

The Web Laboratory  
Web Graph Cleaning using Map Reduce  
Fall 2008 Report

Zhiyu Zhang (zz98@cornell.edu)

Computer Science  
Cornell University  
Advisor: Professor William Arms

## Table of Contents

Abstract.....	3
Introduction.....	4
Problem Statement.....	5
Acknowledgements.....	13
References.....	14
Appendix.....	15

## Abstract

The Web Graph Cleaning project this semester is based on the work done in Spring 2008 by Manu Jain. The first goal of this semester is to improve the scalability of previous work, so that it can take input of larger dataset like the amazonLinks. Efforts were made to tune hadoop cluster using different settings. However, this does not solve the out of memory problem exposed last semester. Also, while we were trying to adapt the previous program to larger dataset, we ran into problem that crashed part of hadoop cluster. This issue is bypassed by simply changing the Map-Reduce logic and rewriting all code. Now the new program works fine with current amazonLinks dataset.

Clean graph generated by this program will be fed to programs next step, so the second goal this semester is to make sure the output of Web Graph Cleaning meets the needs of input format for next step. Currently, we have page rank program which is updated this semester, and hub-and-authority program which is recently written. Both of them require different input format. Our graph cleaning program generates different output according to the parameter passed in. It can also be easily changed to meet demands that might appear in the future.

## Introduction

The Web Laboratory project is a joint project of Cornell University and the Internet Archive. The main purpose of this project is to provide data and computing tools for research about the Web and the information in the Web. Several teams are working on specific tasks such as Web Graph Generation, Page Rank Calculation, Data Movement and Tracking, etc.

The Web Graph Cleaning project in this semester is based on the work done in Spring 2008 by Manu Jain [2]. But early effort dates back to the work by Murarka [3], in which the problem of web graph generation is identified, and the first version of URL canonicalizer is introduced. The Web Graph project leaped forward in the work by Kim, Rajan and Seguin when Map-Reduce and Hadoop are introduced as tools for problem solving. The project evolved into the outlook nowadays after their work. In the fall semester of 2007, Jawad and Teng's work further improved the correctness and scalability of previous work. They also formalized the problem and conducted a simple scalability analysis using experimental results. Jain, Kaul and Lyall's work [6] [2] further optimized previous program and migrated the code to Hadoop 0.17.1. And this is the jump-start point of our work this semester.

Background knowledge worth mentioning here is our computation platform. MapReduce[7] is a programming model and an associated implementation for processing and generating large data sets. It treats input and output as key/value pairs, and divides each task as a serial of map functions and reduce functions. Hadoop is an open source implementation of MapReduce programming paradigm. It is a framework for running applications on large clusters built of commodity hardware. The Hadoop framework transparently provides applications both reliability and data motion. It also provides a distributed file system (HDFS) that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. Both Map/Reduce and the distributed file system are designed so that node failures are automatically handled by the framework. This report is updated to Hadoop version 0.18.2.

# Problem Statement

## 1. Web Graph Input

### **Node:**

Node in Web Graph is identified by an Uniform Resource Locator(URL).

### **Edge:**

Edge in Web Graph is directed, and it is a pair of URLs <fromURL, toURL>. The first element identifies the page containing the link, and the second element is the page where the link points to.

### **Tab-Separated-Value(TSV) file**

Physical input file of the Web Graph. Each line records a link in the following form: 'fromURL<TAB>toURL'.

### **Data Set:**

Data Sets are web claws generated in previous work. Each data set generally is stored as a number of TSV files. They are the actual input of our Web Graph Cleaning program. Since we are using Hadoop as our computation platform, data sets would be manually moved to HDFS before calculation. By the implementation of HDFS, data sets are read-only, and all TSV files in one data set can be logically treated as a single file.

## 2. More about Nodes

Each node in our Web Graph is identified by its URL. However, URL in data sets is uncanonicalized. That is to say each URL might contain both upper and lower case letters while some part of real-world URLs is not case sensitive (for example "http" and "www"). Some URLs may contain unnecessary slash at the end. We need a routine to canonicalize the URLs so we have to guarantee that each node has only one form of URL. This is why we should introduce a canonicalizer. In our project, we continue to use the canonicalizer from previous work.

URL is enough to identify each node, but it is not an effective way. Real-world URL is a fairly long string, and different URLs always contain same string patterns like "http", "edu" or "www.amazon.com". These common patterns indicate that the underlying useful information density in URL is pretty low. A better approach would be to assign an exclusive number to each URL, and hash is a natural way to achieve this. An algorithm called FNV (Fowler/Noll/Vo) hash is introduced here. FNV hash is a non-cryptographic hash function created by Glenn Fowler, Landon Curt Noll, and Phong Vo. FNV hash allows one to quickly hash lots of data while maintaining a reasonable collision rate. The high dispersion of the FNV hashes makes them well suited for hashing nearly identical strings such as URLs, hostnames, filenames, text, IP addresses, etc. More information about FNV could be founded in [8]. In our program, we are going to use 64-bit offset. Here is a code snippet:

```

// Algorithm for FNV Hash Code

public static final long FNV1_64_INIT = 0xcbf29ce484222325L;
private static final long FNV_64_PRIME = 0x100000001b3L;
public long hashCode64(String str)
{
    long hval = FNV1_64_INIT;
    int len = str.length();
    for (int i=0; i<len; i++)
    {
        hval *= FNV_64_PRIME;
        hval ^= (long)str.charAt(i);
    }
    return hval;
}
// End of FNV Hash Code

```

Suppose we have an edge  $E: \langle A, B \rangle$ . As we have mentioned above, A and B are two nodes (URLs) in our Web Graph. With this example, let's define degree.

**In Degree:**

Each edge pointing to a certain node is an In Degree for this node. In our example, E is an In Degree for node B.

**Out Degree:**

Each edge starts from a certain node is an Out Degree for this node. In our example, E is an Out Degree for node A.

One thing worth noticing here is that one edge (say  $\langle A, B \rangle$ ) might appear more than once in a data set. And the case is fairly common in large data sets. In our definition, repeated edges are counted only once. Next, we are going to define three types of node in our Web Graph.

**Core Node:**

Core nodes are those nodes with (In Degree  $> 0$ ) and (Out Degree  $> 0$ ).

**Source Node:**

Source node has (In Degree = 0) and (Out Degree  $> 0$ ).

**Dead-end Node:**

Dead-end node has (In Degree  $> 0$ ) and (Out Degree = 0).

Since every node in our Web Graph participates in at least one edge link, it must fall into one and exactly one of three types above mentioned.

### 3. Process and output of previous solution

In Manu Jain's report [2], three steps containing four map-reduces are used to eliminate unwanted nodes and edges. The following graph from [2] is very helpful for understanding his algorithms.

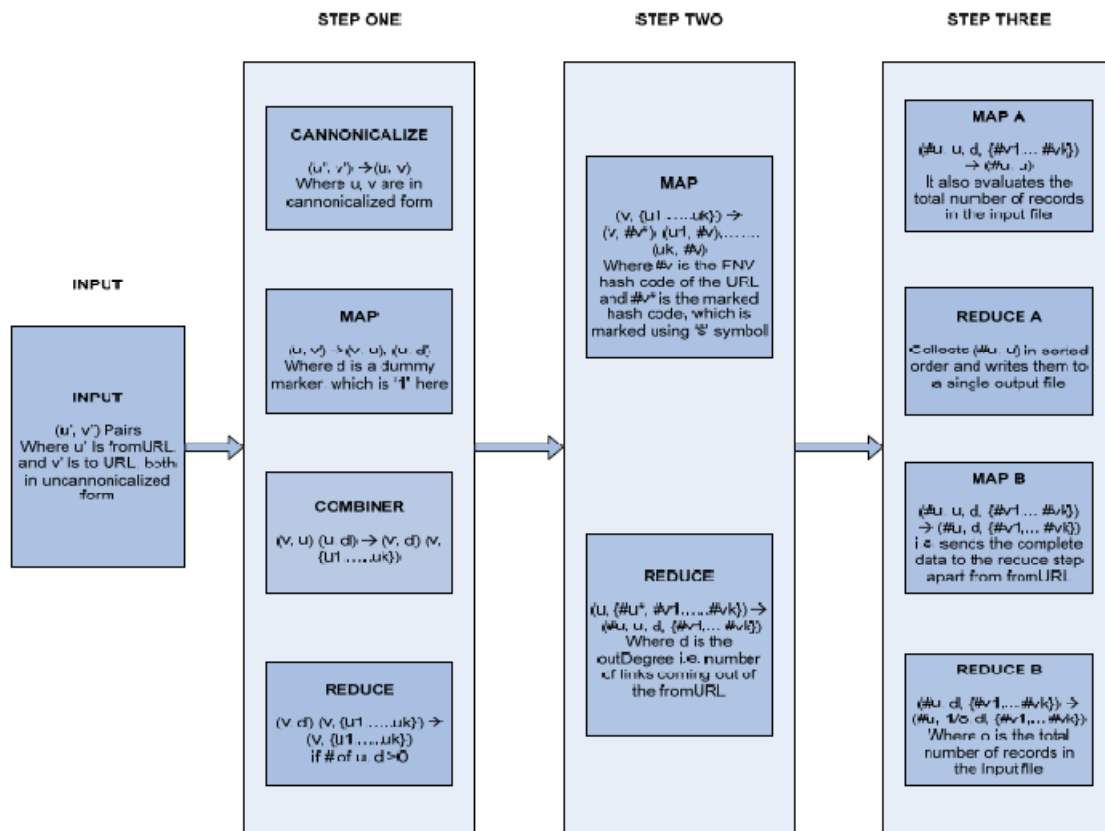


Figure: Process Flow for the Web Graph Generation Algorithm

Here is some information that generalizes the process, which should be enough to understand the usage of each step.

Input:  $(u, v)$   
 uncanonicalized URL pair which represents a link

After Step One:  $(v, \{u1, u2, \dots, uk\})$   
 Canonicalize each URL, reverse the form of all links, remove those links with  $v(\text{toURL})$  never appears as  $u(\text{fromURL})$ .

After Step Two:  $(\#u, u, \text{Out Degree}, \{\#v1, \#v2, \dots, \#vk\})$   
 Reverse all links again, remove duplicate links and those links with  $u(\text{fromURL})$  never appears as  $v(\text{toURL})$ , toURL converted to Hash code.

After Step Three:  $(\#u, 1/o, \text{Out Degree}, \{\#v1, \#v2, \dots, \#vk\})$   
 Count number of nodes, and calculate  $1/o$ .

So far, we have understood the process of previous solution. As we can see, after using the first and second map-reduce, only core nodes (defined above) are left in the final output.

## 4. Problems and new requirements

In 5.1 of Manu Jain's report [2], the problem of "Lack of Memory Space in JVM" is mentioned, and a solution is proposed. However, when we started our project and tried to run previous program with data set amazonLinks, this problem showed again. We tried to reserve all available memory space in Hadoop configuration file, but the running of data set amazonLinks still caused "out of memory space in JVM".

Next, we tried to improve previous code, and see how much we can do. Because out of memory is a problem we are currently facing, we tried to offload the heap space by partly writing in-memory content into HDFS. However, this did not solve our problem because we were stuck in a bug that would crash part of Hadoop nodes. Although this bug was later bypassed and did not get well investigation, we still want to record this problem here because it might appear again in the future. The phenomenon we observed is that some map task took unreasonable length of time and did not report their status to Hadoop master. Master times out after a certain amount of time and assign the map task to some other node. But when timeout happens several times with the same worker node, Hadoop master will then claim that the worker node is in failure and should be kicked out of cluster. More and more timeout and task fail will finally cause the failure of current map-reduce job. But during the process, the cluster experiences super heavy load and fails to respond to user command or connection. After a long time when the cluster finally recovered, several nodes are not in the cluster anymore. We observed this problem in Hadoop 0.17.1, but did not find other similar problem report from Hadoop community. We are not sure whether it would appear again in future version.

New requirements in the project include the following:

- a. Page rank program is being updated, and it requires a new input format with core node, source node and dead-end node.
- b. Hubs-and-authorities algorithm is introduced and requires a different input format.
- c. Output a general graph cleaning result, and let user choose what types nodes should be retained in the result. Four options should be supported: only core nodes, core nodes and source nodes, core nodes and dead-end nodes, all nodes.

## 5. New Program

### 5.1 User interface:

The new Graph Cleaning program has only one executable java class, and has only one user interface. No shell script is now being used. It generates different output according to different parameter passed in. Here is the help information which would be showed in OS shell, and it generalized the usage.

Usage: `hadoop jar <package name> GraphClean <-COMMAND> <src>`  
where COMMAND is one of:

HA	generate Hub and Authority input
Pagerank	generate pagerank input
c	output core node
cs	output core node and source node
cd	output core node and destination node
csd	output core node, source node and destination node

Please make sure to use the right parameters, or you will see this message.

## 5.2 Output Format

Each COMMAND will produce different output:

### (1) HA

Format: `hash(nodeURL), nodeURL, $1, #1, <List of &Hash(toURL)>, <List of %Hash(fromURL)>`

Hub-and-Authority algorithm requires all nodes from original graph to remain in graph after cleanup. So our Graph Cleaning program does not eliminate any node from the original graph. It only changes the representation of the original graph, and formalizes it for the use of Hub-and-Authority. Four special symbols are used: \$ means the following number is hub value, # means the following number is authority value, & means the following number is the hash value of a toURL, % means the following number is the hash value of a fromURL. Usage of these symbols is agreed with Hub-and-Authority program.

### (2) Pagerank

Format: `hash(nodeURL), nodeURL, outDegree, <List of Hash(toURL)>`

The new page rank program also requires all nodes from original graph. So again we are not eliminating any nodes. The format of output is almost the same with output of step two in Manu's solution. The only difference is that outDegree could be 0 so that list of Hash(toURL) is empty. Reason for this quality is because we are now leaving every node in the result output, so that dead-end nodes will not be eliminated, and they have 0 outDegree.

### (3) c

Format: same as (2)

Here "c" means core node. This option will cause Graph Cleaning program to output only core nodes

### (4) cs

Format: same as (2)

"cs" stands for core node and source node.

(5) cd

Format: same as (2)

“cd” stands for core node and dead-end node.

(6) csd

Format: same as (2)

This option prints all core nodes, source nodes and dead-end nodes.

### 5.3 Code Organization

As other Hadoop program, our program is organized as a main function, several Mappers and Reducers. Logic in main function is quite clear. It only check the input parameter, and run the designated map-reduce task. If main function finds that user input does not meet a certain requirement, it will print out an instruction and tell user the correct usage.

Several Mappers and Reducers are being used, and it is easy to notice that their code are highly repeated. It is written in this style because it is possible that we want to extract one or two mapper/reducer from the code for future use, and we'd better write the mappers/reducers in such a style that it is self-explanatory and does not rely on other functions. Next we will conclude for each option, which mappers/reducers are used in sequence.

(1) HA

MapForHA, ReduceForHA

(2) Pagerank

MapForPageRank, ReduceForPageRank

(3) c

Map1ForGeneral, Reduce1ForGeneralc, Map2ForGeneral, Reduce2ForGeneral,  
Map3ForGeneral, Reduce3ForGeneral

(4) cs

Map1ForGeneral, Reduce1ForGeneralcs, Map2ForGeneral, Reduce2ForGeneral,  
Map3ForGeneral, Reduce3ForGeneral

(5) cd

Map1ForGeneral, Reduce1ForGeneralcd, Map2ForGeneral, Reduce2ForGeneral,  
Map3ForGeneral, Reduce3ForGeneral

(6) csd

Map1ForGeneral, Reduce1ForGeneralcsd, Map2ForGeneral, Reduce2ForGeneral, Map3ForGeneral, Reduce3ForGeneral

We have realized that in option cd and csd, only the first mapper and first reducer is required. In another word, after the first map-reduce, the output is already what we want as the final output. However, we still keep the next two mapper/reducers here, because whether they are required depends on output format.

## 5.4 Example

Suppose we have Graph Cleaning program uploaded into the Hadoop cluster, and the compiled jar file is named "Weblab.jar". Also, we have a data set named "cornell" in HDFS. Next we will show the result of different command:

(1) Hadoop jar Weblab.jar GraphClean -HA cornell

Output result is in HDFS "cornellForHA", and it can be fed directly to the Hub-and-Authority calculation.

(2) Hadoop jar Weblab.jar GraphClean -pagerank cornell

Output in HDFS "cornellForPagerank" can be fed directly to the latest pagerank program.

(3) Hadoop jar Weblab.jar GraphClean -c cornell

Output is in HDFS "cornellGeneralc". "cornellGeneralctemp" and "cornellGeneralctemp2" are intermediate files which is useless and should be manually deleted.

The number of core nodes:  $c = 351,093$ , which is same as the number in page 20 of [2].

(4) Hadoop jar Weblab.jar GraphClean -cs cornell

Output is in HDFS "cornellGeneralcs". "cornellGeneralcstemp" and "cornellGeneralcstemp2" are intermediate files which is useless and should be manually deleted.

Number of core nodes and source nodes:  $cs = 422,717$

(5) Hadoop jar Weblab.jar GraphClean -cd cornell

Output is in HDFS "cornellGeneralcd". "cornellGeneralcdtemp" and "cornellGeneralcdtemp2" are intermediate files which is useless and should be manually deleted.

Number of core nodes and dead-end nodes:  $cd = 554,798$

(6) Hadoop jar Weblab.jar GraphClean -csd cornell

Output is in HDFS "cornellGeneralcsd". "cornellGeneralcsdtemp" and "cornellGeneralcsdtemp2" are intermediate files which is useless and should be manually deleted.

Number of all three types of node:  $csd = 626,422$

(\*) Finally, we see the following number, which to some extent justifies the result we get:

$csd - cd = cs - c = 71,624$ , and this is the number of source node.

$csd - cs = cd - c = 203,705$ , and this is the number of dead-end node.

## 5.5 Performance and scalability

Current program has no problem dealing with the amazonLinks data set, which is a major goal of Graph Cleaning in this semester. Next we apply the following command to different data sets and see how much it takes. This computation has 3 required Map-Reduce, and relatively takes a long time. Computation platform we use is the large Hadoop cluster in CAC, and all 48 nodes are online. The cluster is running Hadoop version 0.18.2.

Command: `Hadoop jar Weblab.jar GraphClean -cs <src>`

Data Set	Size(GB)	Time(seconds)	Number of Nodes
cornell	0.5	$30+20+22 = 72$	422,717
edu	4.9	$62+31+27 = 120$	2,811,457
amazonLinks	433	$3460+720+431 = 4611$	16,688,015

Considering the overhead for running small size Data Set, the time consumed is linearly increasing to the size of Data Set, which is consistent to the algorithm we are using in code.

## Acknowledgements

We would like to thank Professor William Arms for his guidance throughout the project. Also, we would like to thank Lucia Walle for helping us make use of the hadoop cluster in advanced computing center and technical support to keep our program running.

This work is funded in part by National Science Foundation grants CNS-0403340, DUE-0127308, SES-0537606, IIS-0634677, and IIS-0705774. The Web Lab is an NSF Next Generation Cyberinfrastructure project.

## References

- [1] Shantanu Shah, Generating a Web Graph, May 2005,  
<http://weblab.infosci.cornell.edu/public/papers/Shah2005a.doc>
- [2] Manu Jain, Web Graph Generation. August 2008,  
<http://www.infosci.cornell.edu/SIN/WebLab/papers/Jain2008b.pdf>
- [3] Murarka, S., Web Graph Project. May 2006,  
<http://www.infosci.cornell.edu/SIN/WebLab/papers/Murarka2006.pdf>
- [4] Sangwoo Kim, Sanjay Rajan, and Sean Seguin, Web Graph Generation. May 2007,  
<http://www.infosci.cornell.edu/SIN/WebLab/papers/Kim2007b.pdf>
- [5] Anthony Jawad and Jie Teng, Web Graph Generation: Fall 2007 Report. December 2007,  
<http://www.infosci.cornell.edu/SIN/WebLab/papers/Jawad2007.pdf>
- [6] Manu Jain, Gayatri Kaul and Aditi Lyall, Web Graph Generation. May 2008,  
<http://www.infosci.cornell.edu/SIN/WebLab/papers/Jain2008a.pdf>
- [7] J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Usenix SDI, 2004.
- [8] FNV hash in Landon Curt Noll's webpage: <http://www.isthe.com/chongo/tech/comp/fnv/>

## Appendix

GraphClean.java

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

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class GraphClean {

    public static class MapForHA extends MapReduceBase implements
        apper<WritableComparable, Writable, LongWritable, Text> {
        private Text fromURL = new Text();
        private Text toURL = new Text();
        private long hashFromURL;
        private long hashToURL;

        // Algorithm for FNV Hash Code
        public static final long FNV1_64_INIT = 0xcbf29ce484222325L;
        private static final long FNV_64_PRIME = 0x100000001b3L;

        public long hashCode64(String str)
        {
            long hval = FNV1_64_INIT;
            int len = str.length();
            for (int i=0; i<len; i++)
            {
                hval *= FNV_64_PRIME;
                hval ^= (long)str.charAt(i);
            }

            return hval;
        }
        // End of FNV Hash Code

        public void map(WritableComparable key, Writable value,
            outputCollector<LongWritable, Text> output, Reporter reporter) throws IOException {
            String splitString[] = ((Text)value).toString().split("\\t");

            Canonicalizer can = new Canonicalizer();
            String canFromURL = can.canonicalize(splitString[0].trim());
            String canToURL = can.canonicalize(splitString[1].trim());

            if (canFromURL.length()>0 && canToURL.length()>0)
            {
```

```

        fromURL.set(canFromURL);
        toURL.set(canToURL);
        hashFromURL = hashCode64(canFromURL);
        hashToURL = hashCode64(canToURL);

        output.collect(new LongWritable(hashToURL), toURL);
        output.collect(new LongWritable(hashToURL), new
            Text("%"+hashFromURL));
        output.collect(new LongWritable(hashFromURL), fromURL);
        output.collect(new LongWritable(hashFromURL), new
            Text("&"+hashToURL));
    }
}

public static class ReduceForHA extends MapReduceBase implements
    Reducer<WritableComparable, Text, WritableComparable, Text>
{
    public void reduce(WritableComparable key, Iterator<Text> values,
        OutputCollector<WritableComparable, Text> output,
        Reporter reporter) throws IOException
    {
        TreeSet<Long> toList = new TreeSet<Long>();
        TreeSet<Long> fromList = new TreeSet<Long>();
        String ownURL = new String();
        boolean gotURL = false;

        while(values.hasNext())
        {
            String s = ((Text)values.next()).toString().trim();
            if(s.startsWith("%"))
            {
                fromList.add(Long.parseLong(s.substring(1)));
            }
            else if(s.startsWith("&"))
            {
                toList.add(Long.parseLong(s.substring(1)));
            }
            else if(!gotURL)
            {
                gotURL = true;
                ownURL = s;
            }
        }

        if(gotURL)
        {
            StringBuffer resultValue = new StringBuffer();

```

```

        resultValue.append(ownURL);
        resultValue.append("\t");
        resultValue.append("$1");
        resultValue.append("\t");
        resultValue.append("#1");
        resultValue.append("\t");

        Iterator<Long> iterator = toList.iterator();
        while(iterator.hasNext())
        {
            resultValue.append("&"+iterator.next().toString());
            resultValue.append("\t");
        }

        iterator = fromList.iterator();
        while(iterator.hasNext())
        {
            resultValue.append("%"+iterator.next().toString());
            resultValue.append("\t");
        }

        output.collect(key, new Text(resultValue.toString()));
//format: hash(nodeURL), nodeURL, $1, #1, <List of &Hash(toURL)>, <List of %Hash(fromURL)>
    }
}
}

```

```

public static class MapForPageRank extends MapReduceBase implements
    Mapper<WritableComparable, Writable, LongWritable, Text> {
    private Text fromURL = new Text();
    private Text toURL = new Text();
    private long hashFromURL;
    private long hashToURL;

    // Algorithm for FNV Hash Code
    public static final long FNV1_64_INIT = 0xcbf29ce484222325L;
    private static final long FNV_64_PRIME = 0x100000001b3L;

    public long hashCode64(String str)
    {
        long hval = FNV1_64_INIT;
        int len = str.length();
        for (int i=0; i<len; i++)
        {
            hval *= FNV_64_PRIME;
            hval ^= (long)str.charAt(i);
        }
    }
}

```

```

        return hval;
    }
    // End of FNV Hash Code

    public void map(WritableComparable key, Writable value,
    OutputCollector<LongWritable, Text> output, Reporter reporter) throws IOException {
        String splitString[] = ((Text)value).toString().split("\t");

        Canonicalizer can = new Canonicalizer();
        String canFromURL = can.canonicalize(splitString[0].trim());
        String canToURL = can.canonicalize(splitString[1].trim());

        if (canFromURL.length()>0 && canToURL.length()>0)
        {
            fromURL.set(canFromURL);
            toURL.set(canToURL);
            hashFromURL = hashCode64(canFromURL);
            hashToURL = hashCode64(canToURL);

            output.collect(new LongWritable(hashToURL), toURL);
            output.collect(new LongWritable(hashToURL), new
                Text("%"+hashFromURL));
            output.collect(new LongWritable(hashFromURL), fromURL);
            output.collect(new LongWritable(hashFromURL), new
                Text("&"+hashToURL));
        }
    }
}

public static class ReduceForPageRank extends MapReduceBase implements
    Reducer<WritableComparable, Text, WritableComparable, Text>
{
    public void reduce(WritableComparable key, Iterator<Text> values,
        OutputCollector<WritableComparable, Text> output,
        Reporter reporter) throws IOException
    {
        TreeSet<Long> toList = new TreeSet<Long>();
        TreeSet<Long> fromList = new TreeSet<Long>();
        String ownURL = new String();
        boolean gotURL = false;

        while(values.hasNext())
        {
            String s = ((Text)values.next()).toString().trim();
            if(s.startsWith("%"))
            {
                fromList.add(Long.parseLong(s.substring(1)));
            }
        }
    }
}

```

```

    }
    else if(s.startsWith("&"))
    {
        toList.add(Long.parseLong(s.substring(1)));
    }
    else if(!gotURL)
    {
        gotURL = true;
        ownURL = s;
    }
}

if(gotURL)
{
    StringBuffer resultValue = new StringBuffer();
    resultValue.append(ownURL);
    resultValue.append("\t");
    resultValue.append(toList.size());
    resultValue.append("\t");

    Iterator<Long> iterator = toList.iterator();
    while(iterator.hasNext())
    {
        resultValue.append(iterator.next().toString());
        resultValue.append("\t");
    }

    output.collect(key, new Text(resultValue.toString()));
    //format: hash(nodeURL), nodeURL, outDegree, <List of Hash(toURL)>
}
}
}

```

```

public static class Map1ForGeneral extends MapReduceBase implements
    Mapper<WritableComparable, Writable, LongWritable, Text> {
    private Text fromURL = new Text();
    private Text toURL = new Text();
    private long hashFromURL;
    private long hashToURL;

    // Algorithm for FNV Hash Code
    public static final long FNV1_64_INIT = 0xcbf29ce484222325L;
    private static final long FNV_64_PRIME = 0x100000001b3L;

    public long hashCode64(String str)
    {
        long hval = FNV1_64_INIT;
        int len = str.length();
    }
}

```

```

        for (int i=0; i<len; i++)
        {
            hval *= FNV_64_PRIME;
            hval ^= (long)str.charAt(i);
        }

        return hval;
    }
    // End of FNV Hash Code

    public void map(WritableComparable key, Writable value,
        OutputCollector<LongWritable, Text> output,
        Reporter reporter) throws IOException {
        String splitString[] = ((Text)value).toString().split("\t");

        Canonicalizer can = new Canonicalizer();
        String canFromURL = can.canonicalize(splitString[0].trim());
        String canToURL = can.canonicalize(splitString[1].trim());

        if (canFromURL.length()>0 && canToURL.length()>0)
        {
            fromURL.set(canFromURL);
            toURL.set(canToURL);
            hashFromURL = hashCode64(canFromURL);
            hashToURL = hashCode64(canToURL);

            output.collect(new LongWritable(hashToURL), toURL);
            output.collect(new LongWritable(hashToURL), new
                Text("%"+hashFromURL));           //% indicates fromURL
            output.collect(new LongWritable(hashFromURL), fromURL);
            output.collect(new LongWritable(hashFromURL), new
                Text("&"+hashToURL));           //& indicates toURL
        }
    }
}

public static class Reduce1ForGeneralc extends MapReduceBase implements
    Reducer<WritableComparable, Text, WritableComparable, Text>
{
    public void reduce(WritableComparable key, Iterator<Text> values,
        OutputCollector<WritableComparable, Text> output,
        Reporter reporter) throws IOException
    {
        TreeSet<Long> toList = new TreeSet<Long>();
        TreeSet<Long> fromList = new TreeSet<Long>();
        String ownURL = new String();
        boolean gotURL = false;
    }
}

```

```

while(values.hasNext())
{
    String s = ((Text)values.next()).toString().trim();
    if(s.startsWith("%"))
    {
        fromList.add(Long.parseLong(s.substring(1)));
    }
    else if(s.startsWith("&"))
    {
        toList.add(Long.parseLong(s.substring(1)));
    }
    else if(!gotURL)
    {
        gotURL = true;
        ownURL = s;
    }
}

if(gotURL && !toList.isEmpty() && !fromList.isEmpty())
{
    StringBuffer resultValue = new StringBuffer();
    resultValue.append(ownURL);
    resultValue.append("\t");
    resultValue.append(toList.size());
    resultValue.append("\t");

    Iterator<Long> iterator = toList.iterator();
    while(iterator.hasNext())
    {
        resultValue.append(iterator.next().toString());
        resultValue.append("\t");
    }
    output.collect(key, new Text(resultValue.toString()));
}
}

public static class Reduce1ForGeneralcs extends MapReduceBase implements
    Reducer<WritableComparable, Text, WritableComparable, Text>
{
    public void reduce(WritableComparable key, Iterator<Text> values,
        OutputCollector<WritableComparable, Text> output,
        Reporter reporter) throws IOException
    {
        TreeSet<Long> toList = new TreeSet<Long>();
        TreeSet<Long> fromList = new TreeSet<Long>();
        String ownURL = new String();
        boolean gotURL = false;

```

```

while(values.hasNext())
{
    String s = ((Text)values.next()).toString().trim();
    if(s.startsWith("%"))
    {
        fromList.add(Long.parseLong(s.substring(1)));
    }
    else if(s.startsWith("&"))
    {
        toList.add(Long.parseLong(s.substring(1)));
    }
    else if(!gotURL)
    {
        gotURL = true;
        ownURL = s;
    }
}

if(gotURL && !toList.isEmpty()) //already get URL and has tolist
{
    StringBuffer resultValue = new StringBuffer();
    resultValue.append(ownURL);
    resultValue.append("\t");
    resultValue.append(toList.size());
    resultValue.append("\t");

    Iterator<Long> iterator = toList.iterator();
    while(iterator.hasNext())
    {
        resultValue.append(iterator.next().toString());
        resultValue.append("\t");
    }
    output.collect(key, new Text(resultValue.toString()));
}
}

public static class Reduce1ForGeneralcd extends MapReduceBase implements
    Reducer<WritableComparable, Text, WritableComparable, Text>
{
    public void reduce(WritableComparable key, Iterator<Text> values,
        OutputCollector<WritableComparable, Text> output,
        Reporter reporter) throws IOException
    {
        TreeSet<Long> toList = new TreeSet<Long>();
        TreeSet<Long> fromList = new TreeSet<Long>();
        String ownURL = new String();

```

```

boolean gotURL = false;

while(values.hasNext())
{
    String s = ((Text)values.next()).toString().trim();
    if(s.startsWith("%"))
    {
        fromList.add(Long.parseLong(s.substring(1)));
    }
    else if(s.startsWith("&"))
    {
        toList.add(Long.parseLong(s.substring(1)));
    }
    else if(!gotURL)
    {
        gotURL = true;
        ownURL = s;
    }
}

if(gotURL && !fromList.isEmpty()) //already get URL and has fromlist
{
    StringBuffer resultValue = new StringBuffer();
    resultValue.append(ownURL);
    resultValue.append("\t");
    resultValue.append(toList.size());
    resultValue.append("\t");

    Iterator<Long> iterator = toList.iterator();
    while(iterator.hasNext())
    {
        resultValue.append(iterator.next().toString());
        resultValue.append("\t");
    }
    output.collect(key, new Text(resultValue.toString()));
}
}

public static class Reduce1ForGeneralcsd extends MapReduceBase implements
    Reducer<WritableComparable, Text, WritableComparable, Text>
{
    public void reduce(WritableComparable key, Iterator<Text> values,
        OutputCollector<WritableComparable, Text> output,
        Reporter reporter) throws IOException
    {
        TreeSet<Long> toList = new TreeSet<Long>();
        TreeSet<Long> fromList = new TreeSet<Long>();

```

```

String ownURL = new String();
boolean gotURL = false;

while(values.hasNext())
{
    String s = ((Text)values.next()).toString().trim();
    if(s.startsWith("%")
    {
        fromList.add(Long.parseLong(s.substring(1)));
    }
    else if(s.startsWith("&")
    {
        toList.add(Long.parseLong(s.substring(1)));
    }
    else if(!gotURL)
    {
        gotURL = true;
        ownURL = s;
    }
}

if(gotURL ) //already get URL, this include all nodes
{
    StringBuffer resultValue = new StringBuffer();
    resultValue.append(ownURL);
    resultValue.append("\t");
    resultValue.append(toList.size());
    resultValue.append("\t");

    Iterator<Long> iterator = toList.iterator();
    while(iterator.hasNext())
    {
        resultValue.append(iterator.next().toString());
        resultValue.append("\t");
    }
    output.collect(key, new Text(resultValue.toString()));
}
}
}

```

```

public static class Map2ForGeneral extends MapReduceBase implements
    Mapper<WritableComparable, Writable, Text, Text> {
    private Text fromURL = new Text();

    public void map(WritableComparable key, Writable value, OutputCollector<Text, Text>
        output, Reporter reporter) throws IOException {
        String splitString[] = ((Text)value).toString().split("\t");
    }
}

```

```

        fromURL.set(splitString[1]);
        String strHashFrom = splitString[0];

        for(int i=3; i < splitString.length; i++)
        {
            output.collect(new Text(splitString[i]), new Text(strHashFrom));
        }

        output.collect(new Text(strHashFrom), new Text("\t"+splitString[1]));
    }
}

public static class Reduce2ForGeneral extends MapReduceBase implements
    Reducer<WritableComparable, Text, WritableComparable, Text>
{
    public void reduce(WritableComparable key, Iterator<Text> values,
        OutputCollector<WritableComparable, Text> output,
        Reporter reporter) throws IOException
    {
        ArrayList<String> fromList = new ArrayList<String>();
        boolean validURL = false;

        while(values.hasNext())
        {
            String s = ((Text)values.next()).toString().trim();
            if(s.startsWith("$"))
            {
                validURL = true;
                output.collect(key, new Text(s));
            }
            else
            {
                fromList.add(s);
            }
        }

        if(validURL)
        {
            Iterator<String> iterator = fromList.iterator();
            while(iterator.hasNext())
            {
                output.collect(key, new Text(iterator.next()));
            }
        }
    }
}

```

```

public static class Map3ForGeneral extends MapReduceBase implements
    Mapper<WritableComparable, Writable, Text, Text> {

    public void map(WritableComparable key, Writable value, OutputCollector<Text, Text>
        output, Reporter reporter) throws IOException {
        String splitString[] = ((Text)value).toString().split("\t");
        if(splitString.length == 3)
        {
            output.collect(new Text(splitString[0]), new Text("$\t"+splitString[2]));
        }
        else if (splitString.length==2)
        {
            output.collect(new Text(splitString[1]), new Text(splitString[0]));
        }
    }
}

```

```

public static class Reduce3ForGeneral extends MapReduceBase implements
    Reducer<WritableComparable, Text, WritableComparable, Text>
{
    public void reduce(WritableComparable key, Iterator<Text> values,
        OutputCollector<WritableComparable, Text> output,
        Reporter reporter) throws IOException
    {
        ArrayList<String> toList = new ArrayList<String>();
        String ownURL = new String();
        boolean validURL = false;

        while(values.hasNext())
        {
            String s = ((Text)values.next()).toString().trim();
            if(s.startsWith("$"))
            {
                ownURL = s.substring(1).trim();
                validURL = true;
            }
            else
            {
                toList.add(s);
            }
        }

        if(validURL)
        {
            StringBuffer resultValue = new StringBuffer();
            resultValue.append(ownURL);

```

```

        resultValue.append("\t");
        resultValue.append(toList.size());
        resultValue.append("\t");

        Iterator<String> iterator = toList.iterator();
        while(iterator.hasNext())
        {
            resultValue.append(iterator.next().toString());
            resultValue.append("\t");
        }
        output.collect(key, new Text(resultValue.toString()));
    }
}

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

    boolean parameterFault = true;
    if (args.length==2)
    {

        if (args[0].equals("-pagerank"))
        {
            parameterFault = false;
            JobConf conf1 = new JobConf(GraphClean.class);
            conf1.setJobName("PageRankGraphClean "+args[1].trim());
            conf1.setMapOutputKeyClass(LongWritable.class);
            conf1.setMapOutputValueClass(Text.class);
            conf1.setOutputKeyClass(Text.class);
            conf1.setOutputValueClass(Text.class);
            conf1.setMapperClass(MapForPageRank.class);
            conf1.setReducerClass(ReduceForPageRank.class);
            conf1.setInputFormat(TextInputFormat.class);
            conf1.setOutputFormat(TextOutputFormat.class);
            FileInputFormat.setInputPaths(conf1, new Path(args[1].trim()));
            FileOutputFormat.setOutputPath(conf1,
                new Path(args[1].trim()+"ForPageRank"));
            JobClient.runJob(conf1);
        }
        else if (args[0].equals("-HA"))
        {
            parameterFault = false;
            JobConf conf1 = new JobConf(GraphClean.class);
            conf1.setJobName("HAGraphClean "+args[1].trim());
            conf1.setMapOutputKeyClass(LongWritable.class);
            conf1.setMapOutputValueClass(Text.class);
            conf1.setOutputKeyClass(Text.class);

```

```

        conf1.setOutputValueClass(Text.class);
        conf1.setMapperClass(MapForHA.class);
        conf1.setReducerClass(ReduceForHA.class);
    conf1.setInputFormat(TextInputFormat.class);
    conf1.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(conf1, new Path(args[1].trim()));
    FileOutputFormat.setOutputPath(conf1, new Path(args[1].trim()+"ForHA"));
        JobClient.runJob(conf1);
    }
    else if(args[0].equals("-c") | args[0].equals("-cs")
            | args[0].equals("-cd") | args[0].equals("-csd"))
    {
        parameterFault = false;

        JobConf conf1 = new JobConf(GraphClean.class);
        conf1.setJobName("GeneralCleanup "+
            args[0].substring(1)+"1 "+args[1].trim());
        conf1.setMapOutputKeyClass(LongWritable.class);
        conf1.setMapOutputValueClass(Text.class);
        conf1.setOutputKeyClass(Text.class);
        conf1.setOutputValueClass(Text.class);
        conf1.setMapperClass(Map1ForGeneral.class);

        if(args[0].equals("-c"))
        {
            conf1.setReducerClass(Reduce1ForGeneralc.class);
        }
        else if(args[0].equals("-cs"))
        {
            conf1.setReducerClass(Reduce1ForGeneralcs.class);
        }
        else if(args[0].equals("-cd"))
        {
            conf1.setReducerClass(Reduce1ForGeneralcd.class);
        }
        else if(args[0].equals("-csd"))
        {
            conf1.setReducerClass(Reduce1ForGeneralcsd.class);
        }
        conf1.setInputFormat(TextInputFormat.class);
        conf1.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf1, new Path(args[1].trim()));
        FileOutputFormat.setOutputPath(conf1, new
Path(args[1].trim()+"General"+args[0].substring(1)+"temp"));
        JobClient.runJob(conf1);

        JobConf conf2 = new JobConf(GraphClean.class);
        conf2.setJobName("GeneralCleanup "+

```

```

        args[0].substring(1)+"2 "+args[1].trim());
        conf2.setOutputKeyClass(Text.class);
        conf2.setOutputValueClass(Text.class);
        conf2.setMapperClass(Map2ForGeneral.class);
        conf2.setReducerClass(Reduce2ForGeneral.class);
        conf2.setInputFormat(TextInputFormat.class);
        conf2.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf2,
            new Path(args[1].trim()+"General"+args[0].substring(1)+"temp"));
        FileOutputFormat.setOutputPath(conf2,
            new Path(args[1].trim()+"General"+args[0].substring(1)+"temp2"));
        JobClient.runJob(conf2);

        JobConf conf3 = new JobConf(GraphClean.class);
        conf3.setJobName("GeneralCleanup "+
            args[0].substring(1)+"3 "+args[1].trim());
        conf3.setOutputKeyClass(Text.class);
        conf3.setOutputValueClass(Text.class);
        conf3.setMapperClass(Map3ForGeneral.class);
        conf3.setReducerClass(Reduce3ForGeneral.class);
        conf3.setInputFormat(TextInputFormat.class);
        conf3.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf3,
            new Path(args[1].trim()+"General"+args[0].substring(1)+"temp2"));
        FileOutputFormat.setOutputPath(conf3,
            new Path(args[1].trim()+"General"+args[0].substring(1)));
        JobClient.runJob(conf3);
    }
}

if(parameterFault)
{
    System.out.print("Usage: hadoop jar <package name> GraphClean <-COMMAND>
        <src>\n" +
        "where COMMAND is one of:\n" +
        " HA    generate Hub and Authority input\n" +
        " pagerank generate pagerank input\n" +
        " c      output core node\n" +
        " cs     output core node and source node\n" +
        " cd     output core node and destination node\n" +
        " csd    output core node, source node and destination node\n" +
        "\n" +
        "Please make sure to use the right parameters, or you will see this
        message.\n" + "\n");
}
}
}

```