Ex. No. 4	**Matrix Multiplication Using RDDs in Spark**
Youtube Link	**https://youtu.be/aqhQ5MDyEuI**
Date of Exercise	13.10.25

AIM

To compute the product of two matrices **A** ($m \times n$) and **B** ($n \times p$) using **Apache Spark RDDs**

Procedure:

1. Data Preparation

- Define two small matrices **A** and **B** (e.g., A of shape 2×3, B of shape 3×2) using MatrixEntry.

2. Spark Logic

1. Create two RDDs: entriesA and entriesB, each as RDD[MatrixEntry].
2. Map:
   - aKeyed = entriesA.map(e => (e.j, (e.i, e.value)))
   - bKeyed = entriesB.map(e => (e.i, (e.j, e.value)))
3. Join on key j (common index).
4. Map to ((i, k), product) and use reduceByKey to sum contributions.
5. Construct CoordinateMatrix from summed entries and collect the results.

Program:

```
from pyspark.sql import SparkSession

from pyspark import SparkContext

from pyspark.mllib.linalg.distributed import MatrixEntry, CoordinateMatrix
```

```
# Initialize SparkSession (CoordinateMatrix internally converts RDD -> DataFrame)
spark = SparkSession.builder.appName("PyMatrixMultiplyRDD").getOrCreate()
sc = spark.sparkContext

entriesA = sc.parallelize([
    MatrixEntry(0, 0, 1.0), MatrixEntry(0, 2, 2.0),
    MatrixEntry(1, 1, 3.0), MatrixEntry(1, 2, -1.0)
])
entriesB = sc.parallelize([
    MatrixEntry(0, 0, 1.0), MatrixEntry(1, 0, 3.0), MatrixEntry(2, 0, 5.0),
    MatrixEntry(0, 1, 2.0), MatrixEntry(1, 1, 4.0), MatrixEntry(2, 1, 6.0)
])

aKeyed = entriesA.map(lambda e: (e.j, (e.i, e.value)))
bKeyed = entriesB.map(lambda e: (e.i, (e.j, e.value)))

product = (aKeyed.join(bKeyed)
           .map(lambda kv: ((kv[1][0][0], kv[1][1][0]), kv[1][0][1] * kv[1][1][1]))
           .reduceByKey(lambda a, b: a + b)
           .map(lambda ik_sum: MatrixEntry(ik_sum[0][0], ik_sum[0][1], ik_sum[1])))

result = CoordinateMatrix(product)
for e in result.entries.collect():
    print(f"({e.i},{e.j}) = {e.value}")

spark.stop()
```

Output:

```
--- Retrieval Actions ---

Collect: ['Spark', 'is', 'fast', 'Spark', 'is', 'powerful', 'Spark', 'is', 'easy', 'to', 'use']

Take(5): ['Spark', 'is', 'fast', 'Spark', 'is']

TakeSample (no replacement, 4): ['is', 'to', 'easy', 'Spark']

TakeOrdered (alphabetical, 5): ['Spark', 'Spark', 'Spark', 'easy', 'fast']

Top(5): ['use', 'to', 'powerful', 'is', 'is']

First element: Spark

IsEmpty?: False


Foreach output:

Word: Spark

Word: is

Word: fast

Word: Spark

Word: is

Word: powerful

Word: Spark

Word: is

Word: easy

Word: to

Word: use
```

```
--- Aggregation Actions ---

Count: 11

CountByValue: {'Spark': 3, 'is': 3, 'fast': 1, 'powerful': 1, 'easy': 1, 'to': 1, 'use': 1}

Reduce (total word count): 11

Fold (total word count): 11

Aggregate (avg word length): 3.8181818181818183
```

Result :

The resulting **2×2** product matrix is correctly computed using distributed operations.