PAGERANK ALGORITHM

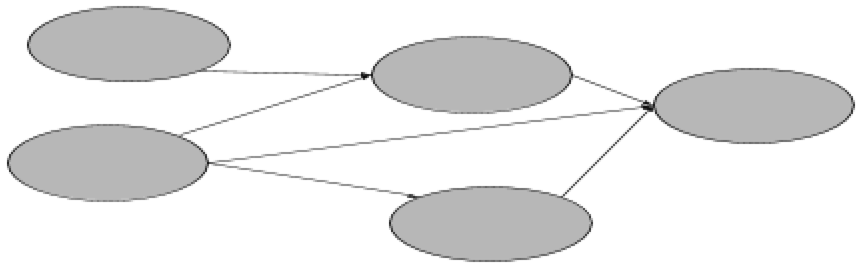
Big Data Analytics Project Report

Project Group:

Crystal Cuthinho	7053
Leon Dsouza	7058
Nevil Dsouza	7059
Melwyn Saldanha	7097

PROBLEM STATEMENT: PAGERANK ALGORITHM

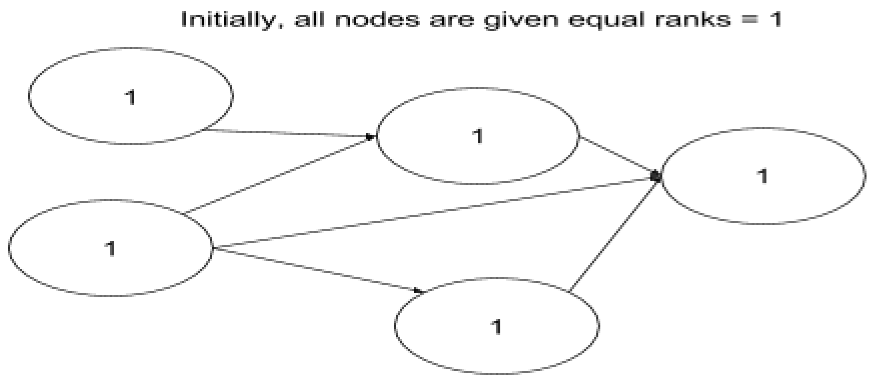
Almost everyone uses search engines like Google Search, duckduckgo, Microsoft Bing- just type in the key words, and the search engine will display the pages relevant for your search. But while displaying 100 results to the user, the search engine needs some heuristic to sort the result list, such that the top result must be most relevant to the user. The Internet is a huge distributed network of html documents linked to each other. Now these documents are represented by nodes in a graph data structure. The edges represent the hyperlink from one html document to another. This kind of data structure representing the Internet is called the Web Graph.



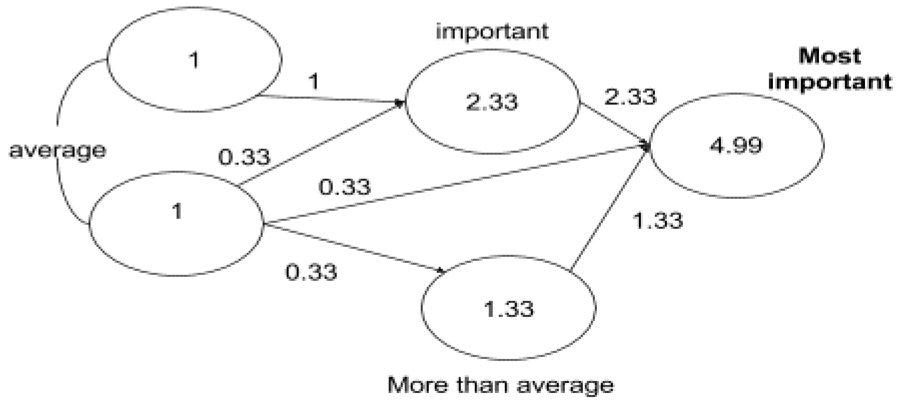
The traditional search engine algorithms used a simple keyword matching approach to list out the relevant pages. A common heuristic approach for search engines would be to sort the nodes in terms of connectivity within the Web graph & relevance. One of the most successful algorithms is Google's PageRank.

THEORY

The basic idea behind PageRank is sort the search result pages according to a Rank assigned to them by the algorithm. It has many factors to consider but for simplicity, we'll discuss an abstract idea. Initially, all nodes which are extracted from search query relevance are represented in a sub graph. Each node is assigned rank of unity.

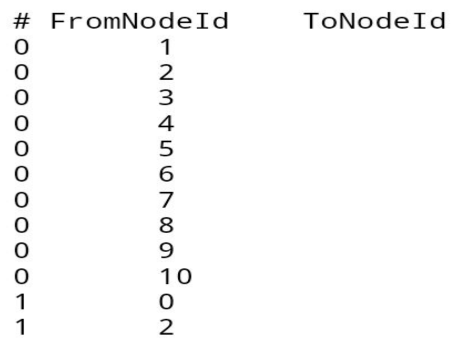


In the next step, the rank is allowed to flow within the sub graph. The rank flows from a node through its outgoing edges. The node contributes a specific value, which is assigned equally to all its outgoing edges within the sub graph. This value is called weight of the outgoing edge. The weight of each edge is equal to the the node's rank divided by the number of outgoing edges. Thus the node which has the most incoming relevant hyperlinks gets the highest rank.



DATA SET

PageRank algorithm typically requires a dataset which has a graph structure of connected Web pages on the Internet. Such Web graph data sets are made open source for analytics. We have selected a Web graph which represents e-commerce company Amazon’s product purchasing network on the Internet. Here is the snapshot of the data set:



```
# FromNodeId    ToNodeId
0       1
0       2
0       3
0       4
0       5
0       6
0       7
0       8
0       9
0       10
1       0
1       2
```

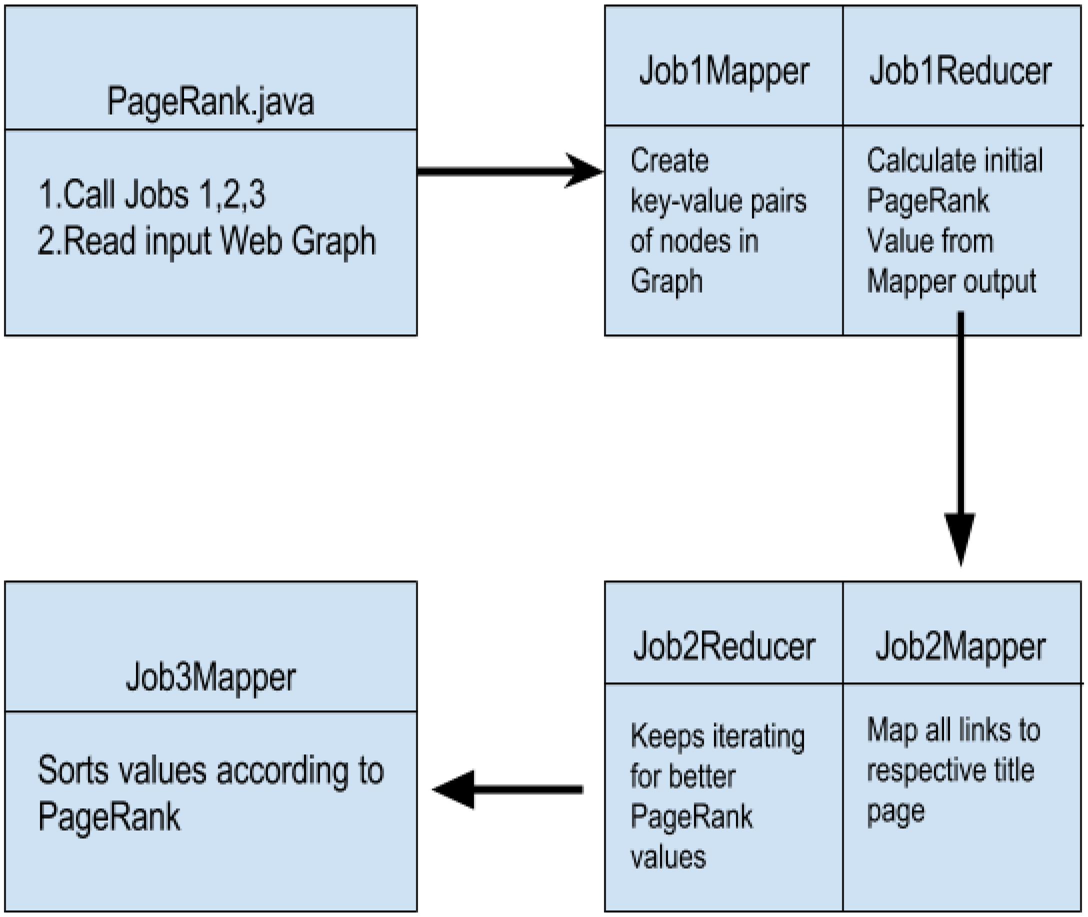
ALGORITHM

1. The Web Graph is represented by a directed graph G=(V,E) where V: number of web pages and E: the hyperlinks from and to web pages.
2. Initialize each node with pagerank score of 1 i.e. PG(i)=1 for all 0<=i<=n.
3. Calculate pagerank for each node using the formula:

$$PG(i) = \sum_{(j,i)\ \varepsilon\ E}^{n} PG(j) \div O(j) \text{ where } O(j) : \text{outlinks of } j$$

IMPLEMENTATION

The entire implementation of the algorithm is divided into 4 main parts :


PageRank.java
1.Call Jobs 1,2,3
2.Read input Web Graph
Job1Mapper
Create key-value pairs of nodes in Graph
Job1Reducer
Calculate initial PageRank Value from Mapper output
Job2Reducer
Keeps iterating for better PageRank values
Job2Mapper
Map all links to respective title page
Job3Mapper
Sorts values according to PageRank

TECHNOLOGIES USED :

1. Cloudera (Hadoop on Cent OS)
2. VMWare
3. Java
4. Eclipse

CODE :

PageRank.java

```
package job1;
/**
 * @author cloudera
 */
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.HashSet;
import java.util.Set;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.Job;

import job1.PageRankJob1Mapper;
import job1.PageRankJob1Reducer;
import job2.PageRankJob2Mapper;
import job2.PageRankJob2Reducer;
import job3.PageRankJob3Mapper;


public class PageRank {
  // parsing of file standard format requirements
  public static NumberFormat NF = new DecimalFormat("00");
  public static Set<String> NODES = new HashSet<String>();
  public static String LINKS_SEPARATOR = "|";

  // configuration values
  public static Double DAMPING = 0.85;
  public static int ITERATIONS = 2;
  //input file
  public static final String IN_PATH = "Amazon0601.txt";
  //output folder
```

```
    public static final String OUT_PATH = "1.txt";


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

        try {
        // delete output path if it exists already
        FileSystem fs = FileSystem.get(new Configuration());
        if (fs.exists(new Path(PageRank.OUT_PATH)))
            fs.delete(new Path(PageRank.OUT_PATH), true);

        // print current configuration in the console
        System.out.println("Damping factor: " + PageRank.DAMPING);
        System.out.println("Number of iterations: " + PageRank.ITERATIONS);
        System.out.println("Input directory: " + PageRank.IN_PATH);
        System.out.println("Output directory: " + PageRank.OUT_PATH);
        System.out.println("---------------------------");

        Thread.sleep(1000);

        String inPath = null;;
        String lastOutPath = null;
        PageRank pagerank = new PageRank();

        System.out.println("Running Job#1 (graph parsing) ...");
        //check if job is completed
        boolean isCompleted = pagerank.job1(IN_PATH, OUT_PATH + "/iter00");
        if (!isCompleted) {
            System.exit(1);
        }

        //run pagerank algorithm for so many times
        for (int runs = 0; runs < ITERATIONS; runs++) {
            inPath = OUT_PATH + "/iter" + NF.format(runs);
            lastOutPath = OUT_PATH + "/iter" + NF.format(runs + 1);
            System.out.println("Running Job#2 [" + (runs + 1) + "/" + PageRank.ITERATIONS + "] (PageRank calculation)
...");
            isCompleted = pagerank.job2(inPath, lastOutPath);
            if (!isCompleted) {
                System.exit(1);
            }
        }

        System.out.println("Running Job#3 (rank ordering) ...");
        isCompleted = pagerank.job3(lastOutPath, OUT_PATH + "/result");
    if (!isCompleted) {
        System.exit(1);
    }

    System.out.println("DONE!");
    System.exit(0);
}

//parses graph and initializes pagerank algorithm
public boolean job1(String in, String out) throws IOException,
                                                  ClassNotFoundException,
                                                  InterruptedException {

    Job job = Job.getInstance(new Configuration(), "Job #1");
    job.setJarByClass(PageRank.class);

    // input / mapper
    FileInputFormat.addInputPath(job, new Path(in));
    job.setInputFormatClass(TextInputFormat.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    job.setMapperClass(PageRankJob1Mapper.class);

    // output / reducer
    FileOutputFormat.setOutputPath(job, new Path(out));
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    job.setReducerClass(PageRankJob1Reducer.class);

    return job.waitForCompletion(true);

}

/**
 * Calculates new ranking of data in same format as input for next iteration
 * More iterations = More accuracy
 */
public boolean job2(String in, String out) throws IOException,
                                                  ClassNotFoundException,
                                                  InterruptedException {

    Job job = Job.getInstance(new Configuration(), "Job #2");
    job.setJarByClass(PageRank.class);
```

```
        // input / mapper
        FileInputFormat.setInputPaths(job, new Path(in));
        job.setInputFormatClass(TextInputFormat.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setMapperClass(PageRankJob2Mapper.class);

        // output / reducer
        FileOutputFormat.setOutputPath(job, new Path(out));
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setReducerClass(PageRankJob2Reducer.class);

        return job.waitForCompletion(true);

    }

    //sort as per ranking order
    public boolean job3(String in, String out) throws IOException,
                                                ClassNotFoundException,
                                                InterruptedException {

        Job job = Job.getInstance(new Configuration(), "Job #3");
        job.setJarByClass(PageRank.class);

        // input / mapper
        FileInputFormat.setInputPaths(job, new Path(in));
        job.setInputFormatClass(TextInputFormat.class);
        job.setMapOutputKeyClass(DoubleWritable.class);
        job.setMapOutputValueClass(Text.class);
        job.setMapperClass(PageRankJob3Mapper.class);

        // output
        FileOutputFormat.setOutputPath(job, new Path(out));
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setOutputKeyClass(DoubleWritable.class);
        job.setOutputValueClass(Text.class);

        return job.waitForCompletion(true);
    }
}
```

PageRankJob1Mapper.java

```
public class PageRankJob1Mapper extends Mapper<LongWritable, Text, Text, Text> {
  //creates key value pairs of nodes in graph
  @Override
  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    //skip lines with # in dataset
    if (value.charAt(0) != '#') {
      //tab is node separator
      int tabIndex = value.find("\t");
      //get node names
      String nodeA = Text.decode(value.getBytes(), 0, tabIndex);
      String nodeB = Text.decode(value.getBytes(), tabIndex + 1, value.getLength() - (tabIndex + 1));
      context.write(new Text(nodeA), new Text(nodeB));

      // add the current source node to the node list so we can
      // compute the total amount of nodes of our graph in Job#2
      PageRank.NODES.add(nodeA);
      // also add the target node to the same list: we may have a target node
      // with no outlinks (so it will never be parsed as source)
      PageRank.NODES.add(nodeB);
    }
  }
}
```

PageRankJob1Reducer.java

```
public class PageRankJob1Reducer extends Reducer<Text, Text, Text, Text> {

  @Override
  public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
    /* Give tab separated file with following format
     *    <title>    <page-rank>    <link1>,<link2>,<link3>,<link4>,...,<linkN>
     * initial value = Damping factor / no. of nodes
     */
    boolean first = true;
    String links = (PageRank.DAMPING / PageRank.NODES.size()) + "\t";
    //iterate over every node
    for (Text value : values) {
      if (!first)
        links += ",";
      links += value.toString();
      first = false;
    }
    context.write(key, new Text(links));
  }
}
```

PageRankJob2Mapper.java

```
public class PageRankJob2Mapper extends Mapper<LongWritable, Text, Text, Text> {
   @Override
   public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException

{
      /*
       * Input file from job1
       *    <title>    <page-rank>    <link1>,<link2>,<link3>,<link4>,... ,<linkN>
       * Output part 1
       *    <title>   |<link1>,<link2>,<link3>,<link4>, ... , <linkN>
       * Output part 2
       *    <link>    <page-rank>    <total-links>
       */
      //find separator between node names
      int tIdx1 = value.find("\t");
      int tIdx2 = value.find("\t", tIdx1 + 1);

      // extract tokens from the current line
      String page = Text.decode(value.getBytes(), 0, tIdx1);
      String pageRank = Text.decode(value.getBytes(), tIdx1 + 1, tIdx2 - (tIdx1 + 1));
      String links = Text.decode(value.getBytes(), tIdx2 + 1, value.getLength() - (tIdx2 + 1));

      //link pages are separated by commas
      String[] allOtherPages = links.split(",");
      for (String otherPage : allOtherPages) {
         Text pageRankWithTotalLinks = new Text(pageRank + "\t" + allOtherPages.length);
         context.write(new Text(otherPage), pageRankWithTotalLinks);
      }

      // put the original links so the reducer is able to produce the correct output
      context.write(new Text(page), new Text(PageRank.LINKS_SEPARATOR + links));

   }

}
```

PageRankJob2Reducer.java

```
public class PageRankJob2Reducer extends Reducer<Text, Text, Text, Text> {
  @Override
  public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
                                                              InterruptedException {
    /* PageRank calculation algorithm (reducer)
     * Input comes from 2 files of Job2 mapper
     */

    String links = "";
    double sumShareOtherPageRanks = 0.0;

    for (Text value : values) {

      String content = value.toString();

      if (content.startsWith(PageRank.LINKS_SEPARATOR)) {
        // if this value contains node links append them to the 'links' string
        // for future use: this is needed to reconstruct the input for Job#2 mapper
        // in case of multiple iterations of it.
        links += content.substring(PageRank.LINKS_SEPARATOR.length());
      } else {

        String[] split = content.split("\\t");

        // extract tokens i.e. respective distances
        double pageRank = Double.parseDouble(split[0]);
        int totalLinks = Integer.parseInt(split[1]);

        // add the contribution of all the pages having an outlink pointing
        // to the current node: we will add the DAMPING factor later when recomputing
        // the final pagerank value before submitting the result to the next job.
        sumShareOtherPageRanks += (pageRank / totalLinks);
      }
    }

    double newRank = PageRank.DAMPING * sumShareOtherPageRanks + (1 - PageRank.DAMPING);
    context.write(key, new Text(newRank + "\t" + links));
   }
}
```

PageRankJob3Mapper.java

```
public class PageRankJob3Mapper extends Mapper<LongWritable, Text, DoubleWritable, Text> {
  @Override
  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {



    /* Maps pages according to rank and puts in an output file
     */

    int tIdx1 = value.find("\t");
    int tIdx2 = value.find("\t", tIdx1 + 1);

    // extract tokens from the current line
    String page = Text.decode(value.getBytes(), 0, tIdx1);
    float pageRank = Float.parseFloat(Text.decode(value.getBytes(), tIdx1 + 1, tIdx2 - (tIdx1 + 1)));

    context.write(new DoubleWritable(pageRank), new Text(page));

  }

}
```

RESULTS

Rank	PAGERANK SCORE	NODE NUMBERS
1	58.115928649902344	1041
2	50.678245544433594	45
3	45.4197883605957	50

CONCLUSION

The code is able to process any Web graph and assigns page ranks to each of the pages in the dataset. Thus, PageRank algorithm is applied in order to find relevance among different pages and because page rank values tend to be high, the program tends to normalize them to a certain extent in order to make it easier to read and provides a sorted output using the page ranks. The PageRank algorithm is used to rank pages along with a certain other parameters of ranking pages for different users.