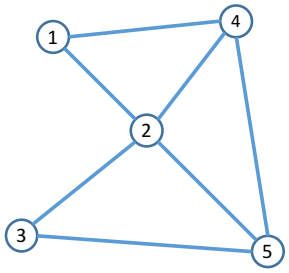
Graph-Matching via Map-Reduce

A $G(V, E)$ is defined by a set of *nodes* V and a set of *edges* $E \subseteq V \times V$. The set of nodes V is an arbirary finite set, while edges are unordered pairs of elements in V. For example, the graph:



is represented as $G(V, E)$ where $V = \{1, 2, 3, 4, 5\}$, and

$$E = \{(1, 2), (1, 4), (2, 3), (2, 4), (2, 5), (3, 5), (4, 5)\}.$$

Note that both (u, v) and (v, u) represent the same edge. Two nodes $u, v \in V$ are *neighbors* if $(u, v) \in E$. The *neighborhood* N_u of a node $u \in V$ is the subset of V containing all of u's neighbors, i.e.: $N_u = \{v : (u, v) \in E\}$. The *degree* $d_u \in \mathbb{N}$ of a node $u \in V$ is the size of its neighborhood, i.e., $d_u = |N_u|$. For example, the neighborhood of node 1 in the graph G shown above is $N_1 = \{2, 4\}$, and its degree is $d_1 = 2$. A graph is called k-*regular* if all its nodes have degree k, i.e., $d_u = k$ for all $u \in V$.

Two graphs $G(V, E)$, $G(V', E')$ with the same number of nodes $n = |V| = |V'|$ are called *isomorphic* if there exists a one-to-one mapping $\sigma : V' \to V$ such that

$$(u, v) \in E \text{ if and only if } (\sigma(u), \sigma(v)) \in E'. \tag{1}$$

The Weisfeiler-Lehman (WL) algorithm is a heuristic determining whether two graphs are isomorphic. To get some intuition behind teh algorithm, note that two graphs $G(V, E), G(V', E')$ cannot be isomorphic if they have different *degree distributions*: the number of nodes having, e.g., degree 5, must be the same in both graphs. If, on the other hand, the two graphs have the same degree distribution then, necessarily, a mapping σ that satisfies (1) must map a node $u \in V$ to a node in $u' \in V'$ s.t. $d_u = d_{u'}$.

Building on this idea, to test whether two graphs G, G' are isomorphic, WL first *colors* the nodes of each graph iteratively as follows. Initially, at iteration $k = 0$, all nodes in a graph G receive the same *color*, which is a number in $\mathbb{N}$. E.g., all $u \in V$ receive the identical color:

$$c^k(u) := 1, \qquad \text{for } k = 0.$$

Colors at iteration later iterations are defined recursively. In particular, the color $c^{k+1}(u) \in \mathbb{N}$ of node $u \in V$ at the $k + 1$-th iteration is:

$$c^{k+1}(u) := \mathsf{hash}\left(\mathsf{sort}\left(\mathsf{clist}_u^k\right)\right), \qquad \text{for } k = 0, 1, 2, \ldots,$$

where hash is a hash function, sort is a function sorting its argument, and

$$\mathsf{clist}_u^k = [c^k(v) : v \in N_u]$$

is a list containing the colors of all of u's neighbors at iteration k. This procedure terminates when the coloring becomes *stable*: the sets of nodes that receive the same color stays the same from one iteration to the next.

Having colored two graphs $G(V, E), G'(V', E')$ thusly, to determine if they are isomorphic, the WL algorithm checks if the two graphs have the same color distributions, i.e., (a) the same colors are present in both graphs, and (b) the number of nodes having a certain color is the same in both graphs. If the two graphs do not have the same color distributions, the algorithm outputs "not isomorphic". If the two graphs have the same color distributions, and the number of colors is exactly n, the algorithm outputs "isomorphic": the one-to-one mapping σ that satisfies (1) is precisely the one that maps a node $u \in V$ to the unique node in $u' \in V'$ with the same color. Finally, if the two graphs have the same color distribution, but the number of colors is less than n, the graph outputs "maybe isomophic": in the latter case, WL implies a set of *constraints* on candidate functions σ: any σ that satisfies (1) must map nodes in V to nodes in V' that have the same color.

Question 1. Write a Spark program that (a) reads a file containing pairs representing the edges of a graph, one per line, (b) computes the degree of each node, and (c) stores the result in a file. Include your code in your report, explaining what each operation (`map`, `reduce`, `reduceByKey`, etc.) does. Run this on the small graph shown above and include the output in your report.

Question 2. Write a Spark program that (a) reads a file containing pairs representing the edges of a graph, one per line, (b) computes the WL coloring of each node, and (c) stores the result in a file. You may use python's internal hash function as `hash`. Moreover, you may use the fact that the total number of colors does not change from one iteration to the next as a heuristic condition to detect that the algorithm has converged. Include your code in your report, and explain how you implemented the algorithm using `join`, `map`, `reduce`, `reduceByKey` operations, etc. Give an example of its execution on some small graphs.

Question 3. Write a Spark program that reads the WL colors of two graphs $G(V, E), G'(V', E')$ and outputs whether they are "not isomorphic", "isomorphic" or "maybe isomorphic". In the latter two cases it should produce a file containing all pairs of the form:

(u, u') where $u \in V$, $u' \in V'$, and both u,u' have received the same color.

Include your code in your report, and explain how you implemented the algorithm using `join`, `map`, `reduce` operations, etc. Give an example of its execution on some small graphs.

Question 4. Run the WL algorithm on the two graphs in files `graph1` and `graph2`. Report whether the algorithm declares them "not isomorphic", "isomorphic", or "maybe isomorphic"; if they are isomorphic, provide the mapping σ between the nodes of the two graphs.

Question 5. Prove that WL always outputs "maybe isomorphic" when given as input two graphs G, G' that are k-regular.