BIG DATA ANALYTICS

LAB ASSIGNMENT 10

Mahesh Pachare

FINAL YEAR B.TECH

IT 191080054

Aim :

Flight Data Analysis using Spark GraphX

- Compute the total number of flight routes.
- Compute and sort the longest flight routes.
- Display the airport with the highest degree vertex.
- List the most important airports according to PageRank.
- List the routes with the lowest flight costs.

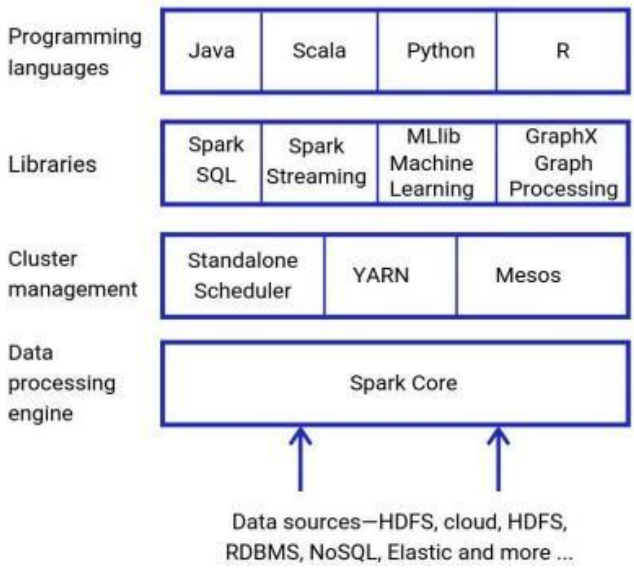
Theory :

Apache Spark :

Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. These two qualities are key to the worlds of big data and machine learning, which require the marshalling of massive computing power to crunch through large data stores. Spark also takes some of the programming burdens of these tasks off the shoulders of developers with an easy-to-use API that abstracts away much of the grunt work of distributed computing and big data processing.

Apache Spark architecture :

Apache Spark has a well-defined and layered architecture where all the spark components and layers are loosely coupled and integrated with various extensions and libraries. However, before delving deep into understanding how spark works, it is important to understand the Apache Spark Ecosystem and what are the key components of the Spark run-time architecture.



Spark GraphX :

GraphX is a new component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and aggregateMessages) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

EXECUTION STEPS AND OUTPUT :

- **Starting all daemons**

```
hduser@master:/usr/local/spark/sbin$ jps
4432 Jps
2950 NameNode
3320 SecondaryNameNode
4089 Master
4217 Worker
3098 DataNode
3549 ResourceManager
3694 NodeManager
hduser@master:/usr/local/spark/sbin$
```

```
hduser@slave:~/Desktop$ jps
2257 DataNode
2420 NodeManager
2970 Jps
2607 Worker
hduser@slave:~/Desktop$
```

- **Copying flights dataset into HDFS**

```
hduser@master:~$ hdfs dfs -put Flights.csv /test_input
hduser@master:~$ hdfs dfs -ls /test_input
Found 2 items
-rw-r--r--   2 hduser supergroup    1084310 2023-04-30 23:12 /test_input/Flights.csv
-rw-r--r--   2 hduser supergroup       9125 2023-04-29 23:58 /test_input/seeds.csv
hduser@master:~$
```

- **Scala code for analysing flight data using GraphX :**

```
var
 df=spark.read.format("csv").option("header","true").option("inferSchema"
,"true").lo
 ad("hdfs://master:9000/test_input/Flights.csv")
 df.show()
df.printSchema()
```

```
df.columns
df.count()
df.columns.size
df.describe().show(
)
var final_df = df.select("Source", "Destination", "Duration", "Price")
var vertices_df =
final_df.select("Source").union(final_df.select("Destination")).distinct
()
vertices_df = vertices_df.withColumn("id", monotonically_increasing_id +
0)
vertices_df.cache()
vertices_df.show()
var vertices_rdd = vertices_df.rdd
vertices_rdd.collect()
var kv_rdd = vertices_df.rdd.map(row =>
(row.getAs[Long](1),(row.getAs[String](0))))
kv_rdd.collect()
import org.apache.spark.graphx._
var joined = final_df.join(vertices_df, Seq("Source"), "left_outer")
joined.show()
import org.apache.spark.sql.functions._
joined = joined.select($"Destination", $"Duration",
$"Price",$"id".alias("origin")).drop("id")
vertices_df =
vertices_df.select($"Source".alias("Destination"),$"id").drop("Source")
vertices_df.show()
joined = joined.join(vertices_df, Seq("Destination"), "left_outer")
joined = joined.select($"origin", $"id".alias("destination"),
$"Duration",
$"Price").drop("id")
joined.show()
var edges_rdd1 = joined.rdd.map(row => Edge(row.getAs[Long]("origin"),
row.getAs[Long]("destination"), (row.getAs[Int]("Price"),
row.getAs[Double]("Duration"))))
edges_rdd1.collect() val
nowhere=”nowhwere”
```

```
val graph = Graph(kv_rdd, edges_rdd1, nowhere)
graph.vertices.collect.take(100)
graph.edges.collect.take(100)
println(s"Number of Flight Routes: ${graph.numEdges} \n")
println(s"Number of Airports: ${graph.numVertices} \n")
var sortedEdges = graph.edges.distinct().sortBy(edge => -edge.attr._2)
println("Longest Routes (time):")
sortedEdges.take(100)
sortedEdges.take(20).foreach(edge => println(s"Source:
${kv_rdd.lookup(edge.srcId)(0)}; Destination:
${kv_rdd.lookup(edge.dstId)(0)};
Cost: ${edge.attr._1}; Duration:${edge.attr._2}"))
println("\n")
println("Indegrees of each Vertex (Airport):")
graph.inDegrees.collect()
println("\n")
var inDegrees = graph.inDegrees.distinct().sortBy(-1*_._2)
println("Vertices (Airports) with highest Indegrees:")
inDegrees.take(100)
println("\n")
println(s"Airport with highest Indegree vertex:
${kv_rdd.lookup(inDegrees.take(1)(0)._1)(0)} with an Indegree of
${inDegrees.take(1)(0)._2}")
println("\n")
var pgrank = graph.pageRank(0.00001)
pgrank.vertices.sortBy(-_._2).collect()
println(s"The Airport with highest PageRank
is
${kv_rdd.lookup(pgrank.vertices.sortBy(-_._2).take(1)(0)._1)(0)} and has
a PageRank
value of ${pgrank.vertices.sortBy(-_._2).take(1)(0)._2}")
println("\n")
pgrank.vertices.sortBy(-_._2).take(20).foreach(vertex =>
println(s"Airport:
${kv_rdd.lookup(vertex._1)(0)}; PageRank value: ${vertex._2}"))
println("\n")
var sortedPrices = graph.edges.distinct().sortBy(edge => edge.attr._1)
```

```
println("Routes with lowest cost:")
 sortedPrices.take(100)
 sortedPrices.take(20).foreach(edge
 =>
println(s"Source:{kv_rdd.lookup(edge.srcId)(0)}; Destination:
${kv_rdd.lookup(edge.dstId)(0)};
Cost: ${edge.attr._1};Duration:${edge.attr._2)"))
 :quit
```

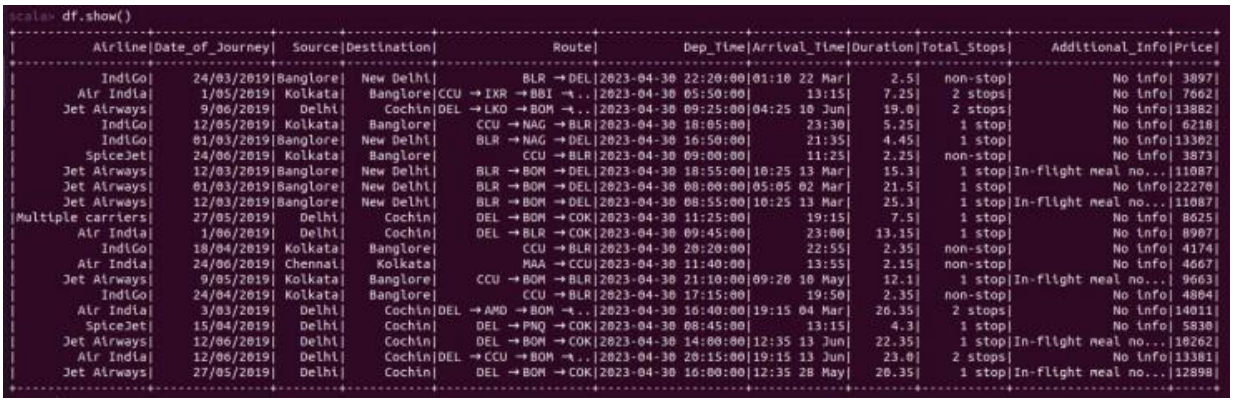
- **Running code in Spark Shell to analyse flight data:**

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/04/30 23:16:49 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform.
.. using builtin-java classes where applicable
Spark context Web UI available at http://master:4040
Spark context available as 'sc' (master = local[*], app id = local-1682876812731).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.4.0
      /_/

Using Scala version 2.12.17 (OpenJDK 64-Bit Server VM, Java 1.8.0_362)
Type in expressions to have them evaluated.
Type :help for more information.

scala> var df=spark.read.format("csv").option("header","true").option("inferSchema","true").load
("hdfs://master:9000/test_input/Flights.csv")
df: org.apache.spark.sql.DataFrame = [Airline: string, Date_of_Journey: string ... 9 more fields
]
```

- **The output of the Python script**



```
scala> df.show()
+-----------------+---------------+-------+-----------+--------------------+----------+------------+--------+-----------+--------------------+-----+
|          Airline|Date_of_Journey| Source|Destination|               Route|  Dep_Time|Arrival_Time|Duration|Total_Stops|     Additional_Info|Price|
+-----------------+---------------+-------+-----------+--------------------+----------+------------+--------+-----------+--------------------+-----+
|          IndiGo|     24/03/2019|Banglore|  New Delhi|         BLR → DEL|22:20:00|01:10 22 Mar|    2.5|   non-stop|            No info| 3897|
|        Air India|      1/05/2019|Kolkata|  Banglore|CCU → IXR → BBI →...|05:50:00|       13:15|    7.25|    2 stops|            No info| 7662|
|      Jet Airways|      9/06/2019|  Delhi|     Cochin|DEL → LKO → BOM →...|09:25:00|04:25 10 Jun|    19.0|    2 stops|            No info|13882|
|          IndiGo|     12/05/2019|Kolkata|  Banglore|     CCU → NAG → BLR|18:05:00|       23:30|    5.25|     1 stop|            No info| 6218|
|          IndiGo|     01/03/2019|Banglore|  New Delhi|     BLR → NAG → DEL|16:50:00|       21:35|    4.45|     1 stop|            No info|13302|
|        SpiceJet|     24/06/2019|Kolkata|  Banglore|         CCU → BLR|09:00:00|       11:25|    2.25|   non-stop|            No info| 3873|
|      Jet Airways|     12/03/2019|Banglore|  New Delhi|     BLR → BOM → DEL|18:55:00|10:25 13 Mar|    15.3|     1 stop|In-flight meal no...|11087|
|      Jet Airways|     01/03/2019|Banglore|  New Delhi|     BLR → BOM → DEL|08:00:00|05:05 02 Mar|    21.5|     1 stop|            No info|22270|
|      Jet Airways|     12/03/2019|Banglore|  New Delhi|     BLR → BOM → DEL|08:55:00|10:25 13 Mar|    25.3|     1 stop|In-flight meal no...|11087|
|Multiple carriers|     27/05/2019|  Delhi|     Cochin|     DEL → BOM → COK|11:25:00|       19:15|     7.5|     1 stop|            No info| 8625|
|        Air India|      1/06/2019|  Delhi|     Cochin|     DEL → BLR → COK|09:45:00|       23:00|   13.15|     1 stop|            No info| 8907|
|          IndiGo|     18/04/2019|Kolkata|  Banglore|         CCU → BLR|20:20:00|       22:55|    2.35|   non-stop|            No info| 4174|
|        Air India|     24/06/2019|Chennai|    Kolkata|         MAA → CCU|11:40:00|       13:55|    2.15|   non-stop|            No info| 4667|
|      Jet Airways|      9/05/2019|Kolkata|  Banglore|     CCU → BOM → BLR|21:10:00|09:20 10 May|    12.1|     1 stop|In-flight meal no...| 9663|
|          IndiGo|     24/04/2019|Kolkata|  Banglore|         CCU → BLR|17:15:00|       19:50|    2.35|   non-stop|            No info| 4804|
|        Air India|      3/03/2019|  Delhi|     Cochin|DEL → AMD → BOM →...|16:40:00|19:15 04 Mar|   26.35|    2 stops|            No info|14011|
|        SpiceJet|     15/04/2019|  Delhi|     Cochin|     DEL → PNQ → COK|08:45:00|       13:15|     4.3|     1 stop|            No info| 5830|
|      Jet Airways|     12/06/2019|  Delhi|     Cochin|     DEL → BOM → COK|14:00:00|12:35 13 Jun|   22.35|     1 stop|In-flight meal no...|10262|
|        Air India|     12/06/2019|  Delhi|     Cochin|DEL → CCU → BOM →...|20:15:00|19:15 13 Jun|    23.0|    2 stops|            No info|13381|
|      Jet Airways|     27/05/2019|  Delhi|     Cochin|     DEL → BOM → COK|16:00:00|12:35 28 May|   20.35|     1 stop|In-flight meal no...|12898|
+-----------------+---------------+-------+-----------+--------------------+----------+------------+--------+-----------+--------------------+-----+
```

```
scala> df.printSchema()
root
 |-- Airline: string (nullable = true)
 |-- Date_of_Journey: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Destination: string (nullable = true)
 |-- Route: string (nullable = true)
 |-- Dep_Time: timestamp (nullable = true)
 |-- Arrival_Time: string (nullable = true)
 |-- Duration: double (nullable = true)
 |-- Total_Stops: string (nullable = true)
 |-- Additional_Info: string (nullable = true)
 |-- Price: integer (nullable = true)


scala> df.columns
res2: Array[String] = Array(Airline, Date_of_Journey, Source, Destination, Route, Dep_Time, Arri
val_Time, Duration, Total_Stops, Additional_Info, Price)

scala> df.count()
res3: Long = 10683

scala> df.columns.size
res4: Int = 11
```

```
scala> var final_df = df.select("Source", "Destination", "Duration", "Price")
final_df: org.apache.spark.sql.DataFrame = [Source: string, Destination: string ... 2 more field
s]

scala> var vertices_df = final_df.select("Source").union(final_df.select("Destination")).distinc
t()
vertices_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Source: string]

scala> vertices_df = vertices_df.withColumn("id", monotonically_increasing_id + 0)
vertices_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Source: string, id: bigin
t]

scala> vertices_df.cache()
res6: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Source: string, id: bigint]

scala> vertices_df.show()
+---------+-------------+
|   Source|           id|
+---------+-------------+
|  Chennai| 300647710720|
|   Mumbai| 438086664192|
|  Kolkata| 592705486848|
|    Delhi|1108101562368|
|   Cochin|1297080123392|
|New Delhi|1374389534720|
| Banglore|1425929142272|
|Hyderabad|1425929142273|
+---------+-------------+
```

```
scala> var vertices_rdd = vertices_df.rdd
vertices_rdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[50] at rdd a
t <console>:23

scala> vertices_rdd.collect()
res8: Array[org.apache.spark.sql.Row] = Array([Chennai,300647710720], [Mumbai,438086664192], [Ko
lkata,592705486848], [Delhi,1108101562368], [Cochin,1297080123392], [New Delhi,1374389534720], [
Banglore,1425929142272], [Hyderabad,1425929142273])

scala> var kv_rdd = vertices_df.rdd.map(row => (row.getAs[Long](1),(row.getAs[String](0))))
kv_rdd: org.apache.spark.rdd.RDD[(Long, String)] = MapPartitionsRDD[51] at map at <console>:23

scala> kv_rdd.collect()
res9: Array[(Long, String)] = Array((300647710720,Chennai), (438086664192,Mumbai), (592705486848
,Kolkata), (1108101562368,Delhi), (1297080123392,Cochin), (1374389534720,New Delhi), (1425929142
272,Banglore), (1425929142273,Hyderabad))

scala> import org.apache.spark.graphx._
import org.apache.spark.graphx._

scala> var joined = final_df.join(vertices_df, Seq("Source"), "left_outer")
joined: org.apache.spark.sql.DataFrame = [Source: string, Destination: string ... 3 more fields]
```

```
scala> joined.show()
+--------+-----------+--------+-----+-------------+
|  Source|Destination|Duration|Price|           id|
+--------+-----------+--------+-----+-------------+
|Banglore|  New Delhi|     2.5| 3897|1425929142272|
| Kolkata|   Banglore|    7.25| 7662| 592705486848|
|   Delhi|     Cochin|    19.0|13882|1108101562368|
| Kolkata|   Banglore|    5.25| 6218| 592705486848|
|Banglore|  New Delhi|    4.45|13302|1425929142272|
| Kolkata|   Banglore|    2.25| 3873| 592705486848|
|Banglore|  New Delhi|    15.3|11087|1425929142272|
|Banglore|  New Delhi|    21.5|22270|1425929142272|
|Banglore|  New Delhi|    25.3|11087|1425929142272|
|   Delhi|     Cochin|     7.5| 8625|1108101562368|
|   Delhi|     Cochin|   13.15| 8907|1108101562368|
| Kolkata|   Banglore|    2.35| 4174| 592705486848|
| Chennai|    Kolkata|    2.15| 4667| 300647710720|
| Kolkata|   Banglore|    12.1| 9663| 592705486848|
| Kolkata|   Banglore|    2.35| 4804| 592705486848|
|   Delhi|     Cochin|   26.35|14011|1108101562368|
|   Delhi|     Cochin|     4.3| 5830|1108101562368|
|   Delhi|     Cochin|   22.35|10262|1108101562368|
|   Delhi|     Cochin|    23.0|13381|1108101562368|
|   Delhi|     Cochin|   20.35|12898|1108101562368|
+--------+-----------+--------+-----+-------------+
only showing top 20 rows


scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._

scala> joined = joined.select($"Destination", $"Duration", $"Price",$"id".alias("origin")).drop(
"id")
joined: org.apache.spark.sql.DataFrame = [Destination: string, Duration: double ... 2 more field
s]
```

```
scala> vertices_df = vertices_df.select($"Source".alias("Destination"),$"id").drop("Source")
vertices_df: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Destination: string, id:
bigint]

scala> vertices_df.show()
+-----------+-------------+
|Destination|           id|
+-----------+-------------+
|    Chennai| 300647710720|
|     Mumbai| 438086664192|
|    Kolkata| 592705486848|
|      Delhi|1108101562368|
|     Cochin|1297080123392|
|  New Delhi|1374389534720|
|   Banglore|1425929142272|
|  Hyderabad|1425929142273|
+-----------+-------------+


scala> joined = joined.join(vertices_df, Seq("Destination"), "left_outer")
joined: org.apache.spark.sql.DataFrame = [Destination: string, Duration: double ... 3 more field
s]

scala> joined = joined.select($"origin", $"id".alias("destination"), $"Duration", $"Price").drop
("id")
joined: org.apache.spark.sql.DataFrame = [origin: bigint, destination: bigint ... 2 more fields]
```

```
scala> joined.show()
+-------------+-------------+--------+-----+
|       origin|  destination|Duration|Price|
+-------------+-------------+--------+-----+
|1425929142272|1374389534720|     2.5| 3897|
| 592705486848|1425929142272|    7.25| 7662|
|1108101562368|1297080123392|    19.0|13882|
| 592705486848|1425929142272|    5.25| 6218|
|1425929142272|1374389534720|    4.45|13302|
| 592705486848|1425929142272|    2.25| 3873|
|1425929142272|1374389534720|    15.3|11087|
|1425929142272|1374389534720|    21.5|22270|
|1425929142272|1374389534720|    25.3|11087|
|1108101562368|1297080123392|     7.5| 8625|
|1108101562368|1297080123392|   13.15| 8907|
| 592705486848|1425929142272|    2.35| 4174|
| 300647710720| 592705486848|    2.15| 4667|
| 592705486848|1425929142272|    12.1| 9663|
| 592705486848|1425929142272|    2.35| 4804|
|1108101562368|1297080123392|   26.35|14011|
|1108101562368|1297080123392|     4.3| 5830|
|1108101562368|1297080123392|   22.35|10262|
|1108101562368|1297080123392|    23.0|13381|
|1108101562368|1297080123392|   20.35|12898|
+-------------+-------------+--------+-----+
only showing top 20 rows
```

- **Creating a Graph using VerticesRDD and EdgesRDD:**

```
scala> var edges_rdd1 = joined.rdd.map(row => Edge(row.getAs[Long]("origin"), row.getAs[Long]("d
estination"), (row.getAs[Int]("Price"), row.getAs[Double]("Duration"))))
edges_rdd1: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[(Int, Double)]] = MapPartition
sRDD[96] at map at <console>:29

scala> edges_rdd1.collect()
res13: Array[org.apache.spark.graphx.Edge[(Int, Double)]] = Array(Edge(1425929142272,13743895347
20,(3897,2.5)), Edge(592705486848,1425929142272,(7662,7.25)), Edge(1108101562368,1297080123392,(
13882,19.0)), Edge(592705486848,1425929142272,(6218,5.25)), Edge(1425929142272,1374389534720,(13
302,4.45)), Edge(592705486848,1425929142272,(3873,2.25)), Edge(1425929142272,1374389534720,(1108
7,15.3)), Edge(1425929142272,1374389534720,(22270,21.5)), Edge(1425929142272,1374389534720,(1108
7,25.3)), Edge(1108101562368,1297080123392,(8625,7.5)), Edge(1108101562368,1297080123392,(8907,1
3.15)), Edge(592705486848,1425929142272,(4174,2.35)), Edge(300647710720,592705486848,(4667,2.15)
), Edge(592705486848,1425929142272,(9663,12.1)), Edge(592705486848,1425929142272,(4804,2.35)), E
dge...

scala> val nowhere="nowhere"
nowhere: String = nowhere

scala> val graph = Graph(kv_rdd, edges_rdd1, nowhere)
graph: org.apache.spark.graphx.Graph[String,(Int, Double)] = org.apache.spark.graphx.impl.GraphI
mpl@57297a27

scala> graph.vertices.collect.take(100)
res14: Array[(org.apache.spark.graphx.VertexId, String)] = Array((1108101562368,Delhi), (3006477
10720,Chennai), (1297080123392,Cochin), (438086664192,Mumbai), (1374389534720,New Delhi), (14259
29142272,Banglore), (1425929142273,Hyderabad), (592705486848,Kolkata))

scala> graph.edges.collect.take(100)
res15: Array[org.apache.spark.graphx.Edge[(Int, Double)]] = Array(Edge(300647710720,592705486848
,(4667,2.15)), Edge(300647710720,592705486848,(3687,2.15)), Edge(300647710720,592705486848,(3687
,2.15)), Edge(300647710720,592705486848,(4667,2.15)), Edge(300647710720,592705486848,(7414,2.15)
), Edge(300647710720,592705486848,(3687,2.15)), Edge(300647710720,592705486848,(6297,2.15)), Edg
e(300647710720,592705486848,(3332,2.15)), Edge(300647710720,592705486848,(3540,2.25)), Edge(3006
47710720,592705486848,(3540,2.25)), Edge(300647710720,592705486848,(3597,2.2)), Edge(30064771072
0,592705486848,(3858,2.2)), Edge(300647710720,592705486848,(3543,2.15)), Edge(300647710720,59270
5486848,(3543,2.15)), Edge(300647710720,592705486848,(4842,2.15)), Edge(300647710720,59270548684
8,(...
```

- **Total Number of Flight Routes:**

```
scala> println(s"Number of Flight Routes: ${graph.numEdges} \n")
Number of Flight Routes: 10683


scala> println(s"Number of Airports: ${graph.numVertices} \n")
Number of Airports: 8
```

- **List of Longest Flight Routes**

```
scala> var sortedEdges = graph.edges.distinct().sortBy(edge => -edge.attr._2)
sortedEdges: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[(Int, Double)]] = MapPartiti
nsRDD[115] at sortBy at <console>:29

scala> println("Longest Routes (time):")
Longest Routes (time):

scala> sortedEdges.take(100)
res19: Array[org.apache.spark.graphx.Edge[(Int, Double)]] = Array(Edge(1108101562368,1297080123
92,(20694,47.4)), Edge(1108101562368,1297080123392,(20064,47.0)), Edge(1108101562368,1297080123
92,(11664,42.5)), Edge(1108101562368,1297080123392,(12819,42.45)), Edge(592705486848,1425929142
72,(13990,41.2)), Edge(1425929142272,1374389534720,(11791,40.2)), Edge(1425929142272,1374389534
20,(17135,39.5)), Edge(1425929142272,1374389534720,(10783,38.35)), Edge(1425929142272,137438953
720,(25430,38.35)), Edge(1108101562368,1297080123392,(8938,38.35)), Edge(1108101562368,12970801
3392,(8098,38.35)), Edge(1108101562368,1297080123392,(16389,38.2)), Edge(1108101562368,12970801
3392,(16914,38.2)), Edge(1108101562368,1297080123392,(9254,38.2)), Edge(1108101562368,129708012
392...

scala> sortedEdges.take(20).foreach(edge => println(s"Source: ${kv_rdd.lookup(edge.srcId)(0)};
estination: ${kv_rdd.lookup(edge.dstId)(0)}; Cost: ${edge.attr._1}; Duration:${edge.attr._2}"))
Source: Delhi; Destination: Cochin; Cost: 20694; Duration:47.4
Source: Delhi; Destination: Cochin; Cost: 20064; Duration:47.0
Source: Delhi; Destination: Cochin; Cost: 11664; Duration:42.5
Source: Delhi; Destination: Cochin; Cost: 12819; Duration:42.45
Source: Kolkata; Destination: Banglore; Cost: 13990; Duration:41.2
Source: Banglore; Destination: New Delhi; Cost: 11791; Duration:40.2
Source: Banglore; Destination: New Delhi; Cost: 17135; Duration:39.5
Source: Banglore; Destination: New Delhi; Cost: 10783; Duration:38.35
Source: Banglore; Destination: New Delhi; Cost: 25430; Duration:38.35
Source: Delhi; Destination: Cochin; Cost: 8938; Duration:38.35
Source: Delhi; Destination: Cochin; Cost: 8098; Duration:38.35
Source: Delhi; Destination: Cochin; Cost: 16389; Duration:38.2
Source: Delhi; Destination: Cochin; Cost: 16914; Duration:38.2
Source: Delhi; Destination: Cochin; Cost: 9254; Duration:38.2
Source: Delhi; Destination: Cochin; Cost: 11664; Duration:38.2
Source: Delhi; Destination: Cochin; Cost: 10441; Duration:38.15
Source: Delhi; Destination: Cochin; Cost: 8026; Duration:38.15
Source: Delhi; Destination: Cochin; Cost: 15864; Duration:38.15
Source: Delhi; Destination: Cochin; Cost: 10598; Duration:38.15
Source: Delhi; Destination: Cochin; Cost: 10493; Duration:38.15
```

- **Airport with the highest degree vertex**

```
scala> var pgrank = graph.pageRank(0.00001)
pgrank: org.apache.spark.graphx.Graph[Double,Double] = org.apache.spark.graphx.impl.GraphImpl@17
39a55b

scala> pgrank.vertices.sortBy(-_._2).collect()
res30: Array[(org.apache.spark.graphx.VertexId, Double)] = Array((1297080123392,1.51901467988681
), (1425929142272,1.338162570484757), (1108101562368,1.175099842690898), (1374389534720,1.002697
969420201), (592705486848,0.9623326551591355), (1425929142273,0.9623326551591334), (300647710720
,0.5201798135995329), (438086664192,0.5201798135995329))

scala> println(s"The Airport with highest PageRank is ${kv_rdd.lookup(pgrank.vertices.sortBy(-_.
_2).take(1)(0)._1)(0)} and has a PageRank value of ${pgrank.vertices.sortBy(-_._2).take(1)(0)._2
}")
The Airport with highest PageRank is Cochin and has a PageRank value of 1.51901467988681

scala> println("\n")



scala> pgrank.vertices.sortBy(-_._2).take(20).foreach(vertex => println(s"Airport: ${kv_rdd.look
up(vertex._1)(0)}; PageRank value: ${vertex._2}"))
Airport: Cochin; PageRank value: 1.51901467988681
Airport: Banglore; PageRank value: 1.338162570484757
Airport: Delhi; PageRank value: 1.175099842690898
Airport: New Delhi; PageRank value: 1.002697969420201
Airport: Kolkata; PageRank value: 0.9623326551591355
Airport: Hyderabad; PageRank value: 0.9623326551591334
Airport: Chennai; PageRank value: 0.5201798135995329
Airport: Mumbai; PageRank value: 0.5201798135995329
```

```
scala> println("Indegrees of each Vertex (Airport):")
Indegrees of each Vertex (Airport):

scala> graph.inDegrees.collect()
res23: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((1108101562368,1265), (12970801233
92,4537), (1374389534720,932), (1425929142272,2871), (1425929142273,697), (592705486848,381))

scala> println("\n")



scala> var inDegrees = graph.inDegrees.distinct().sortBy(-1*_._2)
inDegrees: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, Int)] = MapPartitionsRDD[
205] at sortBy at <console>:29

scala> println("Vertices (Airports) with highest Indegrees:")
Vertices (Airports) with highest Indegrees:

scala> inDegrees.take(100)
res26: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((1297080123392,4537), (14259291422
72,2871), (1108101562368,1265), (1374389534720,932), (1425929142273,697), (592705486848,381))

scala> println("\n")



scala> println(s"Airport with highest Indegree vertex: ${kv_rdd.lookup(inDegrees.take(1)(0)._1)(
0)} with an Indegree of ${inDegrees.take(1)(0)._2}")
Airport with highest Indegree vertex: Cochin with an Indegree of 4537

scala> println("\n")
```

- **List of most important airports according to PageRank: 5)**
  List of routes with the lowest flight costs:

```
scala> var sortedPrices = graph.edges.distinct().sortBy(edge => edge.attr._1)
sortedPrices: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[(Int, Double)]] = MapPartitionsRDD[361] at sortBy at <console>:29

scala> println("Routes with lowest cost:")
Routes with lowest cost:

scala> sortedPrices.take(100)
res36: Array[org.apache.spark.graphx.Edge[(Int, Double)]] = Array(Edge(438086664192,1425929142273,(1759,1.3)), Edge(438086664192,1425929142273,(1759,1.2)), Edge(438086664192,1425929142273,(1759,1.25)), Edge(438086664192,1425929142273,(1840,1.3)), Edge(438086664192,1425929142273,(1965,1.25)), Edge(438086664192,1425929142273,(1965,1.3)), Edge(438086664192,1425929142273,(2017,1.2)), Edge(438086664192,1425929142273,(2017,1.25)), Edge(438086664192,1425929142273,(2017,1.3)), Edge(438086664192,1425929142273,(2050,1.3)), Edge(438086664192,1425929142273,(2050,1.25)), Edge(438086664192,1425929142273,(2050,1.2)), Edge(438086664192,1425929142273,(2050,1.15)), Edge(438086664192,1425929142273,(2071,1.3)), Edge(438086664192,1425929142273,(2175,1.3)), Edge(438086664192,1425929...

scala> sortedPrices.take(20).foreach(edge => println(s"Source: ${kv_rdd.lookup(edge.srcId)(0)}; Destination: ${kv_rdd.lookup(edge.dstId)(0)}; Cost: ${edge.attr._1};Duration:${edge.attr._2}"))
Source: Mumbai; Destination: Hyderabad; Cost: 1759;Duration:1.3
Source: Mumbai; Destination: Hyderabad; Cost: 1759;Duration:1.2
Source: Mumbai; Destination: Hyderabad; Cost: 1759;Duration:1.25
Source: Mumbai; Destination: Hyderabad; Cost: 1840;Duration:1.3
Source: Mumbai; Destination: Hyderabad; Cost: 1965;Duration:1.25
Source: Mumbai; Destination: Hyderabad; Cost: 1965;Duration:1.3
Source: Mumbai; Destination: Hyderabad; Cost: 2017;Duration:1.2
Source: Mumbai; Destination: Hyderabad; Cost: 2017;Duration:1.25
Source: Mumbai; Destination: Hyderabad; Cost: 2017;Duration:1.3
Source: Mumbai; Destination: Hyderabad; Cost: 2050;Duration:1.3
Source: Mumbai; Destination: Hyderabad; Cost: 2050;Duration:1.25
Source: Mumbai; Destination: Hyderabad; Cost: 2050;Duration:1.2
Source: Mumbai; Destination: Hyderabad; Cost: 2050;Duration:1.15
Source: Mumbai; Destination: Hyderabad; Cost: 2071;Duration:1.3
Source: Mumbai; Destination: Hyderabad; Cost: 2175;Duration:1.3
Source: Mumbai; Destination: Hyderabad; Cost: 2175;Duration:1.25
Source: Mumbai; Destination: Hyderabad; Cost: 2175;Duration:1.2
Source: Mumbai; Destination: Hyderabad; Cost: 2227;Duration:1.35
Source: Mumbai; Destination: Hyderabad; Cost: 2227;Duration:1.3
Source: Mumbai; Destination: Hyderabad; Cost: 2227;Duration:1.25
```

CONCLUSION :

Thus, I have studied Spark GraphX and used it to perform flight data analysis.