

CPS 534: Distributed Computing with Big Data

Programming Assignment #2, 100 pts, 2 weeks

At most two students in a team. One submission per team

No late submission will be accepted

Receive 5 bonus points if turn in the complete work **without errors** at least one day before deadline

Receive an **F** for this course if any academic dishonesty occurs

1. Purpose

This project creates a MapReduce project for computing PageRank of each node on a large graph. In this assignment, “page” or “web page” means a node on the web graph.

2. Description

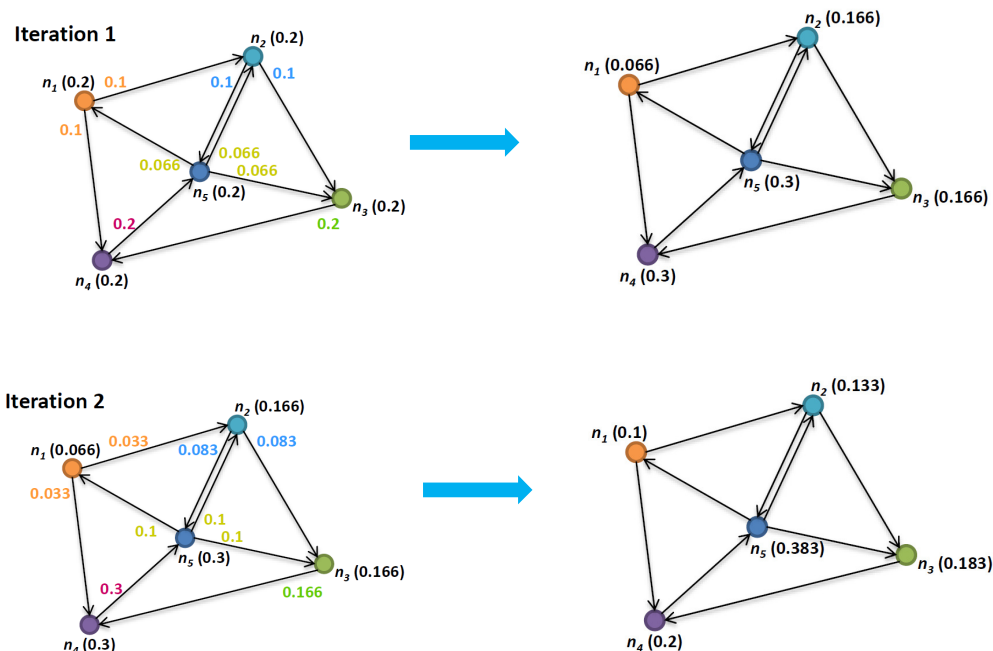
2.1 Basics (40 points)

PageRank is an algorithm used to compute the importance of each node on the graph. This algorithm is built upon a Markov chain that models user surfing on the web as users click hyperlinks on web pages to visit different web pages. The PageRank of page n (i.e., a number between 0 and 1 indicating the importance of page n) is computed based on this formula:

$$P(n) = \alpha \frac{1}{|V|} + (1 - \alpha) \sum_{m \in L(n)} \frac{P(m)}{C(m)}$$

where $0 \leq \alpha < 1$ is a constant (e.g., $\alpha = 0.1$), $L(n)$ is the set of in-coming neighbors of page n , $P(m)$ is the PageRank of node m , $C(m)$ is node m 's out-degree, and $|V|$ is the number of nodes on the graph.

As an example (from Lin and Dyer's book), the following figures show how node PageRank is recomputed at each iteration, where $\alpha = 0$:



Text Input format:

Each Line: nodeID<tab>PageRank<tab>adjacency-List

Sample input text file:

```
n1    0.2    n2,n4
n2    0.2    n3,n5
n3    0.2    n4
n4    0.2    n5
n5    0.2    n1,n2,n3
```

The output at the end of the first iteration should be:

```
n1    0.066  n2,n4
n2    0.166  n3,n5
n3    0.166  n4
n4    0.3     n5
n5    0.3     n1,n2,n3
```

which will be the input text file to the 2nd iteration. Note that the output may be split into several output files, determined by the number of reduce tasks in your project.

When debugging your code, you should use the example above (a small graph) to check if the output of each iteration is correct. Once you pass this step, move onto a larger graph (at isidore) and enjoy the power of parallel computing using VMs in clouds!

PageRank pseudocode is given in the following:

```
map(nid n, node N)    // N stores node's current PageRank and adjacency list
  p = N.pageRank / |N.adjacencyList|    // P(n)/C(n)
  emit(nid n, N)        // Pass along graph structure
  for all nid m in N.adjacencyList do
    emit(nid m, p)    // Pass PageRank mass to neighbors
```

```
reduce(nid m, [p1,p2,...])
  s = 0; M = ∅
  for all p in [p1,p2,...] do
    if isNode(p) then
      M = p    // Recover graph structure
    else
      s += p    // Sum incoming PageRank contributions
  M.pageRank =  $\alpha / |V| + (1 - \alpha)s$ 
  emit(nid m, node M)
```

2.2 Dangling Nodes (15 points)

However, the pseudocode above does not deal with dangling node (i.e., nodes without any outgoing links). To fix this, we should add a value δ (initial value is 0), representing missing PageRank mass of all dangling nodes. The PageRank equation is then updated to be:

$$P(n) = \alpha \frac{1}{|V|} + (1 - \alpha) \left(\frac{\delta}{|V|} + \sum_{m \in L(n)} \frac{P(m)}{C(m)} \right)$$

In your project, reserve a special key for storing PageRank mass from dangling nodes. When the mapper encounters a dangling node, its PageRank mass is emitted with the special key. The reducer must be modified to contain special logic for handling the missing PageRank mass.

2.3 Multiple Iterations (15 points)

PageRank computation iterates until convergence; that is, PageRank of all nodes remain the same (or within small tolerance). **In class Reduce**, we can define a counter used for all (scaled) changes:

```
public class MyReduce extends Reducer...
{
    public static enum Counter { // group name
        DELTAS // count name
    }
    ...
}
```

In reduce() method:

```
double delta = originalNode.getPageRank() - newPageRank;
int scaledDelta = Math.abs((int) (delta * CONVERGENCE_SCALING_FACTOR));
// e.g., CONVERGENCE_SCALING_FACTOR is 1000
context.getCounter(Counter.DELTAS).increment(scaledDelta);
```

In the driver, within a loop:

```
while (true)
{
    ...
    long summedConvergence =
        job.getCounters().findCounter(MyReduce.Counter.DELTAS).getValue();
    if (summedConvergence < desiredConvergence) // designedConvergence is fixed
        break; // done with all iterations
    ...
}
```

2.3 Submission requirements

In your report (word or pdf file), take snapshots on how you run .jar on the amazon hadoop cluster. Make sure you report the final PageRank value for each node on the graph.

You should zip the following:

1. The report (**explain α , δ , and how many iterations for each graph you've tested for your assignment, explain how you handle dangling nodes, snapshots of sample runs, final results, the fun part of this assignment, and the difficulties you encounter**)
2. A README text file (hadoop version, java version, how to run .jar)

3. All **.java** and **.jar** files.

No need to submit input text files. Submit the **zip** to isidore. One submission per team. Make sure you write team members' names in each file.

2.4 Grading Notes:

- 1) 30 points for the report
- 2) 70 points for code (40points for the basics)

3. Parallel Breadth First Search (PBFS)

This section shows the complete MapReduce code for finding shortest paths in parallel. Following steps in Assignment 1, you may create a Maven project for PBFS, export the jar package, upload files to aws, and then run the jar on the hadoop cluster at aws. This will help you understand how multiple iterations in PBFS works and get ready for your PageRank project.

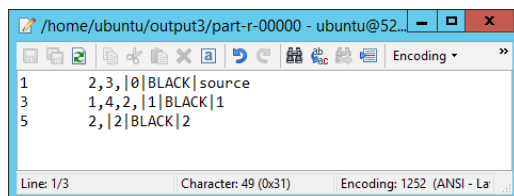
Text Input format:

Each Line: nodeID<tab>List_of_adjacent_nodes|distance_from_source|color|parent

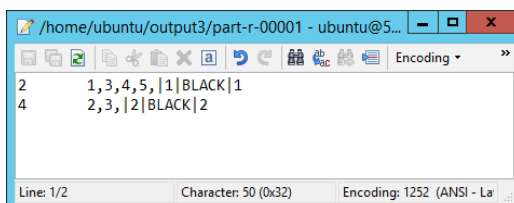
Sample input text file:

```
1<tab>2,3|0|GRAY|source
2<tab>1,3,4,5|Integer.MAX_VALUE|WHITE|null
3<tab>1,4,2|Integer.MAX_VALUE|WHITE|null
4<tab>2,3|Integer.MAX_VALUE|WHITE|null
5<tab>2|Integer.MAX_VALUE|WHITE|null
```

Sample output:



```
/home/ubuntu/output3/part-r-00000 - ubuntu@52...
1 2,3,|0|BLACK|source
3 1,4,2,|1|BLACK|1
5 2,|2|BLACK|2
Line: 1/3 Character: 49 (0x31) Encoding: 1252 (ANSI - La
```



```
/home/ubuntu/output3/part-r-00001 - ubuntu@5...
2 1,3,4,5,|1|BLACK|1
4 2,3,|2|BLACK|2
Line: 1/2 Character: 50 (0x32) Encoding: 1252 (ANSI - La
```

```

/* Node.java */

package YourPackageName;
import java.util.*;
import org.apache.hadoop.io.Text;

public class Node
{
    public static enum Color {
        WHITE, GRAY, BLACK // WHITE: unvisited, GRAY: visited, BLACK: finished
    };

    private String id; // id of the node
    private int distance; // distance of the node from the source
    private List<String> edges = new ArrayList<String>(); // list of edges
    private Color color = Color.WHITE;
    private String parent; // parent/predecessor of the node
    public Node() // initialize a empty node
    {
        distance = Integer.MAX_VALUE;
        color = Color.WHITE;
        parent = null;
    }
    /* the parameter nodeInfo is the line that is passed from the input, this nodeInfo is then split
    into key, value pair where the key is the node id and the value is the information associated
    with the node */
    public Node(String nodeInfo)
    {
        String[] inputLine = nodeInfo.split("\t");
        String key = "", value = ""; //initializing the strings 'key' and 'value'
        try {
            key = inputLine[0]; // node id
            value = inputLine[1]; // the list of adjacent nodes, distance, color, parent
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
        String[] tokens = value.split("\\|"); // split the value into tokens where
        this.id = key; // set the id of the node
        // setting the edges of the node
        for (String s : tokens[0].split(",")) {
            if (s.length() > 0) {
                edges.add(s);
            }
        }
        // setting the distance of the node
        if (tokens[1].equals("Integer.MAX_VALUE")) {
            this.distance = Integer.MAX_VALUE;
        } else {
            this.distance = Integer.parseInt(tokens[1]);
        }
        // setting the color of the node
        this.color = Color.valueOf(tokens[2]);
        // setting the parent of the node
        this.parent = tokens[3];
    }
    /* this method appends the list of adjacent nodes, distance, color, and parent and returns all
    these information as a single Text */
    public Text getNodeInfo()
    {
        StringBuffer s = new StringBuffer();
        try {
            for (String v : edges) {
                s.append(v).append(",");
            }
        } catch (NullPointerException e) {
            e.printStackTrace();
            System.exit(1);
        }
        s.append("|");
        if (this.distance < Integer.MAX_VALUE) {
            s.append(this.distance).append("|");
        } else {
            s.append("Integer.MAX_VALUE").append("|");
        }
        // append the color of the node
        s.append(color.toString()).append("|");
        // append the parent of the node
        s.append(getParent());
        return new Text(s.toString());
    }
}
// next page

```

```

/* Node.java */

// in public class Node:
// previous page

// getter and setter methods
public String getId() {
    return this.id;
}

public int getDistance() {
    return this.distance;
}

public void setId(String id) {
    this.id = id;
}

public void setDistance(int distance) {
    this.distance = distance;
}

public Color getColor() {
    return this.color;
}

public void setColor(Color color) {
    this.color = color;
}

public List<String> getEdges() {
    return this.edges;
}

public void setEdges(List<String> edges) {
    this.edges = edges;
}

public void setParent(String parent) {
    this.parent = parent;
}

public String getParent() {
    return parent;
}

} // end of class Node

```

```

/* MyMapper.java */

package YourPackageName;
import java.io.IOException;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Mapper.Context;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;

public class MyMapper extends Mapper<Object, Text, Text, Text>
{
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException
    {
        Node inNode = new Node(value.toString()); // input node

        if (inNode.getColor() == Node.Color.GRAY)
        {
            for (String neighbor : inNode.getEdges()) { // for all the adjacent nodes of gray node

                Node adjacentNode = new Node(); // create a new node

                adjacentNode.setId(neighbor); // set the id of the node
                adjacentNode.setDistance(inNode.getDistance() + 1); // unit edge weight
                adjacentNode.setColor(Node.Color.GRAY); // set the color of the node to be GRAY
                adjacentNode.setParent(inNode.getId()); // set the parent of the node,

                // for the nodes emitted here, the list of adjacent nodes will be empty
                context.write(new Text(adjacentNode.getId()), adjacentNode.getNodeInfo());
            }
            // this node is done, color it black
            inNode.setColor(Node.Color.BLACK);
        }

        // No matter what, emit the input node
        // If the node came into this method GRAY, it will be output as BLACK
        // otherwise, the node colored BLACK or WHITE is emitted
        context.write(new Text(inNode.getId()), inNode.getNodeInfo());
    } // end of map()
} // end of class MyMapper

```

```

/* MyReducer.java */

package YourPackageName;
import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MyReducer extends Reducer<Text, Text, Text, Text>
{
    /* counter to determine if more iterations are required to execute the map
    and reduce functions */
    public static enum Counter { // group name
        numberOfIterations // count name, initial value is 0
    }

    public void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException
    {
        Node outNode = new Node(); // create a new out node and set its values
        outNode.setId(key.toString()); // key is node ID

        // for all the values corresponding to a particular node id
        for (Text value : values)
        {
            Node inNode = new Node(key.toString() + "\t" + value.toString());

            // only one value contains neighbor list
            if (inNode.getEdges().size() > 0)
            {
                outNode.setEdges(inNode.getEdges());
            }

            // Update minimum distance:
            if (inNode.getDistance() < outNode.getDistance()) {
                outNode.setDistance(inNode.getDistance());
                outNode.setParent(inNode.getParent());
            }

            // Save the darkest color, which is at higher index:
            if (inNode.getColor().ordinal() > outNode.getColor().ordinal())
            {
                outNode.setColor(inNode.getColor());
            }

        } // end of for()

        // output: emit key: node id, value: node's information
        context.write(key, new Text(outNode.getNodeInfo()));

        // if the color is gray, the execution has to continue, incrementing the counter by 1
        if (outNode.getColor() == Node.Color.GRAY)
            context.getCounter(Counter.numberOfIterations).increment(1L);

    } // end of reduce()
} // end of class MyReducer

```



```

/* MyJob.java */

package YourPackageName;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Counters;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.ToolRunner;

public class MyJob extends Configured implements Tool
{
    public int run(String[] args) throws Exception {
        int iterationCount = 0;
        Job job;
        long greyCounter = 1;
        String jobName = "bfsjob";
        while (greyCounter > 0) // while there are more gray nodes to process
        {
            String input = "", output = "";
            job = new Job(new Configuration(), jobName);
            job.setMapperClass(MyMapper.class);
            job.setReducerClass(MyReducer.class);
            job.setNumReduceTasks(2); // # of reducers is 2 (have 2 outputs). Change it if want
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);
            job.setInputFormatClass(TextInputFormat.class); // default already
            job.setOutputFormatClass(TextOutputFormat.class);
            job.setJarByClass(MyJob.class);
            /* during 1st the user-specified file will be the input whereas for subsequent
               iterations the output of the previous iteration will be the input */
            if (iterationCount == 0) input = args[0];
            else input = args[1] + iterationCount; // "foo" + 1 + 1 equals "foo11"

            output = args[1] + (iterationCount + 1); // 9 + 9 + "foo" equals "18foo"

            FileInputFormat.setInputPaths(job, new Path(input));
            FileOutputFormat.setOutputPath(job, new Path(output));

            job.waitForCompletion(true); // wait for the job to complete
            Counters jobCnters = job.getCounters();
            /* if counter (initial=0 for each iteration) is incremented in reducer(s), then
               iteration has to be continued. */
            // Find counter, defined in class MyReducer, and then its current value:
            greyCounter = jobCnters.findCounter(MyReducer.Counter.numberofIterations).getValue();
            iterationCount++;
        } // end of while

        return 0;
    } // end of run()

    public static void main(String[] args) throws Exception
    {
        int res = ToolRunner.run(new Configuration(), new MyJob(), args);
        if(args.length != 2) {
            System.err.println("Usage: <in> <output name>");
            System.exit(res);
        }
    } // end of main()
} // end of class MyJob

```