Atividade Dirigida

CSC-27
DCTA - ITA - IEC
Gabriel Gandour

Goal: To present the Pregel framework, as well as its alternative Go-Pregel for distributed graph processing. Furthermore, it is also a goal to execute a hands-on activity to convert a sequential centralized graph algorithm to a distributed version.

Background

Pregel is a Google framework used for writing distributed graph algorithms.
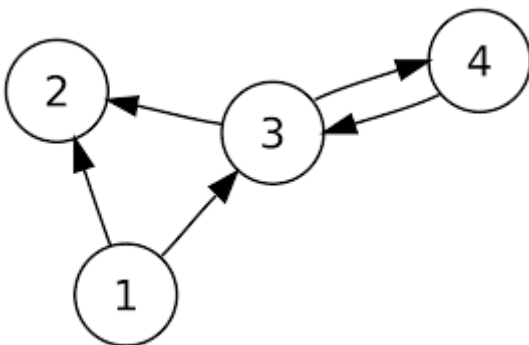
Go-Pregel is a Golang version of Pregel, written by Gandour for his Graduation Thesis. Its basic functionality is the same as the original Pregel, and both frameworks work in the same way (although Google's Pregel is written in C++). From now on, for simplicity, all mentions of Pregel refer to Go-Pregel.

Input and Output

The input of a Pregel algorithm is always a graph, and the output of a Pregel algorithm is always a graph (there are some exceptions that won't be covered in this document). However, in some algorithms, the desired output is not a graph. For example, in the SSSP problem, the output is maybe a path from a source to the destiny, along with the distance traveled.

We can't make the Pregel algorithm print the path, nor the distance, but we can store the minimum distance in each vertex, and we can store the neighbor vertex that leads to the shortest path to the origin.

The key idea is to always save important information inside each vertex of the graph, even in the middle of the algorithm.

Bulk Synchronous Parallel (BSP)

Pregel uses the BSP paradigm to execute the algorithm, together with a master-worker architecture.

In BSP, the algorithm is divided into several blocks named Supersteps. In each superstep, each worker (or node) does a certain amount of local computation, followed by communicating the necessary information with the other workers. After communication, each node meets a synchronization barrier, which stops them from proceeding to the next superstep until every other worker arrives at the same barrier. When every node is at the barrier, they are allowed to go to the next superstep.

In Pregel, the Synchronization barrier is managed by the master machine. The main responsibility of the master machine in Pregel is telling the workers when they can proceed to the next superstep.
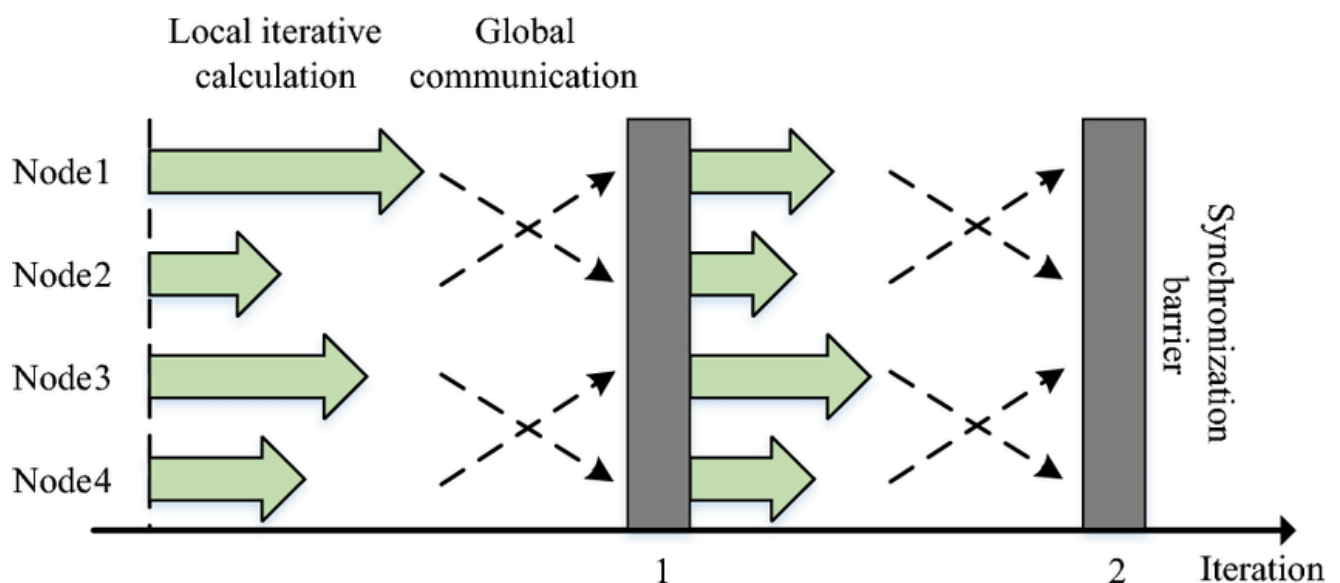

Pregel and BSP

In Pregel, each vertex of the graph acts as an independent node, with its own computation. Each vertex is capable of changing its own

internal state, sending messages to other vertices, receiving messages and interpreting them.

BSP shows us how Pregel executes its computation, but it doesn't show us how the algorithm stops. The algorithm stops by unanimity. After each superstep, each vertex votes if Pregel should or not halt. If every vertex agrees to halt the algorithm, it stops, and the final graph is written. When a vertex votes to halt, it becomes inactive, and it won't take part in the next computation steps. The vertex stays inactive forever, or until it receives a message from another vertex (after which it becomes active again, and the vote to halt is nullified until it votes again).

The best way to understand it is by watching a real Pregel execution. The next image shows the execution of a Pregel algorithm. The input and output are strongly connected graphs in which each vertex has an integer value. The algorithm finds the minimum value among all the vertices and stores it in every vertex. The white vertices are active and the gray ones are inactive. The full arrows are edges and the dotted arrows are messages being passed.
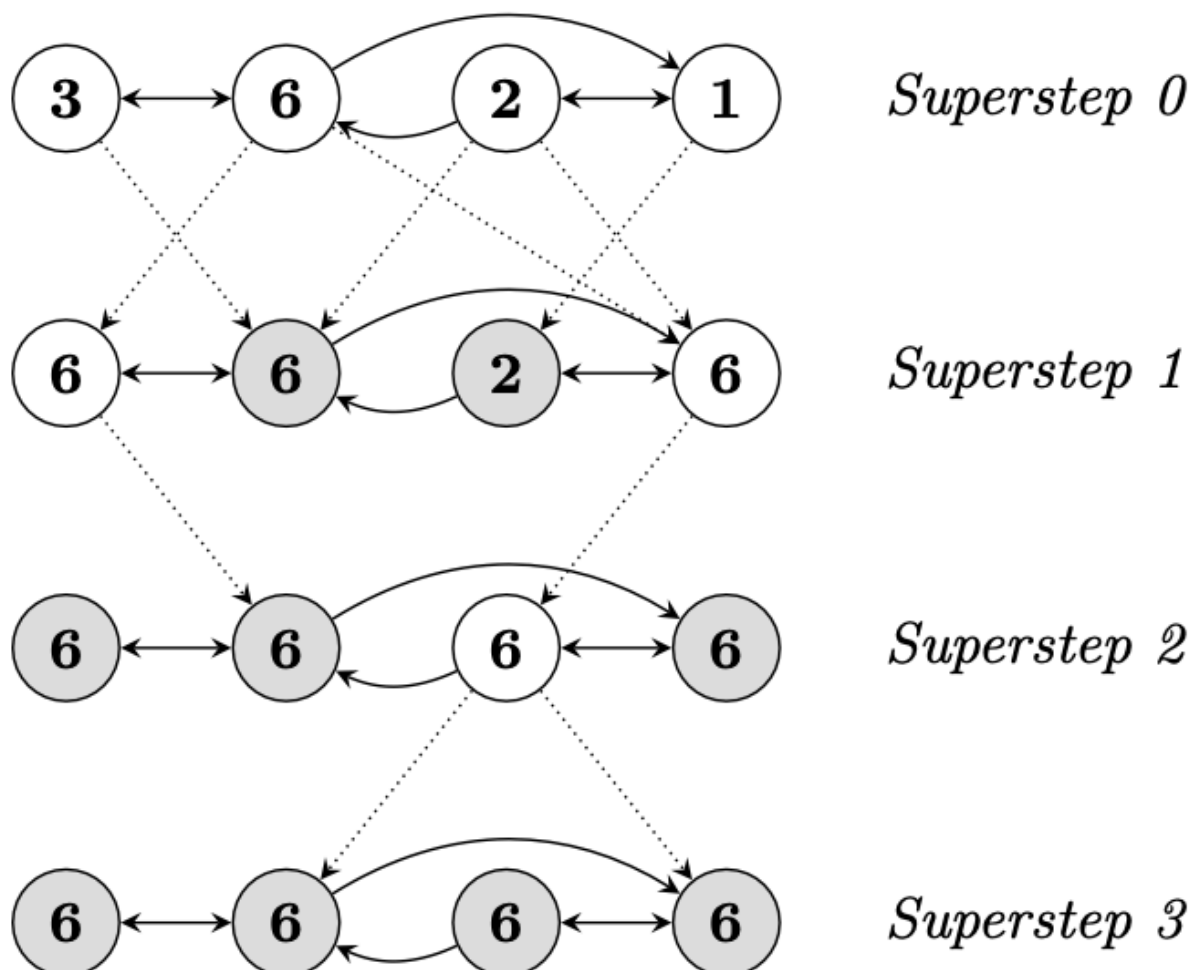
Activity

1. Gather in pairs. If you want, you can also do the activity alone. However, even if you do it in pairs, each person has to write the code and make it work on their own computer. Each pair will hand in a single report.
2. Go to https://github.com/GaGandour/Go-Pregel and clone the repo.
3. Read the pregel_writing_guide.md file to understand which files you have to modify, if necessary.
4. Start coding!

There will be three levels of dificulty, to be done in order. For each level of dificulty, you have to:

1. Write the algorithm in Go-Pregel.
2. Test the algorithm using the provided testing script (read the repo README.md)
3. Be sure that you algorithm passes all tests.

What needs to be written on the report

1. The code of the algorithm for each level of dificulty.
2. The reasoning behind it and how you came up with it.
3. The print of the terminal saying that the algorithm passed all tests.
4. An image of the last graph for the algorithm, generated by the visualization tool.

Every algorithm inside the `/graphs/` folder has files named `graph1.json`, `graph2.json` and so on. The last graph is the one that has the highest number.

The Levels

Level 1 (Easy)

Choose between one of the following algorithms:

1. Connected Components
2. Topological Sort

Level 2 (Medium)

Implement the Bipartite Graph algorithm in Go-Pregel, and then test the algorithm on the given graphs.

Level 3 (Ultra hard - Optional)

Solve Sudoku using Pregel. You have to be creative here.