# 华东师范大学数据科学与工程学院实验报告

| | | |
|---|---|---|
| **课程名称：** 分布式模型与编程 | **年级：**2016 级 | **上机实践成绩：** |
| **指导教师：**徐辰 | **姓名：**杜云滔 | |
| **上机实践名称** Dataflow 图算法编程 | **学号：** 10153903105 | **上机实践日期：**2018.12.21 |
| **上机实践编号：**14 | **组号：** | **上机实践时间：**2018.12.31 |

一、 **实验目的**

熟悉图算法在分布式系统上的实现与算法流程

二、 **实验任务**

分别使用 MapReduce、Spark、Flink 实现以下算法：

1. PageRank
2. 连通分量 （可选）
3. 单源最短路径 （可选）

三、 **使用环境**

Ubuntu

四、 **实验过程**

# 常用图算法实现--Hadoop

## PageRank

### 数据准备

边：

```
1 2
1 15
2 3
2 4
2 5
2 6
2 7
3 13
4 2
5 11
5 12
6 1
6 7
6 8
7 1
7 8
8 1
```

```
8 9
8 10
9 14
9 1
10 1
10 13
11 12
11 1
12 1
13 14
14 12
15 1
```

网页：

```
1 2
2 5
3 1
4 1
5 2
6 3
7 2
8 3
9 2
10 2
11 2
12 1
13 1
14 1
15 1
```

将这两个文件放入 HDFS：

```
hdfs dfs -mkdir input/PageRank
hdfs dfs -put links.txt input/PageRank
hdfs dfs -put pagesHadoop.txt input/PageRank
```

# 编写程序

## PageRank

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.BufferedReader;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.net.URISyntaxException;

import static java.lang.StrictMath.abs;


public class PageRank {

    private static final String CACHED_PATH = "output/cache";
    private static final String ACTUAL_PATH = "output/Graph/HadoopPageRank";
    public static final int maxIterations = 500;
    public static final double threshold = 0.0001;
    public static final double dumping = 0.85;
    public static int pageNum = 0;

    public static void main(String[] args) throws IOException,
            InterruptedException, ClassNotFoundException, URISyntaxException {

        Configuration conf = new Configuration();
        String[] otherArgs = (new GenericOptionsParser(conf, args)).getRemainingA
rgs();
        if (otherArgs.length != 3) {
            System.err.println("Usage: PageRank <PagePath> <LinksPath> <PageNum>
");
            System.exit(2);
        }

        int code = 0;

        Path PagePath = new Path(otherArgs[0]);
        Path LinksPath = new Path(otherArgs[1]);
        pageNum = Integer.parseInt(otherArgs[2]);

        conf.set("pageNum", pageNum + "");
        conf.set("dumping", dumping + "");


        Path cachePath = new Path(CACHED_PATH);
        Path actualPath = new Path(ACTUAL_PATH);

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(actualPath))
            hdfs.delete(actualPath, true); // recursive delete

        // prepare original rank
        for (int i = 1; i <= pageNum; i++)
            writeFileByline(ACTUAL_PATH + "/part-r-00000", i + " " + 1.0 / pageNu
m);


        int counter = 0;
        boolean changed = true;
```

```
        while (counter < maxIterations && changed) {

            // Delete output if exists
            if (hdfs.exists(cachePath))
                hdfs.delete(cachePath, true);
            //moving the previous iteration file to the cache directory
            hdfs.rename(actualPath, cachePath);

            conf.set("mapreduce.output.textoutputformat.separator", " ");
            conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separato
r", " ");


            Job PageRank = Job.getInstance(conf, "PageRank " + (counter + ""));

            // add cache
            PageRank.addCacheFile(PagePath.toUri());

            PageRank.setJarByClass(PageRankMapper.class);
            FileInputFormat.addInputPath(PageRank, LinksPath);
            // set out put path : output/means
            FileOutputFormat.setOutputPath(PageRank, actualPath);

            PageRank.setMapperClass(PageRankMapper.class);
            PageRank.setInputFormatClass(KeyValueTextInputFormat.class);
            PageRank.setMapOutputKeyClass(IntWritable.class);
            PageRank.setMapOutputValueClass(DoubleWritable.class);

            PageRank.setReducerClass(PageRankReducer.class);
            PageRank.setOutputKeyClass(IntWritable.class);
            PageRank.setOutputValueClass(DoubleWritable.class);

            // Execute job
            code = PageRank.waitForCompletion(true) ? 0 : 1;

            //checking if the mean is stable
            BufferedReader file1Reader = new BufferedReader(new InputStreamReader
(hdfs.open(new Path(CACHED_PATH + "/part-r-00000"))));
            BufferedReader file2Reader = new BufferedReader(new InputStreamReader
(hdfs.open(new Path(ACTUAL_PATH + "/part-r-00000"))));
            for (int i = 0; i < pageNum; i++) {
                double rank1 = Double.parseDouble(file1Reader.readLine().split("
")[1]);
                double rank2 = Double.parseDouble(file2Reader.readLine().split("
")[1]);

                if (abs(rank1 - rank2) <= threshold) {
                    changed = false;
                } else {
                    changed = true;
                    break;
                }
            }
            file1Reader.close();
```

```
            file2Reader.close();
            counter++;
            System.out.println("PageRank finished iteration:>> " + counter + " ||
rank change: " + changed);

        }

        System.exit(code);

    }


    public static void writeFileByline(String dst, String contents) throws IOExce
ption {
        Configuration conf = new Configuration();
        Path dstPath = new Path(dst);
        FileSystem fs = dstPath.getFileSystem(conf);
        FSDataOutputStream outputStream = null;

        if (!fs.exists(dstPath)) {
            outputStream = fs.create(dstPath);
        } else {
            outputStream = fs.append(dstPath);
        }
        contents = contents + "\n";
        outputStream.write(contents.getBytes("utf-8"));
        outputStream.close();
    }

}
```

## PageRankMapper

```
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

public class PageRankMapper extends Mapper<Text, Text, IntWritable, DoubleWritab
le> {

    Map<Integer, Double> rank = new HashMap<>();
    Map<Integer, Integer> pages = new HashMap<>();
```

```
    /**
     * reading the rank from the distributed cache
     */
    public void setup(Context context) throws IOException, InterruptedException {
        String lineString = null;
        // read rank file
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);
        FSDataInputStream hdfsInStream = fs.open(new Path("output/cache/part-r-00
000"));
        InputStreamReader isr = new InputStreamReader(hdfsInStream, "utf-8");
        BufferedReader br = new BufferedReader(isr);

        while ((lineString = br.readLine()) != null) {
            String[] keyValue = StringUtils.split(lineString, " ");
            rank.put(Integer.parseInt(keyValue[0]), Double.parseDouble(keyValue
[1]));

        }
        br.close();

        // read pages file
        String PagesFiles = context.getLocalCacheFiles()[0].getName();
        br = new BufferedReader(new FileReader(PagesFiles));
        while ((lineString = br.readLine()) != null) {
            String[] keyValue = StringUtils.split(lineString, " ");
            pages.put(Integer.parseInt(keyValue[0]), Integer.parseInt(keyValue
[1]));
        }
        br.close();

    }

    public void map(Text from, Text to, Context context) throws IOException, Inte
rruptedException {
        int fromPoint = Integer.parseInt(from.toString());
        int toPoint = Integer.parseInt(to.toString());
        double newRank = rank.get(fromPoint) * (1.0 / pages.get(fromPoint));

        context.write(new IntWritable(toPoint), new DoubleWritable(newRank));
    }

}
```

### PageRankReducer

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class PageRankReducer extends Reducer<IntWritable, DoubleWritable, IntWri
table, DoubleWritable> {
```

```
    public void reduce(IntWritable key, Iterable<DoubleWritable> values, Context
context) throws IOException,
            InterruptedException {

        Configuration conf = context.getConfiguration();
        int pageNum = Integer.parseInt(conf.get("pageNum"));
        double dumping = Double.parseDouble(conf.get("dumping"));

        double rank = 0.0;
        for (DoubleWritable value : values)
            rank += value.get();

        rank = (1 - dumping) * (1.0/pageNum) + dumping * rank;

        context.write(key, new DoubleWritable(rank));

    }
}
```

思路：

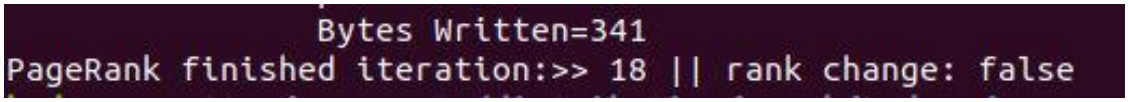
1. 首先指定 KeyValueTextInputFormat，并指定 page 个数（在 Hadoop 中不太好直接求）
2. 将每个顶点的出度文件 pagesHadoop 作为 distributionCache，并首先将初始 rank 值写入 cache 文件中
3. 每次读 cache 文件中的 rank 值，再进行计算，写入目标文件中，前后的 rank 值进行比较，若不满足阈值，将更新后的 rank 值写入 cache 中继续进行迭代

## 运行

```
hadoop jar PageRank.jar input/PageRank/pagesHadoop.txt input/PageRank/links.txt
15
```

可以发现，Hadoop 执行循环操作，比 spark、flink 慢很多

查看结果：




```
hdfs dfs -cat output/Graph/HadoopPageRank/*
```

```
hadoop@scott:~$ hdfs dfs -cat output/Graph/HadoopPageRank/*
1 0.25118644537468443
2 0.14645954992741744
3 0.03488574758399919
4 0.03488574758399919
5 0.03488574758399919
6 0.03488574758399919
7 0.04477462810650671
8 0.03892230960844245
9 0.021024603639025516
```

# ConnectedComponents

## 数据准备

提供基本数据集，与 PageRank 一样，指定顶点和边

### vertices.txt

准备一些顶点，例如 1-16

### edges.txt

准备一些连接边：

```
1 2
2 3
2 4
3 5
6 7
8 9
8 10
5 11
11 12
10 13
9 14
13 14
1 15
16 1
```

放入 HDFS：

```
hdfs dfs -mkdir input/ConnectedComponents
hdfs dfs -put edges.txt input/ConnectedComponents
```

## 编写程序

### ConnectedComponents

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URISyntaxException;


public class ConnectedComponents {

    private static final String CACHED_PATH = "output/cache";
    private static final String ACTUAL_PATH = "output/Graph/HadoopConnectedCompon
ents";
    public static final int maxIterations = 100;
    public static int verticesNum = 0;

    public static void main(String[] args) throws IOException,
            InterruptedException, ClassNotFoundException, URISyntaxException {

        Configuration conf = new Configuration();
        String[] otherArgs = (new GenericOptionsParser(conf, args)).getRemainingA
rgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: PageRank <EdgesPath> <verticesNum>");
            System.exit(2);
        }

        int code = 0;

        Path EdgesPath = new Path(otherArgs[0]);
        verticesNum = Integer.parseInt(otherArgs[1]);

        conf.set("verticesNum", verticesNum + "");

        Path cachePath = new Path(CACHED_PATH);
        Path actualPath = new Path(ACTUAL_PATH);

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(actualPath))
            hdfs.delete(actualPath, true); // recursive delete

        // prepare original ConnectedComponents
        for (int i = 1; i <= verticesNum; i++)
            writeFileByline(ACTUAL_PATH + "/part-r-00000", i + " " + i);


        int counter = 0;
        boolean changed = true;

        while (counter < maxIterations && changed) {

            // Delete output if exists
```

```
            if (hdfs.exists(cachePath))
                hdfs.delete(cachePath, true);
            //moving the previous iteration file to the cache directory
            hdfs.rename(actualPath, cachePath);

            conf.set("mapreduce.output.textoutputformat.separator", " ");
            conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separato
r", " ");


            Job PageRank = Job.getInstance(conf, "ConnectedComponents " + (counte
r + ""));


            PageRank.setJarByClass(ConnectedComponents.class);
            FileInputFormat.addInputPath(PageRank, EdgesPath);
            FileOutputFormat.setOutputPath(PageRank, actualPath);

            PageRank.setMapperClass(ConnectedComponentsMapper.class);
            PageRank.setInputFormatClass(KeyValueTextInputFormat.class);
            PageRank.setMapOutputKeyClass(IntWritable.class);
            PageRank.setMapOutputValueClass(IntWritable.class);

            PageRank.setReducerClass(ConnectedComponentsReduer.class);
            PageRank.setOutputKeyClass(IntWritable.class);
            PageRank.setOutputValueClass(IntWritable.class);

            // Execute job
            code = PageRank.waitForCompletion(true) ? 0 : 1;

            //checking if the mean is stable
            BufferedReader file1Reader = new BufferedReader(new InputStreamReader
(hdfs.open(new Path(CACHED_PATH + "/part-r-00000"))));
            BufferedReader file2Reader = new BufferedReader(new InputStreamReader
(hdfs.open(new Path(ACTUAL_PATH + "/part-r-00000"))));
            for (int i = 0; i < verticesNum; i++) {
                double component1 = Double.parseDouble(file1Reader.readLine().spl
it(" ")[1]);
                double component2 = Double.parseDouble(file2Reader.readLine().spl
it(" ")[1]);

                if (component1 == component2) {
                    changed = false;
                } else {
                    changed = true;
                    break;
                }
            }
            file1Reader.close();
            file2Reader.close();
            counter++;
            System.out.println("ConnectedComponents finished iteration:>> " + cou
nter + " || component change: " + changed);
```

```
        }

        System.exit(code);

    }


    public static void writeFileByline(String dst, String contents) throws IOExce
ption {
        Configuration conf = new Configuration();
        Path dstPath = new Path(dst);
        FileSystem fs = dstPath.getFileSystem(conf);
        FSDataOutputStream outputStream = null;

        if (!fs.exists(dstPath)) {
            outputStream = fs.create(dstPath);
        } else {
            outputStream = fs.append(dstPath);
        }
        contents = contents + "\n";
        outputStream.write(contents.getBytes("utf-8"));
        outputStream.close();
    }

}
```

## ConnectedComponentsMapper

```
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

public class ConnectedComponentsMapper extends Mapper<Text, Text, IntWritable, I
ntWritable> {

    Map<Integer, Integer> components = new HashMap<>();

    /**
     * reading the rank from the distributed cache
     */
    public void setup(Context context) throws IOException, InterruptedException {
        String lineString = null;
        // read rank file
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);
```

```
        FSDataInputStream hdfsInStream = fs.open(new Path("output/cache/part-r-00
000"));
        InputStreamReader isr = new InputStreamReader(hdfsInStream, "utf-8");
        BufferedReader br = new BufferedReader(isr);

        while ((lineString = br.readLine()) != null) {
            String[] keyValue = StringUtils.split(lineString, " ");
            components.put(Integer.parseInt(keyValue[0]), Integer.parseInt(keyVal
ue[1]));

        }
        br.close();
    }

    public void map(Text from, Text to, Context context) throws IOException, Inte
rruptedException {
        int fromPoint = Integer.parseInt(from.toString());
        int toPoint = Integer.parseInt(to.toString());

        context.write(new IntWritable(toPoint), new IntWritable(components.get(fr
omPoint)));
        context.write(new IntWritable(fromPoint), new IntWritable(components.get
(fromPoint)));
    }

}
```

## ConnectedComponentsReduer

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ConnectedComponentsReduer extends Reducer<IntWritable, IntWritable,
 IntWritable, IntWritable> {


    public void reduce(IntWritable key, Iterable<IntWritable> values, Context con
text) throws IOException,
            InterruptedException {

        Configuration conf = context.getConfiguration();
        int component = Integer.parseInt(conf.get("verticesNum"));

        for (IntWritable value : values) {
            if (value.get() < component)
                component = value.get();
        }

        context.write(key, new IntWritable(component));
    }
}
```

思路：

1. 与 PageRank 一样，需要准备 cache 文件作为初始化连通分量，每次得到新的结果与 cache 文件进行比较，如果有更新则继续迭代
2. 在 map 中，为了保证每个点都会出现在 reduce 中，将 `from` 点和 `to` 点都输入到 reduce 中

## 运行

```
hadoop jar ConnectedComponents.jar input/ConnectedComponents/edges.txt 16
```

迭代了 6 次：

```
                Bytes Read=79
        File Output Format Counters
                Bytes Written=72
ConnectedComponents finished iteration:>> 6 || component change: false
```

```
hdfs dfs -cat output/Graph/HadoopConnectedComponents/*
```

最后结果为：

```
hadoop@scott:~$ hdfs dfs -cat output/Graph/HadoopConnectedComponents/*
1 1
2 1
3 1
4 1
5 1
6 6
7 6
8 8
9 8
```

# SingleSourceShortestPaths

## 数据准备

首先我们需要准备边和点

边：

```
1 2 12.0
1 3 13.0
2 3 23.0
3 4 34.0
3 5 35.0
4 5 45.0
5 1 51.0
```

放入 HDFS：

```
hdfs dfs -mkdir input/SingleSourceShortestPaths
hdfs dfs -put edges.txt input/SingleSourceShortestPaths
```

# 编写程序

## SingleSourceShortestPaths

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URISyntaxException;

import static java.lang.StrictMath.abs;


public class SingleSourceShortestPaths {

    private static final String CACHED_PATH = "output/cache";
    private static final String ACTUAL_PATH = "output/Graph/HadoopSingleSourceSho
rtestPaths";
    public static final int maxIterations = 100;
    private static final double EPSILON = 0.0001;
    public static int sourcePoint = 1;

    public static void main(String[] args) throws IOException,
            InterruptedException, ClassNotFoundException, URISyntaxException {

        Configuration conf = new Configuration();
        String[] otherArgs = (new GenericOptionsParser(conf, args)).getRemainingA
rgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: PageRank <EdgesPath> <verticesNum>");
            System.exit(2);
        }

        int code = 0;

        Path EdgesPath = new Path(otherArgs[0]);
        int verticesNum = Integer.parseInt(otherArgs[1]);

        conf.set("verticesNum", verticesNum + "");

        Path cachePath = new Path(CACHED_PATH);
        Path actualPath = new Path(ACTUAL_PATH);

        // Delete output if exists
```

```
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(actualPath))
            hdfs.delete(actualPath, true); // recursive delete

        // prepare original distance
        for (int i = 1; i <= verticesNum; i++) {
            if (i == sourcePoint)
                writeFileByline(ACTUAL_PATH + "/part-r-00000", i + " " + 0.0);
            else
                writeFileByline(ACTUAL_PATH + "/part-r-00000", i + " " + Double.PO
SITIVE_INFINITY);
        }


        int counter = 0;
        boolean changed = true;

        while (counter < maxIterations && changed) {

            // Delete output if exists
            if (hdfs.exists(cachePath))
                hdfs.delete(cachePath, true);
            //moving the previous iteration file to the cache directory
            hdfs.rename(actualPath, cachePath);

            conf.set("mapreduce.output.textoutputformat.separator", " ");

            Job PageRank = Job.getInstance(conf, "SingleSourceShortestPaths " +
(counter + ""));


            PageRank.setJarByClass(SingleSourceShortestPaths.class);
            FileInputFormat.addInputPath(PageRank, EdgesPath);
            FileOutputFormat.setOutputPath(PageRank, actualPath);

            PageRank.setMapperClass(SingleSourceShortestPathsMapper.class);
            PageRank.setMapOutputKeyClass(IntWritable.class);
            PageRank.setMapOutputValueClass(DoubleWritable.class);

            PageRank.setReducerClass(SingleSourceShortestPathsReducer.class);
            PageRank.setOutputKeyClass(IntWritable.class);
            PageRank.setOutputValueClass(DoubleWritable.class);

            // Execute job
            code = PageRank.waitForCompletion(true) ? 0 : 1;

            //checking if the mean is stable
            BufferedReader file1Reader = new BufferedReader(new InputStreamReader
(hdfs.open(new Path(CACHED_PATH + "/part-r-00000"))));
            BufferedReader file2Reader = new BufferedReader(new InputStreamReader
(hdfs.open(new Path(ACTUAL_PATH + "/part-r-00000"))));
            for (int i = 0; i < verticesNum; i++) {
                double distance1 = Double.parseDouble(file1Reader.readLine().spli
t(" ")[1]);
```

```
                double distance2 = Double.parseDouble(file2Reader.readLine().spli
t(" ")[1]);

                if (abs(distance1 - distance2) < EPSILON) {
                    changed = false;
                } else {
                    changed = true;
                    break;
                }
            }
            file1Reader.close();
            file2Reader.close();
            counter++;
            System.out.println("SingleSourceShortestPaths finished iteration:>> "
 + counter + " || distance change: " + changed);

        }

        System.exit(code);

    }


    public static void writeFileByline(String dst, String contents) throws IOExce
ption {
        Configuration conf = new Configuration();
        Path dstPath = new Path(dst);
        FileSystem fs = dstPath.getFileSystem(conf);
        FSDataOutputStream outputStream = null;

        if (!fs.exists(dstPath)) {
            outputStream = fs.create(dstPath);
        } else {
            outputStream = fs.append(dstPath);
        }
        contents = contents + "\n";
        outputStream.write(contents.getBytes("utf-8"));
        outputStream.close();
    }

}
```

## SingleSourceShortestPathsMapper

```
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.BufferedReader;
```

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

public class SingleSourceShortestPathsMapper extends Mapper<Object, Text, IntWri
table, DoubleWritable> {

    Map<Integer, Double> PointDistance = new HashMap<>();

    /**
     * reading the rank from the distributed cache
     */
    public void setup(Context context) throws IOException, InterruptedException {
        String lineString = null;
        // read rank file
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);
        FSDataInputStream hdfsInStream = fs.open(new Path("output/cache/part-r-00
000"));
        InputStreamReader isr = new InputStreamReader(hdfsInStream, "utf-8");
        BufferedReader br = new BufferedReader(isr);

        while ((lineString = br.readLine()) != null) {
            String[] keyValue = StringUtils.split(lineString, " ");
            PointDistance.put(Integer.parseInt(keyValue[0]), Double.parseDouble(k
eyValue[1]));

        }
        br.close();
    }

    public void map(Object object, Text line, Context context) throws IOException,
 InterruptedException {

        String[] lineData = line.toString().split(" ");

        int fromPoint = Integer.parseInt(lineData[0]);
        int toPoint = Integer.parseInt(lineData[1]);
        double distance = Double.parseDouble(lineData[2]);

        if (distance < Double.POSITIVE_INFINITY) {
            context.write(new IntWritable(toPoint), new DoubleWritable(PointDista
nce.get(fromPoint) + distance));
            context.write(new IntWritable(fromPoint), new DoubleWritable(PointDis
tance.get(fromPoint)));
        } else
            context.write(new IntWritable(toPoint), new DoubleWritable(Double.POS
ITIVE_INFINITY));
    }
}
```

## SingleSourceShortestPathsReducer

```
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
```

```
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class SingleSourceShortestPathsReducer extends Reducer<IntWritable, Doubl
eWritable, IntWritable, DoubleWritable> {


    public void reduce(IntWritable key, Iterable<DoubleWritable> values, Context
context) throws IOException,
            InterruptedException {

        double dis = Double.POSITIVE_INFINITY;

        for (DoubleWritable value : values) {
            if (value.get() < dis)
                dis = value.get();
        }

        context.write(key, new DoubleWritable(dis));
    }
}
```

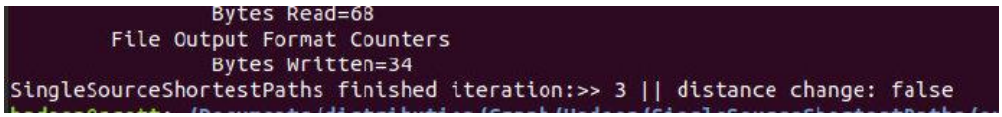
思想：

1. 主要想法和之前一样，不再赘述
2. 需要注意的是，每次 map 需要把前一次的结果也发给 reduce 进行比较，不然 reduce 出来的点个数会变少（例如原点就不会有）

## 运行

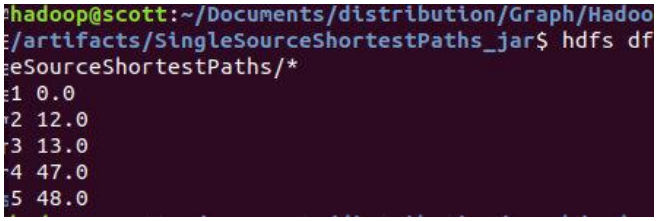
```
hadoop jar SingleSourceShortestPaths.jar input/SingleSourceShortestPaths/edges.t
xt 5
```

一共迭代了 4 次：




查看结果

```
hdfs dfs -cat output/Graph/HadoopSingleSourceShortestPaths/*
```




# 常用图算法实现--Spark

## PageRank

### 数据准备

边：

```
1 2
1 15
2 3
2 4
2 5
2 6
2 7
3 13
4 2
5 11
5 12
6 1
6 7
6 8
7 1
7 8
8 1
8 9
8 10
9 14
9 1
10 1
10 13
11 12
11 1
12 1
13 14
14 12
15 1
```

网页：

```
1
2
3
4
```

```
5
6
7
8
9
10
11
12
13
14
15
```

将这两个文件放入 HDFS：

```
hdfs dfs -mkdir input/PageRank
hdfs dfs -put links.txt input/PageRank
hdfs dfs -put pages.txt input/PageRank
```

## 编写程序

```
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.*;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.api.java.function.PairFunction;
import scala.Tuple2;

import static java.lang.Math.abs;


public class PageRank {
    private static int MaxIteration = 100;
    private static final double DAMPENING_FACTOR = 0.85;
    private static final double EPSILON = 0.0001;

    public static void main(String[] args) {
        SparkConf conf = new SparkConf().setAppName("PageRank");
        JavaSparkContext sc = new JavaSparkContext(conf);
        sc.setLogLevel("WARN");

        String linksFile = "hdfs:///user/hadoop/input/PageRank/links.txt";
        String pagesFile = "hdfs:///user/hadoop/input/PageRank/pages.txt";
        String rankFile = "hdfs:///user/hadoop/output/Graph/SparkPageRank";

        /**
         *  neighborRDD: (from, s)
         *  linksRDD: tuple (from, [to,1/m])
         *  pageRDD: vertex
         *  pageRankRDD: (point, 1/n)
         */


        JavaPairRDD<Integer, Integer> neighborRDD = sc.textFile(linksFile)
                .mapToPair(
                        line -> new Tuple2<>(
                                Integer.parseInt(line.split(" ")[0]), 1))
```

```
                .reduceByKey((x, y) -> x + y);

        JavaPairRDD<Integer, Tuple2<Integer, Integer>> linksRDD = sc.textFile(lin
ksFile)
                .mapToPair(
                        line -> new Tuple2<>(
                                Integer.parseInt(line.split(" ")[0]),
                                Integer.parseInt(line.split(" ")[1])
                        ))
                .join(neighborRDD);

        JavaRDD<Integer> pagesRDD = sc.textFile(pagesFile).map(line -> Integer.pa
rseInt(line));
        long pageCount = pagesRDD.count();
        JavaPairRDD<Integer, Double> pageRankRDD = pagesRDD.mapToPair(
                vertex -> new Tuple2<>(vertex, 1.0 / pageCount)
        );

        int count = 0;
        while (count < MaxIteration) {
            JavaPairRDD<Integer, Double> NewPageRankRDD = linksRDD.join(pageRankR
DD)
                    .mapToPair(
                            new PairFunction<Tuple2<Integer, Tuple2<Tuple2<Integer,
 Integer>, Double>>, Integer, Double>() {
                                @Override
                                public Tuple2<Integer, Double> call(Tuple2<Integer,
 Tuple2<Tuple2<Integer, Integer>, Double>> ans) throws Exception {
//                                    // [ toNode, fraction * rank]
                                    return new Tuple2<>(ans._2._1._1, ans._2._2/ans.
_2._1._2);
                                }
                            })
                    .reduceByKey((v1, v2) -> v1 + v2)
                    .mapValues(
                            new Function<Double, Double>() {
                                double dampening = DAMPENING_FACTOR;
                                double randomJump = (1 - DAMPENING_FACTOR) / pageCo
unt;

                                @Override
                                public Double call(Double value) throws Exception {
                                    value = value * dampening + randomJump;
                                    return value;
                                }
                            }
                    );
            count++;
            JavaPairRDD<Integer, Tuple2<Double, Double>> compare = pageRankRDD.jo
in(NewPageRankRDD).filter(each -> abs(each._2._1 - each._2._2) > EPSILON);
            if (compare.isEmpty() || count > MaxIteration)
                break;
            pageRankRDD = NewPageRankRDD;
```

```
        }

        pageRankRDD.saveAsTextFile(rankFile);

    }
}
```

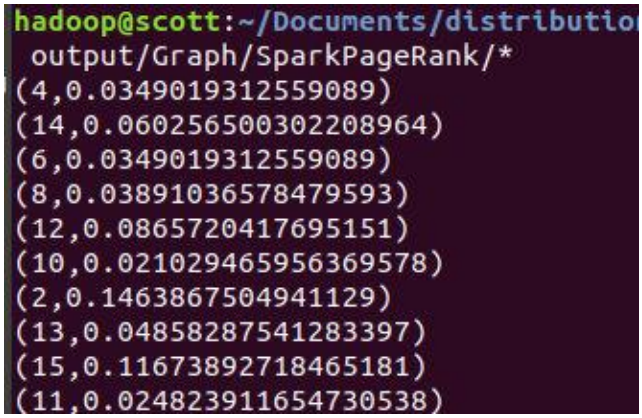
思路：

3. 全部使用 Lambda 表达式进行，首先需要找到所有的边的条数，初始化 Rank 值
4. 然后使用 Join 进行合并，并计算下一轮 Rank
5. 使用 DAMPENING_FACTOR 进行随机跳转

## 运行

```
spark-submit  --class PageRank PageRank-1.0.jar
hdfs dfs -cat output/Graph/SparkPageRank/*
```

结果为：




# ConnectedComponents

## 数据准备

提供基本数据集，与 PageRank 一样，指定顶点和边

### vertices.txt

准备一些顶点，例如 1-16

### edges.txt

准备一些连接边：

```
1 2
2 3
2 4
```

```
3 5
6 7
8 9
8 10
5 11
11 12
10 13
9 14
13 14
1 15
16 1
```

将这两个文件放入 HDFS：

```
hdfs dfs -mkdir input/ConnectedComponents
hdfs dfs -put edges.txt input/ConnectedComponents
hdfs dfs -put vertices.txt input/ConnectedComponents
```

## 编写程序

```
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaSparkContext;
import scala.Tuple2;

import static java.lang.StrictMath.min;

public class ConnectedComponents {

    public static int MaxIteration = 100;

    public static void main(String[] args) {
        SparkConf conf = new SparkConf().setAppName("ConnectedComponents");
        JavaSparkContext sc = new JavaSparkContext(conf);
        sc.setLogLevel("WARN");

        String edgesFile = "hdfs:///user/hadoop/input/ConnectedComponents/edges.t
xt";
        String verticesFile = "hdfs:///user/hadoop/input/ConnectedComponents/vert
ices.txt";
        String outFile = "hdfs:///user/hadoop/output/Graph/SparkConnectedComponen
ts";


        /**
         * edgesRDD: [x,y]
         * componentsRDD: [x,x] init
         */

        JavaPairRDD<Integer, Integer> edgesRDD = sc.textFile(edgesFile)
                .mapToPair(
                        line -> new Tuple2<>(
                                Integer.parseInt(line.split(" ")[0]),
                                Integer.parseInt(line.split(" ")[1])
                        )
```

```
                );

        JavaPairRDD<Integer, Integer> componentsRDD = sc.textFile(verticesFile)
                .mapToPair(
                        line -> new Tuple2<>(Integer.parseInt(line), Integer.parse
Int(line))
                );

        int count = 0;

        while (count < MaxIteration) {
            JavaPairRDD<Integer, Integer> newcomponentsRDD = componentsRDD.join(e
dgesRDD)
                    .mapToPair(
                            x -> new Tuple2<>(x._2._2, x._2._1)
                    )
                    .reduceByKey(
                            (v1, v2) -> min(v1, v2)
                    );

            JavaPairRDD<Integer, Tuple2<Integer, Integer>> filterRDD = newcompone
ntsRDD.join(componentsRDD)
                    .filter(
                            each -> each._2._1 < each._2._2
                    );

            if (filterRDD.isEmpty())
                break;

            // update to componentsRDD
            componentsRDD = componentsRDD.leftOuterJoin(newcomponentsRDD).
                    mapValues(
                            v -> min(v._1, v._2.orElse(v._1))
                    );

            count++;
        }

        componentsRDD.saveAsTextFile(outFile);
    }
}
```

思路：

6. 首先需要将每个点映射成自己的强连通分支
7. 每次迭代，更新与自己相连的点的强连通分支，取最小值
8. 使用左连接更新原始的强连通分支

## 运行

```
spark-submit  --class ConnectedComponents ConnectedComponents-1.0.jar
hdfs dfs -cat output/Graph/SparkConnectedComponents/*
```

查看结果：

```
hadoop@scott:~$ hdfs dfs -cat output/Graph/SparkConnectedComponents/*
(4,1)
(16,16)
(14,8)
(6,6)
(8,8)
(12,1)
```

## SingleSourceShortestPaths

### 数据准备

首先我们需要准备边和点

边：

```
1 2 12.0
1 3 13.0
2 3 23.0
3 4 34.0
3 5 35.0
4 5 45.0
5 1 51.0
```

点：

```
1
2
3
4
5
```

将这两个文件放入 HDFS：

```
hdfs dfs -mkdir input/SingleSourceShortestPaths
hdfs dfs -put edges.txt input/SingleSourceShortestPaths
hdfs dfs -put vertices.txt input/SingleSourceShortestPaths
```

### 编写程序

```
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaSparkContext;
import scala.Tuple2;

import javax.validation.constraints.Max;

import static java.lang.StrictMath.min;

public class SingleSourceShortestPaths {
    public static int sourceVerticeID = 1;
    public static int MaxIteration = 100;
```

```
    public static void main(String[] args) throws Exception {
        SparkConf conf = new SparkConf().setAppName("ConnectedComponents");
        JavaSparkContext sc = new JavaSparkContext(conf);
        sc.setLogLevel("WARN");

        String edgesFile = "hdfs:///user/hadoop/input/SingleSourceShortestPaths/edges.txt";
        String verticesFile = "hdfs:///user/hadoop/input/SingleSourceShortestPaths/vertices.txt";
        String outFile = "hdfs:///user/hadoop/output/Graph/SparkSingleSourceShortestPaths";

        /**
         * edgesRDD: [from, to, dis ]
         * verticesRDD: [vertice, dis]
         */


        JavaPairRDD<Integer, Tuple2<Integer, Double>> edgesRDD = sc.textFile(edgesFile)
                .mapToPair(
                        line -> {
                            int from = Integer.parseInt(line.split(" ")[0]);
                            int to = Integer.parseInt(line.split(" ")[1]);
                            double dis = Double.parseDouble(line.split(" ")[2]);
                            return new Tuple2<>(from, new Tuple2<>(to, dis));
                        }
                );

        JavaPairRDD<Integer, Double> verticesRDD = sc.textFile(verticesFile)
                .mapToPair(
                        line -> {
                            int vertice = Integer.parseInt(line);
                            if (vertice == sourceVerticeID)
                                return new Tuple2<>(vertice, 0.0);
                            return new Tuple2<>(vertice, Double.POSITIVE_INFINITY);
                        }
                );

        int count = 0;
        while (count < MaxIteration) {
            // get new dis
            JavaPairRDD<Integer, Double> newVerticesRDD = verticesRDD
                    .join(edgesRDD)
                    .mapToPair(
                            line -> {
                                if (line._2._1 != Double.POSITIVE_INFINITY)
                                    return new Tuple2<>(line._2._2._1, line._2._1 + line._2._2._2);
                                return new Tuple2<>(line._2._2._1, Double.POSITIVE_INFINITY);
                            }
                    ).reduceByKey(
```

```
                        (v1, v2) -> min(v1, v2));

            JavaPairRDD<Integer, Tuple2<Double, Double>> filterRDD = newVerticesR
DD.join(verticesRDD)
                    .filter(
                            each -> each._2._1 < each._2._2);

            if (filterRDD.isEmpty())
                break;

            // update to verticesRDD
            verticesRDD = verticesRDD.leftOuterJoin(newVerticesRDD).
                    mapValues(
                            v -> min(v._1, v._2.orElse(v._1)));
        }
        verticesRDD.saveAsTextFile(outFile);

    }
}
```

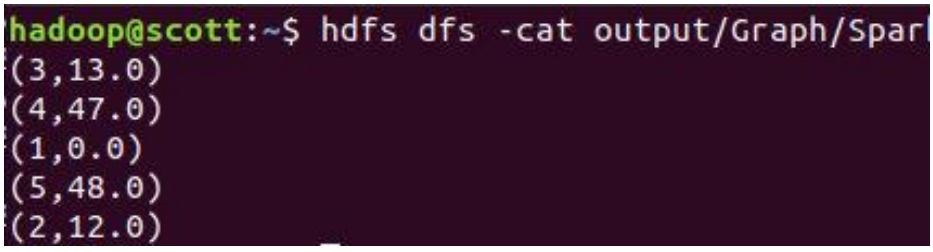
思路：

9. 首先需要初始化每个顶点的距离，将原始点设置为 0，其余设置为无穷
10. 每次迭代得到新的顶点距离，并使用 reduceByKey 最小化，比较是否更新
11. 然后将更新得到的顶点距离加入原始 RDD 中

## 运行

```
spark-submit  --class SingleSourceShortestPaths SingleSourceShortestPaths-1.0.ja
r
hdfs dfs -cat output/Graph/SparkSingleSourceShortestPaths/*
```

查看结果：




# 常用图算法实现--Flink

## PageRank

主要参考官网的 example

# 算法流程

每次计算当前每个网页的转移概率，计算下一时刻到达每个网页的概率并加入随机跳转

# 数据准备

## pages.txt

准备一些顶点，例如 1-15

## links.txt

准备一些连接边（也就是链接数）：

```
1 2
1 15
2 3
2 4
2 5
2 6
2 7
3 13
4 2
5 11
5 12
6 1
6 7
6 8
7 1
7 8
8 1
8 9
8 10
```

# PageRank.java

```
@SuppressWarnings("serial")
public class PageRank {

    private static final double DAMPENING_FACTOR = 0.85;
    private static final double EPSILON = 0.0001;

    // ************************************************************************
    //     PROGRAM
    // ************************************************************************

    public static void main(String[] args) throws Exception {

        ParameterTool params = ParameterTool.fromArgs(args);

        final int numPages = params.getInt("numPages", PageRankData.getNumberOfPa
ges());
        final int maxIterations = params.getInt("iterations", 10);
```

```
        // set up execution environment
        final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnviron
ment();

        // make the parameters available to the web ui
        env.getConfig().setGlobalJobParameters(params);

        // get input data
        DataSet<Long> pagesInput = getPagesDataSet(env, params);
        DataSet<Tuple2<Long, Long>> linksInput = getLinksDataSet(env, params);

        // assign initial rank to pages pi = ([1,1/n] ,... [n,1/n])
        DataSet<Tuple2<Long, Double>> pagesWithRanks = pagesInput.
                map(new RankAssigner((1.0d / numPages)));

        // build adjacency list from link input (1,[2,3,5])...
        DataSet<Tuple2<Long, Long[]>> adjacencyListInput =
                linksInput.groupBy(0).reduceGroup(new BuildOutgoingEdgeList());

        // set iterative data set
        IterativeDataSet<Tuple2<Long, Double>> iteration = pagesWithRanks.iterate
(maxIterations);

        DataSet<Tuple2<Long, Double>> newRanks = iteration
                // join pages with outgoing edges and distribute rank [1,1/n] join
 1,[1,3,5] => [1,1/3n],[3,1/3n],[5,1/3n]
                .join(adjacencyListInput).where(0).equalTo(0).flatMap(new JoinVer
texWithEdgesMatch())
                // collect and sum ranks
                .groupBy(0).aggregate(SUM, 1)
                // apply dampening factor choosing stay or leave
                .map(new Dampener(DAMPENING_FACTOR, numPages));

        DataSet<Tuple2<Long, Double>> finalPageRanks = iteration.closeWith(
                newRanks,
                newRanks.join(iteration).where(0).equalTo(0)
                        // termination condition
                        .filter(new EpsilonFilter()));

        // emit result
        if (params.has("output")) {
            finalPageRanks.writeAsCsv(params.get("output"), "\n", " ");
            // execute program
            env.execute("Basic Page Rank Example");
        } else {
            System.out.println("Printing result to stdout. Use --output to specif
y output path.");
            finalPageRanks.print();
        }
    }

    // *************************************************************************
    //     USER FUNCTIONS
    // *************************************************************************
```

```
    /**
     * A map function that assigns an initial rank to all pages.
     */
    public static final class RankAssigner implements MapFunction<Long, Tuple2<Lo
ng, Double>> {
        Tuple2<Long, Double> outPageWithRank;

        public RankAssigner(double rank) {
            this.outPageWithRank = new Tuple2<Long, Double>(-1L, rank);
        }

        @Override
        public Tuple2<Long, Double> map(Long page) {
            outPageWithRank.f0 = page;
            return outPageWithRank;
        }
    }

    /**
     * A reduce function that takes a sequence of edges and builds the adjacency
list for the vertex where the edges
     * originate. Run as a pre-processing step.
     */
    @ForwardedFields("0")
    public static final class BuildOutgoingEdgeList implements GroupReduceFunctio
n<Tuple2<Long, Long>, Tuple2<Long, Long[]>> {

        private final ArrayList<Long> neighbors = new ArrayList<Long>();

        @Override
        public void reduce(Iterable<Tuple2<Long, Long>> values, Collector<Tuple2<
Long, Long[]>> out) {
            neighbors.clear();
            Long id = 0L;

            for (Tuple2<Long, Long> n : values) {
                id = n.f0;
                neighbors.add(n.f1);
            }
            out.collect(new Tuple2<Long, Long[]>(id, neighbors.toArray(new Long[n
eighbors.size()])));
        }
    }

    /**
     * Join function that distributes a fraction of a vertex's rank to all neighb
ors.
     */
    public static final class JoinVertexWithEdgesMatch implements FlatMapFunction
<Tuple2<Tuple2<Long, Double>, Tuple2<Long, Long[]>>, Tuple2<Long, Double>> {

        @Override
        public void flatMap(Tuple2<Tuple2<Long, Double>, Tuple2<Long, Long[]>> va
```

```
lue, Collector<Tuple2<Long, Double>> out){
            Long[] neighbors = value.f1.f1;
            double rank = value.f0.f1;
            double rankToDistribute = rank / ((double) neighbors.length);

            for (Long neighbor: neighbors) {
                out.collect(new Tuple2<Long, Double>(neighbor, rankToDistribute));
            }
        }
    }

    /**
     * The function that applies the page rank dampening formula.
     */
    @ForwardedFields("0")
    public static final class Dampener implements MapFunction<Tuple2<Long, Doubl
e>, Tuple2<Long, Double>> {

        private final double dampening;
        private final double randomJump;

        public Dampener(double dampening, double numVertices) {
            this.dampening = dampening;
            this.randomJump = (1 - dampening) / numVertices;
        }

        @Override
        public Tuple2<Long, Double> map(Tuple2<Long, Double> value) {
            value.f1 = (value.f1 * dampening) + randomJump;
            return value;
        }
    }

    /**
     * Filter that filters vertices where the rank difference is below a threshol
d.
     */
    public static final class EpsilonFilter implements FilterFunction<Tuple2<Tupl
e2<Long, Double>, Tuple2<Long, Double>>> {

        @Override
        public boolean filter(Tuple2<Tuple2<Long, Double>, Tuple2<Long, Double>>
value) {
            return Math.abs(value.f0.f1 - value.f1.f1) > EPSILON;
        }
    }

    // *************************************************************************
    //     UTIL METHODS
    // *************************************************************************

    private static DataSet<Long> getPagesDataSet(ExecutionEnvironment env, Param
eterTool params) {
        if (params.has("pages")) {
```

```
            return env.readCsvFile(params.get("pages"))
                    .fieldDelimiter(" ")
                    .lineDelimiter("\n")
                    .types(Long.class)
                    .map(new MapFunction<Tuple1<Long>, Long>() {
                        @Override
                        public Long map(Tuple1<Long> v) {
                            return v.f0;
                        }
                    });
        } else {
            System.out.println("Executing PageRank example with default pages dat
a set.");
            System.out.println("Use --pages to specify file input.");
            return PageRankData.getDefaultPagesDataSet(env);
        }
    }

    private static DataSet<Tuple2<Long, Long>> getLinksDataSet(ExecutionEnvironm
ent env, ParameterTool params) {
        if (params.has("links")) {
            return env.readCsvFile(params.get("links"))
                    .fieldDelimiter(" ")
                    .lineDelimiter("\n")
                    .types(Long.class, Long.class);
        } else {
            System.out.println("Executing PageRank example with default links dat
a set.");
            System.out.println("Use --links to specify file input.");
            return PageRankData.getDefaultEdgeDataSet(env);
        }
    }
}
```

## 注意点

12. 处理逻辑为：首先将输入数据转为邻接链表，然后迭代计算每一次的 Rank，再加上每一次 dampening（可能停留，可能随机），得到下一次的 Rank

13. 最后在 closeWith 中与前一次的 Rank 值进行对比，小于阈值则退出循环

## 运行

打包成 Jar 包，并执行：

```
flink run -c PageRank PageRank.jar --links /home/hadoop/Documents/distribution/F
link/PageRank/links.txt --pages /home/hadoop/Documents/distribution/Flink/PageRa
nk/pages.txt
```

结果为：

```
Printing result to stdout. Use --out
(1,0.2507740474831439)
(2,0.14680796376156166)
(3,0.03476167589543225)
(4,0.03476167589543225)
(5,0.03476167589543225)
(6,0.03476167589543225)
(7,0.04464948608422618)
(8,0.03891448491322475)
(9,0.021024659120945137)
```

## ConnectedComponents

# 数据准备

提供基本数据集，与 PageRank 一样，指定顶点和边

### vertices.txt

准备一些顶点，例如 1-16

### edges.txt

准备一些连接边：

```
1 2
2 3
2 4
3 5
6 7
8 9
8 10
5 11
11 12
10 13
9 14
13 14
1 15
16 1
```

## ConnectedComponents.java

```
import org.apache.flink.api.common.functions.FlatJoinFunction;
import org.apache.flink.api.common.functions.FlatMapFunction;
import org.apache.flink.api.common.functions.JoinFunction;
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.DataSet;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.api.java.aggregation.Aggregations;
import org.apache.flink.api.java.functions.FunctionAnnotation.ForwardedFields;
import org.apache.flink.api.java.functions.FunctionAnnotation.ForwardedFieldsFir
st;
import org.apache.flink.api.java.functions.FunctionAnnotation.ForwardedFieldsSec
ond;
import org.apache.flink.api.java.operators.DeltaIteration;
```

```
import org.apache.flink.api.java.tuple.Tuple1;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.util.Collector;


@SuppressWarnings("serial")
public class ConnectedComponents {

    // *************************************************************************
    //     PROGRAM
    // *************************************************************************

    public static void main(String... args) throws Exception {

        // Checking input parameters
        final ParameterTool params = ParameterTool.fromArgs(args);

        // set up execution environment
        ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

        final int maxIterations = params.getInt("iterations", 10);

        // make parameters available in the web interface
        env.getConfig().setGlobalJobParameters(params);

        // read vertex and edge data
        DataSet<Long> vertices = getVertexDataSet(env, params);
        DataSet<Tuple2<Long, Long>> edges = getEdgeDataSet(env, params).flatMap(n
ew UndirectEdge());

        // assign the initial components (equal to the vertex id) [1,1],[2,2]
        DataSet<Tuple2<Long, Long>> verticesWithInitialId =
                vertices.map(new DuplicateValue<Long>());

        // open a delta iteration
        DeltaIteration<Tuple2<Long, Long>, Tuple2<Long, Long>> iteration =
                verticesWithInitialId.iterateDelta(verticesWithInitialId, maxIter
ations, 0);

        // apply the step logic: join with the edges, select the minimum neighbor,
 update if the component of the candidate is smaller
        DataSet<Tuple2<Long, Long>> changes = iteration.getWorkset().join(edges).
where(0).equalTo(0).with(new NeighborWithComponentIDJoin())
                .groupBy(0).aggregate(Aggregations.MIN, 1)
                .join(iteration.getSolutionSet()).where(0).equalTo(0)
                .with(new ComponentIdFilter());

        // close the delta iteration (delta and new workset are identical)
        DataSet<Tuple2<Long, Long>> result = iteration.closeWith(changes, change
s);

        // emit result
        if (params.has("output")) {
```

```
            result.writeAsCsv(params.get("output"), "\n", " ");
            // execute program
            env.execute("Connected Components Example");
        } else {
            System.out.println("Printing result to stdout. Use --output to specif
y output path.");
            result.print();
        }
    }

    // *************************************************************************
    //     USER FUNCTIONS
    // *************************************************************************

    /**
     * Function that turns a value into a 2-tuple where both fields are that valu
e.
     */
    @ForwardedFields("*->f0")
    public static final class DuplicateValue<T> implements MapFunction<T, Tuple2<
T, T>> {
        @Override
        public Tuple2<T, T> map(T vertex) {
            return new Tuple2<T, T>(vertex, vertex);
        }
    }

    /**
     * Undirected edges by emitting for each input edge the input edges itself an
d an inverted version.
     */
    public static final class UndirectEdge implements FlatMapFunction<Tuple2<Long,
 Long>, Tuple2<Long, Long>> {
        Tuple2<Long, Long> invertedEdge = new Tuple2<Long, Long>();

        @Override
        public void flatMap(Tuple2<Long, Long> edge, Collector<Tuple2<Long, Lon
g>> out) {
            invertedEdge.f0 = edge.f1;
            invertedEdge.f1 = edge.f0;
            out.collect(edge);
            out.collect(invertedEdge);
        }
    }

    /**
     * UDF that joins a (Vertex-ID, Component-ID) pair that represents the curren
t component that
     * a vertex is associated with, with a (Source-Vertex-ID, Target-VertexID) ed
ge. The function
     * produces a (Target-vertex-ID, Component-ID) pair.
     */
    @ForwardedFieldsFirst("f1->f1")
    @ForwardedFieldsSecond("f1->f0")
```

```
    public static final class NeighborWithComponentIDJoin implements JoinFunction
<Tuple2<Long, Long>, Tuple2<Long, Long>, Tuple2<Long, Long>> {

        @Override
        public Tuple2<Long, Long> join(Tuple2<Long, Long> vertexWithComponent, Tu
ple2<Long, Long> edge) {
            return new Tuple2<Long, Long>(edge.f1, vertexWithComponent.f1);
        }
    }

    /**
     * Emit the candidate (Vertex-ID, Component-ID) pair if and only if the
     * candidate component ID is less than the vertex's current component ID.
     */
    @ForwardedFieldsFirst("*")
    public static final class ComponentIdFilter implements FlatJoinFunction<Tuple
2<Long, Long>, Tuple2<Long, Long>, Tuple2<Long, Long>> {

        @Override
        public void join(Tuple2<Long, Long> candidate, Tuple2<Long, Long> old, Co
llector<Tuple2<Long, Long>> out) {
            if (candidate.f1 < old.f1) {
                out.collect(candidate);
            }
        }
    }

    // *************************************************************************
    //     UTIL METHODS
    // *************************************************************************

    private static DataSet<Long> getVertexDataSet(ExecutionEnvironment env, Para
meterTool params) {
        if (params.has("vertices")) {
            return env.readCsvFile(params.get("vertices")).types(Long.class).map(
                    new MapFunction<Tuple1<Long>, Long>() {
                        public Long map(Tuple1<Long> value) {
                            return value.f0;
                        }
                    });
        } else {
            System.out.println("Executing Connected Components example with defau
lt vertices data set.");
            System.out.println("Use --vertices to specify file input.");
            return ConnectedComponentsData.getDefaultVertexDataSet(env);
        }
    }

    private static DataSet<Tuple2<Long, Long>> getEdgeDataSet(ExecutionEnvironme
nt env, ParameterTool params) {
        if (params.has("edges")) {
            return env.readCsvFile(params.get("edges")).fieldDelimiter(" ").types
(Long.class, Long.class);
        } else {
```

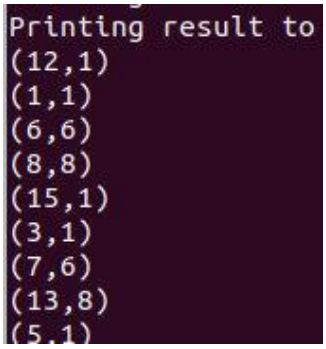
```
            System.out.println("Executing Connected Components example with defau
lt edges data set.");
            System.out.println("Use --edges to specify file input.");
            return ConnectedComponentsData.getDefaultEdgeDataSet(env);
        }
    }
}
```

## 注意点

14. 首先将每个点映射成（id，id），表示初始化每个点都是自己的连通分量。
15. 对当前的连通分量与边进行 join，得到(Target-vertex-ID，Component-ID)的 pair，并保留最小的 ID 作为当前的连通分量。
16. 在 DeltaIteration 中，将 WorkSet 计算得到的新的强连通分量与 SolutionSet 进行比较，得到 changes，若 changes 存在（不为空），则继续迭代，同时，将 changes 传给 SolutionSet 和 WorkSet。

## 运行

```
flink run -c ConnectedComponents ConnectedComponents.jar --edges /home/hadoop/Do
cuments/distribution/Flink/ConnectedComponents/edges.txt --vertices /home/hadoop
/Documents/distribution/Flink/ConnectedComponents/vertices.txt
```




# SingleSourceShortestPaths

## 数据准备

首先我们需要准备边和点

边：

```
1 2 12.0
1 3 13.0
2 3 23.0
3 4 34.0
3 5 35.0
4 5 45.0
5 1 51.0
```

点：

1
2
3
4
5

## SingleSourceShortestPaths.java

```
import org.apache.flink.api.common.functions.FlatJoinFunction;
import org.apache.flink.api.common.functions.JoinFunction;
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.api.java.DataSet;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.api.java.aggregation.Aggregations;
import org.apache.flink.api.java.functions.FunctionAnnotation;
import org.apache.flink.api.java.operators.DeltaIteration;
import org.apache.flink.api.java.tuple.Tuple1;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.util.Collector;

@SuppressWarnings("serial")
public class SingleSourceShortestPaths {
    public static int sourceVerticeID = 1;

    public static void main(String[] args) throws Exception {


        final ParameterTool params = ParameterTool.fromArgs(args);

        ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

        DataSet<Tuple3<Integer, Integer, Double>> edges = getEdgesDataSet(params,
 env);
        DataSet<Tuple2<Integer, Double>> vertices = getVerticesDataSet(params, en
v);

        DeltaIteration<Tuple2<Integer, Double>, Tuple2<Integer, Double>> iteratio
n = vertices
                .iterateDelta(vertices, 100,0);


        DataSet<Tuple2<Integer, Double>> NewSolutionSet = iteration.getWorkset()
                .join(edges).where(0).equalTo(0)
                .with(new FindDistance())
                .groupBy(0).aggregate(Aggregations.MIN, 1)
                .join(iteration.getSolutionSet()).where(0).equalTo(0)
                .with(new DistanceFilter());

        // close the delta iteration (changes are empty)
        DataSet<Tuple2<Integer, Double>> result = iteration.closeWith(NewSolution
Set, NewSolutionSet);
```

```
        // emit result
        if (params.has("output")) {
            result.writeAsCsv(params.get("output"), "\n", " ");
            // execute program
            env.execute("Connected Components Example");
        } else {
            System.out.println("Printing result to stdout. Use --output to specif
y output path.");
            result.print();
        }

    }


    public static final class DistanceFilter implements FlatJoinFunction<Tuple2<I
nteger, Double>, Tuple2<Integer, Double>, Tuple2<Integer, Double>> {
        @Override
        public void join(Tuple2<Integer, Double> candidate, Tuple2<Integer, Doubl
e> old, Collector<Tuple2<Integer, Double>> out) throws Exception {
            if (candidate.f1 < old.f1)
                out.collect(candidate);
        }
    }


    /**
     * (from,to,dis) join (point,dis)
     * produces a (point, distance) pair.
     */
    @FunctionAnnotation.ForwardedFieldsSecond("f1->f0")
    public static final class FindDistance implements JoinFunction<Tuple2<Integer,
 Double>, Tuple3<Integer, Integer, Double>, Tuple2<Integer, Double>> {
        @Override
        public Tuple2<Integer, Double> join(Tuple2<Integer, Double> vertices, Tup
le3<Integer, Integer, Double> edges) throws Exception {
            return Tuple2.of(edges.f1, vertices.f1 < Double.POSITIVE_INFINITY ? v
ertices.f1 + edges.f2 : Double.POSITIVE_INFINITY);
        }
    }


    /**
     * Get Edges data
     *
     * @param params
     * @param env
     * @return
     */
    private static DataSet<Tuple3<Integer, Integer, Double>> getEdgesDataSet(Par
ameterTool params, ExecutionEnvironment env) {
        if (params.has("edges")) {
            return env.readCsvFile(params.get("edges"))
```

```
                    .fieldDelimiter(" ")
                    .types(Integer.class, Integer.class, Double.class);
        } else {
            return SingleSourceShortestPathsData.getDefaultEdgeDataSet(env);
        }
    }

    /**
     * Get Vertices data
     *
     * @param params
     * @param env
     * @return
     */
    private static DataSet<Tuple2<Integer, Double>> getVerticesDataSet(Parameter
Tool params, ExecutionEnvironment env) {
        DataSet<Integer> vertices;
        if (params.has("vertices")) {
            vertices = env.readCsvFile(params.get("vertices")).types(Integer.clas
s).map(
                    new MapFunction<Tuple1<Integer>, Integer>() {
                        public Integer map(Tuple1<Integer> value) {
                            return value.f0;
                        }
                    });
        } else
            vertices = env.fromElements(1, 2, 3, 4, 5);
        return vertices.map(new MapFunction<Integer, Tuple2<Integer, Double>>() {
            @Override
            public Tuple2<Integer, Double> map(Integer integer) throws Exception
{
                if (integer == sourceVerticeID)
                    return Tuple2.of(integer, 0.0);
                else
                    return Tuple2.of(integer, Double.POSITIVE_INFINITY);
            }
        });
    }
}
```

## 注意点：

17. 正确使用 DeltaIteration，分清楚 SolutionSet 和 WorkSet，其中，CloseWith 的第一个是要 merge 到 SolutionSet，第二个作为 WorkSet。

18. 通过比较是否有新的最短路径产生来结束循环

## 运行

默认数据运行：

```
flink run -c SingleSourceShortestPaths SingleSourceShortestPaths.jar
```

```
hadoop@scott:~/Documents/distribution/Flink/SingleSourceShortestPaths/out/artifa
cts/SingleSourceShortestPaths_jar$ flink run -c SingleSourceShortestPaths Single
SourceShortestPaths.jar --edges /home/hadoop/Documents/distribution/Flink/Single
SourceShortestPaths/edges.txt --vertices /home/hadoop/Documents/distribution/Fli
nk/SingleSourceShortestPaths/vertices.txt
Starting execution of program
Printing result to stdout. Use --output to specify output path.
(4,47.0)
(1,0.0)
(2,12.0)
(5,48.0)
(3,13.0)
Program execution finished
```

使用指定参数运行：

```
flink run -c SingleSourceShortestPaths SingleSourceShortestPaths.jar --edges /ho
me/hadoop/Documents/distribution/Flink/SingleSourceShortestPaths/edges.txt --ver
tices /home/hadoop/Documents/distribution/Flink/SingleSourceShortestPaths/vertic
es.txt
```

```
hadoop@scott:~/Documents/distribution/Flink/SingleSourceShortestPaths/out/artifa
cts/SingleSourceShortestPaths_jar$ flink run -c SingleSourceShortestPaths Single
SourceShortestPaths.jar --edges /home/hadoop/Documents/distribution/Flink/Single
SourceShortestPaths/edges.txt --vertices /home/hadoop/Documents/distribution/Fli
nk/SingleSourceShortestPaths/vertices.txt
Starting execution of program
Printing result to stdout. Use --output to specify output path.
(4,47.0)
(1,0.0)
(2,12.0)
(5,48.0)
(3,13.0)
Program execution finished
```

## 五、总结

本次实验熟悉了图算法在分布式系统中的实现过程，并更加熟悉了 hadoop、spark、flink 编程，深切的体会了他们的差别和各自的优缺点。