**Distributed Graph Lab**

Semantic Data Management

Project Report

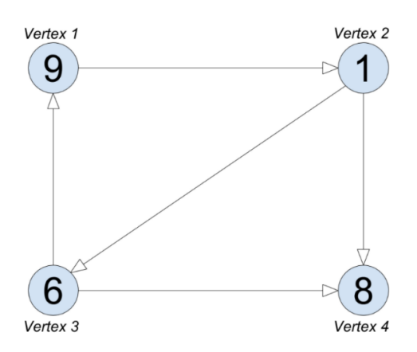
Víctor Diví

Sergio Postigo



April, 2022

## Exercise 1: Getting familiar with GraphX’s Pregel API

For the given graph:

The next supersteps will occur:

1. First superstep:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vertex id | Initial state | Messages received | Result of calculations (merge) | Resulting state (apply) | Message sent (sendMsg) | Destination nodes (sendMsg) |
| 1 | 9 | Integer.MAX\_VALUE | Integer.MAX\_VALUE | 9 | 9 | 2 |
| 2 | 1 | Integer.MAX\_VALUE | Integer.MAX\_VALUE | 1 | - | - |
| 3 | 6 | Integer.MAX\_VALUE | Integer.MAX\_VALUE | 6 | - | - |
| 4 | 8 | Integer.MAX\_VALUE | Integer.MAX\_VALUE | 8 | - | - |

1. Second superstep:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vertex id | Initial state | Messages received | Result of calculations (merge) | Resulting state (apply) | Message sent (sendMsg) | Destination nodes (sendMsg) |
| 1 | 9 | - | - | 9 | - | - |
| 2 | 1 | 9 | 9 | 9 | 9 | 3, 4 |
| 3 | 6 | - | 6 | 6 | - | - |
| 4 | 8 | - | 8 | 8 | - | - |

1. Third superstep:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vertex id | Initial state | Messages received | Result of calculations (merge) | Resulting state (apply) | Message sent (sendMsg) | Destination nodes (sendMsg) |
| 1 | 9 | - | 9 | 9 | - | - |
| 2 | 9 | - | 9 | 9 | - | - |
| 3 | 6 | 9 | 9 | 9 | - | - |
| 4 | 8 | 9 | 9 | 9 | - | - |

After the third superstep, no messages are sent, so no vertex becomes active. Therefore, the algorithm stops with a max value of 9.

## Exercise 2: Computing shortest path using Pregel

A picture containing text, clock

Description automatically generated

Functions implemented:

* *VProg*:

We keep the minimum value between the vertex value and the message received, since it represents the cost of the path.

* *sendMsg*:

If the vertex hasn’t been visited before (i.e. the value is equal to Integer.MAX\_VALUE) it does not send any message. If the vertex has been visited before and the sum of its value and the edge connecting it into a neighbor is less than the value of this neighbor, it sends him a message (this means that the message is only sent if the source vertex “knows” that he will be able to activate the destination vertex).

To make this function more readable, an intermediate class with helper functions has been defined (sendMessage). This class has been used in this and the next exercise.

* *merge:*

Compare two messages received by a vertex and take the one with the smallest value.

The outcome of the algorithm gives:

|  |  |
| --- | --- |
| Vertex id | Distance to A |
| A | 0 | |
| B | 4 | |
| C | 2 | |
| D | 9 | |
| E | 5 | |
| F | 20 | |

## Exercise 3: Extending shortest path’s computation

Functions implemented:

* *VProg*:

We keep the minimum value between the vertex value and the message received, since it represents the cost of the path.

* *sendMsg*:

If the vertex hasn’t been visited before (i.e. the value is equal to Integer.MAX\_VALUE) it does not send any message. If the vertex has been visited before and the sum of its value and the edge connecting it into a neighbor is less than the value of this neighbor, it sends him a message (this means that the message is only sent if the source vertex “knows” that he will be able to activate the destination vertex). The message sent contains the new value and path, which is constructed by appending the source vertex id to the path hold by the source vertex.

* *merge:*

Compare two messages received by a vertex and take the one with the smallest value.

Vertex structure:

In addition to the value of the shortest path, each vertex is also holding the actual sequence of vertex of that shortest path. Therefore, the value that holds is a tuple of an integer and a list (holding the path) instead of just an integer as in the previous exercise.

The outcome of the algorithm gives:

|  |  |  |  |
| --- | --- | --- | --- |
| Vertex id | Distance to A | Previous | |
| A | 0 | | - |
| B | 4 | | A |
| C | 2 | | A |
| D | 9 | | E |
| E | 5 | | C |
| F | 20 | | D |

## Exercise 4: Spark Graph Frames

Assumptions:

The damping factor used for the algorithm will be 0.85, which is the value normally used according to Wikipedia [1].

Number of iterations:

To decide the number of iterations, we run the algorithm using different candidate values obtaining the following results:

|  |  |
| --- | --- |
| **# of iterations** | **Ranking** |
| 13 | 3124.2846184169443: University of California, Berkeley  1574.90977265344: Berkeley, California  384.5868549805489: Uc berkeley  214.49567822339105: Berkeley Software Distribution  194.14981006269468: Lawrence Berkeley National Laboratory  193.91408400921236: George Berkeley  114.26993162191906: Busby Berkeley  106.71180918538985: Berkeley Hills  71.88083610563913: Xander Berkeley  69.43485723833615: Berkeley County, South Carolina |
| 14 | 3123.740036305703: University of California, Berkeley  1573.2220850332903: Berkeley, California  384.3556998595081: Uc berkeley  214.17674950544043: Berkeley Software Distribution  193.90114084964398: Lawrence Berkeley National Laboratory  193.627903823603: George Berkeley  112.91495422451158: Busby Berkeley  106.28293023021543: Berkeley Hills  71.72119170552183: Xander Berkeley  68.44701243713783: Berkeley County, South Carolina |
| 15 | 3124.20449943633: University of California, Berkeley  1573.2671443156498: Berkeley, California  384.3679137785905: Uc berkeley  214.19545632690213: Berkeley Software Distribution  193.8568183255376: Lawrence Berkeley National Laboratory  193.73766909136964: George Berkeley  113.66205541521822: Busby Berkeley  106.1902727437036: Berkeley Hills  71.84793738331805: Xander Berkeley  68.90254856824049: Berkeley County, South Carolina |
| 16 | 3124.0769801168676: University of California, Berkeley  1572.754300750487: Berkeley, California  384.2978725303647: Uc berkeley  214.10572779193114: Berkeley Software Distribution  193.77327161576494: Lawrence Berkeley National Laboratory  193.65860868232372: George Berkeley  113.12472530188563: Busby Berkeley  106.04937426580793: Berkeley Hills  71.80422786731432: Xander Berkeley  68.48576311082452: Berkeley County, South Carolina |
| 17 | 3124.221568101808: University of California, Berkeley  1572.7118813059612: Berkeley, California  384.2928143706957: Uc berkeley  214.10254754384889: Berkeley Software Distribution  193.75114604754464: Lawrence Berkeley National Laboratory  193.68901047436088: George Berkeley  113.41823592938348: Busby Berkeley  106.00426835497727: Berkeley Hills  71.84602062606416: Xander Berkeley  68.66539932290776: Berkeley County, South Carolina |

We can see that the convergence starts at around 15 iterations and thus, this is the value we choose for the algorithm.

## References

[1] Wikipedia contributors. (2022, 24 abril). PageRank. Wikipedia. https://en.wikipedia.org/wiki/PageRank#Damping\_factor