# Semantic Data Management
# Master in Data Science
# Project 2: Distributed Graph Processing Lab



Andrea Iglesias…..……andrea.iglesias@estudiantat.upc.edu
Meritxell Arbiol López……meritxell.arbiol@estudiantat.upc.edu

## Exercise 1

Super step 0

The algorithm starts with all nodes active and all of them receive the initial message of MAX_VALUE. Each node runs vertex program which does not cause any change to the vertex value of any node because the initial message is MAX_VALUE.
Next sendMsg function is triggered across all edges:

- Node 1 sends a message to node 2 and the value of this message is 9, because 9 is the value of the source vertex that is bigger than 1 the destination vertex.
- Node 2 does not generate any message because its value (1) is smaller than all the other nodes (vertex 3 and 4 with values 6 and 8 respectively) which are connected to node 2.
- Node 3 does not generate any message because its value (6) is smaller than all the other nodes (vertex 1 and 4 with values 9 and 8 respectively) which are connected to node 3.
- Node 4 does not generate any message because it does not have any edge coming out of the vertex.

Super step 1

Only node 2 runs vertex program with changes. The message received by vertex 1 is 9, which is bigger than the vertex's value that is 1, therefore the vertex's value is updated.
Next sendMsg function is triggered across all edges:

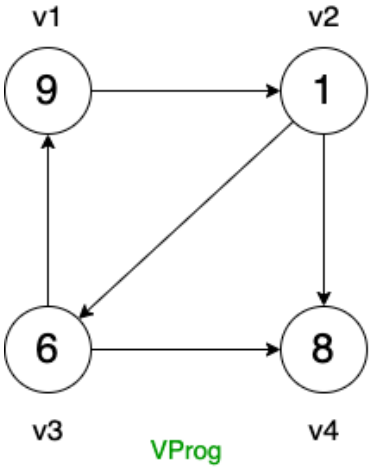
- Node 1 does not generate any message because its value (9) is equal to the vertex value of node 2 (9).
- Node 2 sends a message to node 3 and 4 and the value of the message is 9, because 9 is the value of the source vertex that is bigger than 6 and bigger than 8, the destination vertices 3 and 4 respectively.
- Node 3 as in the previous super step does not generate any message because its value (6) is smaller than all the other nodes (vertex 1 and 4 with values 9 and 8 respectively) which are connected to node 3.
- Node 4 as before does not generate any message because it does not have any edge coming out of the vertex.

Super step 2

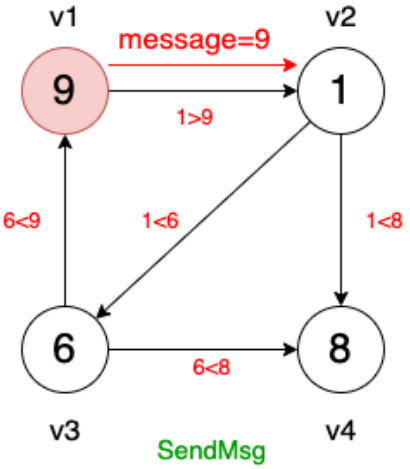
Vertex program is run with changes for nodes 3 and 4, they receive the same message (9) from node 2, which is bigger than their actual vertex values (6 and 8 respectively), therefore the vertices' values are updated.
Next sendMsg function is triggered across all edges:

- Node 1 as in the previous super step does not generate any message because its value (9) is equal to the vertex value of node 2 (9).
- Node 2 does not generate any message because its value (9) is equal to the vertex value of node 3 (9) and node 4 (9).
- Node 3 does not generate any message because its value (9) is equal to the vertex value of node 1 (9) and node 4 (9).
- Node 4 as before does not generate any message because it does not have any edge coming out of the vertex.

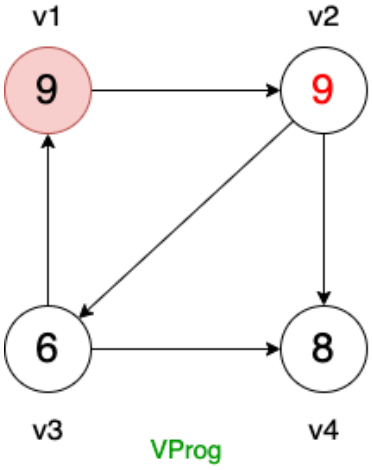
## SUPERSTEP 0



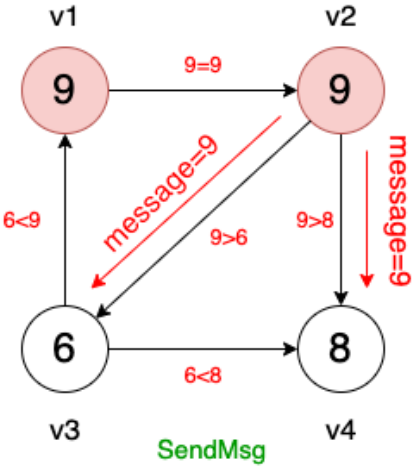

## SUPERSTEP 0



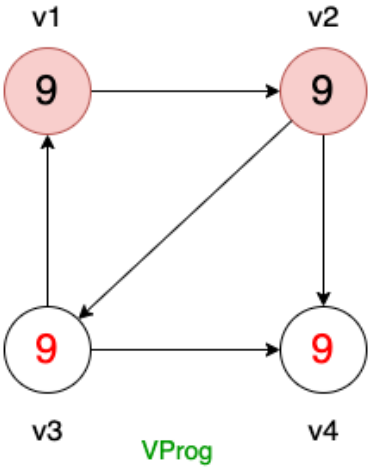

## SUPERSTEP 1



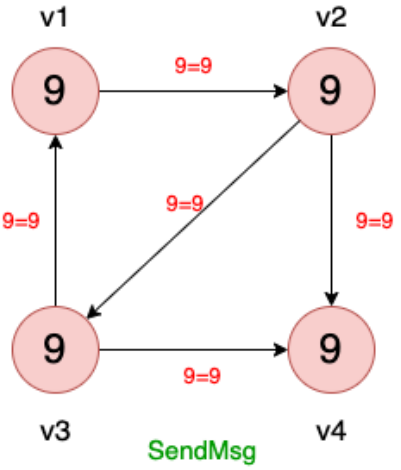

## SUPERSTEP 1




## SUPERSTEP 2




## SUPERSTEP 2



## Exercise 2

In exercise 2 we have reused the code we have for exercise 1 making a series of modifications.

In the VProg function, the only change we have made is to substitute the maximum function for the minimum one, so that the chosen path is the shortest. Otherwise, for the sendMsg function we have modified the message to be the sum of the weights of the path. And finally we have transformed the pregel output to have the distances for all vertices and the result sorted by the vertex letter.

## Exercise 3

To do exercise 3, we have used the same code that we have created for exercise 2, but making small modifications.

The first and biggest modification we have made is in the initialization of vertices of the graph. Before we had each of the vertices defined by a list of tuples composed by an Object and an Integer number. Now instead we have a list of tuples composed by an object and a tuple containing an integer number and a list. In this list that we have now is where we are going to add the paths between the vertices.

In the VProg function we use the same logic as in Exercise 2, returning the value of the message when it is less than the vertex value, and otherwise (when the *message* is greater than or equal to the *vertex value*) we return the *vertex value*. In this way we are returning the cost and the path, in this way we store both values in the vertice.

In the sendMsg function we check if two conditions are fulfilled. The first condition is that the vertex cost is less than the predefined maximum value (*MAX_VALUE*) and the second one to be fulfilled is that the vertex source value is less than the vertex destination value. When both conditions are met, we are going to add the new value of the destination vertex in our list called *path*.

## Exercise 4

For this exercise we have generated the graph using a couple of files that represents the edges and vertex attributes:

- The wiki-vertex.txt file, where we have the identifier, and the other with the movie title.
- The wiki-edges.txt file, we have the source vertex identifier, and the destination vertex identifier.

We want to compute in this exercise the PageRank in order to get the 10 most important pages of Wikipedia.

To run the PageRank algorithm, we have to find the most optimal values for the "MAX_ITERATIONS" and "DAMPING_FACTOR" parameters. For them, we have been doing

several tests. After some research we have seen that the most common Damping Factor is 0.85. We have tried higher values and the results vary a little, and if we take very low numbers the same thing happens.

Regarding the number of iterations, we have been testing several options of values. We have tested with the value 300 and see that it does not converge. With 200 iterations we see that the algorithm takes a long time to run. Furthermore, if we compare the results obtained with 200 iterations with the results obtained with 100 iterations we see that we obtain the same list of titles. Seeing that the results are the same and with half the number of iterations the execution is much faster, it would be a much more optimal solution.

Here we have the results with the DAMPING FACTOR = 0.85 and MAX_ITERATIONS = 100:

| id | title | pagerank |
|---|---|---|
| 8830299306937918434 | University of Cal... | 3124.264171486379 |
| 1746517089350976281 | Berkeley, California | 1572.4151005334245 |
| 8262690695090170653 | Uc berkeley | 384.25144673972784 |
| 7097126743572404313 | Berkeley Software... | 214.0572551757324 |
| 8494280508059481751 | Lawrence Berkeley... | 193.68997507337514 |
| 1735121673437871410 | George Berkeley | 193.6716555575404 |
| 6990487747244935452 | Busby Berkeley | 113.25606403048357 |
| 1164897641584173425 | Berkeley Hills | 105.89779003610528 |
| 5820259228361337957 | Xander Berkeley | 71.85250422941961 |
| 6033170360494767837 | Berkeley County, ... | 68.47601926652182 |