Name: Prerna Sunil Jadhav
Sap ID: 60004220127
Batch: C22
Course: Big Data Infrastructure laboratory
Course code: DJ19CEEL6011

EXPERIMENT 07

AIM: Implement RDD using Python.

THEORY:

SPARK

→ Apache spark is an open-source, distributed processing system used for dig data workloads
→ It utilizes in-memory caching, and optimized query execution for fast analytics queries against data of any size.
→ Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory and 10 times faster when running on disk. This is possible by reducing the no. of read/write operations to disk.
→ Spark not only supports 'Map' and 'Reduce'. It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads like batch processing, interactive queries, real-time analytics, machine learning & graph processing

RDD (Resilient Distributed Dataset)

→ It is the fundamental data structure of Apache Spark

→ RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster.

→ Decomposing the name RDD:

1) Resilient: It is fault-tolerant with the help of RDD lineage graph (DAG) and so able to recompute missing or damaged partitions due to node failure.

2) Distributed: Since Data resides on multiple nodes

3) Dataset: It represents records of the data you work with. The user can data the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.

```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 4.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packag
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=3174884
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
```

```
from pyspark import SparkContext, SparkConf
import math

# Create a SparkContext
conf = SparkConf().setAppName("SquareRootNumbers").setMaster("local")
sc = SparkContext(conf=conf)

# Create an RDD containing numbers from 1 to 20
numbers_rdd = sc.parallelize(range(1, 21))

# Apply the square root function to each element in the RDD
square_root_rdd = numbers_rdd.map(lambda x: math.sqrt(x))

# Collect and print square roots
square_roots = square_root_rdd.collect()
for square_root in square_roots:
    print(square_root)

# Stop the SparkContext
sc.stop()
```

```
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
3.1622776601683795
3.3166247903554
3.4641016151377544
3.605551275463989
3.7416573867739413
3.872983346207417
4.0
4.123105625617661
4.242640687119285
```

```
4.358898943540674
4.47213595499958
```

```
from pyspark import SparkContext, SparkConf

# Initialize Spark
conf = SparkConf().setAppName("ArmstrongNumbers").setMaster("local[*]")
sc = SparkContext(conf=conf)

# Define a function to check if a number is an Armstrong number
def is_armstrong(num):
    order = len(str(num))
    sum = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        sum += digit ** order
        temp //= 10
    return num == sum

# Create an RDD containing numbers from 100 to 9999
numbers_rdd = sc.parallelize(range(100, 10000))

# Filter the RDD to keep only Armstrong numbers
armstrong_rdd = numbers_rdd.filter(is_armstrong)

# Collect and print the Armstrong numbers
armstrong_numbers = armstrong_rdd.collect()
print("Armstrong numbers between 100 and 9999:", armstrong_numbers)

# Stop Spark
sc.stop()
```

```
Armstrong numbers between 100 and 9999: [153, 370, 371, 407, 1634, 8208, 9474]
```

```
 from pyspark import SparkContext, SparkConf

# Create a SparkContext
conf = SparkConf().setAppName("SquaredNumbers").setMaster("local")
```

```
from pyspark import SparkContext, SparkConf

# Initialize Spark
conf = SparkConf().setAppName("PerfectNumbers").setMaster("local[*]")
sc = SparkContext(conf=conf)

# Define a function to check if a number is a Perfect number
def is_perfect(num):
    sum_divisors = sum([i for i in range(1, num) if num % i == 0])
    return sum_divisors == num

# Create an RDD containing numbers from 1 to 100
numbers_rdd = sc.parallelize(range(1, 101))

# Filter the RDD to keep only Perfect numbers
perfect_rdd = numbers_rdd.filter(is_perfect)

# Collect and print the Perfect numbers
perfect_numbers = perfect_rdd.collect()
print("Perfect numbers between 1 and 100:", perfect_numbers)

# Stop Spark
sc.stop()
```

```
Perfect numbers between 1 and 100: [6, 28]
```