Universidad de los Andes

Estructura de datos

Proyecto 2: Primera entrega.

Integrante Uno: Santiago Cala Código: 201729218

Integrante Dos: Sofía Gutiérrez Código: 201612121

Documentación y Complejidad Requerimientos (Notación O)

Listado de requerimientos funcionales:

|  |  |
| --- | --- |
| Nombre | **R1 (1A) - Obtener el ranking de las N franjas horarias** |
| Resumen | Obtiene el ranking de las N franjas horarias que tengan más infracciones. Se define las franjas horarias válidas |
| Descripción | Con los datos ya cargados, se tiene una Cola de Prioridad orientada a mayor por el número de infracciones en cada franja horaria y se hace un recorrido total para realizar el conteo de infracciones por hora e ir guardándolo en la Cola de prioridad. Posteriormente, se imprimen las N primeras posiciones de la Cola de Prioridad |
| Entradas | N |
| Complejidad | Al hacer un recorrido total la operación que más se repite son los accesos y por lo tanto la complejidad es O(N). |
| Resultados | Muestra un texto indicando el intervalo que cubre la franja, el total de infracciones, el porcentaje de infracciones sin accidente, el porcentaje de infracciones con accidente y el valor total a pagar por las infracciones. |

|  |  |
| --- | --- |
| **Nombre** | **R2 (2A) - Ordenamiento y búsqueda de las infracciones por Localización Geográfica (Xcoord, Ycoord)** |
| Resumen | Realiza el ordenamiento de las infracciones en una Tabla de Hash por localización geográfica y retorna la información de la localización (Xcoord, Ycoord) dada. |
| Descripción | Como precondidción se han cargado objetos tipo VOGeographicLocation en una tabla de Hash. Para lo que se tiene que la llave de búsqueda es la tupla (Xcoord, Ycoord). Posteriormente para ordenarlos se utiliza el compareTo de dicha clase. En donde, primero se tienen en cuenta el valor Xcoord. Si hay valores de Xcoord iguales, el criterio de ordenamiento lo define el valor Ycoord. Posteriormente, se busca la Localización Geográfica dada (Xcoord, Ycoord). |
| Entradas | - Xcoord  - Ycoord |
| Complejidad | La complejidad del ordenamiento es O(Nlog(N)) ya que utiliza mergeSort, y la búsqueda al estar ordenado es O(log\_2(N)). |
| Resultados | Muestra el total de infracciones, el porcentaje de infracciones sin accidente, el porcentaje de infracciones con accidente, el valor total a pagar por las infracciones, el location, el Id de la dirección (AddressID) y el Id de la calle (StreetSegId). |

|  |  |
| --- | --- |
| Nombre | **R3 (3A) - Buscar las infracciones por rango de fechas** |
| Resumen | Consultar las infracciones por rango de fechas [Fecha Inicial (Año/Mes/Día), Fecha Final (Año/Mes/Día)]. |
| Descripción | En primer lugar, se ordenan cronológicamente de manera ascendente las infracciones. Luego, se recorre de límite inferior al superior dado agregando las infracciones en un Árbol Balanceado por Fecha (TicketIssueDate) el cual se retorna para en el view hacer la respectiva impresión de datos. |
| Entradas | Fecha Inicial (Año/Mes/Día), Fecha Final (Año/Mes/Día) |
| Complejidad | La complejidad del recorrido total es O(N)  La del ordenamiento es O(Nlog(N)) ya que se usa mergeSort y la de agregar todos los elementos es O(log\_2(N)).  En total la operación que más se repite son los accesos del recorrido total, entonces la complejidad de todo el método corresponde a O(N) |
| Resultados | Se muestra al usuario la fecha (Año/Mes/Día), el total de infracciones, el porcentaje de infracciones sin accidente, el porcentaje de infracciones con accidente y el valor total a pagar por las infracciones para cada posible fecha en el rango definido con al menos una infracción. |

-----------------------------------------------------------------------------------------------------------------------------------------

|  |  |
| --- | --- |
| Nombre | **R4 (1B) - Obtener el ranking de las N tipos de infracción (ViolationCode) que tengan más infracciones.** |
| Resumen | Deben aparecer en el ranking los tipos de infracción ordenadas de mayor a menor por número de infracciones. |
| Descripción | Para cada tipo de infracción, además toca mostrar su código, la cantidad de ocurrencias de la infracción, su porcentaje de infracciones sin accidente y con accidente y el valor total a pagar por las infracciones de ese tipo. Toca usar una cola de prioridad. |
| Entradas | * N: Cantidad de posiciones que se quieren mostrar del ranking de infracciones. |
| Complejidad | * Como vamos a usar un arreglo dinámico, la complejidad de ordenar los objetos por tipo de infracción por Merge Sort va a ser Nlog(N). * Ya después cuando se tenga el arreglo ordenado, vamos a poder recorrer la cantidad de elementos que ingresó por parámetro y se va a cumplir la condición de que son los que tienen mayor número de infracciones. Esto tiene complejidad O(n) * La complejidad total sería O(N) + O(n) [n siendo el parámetro y N siendo la cantidad de MovingViolations]. |
| Resultados | Muestra el ranking de los N tipos de infracciones con más ocurrencias. En cada una de las posiciones del ranking también toca mostrar el código, la cantidad de ocurrencias, el porcentaje de accidentalidad y el valor a pagar. |

|  |  |
| --- | --- |
| Nombre | **R5 (2B) - Realizar el ordenamiento de las infracciones por Localización Geográfica (Xcoord, Ycoord).** |
| Resumen | El método debe ordenar las infracciones por Localización Geográfica y después debe permitir poder hacer consultas sobre la estructura dado un tipo específico de localización. |
| Descripción | Para realizar el ordenamiento se agregan todos los elementos a un árbol Binario balanceado, el cual va a permitir hacer búsquedas muy rápidas y efectivas. A pesar de que el proceso de agregar y formar el árbol puede ser un poco lento, las consultas van a ser considerablemente más rápidas. |
| Entradas | * El método de buscar va a recibir una tupla (XCOORD y YCOORD) para encontrar los elementos. |
| Complejidad | * Como se había mencionado anteriormente nosotros vamos a hacer uso de un Árbol Binario balanceado. Cuando los elementos se encuentren en el árbol binario balanceado y se hagan las consultas, el algoritmo va a tener que hacer en el peor de los casos N comparaciones e ir escogiendo cada uno. * Debido a la forma en la que funciona el algoritmo la complejidad de las búsquedas va a ser simplemente log(N) (base 2) para el peor de los casos (asumiendo también que todos los nodos son dobles y no hay ninguno triple ya en el árbol balanceado) |
| Resultados | Al final de cada consulta el método lo que debería hacer retornar todas las infracciones que sucedieron en una misma ubicación geográfica. Además debe poder dar información básica sobre lo que tienen en común todas estas infracciones. |

|  |  |
| --- | --- |
| Nombre | **R6 (3B) - Buscar las franjas de fecha-hora donde se tiene un valor acumulado de infracciones en un rango dado [US$ valor inicial, US$ valor final].** |
| Resumen | Devuelve las franjas de fecha hora donde a medida que se van recorriendo las infracciones y se va obteniendo un valor acumulado de multas. Devuelve solamente las que están en un rango dado por parámetro. |
| Descripción | Para poder hacer este método, toca usar un Árbol binario balanceado que tenga como llave una franja de fecha hora que es única en todo el periodo de tiempo. Para poder hacer esto primero toca ordenar un arreglo dinámico con todas las infracciones por fechas y con este arreglo podemos crear objetos de tipo franja-hora y meterlos al árbol binario.  Ya con el árbol binario balanceado bien armado podemos hacer recurrencia sobre los objetos y hacer un método que agregue a un arreglo que entre por parámetro, todos los valores de infracciones que cumplen una condición, también dada por parámetro. |
| Entradas | * Valor de infracciones inferior * Valor de infracciones superior |
| Complejidad | * En el peor de los casos, el programa va a tener que recorrer todos los elementos del arreglo porque los parámetros incluyen a todo el arreglo. Entonces la complejidad sería O(N). También sería O(N) si en el arreglo original todas las franjas horas tienen una sola infracción. |
| Resultados | Al final el método debe mostrar una lista con todas las fechas horas que tienen un valor acumulado dentro del rango dado por parámetro. |

-----------------------------------------------------------------------------------------------------------------------------------------

|  |  |
| --- | --- |
| Nombre | **R7 (1C)- Obtener la información de una localización dada** |
| Resumen | Consulta todas infracciones con la localización dada |
| Descripción | Como precondición se han guardado todas las infracciones en un arbol balanceado con llave AddressID. Por lo tanto en dicho árbol se busca el AddressID dado por parámetro, cuyo valor es un objeto con parámetros: total de infracciones, número de infracciones sin accidente, número de infracciones con accidente, (para posteriormente calcular el porcentaje) el valor total a pagar por las infracciones, y el Id de la calle (StreetSegId). |
| Entradas | AddressID |
| Complejidad | * Como nosotros vamos a utilizar un árbol binario balanceado, la complejidad de esta estructura para realizar búsquedas es 2ln(N)=O(ln(N)). Y por esto se utiliza esta estructura. * Esta complejidad se puede calcular porque para el árbol balanceado se van recorriendo las ramas de tal manera que cada iteración se van descartando la mitad de los elementos de la estructura. Esto refleja un comportamiento logarítmico. |
| Resultados | Muestra el total de infracciones, el porcentaje de infracciones sin accidente, el porcentaje de infracciones con accidente, el valor total a pagar por las infracciones, y el Id de la calle (StreetSegId), para la dirección consultada. |

|  |  |
| --- | --- |
| Nombre | **R8 (2C)- Obtener las infracciones en un rango de horas**  **[HH:MM:SS inicial, HH:MM:SS final].** |
| Resumen | Consulta las infracciones en un rango de horas dado |
| Descripción | Ya que en estos métodos se miden tiempos y en el primer método se agrupan las infracciones por hora se agrega una línea de código a dicho método para agregarlas a un Arreglo con 23 posiciones ya que esto permitirá que agregarlos y los accesos tengan complejidad O(1). Posteriormente se recorre desde el límite inferior hasta el límite superior, y se van agregando a un BST objetos de tipo ViolationCode para después sacar la información de total de infracciones por ViolationCode. Posteriormente se imprime toda la información recopilada en el método. |
| Entradas | **-** hora inicial  **-** hora final |
| Complejidad | * Según la descripción anterior la operación que más se repite es el agregar pero al ser un BST hace que la complejidad sea O(log\_2(N)). Por esto se hace uso de un BST, ya que en cualquier otra estructura será O(N). |
| Resultados | Muestra al usuario el total de infracciones, el porcentaje de infracciones sin accidente, el porcentaje de infracciones con accidente, y el valor total a pagar por las infracciones. Y posteriormente las estadísticas por ViolationCode. |

|  |  |
| --- | --- |
| Nombre | **R9 (3C) - Obtener el ranking de las N localizaciones geográficas (Xcoord, Ycoord) con la mayor cantidad de infracciones** |
| Resumen | Obtiene el ranking de las N localizaciones geográficas (Xcoord, Ycoord) con la mayor cantidad de infracciones y las muestra al usuario |
| Descripción | Para obtener el ranking de las N localizaciones con mayor cantidad de infracciones, nosotros planeamos en usar un arreglo dinámico con objetos de tipo LocationVO cuya llave son la coordenada X y Y, el cual se ordena de acuerdo a la cantidad de infracciones y posteriormente se imprime en consola las N mayores |
| Entradas | N |
| Complejidad | * Se usa un arreglo dinámico, para que la complejidad de ordenar los objetos por localizaciones geográficas por Merge Sort sea Nlog(N). * Ya después cuando se tenga el arreglo ordenado, vamos a poder recorrer la cantidad de elementos que ingresó por parámetro y se va a cumplir la condición de que son los que tienen mayor número de infracciones. Esto tiene complejidad O(n) * La complejidad total sería O(N) + O(n) [n siendo el parámetro y N siendo la cantidad de MovingViolations]. |
| Resultados | Muestra al usuario el total de infracciones, su porcentaje de infracciones sin accidente, su porcentaje de infracciones con accidente, el location, el Id de la dirección (AddressID) y el Id de la calle (StreetSegId) |

|  |  |
| --- | --- |
| Nombre | **R10 (4C) - Generar un Grafica ASCII con la información de las infracciones por código (ViolationCode)** |
| Resumen | Crea una Grafica ASCII con la información de las infracciones por código (ViolationCode) por pocentajes |
| Descripción | En primer lugar, se ordena un arreglo dinámico por código y se va contando el número de infracciones mientras el siguiente código sea igual al actual, si cambia se guarda el resultado y reinicia el contador y así sea va recorriendo todo el arreglo. Y con el fin de reducir la complejidad se imprimen en consola “\*” de acuerdo al porcentaje dividiendo el contador antes de que se reinicie dividió el número de infracciones total. |
| Entradas | Ninguna |
| Complejidad | * Para poder generar la tabla ASCII vamos a usar un arreglo dinámico ya que al ordenar usando MergeSort se tiene una complejidad de Nlog(N). * Ya con el arreglo ordenado, podemos tener contadores que vayan agregando la cantidad de infracciones que hay. * La complejidad total va a ser de Nlog(N) para hacer la tabla ASCII |
| Resultados | Muestra al usuario una gráfica de dos columnas con el código y el porcentaje |