Distributed Graphs Lab

Semantic Data Management

Lab Report

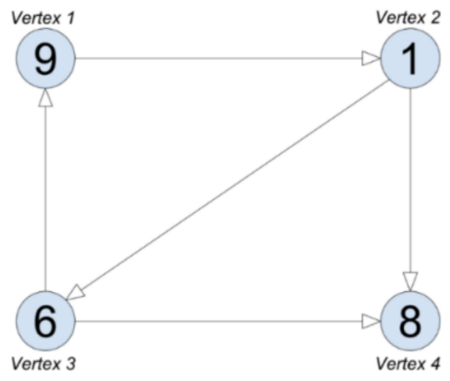
Víctor Diví

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`)	**Final 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	-	-

2. Second superstep:

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

3. Third superstep:

Vertex id	**Initial state**	**Messages received**	**Result of calculations** (`merge`)	**Final state** (`apply`)	**Message sent** (`sendMsg`)	**Destination nodes** (`sendMsg`)
1	9	-	-	9	-	-
2	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.

In the previous tables, we have filled some values in the "Result of calculations (`merge`)" and "Final state (`apply`)" columns, even though the function (`merge` and `apply` respectively) was not run in that particular case. In particular, `merge` was never run, since in no superstep a vertex received more than one message, and `apply` was only run in those cases in which the vertex received a message.

Similarly, in the "Message sent (`sendMsg`)" and "Destination nodes (`sendMsg`)" columns, we make no distinction in whether the `sendMsg` function was not run or it was but no message was sent. Like the apply function, `sendMsg` was only run in the cases that a vertex received a message.

Exercise 2: Computing shortest path using Pregel

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 has been visited before (i.e., the value is not equal to `Integer.`*`MAX_VALUE`*) and the sum of its value and the edge connecting it into a neighbor is less than the value of this neighbor (i.e., a shorter path has been found for the neighbor), it sends it a message.
  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`*:*
  We keep the message with the lowest value (shortest path).

Exercise 3: Extending shortest path's computation

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 of each vertex is a tuple of an integer and a list (holding the path) instead of just an integer as in the previous exercise.

Functions implemented:

- `VProg`:
  Same as exercise 2 (keep minimum value).

- `sendMsg`:
  Same as exercise 2, but since we are also keeping the path, the message contains the new value and the new path, which is constructed by appending the source vertex id to the path hold by the source vertex.

- `merge`*:*
  Same as exercise 2 (keep minimum value).

Exercise 4: Spark Graph Frames

Since the damping factor doesn't affect the "accuracy" of the values (i.e., changing it just makes the PageRank algorithm yield other values, but they are neither better nor worse), we use the generally accepted value, proposed in the original paper by (Brin & Page, 1998).

We choose the number of iterations so that the average difference in the PageRank values is below a certain threshold (we use 0.0001, the same value used for TextRank (Mihalcea & Tarau, 2004)). Table 1 shows the differences for each iteration. As can be seen, this value falls below the threshold at 20 iterations.

#Iterations	Average PageRank Difference
10	0,008594
11	0,006297
12	0,002808
13	0,002065
14	0,001026
15	0,000719
16	0,000391
17	0,000266

18	0,000151
19	0,000101
20	0,000059

Table 1 Average difference in PageRank values per iteration

The results obtained are shown in Table 2:

Entry	**PageRank value**
University of California, Berkeley	3124,23
Berkeley, California	1572,47
Uc berkeley	384,26
Berkeley Software Distribution	214,06
Lawrence Berkeley National Laboratory	193,70
George Berkeley	193,67
Busby Berkeley	113,24
Berkeley Hills	105,92
Xander Berkeley	71,85
Berkeley County, South Carolina	68,49

Table 2 Top entries with their PageRank value

References

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems, 30*(1-7), 107-117. doi:10.1016/S0169-7552(98)00110-X

Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing Order into Text. *Conference on Empirical Methods in Natural Language Processing*, (pp. 404–411). Barcelona. Retrieved from https://aclanthology.org/W04-3252