Large Project

1. Introduction

With the development of mobile network and smartphone, it becomes more convenient for people to write and release their microblogs. According to the CEO of Twitter Dick costolo, up to 2012 March, there are 140 million active users using twitter, and these users create 340 million microblogs every day. This is only the data from one social network company. As of May 2007, there are 111 microblogging websites in the world, which will create huge amount of data every day. Although there are many no subject matters and daily chores in the data, but if do some statistics on the frequency of words or phrase, then the popular things and their lifecycle can be found from mining.

The aim of the project is mining out the popular things that happened recently and their life circle. This kind of information can be regard a commercial use. Such as, it can be used by the advertisement company to add the popular things in their website to increase visitor value and know the time when should they change to other advertisement. More importantly, not only the popular word will be find, but also their life circle (occurred time, spreading speed, slow down trend). With a life circle analysis, the rate of a popular thing can be measured and their commercial value can be estimate. Then the information can be use more wisely.

2. Dataset

2.1 Data obtained

We got 2 kind of data base on different use: testing data and real data.

- Testing data: First, we have some small sets of data at the number of 100. This data set is created by ourselves and for testing the basic process of the code and help programming. Then, we have a small set of data that contains 50000 of data which is selected in the real data sets. This data set is use for test some situation that we may not covered with the data set created by ourselves.
- Real data: this is the real data set that collected by Memetracker (memetracker.org). The data collected the phrases/quote in the blogs and news media. The real data we use is quotes-2008-08.txt which is about 4 GB big.

2.2 Format of the data and preprocessing

The original data format is like this:

```
P    http://blogs.abcnews.com/politicalpunch/2008/09/obama-says-mc-1.html
T    2008-09-09 22:35:24
Q    that's not change
Q    you know you can put lipstick on a pig
Q    what's the difference between a hockey mom and a pit bull lipstick
Q    you can wrap an old fish in a piece of paper called change
L    http://reuters.com/article/politicsnews/idusn2944356420080901?pagenumber=1&virtualbrandchannel=10112
L    http://cbn.com/cbnnews/436448.aspx
L    http://voices.washingtonpost.com/thefix/2008/09/bristol_palin_is_pregnant.html?hpid=topnews
```

(P for URL, T for time of the post,
Q for phrase extracted from the text of the document,
L for hyper-links in the document)
In our situation we don’t need the Link part so the original data is preprocessing into this:

```
T    2008-09-09 22:35:24  C    that's not change you know you can put lipstick on a pig what's the difference between a hockey mom and a pit bull lipstick you can wrap an old fish in a piece of paper called change
```

(T for time, C for contents)

2.3 General idea.

We think about using Map-Reduce to doing this job. The original data is already like a map and we will do thing in following steps:

1) Get the meaningful words in the contents. Get rid of the words like: am, is, are, my, I, you. And get the words after like some verb word or after a special symbol like: “”, ——, my Pokémon.
2) Calculate the frequency of the word that appeared in a time area (a day is the minimum time area)
3) Find the most frequently appeared 10 words in a time area.
4) Find the hot level (times that mentioned in a day) of the word day by day.
5) Find the life cycle of the hot words.、

3. Method

3.1 Key Idea

Our key idea is to use the advantage of distribute running process of MapReduce to get rid of the no meaning words and get the meaningful words counted and listed.

The most difficult part is how to get the meaningful words out of a lot of other words. But the thing is that in a big set of documents, the most frequently appeared words will always be some words which are verbs, prepositions, articles, conjunctions, connections, etc. Like “the”, “of”, “and”. After get rid of all these words, the most frequently appeared word will be the key word of this set of documents. For example:

Trying to get an awesome new #PokemonGO episode up but everything's going wrong! Trying to get it up ASAP!

In this sentence, after we get rid of all that kind of words, only “#PokemonGO”, “episode”. And the word “PokemonGO” is the word we need to find.

To get the list of these words, we have make use of the website: http://www.101languages.net/. From this website we get the most frequently used words in different language. We also use the list that in https://github.com/first20hours/google-10000-english . Using these two sources we have built up a dictionary that contains about 10000 no meaning English word that are most frequent used and 1000 each in other language through 7 languages.

But compare the content with the dictionary need a lot of time. So we think about to use the MapReduce to spread the calculation to a lot of hardware to save time.

3.2 Main process of the MapReduce

The source data we get after the original data is preprocess are like below:

T #time C #content

The MapReduce process we will make the following task done:

1.All the letter to lower cases.
2.Get all the letter which are not 0-9 and a-z replace with a space.
3.Delete all the words that not contains any letter from a to z
4.Count the total appeared times of each words
5.Get the times happened in a day of each words

The task from 1 to 3 is finished in the Map part and the task 4 and 5 is finished in Reduce part.

3.1.1 Map part

Task 1: Use String.toLowerCases() method to make every letter to the lower case.

Task 2: Use String.replaceAll() method to make everything that is not a-z and 0-9 replaced with a single blank.

Task 3: Use String.replaceAll() method to replace all the thing except a-z then check the String length. If the length is 0, then delete the word.(not send out the word)

At the last the Map part will send out the content in the following structure:

[word, time]

e.g.: [obama,2008-09-09]

3.1.2 Reduce part

Task 4: using a counter to count the total words.

Task 5: create a hash table defined the key as the day, and value as the times appeared in this day.

At the last the reduces part will send output in the following structure:

[word, day_times, day_times,......]

eg: [obama, 2008-09-09 45, 2008-09-12 36]

4 Outcome

4.1 Developing part

4.1.1 Preprocess

We have built a java program to make the data into a format that we need. (Source code in Appendix A)

This part we are changing the data into the format for the MapReduce to read. (read line by line) We put the time at the head of each line and the content following behind. The time and content is split with a single tab and in the content each word is split with a single space.

4.1.2 Dictionary

We create several dictionaries for get rid of the no meaning words. The size of all the dictionaries is about 150kb. It is easy to send them between the distribute computers. The dictionary has covered 9 languages that most used in twitter and other media. English dictionary have about 10000 most used word because it is the most used language in the world. Dutch, French, German, Indonesian, Italian, Portuguese, Spanish, Swedish are the other 8 dictionaries and each of them has about 1000 mostly used words in it. The dictionary data is collected from http://www.101languages.net/ & https://github.com/first20hours/google-10000-english. The dictionary is created with one word one-line structure and in a decrease frequency order.

4.1.3 Main program (MapReduce)

We create a MapReduce program in JAVA with Hadoops. The program take the data set created by the preprocess and send out a word list that have a appeared frequency and a list of frequency in each day.

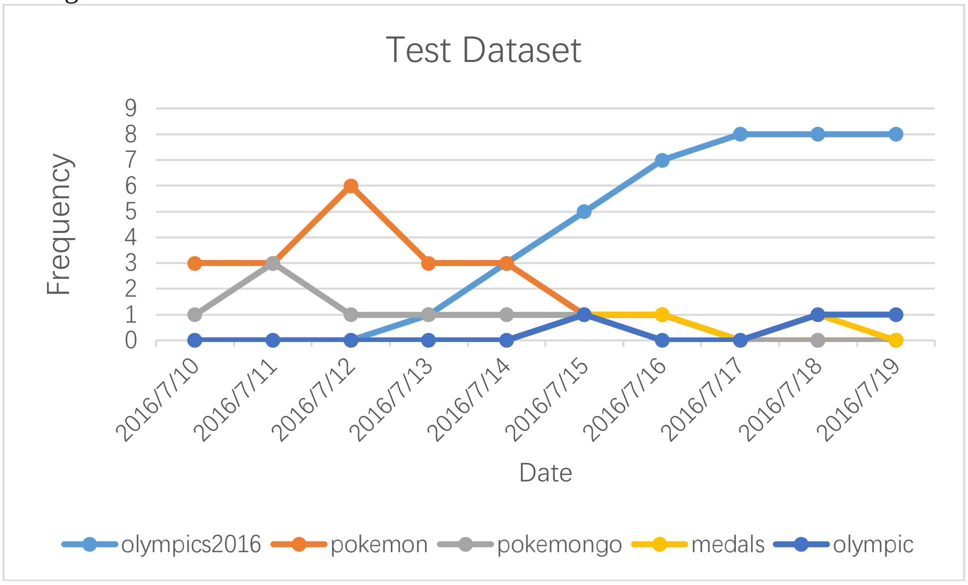
4.1.4 Data visualization

We create a java program read from the main program and generate a csv file that can open in the Excel. Each row is a word and each column is a date. At last we generate a chart that show the trend of the frequency of a word trough a month.

4.2 Testing part

4.2.1 Testing data

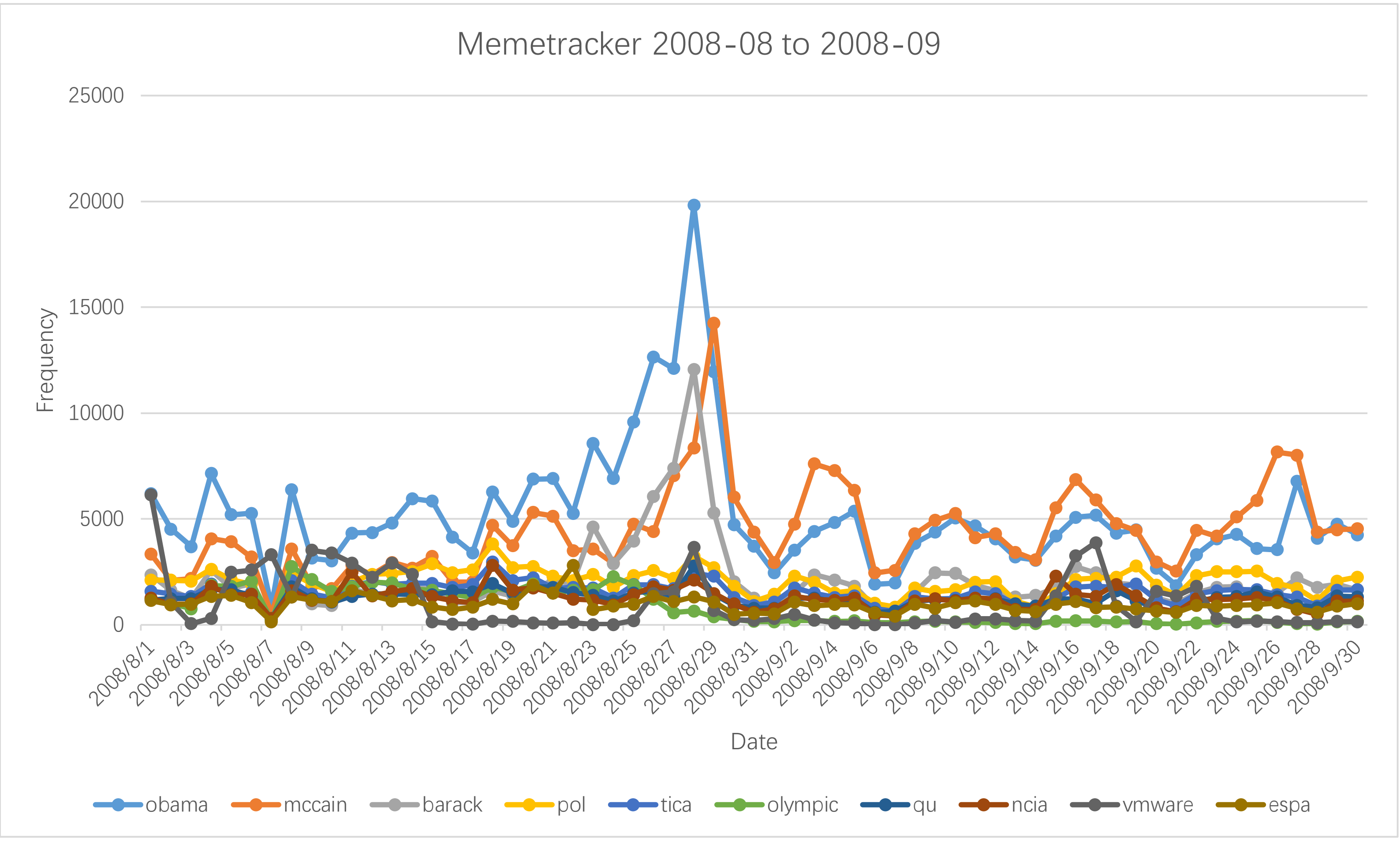
We have created a testing dataset that only contains 100 lines of data. The data is created by us and if the program runs well the result should be a decreasing trend of the word “Pokemon” and an increasing trend of word “Olympic”. And here is the result we run our program with the testing data.



The result is right but some of the same meaning word is separate because they have different appearance. But they didn't have any influence when the data becomes big and can be ignored.

4.2.2 Real data

We use the dataset quotes-2008-08.txt and quotes-2008-09. to test our program and the result is showing below.



The word "obama", "mccain", "barack" and "olympic" are the meaningful words we have found. After checking in the internet. On August 23, Obama announced his selection of Delaware Senator Joe Biden as his vice presidential running mate. Obama announced on November 13, 2008, that he would resign his Senate seat on November 16, 2008, before the start of the lame-duck session, to focus on his transition period for the presidency. Through the chart set we can see that after 2008-8-23 the words "obama" have an increasing trend. With the trend of the word "mccain" is increasing too. Olympic was start at 2008-08-08.

5 Analysis

After generate the result both from the testing data and real data. We have done some analysis about them. The testing data is working well with our program and generate a great looking chart. But when comes to the real data the chart only showing a trend that all the words will follows and there are so many unknown words among them.

We have think that the testing data is create by us and have no noise data so the chart will be great. But when it comes to the real data, there are lots of noise data that is hard to get rid of. Using better dictionaries will do some help in this part but can't fix it. Maybe evolve the dictionary with machine-learning can solve this problem. The real data have the same trend may cause by the data we have collected and the data that created in that day. After Obama announced, there must be a lot of people talking about this thing and will be a lot more data create in this day and other information will be related and then the trend will be the same. Extend the whole time area may help with this problem, for example, record for the whole year and then create a trend by month.

Future Work

Need to improve the dictionary, machine learning is a way to do it. The work right now still has some noise inside and the output we can still see some no meaning words.

Different words that have the same meaning should be combined into one word. Such as a word in two different language.

Figure out a way to get the real trend of a word without the influence of the volume of data. To use the percentage of all the data that created in a day may be a way to deal with it.

Combine different part of the program into one program and do all the thing automatically.

Appendix A: Preprocess

```
package main;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Scanner;

public class preprocess {
	HashMap<String, Integer> dic = new HashMap<String,Integer>();



	public void pre(String path, String output_path){
		File fi = new File(path);
		File fo = new File(output_path);
		String content = "";
		try {
			Scanner input = new Scanner(fi);
			BufferedWriter output = new BufferedWriter(new FileWriter(fo));
			boolean first = true;
			long count = 1;
			long show = 0;
			while(input.hasNextLine()){
				String line = input.nextLine().toLowerCase();

				count ++;
				if(count > 1000000){
					count = count - 1000000;
					show++;
					System.out.println(show +"000000");
				}
				if(line.startsWith("t")){
					if(!first){
						output.write("C\t" + content + "\n");
					}
					output.write(line.split(" ")[0] + "\t");
					content = "";
					output.flush();
					first = false;
				}
				if(line.startsWith("q")){
					String[] words = line.substring(2, line.length()).split(" ");
					for(int i = 0; i < words.length; i ++){
						if(dic.get(words[i]) == null){
							content = content + words[i] + " ";
						}
					}

				}
			}
			output.write("C\t" + content + "\n");
			output.flush();
			input.close();
			output.close();

		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}


	public void get_dicx(String path){
		File dicx = new File(path);
		try {
			Scanner input = new Scanner(dicx);
			while(input.hasNextLine()){
				String line = input.nextLine().toLowerCase();
				dic.put(line, 0);
			}
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}



	public void pre1(String path, String output_path){
		File fi = new File(path);
		File fo = new File(output_path);
		String content = "";
		try {
			Scanner input = new Scanner(fi);
			BufferedWriter output = new BufferedWriter(new FileWriter(fo));
			boolean first = true;
			long count = 1;
			long show = 0;
			while(input.hasNextLine()){
//				String line = input.nextLine();
				String line = input.nextLine().toLowerCase();

//				System.out.println(line);
				count ++;
				if(count > 10000){
					count = count - 10000;
					show++;
					System.out.println(show +"0000");
				}
				String[] tmp0 = line.split("\t");
				if(tmp0.length>3){
					String[] tmp = tmp0[3].split(" ");
					content = tmp0[0] + "\t" + tmp0[1]+ "\t" + tmp0[2] + "\t";
					for(int i = 1; i < tmp.length; i++){
						String delete = tmp[i].replaceAll("[^a-z0-9]", "");

						if(dic.get(delete)==null){
							content = content + delete + " ";
						}
					}
				}
				output.write(content + "\n");
				output.flush();
			}

			input.close();
			output.close();

		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}



	public static void main(String[] args){
		preprocess p = new preprocess();


		//get the dicx
		p.get_dicx("./dic/dic-italian.txt");
		p.get_dicx("./dic/dic-german.txt");
		p.get_dicx("./dic/dic-spanish.txt");
		p.get_dicx("./dic/dic-swedish.txt");
		p.get_dicx("./dic/dic-portuguese.txt");
		p.get_dicx("./dic/dic-dutch.txt");
		p.get_dicx("./dic/dic-indonesian.txt");
		p.get_dicx("./dic/dic-french.txt");

		p.get_dicx("dicx.txt");


//		p.pre("quotes_2008-08.txt", "2008-08.txt");
		p.pre("quotes_2008-09.txt", "2008-09.txt");




	}
}
```

Appendix B: Main Program

```
package main;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map.Entry;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Reducer.Context;
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.util.Tool;
import org.apache.hadoop.util.ToolRunner;


public class popular extends Configured implements Tool  {

        public static void main(String[] args) throws Exception {
              System.out.println(Arrays.toString(args));
              int res = ToolRunner.run(new Configuration(), new popular(), args);

              System.out.exit(res);
           }

           @Override
           public int run(String[] args) throws Exception {
              System.out.println(Arrays.toString(args));
              Job job = new Job(getConf(), "popular");
              job.setJarByClass(popular.class);
              job.setOutputKeyClass(Text.class);
              job.setOutputValueClass(Text.class);

              job.setMapperClass(Map.class);
              job.setReducerClass(Reduce.class);

              job.setInputFormatClass(TextInputFormat.class);
              job.setOutputFormatClass(TextOutputFormat.class);

              FileInputFormat.addInputPath(job, new Path(args[0]));
              FileOutputFormat.setOutputPath(job, new Path(args[1]));

              job.waitForCompletion(true);

              return 0;
```

```
            }

            public static class Map extends Mapper<LongWritable, Text, Text, Text> {
              private Text word = new Text();
              private Text time = new Text();

              @Override
              public void map(LongWritable key, Text value, Context context)
                   throws IOException, InterruptedException {
                String line = value.toString();
                String[] tmp = line.split("\t");
//              System.out.println(line);
                if(tmp.length > 3){
                     String[] words = tmp[3].split("\\s+");


                     for(int i = 0; i < words.length; i++){
                          String meaning = words[i].replaceAll("[^a-z]", "");
                          if(meaning.length()!=0){
                               word.set(words[i]);
                               time.set(tmp[1]);
                               context.write(word, time);
                          }
                     }
                }
              }
            }

            public static class Reduce extends Reducer<Text, Text, Text, Text> {
              @Override
              public void reduce(Text key, Iterable<Text> values, Context context)
                   throws IOException, InterruptedException {
                int sum_all = 0;
                HashMap<String,Integer> times = new HashMap<String,Integer>();

                for (Text val : values) {
                  sum_all ++;
                  String time = val.toString();
                  if(times.get(time) == null){
                     times.put(time, 1);
                  }else{
                     times.put(time, times.get(time) + 1);
                  }
                }
                String data = sum_all + "\t";
                for(Entry<String,Integer> entry : times.entrySet()){
                     data = data + entry.getKey() + " " + entry.getValue() + "\t";
                }
                Text out = new Text();
                out.set(data);
                context.write(key, out);
              }
            }

}
```

Appendix C: Data Visualization

```
package main;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Scanner;

public class summary {

    public void run(String path, String output_path){
        File fi = new File(path);
        File fo = new File(output_path);
        ArrayList<String> data = new ArrayList<String>();
        HashMap<String, Integer> date = new HashMap<String,Integer>();
        ArrayList<String> first = new ArrayList<String>();
        ArrayList<String[]> chaos = new ArrayList<String[]>();

        try {
            Scanner input = new Scanner(fi);
            while(input.hasNextLine()){
                String line = input.nextLine();
                String[] content = new String[2];
                content[0] = line.split("\t")[1];
                content[1] = line;
                chaos.add(content);
            }
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        Comparator comparator = new Comparator<String[]>() {
            @Override
            public int compare(String[] o1, String[] o2) {
                int temp1 = Integer.parseInt(o1[0]);
                int temp2 = Integer.parseInt(o2[0]);
                return Integer.compare(temp2, temp1);
            }
        };
        Collections.sort(chaos,comparator);
        for(int i = 0; i < chaos.size(); i++){
            data.add(chaos.get(i)[1]);
        }



        for(int i = 0; i < data.size(); i++){
//              System.out.println(data.get(i));
            String[] tmp = data.get(i).split("\t");
            for(int j = 2; j < tmp.length; j++){
                date.put(tmp[j].split(" ")[0], 1);
            }
        }

        for(String key : date.keySet()){
            first.add(key);
        }

        Collections.sort(first);
        String[] order = new String[first.size()];
        first.toArray(order);

//      int collect = data.size();
        int collect = 1000;



        try {
            BufferedWriter output = new BufferedWriter(new FileWriter(fo));
            String out = ",";
            for(int i = 0; i < order.length; i ++){
                out = out + order[i] + ",";
            }
            output.write(out + "\n");
            output.flush();

            for(int i = 0; i < collect && i <data.size(); i++){
                String[] tmp = data.get(i).split("\t");
                out = tmp[0];
                for(int k = 0; k < order.length; k++){
                    boolean flag = true;
                    for(int j = 2; j < tmp.length; j++){
                        String[] temp = tmp[j].split(" ");
                        if(temp[0].equals(order[k])){
                            out = out + "," +temp[1];
                            flag = false;
                        }
                    }
                    if(flag){
                        out = out + ",0";
                    }
                }
                output.write(out + "\n");
                output.flush();
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void main(String[] args){
        summary s = new summary();
//      s.run(args[0], args[1]);
        s.run("./output/part-r-00000", "result.csv");
    }
}
```