GraphFrame Implementation using Java

GraphFrame is nothing but consist of two DataFrame. It's vertex and edge represent DataFrame.

A vertex DataFrame should contain a unique Id or name which will help to recognize the vertex while creating edges.

A edge DataFrame contains 3 fields : source, destination and relationship between them. Source and destination are mainly unique property(Name/ID) of two different rows.

For GraphFrame details, please refer http://graphframes.github.io/user-guide.html

spark path:

```
SparkConf conf = new SparkConf().setAppName("test").setMaster("local");

JavaSparkContext sc = new JavaSparkContext(conf);
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);
```

Vertex Row creation:

```
JavaRDD<Row> verRow =
      sc.parallelize(Arrays.asList(RowFactory.create(101L,"Trina",27),
                        RowFactory.create(201L,"Raman",45),
                        RowFactory.create(301L,"Ajay",32),
                        RowFactory.create(401L,"Sima",23)));
```

Creating column and declaring dataType for vertex:

```
List<StructField> verFields = new ArrayList<StructField>();

verFields.add(DataTypes.createStructField("id",DataTypes.LongType, true));
verFields.add(DataTypes.createStructField("name",DataTypes.StringType, true));
verFields.add(DataTypes.createStructField("age",DataTypes.IntegerType, true));
```

Edge row Creation:

```
JavaRDD<Row> edgRow = sc.parallelize(Arrays.asList(
                        RowFactory.create(101L,301L,"Colleague"),
                        RowFactory.create(101L,401L,"Friends"),
                        RowFactory.create(401L,201L,"Reports"),
                        RowFactory.create(301L,201L,"Reports"),
                        RowFactory.create(201L,101L,"Reports")));
```

Edge column Creation with dataType:

```
List<StructField> EdgFields = new ArrayList<StructField>();
            EdgFields.add(DataTypes.createStructField("src",DataTypes.LongType,
true));
            EdgFields.add(DataTypes.createStructField("dst",DataTypes.LongType,
true));


EdgFields.add(DataTypes.createStructField("relationType",DataTypes.StringType,
true));
```

Creating Schema:

```
StructType verSchema = DataTypes.createStructType(verFields);
StructType edgSchema = DataTypes.createStructType(EdgFields);
```

Creating vertex DataFrame and edge DataFrame:

```
DataFrame verDF = sqlContext.createDataFrame(verRow, verSchema);
DataFrame edgDF = sqlContext.createDataFrame(edgRow, edgSchema);
```

Creating a GraphFrame:

```
GraphFrame g = new GraphFrame(verDF,edgDF);
```

Basic GraphFrame and DataFrame queries:

```
g.vertices().show();
```

Result:

```
+---+-----+---+
| id| name|age|
+---+-----+---+
|101|Trina| 27|
|201|Raman| 45|
|301| Ajay| 32|
|401| Sima| 23|
+---+-----+---+
```

```
g.edges().show();
```

Result:

```
+---+---+------------+
|src|dst|relationType|
+---+---+------------+
|101|301|   Colleague|
|101|401|     Friends|
|401|201|     Reports|
|301|201|     Reports|
|201|101|     Reports|
+---+---+------------+
```

- ***Get a DataFrame with columns "id" and "inDeg" (in-degree)***

```
g.inDegrees().show();
```

```
+---+--------+
| id|inDegree|
+---+--------+
|301|       1|
|101|       1|
|401|       1|
|201|       2|
+---+--------+
```

- ***Find the youngest user's age in the graph by querying vertex DataFrame***

```
g.vertices().groupBy().min("age").show();
```

```
+--------+
|min(age)|
+--------+
|      23|
+--------+
```

- ***Find the number of "Friends" relationship in the graph by querying edge DataFrame***

```
long numFriends = g.edges().filter("relationType = 'Friends'").count();
System.out.println("Print total count of Friends relationship :: "+
numFriends);
```

Result:

```
Print total count of Friends relationship :: 1
```

Motif finding

Motif finding refers to searching for structural patterns in a graph.Please

refer GraphFrame docs for details.

```
DataFrame motifs = g.find("(a)-[e]->(b)");
motifs.filter("b.age>40").show();
```

Result:

```
+-------------+-----------------+-------------+
|            a|                e|            b|
+-------------+-----------------+-------------+
|[301,Ajay,32]|[301,201,Reports]|[201,Raman,45]|
|[401,Sima,23]|[401,201,Reports]|[201,Raman,45]|
+-------------+-----------------+-------------+
```

Subgraphs:

subgraph creation based on vertex and edge filters

- **Simple subgraph:**

```
DataFrame v2 = g.vertices().filter("age > 30");
DataFrame e2 = g.edges().filter("relationType = 'Reports'");
GraphFrame g2 = new GraphFrame(v2,e2);
g2.vertices().show();
```

Result:

```
+---+-----+---+
| id| name|age|
+---+-----+---+
|201|Raman| 45|
|301| Ajay| 32|
+---+-----+---+
```

```
g2.edges().show();
```

Result:

```
+---+---+------------+
|src|dst|relationType|
+---+---+------------+
|401|201|     Reports|
|301|201|     Reports|
|201|101|     Reports|
+---+---+------------+
```

- **Complex subgraph: triplet filters:**

subgraph creation based upon triplet filters which operate on an edge and its src and dst vertices

```
DataFrame paths = g.find("(a)-[e]->(b)")
              .filter("e.relationType = 'Reports'")
              .filter("a.age < b.age");

DataFrame  e2 = paths.select("e.src", "e.dst", "e.relationType");
GraphFrame g2 = new GraphFrame(g.vertices(),e2);
g2.vertices().show();
```

Result:

```
+---+-----+---+
| id| name|age|
+---+-----+---+
|101|Trina| 27|
|201|Raman| 45|
|301| Ajay| 32|
|401| Sima| 23|
+---+-----+---+
```

```
g2.edges().show();
```

Result:

```
+---+---+------------+
|src|dst|relationType|
+---+---+------------+
|301|201|     Reports|
|401|201|     Reports|
+---+---+------------+
```

Breadth-first search (BFS):

Breadth-first search (BFS) finds the shortest path(s) from one vertex (or a set of vertices) to another vertex (or a set of vertices).

```
//Search from "Esther" for users of age >27.
DataFrame paths = g.bfs().fromExpr("name = 'Trina'").toExpr("age > 27").run();

paths.show();
```

Result:

```
+--------------+------------------+-------------+
|          from|                e0|           to|
+--------------+------------------+-------------+
|[101,Trina,27]|[101,301,Colleague]|[301,Ajay,32]|
+--------------+------------------+-------------+
```

```
DataFrame paths = g.bfs().fromExpr("name = 'Trina'").toExpr("age > 30")
              .edgeFilter("relationType != 'Colleague'")
              .maxPathLength(3)
              .run(); // Specify edge filters or max path lengths.

paths.show();
```

Result:

```
+--------------+-----------------+-------------+-----------------+--------------+
|          from|               e0|           v1|               e1|            to|
+--------------+-----------------+-------------+-----------------+--------------+
|[101,Trina,27]|[101,401,Friends]|[401,Sima,23]|[401,201,Reports]|[201,Raman,45]|
+--------------+-----------------+-------------+-----------------+--------------+
```

Connected components:

Computes the connected component membership of each vertex and returns a graph with each vertex assigned a component ID.

```
 DataFrame result = g.connectedComponents().run();
result.select("id", "component").orderBy("component").show();
```

Result:

```
+---+---------+
| id|component|
+---+---------+
|101|      101|
|301|      101|
|201|      101|
|401|      101|
+---+---------+
```

Label Propagation Algorithm (LPA):

```
DataFrame result =  g.labelPropagation().maxIter(5).run();
result.select("id", "label").show();
```

Result:

```
+---+-----+
| id|label|
+---+-----+
|101|  301|
|301|  101|
|201|  301|
|401|  101|
+---+-----+
```

PageRank:

```
GraphFrame results = g.pageRank().resetProbability(0.15).tol(0.01).run();
      // Display resulting pageranks and final edge weights

results.vertices().select("id", "pagerank").show();
```

Result:

```
+---+------------------+
| id|          pagerank|
+---+------------------+
|101|1.1879011205028784|
|301|0.6548579762137234|
|201|1.2210601417680924|
|401|0.6548579762137234|
+---+------------------+
```

```
results.edges().select("src", "dst", "weight").show();
```

Result:

```
+---+---+------+
|src|dst|weight|
+---+---+------+
|101|301|   0.5|
|401|201|   1.0|
|101|401|   0.5|
|201|101|   1.0|
|301|201|   1.0|
+---+---+------+
```

Shortest Paths:

Computes shortest paths from each vertex to the given set of landmark vertices, where landmarks are specified by vertex ID.

```
ArrayList<Object> l1st = new ArrayList<Object>();
l1st.add(101L);
l1st.add(401L);
DataFrame results = g.shortestPaths().landmarks(l1st).run();
results.select("id", "distances").show();
```

Result:

```
+---+--------------------+
| id|           distances|
+---+--------------------+
|101|Map(101 -> 0, 401...|
|301|Map(101 -> 2, 401...|
|201|Map(101 -> 1, 401...|
|401|Map(401 -> 0, 101...|
+---+--------------------+
```

Triangle count:

Computes the number of triangles passing through each vertex.

```
DataFrame results = g.triangleCount().run();
results.select("id", "count").show();
```

Result:

```
+---+-----+
| id|count|
+---+-----+
|101|    2|
|301|    1|
|201|    2|
|401|    1|
+---+-----+
```

Saving and loading GraphFrames:

```
// Save vertices and edges as Parquet to some location.
g.vertices().write().parquet("hdfs://myLocation/vertices");
g.edges().write().parquet("hdfs://myLocation/edges");

// Load the vertices and edges back.
DataFrame sameV = sqlContext.read().parquet("hdfs://myLocation/vertices");
DataFrame sameE = sqlContext.read().parquet("hdfs://myLocation/edges");

// Create an identical GraphFrame.
GraphFrame sameG =new GraphFrame(sameV, sameE);
```

GraphFrame to GraphX and GraphX to GraphFrame conversion:

```
// Convert to GraphX
Graph<Row, Row> graphX = g.toGraphX();
```