OD - Distributed Graph Lab: Team OD-L2-11-G

MEYSAM ZAMANI FOROOSHANI - BOYAN ZHANG

meysam.zamani@est.fib.upc.edu - boyan.zhang@est.fib.upc.edu

Exercise 1: Getting familiar with GraphX's Pregel API

In order to make easier the understanding of the superstep explanations we will define the relation between some concepts and how they are represented in the following images:

1. **Vertices:**
   - I The vertices are represented by the circles.
   - II They are accompanied by a label that identifies them.
   - III The circle color shows the current vertex state: Green ones are active and red ones are inactive.
   - IV Inside the circle, we can find two numbers: the number with parentheses is the initial value and the number without parentheses is the current value for the vertex.
2. **Edges:**
   - I The edges are represented with the grey thick arrows and shows the connections between the vertices.
3. **Messages:**
   - I The messages sent between the vertices are represented with black thin arrows.
   - II On the left figures, we show the messages that the vertices receive from the previous superstep or as init messages.
   - III On the right figures, we show the messages that the vertices send to another one.
   - IV The messages are labeled with a number that represents the message content.

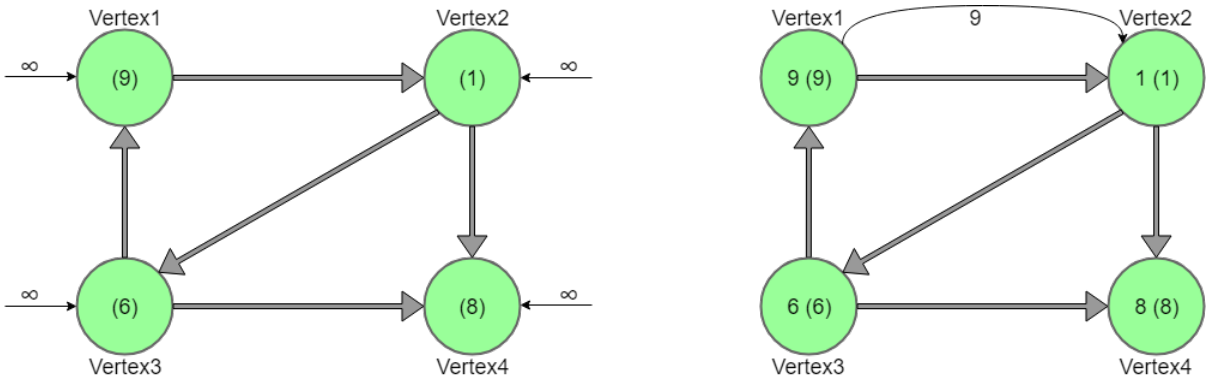
Finally, we have two figures by each superstep. The left one is the initial one and the right one is the last one inside a superstep.

Superstep 0

In the first superstep, all the vertices are active because all receive the initial message.

- **Apply:** All vertices receive an initial message MAX_VALUE. Therefore for all vertices in the apply phase will just apply (return) their current values.
- **Scatter:** The scatter phase receives a triplet containing the source, destination and edge. The source vertex value is compared to the destination vertex value: a message is sent only if the source vertex value is greater than the destination vertex value. Only vertex 1 has any outgoing edge that satisfies this condition (9>1) and therefore is the only vertex that sends a message, namely 9 to vertex 2.
- **Gather:** All nodes receive just one message and so the gather phase is not used.

FIGURA 1 – SuperStep0

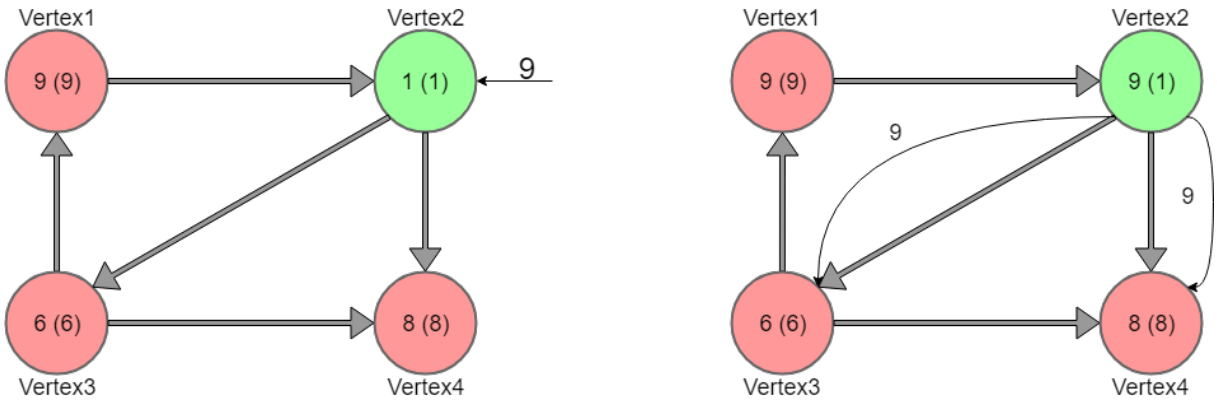


Superstep 1

On the second superstep, Vertex2 is the only active vertex.

- **Apply:** Vertex 2 evaluates the message by comparing against the current value. Given that the received value (9) is bigger than the current one (1), it updates its value to 9.
- **Scatter:** Vertex 2 has outgoing edges that satisfies the condition (see description in S0): 9 > 8 and 9 > 6. Therefore it sends messages to vertex 3 and 4 with the message 9.
- **Gather:** Vertex 2 receives just one message and so the gather phase is not used.

FIGURA 2 – SuperStep1

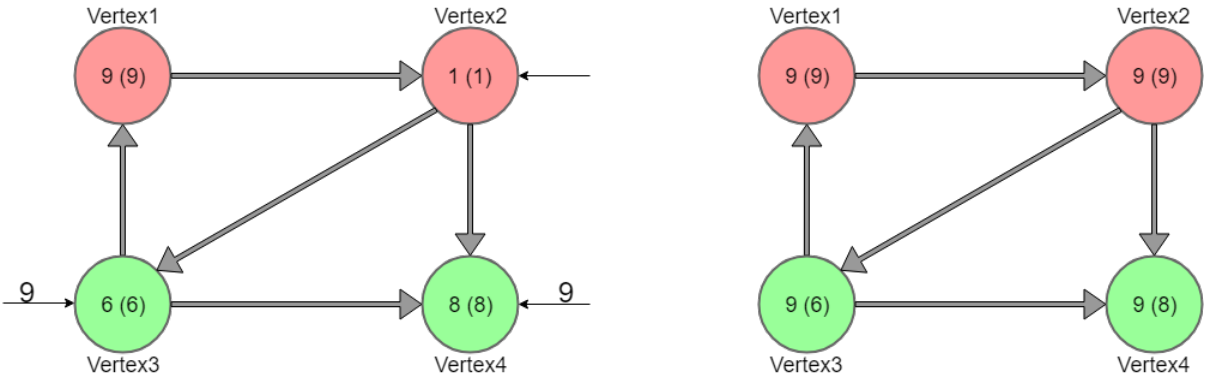


Superstep 2

On the third superstep, Vertex3 and Vertex4 are the active vertices.

- **Apply:** Vertex 3 and 4 evaluates their messages by comparing against the current value. Given that the received value (9) is bigger than the current ones (6 and 8), they update their values to 9.
- **Scatter:** Vertex 3 has outgoing edges but since their value is not greater than 9 (they are already 9), sends no message. Vertex 4 does not have any outgoing edges and finally the algorithm finishes.
- **Gather:** Vertex 3 and 4 receives just one message and so the gather phase is not used.

FIGURA 3 – SuperStep2



Exercise 2: Computing shortest paths using Prege

In order to clarify the code implemented in the three methods, we will describe it in the following sections.

- **VProg:** The apply method, in the VProg class, compares the current path cost and the received one in the message and select the smallest. In the first superstep, the value of the message always is the current path cost.
- **sendMsg:** The apply method, in the sendMsg class, checks if from the current vertex exists a better path to arrive at some neighbor vertex. So, if in this method the source vertex has an infinite number (it doesn't have any path to it) or the new possible path is worse than the current one, the vertex doesn't send the message. If instead, a new better path is possible, the vertex sends a message with the cost of this new path.
- **merge:** The apply method, in the merge class, takes two messages and select the message with the best path proposed (with the smallest cost).

Note: results are sorted by adding the following sortBy:

```
ops.pregel(Integer.MAX_VALUE,
        Integer.MAX_VALUE,
        EdgeDirection.Out(),
        new VProg(),
        new sendMsg(),
        new merge(),
        ClassTag$.MODULE$.apply(Integer.class)) Graph
        .vertices() VertexRDD
        .toJavaRDD() JavaRDD
        .sortBy(v1 -> labels.get(((Tuple2) v1)._1), ascending: true, numPartitions: 1)
        .foreach(v -> {
            Tuple2<Object, Integer> vertex = (Tuple2<Object, Integer>) v;
            System.out.println("Minimum cost to get from " + labels.get(11) + " to " + labels.get(vertex._1) + " is " + vertex._2);
        });
    }
}
```

And finally we obtain the expected output like as the one written in the statement of the assignment.

Exercise 3: Extending shortest paths computation

In general terms, the algorithm is the same than in the previous exercise but saving the nodes that compose the path and not only the cost of the path.

Note: results are sorted by adding the following sortBy:

```
        ops.pregel(new Tuple2("", Integer.MAX_VALUE),
                Integer.MAX_VALUE,
                EdgeDirection.Out(),
                new Exercise_3.VProg(),
                new Exercise_3.sendMsg(),
                new Exercise_3.merge(),
                ClassTag$.MODULE$.apply(Tuple2.class)) Graph
                .vertices() VertexRDD
                .toJavaRDD() JavaRDD
                .sortBy(v1 -> ((Tuple2) ((Tuple2) v1)._2)._1, ascending: true, numPartitions: 1)
                .foreach(v -> {
                    Tuple2<Object, Tuple2> vertex = (Tuple2<Object, Tuple2>) v;
                    System.out.println("Minimum cost to get from " + labels.get(11) + " to " + labels.get(vertex._1) +
                });
    }
}
```

And finally we obtain the expected output like as the one written in the statement of the assignment except that our result is sorted by the cost number as well.

Exercise 4: Spark Graph Frames

The solution is very similar to the warmup exercise and The only different part from the warmup exercise aside from the pagerank function is the reading part. In order to make the code clean and simple we used the map function so that it applies the function spaceForBeauty to the edges and vertices.

You can find the screanshot of the result hereunder:

```
+------------------+------------------+
|               src|               dst|
+------------------+------------------+
| 36359329835505530|6843358505416683693|
|16843740093114490 | 961421098734626813|
|16843740093114490 |1367968407401217879|
|16843740093114490 |2270437664547777682|
|16843740093114490 |2381426201672413470|
|16843740093114490 |3694025250309277803|
|16843740093114490 |4289177408495534056|
|16843740093114490 |4876378461651498574|
|16843740093114490 |4898810637927996530|
|16843740093114490 |5397795112799724806|
|16843740093114490 |5513532497162072006|
|16843740093114490 |5513532497446264506|
|16843740093114490 |5781525238152196030|
|16843740093114490 |6033170360494767837|
|16843740093114490 |7647287135606994048|
|16843740093114490 |8057641240036215591|
|16843740093114490 |8069908956126960798|
|16843740093114490 |8405829285832207680|
|17989888283219993 |4315052278831489964|
|3012818025318676 54|8830299306937918434|
+------------------+------------------+
only showing top 20 rows
```

```
+-------------------+--------------------+
|                 id|                name|
+-------------------+--------------------+
|6598434222544540151|Adelaide Hanscom ...|
|7814958205460279317|David Dodge (nove...|
|3858831448322232257|Howard League for...|
|1778261942684788432|Chelsea Quinn Yarbro|
|4201849915685975228|  Dominick Montiglio|
|7447657998247897725|  Joseph W. Estabrook|
|7816201114341688960|   Spencer J. Palmer|
| 290343697355110541|Aquatic Park (Ber...|
|1748906252821810168|       Johan Galtung|
|3683742251178101007|Henry Howard, 7th...|
|8277480409175395091|         Carl Dennis|
|8847001713698823343|      Robert Axelrod|
|4011285656061867661|National Parliame...|
|1547980201223606474|      Candice Nelson|
|5513532495900171287|             Cartoon|
|1629895207124579982|        Be Our Guest|
|7866692046732241596|   Music of Zimbabwe|
|4898810267628502395|            Scotland|
|6737425368859109805|Edward Clive, 1st...|
|1765549826147530806|        Google China|
+-------------------+--------------------+
only showing top 20 rows
```

```
+-------------------+--------------------+------------------+
|                 id|                name|          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| 70.85043533114853|
|6033170360494767837|Berkeley County, ...| 67.72143888865237|
+-------------------+--------------------+------------------+
only showing top 10 rows


Process finished with exit code 0
```