

# **Máster Universitario en Big Data y Ciencia de Datos**

## *Actividad 1 – Programación en Hadoop*

Alumno: Corral Pazos, Manuel José

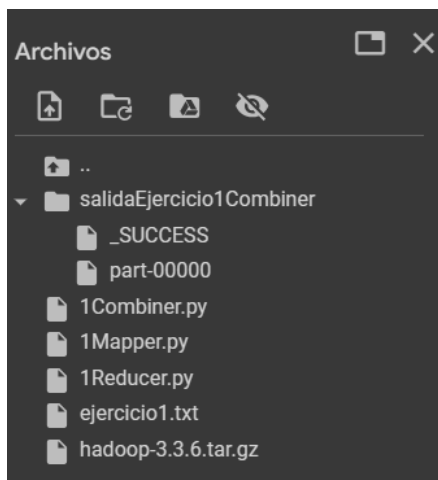
Edición Octubre 2023 a 15/01/2024

# Índice

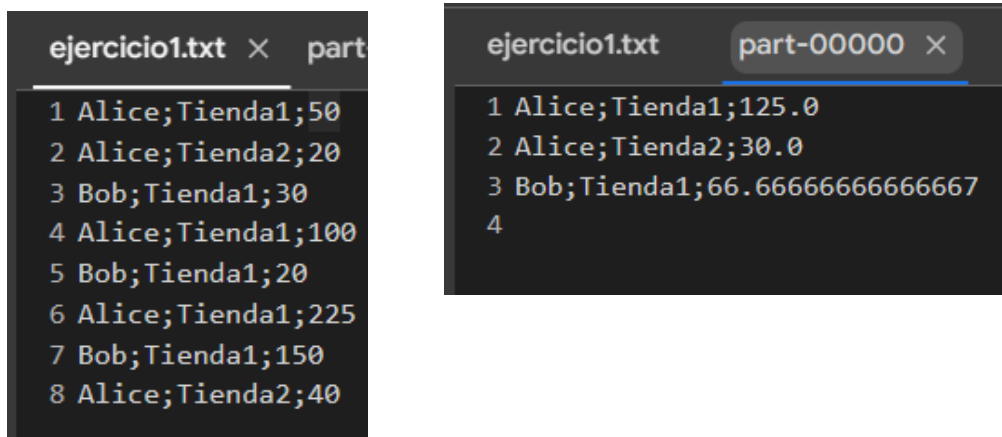
1. Ejercicio 1 .....	3
2. Ejercicio 2 .....	5
3. Ejercicio 3 .....	8

# 1. Ejercicio 1

En primer lugar, mostramos la estructura de ficheros obtenida después de la ejecución. La carpeta salidaEjercicio1Combiner contiene los ficheros resultantes de la ejecución del programa completo. Los archivos .py son el combiner, el mapper y el reducer correspondientes a este ejercicio, explicaré su funcionamiento más adelante. El fichero ejercicio1.txt es la entrada en texto del programa.



A continuación, se muestran los archivos de entrada y salida del programa ejecutado:



Ahora veamos el código del **Mapper**:

```

1Mapper.py x
1 #!/usr/bin/python3
2
3 import sys
4
5 for linea in sys.stdin:
6     linea = linea.strip()
7     persona, tienda, gasto = linea.split(";", 2)
8     print("%s;%s;%s" % (persona, tienda, gasto))
9

```

Se lee directamente del archivo de entrada del programa. Las líneas de entrada tienen formato “persona;tienda;gasto”, y por tanto hacemos un split con delimitador “;” y creamos una variable por cada campo de entrada. El mapper imprime las tres variables juntas como salida.

Con el fin de aligerar la cantidad de información que se transmite por el clúster, utilizaremos un **combiner** para calcular la suma para cada uno de los subproblemas persona-tienda que nos encontremos. Su código:

```

1Mapper.py 1Combiner.py x
1 #!/usr/bin/python3
2
3 import sys
4
5 subproblema = None
6 sumaGasto = None
7 count = None
8
9 for claveValor in sys.stdin:
10     persona, tienda, gasto = claveValor.split(";", 2)
11     gasto = float(gasto)
12     if subproblema == None:
13         subproblema = [persona, tienda]
14         sumaGasto = 0
15         count = 0
16     if subproblema == [persona, tienda]:
17         sumaGasto += gasto
18         count += 1
19     else:
20         print("%s;%s;%s;%s" % (subproblema[0], subproblema[1], sumaGasto, count))
21         subproblema = [persona, tienda]
22         sumaGasto = gasto
23         count = 1
24 print("%s;%s;%s;%s" % (subproblema[0], subproblema[1], sumaGasto, count))

```

De esta forma, el combiner toma como entrada la salida del mapper, e imprime líneas en formato “persona;tienda;sumaGasto;count”. El combiner va sumando las cantidades de gasto de cada uno de los subproblemas persona-tienda en una misma variable llamada sumaGasto, y cada vez que realice una suma va aumentando en 1 el

valor de count. De esta forma, reducimos la información que se enviará al reducer, para que éste sólo tenga que calcular la media.

Veamos el código del **Reducer**:

```
1Mapper.py 1Combiner.py 1Reducer.py X
1 #!/usr/bin/python3
2
3 import sys
4
5 subproblema = None
6 gastoMedio = None
7 contador = None
8
9 for claveValor in sys.stdin:
10
11     persona, tienda, sumaGasto, count = claveValor.split(";", 3)
12     sumaGasto = float(sumaGasto)
13     count = float(count)
14     if subproblema == None:
15         subproblema = [persona, tienda]
16         contador = 0
17         gastoMedio = 0
18
19     if subproblema == [persona, tienda]:
20         contador += count;
21         gastoMedio = sumaGasto/contador
22
23     else:
24         print("%s;%s;%s" % (subproblema[0], subproblema[1], gastoMedio))
25         subproblema = [persona, tienda]
26         contador = count
27         gastoMedio = sumaGasto / contador
28
29 print("%s;%s;%s" % (subproblema[0], subproblema[1], gastoMedio))
30
```

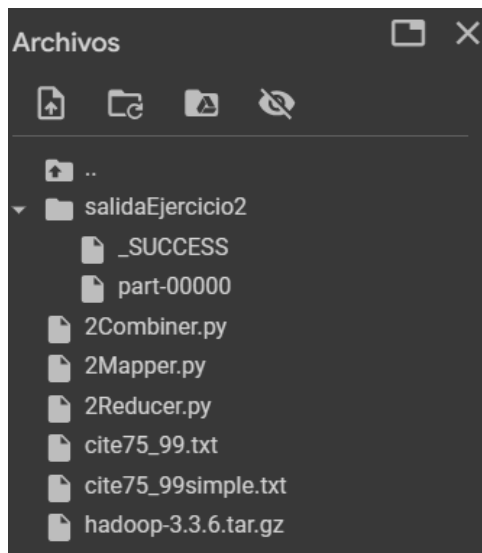
Como ya dijimos anteriormente, el reducer toma como entrada la salida del combiner, para calcular la media a partir de los valores que le entran, dividiendo la sumaGasto entre el contador (veces que se realizó una suma).

El comando para la compilación es el siguiente:

```
!hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -files
./1Mapper.py,./1Reducer.py,./1Combiner.py -mapper ./1Mapper.py -reducer
./1Reducer.py -combiner ./1Combiner.py -input ejercicio1.txt -output
./salidaEjercicio1Combiner
```

## 2. Ejercicio 2

La estructura de ficheros es similar a la del ejercicio anterior.



Para realizar pruebas, utilicé cite75\_99simple.txt, archivo con las primeras líneas del archivo original. A continuación, se muestran algunas líneas de la salida programa ejecutado

```
part-00000
1 100000 5031388,
2
3 1000006 4714284,
4
5 1000007 4766693,
6
7 1000011 5033339,
8
9 1000017 3908629,
10
11 1000026 4043055,
12
13 1000033 4975983,
14
15 1000043 4091523,
16
17 1000044 4055371,4082383,
18
19 1000045 4290571,
20
21 1000046 5525001,
22
23 1000049 5996916,
24
25 1000051 4541310,
26
27 10000 4539112,
28
29 1000054 4946631,
30
31 1000065 4748968,
32
33 1000067 4944640,5071294,
34
35 1000070 4928425,5009029,
36
37 1000073 5474494,
38
39 1000076 5845593,
```

Veamos el código del **Mapper**:

```

2Combiner.py  2Reducer.py  2Mapper.py x
1 #!/usr/bin/python3
2
3 import sys
4
5 is_first_line = True
6
7 for linea in sys.stdin:
8     linea = linea.strip()
9
10    if is_first_line:
11        is_first_line = False
12        continue
13
14    citante, citada = linea.split(",", 1)
15    print("%s;%s" % (citada, citante))
16

```

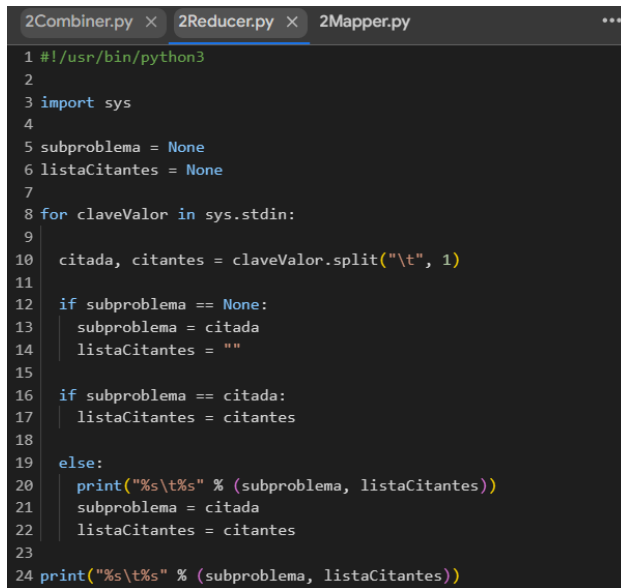
En este caso, creamos una variable booleana para no leer la primera línea. Tras esto, el mapper lee cada una de las líneas del archivo e invierte citante-citada para que entre en el orden deseado al **combiner**. Veamos el código de éste:

```

2Combiner.py x  2Reducer.py  2Mapper.py  ...
1 #!/usr/bin/python3
2
3 import sys
4
5 subproblema = None
6 listaCitantes = None
7
8 for claveValor in sys.stdin:
9     citada, citante = claveValor.split(";", 1)
10
11    if subproblema == None:
12        subproblema = citada
13        listaCitantes = ""
14
15    if subproblema == citada:
16        citante = citante.strip()
17        listaCitantes += citante + ","
18
19    else:
20        print("%s\t%s" % (subproblema, listaCitantes))
21        subproblema = citada
22        citante = citante.strip()
23        listaCitantes = citante + ","
24
25 print("%s\t%s" % (subproblema, listaCitantes))

```

En este caso, el subproblema sólo es la variable citada, y en la variable citante se irán almacenando los diferentes valores correspondientes a la clave "citada". De esta forma, la salida del combiner tiene la estructura "citada\tcitante1,citante2,citante3,...". Ésta será la entrada del **reducer**:



```

1 #!/usr/bin/python3
2
3 import sys
4
5 subproblema = None
6 listaCitantes = None
7
8 for claveValor in sys.stdin:
9
10     citada, citantes = claveValor.split("\t", 1)
11
12     if subproblema == None:
13         subproblema = citada
14         listaCitantes = ""
15
16     if subproblema == citada:
17         listaCitantes = citantes
18
19     else:
20         print("%s\t%s" % (subproblema, listaCitantes))
21         subproblema = citada
22         listaCitantes = citantes
23
24 print("%s\t%s" % (subproblema, listaCitantes))

```

El reducer recoge la salida del combiner y la ordena. En este caso, no es necesario ordenarla, pues en el dataset de origen las patentes están ordenadas numéricamente por el campo “citante”. Por lo tanto, la lista (también de citantes) que obtenemos en el combiner, ya está ordenada siempre.

Para compilar el ejercicio, utilizamos el comando:

```

!hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -files
./2Mapper.py,./2Reducer.py,./2Combiner.py -mapper ./2Mapper.py -reducer
./2Reducer.py -combiner ./2Combiner.py -input cite75_99.txt -output ./salidaEjercicio2

```

### 3. Ejercicio 3

El ejercicio 3 está resuelto en el archivo *PersonasQueCompranEnMuchasTiendas.pdf*, tal y como se indica en el guion de la actividad.