a boxes (puestos de control) según su prioridad y disponibilidad.

Si un box está libre, el pasajero entra directamente para ser atendido pero si todos los boxes están ocupados, se coloca al pasajero en la cola de espera del box correspondiente.
Cada box tiene su propia cola y estado. Durante la simulación, el tiempo de atención disminuye paso a paso, y cuando finaliza, el box se libera automáticamente y atiende al siguiente pasajero en espera si lo hay.

Este comportamiento imita un sistema en tiempo real donde se gestiona tanto la atención como la espera, respetando las prioridades de los pasajeros.

3. Simulación del paso del tiempo

Mediante el método simularPasoTiempo(), se simula el avance del tiempo en el sistema. En cada paso:

Se actualiza el tiempo restante de atención en cada box.

Si un pasajero termina su control, el box se libera.

Automáticamente se asigna el siguiente pasajero en espera, si existe.

Esto permite ejecutar una simulación completa del funcionamiento del aeropuerto durante un periodo dado.

4. Generación de estadísticas

El ABB almacena los pasajeros agrupados por país, lo que permite ejecutar consultas complejas y generar informes, como:

-Visualización ordenada de todos los países y sus pasajeros.
-Búsqueda de pasajeros por ID independientemente de su país.
-Cálculo del tiempo medio de estancia por país, lo cual es útil para evaluar la eficiencia del control migratorio según el origen de los pasajeros.

5. Interacción con el usuario

El sistema cuenta con métodos que permiten la interacción con el usuario desde la consola, como el ingreso manual de pasajeros y la visualización del estado actual del sistema (pasajeros en boxes, en espera, etc.).

6. Encapsulamiento y modularidad

El programa está construido de forma modular y orientada a objetos, lo que permite una clara separación de responsabilidades:

Pasajero: almacena la información personal y funcional del individuo.

Box: gestiona la atención y las colas.

ABB: organiza los datos globales por país y permite búsquedas eficientes.
Aeropuerto: orquesta la lógica general del sistema, centralizando el control de boxes y el ABB.
Este diseño permite escalar el sistema fácilmente (por ejemplo, aumentando el número de boxes o integrando nuevas métricas).

8- Bibliografía y referencias utilizadas

Apuntes proporcionados en clase.