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 | En primer lugar se cargan los datos  Se usa una Cola de Prioridad orientada a mayor por el número de infracciones en cada franja horaria |
| Entradas | N |
| Complejidad |  |
| 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 | La llave de búsqueda es la tupla (Xcoord, Ycoord): Primero se tienen en cuenta el valor Xcoord. Si hay valores de Xcoord iguales, el criterio de ordenamiento lo define el valor Ycoord. Todas las infracciones con la llave (Xcoord, Ycoord) deben agruparse bajo un mismo objeto. |
| Entradas | - Xcoord  - Ycoord |
| Complejidad | La complejidad del ordenamiento es\* |
| 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 | Buscar las infracciones por rango de fechas [Fecha Inicial (Año/Mes/Día), Fecha Final (Año/Mes/Día)]. |
| Descripción | Representar las infracciones por fecha (Año/Mes/Día) en un Árbol Balanceado por Fecha (TicketIssueDate). Para cada posible fecha en el rango definido con al menos una infracción, se requiere informar la fecha (Año/Mes/Día), el total de infracciones, su porcentaje de infracciones sin accidente, su porcentaje de infracciones con accidente y el valor total a pagar por las infracciones. Las fechas deben informarse en orden ascendente (cronológicamente). |
| Entradas | Fecha Inicial (Año/Mes/Día), Fecha Final (Año/Mes/Día) |
| Complejidad | La complejidad de la búsqueda es  La del ordenamiento es. |
| 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 infraccion ordenadas de mayor a menor por numero 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 específio de localización. |
| Descripción | Para realizar el ordenamiento se agregan todos los elementos a un árbol Binario balanceado, el cual va a permintir 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 consdierablemente más rápidas. |
| Entradas | * El método de buscar va a recibir una tupla (XCOORD y YCOORD) para econtrar 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 poder hacer retornar todas las infracciones que sucedieron en una misma ubicación geográfica. Ademas debe poder dar informacion basica sobre lo que tienen en comun 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 acomulado de multas. Devuelve sólamente las que están en un rango dado por parámetro. |
| Descripción | Para poder hacer este método, toca usar un Arbol binario balanceado que tenga como llave una franja de fecha hora que es unica en todo el periodo de tiempo. Para poder hacer esto primero toca ordenar un arreglo dinamico con todas las infracciones por fechas y con este arreglo podemos crear objetos de tipo franja-hora y meterlos al arbol binario.  Ya con el arbol binario balanceado bien armado podemos hacer recurrencia sobre los objetos y hacer un metodo que agregue a un arreglo que entre por parametro, todos los valores de infracciones que cumplen una condicion, tambien dada por parametro. |
| 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 acomulado dentro del rango dado por parametro. |

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

|  |  |
| --- | --- |
| Nombre | **R7 (1C)- Obtener la información de una localización dada** |
| Resumen | Consulta todas infraccionescon 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). * 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 | Como precondición en el método cargar que se agreguen en un arreglo dinámico las infracciones agregadas en orden de hora. Posteriormente se recorre desde el límite inferior hasta el límite superior, y se van agregando ordenadamente por ViolationCode para despúes 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 | * Para hacer este método nosotros pensamos hacer uso de un ArregloDinamico que como precondición se encuentre ya ordenado por la fecha. Si se tiene este arreglo dinámico, para poder obtener todas las infracciones que estén en el rango es necesario recorrer el arreglo. * En el peor de los casos tocaría recorrer todo el arreglo (asumiendo que se incluyen todas las infracciones dentro del rango), pero en realidad la cantidad de comparaciones que se hacen son dependientes y proporcionales al rango. Por esto, podemos concluir que la complejidad es O(N) en el peor de los casos y c(O(N)) [c<1] en un caso promeido. * Ya después, para poder ordenar el arreglo resultante vamos a usar el algoritmo de MergeSort que va a tener una complejidad de Nlog(N). Esta complejidad entonces se va a sumar a la original * La complejidad total sería O(N) + Nlog(N) en el peor de los casos. |
| 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 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 | * Como vamos a usar un arreglo dinámico, la complejidad de ordenar los objetos por localizaciones geográficas 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 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 arrgelo dinámico por código y se va contando el número de infracciones mientras el siguiente codigo 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 diviendo el contador antes de que se reinicie dividio el número de infracciones total. |
| Entradas | Ninguna |
| Complejidad | * Para poder generar la tabla ASCII vamos a usar un arreglo dinámico y lo vamos a ordenar usando MergeSort lo cual va a tener 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 |