UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Semantic Data Management

Lab: Distributed Graph Processing

Group Members:

- Khushnur Binte Jahangir
- Niccolò Morabito
- Zyrako Musaj

Exercise 1: Getting familiar with GraphX's Pregel API

For this exercise, the explanation of each super step that will be performed for the provided graph is given below:

Superstep 0:
The algorithm starts with all active vertices (vertex1, vertex2, vertex3, and vertex4) and all of them receive the initial message with the int max value. In VProg, the vertex value of any vertex is not affected because the initial message is equal to int max and it is not less than any vertex value.

```
At Super step-0: Vertex Value=6 and Message=2147483647
At Super step-0: Vertex Value=9 and Message=2147483647
At Super step-0: Vertex Value=1 and Message=2147483647
At Super step-0: Vertex Value=8 and Message=2147483647
```

Figure 1. Vertices initial value and message at Super step-0

In sendMsg, all outbound edges (since actDir is set to EdgeDirection.Out()) of all active vertices are activated. For example, vertex1 sends a message containing its value 9 to vertex2 and since the source vertex value is greater than the destination vertex value (which is 1), vertex2 will be ready to propagate the message to its neighbouring vertices in next super step. Vertex3 and vertex4 will not generate any message to propagate since they do not meet the condition.

```
In sendMsg: Vertex-2 to Vertex-3
do nothing
In sendMsg: Vertex-3 to Vertex-4
do nothing
In sendMsg: Vertex-3 to Vertex-1
do nothing
```

Figure 2. In sendMsg, vertex 3 and vertex 4 will not generate any message

Superstep 1:
Now, vertex2 is active since it only received messages from the previous step. In VProg, vertex2 will update its value to 9 since it is the maximum value between vertex value and message.
In sendMsg, vertex2 sends messages to vertex3 and vertex4 with value 9 and they will be ready to propagate messages to the next super step.

```
In sendMsg: Vertex-2 Propagates value 9 to Vertex-3
In sendMsg: Vertex-2 Propagates value 9 to Vertex-4
```

Figure 3. Vertex 2 send messages to vertices 3 and 4

Superstep 2:
VProg will be called on vertex3 and vertex4 since these two vertices received messages during the previous superstep. Vertex3 will update its vertex value to 9 since its current vertex value is smaller than 9; equally, vertex4 will update its value to 9.
In sendMsg, vertex4 will not generate any message because it does not have any outbound edge. Vertex3 will propagate a message to vertex1 and vertex2 but since it meets the condition of source vertex value equals to destination vertex, vertex1 and vertex2 will not generate any message.

```
In sendMsg: Vertex-3 to Vertex-1
do nothing
In sendMsg: Vertex-3 to Vertex-4
do nothing
```

Figure 4. Vertex 3 propagate message to vertices 1 and 2

Since there is no active vertex left to generate a message, the API will return the maximum vertex value in the graph which is 9.

```
9 is the maximum value in the graph
```

Figure 5. The final output with the maximum value

Exercise 2: Computing shortest paths using Pregel

VProg - returns the minimum between the message received and the actual value (distance) of the node
sendMsg - is the function used for sending messages from one node to the other. The algorithm iterates in a loop, where each iteration is called super step. All nodes start the algorithm as active (the first source node A of value 0, and other nodes of value *Integer.MAX_VALUE*) and the loop ends when all nodes are halted, which means that no new messages are generated. All nodes start the algorithm as active and then can be halted if it generates no messages during one of the steps.
The function checks if the sum of the value of the source (current) node and the edge is less than the value of the destination node. If that is true, a new message is generated across the edge. The message is a tuple of the target node to send the message to and the new distance value. If the condition evaluates to false, there will be no new messages generated from the current node and the node will halt for the remaining iterations unless being activated by another message sent to it later.
Merge - is the function that takes care of situations when a node receives more than one message (has more than one incoming edge). If that is the case, it chooses the message with the minimum value (shortest distance).

```
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
```

Figure 6. The output of the second exercise

Exercise 3: Extending shortest path's computation

For this exercise, we have the same functions as mentioned in the last exercise, but we change the data structure of the vertex from *Tuple2<Object, Integer>* to *Tuple2<Object, Tuple2<Integer, String>>*, so we can be able to hold on it not only the distance, but also the path. So, when sending messages from one node to the other, not only we send the minimum distance to that node, but also the labels of the vertices of that path.

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

Figure 7. The output of the third exercise

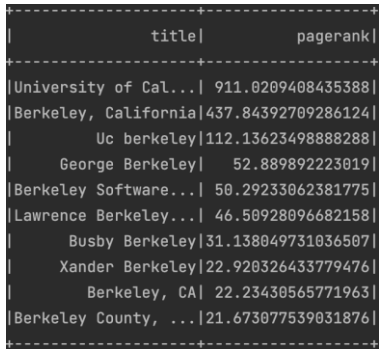
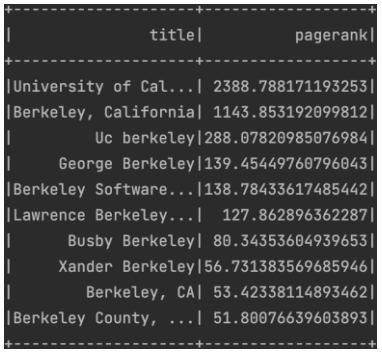
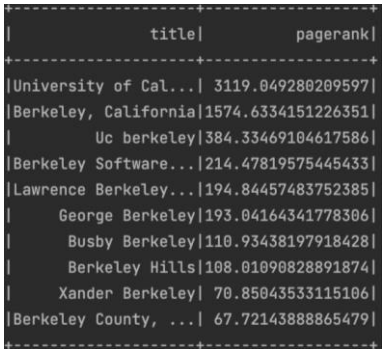
Exercise 4: Spark Graph Frames

For this exercise, in addition to load the vertices and the edges from the txt files and to run the PageRank to get the most important pages in the Wikipedia graph, it is also required to parametrize the set the PageRank's parameters, i.e. the maximum number of iterations and the damping factor.

The former intuitively stops the iterative algorithm that, at each iteration, spread the PR one hop. Different values have been tested from 10 until 200, where the computation starts to be too slow to wait for it, but the results are affected only marginally: the ranking remains the same (i.e. the top 10 pages do not change for any tested value) and the pagerank values have only very small or imperceptible changes (2-3%). So, in the end, the right choice for the maximum number of iterations seems to be 10, the smallest value.

+--------------------+------------------+ |title| pagerank| +--------------------+------------------+ |University of Cal...| 3119.049280209597| |Berkeley, California|1574.6334151226351| | Uc berkeley|384.33469104617586| |Berkeley Software...|214.47819575445433| |Lawrence Berkeley...|194.84457483752385| | George Berkeley|193.04164341778306| | Busby Berkeley|110.93438197918428| | Berkeley Hills|108.01090828891874| | Xander Berkeley| 70.85043533115106| |Berkeley County, ...| 67.72143888865479| +--------------------+------------------+	+--------------------+------------------+ |title| pagerank| +--------------------+------------------+ |University of Cal...|3124.2641714866018| |Berkeley, California| 1572.415100533493| | Uc berkeley| 384.251446739748| |Berkeley Software...|214.05725517574425| |Lawrence Berkeley...|193.68997507338526| | George Berkeley| 193.6716555575499| | Busby Berkeley|113.25606403048869| | Berkeley Hills|105.89779003611068| | Xander Berkeley| 71.85250422942285| |Berkeley County, ...| 68.47601926652493| +--------------------+------------------+
10 most relevant pages with 10 iterations	10 most relevant pages with 200 iterations

The latter, instead, affects more visibly the pagerank values and also part of the ranking. The damping factor is the probability that the imaginary surfer who is randomly clicking on links will continue clicking. The selection of the damping ratio requires a tradeoff between decreasing the maximum percent overshoot and decreasing the time where the peak overshoot occurs, tp. The original paper set it to 0.85, and actually increasing by 0.05 changes the order of the ranking's last positions. We tried a value of 0.25, just to see how it would affect the result. The output shows that we have a slightly different rank.

 +--------------------+------------------+ |title| pagerank| +--------------------+------------------+ |University of Cal...| 911.0209408435388| |Berkeley, California|437.84392709286124| | Uc berkeley|112.13623498888288| | George Berkeley| 52.889892223019| |Berkeley Software...| 50.29233062381775| |Lawrence Berkeley...| 46.50928096682158| | Busby Berkeley|31.138049731036507| | Xander Berkeley|22.920326433779476| | Berkeley, CA| 22.23430565771963| |Berkeley County, ...|21.673077539031876| +--------------------+------------------+	 +--------------------+------------------+ |title| pagerank| +--------------------+------------------+ |University of Cal...| 2388.788171193253| |Berkeley, California| 1143.853192099812| | Uc berkeley|288.07820985076984| | George Berkeley|139.45449760796043| |Berkeley Software...|138.78433617485442| |Lawrence Berkeley...| 127.862896362287| | Busby Berkeley| 80.34353604939653| | Xander Berkeley|56.731383569685946| | Berkeley, CA| 53.42338114893462| |Berkeley County, ...| 51.80076639603893| +--------------------+------------------+	 +--------------------+------------------+ |title| pagerank| +--------------------+------------------+ |University of Cal...| 3119.049280209597| |Berkeley, California|1574.6334151226351| | Uc berkeley|384.33469104617586| |Berkeley Software...|214.47819575445433| |Lawrence Berkeley...|194.84457483752385| | George Berkeley|193.04164341778306| | Busby Berkeley|110.93438197918428| | Berkeley Hills|108.01090828891874| | Xander Berkeley| 70.85043533115106| |Berkeley County, ...| 67.72143888865479| +--------------------+------------------+
10 most relevant papers with dampingFactor=0.15	10 most relevant papers with dampingFactor=0.5	10 most relevant papers with dampingFactor=0.85