Entrega 2

Integrantes

Ignacio Babbini	10475/1
Jonathan Loscalzo	12016/5

Ejercicio 1

Implemente un script de Spark que permita conocer cuántos viajes realizó cada vehículo. Recordar que un viaje es una serie de coordenadas que finalizan en un destino determinado

Script

Entrega2/01/01.py

Ejecución del script

spark-submit 01.py direccion/carpeta/trafijo.txt /direccion/carpeta/salida

Nota: no tiene que existir la carpeta de salida.

Detalle

Leemos los datos del archivo de tráfico en el RDD **lines** con el siguiente formato: `#(vehiculo, lugar)`

Luego para conocer los lugares por auto, necesitamos filtrar los lugares que son terminación de un viaje, es decir, todas los puntos donde (lugar != “”)

Luego podemos realizar de varias maneras, pero siempre es similar, agrupar por key, y luego reducir y contar cuántos elementos hay por auto. En particular, spark tiene la operación “countByKey”.

Luego guardamos los resultados en la carpeta provista como segundo argumento de ejecución del script.

Ejercicio 2

Implemente un script de Spark que permita conocer cuál es el top 3 de los "tipos" de destinos más visitados.
Los "tipos" de destino válidos son "Hospital", "Escuela", "Plaza", "Ferretería", "Farmacia","Supermercado", "Museo", NO interesa contar a los destinos "Otro".

Script

Entrega2/02/02.py

Ejecución del script

spark-submit 02.py direccion/carpeta/trafijo.txt /direccion/carpeta/salida

Nota: no tiene que existir la carpeta de salida.

Detalle

Leemos los datos del archivo de tráfico en el RDD **lines** con el siguiente formato: `#(lugar, 1)`

Luego necesitamos filtrar los lugares que son terminación de un viaje, además filtrar los lugares que no son “Otros”, es decir: `lambda a: a[0] != "" and a[0] != "Otro"`

El archivo tiene dos maneras de resolverse, pero la 2da es la más eficiente porque utiliza la mayor cantidad de procesamiento en spark.
Contabilizamos la cantidad de lugares, realizando un reduceByKey y sumarizando.
Luego con la operación takeOrdered tomamos 3 ordenando por la cantidad de ocurrencias:
En la manera 1, no se está ordenando ni tomando los tres primeros, sino que se está ejecutando en código del driver.

Luego paralelizamos los resultados y los guardamos en la carpeta provista como segundo argumento de ejecución del script. Este paso no es necesario hacerlo así podríamos directamente escribir en un archivo de salida.

Ejercicio 3

Implemente un script de Spark que permita conocer cuál es el top 10 de los destinos (coordenadas latitud, longitud) más visitados.
En esta consulta no interesa el "tipo" de destino, sólo la coordenada.

Script

Entrega2/03/03.py

Ejecución del script

spark-submit 03.py direccion/carpeta/trafijo.txt /direccion/carpeta/salida

Nota: no tiene que existir la carpeta de salida.

Detalle

Leemos los datos del archivo de tráfico en el RDD **lines** con el siguiente formato: `#((lat,log), lugar)`
Luego necesitamos filtrar los lugares que son terminación de un viaje, es decir: `lambda a: a[0] != ""`

Contabilizamos la cantidad de coordenadas, realizando un reduceByKey y sumarizando.
Luego con la operación takeOrdered tomamos 10 ordenando por la cantidad de ocurrencias.

Luego paralelizamos los resultados y los guardamos en la carpeta provista como segundo argumento de ejecución del script. Este paso no es necesario hacerlo así podríamos directamente escribir en un archivo de salida.

Ejercicio 4

Implemente un script de Spark que permita conocer cuántos vehículos estaban en movimiento en una franja horaria determinada.
El timestamp de inicio y de fin son parámetros de la consulta.

Script

Entrega2/04/04.py

Ejecución del script

spark-submit 04.py direccion/carpeta/trafijo.txt /direccion/carpeta/salida 3000 6000

Nota: no tiene que existir la carpeta de salida.

Detalle

Leemos los datos del archivo de tráfico en el RDD **lines** con el siguiente formato: `#(vehiculo, timestamp)`

Luego necesitamos filtrar por la franja de timestamp `a: ini <= a[1] <= fin`

Tenemos varias maneras de contabilizar la cantidad de vehículos en movimiento:
Manera 1: no utiliza el driver para hacer la contabilidad, sino que usa la función len de python. De las soluciones, esta no parece la más “big data” posible.

Manera 2: agrupamos los resultados por key, y luego hacemos “count” para contar los ítems. Tener en cuenta, que no es un “count by key”, sino un count de la cantidad de elementos de la RDD.

Manera 3: parece ser la manera más legible, filtramos por la franja horaria, mapeamos para quedarnos solo con los autos, hacemos un distinct para quedarnos con los que no están repetidos, y luego hacemos el “count” anterior. Habría que analizar cual de las dos operaciones es más rápida. En la documentación de spark recomiendan utilizar reduceByKey y aggregateByKey por sobre groupByKey.
Se podría agregar una cuarta propuesta con los anteriores métodos similar al aggregateByKey del ejercicio 1, y al reduceByKey del ejercicio 3.

Luego paralelizamos los resultados y los guardamos en la carpeta provista como segundo argumento de ejecución del script. Este paso no es necesario hacerlo así podríamos directamente escribir en un archivo de salida.

Ejercicio 5

Implemente un script de Spark que permita conocer cuántos vehículos estaban en movimiento en una franja horaria determinada.
El timestamp de inicio y de fin son parámetros de la consulta.

Script

Entrega2/04/04.py

Ejecución del script

spark-submit 04.py direccion/carpeta/trafijo.txt /direccion/carpeta/salida 3000

Nota: no tiene que existir la carpeta de salida.

Detalle

Leemos los datos del archivo de tráfico en el RDD **lines** con el siguiente formato: `#(vehiculo, timestamp)`

Luego, mapeamos a cada elemento de lines, a cuál intervalo pertenecen a partir de la función “get_interval”.

Removemos los repetidos, porque solo contabilizamos una tupla (vehículo, intervalo).
Luego mapeamos a #(intervalo, 1), y hacemos una reducción por key para sumarizar los resultados. Persistimos la RDD en disco en la carpeta provista como segundo parámetro

Ejercicio 6

Implemente un script de Spark que permita conocer la calle en longitud y la calle en latitud más recorrida.
La calle más recorrida es aquella por la que transitaron más vehículos en cualquiera de sus tramos (cuadras).

Script

Entrega2/06/06.py

Ejecución del script

spark-submit 06.py direccion/carpeta/trafico.txt

Detalle

El problema que surge en este ejercicio es que necesitamos asociar una coordenada con su siguiente en timestamp y vehículo. Es decir, tenemos que saber la siguiente (o anterior) esquina por la que un vehículo circuló.

Hay que tener en cuenta la parte del tiempo, para no contar combinaciones que no existen, por ejemplo:
Si el vehículo V circula por la esquina A hacía la B, en tiempo T1 y T2 respectivamente.
Luego, el vehículo V, circula por la esquina A hacia B, en tiempo Tx y Ty distintos de 1 y 2.

Si no se tuviera en cuenta el tiempo para realizar los cálculos se podrían dar combinaciones erróneas, por ejemplo:
circula V desde A hacia B en T1 a Ty -> error
circula V desde A hacia B en Tx a T2 -> error
etc…

La única manera que tenemos de asociar una Row con la anterior, es por medio de un “window” utilizando Dataframes y Pyspark.Sql.

Cargamos los datos en un Dataframe.

```
lines = sc.textFile(textfile).map(lambda a: a.split("\t")).map(lambda a: a[0:4])

# lines = lines.map(lambda a: Row(vehiculo=a[0], latitud = a[1], longitud = a[2], timestamp = a[3]))
df = sqlContext.createDataFrame(lines, ["vehiculo", "latitud", "longitud", "timestamp"])
```

Generamos una especificación de window con:

- generamos grupos por vehículos.

- ordenamos por timestamp.

con la función lag obtenemos el valor anterior de la row, sobre window.
http://spark.apache.org/docs/2.1.0/api/python/pyspark.sql.html#pyspark.sql.functions.lag

```
w = Window.partitionBy(df.vehiculo).orderBy(df.timestamp.cast("int"))

# df = df.withColumn("vehiculo_prev", f.lag(df.vehiculo).over(w))
df = df.withColumn("lat_prev", f.lag(df.latitud).over(w))
df = df.withColumn("long_prev", f.lag(df.longitud).over(w))
df = df.withColumn("timestamp_prev", f.lag(df.timestamp).over(w))
```

Es decir, si tenemos el siguiente dato:

Vehículo	Latitud	Longitud	timestamp
V	1	1	0
V	1	2	1

Y estamos analizando la row 1, nos va a devolver el valor de la row 0 (por parámetro default es la anterior) en la columna que le pidamos. De alguna manera, recibe un valor de “orden” por medio de la función over, dentro del window que le pasamos por parámetro.

Al poseer todos los valores anteriores, podemos calcular fácilmente por latitud o longitud, cuantas cuadras fueron recorridas por cada una.

En esta caso, calculamos por latitud y obtenemos Row con los datos <latitud, diff>, donde diff representa 0, 1 o None según el caso. Filtramos las que tengan valor mayor a 0.

```
por_latitud = df.select(
    df.latitud, f.abs((df.longitud - df.long_prev)).alias("diff")
).where(f.col("diff") > 0)
```

Luego hacemos la sumatoria por latitud, para obtener cuantas cuadras fueron recorridas, obtenemos el máximo y nos quedamos con la que tenga mayor cantidad:

```
max_lat = (
    por_latitud.groupBy("latitud")
    .agg(f.sum("diff").alias("suma"))
    .sort("suma", ascending=False)
    .take(1)
)
```

Ejercicio 7

Ejecute un clustering entre las coordenadas transitadas usando el algoritmo de k-means con k=5. ¿Cuáles son las coordenadas centroides resultantes? ¿Cuántos vehículos transitaron por cada cluster?

Script

Entrega2/0/07.py

Ejecución del script

spark-submit 07.py direccion/carpeta/trafijo.txt

Nota: no tiene que existir la carpeta de salida.

Detalle

El algoritmo k-means tiene 2 partes:
Asignación: a partir de los centroides, se agrupa todos los elementos de la muestra en cada uno de estos centroides según su cercanía mínima con respecto a todos ellos.
Actualización: a partir de los grupos generados, se calcula los nuevos centroides y se ejecuta nuevamente el algoritmo en caso que no haya llegado a converger.

Para la parte de la convergencia, sumamos todas las diferencias entre las distancias de los centroides viejos y nuevo, y si es menor a un valor dado por la variable “error”, frenamos el loop. Tenemos de cota 10 iteraciones.

Posterior a esto, calculamos cuántos vehículos transitaron por cluster. Para esto, generamos una RDD con ((lat,long),vehículo), y calculamos el centroide más cercano a partir del punto (lat,long) que es la clave de las tuplas. Luego obtenemos todos los vehículos distintos de #(centroide, vehículo), y contabilizamos por clave.

Mejora: Lo que se podría mejorar es que en vez de hacer todos los cálculos con centroides como una variable RDD, podríamos haber realizado un producto con las RDD que representan los puntos contra todos los centroides.