Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

PROYECTO 2: ENTREGA DE DISEÑO

En el siguiente documento se anexa la primera entrega del proyecto (entrega de diseño). En ella, se realizará un análisis de los algoritmos para la implementación de los requerimientos funcionales y, de manera adicional, se planteará (provisionalmente) la complejidad de estos a partir de las estructuras de datos expuestas a lo largo del curso.

Este documento no es una camisa de fuerza, por el contrario, de ser necesario se realizarán modificaciones pertinentes en caso de encontrar una solución más óptima.

Identificador y Título: Requerimiento 00. Cargar los datos del archivo GeoJSON**.**

Descripción:
Se leen los datos del archivo **GeoJSON** y se crea una lista doblemente encadenada donde se guardan los comparendos. La lectura del archivo debe realizarse según el formato esperado (véase especificaciones de diseño en la guía).

Datos de Entrada:
No hay datos de entrada.

Datos de Salida:

1. **Total de comparendos** cargados en la lista.
2. Información del comparendo con el mayor **OBJECTID** (véase especificaciones de diseño en la guía).

Estructura(s) Genérica(s) de Datos a utilizar:

1. Lista doblemente encadenada.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará una lista encadenada para este requerimiento.
Las clases utilizadas son:

1. **ListaEncadenada<T extends Comparable<T>>** con T igual a la clase Comparendo.
2. Dado que ListaEncadenada utiliza nodos, también se implementará la clase **NodoLista<T>** con T igual a la clase Comparendo.
3. De igual manera la clase ListaEncadenada<T extends Comparable<T>> implementa la interfaz **IListaEncadenada< T extends Comparable<T>>** que extiende **Iterable<T>** con T igual a la clase Comparendo.

Complejidad Temporal y justificación:
La complejidad temporal es $\boldsymbol{O(n)}$, es decir, una **complejidad lineal**. Porque se requiere recorrer todos los comparendos en el archivo JSON para añadirlos a la lista doblemente encadenada.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 1A. Obtener los M comparendos con mayor gravedad.

Descripción:
Carga los comparendos a una estructura de datos tipo **Heap**, posteriormente retorna de manera ordenada según gravedad (de mayor a menor) un número M de comparendos.

Datos de Entrada:
1. M número de comparendos con **mayor gravedad**.

Datos de Salida:
1. Información de los comparendos ordenados de **mayor a menor** según gravedad.

Estructura(s) Genérica(s) de Datos a utilizar:
1. MaxHeapCP.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará un heap para este requerimiento.
Las clases utilizadas son:
1. **MaxHeapCP<T extends Comparable<T>>** con T igual a la clase Comparendo.
2. De igual manera la clase MaxHeapCP<T extends Comparable<T>> implementa la interfaz **IMaxHeapCP< T extends Comparable<T>>** con T igual a la clase Comparendo.

Se utiliza como comparador de prioridad la gravedad del comparendo. Primero se revisa el tipo de servicio, y después se revisa el código de infracción por orden lexicográfico.

Complejidad Temporal y justificación:
La complejidad temporal es $\boldsymbol{O(log_2 n)}$, es decir, una **complejidad algorítmica**. Porque la implementación del arreglo en el Heap permite una representación de la estructura como un árbol, por lo que simplemente se debe recorrer la altura del mismo para insertar y/o sacar un elemento.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 2A. Buscar los comparendos por mes y día de la semana.

Descripción: Carga los comparendos a una estructura de datos de tipo tabla de Hash Separate Chaining, y envía como llave un String con el mes y día de la semana que el usuario ingresa por parámetro. Posterior a esto retorna la información de los N comparendos que cumplan los requisitos de búsqueda (encontrarse en el rango del mes y día) sin ningún orden específico.

Datos de Entrada:

1. Número del mes (1-12).
2. Día de la semana (L, M, I, J, V, S, D).

Datos de Salida:

1. Información de los comparendos que cumplan los criterios de búsqueda por mes y día.

Estructura(s) Genérica(s) de Datos a utilizar:

1. Implementar una tabla de Hash de tipo Separate Chaining.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará una tabla de Hash de tipo Separate Chaining para este requerimiento.
Las clases utilizadas son:

1. **HashSeparateChaining<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del mes y día de la semana (String); y con los valores V igual a la clase Comparendo.
2. De igual manera la clase HashSeparateChaining<K extends Comparable<K>, V extends Comparable<V>> implementa la interfaz **IHashTable<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del mes y día de la semana (String); y con los valores V igual a la clase Comparendo.

Complejidad Temporal y justificación:
La complejidad temporal en el peor caso es $\boldsymbol{O(n)}$, es decir, una **complejidad lineal**. Sin embargo, en la mayoría de casos se presenta una complejidad de tipo $\boldsymbol{O(k)}$, es decir, una **complejidad constante**. ¿Por qué? Porque si se excede el factor de carga máximo la complejidad va a ser lineal porque se tendrá que copiar todo el arreglo en otro. Por otro lado, la complejidad es constante en la mayoría de los casos debido a que al aplicar la función Hash se obtiene directamente el índice con el cuál se puede extraer la llave y los valores.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 3A. Buscar los comparendos que tienen una fecha-hora en un rango y que son de una localidad dada.

Descripción: Carga los comparendos en una estructura de datos de tipo árbol rojo-negro, busca por un rango de fecha-hora y una localidad dada ingresada por el usuario. Posteriormente se muestra en la consola la información de los N comparendos que cumplen el criterio de búsqueda.

Datos de Entrada:

1. Rango de la fecha y hora (Límite_bajo, Límite alto) en formato "YYYY/MM/DD-HH:MM:ss".
2. Localidad.

Datos de Salida:

1. Información de los comparendos que cumplan los criterios de búsqueda por fecha-hora (rango) y localidad.

Estructura(s) Genérica(s) de Datos a utilizar:

1. Se implementará un árbol rojo-negro.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará un árbol rojo-negro para este requerimiento.
Las clases utilizadas son:

1. **RedBlackBST<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del rango (fecha y hora) y la localidad (String); y con los valores V igual a la clase Comparendo.
2. De igual manera la clase RedBlackBST<K extends Comparable<K>, V extends Comparable<V>> implementa la interfaz **IRedBlackBST<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del rango (fecha y hora) y la localidad (String); y con los valores V igual a la clase Comparendo.

El criterio de ordenamiento del árbol es en base al rango de fecha-hora.

Complejidad Temporal y justificación:
La complejidad temporal en el peor caso es $\boldsymbol{O(log_2 n)}$, es decir, una **complejidad algorítmica**. Porque se recorre la altura del árbol para realizar búsquedas o agregar.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 1B. Buscar los M comparendos más cercanos a la estación de policía.

Descripción:
Carga los comparendos a una estructura de datos tipo **Heap**, posteriormente retorna de manera ordenada según cercanía a la estación de policía (de mayor a menor) un número M de comparendos.

Datos de Entrada:
1. M número de comparendos con **mayor cercanía**.

Datos de Salida:
1. Información de los comparendos ordenados de **mayor a menor** según cercanía.

Estructura(s) Genérica(s) de Datos a utilizar:
1. MaxHeapCP.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará un heap para este requerimiento.
Las clases utilizadas son:
1. **MaxHeapCP<T extends Comparable<T>>** con T igual a la clase Comparendo.
2. De igual manera la clase MaxHeapCP<T extends Comparable<T>> implementa la interfaz **IMaxHeapCP< T extends Comparable<T>>** con T igual a la clase Comparendo.

Se utiliza como comparador de prioridad la cercanía del comparendo. Se revisa latitud y longitud.

Complejidad Temporal y justificación:
La complejidad temporal es $\boldsymbol{O(log_2 n)}$, es decir, una **complejidad algorítmica**. Porque la implementación del arreglo en el Heap permite una representación de la estructura como un árbol, por lo que simplemente se debe recorrer la altura del mismo para insertar y/o sacar un elemento.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 2B. Buscar los comparendos por medio de detección, clase de vehículo, tipo de servicio y localidad.

Descripción: Carga los comparendos a una estructura de datos de tipo tabla de Hash Separate Chaining, y envía como llave un String con el medio de detección, clase de vehículo, tipo de servicio y localidad que el usuario ingresa por parámetro. Posterior a esto retorna la información de los N comparendos que cumplan los requisitos de búsqueda ordenados por fecha (menor a mayor).

Datos de Entrada:

1. Medio de detección.
2. Clase de vehículo.
3. Tipo de servicio.
4. Localidad.

Datos de Salida:

1. Información de los comparendos que cumplan los criterios de búsqueda.

Estructura(s) Genérica(s) de Datos a utilizar:

1. Implementar una tabla de Hash de tipo Separate Chaining.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará una tabla de Hash de tipo Separate Chaining para este requerimiento.
Las clases utilizadas son:

1. **HashSeparateChaining<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del medio de detección, la clase de vehículo, tipo de servicio y localidad (String); y con los valores V igual a la clase Comparendo.
2. De igual manera la clase HashSeparateChaining<K extends Comparable<K>, V extends Comparable<V>> implementa la interfaz **IHashTable<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del medio de detección, la clase de vehículo, tipo de servicio y localidad (String); y con los valores V igual a la clase Comparendo.

Complejidad Temporal y justificación:
La complejidad temporal en el peor caso es $\boldsymbol{O(n)}$, es decir, una **complejidad lineal**. Sin embargo, en la mayoría de los casos se presenta una complejidad de tipo $\boldsymbol{O(k)}$, es decir, una **complejidad constante**. ¿Por qué? Porque si se excede el factor de carga máximo la complejidad va a ser lineal porque se tendrá que copiar todo el arreglo en otro. Por otro lado, la complejidad es constante en la mayoría de los casos debido a que al aplicar la función Hash se obtiene directamente el índice con el cuál se puede extraer la llave y los valores.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 3B. Buscar los comparendos que tienen una latitud en un rango dado y que involucraron un tipo de vehículo particular.

Descripción: Carga los comparendos en una estructura de datos de tipo árbol rojo-negro, busca por un rango de latitudes y una clase de vehículo dada ingresada por el usuario. Posteriormente se muestra en la consola la información de los N comparendos que cumplen el criterio de búsqueda.

Datos de Entrada:

1. Rango de las latitudes (Límite_bajo, Límite alto) en formato "YYYY/MM/DD-HH:MM:ss".
2. Clase de vehículo.

Datos de Salida:

1. Información de los comparendos que cumplan los criterios de búsqueda.

Estructura(s) Genérica(s) de Datos a utilizar:

1. Se implementará un árbol rojo-negro.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará un árbol rojo-negro para este requerimiento.
Las clases utilizadas son:

1. **RedBlackBST<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del rango (latitud) y la clase de vehículo (String); y con los valores V igual a la clase Comparendo.
2. De igual manera la clase RedBlackBST<K extends Comparable<K>, V extends Comparable<V>> implementa la interfaz **IRedBlackBST<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del rango (latitud) y la clase de vehículo (String); y con los valores V igual a la clase Comparendo.

El criterio de ordenamiento del árbol es en base al rango de latitud.

Complejidad Temporal y justificación:
La complejidad temporal en el peor caso es $\boldsymbol{O(log_2 n)}$, es decir, una **complejidad algorítmica**. Porque se recorre la altura del árbol para realizar búsquedas o agregar.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 1C. Visualizar Datos en una Tabla ASCII
Descripción: Carga los comparendos en una estructura de datos de tipo árbol rojo-negro, busca por un rango de días D ingresado por el usuario. Posteriormente se muestra en la consola la tabla el número de comparendos en el rango según el diseño requerido.
Datos de Entrada: 1. Rango de días (1-365).
Datos de Salida: 1. Tabla de los comparendos según el diseño expuesto en la guía.
Estructura(s) Genérica(s) de Datos a utilizar: 1. Se implementará un árbol rojo-negro.
Parametrización de Estructura(s) Genérica(s) de Datos a utilizar: Se implementará un árbol rojo-negro para este requerimiento. Las clases utilizadas son: 1. **RedBlackBST<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del rango de días (String); y con los valores V igual a la clase Comparendo. 2. De igual manera la clase RedBlackBST<K extends Comparable<K>, V extends Comparable<V>> implementa la interfaz **IRedBlackBST<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual a una unión del rango de días (String); y con los valores V igual a la clase Comparendo. El criterio de ordenamiento del árbol es en base al rango de fechas.
Complejidad Temporal y justificación: La complejidad temporal en el peor caso es $\boldsymbol{O(log_2 n)}$, es decir, una **complejidad algorítmica**. Porque se recorre la altura del árbol para realizar búsquedas o agregar.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 2C. El costo de los tiempos de espera hoy en día (cola).

Descripción: Carga los comparendos a una estructura de datos de tipo tabla de Hash Separate Chaining, y envía como llave un String con la fecha actual (según formato de diseño); además, se cargan los comparendos a otra estructura de datos de tipo Heap, es decir una cola de prioridad para manejar los comparendos que se encuentran en espera. Posterior a esto retorna la información de los N comparendos que cumplan los requisitos de búsqueda ordenados por fecha (menor a mayor).

Datos de Entrada:
No hay datos de entrada.

Datos de Salida:

1. Tabla de los comparendos según el diseño expuesto en la guía (muestra comparendos procesados y los comparendos que están en espera).

Estructura(s) Genérica(s) de Datos a utilizar:

1. Se implementará una tabla de Hash de tipo Separate Chaining.
2. Se implementará un Heap.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará una tabla de Hash de tipo Separate Chaining para este requerimiento. Además, se implementará un Heap para este requerimiento.
Las clases utilizadas son:

1. **HashSeparateChaining<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual la fecha (String); y con los valores V igual a la clase Comparendo.
2. De igual manera la clase HashSeparateChaining<K extends Comparable<K>, V extends Comparable<V>> implementa la interfaz **IHashTable<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual la fecha (String); y con los valores V igual a la clase Comparendo.
3. **MaxHeapCP<T extends Comparable<T>>** con T igual a la clase Comparendo.
4. De igual manera la clase MaxHeapCP<T extends Comparable<T>> implementa la interfaz **IMaxHeapCP< T extends Comparable<T>>** con T igual a la clase Comparendo.

Complejidad Temporal y justificación:
La complejidad temporal en el Hash Linear Probing en el peor caso es $\boldsymbol{O(n)}$, es decir, una **complejidad lineal**. Sin embargo, en la mayoría de los casos se presenta una complejidad de tipo $\boldsymbol{O(k)}$, es decir, una **complejidad constante**. ¿Por qué? Porque si se excede el factor de carga máximo la complejidad va a ser lineal porque se tendrá que copiar todo el arreglo en otro. Por otro lado, la complejidad es constante en la mayoría de los casos debido a que al aplicar la función Hash se obtiene directamente el índice con el cuál se puede extraer la llave y los valores.

Por otro lado, en el MaxHeap la complejidad temporal es $\boldsymbol{O(log_2 n)}$, es decir, una **complejidad algorítmica**. Porque la implementación del arreglo en el Heap permite una representación de la estructura como un árbol, por lo que simplemente se debe recorrer la altura del mismo para insertar y/o sacar un elemento.

Juan Andrés Ariza Gacharna (201911442)
Sergio Julian Zona Moreno (201914936)

Identificador y Título: Requerimiento 3C. El costo del tiempo de espera usando el nuevo sistema.

Descripción: Carga los comparendos a una estructura de datos de tipo tabla de Hash Separate Chaining, y envía como llave un String con la fecha actual (según formato de diseño); además, se cargan los comparendos a otra estructura de datos de tipo Heap, es decir una cola de prioridad para manejar los comparendos que se encuentran en espera. Posterior a esto retorna la información de los N comparendos que cumplan los requisitos de búsqueda ordenados por fecha (menor a mayor).

Datos de Entrada:
No hay datos de entrada.

Datos de Salida:

1. Tabla de los comparendos según el diseño expuesto en la guía (muestra comparendos procesados y los comparendos que están en espera).

Estructura(s) Genérica(s) de Datos a utilizar:

1. Se implementará una tabla de Hash de tipo Separate Chaining.
2. Se implementará un Heap.

Parametrización de Estructura(s) Genérica(s) de Datos a utilizar:
Se implementará una tabla de Hash de tipo Separate Chaining para este requerimiento. Además, se implementará un Heap para este requerimiento.
Las clases utilizadas son:

1. **HashSeparateChaining<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual la fecha (String); y con los valores V igual a la clase Comparendo.
2. De igual manera la clase HashSeparateChaining<K extends Comparable<K>, V extends Comparable<V>> implementa la interfaz **IHashTable<K extends Comparable<K>, V extends Comparable<V>>** con la llave K igual la fecha (String); y con los valores V igual a la clase Comparendo.
3. **MaxHeapCP<T extends Comparable<T>>** con T igual a la clase Comparendo.
4. De igual manera la clase MaxHeapCP<T extends Comparable<T>> implementa la interfaz **IMaxHeapCP< T extends Comparable<T>>** con T igual a la clase Comparendo.

Complejidad Temporal y justificación:
La complejidad temporal en el Hash Linear Probing en el peor caso es $\boldsymbol{O(n)}$, es decir, una **complejidad lineal**. Sin embargo, en la mayoría de los casos se presenta una complejidad de tipo $\boldsymbol{O(k)}$, es decir, una **complejidad constante**. ¿Por qué? Porque si se excede el factor de carga máximo la complejidad va a ser lineal porque se tendrá que copiar todo el arreglo en otro. Por otro lado, la complejidad es constante en la mayoría de los casos debido a que al aplicar la unción Hash se obtiene directamente el índice con el cuál se puede extraer la llave y los valores.

Por otro lado, en el MaxHeap la complejidad temporal es $\boldsymbol{O(log_2 n)}$, es decir, una **complejidad algorítmica**. Porque la implementación del arreglo en el Heap permite una representación de la estructura como un árbol, por lo que simplemente se debe recorrer la altura del mismo para insertar y/o sacar un elemento.