Ejercicios Práctica 1 de Computación de Altas Prestaciones

Motivación de realizar este documento

Aunque hayamos añadido también el documento *Tutorial_Spark_P1_CAP.ipynb* a la entrega de la práctica, encontramos que poder leer los resultados de los ejercicios en el propio documento es complicado al tener tanto código que no se utiliza per se en los ejercicios. Por eso hemos decidido realizar este documento PDF que contiene unicamente los ejercicios y las respuestas, así como el resultado en los casos pertinentes.

Ejercicio 1

Explica el uso y propósito de las operaciones anteriores.

Comenta también sobre el tamaño del RDD resultante en relación con el tamaño del RDD original.
Por ejemplo, si el RDD original tiene tamaño N, entonces `rdd.filter()` tendrá tamaño $K \leq N$.

Respuesta:

Para comentar las operaciones anteriores, hemos puesto un comentario encima de cada línea, explicando lo que debe suceder tras la ejecución de las operaciones concretas, más allá de unicamente las transformaciones.

In [8]:
```
# nos devuelve un nuevo conjunto de datos con la longitud de cada línea.
charsPerLine = quijote.map(lambda s: len(s))

# nos devuelve un nuevo conjunto de datos conformado por las palabras del
allWords = quijote.flatMap(lambda s: s.split())

# parte del dataset \textit{allWords} y primero pasa las palabras a minús
allWordsNoArticles = allWords.filter(lambda a: a.lower() not in ["el", "l
```

```
# utiliza también el dataset allWords. En este caso, pone las palabras en
allWordsUnique = allWords.map(lambda s: s.lower()).distinct()

# realiza  una transformación sample sobre el dataset allWords con una fr
sampleWords = allWords.sample(withReplacement=True, fraction=0.2, seed=66

# realiza una unión entre el dataset obtenido antes con el sample y con u
# frecuencia 0.3 y sin reemplazamiento.
weirdSampling = sampleWords.union(allWordsNoArticles.sample(False, fracti
```

Además, comentamos ahora más en cooncreto sobre las transformaciones que nos especifica el enunciado:

- `map` : devuelve un nuevo conjunto de datos distribuido formado al pasar cada elemento de la fuente a través de una función *func*. Con respecto al tamaño resultante, `map` acaba con el tamaño original N.
- `flatmap` : función similar a `map` , pero cada elemento de entrada se puede asignar a 0 o más elementos de salida. Con respecto al tamaño resultante, `flatmap` puede acabar con cualquier tamaño, ya sea $K\leq N$ ó $K\geq N$.
- `filter` : devuelve un nuevo conjunto de datos formado al seleccionar aquellos elementos del conjunto de datos original en los que *func* devuelve `True` . En este caso, el tamaño del dataset de salida debe ser $K\leq N$.
- `distinct` : devuelve un nuevo conjunto de datos que contiene los distintos elementos del conjunto de datos original. La función `distinct` tiene como salida un conjunto de datos de tamaño $K\leq N$.
- `sample` : esta función muestrea una fracción de los datos, con o sin reemplazamiento, utilizando una semilla generadora de números aleatorios dada. En cuanto al tamaño, `sample` puede ser variable. El tamaño esperado será $N\times fraction$, donde $fraction$ es el argumento pasado a la función.
- `union` : devuelve un nuevo dataset que contiene la unión de los elementos en el dataset original y en el pasado por argumento. Por lo tanto, el tamaño del nuevo conjunto de datos será $N+M$, siendo M el tamaño del conjunto de datos pasado por argumento.

Ejercicio 2

Explica el uso y propósito de las transformaciones anteriores.

¿Cómo contarías los elementos de un RDD utilizando únicamente `map` y/o `reduce` ?

Respuesta:

Tal y como hemos hecho en el ejercicio anterior, comentamos el funcionamiento de cada transformación/acción comentada delante de cada fragmento de código para

facilitar la lectura del ejercicio.

In [ ]:
```
# Nos cuenta el número de lineas del Quijote.
# Como el dato principal de nuestro RDD son líneas, el count nos devuelve
numLines = quijote.count()

# Hace un reduce y obtiene el número de caracteres en el Quijote. Para el
# en el dataset de caracteres por línea.
numChars = charsPerLine.reduce(lambda a,b: a+b) # also charsPerLine.sum()

# Nos muestra las 10 palabras más largas del dataset allWordsNoArticles.
sortedWordsByLength = allWordsNoArticles.takeOrdered(10, key=lambda x: -l
```

Por otro lado, para contar los elementos de un RDD utilizando únicamente `map` y/o `reduce` lo hacemos de la siguiente manera:

In [ ]:
```
# Utilizando map y reduce
count = rdd.map(lambda a: 1).reduce(lambda a,b: a+b)

# Utilizando solo reduce
count = rdd.reduce(lambda a,b: 1+1)
```

Ejercicio 3

Explica el uso y propósito de las funciones usadas con RDDs clave-valor.

¿Cuáles son acciones y cuáles transformaciones?

¿Qué hace la celda anterior?

Respuesta:

Como hemos hecho en los ejercicios anteriores, comentaremos el propósito y uso de las funciones justo encima de ellas como comentario para facilitar la lectura del ejercicio.

In [ ]:
```
# Obtenemos un nuevo conjunto de datos donde quitamos los caracteres espe
# en las palabras pasadas primero a minúsculas. Después de sustituirlo, v
# y más tarde filtra para quedarse únicamente con las palabras que tienen
allWords = allWords.flatMap(lambda w: re.sub("""[;|:|\.|,|-|—|"|'|\s""","

# Creamos un nuevo RDD con el Quijote II.
allWords2 = sc.parallelize(requests.get("https://gist.githubusercontent.c

# Realizamos exactamente lo mismo que hemos realizado en la línea 1 para
allWords2 = allWords2.flatMap(lambda w: re.sub("""[;|:|\.|,|-|—|"|'|\s""",
```

In [22]:
```
# En las dos siguientes líneas creamos un RDD donde sustituimos un elemen
words = allWords.map(lambda e: (e,1))
words2 = allWords2.map(lambda e: (e,1))
```

In [ ]:
```
# Hace la operación reduceByKey, que nos permite realizar una función sob
# En este caso, suma el número de veces que cierta palabra, la clave, apa
frequencies = words.reduceByKey(lambda a,b: a+b)
frequencies2 = words2.reduceByKey(lambda a,b: a+b)
```

In [18]:
```
# Juntamos las frecuencias de ambos libros Quijote I y Quijote II. El nue
joinFreq = frequencies.join(frequencies2)
```

In [ ]:
```
# Calcula el ratio, o diferencia relativa, entre los porcentajes del prim
# 10 palabras con mayor ratio, mientras que la segunda, obtiene las 10 pa
result = joinFreq.map(lambda e: (e[0], (e[1][0] - e[1][1])/(e[1][0] + e[1
freq_quijote1 = result.takeOrdered(10, lambda v: -v[1])
freq_quijote2 = result.takeOrdered(10, lambda v: +v[1])
```

Ahora, especificamos cuales de ellas son acciones y cuales transformaciones:

- Acciones: `takeOrdered`
- Transformaciones: `map`, `flatmap`, `filter`, `join`, `reduceByKey`, `groupByKey`

Ejercicio 4

Obten los usuarios que han subido a reddit más de 10 posts de más de 100 carácteres.

Respuesta:

In [63]:
```
# Obtenemos primero aquellas entradas en las que la longitud del reddit t
# Despues, agrupamos por autor y contamos aquellos que tienen mas de 10 e
resultado = df.where("length > 100").groupBy("author").count().where("cou
resultado.toPandas()
```

Out[63]:

	author	count
0	boardgamerecommender	12
1	dota2matchbot	230
2	[deleted]	529
3	JmodTracker	11
4	lolretkj	12
5	gyfaglover4	242
6	AutoModerator	616
7	Jesupekka	21
8	autotldr	221
9	Chabombs	13
10	removalbot	305
11	unityrufa	11
12	ShitSeattleSays	11
13	ccmanga	57
14	MartyMcfly6	15
15	james4398	15
16	asamale	12
17	victorantos2	12
18	Kmoore75801	54
19	scamcop	18
20	fiplefip	193
21	NHLStreamsBot	34
22	albedrio	12
23	Ulimit200	24
24	FFBot	14
25	soneo143	14
26	cruyff8	83
27	leaopeng	102
28	stephini	23
29	juriah121	102
30	skdgldgajdgkj	11
