El presente trabajo práctico tiene como objetivo aplicar conceptos fundamentales sobre estructuras de datos jerárquicas, árboles balanceados y grafos, a través de la resolución de tres problemas independientes. En primer lugar, se aborda la simulación de una sala de emergencias en la que se prioriza la atención de pacientes según el nivel de riesgo, utilizando una estructura genérica que garantice orden y prioridad. En segundo lugar, se desarrolla una base de datos para almacenar y consultar registros de temperaturas en función de fechas, utilizando un árbol AVL como soporte estructural para lograr la eficiencia en operaciones de búsqueda, inserción y eliminación. Por último, se resuelve un problema de comunicación entre aldeas mediante el uso de grafos, con el fin de determinar rutas óptimas de transmisión de mensajes empleando palomas mensajeras.
En cada módulo se incluyen pruebas de validación y una aplicación principal , lo cual permite comprobar el correcto funcionamiento de los métodos implementados y facilitar la interacción con el usuario.

Proyecto 1: Sala de emergencias
Para la simulación de prioridad en una sala de emergencias, se implementó un montículo binario para construir una cola de prioridad. Esta estructura permite gestionar eficientemente a los pacientes priorizando siempre al de mayor riesgo.

Se definió la clase MonticuloBinario, que modela un heap de mínimo acorde a nuestro caso donde el paciente más urgente tiene menor valor numérico de riesgo: 1 = crítico, 2 = moderado, 3 = bajo.

La inserción de pacientes se realiza cpn complejidad logaritmica, manteniendo el orden de prioridades mediante la función infiltrar_arriba.

La extracción del paciente más prioritario se hace también en tiempo logarítmico usando infiltrar_abajo extrayendo el mínimo.

Criterio de prioridad
La prioridad se determina por el número de riesgo del paciente. Cómo segundo criterio si ambos pacientes tienen la misma prioridad nuemrica se tomó en cuenta en tiempo de llegada.

Complejidades algorítmicaSñs
Inserción → O(log n)
Eliminación → O(log n)
Buscar mínimo → O(1)

Donde n es la cantidad de elementos (pacientes) actualmente en la cola.

Proyecto 2: Temperaturas_DB

El objetivo de este proyecto es facilitar de manera eficiente las consultas de las temperaturas registradas por el científico. Para lograrlo se implementó una base de datos en memoria llamada Temperaturas_DB, que utilizará internamente un árbol AVL, esta estructura permite realizar inserciones,eliminaciones y búsquedas rápidas, incluso cuando el volumen de datos crece constantemente, además de mantener el equilibrio, garantizando operaciones logarítmicas.

Para lograr este objetivo se creó un módulo llamado Arbol_AVL y otro llamado TemperaturasDB.

Arbol_AVL consta de dos clases principales:

- NodoAVL : esta clase representa cada elemento individual,llamado nodo, dentro del árbol AVL, cada nodo contiene :
  fecha: la clave que identifica al nodo
  temperatura: el valor asociado a la clave, en este caso, la lectura de temperatura.
  altura: un entero que indica la altura del nodo dentro del árbol, utilizada para determinar el balance del árbol.
  izquierda y derecha: punteros a los nodos hijos izquierdo y derecho.
- AVLTree: esta clase contiene lógica para insertar, eliminar, buscar y recorrer nodos en el árbol AVL. El árbol AVL es una versión de árbol binario de búsqueda que mantiene su altura balanceada automáticamente, lo que garantiza que todas las operaciones se ejecuten en tiempo logarítmico O(log n).
  Métodos:
  insertar: es el núcleo de la clase. Inserta un nuevo nodo en la posición adecuada, según el orden cronológico, actualiza la altura del nodo padre, calcula el factor de balance(altura de subárbol izquierdo-altura subárbol derecho) y aplica rotaciones si el árbol está desbalanceado.
  rotar_derecha:promueve el hijo izquierdo como nuevo nodo raíz, el hijo derecho del hijo izquierdo se convierte en hijo izquierdo del nodo original, se actualizan las alturas de los nodos modificados.
  rotar_izquierda: similar a la rotación derecha pero en sentido opuesto, promueve el hijo derecho como raíz y reacomoda subárboles.
  get_altura: devuelve la altura del nodo actual, o 0 si es None, dato necesario para calcular el balance de cada nodo y decidir cuándo y cómo hacer rotaciones.
  get_balance: calcula el factor de balance del noso, si el balance está en [-1,0,1] el nodo está equilibrado, de lo contrario requiere rotaciones para corregirlo.
  buscar: implementa una búsqueda binaria, compara la fecha buscada con la del nodo actual, si son iguales devuelve el nodo, si la que se busca es menor busca el subárbol izquierdo, si es mayor busca el derecho
  listar_en_rango:realiza un recorrido en orden limitado al rango de fechas dado. Si el nodo actual está dentro del rango se agrega al resultado, si hay posibilidad de fechas menores se recorre el subárbol izquierdo, y si hay posibilidad de fechas mayores se recorre el derecho. Devuelve una lista ordenada cronológicamente de pares (fecha,temperatura) dentro del rango solicitado.

obtener_minimo: encuentra el nodo con la fecha más temprana en un subárbol dado, se mueve por los nodos hijos izquierdos hasta encontrar el nodo más a la izquierda. Es útil para la eliminación de nodos con dos hijos.

eliminar: elimina un nodo con la fecha especificada. Se navega recursivamente para encontrar el nodo, se contemplan tres casos(el nodo no tiene hijos, entonces se elimina directamente, si tiene uno solo se reemplaza por su hijo, y si tiene dos se reemplaza por el mínimo del subárbol derecho), se actualiza la altura del nodo padre, se calcula el balance y si es necesario, se aplica una rotación para restaurar el equilibrio.

TemperaturasDB consta de la clase Temperaturas_DB, la cual implementa los siguientes métodos:

Método	Complejidad Big-O	Explicación	Naturaleza
guardar_temperatura	O(log n)	Inserta una temperatura asociada a una fecha, convierte la fecha utilizando datetime.date y recorre la altura del árbol para insertar al nodo correspondiente, realizando como máximo una rotación	Inserción en árbol AVL
devolver_temperatura	O(log n)	Recupera la temperatura correspondiente a una fecha dada, devuelve None si la fecha no está en el árbol	Búsqueda en árbol AVL
listar_temperaturas_en_rango	O(k+log n)	Devuelve una lista de pares(fecha,temperatura) que se encuentren entre dos fechas dadas.	Recorrido parcial + búsqueda nodo inicial
max_temp_rango	O(k+log n)	Retorna la temperatura máxima en un intervalo de fechas.	Listar+obtener máximo
min_temp_rango	O(k+log n)	Retorna la temperatura mínima en un intervalo de fechas	Listar+obtener mínimo
temp_extremos_rango	O(k+log n)	Devuelve una tupla con la temperatura mínima y máxima del rango	Listar+obtener extremos
borrar_temperatura	O(log n)	Busca el nodo que se desea eliminar, lo elimina (si existe) y luego se balancea el árbol nuevamente	Búsqueda+eliminación de nodo
cantidad_muestras	O(1)	Devuelve la cantidad total de temperaturas almacenadas	Acceso directo

devolver_temperaturas	O(k+log n)	Devuelve una lista formateada de temperaturas entre dos fechas	Listar+formateo

Con el fin de verificar el correcto funcionamiento de los métodos de la clase Temperaturas_DB se desarrollaron pruebas, incluidas en el módulo tests. Además se implementó una aplicación principal, que proporciona una interfaz con el usuario a través de un menú de opciones, donde las opciones utilizan los métodos de la clase Temperaturas_DB.

Proyecto 3: Red de Comunicación con Palomas Mensajeras

El proyecto desarrolla una solución computacional para optimizar la transmisión de mensajes desde la aldea de "Peligros" hacia otras 21 aldeas, utilizando el modelo de las palomas mensajeras. El objetivo es encontrar una forma eficiente de comunicar una noticia desde una aldea origen a todas las demás, replicando el mensaje a través de una red mínimamente costosa. Se modela la red de aldeas como un grafo no dirigido y ponderado, donde:

- Los **nodos** representan **aldeas**.
- Las **aristas** representan **conexiones** posibles por palomas mensajeras entre aldeas vecinas.
- Los **pesos** de las aristas indican la **distancia** (en leguas).

Para encontrar la solución más eficiente, se utiliza el algoritmo de **Prim** para construir el **árbol de expansión mínima (MST)**, partiendo desde la aldea "Peligros". Este árbol garantiza que:

- Todos los nodos estén conectados.
- La suma total de distancias (leguas recorridas) sea la mínima posible.

Algoritmo Prim:

1. Se inicializa el conjunto de nodos visitados con la aldea origen.
2. Se insertan en una estructura de prioridad (montículo binario) todas las aristas que parten desde la aldea origen.
3. Mientras existan aristas en la estructura de prioridad:
   a. Se extrae la arista de menor peso.
   b. Si el nodo destino ya fue visitado, se ignora la arista.
   c. En caso contrario, se agrega la arista al MST, se marca el destino como visitado, y se insertan las nuevas aristas que parten de este nodo.

Procesamiento de datos:

- El grafo se construye leyendo el archivo **aldeas.txt.**
- Se realiza validación para descartar líneas con formato incorrecto o datos inválidos (p. ej. peso no numérico).
- El grafo resultante es simétrico (no dirigido), ya que cada conexión representa una ruta bidireccional.

Concluciones:

Esta solución garantiza un uso óptimo de recursos (palomas, tiempo y esfuerzo) mediante la minimización de distancias recorridas entre aldeas. Además, asegura que cada aldea reciba el mensaje por una única ruta, evitando redundancias y posibles conflictos en la red de comunicación.