Objetivo

El objetivo de este proyecto es poner en práctica los conceptos vistos en clase sobre heaps, tablas de hash y árboles binarios balanceados.

Contexto

Debido al crecimiento continuo de las ciudades en población, infraestructura y servicios, las organizaciones y agencias que intervienen en su administración y funcionamiento tienen como preocupación contar con información actualizada en los aspectos que afectan la vida de sus ciudadanos: educación, salud, transporte, vivienda, infraestructura, entretenimiento, seguridad, economía, entre otras. Esta información permite mantener informados a sus ciudadanos en sus actividades comunes y a los administradores y autoridades locales tomar decisiones que mejoren la calidad de vida de sus ciudadanos.

El transporte es una de las problemáticas importantes en una ciudad. En particular, el transporte público debe ofrecer soluciones eficientes para transportar un alto volumen de la población, con buena calidad, para así aumentar la productividad de la ciudad y la calidad de vida de sus ciudadanos. Además, un buen servicio de transporte público puede incentivar la reducción del transporte privado. En este aspecto, el servicio de taxis se viene transformando y mejorando con la prestación del servicio a través de plataformas tecnológicas. Entre las empresas que vienen impulsando este cambio a nivel mundial están Uber, Lyft y Hailo, entre otras.

El tema del proyecto está relacionado con el manejo de información de la prestación del servicio de taxi en la ciudad de Chicago (USA). Esta ciudad está a la vanguardia en el registro de información en diferentes aspectos de su funcionamiento (portal oficial de datos *Chicago Data Portal* https://data.cityofchicago.org). Para el análisis de su servicio de taxis utilizaremos como fuente de información, el sitio web de consulta de información disponible en https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew.

Como parte de los servicios ofrecidos por el Portal se tiene el API SODA (*Socrata Open Data* API) que permite hacer consultas sobre la información disponible usando como punto de consulta la URL https://data.cityofchicago.org/resource/wrvz-psew.json. Para información del uso del API se puede consultar la URL https://dev.socrata.com/. En este proyecto, les suministraremos la información obtenida a través del API, por lo que **no es necesario** hacer uso directo de las APIs, pero es importante entender de donde proviene la información y para qué es utilizada.

En este proyecto, vamos a construir una aplicación que le permita entender a los administradores y autoridades de la ciudad de Chicago un conjunto de consultas importantes sobre el servicio de taxis.

Las Fuentes de Datos

A continuación, se presenta una descripción de las fuentes de datos que se utilizarán en el proyecto.

1. taxi-trips-wrvz-psew-subset-small.json
2. taxi-trips-wrvz-psew-subset-medium.json
3. taxi-trips-wrvz-psew-subset-large.json

Cada fuente de datos contiene el detalle de un subconjunto de servicios de taxi (carreras) de periodos de longitud de tiempo diferentes. Cada servicio de taxi (carrera) se describe a partir de 23 datos entre los cuales se tienen: el identificador del taxi, el identificador del servicio, la empresa de afiliación del taxi, la ubicación geográfica de recogida y de terminación, la zona de la ciudad de recogida y de terminación, la fecha y la hora estimada de recogida, la fecha y hora estimada de terminación, su duración (en segundos), su distancia (millas), su costo total (US$), entre otros datos. Consulte el sitio Web https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew para conocer el detalle de la información que se incluye en cada servicio.

Carga de Información

Para responder a los requerimientos presentados más adelante, usted deberá cargar la información de todos los archivos .JSON y/o .CSV. Solo es permitido leer una vez la información de los archivos.

Requerimientos - Parte A (estudiante 1 de cada grupo)

1. Utilizar un árbol balanceado para almacenar información referente a las compañías de taxi. En el árbol por cada compañía se almacena su identificador y los IDs de sus taxis (Compañia, {Taxis}); los taxis que no registran una compañía se deben agrupar bajo el nombre de la compañía "Independent Owner". Así mismo, se requiere la construcción de una Tabla Hash para almacenar la información de los servicios realizados por cada taxi, dada la zona de inicio (pickup_community_area) de los servicios (Area, {Servicios}), los servicios de cada taxi deben estar ordenados cronológicamente por su fecha/hora de inicio.

   Tanto el Árbol Balanceado como la Tabla Hash debe crearse al momento de cargar los datos.

   Obtener el taxi que más servicios ha realizado iniciando en una zona dada (pickup_community_area) para una determinada compañía (usando el Árbol Balanceado de compañías y la Tabla de Hash de taxis); es importante resaltar que la respuesta puede mostrar más de un taxi resultante. La información a mostrar del taxi

es el(los) ID del taxi resultante y la fecha/hora de inicio de cada uno de sus servicios (en orden cronológico).

2. Agrupar los servicios por su duración en rangos de 60 segundos: el grupo 1 consiste en los servicios con duración de 1 a 60 segundos, el grupo 2 de 61 a 120 segundos, el grupo 3 de 121 a 180 segundos, y así sucesivamente. Cada grupo tiene un rango de duración único (en segundos) y sus correspondientes servicios. Definir una Tabla de Hash para representar los grupos con (Rango Duración, {Servicios}) donde el Rango Duración es el rango de tiempo en segundos de duración de sus servicios y {Servicios} es el conjunto de servicios asociado con ese rango de duración. Un grupo es válido si su conjunto de servicios No es vacío. Realizar la conformación de grupos al momento de cargar los datos en el orden en que se va leyendo el archivo de servicios.

   Obtener el conjunto {Servicios} a partir de una duración de consulta (en segundos) usando la Tabla de Hash. Con la duración de consulta se deben buscar todos los servicios que pertenezcan al grupo al que pertenezca dicha duración. Por ejemplo, si la consulta es los servicios de duración 150 segundos, se deben buscar los servicios en el grupo 3 (con rango de duración [121, 180] segundos). Por cada servicio resultante mostrar el ID del taxi, el ID del servicio y su duración en segundos.

Parte B (estudiante 2 de cada grupo)

1. Agrupar los servicios por su distancia recorrida en millas (número real con una cifra decimal). Realizar la conformación de grupos al momento de cargar los datos en el orden en que se va leyendo el archivo de servicios.

   Definir un Árbol Balanceado para representar los grupos con (Distancia, {Servicios}) donde la Distancia es la distancia recorrida en algunos servicios y {Servicios} es el conjunto de servicios asociado con esa distancia. Un grupo es válido si la distancia recorrida es mayor a 0.0 millas y su conjunto de servicios No es vacío.

   Obtener el conjunto de servicios cuya distancia recorrida está en un rango [Distancia Mínima, Distancia Máxima] usando el Árbol Binario Ordenado Balanceado.

2. Agrupar los servicios por su zona de recogida y de terminación, en una Tabla de Hash. Para un servicio con zona de recogida X y zona de terminación Y se define su identificador único "X-Y". Bajo este identificador se agrupan todos los servicios que iniciaron en la zona X y que terminaron en la zona Y. Estos servicios deben estar ordenados en un Árbol Balanceado cronológicamente por su día/hora de recogida (puede haber más de un servicio que se recoge en el mismo día/hora de inicio).

   Realizar la conformación de grupos Zona Recogida-Zona Terminación en una Tabla de Hash al momento de cargar los datos en el orden en que se va leyendo el archivo de servicios.

Resolver la consulta de obtener los servicios que iniciaron en una zona de recogida y terminan en una zona de terminación cuya hora de recogida esta entre una fecha/hora inicial y una fecha/hora final (usando la Tabla de Hash).

Parte C (trabajo en grupo)

1. Se desea ordenar los taxis registrados en el sistema, utilizando un sistema de puntos. Para calcular los puntos asignados a un taxi, se toma cada uno de los servicios prestados por dicho taxi, se suma el total de millas recorridas y el total de dinero recibido por dicho taxi en todos sus servicios y se divide el total de dinero recibido entre el total de millas recorridas, multiplicado por el total de servicios prestados (se toman en cuenta los servicios para los cuales se tiene una distancia mayor a 0.0 y se

Pasos para hacer el 2C:

-Cargar los servicios en una lista discriminando los servicios que tienen longitud y latitud 0 (ya que no son servicios validos)

-Definir la localización de referencia (promedio de lats y lons)

2. -Hallar la distancia harvesiana de cada servicio (lon y lat del servicio y lon y lat del promedio)

-hacer una hashtable con Key= de distancia harvesiana y Value = un Arbol binario de servicios que hacen parte de la distancia harvesiana)

```
pu
{
    final int R = 6371*1000; // Radious of the earth in meters
    Double latDistance = toRad(lat2-lat1);
    Double lonDistance = toRad(lon2-lon1);
    Double a = Math.sin(latDistance/2) * Math.sin(latDistance/2) + Math.cos(toRad(lat1))
            * Math.cos(toRad(lat2)) * Math.sin(lonDistance/2) *  Math.sin(lonDistance/2);
    Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    Double distance = R * c;
    Return distance;
}
```

La función *toRad(angle)* transforma un ángulo de grados a radianes.

3. Se tiene un árbol binario balanceado con los servicios registrados en el sistema, ordenados por fecha y hora (en rangos de 15 minutos, empezando por la hora exacta con 00 minutos)). Dada una fecha y hora (con un número arbitrario de minutos en el rango [0, 59]) se debe retornar todos los servicios registrados en el rango de 15 minutos más cercano, que adicionalmente hayan salido de una zona y hayan terminado en otra zona.

Restricciones

- Los datos contenidos en los archivos sólo se pueden leer una vez
- Se deberá trabajar en Java 7
- El proyecto se debe implementar en Eclipse
- La entrada/salida de información adicionales se debe realizar por consola
- **No usar las colecciones del API Java.**