UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

SEMANTIC DATA MANAGEMENT

Distributed Graph Processing

April 2023

Ximena Moure
Fatima Zohra Chriki

Exercise 1: Getting familiar with GraphX's Pregel API

Superstep 0

In Vprog:

All vertices are active and each node runs the Vprog function. They all receive as an initial message the MAX_VALUE integer. The MAX_VALUE does not trigger any change to the vertex value, and all nodes return their attribute value.

In sendMsg:

- Vertex1 sends a message to Vertex 2 with the value 9. This is because the source vertex value (9) is greater than the destination vertex value (1).
- Vertex2 does not send any message because its value (1) is smaller than the destination vertices which are Vertex3 (with value 6) and Vertex4 (with value 8).
- Vertex3 does not send any message because its value (6) is smaller than the destination vertices which are Vertex1 (with value 9) and Vertex4 (with value 8).
- Vertex4 does not send any message because it does not have any edge going out of it.

Superstep 1

In VProg:

Only node 2 is active and runs VProg. It received a message with value 9 which is greater than its current value 1, so the vertex value is updated to 9.

In sendMsg:

- Vertex1 does not send a message because now Vertex2 has a value of 9 which is not greater than its own value (9).
- Vertex2 sends two messages:
  - One to Vertex3 because Vertex2 value (9) is greater than Vertex3 value (6)
  - One to Vertex4 because Vertex2 value is greater than Vertex4 value (8)
- Vertex3 does not send any message because its value (6) is smaller than the destination vertices which are Vertex1 (with value 9) and Vertex4 (with value 8).
- Vertex4 does not send any message because it does not have any edge going out of it.

Superstep 2

In VProg:

VProg is run for vertices 3 and 4, which are the ones that received messages in the previous superstep. As they received the value 9 and it is bigger than their current value, 6 and 8 respectively, they change their value to 9.

In SendMsg:

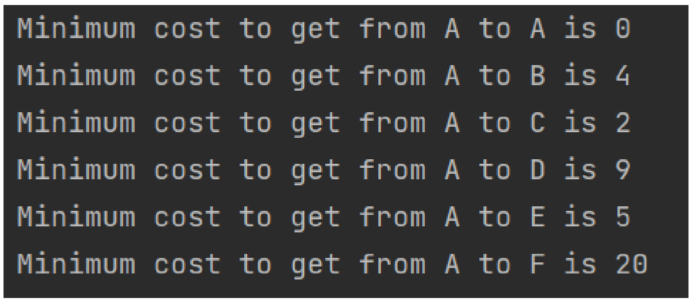
- Vertex1 does not send a message because Vertex2 has a value of 9 which is not greater than its own value (9).
- Vertex2 does not send a message because Vertex3 and Vertex4 now have a value of 9 which is not greater than its own value (9).
- Vertex3 does not send a message because Vertex1 and Vertex4 have a value of 9 which is not greater than its own value (9).
- Vertex4 does not send any message because it does not have any edge going out of it.

Since in this superstep no vertex can send a message, then no vertex was activated. Therefore, this superstep ends the execution.

Exercise 2: Computing shortest paths using Pregel

In this exercise was solved using the previous exercise as baseline, applying the following changes to the code:

- **Vertices:** All the vertex attributes were set as MAX_VALUE, except for node 1, which was set as 0.
- **VProg**: returns the minimum between the vertex value and the message if the message is different from MAX_VALUE. Otherwise, it just returns the vertex value. So, the node can select a value that is less than its current value.
- **SendMsg**: we sent a message:
    - if the value in the node is not MAX_VALUE and
    - if the source node value plus the edge weight is lower than the value of the destination node: `sourceVertex._2 + weight < dstVertex._2`
- **Merge**: if a node receives more than one message, it just needs to keep the lower value, so we return the minimum value.



```
Minimum cost to get from A to A is 0
Minimum cost to get from A to B is 4
Minimum cost to get from A to C is 2
Minimum cost to get from A to D is 9
Minimum cost to get from A to E is 5
Minimum cost to get from A to F is 20
```

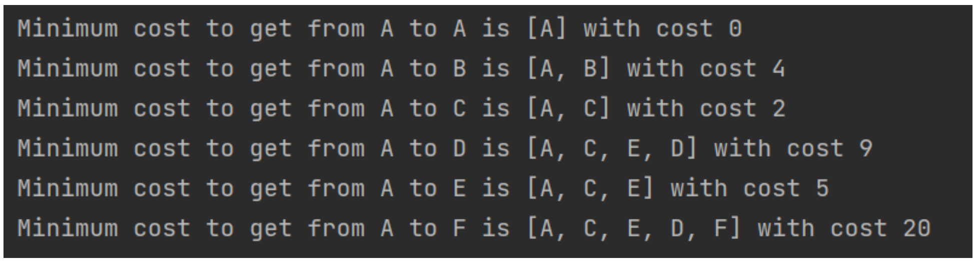
Figure1: Exercice 2 output

Exercise 3: Extending shortest path's computation

The *VProg* and *Merge* functions of this exercise are very similar to the functions of the previous one, the main changes we had to make was modifying the vertex data type by adding a list in order to store the minimum path of each vertex and the *SendMsg* function:

- **Vertices**: we changed the vertex attribute to Tuple2<Integer, List<Long>>. Therefore, we will be able to store as a list the nodes ID which belong to the minimum path.

- **SendMsg**: if the source node value plus the edge weight is lower than the value of the destination node, we send a message adding the source vertex ID in the list of nodes of minimum path of the source node.



```
Minimum cost to get from A to A is [A] with cost 0
Minimum cost to get from A to B is [A, B] with cost 4
Minimum cost to get from A to C is [A, C] with cost 2
Minimum cost to get from A to D is [A, C, E, D] with cost 9
Minimum cost to get from A to E is [A, C, E] with cost 5
Minimum cost to get from A to F is [A, C, E, D, F] with cost 20
```

Figure2: Exercise 3 output

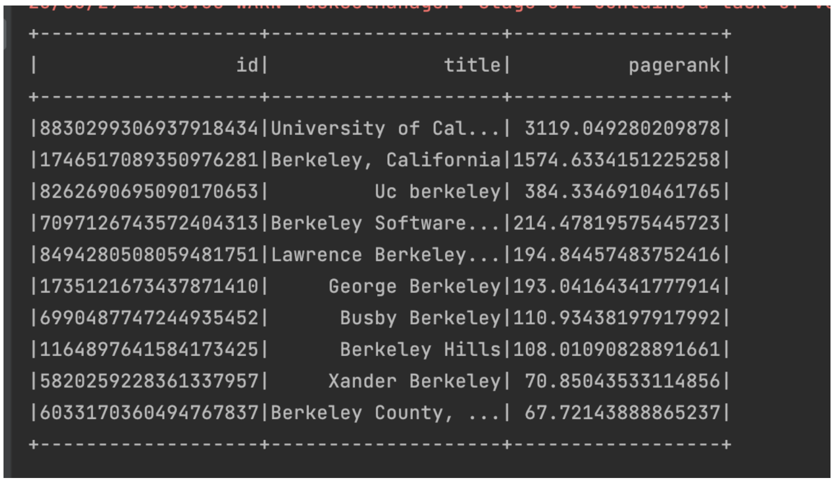
Exercise 4: Spark Graph Frames

We first read and load the vertices and then we read and load the edges of the graph.
For it to be more readable we created two separate functions: *readAndLoadVertices* and *readAndLoadEdges*.

Both of them use a Scanner from the java.util library to read the .txt files. Then we iterate over each scanner until it has no more lines to read. While we iterate we get the data from the files and we add them to a list, vertices_list for the vertices and a edges_list for the edges. After, we followed the warmup example and we defined the schema and created the data frame.

Then, in the wikipedia function we create the graph with the vertices and edges returned form the two previous functions.
Finally, we call the pageRank algorithm. For this, we defined two global variables in order to change the damping factor and the maximum number of iterations.

When running the algorithm with the default parameters, which are resetProbability of 0.15 which is a dampingFactor of 0.85 and maxIter of 10, we get the following result.
We tried increasing the max number of iterations but we concluded that the best number for it is 10.



```
+-------------------+--------------------+------------------+
|                 id|               title|          pagerank|
+-------------------+--------------------+------------------+
|8830299306937918434|University of Cal...| 3119.049280209878|
|1746517089350976281|Berkeley, California|1574.6334151225258|
|8262690695090170653|         Uc berkeley| 384.3346910461765|
|7097126743572404313|Berkeley Software...|214.47819575445723|
|8494280508059481751|Lawrence Berkeley...|194.84457483752416|
|1735121673437871410|     George Berkeley|193.04164341777914|
|6990487747244935452|      Busby Berkeley|110.93438197917992|
|1164897641584173425|      Berkeley Hills|108.01090828891661|
|5820259228361337957|     Xander Berkeley| 70.85043533114856|
|6033170360494767837|Berkeley County, ...| 67.72143888865237|
+-------------------+--------------------+------------------+
```

Figure3: PageRank results with dampingFactor of 0.85 and maxIter 10