PolyU COMP4434 Assignment 3

Introduction

Goal

By doing this assignment, you will understand how to use Giraph to implement basic graph algorithms.

Background

Write a Giraph program to conduct the Breadth-First-Search (BFS) over a graph. As an output, we want to obtain the **level number** (w.r.t. the root) of each node.

Basic idea

BFS is an algorithm for traversing a graph. It starts from an arbitrary node of a graph (as root) and explores the neighbor nodes first, before moving to the next level neighbors. The pseudo-code of a non-recursive implementation of BFS is as follows:

Input: A graph $G(V, E)$ and a root node s of G
Output: All nodes reachable from s labeled with their level number $level(v)$

```
level(s) ← 0                                (the level number of root is zero)
for all v ∈ V − {s} do
    level(v) ← ∞                            (set all unseen nodes' level number to infinity)
Q ← ∅                                       (Q, the queue initially contains no nodes)
Q.push(s)
S ← {s}                                     (S, the set of visited nodes)
while Q ≠ ∅
    u ← Q.pop()                             (u is the node with smallest level number in Q)
    for all v ∈ neighbors[u] do
        if v ∉ S
            level(v) = level(u) + 1         (update the level of v)
            Q.push(v)
            S = S ∪ {v}                     (label v as visited)

return level
```

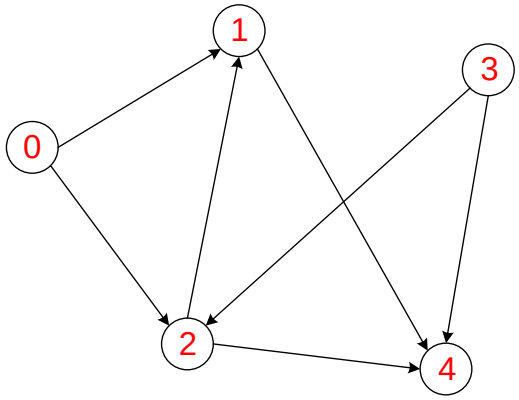
In this assignment, we need to implement BFS **using Pregel's vertex-centric computing model**, which is **different** from above implementation.

Input file format

We use only one plain text file to represent a graph, which follows these rules:

- Each line is in the adjacency-list format
  [source_id,source_value,[[dest_id,edge_weight],[dest_id,edge_weight],...]]
- The node with source id 0 is the root node (with level 0), which is the starting point of BFS

Suppose we want to represent this graph:



Our input file will be this:
[0, -1, [[1, -1], [2, -1]]]
[1, -1, [[4, -1]]]
[2, -1, [[1, -1], [4, -1]]]
[3, -1, [[2, -1], [4, -1]]]
[4, -1, []]

For example, [2, -1, [[1, -1], [4, -1]]] means that Node 2 connects to Node 1 and Node 4 **in a directed manner**. Some remarks in terms of the input format:

- The edge weights are not used, so we set them as -1;
- Initially, the node values are set as -1; after running the BFS, we need to set them as each node's **level number**.

Output file format

The computed result should be like this:

0 0.0
2 1.0
1 1.0
3 1.7976931348623157E308
4 2.0

It represents the node value (i.e., **level number**) of each node. Some remarks in terms of the output format:

- For Node 3 with level number of Double.MAX_VALUE. (i.e., 1.7976931348623157E308), it means that Node 3 is not reachable from the root node (i.e., Node 0).
- The order of lines in the output file can be different.

To begin with

You are given three sets of files:

- ReadMe.pdf (this file)
- **data** folder, it contains two test cases:

    - graph-data1.txt and result1.txt are the first test case:
        - graph-data1.txt is the input file which represents a graph
        - result1.txt is the result for you to check later
    - graph-data2.txt and result2.txt are another test case.
    - **You can use these two cases to test your program, but we will use other test cases to grade your program. So we recommend you to construct your own cases for testing.**
- **project** folder, containing the archive file project.zip. You need to work on it.

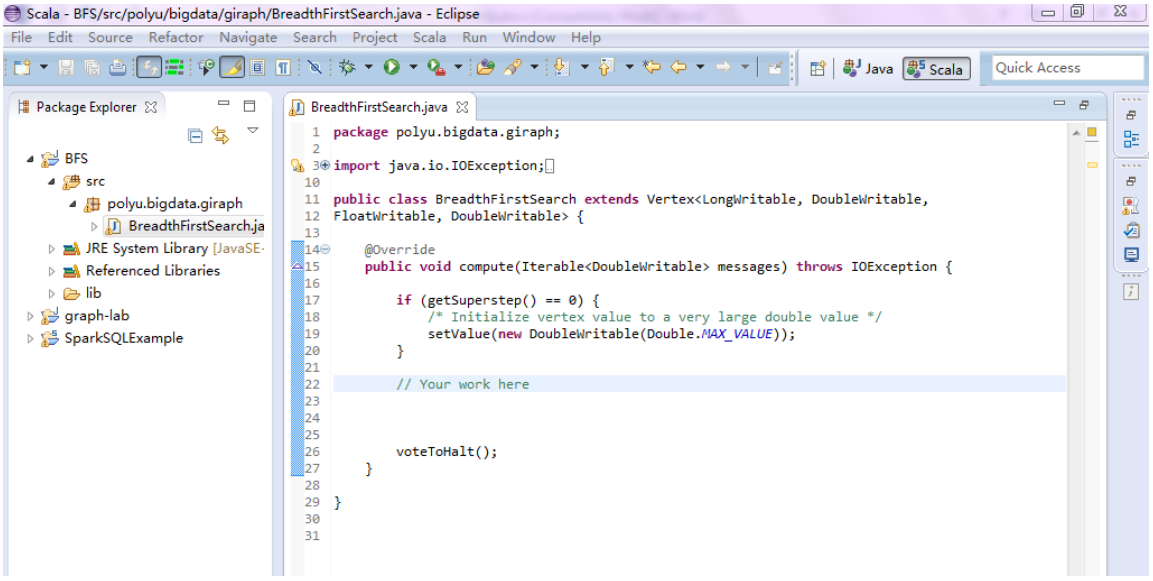
Prerequisite

Giraph is properly installed according to **Installing Giraph Section** in the Giraph lab.

How to run the program

It is very similar to running the examples in the lab. The main steps are as follows (for details of some steps, please refer to the lab material):

- Import project.zip to Eclipse (version 4.3 or above).
- Work on BreadthFirstSearch.java. We have constructed the skeleton of the program for you, and it is runnable; that is to say, even if you edit nothing in BreadthFirstSearch.java, you can still continue the remaining steps, only resulting in obtaining the wrong results finally. So, **don't modify any given parts of the file or otherwise your program will not work correctly. Your job is to complete the compute() function in order to get correct results.**



- Export the project to a **JAR** file and name it as BreadthFirstSearch.jar. As described in lab, do not include the lib folder during the export

- Copy your BreadthFirstSearch.jar to ~/Programs/hadoop-1.2.1/lib in the virtual box image. All the remaining steps will be conducted in the image.
- Start Hadoop. Assume that your current directory is ~/Programs/hadoop-1.2.1/:

```
$ ./bin/start-all.sh
```

- Upload the input file (say, **~/graph-data1.txt**) to HDFS:

```
$ ./bin/hadoop fs -put ~/graph-data1.txt  /user/bigdata/giraph/input/graph-data1.txt
```

- Execute giraph program:

```
$ ./bin/hadoop jar lib/giraph-core.jar org.apache.giraph.GiraphRunner
polyu.bigdata.giraph.BreadthFirstSearch -vif
org.apache.giraph.io.formats.JsonLongDoubleFloatDoubleVertexInputFormat -vip
/user/bigdata/giraph/input/graph-data1.txt -of
org.apache.giraph.io.formats.IdWithValueTextOutputFormat -op
/user/bigdata/giraph/output -w 1
```

  - Please do **NOT** modify the parameters in **black color**.
  - Please specify the command in **green color** according to your current directory.
  - Please specify the giraph core lib in **purple color** according to your current directory.
  - Please specify the input file path in HDFS in **red color**.

- Display output file. If the provided test cases are used, you can compare it with the result file we provided (e.g., result1.txt for graph-data1.txt as input file).

```
$ /hadoop fs -cat /user/bigdata/giraph/output/part-m-00001
```

- Note that by default, the output directory in HDFS cannot be written repeatedly. Therefore, if you want to execute giraph program again, **please remove the output directory first**:

```
$ /hadoop fs -rmr /user/bigdata/giraph/output
```

What to submit

Make a **zip** file called assignment3.zip which contains:

- The exported **JAR** file from Eclipse, i.e., BreadthFirstSearch.jar
- Modified BreadthFirstSearch.java

Lacking in any of two files leads to a failed submission. Submit the zip file through blackboard.

Grading policy

We will use your **JAR** file to check whether the result is correct.
We will use several different input files to test your program, say we use 10 different input files, if your program only passed 8 of the 10, you get 80.

Deadline: 13 April 2015, 14:59

Late Penalty

late x day: your score = raw score * (100 – 20 * x)%

Plagiarism

It is easy to detect the similarity of source files, and cases will be strictly handled according to the University's regulation, so please don't risk doing that.

Questions?

Reach out your teaching assistant Mr. Wenjian Xu (zeroxwj@gmail.com)