Procesamiento de datos masivos
Actividad 2 - Programación en Spark

Instalación de spark:
La instalación de spark fue realizada dentro de un notebook de Colab. De modo que si se quisiera se podría ejecutar cada uno de los puntos descritos a continuación con el notebook ubicado en (Repositorio actividad 2).

Punto 1 -)
La propuesta de solución consiste en implementar un job de spark con RDDs. Cómo se evidencia en el script la lógica usada implica el uso de un filtro de los gastos que no fueron tarjeta de crédito, y la agregación de gastos resultantes por medio del método reduceByKey. Además, para mantener a las personas que no tuvieron gastos con tarjeta de crédito inicialmente se creó un RDD con todos los nombres distintos y se añadieron al conjunto de entrada con valores de 0.

```
#!/usr/bin/python3

import sys
from pyspark.sql import SparkSession

# Inicia la aplicación
spark = SparkSession\
    .builder\
    .appName('personaGastosConTarjetaCredito')\
    .getOrCreate()

entrada = sys.argv[1]
salida = sys.argv[2]

# Lee los datos
datosEntrada = spark.sparkContext.textFile(entrada)

# Incluye personas sin gastos de tarjeta de credito
personas = datosEntrada\
    .map(lambda linea: linea.split(';')[0])\
    .distinct()\
    .map(lambda persona: f"{persona};Tarjeta de crédito;0")

datosEntrada = datosEntrada\
    .union(personas)\

# Agrega los datos
gastosConTDC = datosEntrada\
    .map(lambda linea: linea.split(';'))\
    .filter(lambda x: x[1] == 'Tarjeta de crédito')\
    .map(lambda x: (x[0], float(x[2])))\
    .reduceByKey(lambda x, y: x + y)\
    .map(lambda x: f'{x[0]};{x[1]}')

# Guardar la salida
gastosConTDC.saveAsTextFile(salida)
```

El comando de ejecución usado fue:

```
!rm -rf "actividad2/punto1/data/output/"
!$SPARK_HOME/bin/spark-submit \
    ./actividad2/punto1/src/spark-job.py \
    ./actividad2/punto1/data/input.txt \
    ./actividad2/punto1/data/output
```

Dentro de los resultados se cumple con la salida esperada en el ejercicio. Note que los resultados quedaron como flotantes, y esto es a propósito para controlar entradas con gastos flotantes.

```
24/01/15 21:57:00 INFO BlockManagerMaster: BlockManagerMaster stopped
24/01/15 21:57:00 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
24/01/15 21:57:01 INFO SparkContext: Successfully stopped SparkContext
24/01/15 21:57:01 INFO ShutdownHookManager: Shutdown hook called
24/01/15 21:57:01 INFO ShutdownHookManager: Deleting directory /tmp/spark-4f832a39-9f70-41d1-8822-09f42ebf096f
24/01/15 21:57:01 INFO ShutdownHookManager: Deleting directory /tmp/spark-3ecdb1ea-175b-44ae-921e-0f9eedbc7771
24/01/15 21:57:01 INFO ShutdownHookManager: Deleting directory /tmp/spark-3ecdb1ea-175b-44ae-921e-0f9eedbc7771/pyspark-fa96a9b9
```

```
[160]  1 !cat actividad2/punto1/data/output/part-* | sort > actividad2/punto1/data/output.txt
```

```
[161]  1 !cat actividad2/punto1/data/output.txt
```

```
Alice;250.0
Bob;201.0
Luis;0.0
```

Además, los datos resultantes particionados se unificaron en "`actividad2/punto1/data/output.txt`" para mayor legibilidad.

Punto 2 -)
La propuesta de solución consiste en tres etapas. La primera es leer todos los archivos dentro de la carpeta especificada en el input a excepción del log.txt, lo cual se realiza con wholeTextFiles y un pequeño filtro. La segunda etapa engloba el procesamiento, filtrado y agregado de vistas por categoría, más un pequeño filtro para manejar casos atípicos. Y finalmente, la tercera etapa aplica un reduce para obtener la categoría con menos vistas, además de almacenar la tupla resultante.

```python
import sys
from pyspark.sql import SparkSession

# Inicializar SparkSession
spark = SparkSession\
    .builder\
    .appName('CategoriaDeVideosMenosVista')\
    .getOrCreate()

entrada = sys.argv[1]
salida = sys.argv[2]

# Cargar los datos de entrada en un RDD, excluyendo log.txt
datosEntrada = spark.sparkContext\
    .wholeTextFiles(entrada + "/*.txt")\
    .filter(lambda x: "log.txt" not in x[0])\
    .flatMap(lambda x: x[1].split('\n'))\
    .filter(lambda x: x.strip() != '')

# Procesar los datos
categorias_visitas = datosEntrada\
    .map(lambda linea: linea.split('\t'))\
    .filter(lambda x: len(x) > 6)\
    .map(lambda campos: (campos[3], int(campos[5])))\
    .reduceByKey(lambda x, y: x + y)

# Encontrar la categoría con menos visitas
categoria_menos_vista = categorias_visitas.reduce(
    lambda a, b: a if a[1] < b[1] else b)

# Guardar o mostrar el resultado
with open(salida, 'w') as archivo_salida:
    categoria, visitas = categoria_menos_vista
    archivo_salida.write(f"{categoria};{visitas}\n")

```

Los comandos usados para su ejecución se muestran a continuación. Y para las validaciones se probó con el archivo descomprimido 0302.zip y el presentado en el documentos, y ambos salieron exitosos como también se observa a continuación.

Archivos en "actividad2/punto2/data/input/":

```
!rm -rf "actividad2/punto2/data/output/output.txt"
!$SPARK_HOME/bin/spark-submit \
    ./actividad2/punto2/src/spark-job.py \
    ./actividad2/punto2/data/input \
    ./actividad2/punto2/data/output/output.txt
```

```
24/01/15 21:57:29 INFO BlockManagerMaster: BlockManagerMaster stopped
24/01/15 21:57:29 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator sto
24/01/15 21:57:29 INFO SparkContext: Successfully stopped SparkContext
24/01/15 21:57:29 INFO ShutdownHookManager: Shutdown hook called
24/01/15 21:57:29 INFO ShutdownHookManager: Deleting directory /tmp/spark-a97c0708-d6b0-464b-8389-af3f504e8
24/01/15 21:57:29 INFO ShutdownHookManager: Deleting directory /tmp/spark-a97c0708-d6b0-464b-8389-af3f504e8
24/01/15 21:57:29 INFO ShutdownHookManager: Deleting directory /tmp/spark-48e0d641-8343-4d82-952f-56b749066
```

```
164]  1 !cat actividad2/punto2/data/output/output.txt
```

```
Sports;20
```

Archivos en 0302.zip:

```
1 !rm -rf "actividad2/punto2/data/output/0302_output.txt"
2 !$SPARK_HOME/bin/spark-submit \
3     ./actividad2/punto2/src/spark-job.py \
4     ./actividad2/punto2/data/0302 \
5     ./actividad2/punto2/data/output/0302_output.txt
```

```
24/01/15 21:57:52 INFO BlockManager: BlockManager stopped
24/01/15 21:57:52 INFO BlockManagerMaster: BlockManagerMaster stopped
24/01/15 21:57:52 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator sto
24/01/15 21:57:52 INFO SparkContext: Successfully stopped SparkContext
24/01/15 21:57:52 INFO ShutdownHookManager: Shutdown hook called
24/01/15 21:57:52 INFO ShutdownHookManager: Deleting directory /tmp/spark-ac0de0a0-7e8f-4ff3-891e-644d07f0b
24/01/15 21:57:52 INFO ShutdownHookManager: Deleting directory /tmp/spark-a4e52822-9f82-4afa-a845-ef1c929d0
24/01/15 21:57:52 INFO ShutdownHookManager: Deleting directory /tmp/spark-ac0de0a0-7e8f-4ff3-891e-644d07f0b
```

```
66]  1 !cat actividad2/punto2/data/output/0302_output.txt
```

```
Travel & Places;305409
```

Punto 3 -)

La solución implementada es muy similar al punto 1, sin embargo en esta se toman las compras que no se hicieron con tarjeta de crédito, y además se divide la salida en dos grupos, compras mayores a 1500 euros y menores o iguales a 1500 euros. Una particularidad interesante del script es que para evitar omitir clientes sin compras, usando tarjeta de crédito, fue crear un RDD con los nombres únicos de los clientes y unirlos a los RDDs de ambos grupos antes de hacer el agrupamiento por llave.

```python
import sys
from pyspark.sql import SparkSession

# Inicializar SparkSession
spark = SparkSession\
    .builder\
    .appName('personaYMetodosDePago')\
    .getOrCreate()

entrada = sys.argv[1]
salida = sys.argv[2]
salida_mayor_1500 = '/comprasSinTDCMayorDe1500'
salida_menor_igual_1500 = '/comprasSinTDCMenoroIgualDe1500'

# Lee los datos
datosEntrada = spark.sparkContext.textFile(entrada)

# Incluye personas sin gastos de tarjeta de credito
personas = datosEntrada\
    .map(lambda linea: linea.split(';')[0])\
    .distinct()\
    .map(lambda persona: (persona, 0))

# Filtrar compras con tc
compras_sin_tc = datosEntrada\
    .map(lambda linea: linea.split(';'))\
    .filter(lambda x: x[1] != 'Tarjeta de crédito')\

# Tomar compras mayores a 1500
compras_mayores_1500 = compras_sin_tc\
    .filter(lambda x: int(x[2]) > 1500)\
    .map(lambda x: (x[0], 1))\
    .union(personas)\
    .reduceByKey(lambda x, y: x+y)

# Tomar compras menores o iguales a 1500
compras_menores_iguales_1500 = compras_sin_tc\
    .filter(lambda x: int(x[2]) <= 1500)\
    .map(lambda x: (x[0], 1))\
    .union(personas)\
    .reduceByKey(lambda x, y: x+y)

# Guardar las salidas
compras_mayores_1500.saveAsTextFile(salida + salida_mayor_1500)
compras_menores_iguales_1500.saveAsTextFile(salida + salida_menor_igual_1500)

```

El comando usado para su ejecución fue:

```
!rm -rf "actividad2/punto3/data/output"
!$SPARK_HOME/bin/spark-submit \
    ./actividad2/punto3/src/spark-job.py \
    ./actividad2/punto3/data/input.txt \
    ./actividad2/punto3/data/output
```

La ejecución fue exitosa como se evidencia en la imagen a continuación. Para facilitar la visualización se concatenaron las particiones resultantes con “!cat”.

```
24/01/15 21:58:13 INFO SparkContext: Successfully stopped SparkContext
24/01/15 21:58:13 INFO ShutdownHookManager: Shutdown hook called
24/01/15 21:58:13 INFO ShutdownHookManager: Deleting directory /tmp/spark-f8d52504-cfcf-48e0-a902-43be52
24/01/15 21:58:13 INFO ShutdownHookManager: Deleting directory /tmp/spark-f8d52504-cfcf-48e0-a902-43be52
24/01/15 21:58:13 INFO ShutdownHookManager: Deleting directory /tmp/spark-54346e3d-ed44-40d0-ac52-be237c
```

```
[168]  1 !cat actividad2/punto3/data/output/comprasSinTDCMayorDe1500/part-* | sort > actividad2/punto3/data/o
```

```
[169]  1 !cat actividad2/punto3/data/output/comprasSinTDCMenoroIgualDe1500/part-* | sort > actividad2/punto3/
```

```
[170]  1 !cat actividad2/punto3/data/output/comprasSinTDCMayorDe1500_output.txt
```

```
('Alice', 0)
('Bob', 1)
```

```
[171]  1 !cat actividad2/punto3/data/output/comprasSinTDCMenoroIgualDe1500.txt
```

```
('Alice', 1)
('Bob', 0)
```