Semantic Data Management
Distributed Graph Processing

Mohana Fathollahi and Carol Jhasmin Azparrent Estela

April 28, 2022

1 Exercise 1:

- The initial state of each node *(I.S.)* and the messages that it receives *(Message)*.
  All the nodes initialize with their value that has defined in graph and then they will receive max_value message.

node id	Superstep 0		Superstep 1		Superstep 2		Superstep 3	
	I.S.	Message	I.S.	Message	I.S.	Message	I.S.	Message
1	9	Max_Value	9	6	9	no	9	9
2	1	Max_Value	1	9	9	no	9	no
3	6	Max_Value	6	1	6	9	9	no
4	8	Max_Value	8	1; 6	8	9	9	9

- The result of calculations performed on each node *(Perform.)* and its resulting state *(R.S.)*.
  In each node we should select maximum value for messages that have been sent to this node, as a merge function.

node id	Superstep 0		Superstep 1		Superstep 2		Superstep 3	
	Perform.	R.S.	Perform.	R.S.	Perform.	R.S.	Perform.	R.S.
1	-	9	6 <9	9	-	9	9==9	9
2	-	1	9 >1	9	-	9	-	9
3	-	6	1 <6	6	9 >6	9	-	9
4	-	8	1 <8; 6 <8	8	9 >8	9	9==9	9

- The message(s) to be sent (if any) to other nodes in the next superstep. *(To be sent)*
  For this part we should use kernel function for selectMsg, in this function we access to source and destination node and we can compare their values together. If destination node has smaller value compare to source node, we can send a massage but if destination node has larger value than source node, sending message do not change the states of destination, therefore we will not send a message to destination and save one superstep.

node id	Superstep 0	Superstep 1	Superstep 2	Superstep 3
	To be sent	To be sent	To be sent	To be sent
1	9	no	no	no
2	1; 1	9; 9	no	no
3	6; 6	no	9; 9	no
4	no	no	no	no

2 Exercise 2:

The goal of this part is finding the shortest path from node 1 to other nodes, therefore we need to make some changes in kernel functions. The first node initialized by zero value and other nodes by max_value.

The first change is related to VProg, a node can just choose a value that is less than its current value:

```
        if (message == Integer.MAX_VALUE) {
        return vertexValue;
        } else {
        return Math.min(vertexValue, message);
        }
```

To optimize procedure, we should revise condition in sendMsg for sending a message from a node, so that message should send from nodes that their value is not max_value and message sent to destination should be less than of value for destination node. otherwise message should not be sent.

```
if (sourceVertex._2 != Integer.MAX_VALUE && sourceVertex._2 + triplet.attr() < dstVertex._2){
    return JavaConverters.asScalaIteratorConverter(Arrays.asList
    (new Tuple2<Object, Integer>(triplet.dstId(), sourceVertex._2 + triplet.attr()))
    .iterator()).asScala();
}else {
    return JavaConverters.asScalaIteratorConverter
    (new ArrayList<Tuple2<Object, Integer>>().iterator()).asScala();
}
```

A node can recieve some messages form other nodes, therefore we should select minimum value of them. We applied min in merge function like below:

```
        public Integer apply(Integer o, Integer o2) {
            return Math.min(o, o2);
```

3 Exercise 3:

The main structure of this exercise is same as exercise 2, but we should consider the node that sent the minimum value to the next node and its value. Therefore functions can change like below:

- VProg:

```
    if (message._2 == Integer.MAX_VALUE) {
            return vertexValue;
        } else {
            return new Tuple2(message._1, Math.min(vertexValue._2, message._2));
        }
```

- sendMsg:

```
if ((Integer) sourceVertex._2._2 != Integer.MAX_VALUE && (Integer)
sourceVertex._2._2 + triplet.attr() < (Integer) destinationVertex._2._2) {
        Tuple2 msg = new Tuple2(sourceVertex._2._1 + ","+
        labels.get(destinationVertex._1),
        (Integer) sourceVertex._2._2 + triplet.attr());
        return JavaConverters.asScalaIteratorConverter(Arrays.asList
            (new Tuple2<Object, Tuple2>(triplet.dstId(), msg))
            .iterator()).asScala();
} else {
        return JavaConverters.asScalaIteratorConverter
        (new ArrayList<Tuple2<Object, Tuple2>>().iterator()).asScala();
```

- merge:

```
        Tuple2<String, Integer> result;
            if (o._2 <= o2._2) {
                result = o;
            }else{
                result = o2;
                }
            return result;
        }
```

4 Exercise 4:

In order to do this exercise, we have configured the warmup exercise. First, we added the paths for reading the txt files.

```
    String edgesPath =  "src/main/resources/wiki-edges.txt";
    String verticesPath = "src/main/resources/wiki-vertices.txt";
```

Secondly, to upload the data from the files, we created a `for` iteration for the vertices creation and edges creation as we must need to load the graph with this data.

```
//Vertices creation
for (String line = brver.readLine(); line != null; line = brver.readLine()) {
    String[] item = line.split("\t");
    Row rowvertice = RowFactory.create(item[0], item[1]); //['a', 'b']
    vertices_list.add(rowvertice);
}
```

```
//Edges creation
for (String line = bredg.readLine(); line != null; line = bredg.readLine()) {
        String[] item = line.split("\t");
        Row rowedge = RowFactory.create(item[0], item[1]); //['a', 'b']
        edges_list.add(rowedge);
}
```

Also, we applied the `GraphFrame` for the vertices and edges. Finally, in this exercise, PageRank algorithm is requiremented so we implemented with a damping factor equal to 0.85, as a consequence, a resetProbability equal to 0.15 with a maximum number of iterations equivalent ten.

```
// PageRank: d = 0.85 -> resetProbability = 0.15
    PageRank rank = gf.pageRank();
    GraphFrame results = rank.resetProbability(0.15).maxIter(10).run();
    Dataset<Row> pageRankNodes =  results.vertices().orderBy(functions.desc("pagerank"));
```

The figure 1 shows the result of this exercise exhibiting the top 10 most relevant articles from the Wikipedia dataset.

```
+------------------+------------------+------------------+
|                id|              name|          pagerank|
+------------------+------------------+------------------+
|8830299306937918434|University of Cal...|3119.0492802098793|
|1746517089350976281|Berkeley, California|1574.6334151225265|
|8262690695090170653|         Uc berkeley| 384.3346910461766|
|7097126743572404313|Berkeley Software...|214.47819575445732|
|8494280508059481751|Lawrence Berkeley...|194.84457483752425|
|1735121673437871410|     George Berkeley|193.04164341777923|
|6990487747244935452|      Busby Berkeley|110.93438197917996|
|1164897641584173425|      Berkeley Hills|108.01090828891665|
|5820259228361337957|     Xander Berkeley| 70.85043533114856|
|6033170360494767837|Berkeley County, ...| 67.72143888865239|
+------------------+------------------+------------------+
only showing top 10 rows
```

Figure 1: Result of exercise 4