

Algorithm Instruction

Linghao Li
Dec 19, 2016

- **Target (Strongly Connected Components)**

If every vertex is reachable from every other vertex in a graph (or subgraph), the graph (or subgraph) is thought to be strongly connected. In an arbitrary directed graph, strongly connected components are the subgraphs partitioning the whole graph.

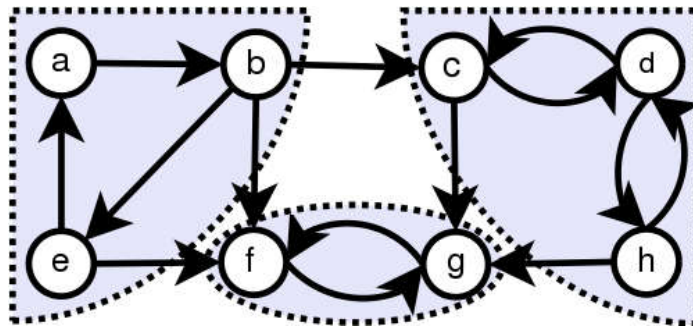


Figure 1: A sample graph with strongly connected components (Wikipedia,
https://en.wikipedia.org/wiki/Strongly_connected_component)

Usually, there will be plenty of strongly connected components in a large network graph. So what we need to do is figure them out by a comparatively simple algorithm due to the volume of data. The algorithm I chose is Bi-directional Label Propagation, which will be discussed explicitly below.

- **Algorithm (Bi-directional Label Propagation)**

This algorithm has 4 parts.

Positive Label Propagation

- Give every vertex a positive label, which is the same as its id number, like Vertex a +0, Vertex b +1.
- Aggregate the child set of each vertex (*child* → *parent*), like (0,+0 4), (1,+1 0), (5,+5 1,4).
The first number is Vertex ID, the second number is its initial positive label, the following numbers splitting by comma are the children.
- Update each vertex's positive label, if its children have a smaller label (absolute value versus absolute value). Use the smaller label as the parent's label. **(Please read the code for update method!)**
- Go to b and repeat until there is no more label update. Keep the converged positive labels.

Reverse

Reverse all arrows in the graph!

Negative Label Propagation

After reversing arrows, give every vertex a negative label, like Vertex a -0, Vertex b -1. The other steps are the same as those in positive label propagation. Finally, keep the converged negative labels.

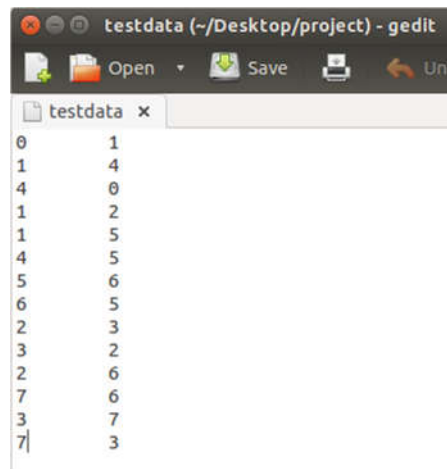
Reduce

Assemble the positive and negative label of each vertex, and those vertexes with the same positive and negative labels are in the same strongly connected component.

Example

Here I use Figure 1 as the example to illustrate the process of the program.

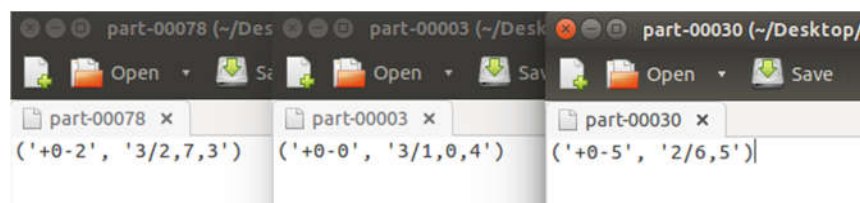
Input data



```
testdata x
0      1
1      4
4      0
1      2
1      5
4      5
5      6
6      5
2      3
3      2
2      6
7      6
3      7
7      3
```

Each line in the input file represents an edge from the 1st id to the 2nd id, like $0 \rightarrow 1$.
(The input data must be in this format)

Output



```
part-00078 x      part-00003 x      part-00030 x
(' +0 -2', '3/2,7,3') (' +0 -0', '3/1,0,4') (' +0 -5', '2/6,5')
```

The output shows that id 2,7,3(c,h,d) are in a component, id 1,0,4(b,a,e) are in another one and 5,6(f,g) are in another one. This graph has 3 strongly connected components.