Lab: Graph Algorithms with GraphX

1 Setting up GraphX and First Steps

- Similairly to the previous labs, GraphX is included by default in any Spark installation. If you followed the instructions in the previous lab on Spark, you should have the environment ready. If this is not the case, follow the instructions in the lab of Week 1, available at https://www.lri.fr/~groz/documents/m1ai-2021/exercices.html
- Read the GraphX programming guide https://spark.apache.org/docs/latest/graphx-programming-guide.html

> There is no Python version of GraphX. The closest implementation are GraphFrames http://graphframes.github.io/graphframes/docs/_site/, but they are not part of the main Spark distribution when this lab was written. In this lab, we will work with the default Scala implementation of GraphX accesible on the container using the `spark-shell` command.

2 A Quick Tour Of GraphX

> This example roughly follows the flow of the (defunct) hands-on lab at https://web.archive.org/web/20200627072401/http://ampcamp.berkeley.edu/big-data-mini-course/graph-analytics-with-graphx.html

- Generate a small example graph and transform it into an RDD.

```
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD

val vertexArray = Array(
        (1L, ("Alice", 28)),
        (2L, ("Bob", 27)),
        (3L, ("Charlie", 65)),
        (4L, ("David", 42)),
        (5L, ("Ed", 55)),
        (6L, ("Fran", 50))
)
val edgeArray = Array(
        Edge(2L, 1L, 7),
        Edge(2L, 4L, 2),
        Edge(3L, 2L, 4),
        Edge(3L, 6L, 3),
        Edge(4L, 1L, 1),
        Edge(5L, 2L, 2),
        Edge(5L, 3L, 8),
        Edge(5L, 6L, 3)
)

val vertexRDD: RDD[(Long, (String, Int))] = sc.parallelize(vertexArray)
val edgeRDD: RDD[Edge[Int]] = sc.parallelize(edgeArray)

val graph: Graph[(String, Int), Int] = Graph(vertexRDD, edgeRDD)
```

- Show the number of nodes, edges.

```
println(graph.numEdges)
println(graph.numVertices)
```

- Use `aggregateMessages` to compute the oldest follower for each node.

```
val oldestFollowers: VertexRDD[(String,Int)] =
        graph.aggregateMessages[(String,Int)](
                triplet => triplet.sendToDst(triplet.srcAttr),
                (a, b) => if (a._2 > b._2) a else b
        )

oldestFollowers.collect()
```

- Use the `pregel` operator to show, for each node, the age of the oldest person that can reach it.

```
val oldReachable = graph.pregel(0)(
        //Vertex Program: takes the oldest age available
        (id, attr, newAge) => (attr._1, math.max(attr._2, newAge)),
        //Vertices only send messages if their age (second attrribute)
        //  is greater than the destination vertex age
        triplet => {
                if (triplet.srcAttr._2 > triplet.dstAttr._2) {
                        Iterator((triplet.dstId, triplet.srcAttr._2))
                } else {
                        Iterator.empty
                }
        },
        //For the same destination id, only the max age is kept
        // (to be used in the Vertex Program)
        (a, b) => math.max(a, b) // Merge Message
)

oldReachable.vertices.collect()
```

- Compute PageRank values, two ways (until convergence and for a static number of iterations). Compare

```
//the first method computes until convergence, when
// the values between two iterations do not change much
val pr = graph.pageRank(0.0001).vertices
pr.collect()

//the second method is the static method, which simply iterates
// for a given number of steps
val pr = graph.staticPageRank(100).vertices
pr.collect()
```

3 Pregel in GraphX

In this part, the objective is to compare the implementations of several algorithms in GraphX and by using Pregel.

Your task is to load a large graph and then use the Pregel operator to compute two graph tasks: PageRank and single-source shortest distances.

> Large graphs shown as a list of files can be loaded using the `GraphLoader.edgeListFile` operator. Large graphs can be found at http://networkrepository.com/

- Compute PageRank values using the `pageRank` operator (exemplified before) and using the `pregel` operator (examples on how this can be done are in the slides). Compare the two results.
- Compute *single-source* shortest path values using Pregel.

> Here you will have to do a pre-processing step: set to 0 the distance from the source vertex, and as ∞ the other distances in the graph. Use `mapVertices` for this.