Tarea Semana 14

¿Cómo funciona RRD map()?

Cuando usas map(), Spark recorre cada elemento del RDD original, aplica la función definida a ese elemento y almacena el resultado en un nuevo RDD. Es importante tener en cuenta que map() es una transformación perezosa (lazy), lo que significa que Spark no ejecutará la operación inmediatamente. La ejecución de la transformación ocurre solo cuando se llama a una acción que obliga a evaluar el RDD.

Explicación de cada paso

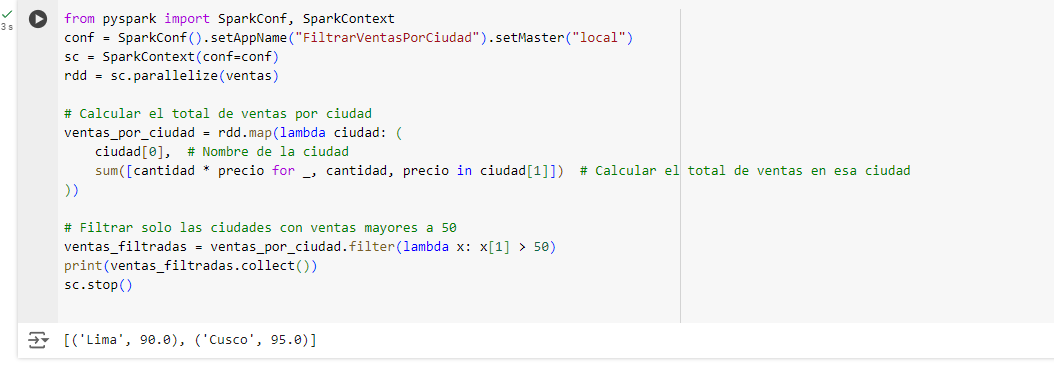
1. Configuración de SparkContext: Creamos sc usando SparkConf y SparkContext, especificando el modo de ejecución (local).
2. Cálculo de ventas por ciudad: Creamos el RDD, aplicamos map y luego reduceByKey para obtener el total de ventas por ciudad.
3. Detención de SparkContext: Es una buena práctica detener el contexto de Spark al final del proceso con sc.stop().

A screenshot of a computer code

Description automatically generated

Ejemplo de uso de filter()

En PySpark, el método filter() se utiliza para filtrar elementos en un RDD de acuerdo con una condición especificada. Cada elemento del RDD se evalúa, y solo los que cumplen con la condición permanecen en el RDD resultante. Esto es útil para reducir un conjunto de datos a solo los elementos



Explicación del resultado

Solo se incluyen las ciudades que tienen un total de ventas mayor a 50. En este caso:

* Lima tiene un total de ventas de 80.0, que es mayor que 50.
* Cusco tiene un total de ventas de 95.0, también mayor que 50.
* Arequipa no aparece en los resultados porque su total de ventas (35.0) es menor que 50.

Ejemplo de uso de flatMap()

En PySpark, flatMap() es una transformación que permite aplicar una función a cada elemento del RDD y luego "aplanar" los resultados. A diferencia de map(), que produce un valor por cada elemento de entrada, flatMap() permite que cada elemento de entrada produzca cero, uno o más elementos de salida. Esto es útil cuando queremos dividir un elemento en varias partes y tratarlas por separado. Supongamos que tenemos datos de ventas en distintas ciudades, y cada entrada contiene una ciudad junto con una lista de productos vendidos, con su cantidad y precio por unidad. Queremos obtener una lista de todas las ventas individuales por producto, sin agruparlas por ciudad.

A computer screen shot of a computer

Description automatically generated

Explicación del código

1. flatMap(): Usamos flatMap() para transformar cada elemento en una lista de ventas individuales.
   * La función lambda toma cada ciudad y produce una lista de tuplas.
   * Cada tupla contiene: el nombre de la ciudad, el nombre del producto, la cantidad vendida y el total de la venta (cantidad \* precio).
2. collect(): Ejecuta el cálculo y devuelve el resultado completo en una lista.

Cada venta se convierte en una entrada individual. Por ejemplo:

* ('Lima', 'Producto1', 5, 50.0) indica que en Lima se vendieron 5 unidades de Producto1 por un total de 50.0.
* ('Cusco', 'Producto3', 1, 15.0) indica que en Cusco se vendió 1 unidad de Producto3 por 15.0.

Ejemplo de uso de union()

En PySpark, el método union() se utiliza para combinar dos RDDs en uno solo, creando un nuevo RDD que contiene los elementos de ambos RDDs de entrada. Es importante destacar que union() no elimina los elementos duplicados; simplemente concatena ambos conjuntos. Esto puede ser útil cuando necesitas combinar diferentes conjuntos de datos que tienen una estructura similar. Supongamos que tienes dos RDDs que contienen información sobre estudiantes de dos colegios distintos, y cada entrada en los RDDs tiene el nombre del estudiante, el colegio al que pertenece y su calificación promedio.

A white screen with text

Description automatically generated with medium confidence

Explicación del resultado

El RDD resultante rdd\_estudiantes\_total contiene los registros de todos los estudiantes de ambos colegios. En este caso:

* Los estudiantes de "Colegio A" y "Colegio B" se encuentran ahora en un solo RDD, facilitando su análisis conjunto.

Ejemplo de uso de intersection()

En PySpark, el método intersection() se utiliza para encontrar los elementos comunes entre dos RDDs. Este método devuelve un nuevo RDD que contiene solo los elementos que están presentes en ambos RDDs, eliminando cualquier duplicado en el proceso. Tienes dos RDDs con datos de ventas de productos, y deseas encontrar los productos que fueron vendidos en ambas tiendas. Los RDDs pueden contener registros de productos vendidos en la tienda A y en la tienda B, y tú quieres saber qué productos fueron vendidos en ambas.

A screenshot of a computer program

Description automatically generated

Explicación del código:

1. Crear los RDDs: Usamos parallelize() para crear dos RDDs, rdd\_vendidos y rdd\_solicitados, con la información de los productos vendidos y solicitados.
2. intersection(): Usamos intersection() para encontrar los productos comunes entre ambos RDDs. El resultado es un RDD que contiene solo aquellos productos que aparecen tanto en los productos vendidos como en los solicitados.

El RDD resultante contiene solo los productos que aparecen en ambos conjuntos de datos:

* ProductoB y ProductoD fueron tanto vendidos como solicitados, por lo que estos son los elementos comunes entre los dos RDDs.

Propósito de distinct()

El método distinct() en PySpark se utiliza para eliminar elementos duplicados de un RDD, devolviendo un nuevo RDD que contiene solo los elementos únicos del RDD original.

* Función principal: Elimina las duplicaciones en los datos. Es útil cuando quieres obtener los valores únicos de un conjunto de datos.
* Uso común: Cuando se tiene un conjunto de datos con elementos repetidos y solo necesitas una lista de elementos únicos.

A screen shot of a computer

Description automatically generated

Explicación del código:

1. Crear el RDD: Usamos parallelize() para crear un RDD a partir de la lista ventas, que contiene algunos productos y cantidades duplicados.
2. distinct(): Aplicamos el método distinct() para eliminar las combinaciones duplicadas de productos y cantidades. Este método devuelve un nuevo RDD con solo los elementos únicos.

El RDD resultante contiene solo las combinaciones únicas de productos y cantidades, eliminando las duplicaciones que estaban presentes en los datos originales.

Propósito de groupByKey()

El método groupByKey() en PySpark se utiliza para agrupar los elementos de un RDD que son pares clave-valor. Específicamente, agrupa todos los valores asociados con una misma clave, creando un nuevo RDD donde la clave está asociada con una lista de valores.

* Función principal: Agrupa los valores por clave.
* Uso común: Este método es útil cuando quieres agrupar los valores asociados con cada clave en un conjunto de datos y realizar algún tipo de operación en esos grupos.

A screenshot of a computer program

Description automatically generated

Explicación del código:

1. Crear el RDD: Usamos parallelize() para crear un RDD a partir de la lista ventas, que contiene pares clave-valor donde la clave es la ciudad y el valor es el número de ventas.
2. groupByKey(): Aplicamos groupByKey() para agrupar las ventas por ciudad. Este método devuelve un RDD de tipo [(clave, [valores])], donde cada clave (ciudad) está asociada con una lista de valores (ventas).

Propósito de reduceByKey()

El método reduceByKey() en PySpark se utiliza para combinar los valores de un RDD de tipo clave-valor de acuerdo con una función de reducción, operando de forma eficiente en paralelo. Es especialmente útil cuando quieres realizar agregaciones o combinaciones de valores para una misma clave.

Propósito de reduceByKey()

* Función principal: Combina los valores asociados a una clave utilizando una función de reducción (por ejemplo, suma, promedio, máximo, etc.).
* Uso común: Cuando quieres realizar una operación agregada (como sumar, contar, etc.) sobre los valores agrupados por clave.

A screenshot of a computer program

Description automatically generated

Explicación del código:

1. Crear el RDD: Usamos parallelize() para crear un RDD a partir de la lista ventas, que contiene pares clave-valor donde la clave es la ciudad y el valor es el número de ventas.
2. reduceByKey(): Aplicamos reduceByKey() con una función de reducción que suma dos valores (lambda x, y: x + y). Esto combina las ventas de la misma ciudad, sumando todas las ventas asociadas con la misma clave (ciudad).

Propósito de sortByKey()

El método sortByKey() en PySpark se utiliza para ordenar un RDD de pares clave-valor según las claves. Este método devuelve un nuevo RDD donde los elementos están ordenados en función de las claves, de forma ascendente o descendente, según se especifique.

* Función principal: Ordenar los elementos de un RDD de tipo clave-valor según las claves.
* Uso común: Se utiliza cuando es necesario ordenar los datos por clave, por ejemplo, para mostrar los resultados en orden ascendente o descendente según la clave.

A screenshot of a computer program

Description automatically generated

Explicación del código:

1. Crear el RDD: Usamos parallelize() para crear un RDD a partir de la lista ventas, que contiene pares clave-valor, donde la clave es la ciudad y el valor es el número de ventas.
2. sortByKey(): Aplicamos sortByKey() para ordenar los elementos del RDD por las claves (las ciudades), en orden ascendente por defecto.

Propósito de join()

El método join() en PySpark se utiliza para combinar dos RDDs de pares clave-valor en base a las claves. Este método realiza una operación de "join" similar a las bases de datos relacionales, donde los elementos de los dos RDDs que tienen la misma clave se combinan en un solo elemento. Los valores de las claves coincidentes se combinan en tuplas o listas, dependiendo de la implementación.

* Función principal: Combina dos RDDs de tipo clave-valor con base en las claves comunes.
* Uso común: Se utiliza para unir información de diferentes fuentes que tienen claves en común.

A screenshot of a computer code

Description automatically generated

Explicación del código:

1. Crear los RDDs: Usamos parallelize() para crear dos RDDs, uno con las ventas y otro con la población, ambos representando pares clave-valor donde la clave es la ciudad.
2. join(): Aplicamos el método join() sobre los dos RDDs. Este método combina los valores asociados a las mismas claves en una tupla. El resultado es un RDD de pares clave-valor donde la clave es la ciudad, y el valor es una tupla con las ventas y la población de esa ciudad.

Propósito de cogroup()

El método cogroup() en PySpark se utiliza para agrupar dos RDDs de tipo clave-valor por su clave, pero a diferencia de join(), que solo devuelve las claves comunes entre ambos RDDs, cogroup() devuelve las claves de ambos RDDs, junto con una colección de valores para cada clave, incluso si la clave solo existe en uno de los RDDs.

* Función principal: Permite agrupar dos RDDs por clave y generar un nuevo RDD donde cada clave tiene dos listas de valores: una para cada RDD.
* Uso común: Se utiliza cuando se quiere combinar dos RDDs pero manteniendo todas las claves de ambos RDDs y asociando listas de valores para cada clave.

A screenshot of a computer program

Description automatically generated

Explicación del código:

1. Crear los RDDs: Usamos parallelize() para crear dos RDDs, uno con las ventas y otro con los salarios, ambos representando pares clave-valor donde la clave es la ciudad.
2. cogroup(): Aplicamos el método cogroup() entre los dos RDDs. Este método devuelve un nuevo RDD donde cada clave tiene dos listas asociadas: una con los valores del primer RDD (ventas) y otra con los valores del segundo RDD (salarios). Si una ciudad no tiene un valor en uno de los RDDs, la lista correspondiente estará vacía.

* Las ciudades Lima, Moyobamba, Arequipa y Cusco tienen tanto ventas como salarios asociados.
* Rioja tiene ventas pero no tiene salario asociado (la lista de salarios está vacía).
* Cusco tiene salario pero no tiene ventas (la lista de ventas está vacía).
* Las listas de valores: Las listas de ventas y salarios para cada ciudad están agrupadas, aunque algunas ciudades tengan listas vacías si no hay datos correspondientes en uno de los RDDs.

Propósito de coalesce()

El método coalesce() en PySpark se utiliza para reducir el número de particiones en un RDD o DataFrame, lo cual puede ser útil para optimizar el rendimiento en operaciones como la escritura de datos a un archivo o la computación en un conjunto más pequeño de particiones. Este método reduce las particiones existentes sin realizar un shuffle costoso, lo cual lo hace más eficiente en términos de rendimiento que el método repartition().

* Reducción de particiones: Este método permite reducir el número de particiones de un RDD o DataFrame, lo que puede ser útil para optimizar la ejecución de las tareas finales, como guardar los resultados.
* Sin shuffle costoso: A diferencia de repartition(), que puede causar un shuffle (reorganización de los datos entre nodos), coalesce() simplemente combina las particiones existentes de manera más eficiente, lo que resulta en un menor costo computacional.

A screenshot of a computer program

Description automatically generated  
Explicación del código:

1. Contexto de Spark: Configuramos y creamos un contexto de Spark para trabajar con los RDDs.
2. Crear RDD: Creamos un RDD a partir de una lista de tuplas que contienen las ventas por ciudad. Al principio, creamos 6 particiones para los datos.
3. coalesce(): Usamos coalesce(2) para reducir el número de particiones de 6 a 2. Esto se hace para optimizar el rendimiento, especialmente cuando vamos a escribir los resultados en un archivo o realizar operaciones de agregación.
4. Antes de usar coalesce(), el RDD tiene 6 particiones.
5. Después de aplicar coalesce(2), el número de particiones se reduce a 2, lo que puede ser más eficiente para ciertas operaciones como la escritura de datos.

Propósito de reduce()

El método reduce() en PySpark se utiliza para combinar los elementos de un RDD (Resilient Distributed Dataset) de forma acumulativa utilizando una función que toma dos elementos del mismo tipo y devuelve un único valor del mismo tipo. El resultado final es un solo valor que es el resultado de aplicar la operación de reducción a través de todas las particiones del RDD.

Combinación acumulativa: Reduce todos los elementos de un RDD a un solo valor usando una función de combinación. Es comúnmente utilizado en tareas como la suma, el producto, la concatenación, la búsqueda del valor máximo o mínimo, etc.

Función binaria: La función proporcionada debe ser asociativa y conmutativa para que se pueda aplicar de manera eficiente en paralelo a través de varias particiones.

A screenshot of a computer program

Description automatically generated

Explicación del resultado:

* La función reduce() toma los valores de ventas (30, 25, 12, 18, 10, 22) y los combina uno por uno utilizando la operación de suma (a + b).
* El resultado final es la suma total de todas las ventas, que en este caso es 117.

Propósito de collect()

El método collect() en PySpark es una acción que se utiliza para recuperar los datos de un RDD (Resilient Distributed Dataset) o DataFrame y devolverlos a la máquina local, es decir, recoger los resultados procesados en el clúster de Spark y traerlos al contexto local (driver).

* Recoger todos los resultados: collect() se usa cuando deseas obtener todos los elementos de un RDD o DataFrame que han sido procesados en el clúster y almacenarlos en una lista en el programa local.
* No apto para grandes volúmenes de datos: Es importante tener en cuenta que collect() trae todos los datos al driver. Si el RDD o DataFrame contiene una gran cantidad de datos, esto puede causar problemas de memoria o incluso una falla en el sistema, ya que puede intentar almacenar demasiados datos en la memoria de la máquina local.
* Acción en lugar de transformación: A diferencia de las transformaciones como map(), filter(), flatMap(), etc., que devuelven un nuevo RDD, collect() es una acción que desencadena la ejecución de todas las transformaciones acumuladas en un RDD.

A screenshot of a computer program

Description automatically generated

Explicación del resultado:

* collect() devuelve todos los elementos del RDD como una lista de Python. En este caso, la salida es una lista de tuplas donde cada tupla representa una ciudad y su respectiva cantidad de ventas.

Propósito de count()

El método count() en PySpark es una acción que se utiliza para contar el número de elementos en un RDD o DataFrame. Esta acción devuelve el número total de elementos presentes en el RDD o DataFrame, es decir, el tamaño del conjunto de datos.

* Contar los elementos: Se utiliza para obtener el número total de elementos en un RDD o DataFrame. Es útil cuando necesitamos saber cuántos elementos se encuentran en el conjunto de datos, sin necesidad de materializar todos los valores.
* Operación de acción: count() es una acción porque desencadena el procesamiento de los datos, a diferencia de las transformaciones (como map(), filter(), etc.), que solo definen una nueva estructura de datos y no ejecutan el procesamiento.

A screenshot of a computer program

Description automatically generated

Explicación del resultado:

* count() devuelve el número de elementos que hay en el RDD. En este caso, hay 6 tuplas en el RDD (una por cada ciudad), por lo que el resultado es 6.

Propósito de first()

El método first() en PySpark es una acción que se utiliza para obtener el primer elemento de un RDD o DataFrame. Este método devuelve el primer elemento del conjunto de datos sin necesidad de procesar todo el RDD o DataFrame. Es útil cuando solo necesitas ver un ejemplo o el primer valor de tu conjunto de datos.

* Obtener el primer elemento: first() se utiliza para obtener el primer elemento de un RDD o DataFrame sin la necesidad de procesar todo el conjunto de datos. Es eficiente si solo deseas una muestra del principio de los datos.
* Operación de acción: Similar a otros métodos como collect() o count(), first() es una acción que desencadena el procesamiento de los datos para recuperar el primer valor.

A screenshot of a computer program

Description automatically generated  
Explicación del resultado:

* first() devuelve el primer elemento del RDD o DataFrame. En este caso, el primer elemento es la tupla ('Lima', 30), lo que indica que las ventas en Lima fueron 30.

Propósito de take(n)  
  
El método take(n) en PySpark es una acción que se utiliza para recuperar los primeros n elementos de un RDD o DataFrame. Este método es útil cuando solo necesitas ver una pequeña muestra de los datos sin procesar todo el conjunto de datos.

* Obtener los primeros n elementos: take(n) devuelve una lista con los primeros n elementos de un RDD o DataFrame. Es útil cuando deseas examinar los primeros registros del conjunto de datos, pero no necesitas cargar todos los datos en memoria.
* Operación de acción: Al igual que otras acciones como first() o count(), take(n) ejecuta un procesamiento real de los datos para recuperar los elementos solicitados.

A screenshot of a computer program

Description automatically generated

Explicación del resultado:

* take(3) devuelve los primeros 3 elementos del RDD. En este caso, las primeras tres ciudades con sus ventas: ('Lima', 30), ('Moyobamba', 25), y ('Rioja', 12).

Propósito de saveAsTextFile()

El método saveAsTextFile() en PySpark es una acción que se utiliza para guardar el contenido de un RDD en un archivo de texto. Este método toma el RDD y lo guarda en el sistema de archivos en forma de archivos de texto, distribuidos entre varias particiones, dependiendo del número de particiones del RDD.

* Guardar RDD en un archivo: Permite exportar los datos contenidos en un RDD a archivos de texto, lo que puede ser útil cuando deseas almacenar los resultados de un procesamiento de datos en un formato legible.
* Operación de acción: Al igual que otras acciones en PySpark, como collect() o first(), saveAsTextFile() inicia un proceso real de escritura en el sistema de archivos y no simplemente transforma los datos en memoria.

A screenshot of a computer program

Description automatically generated

Explicación del código:

1. Detener contexto activo: Si ya existe un contexto de Spark, se detiene antes de crear uno nuevo.
2. Configurar el contexto de Spark: Creamos un contexto de Spark utilizando la configuración dada (nombre de la aplicación y maestro local).
3. Crear el RDD: Convertimos la lista ventas en un RDD. Cada elemento es una tupla con un identificador de transacción, producto y cantidad.
4. map(): Utilizamos map() para extraer solo los productos y las cantidades, generando un nuevo RDD con pares (producto, cantidad).
5. reduceByKey(): Usamos reduceByKey() para agregar las cantidades de cada producto. Esto agrupa los productos por nombre y luego suma las cantidades.
6. map() para formato de texto: Usamos map() nuevamente para formatear cada par clave-valor como una cadena en el formato Producto: Total\_Vendido.
7. Guardar en archivo de texto: Guardamos el RDD resultante en un directorio llamado ventas\_totales\_salida como un archivo de texto.

Uso de max() y min()

En PySpark, los RDDs tienen las funciones max() y min() que se utilizan para obtener el valor máximo o mínimo en un RDD. Estas funciones operan sobre los valores del RDD y pueden ser útiles para obtener el valor más alto o más bajo de un conjunto de datos numéricos o de otro tipo.

* max(): Retorna el valor máximo en el RDD.
* min(): Retorna el valor mínimo en el RDD.

A screenshot of a computer program

Description automatically generated

Explicación del código:

1. Creación del RDD: Se crea un RDD con una lista de ventas, donde cada elemento es una tupla con un ID de transacción, el nombre del producto y la cantidad vendida.
2. map para transformar el RDD: Se usa el map para obtener un nuevo RDD con tuplas de la forma (producto, cantidad), donde producto es el nombre del artículo y cantidad es la cantidad vendida.
3. reduceByKey para agregar las cantidades por producto: Luego, usamos reduceByKey para agregar las cantidades de cada producto y obtener la cantidad total vendida por producto.
4. Uso de max() y min(): Finalmente, usamos las funciones max() y min() para obtener el producto con la mayor y menor cantidad de ventas, respectivamente. La función max() y min() se aplican sobre el segundo elemento de cada tupla (que es la cantidad vendida).

Uso de countByKey()

En PySpark, la función countByKey() se utiliza para contar el número de ocurrencias de cada clave en un RDD de tuplas (clave, valor). Retorna un diccionario con las claves y sus respectivas frecuencias en el RDD.  
countByKey(): Cuenta cuántas veces aparece cada clave en el RDD.

A screenshot of a computer program

Description automatically generated

Explicación del código:

1. Creación del RDD: Se crea un RDD rdd\_ventas con una lista de ventas, donde cada elemento es una tupla (producto, cantidad).
2. countByKey(): Usamos la función countByKey() para contar las ocurrencias de cada clave (producto) en el RDD.
3. La función countByKey() te permite obtener rápidamente la frecuencia de cada clave en un RDD de tuplas, lo cual es útil cuando trabajas con datos categóricos o agrupados.

Uso de foreach()  
En PySpark, el foreach() es una acción que se utiliza para aplicar una función a cada elemento de un RDD o DataFrame, sin devolver un resultado. Es una acción que se ejecuta de manera distribuida en los nodos de tu clúster de Spark.  
No retorna un valor: La función aplicada a cada elemento no devuelve un nuevo RDD, sino que realiza una acción en cada elemento (por ejemplo, escribir en un archivo, actualizar una base de datos, etc.).

Es útil para efectos secundarios: Ideal para situaciones donde necesitas hacer algo con los datos, como escribir en un archivo o enviar los resultados a un sistema externo.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated  
Explicación del código:

1. Creación del RDD: Se crea un RDD rdd\_ventas con una lista de ventas, donde cada elemento es una tupla (producto, cantidad).
2. Definición de la función imprimir\_venta: Esta función toma un elemento del RDD (una tupla) y simplemente imprime el producto y la cantidad vendida.
3. Uso de foreach(): Se aplica foreach(imprimir\_venta) para ejecutar la función imprimir\_venta en cada elemento del RDD. Esto imprimirá cada venta en la consola.