**Proyecto 1**

# Listado de Requerimientos

|  |  |
| --- | --- |
| Nombre | 1A – Dar cola en intervalo de tiempo |
| Resumen | Generar una Cola con todos los viajes de bicicleta que se prestaron en un periodo de tiempo dado, en orden cronológico. |
| Entradas | |
| Fecha y hora inicial  Fecha y hora final | |
| Resultados | |
| Retorna una lista de viajes que contenga, para cada viaje, el identificador del viaje y de la bicicleta y la fecha y hora inicial y final. | |

Complejidad 1A: Este método recorrerá una lista de viajes ya ordenada por fecha y hora de salida, si la fecha de salida y de llegada de cada viaje está dentro del rango que se pasa por parámetro, el viaje se agrega a la cola. El recorrido será parcial, ya que acabará cuando la fecha de salida de algún viaje sea mayor que la fecha y hora final que entra por parámetro. Como solamente se hace un recorrido, este método tendrá una complejidad de O(n), dónde n es el número de viajes que se comenzaron antes de la fecha y hora final que entra como parámetro. En el mejor de los casos, sólo se revisa el primer elemento y se retorna una cola vacía. En el peor de los casos, el método recorrerá toda la lista y añadirá todos los viajes a la cola.

|  |  |
| --- | --- |
| Nombre | **2A – Dar bicicletas ordenadas por número de viajes en intervalo de tiempo** |
| Resumen | Da una lista de bicicletas ordenadas de mayor a menor número de viajes realizados en un período de tiempo. |
| Entradas | |
| Fecha y hora inicial  Fecha y hora final | |
| Resultados | |
| Retorna una lista de bicicletas que contenga, para cada bicicleta, su identificador, el número total de viajes realizados y la distancia total recorrida por la bicicleta. | |

Complejidad 2A: Este método recorrerá una lista de viajes ya ordenada por fecha y hora de salida, si la fecha de salida y de llegada de cada viaje está dentro del rango que se pasa por parámetro, el viaje se agrega a otra lista. Luego, crear una lista de bicicletas y para cada viaje en esa anterior lista agregar la bicicleta o, si ya está agregada, actualizar la información.. Por último, ordenar esa lista de bicicletas usando un algoritmo de inserción y retornar esa lista. Es decir, habrá 3 procesos principales en éste método:

1. Recorrer la lista de viajes parcialmente y agregar los viajes entre las fechas a otra lista (O(n)) dónde n es el número de viajes que se comenzaron antes de la fecha y hora final que entra como parámetro

2. Recorrer la segunda lista de viajes e ir creando las bicicletas y agregarlas a la lista (O(k)) dónde k es el número de elementos de esa lista

3. Ordenar por quickSort, con complejidad (O(nlog(n))) en el mejor de los casos, es decir, cuando los datos están aleatorios (como en esta ocasión).

El paso que más influye en la complejidad es el ordenamiento, por lo que este algoritmo tendrá complejidad de O(nlog(n))

|  |  |
| --- | --- |
| Nombre | 3A – Dar los viajes hechos por una bicicleta en un periodo de tiempo |
| Resumen | Da una lista de viajes realizados por una bicicleta y ordenados cronológicamente por su fecha y hora inicial que se hayan hecho en un intervalo de tiempo determinado. |
| Entradas | |
| Fecha y hora inicial  Fecha y hora final  Identificador de la bicicleta | |
| Resultados | |
| Retorna una lista de viajes que contenga, para cada viaje, su identificador, y la fecha y hora final del viaje. | |

Complejidad 3A: Este método recorrerá una lista de viajes ya ordenada por fecha y hora de salida, si la fecha de salida y de llegada de cada viaje está dentro del rango que se pasa por parámetro y el identificador de la bicicleta coincide con la que llega por parámetro, el viaje se agrega a otra lista. El recorrido será parcial, ya que acabará cuando la fecha de salida de algún viaje sea mayor que la fecha y hora final que entra por parámetro. Como solamente se hace un recorrido, este método tendrá una complejidad de O(n), dónde n es el número de viajes que se comenzaron antes de la fecha y hora final que entra como parámetro. En el mejor de los casos, sólo se revisa el primer elemento y se retorna una lista vacía. En el peor de los casos, el método recorrerá toda la lista y añadirá todos los viajes a la otra lista para retornarlos.

|  |  |
| --- | --- |
| Nombre | 4A – Mostrar viajes que terminaron en una estación en un intervalo de tiempo |
| Resumen | Da una lista de viajes que hayan terminado en una estación específica, ordenados cronológicamente por su fecha y hora final que se hayan hecho en un periodo de tiempo. |
| Entradas | |
| Fecha y hora inicial  Fecha y hora final  Identificador de la estación | |
| Resultados | |
| Retorna una lista de viajes ordenada cronológicamente que contenga, para cada viaje, su identificador, el identificador de la bicicleta y la fecha y hora final. | |

Complejidad 4A: Este método recorrerá una lista de viajes ya ordenada por fecha y hora de salida, si la fecha de salida y de llegada de cada viaje está dentro del rango que se pasa por parámetro y el identificador de la estación final coincide con el que llega por parámetro, el viaje se agrega a otra lista. El recorrido será parcial, ya que acabará cuando la fecha de salida de algún viaje sea mayor que la fecha y hora final que entra por parámetro. Como solamente se hace un recorrido, este método tendrá una complejidad de O(n), dónde n es el número de viajes que se comenzaron antes de la fecha y hora final que entra como parámetro. En el mejor de los casos, sólo se revisa el primer elemento y se retorna una lista vacía. En el peor de los casos, el método recorrerá toda la lista y añadirá todos los viajes a la otra lista para retornarlos.

|  |  |
| --- | --- |
| Nombre | 1B – Dar cola de estaciones en operación dada una fecha de consulta. |
| Resumen | Da una cola con las estaciones que comenzaron su operación en una fecha determinada, ingresada por parámetro. |
| Entradas | |
| Fecha y hora inicial | |
| Resultados | |
| Retorna una cola con las estaciones que iniciaron operación después de la fecha ingresada por parámetro. | |

Complejidad 1B: Este método recorrerá una lista de estaciones, ya ordenada por fecha y hora de inicio de operación. Si la estación inició la operación después de la fecha ingresada por parámetro, la estación se agregará a la cola. El recorrido será total, pues habrá de recorrer toda la lista de estaciones. Como solamente se hace un recorrido, este método tendrá una complejidad de O(N), dónde N es el número de estaciones que iniciaron después de la fecha y hora inicial que entra como parámetro. En el mejor de los casos, sólo se revisa el primer elemento y se retorna una cola vacía. En el peor de los casos, el método recorrerá toda la lista y añadirá todos los viajes a la cola.

|  |  |
| --- | --- |
| Nombre | 2B – Mostrar bicicletas ordenadas, según distancia recorrida, en un intervalo de tiempo. |
| Resumen | Da una lista de bicicletas, en orden descendente (mayor a menor) según distancia recorrida, en un periodo de tiempo delimitado por una fecha y hora inicial y final. |
| Entradas | |
| Fecha y hora inicial  Fecha y hora final | |
| Resultados | |
| Retorna una lista de bicicletas que contenga, para cada una, su identificador, distancia recorrida y cantidad de viajes realizados. | |

Complejidad 2B: Este método recorrerá una lista de viajes ya ordenada por fecha y hora de salida, si la fecha de salida y de llegada de cada viaje está dentro del rango que se pasa por parámetro, el viaje se agrega a otra lista. Luego, crear una lista de bicicletas y para cada viaje en esa anterior lista agregar la bicicleta o, si ya está agregada, actualizar la información de la distancia recorrida. Por último, ordenar esa lista de bicicletas usando un algoritmo de inserción y retornar esa lista. Es decir, habrá 3 procesos principales en éste método:

1. Recorrer la lista de viajes parcialmente y agregar los viajes entre las fechas a otra lista (O(n)) dónde n es el número de viajes que se comenzaron antes de la fecha y hora final que entra como parámetro

2. Recorrer la segunda lista de viajes e ir creando las bicicletas y agregarlas a la lista (O(k)) dónde k es el número de elementos de esa lista

3. Ordenar por quickSort, con complejidad (O(nlog(n))) en el mejor de los casos, es decir, cuando los datos están aleatorios (como en esta ocasión).

El paso que más influye en la complejidad es el ordenamiento, por lo que este algoritmo tendrá complejidad de O(nlog(n))

|  |  |
| --- | --- |
| Nombre | 3B – Mostrar viajes realizados por género y duración. |
| Resumen | Da una lista ordenada cronológicamente, desde una fecha inicial, con los viajes que fueron realizados por un género (femenino o masculino) determinado y que hayan tenido una duración menor a la especificada. |
| Entradas | |
| Género (Femenino o Masculino).  Duración del viaje. | |
| Resultados | |
| Muestra una lista ordenada cronológicamente con el identificador del viaje, la fecha/hora inicial, la fecha/hora final y la duración de todos los viajes que cumplen con las condiciones especificadas de género y duración. | |

Complejidad 3B: Este método recorrerá una lista de viajes ya ordenada por fecha y hora de salida, si el viaje dura menos que lo especificado y se realizó por el género que entra por parámetro, se agrega a otra lista. El recorrido será total, ya que habrá de recorrer la lista ordenada por fechas de todas las bicicletas. Como solamente se hace un recorrido, este método tendrá una complejidad de O(n), dónde n es el número total de viajes realizados. En el peor de los casos, el método recorrerá toda la lista y añadirá todos los viajes a la otra lista para retornarlos.

|  |  |
| --- | --- |
| Nombre | 4B – Mostrar viajes que iniciaron en una estación en un intervalo de tiempo |
| Resumen | Da una lista de viajes que hayan iniciado en una estación específica, ordenados cronológicamente por su fecha y hora inicial que se hayan hecho en un periodo de tiempo. |
| Entradas | |
| Fecha y hora inicial  Fecha y hora final | |
| Resultados | |
| Retorna una lista ordenada cronológicamente de viajes que contenga, para cada viaje, su identificador, el identificador de la bicicleta y la fecha y hora inicial. | |

Complejidad 4B: Este método recorrerá una lista de viajes ya ordenada por fecha y hora de salida, si la fecha de salida y de llegada de cada viaje está dentro del rango que se pasa por parámetro y el identificador de la estación inicial coincide con el que llega por parámetro, el viaje se agrega a otra lista. El recorrido será parcial, ya que acabará cuando la fecha de salida de algún viaje sea mayor que la fecha y hora final que entra por parámetro. Como solamente se hace un recorrido, este método tendrá una complejidad de O(n), dónde n es el número de viajes que se comenzaron antes de la fecha y hora final que entra como parámetro. En el mejor de los casos, sólo se revisa el primer elemento y se retorna una lista vacía. En el peor de los casos, el método recorrerá toda la lista y añadirá todos los viajes a la otra lista para retornarlos.

|  |  |
| --- | --- |
| Nombre | 1C – Cargar la información |
| Resumen | Carga la información de una fuente de datos seleccionada por el usuario. |
| Entradas | |
| Archivo a cargar. | |
| Resultados | |
| Carga el archivo seleccionado por el usuario en las estructuras de datos. | |

Este método recorrerá todos los elementos del archivo a cargar y los agregará a las estructuras de datos necesarias. Por lo tanto, el algoritmo tendrá complejidad de O(n) dónde n es el número de elementos del archivo.

|  |  |
| --- | --- |
| Nombre | **2C – Dar cola de inconsistencias en viajes de una bicicleta** |
| Resumen | Guarda en una pila las consistencias en los viajes de una bicicleta, es decir, si su estación final corresponde con la siguiente estación inicial cronológicamente. Si existe una inconsistencia, ambos valores se guardan en la cola y se reinicia la pila. Al final, retorna la cola. |
| Entradas | |
| Identificación de la bicicleta  Fecha y hora inicial  Fecha y hora final | |
| Resultados | |
| Retorna una cola con pares de viajes inconsistentes. Cada inconsistencia contiene, la fecha y hora final, el nombre de la estación final, la fecha inicial del siguiente viaje y el nombre de la estación inicial del siguiente viaje. | |

Complejidad 2C: Este método recorrerá la lista de viajes y buscará el primer par de viajes que ha realizado la bicicleta con el identificador ingresado por parámetro. Se comparará la estación final del viaje 1 con la estación inicial del viaje 2. Si coinciden, se añade a la pila; si no, se agrega a la cola y se reinicia la pila. Después, el viaje 1 pasa a ser el viaje 2 y, el viaje 2, el siguiente viaje que haya realizado la bicicleta. La complejidad de este método será O(N), donde N es el número de viajes que se comenzaron antes de la fecha y hora final; por tanto, es un recorrido parcial. En el mejor de los casos, sólo se revisa el primer elemento y se retorna una cola vacía. Otro escenario positivo, sería uno en que, tras recorrer toda la lista, no haya inconsistencias y se retorne una cola vacía. En el peor de los casos, se agregan todos los elementos a la cola.

|  |  |
| --- | --- |
| Nombre | **3C – Mostrar las x bicicletas más usadas** |
| Resumen | Mostrar las X bicicletas que más hayan sido usadas según la duración total de sus viajes (ordenadas de mayor a menor duración de uso). |
| Entradas | |
| El número de bicicletas que se quieren mostrar. | |
| Resultados | |
| Retorna una lista de bicicletas que muestra, para cada bicicleta, su identificador y la duración total de sus viajes. | |

Complejidad 3C: Este método creará una lista de bicicletas y recorrerá una lista de viajes. Para cada viaje en la lista anterior, agregar la bicicleta a la lista de bicicletas o, si ya está agregada, actualizar la información. Por último, ordenar esa lista de bicicletas usando un algoritmo de inserción de acuerdo al número de viajes de cada bicicleta y retornar esa lista. Es decir, habrá 2 procesos principales en éste método:

1. Recorrer la lista de viajes e ir creando las bicicletas y agregarlas a la lista (O(k)) dónde k es el número de elementos de esa lista

3. Ordenar por quickSort, con complejidad (O(nlog(n))) en el mejor de los casos, es decir, cuando los datos están aleatorios (como en esta ocasión).

El paso que más influye en la complejidad es el ordenamiento, por lo que este algoritmo tendrá complejidad de O(nlog(n))

|  |  |
| --- | --- |
| Nombre | **4C- Mostrar viajes iniciados y terminados en una estación en un periodo de tiempo** |
| Resumen | Da una lista ordenada cronológicamente, desde una fecha inicial, con los viajes que fueron iniciados o terminados en una estación determinada. |
| Entradas | |
| Fecha y hora inicial  Fecha y hora final  Identificador de la estación. | |
| Resultados | |
| Retorna una lista ordenada cronológicamente de viajes que contenga, para cada viaje, si es de inicio o de terminación, su identificador, el identificador de la bicicleta y la fecha y hora inicial o final. | |

Complejidad 4C: Este método recorrerá una lista de viajes ya ordenada por fecha y hora de salida, si la fecha de salida y de llegada de cada viaje está dentro del rango que se pasa por parámetro y el identificador de la estación inicial o final coincide con el que llega por parámetro, el viaje se agrega a otra lista. El recorrido será parcial, ya que acabará cuando la fecha de salida de algún viaje sea mayor que la fecha y hora final que entra por parámetro. Como solamente se hace un recorrido, este método tendrá una complejidad de O(N), dónde N es el número de viajes que se comenzaron antes de la fecha y hora final que entra como parámetro. En el mejor de los casos, sólo se revisa el primer elemento y se retorna una lista vacía. En el peor de los casos, el método recorrerá toda la lista y añadirá todos los viajes a la otra lista para retornarlos.