ENGG4030/ ESTR4300 Fall 2016 Homework 2

Release date: Oct 14, 2016

Due date: Oct 31, 2016 23:59pm

The solution will be posted right after the deadline, so NO late homework will be accepted!

Every Student MUST include the following statement, together with his/her signature in the submitted homework.

I declare that the assignment submitted on Elearning system is original except for source material explicitly acknowledged, and that the same or related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website http://www.cuhk.edu.hk/policy/acade michonesty/.

Signed (Student_____孫維澤____) Date:_______24/10/2016_____________
Name_________Sun_Weize___________ SID_________1155062041____________

Submission notice:

● Submit your report in a single PDF document on Elearning

General homework policies:

A student may discuss the problems with others. However, the work a student turns in must be

created COMPLETELY by oneself ALONE. A student may not share ANY written work or pictures, nor may one copy answers from any source other than one' s own brain.

Each student **MUST LIST** on the homework paper the **name of every person he/she has discussed or worked with** . If the answer includes content from any other source, the student

MUST STATE THE SOURCE . Failure to do so is cheating and will result in sanctions.

Copying

answers from someone else is cheating even if one lists their name(s) on the homework.

If there is information you need to solve a problem but the information is not stated in the problem, try to find the data somewhere. If you cannot find it, state what data you need, make a

reasonable estimate of its value, and justify any assumptions you make. You will be graded not

only on whether your answer is correct, but also on whether you have done an intelligent analysis.

Q1:

Number of pairs:

$$C_I^2$$

Number of pairs with count > 1:

$$\min(C_b^2 \times B = \frac{b \times (b-1) \times B}{2}, C_I^2)$$

Number of frequent pairs:

$$\min(\frac{C_b^2 \times B}{c} = \frac{b \times (b-1) \times B}{2 \times c}, C_I^2)$$

Matrix method uses 4 bytes per pair

Table method uses 12 bytes per pair

	# of items	# of items > 1	# of frequent items	Storage method used
(a)	19000	19000	19000	Matrix method
(b)	10000	6000	600	Table method
(c)	10000	10000	10000	Matrix method

Q2:

(a)

My code for Apriori implementation:

```java
import java.io.IOException;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.StringTokenizer;
import java.util.Hashtable;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;

public class Apriori{
    public static void main(String[] args) throws IOException{
        double s = 0.005;
        double num = 945127;
        Hashtable<String, Integer>singletons = new Hashtable<String, 
Integer>();
        Hashtable<String, Integer>doubletons = new Hashtable<String, 
Integer>();
        BufferedReader bf = new BufferedReader(new FileReader("shakespeare-
basket"));
```

```
        String line;
        System.out.println("Now generating singletons...");
        while((line = bf.readLine()) != null){
            StringTokenizer words = new StringTokenizer(line);
            while(words.hasMoreTokens()){
                String word = words.nextToken();
                if(singletons.containsKey(word)){
                    singletons.put(word, singletons.get(word)+1);
                }
                else{
                    singletons.put(word, 1);
                }
            }
        }
        System.out.println("In total " + singletons.size() + " candidates");
        System.out.println("Now cutting singletons...");
        for(Iterator<Map.Entry<String, Integer>>it = 
singletons.entrySet().iterator(); it.hasNext(); ){
            Map.Entry<String, Integer>temp = it.next();
            if(temp.getValue()/num < s){
                it.remove();
            }
        }
        System.out.println("In total " + singletons.size() + " singletons");
        System.out.println("Now generating doubletons...");
        BufferedReader bf2 = new BufferedReader(new FileReader("shakespeare-
basket"));
        while((line = bf2.readLine()) != null){
            StringTokenizer words = new StringTokenizer(line);
            List<String> candidates = new ArrayList<String>();
            while(words.hasMoreTokens()){
                String word = words.nextToken();
                if(singletons.containsKey(word)){
                    candidates.add(word);
                }
            }
            for(String key_one: candidates){
                for(String key_two: candidates){
                    if(key_one.compareTo(key_two) < 0){
                        if(doubletons.containsKey(key_one+","+key_two)){
                            doubletons.put(key_one+","+key_two, 
doubletons.get(key_one+","+key_two)+1);
                        }
                        else{
                            doubletons.put(key_one+","+key_two, 1);
                        }
                    }
                }
            }
        }
        System.out.println("In total " + doubletons.size() + " candidates");
        System.out.println("Now cutting doubletons...");
        for(Iterator<Map.Entry<String, Integer>>it = 
doubletons.entrySet().iterator(); it.hasNext(); ){
            Map.Entry<String, Integer>temp = it.next();
            if(temp.getValue()/num < s){
                it.remove();
            }
        }
        System.out.println("In total " + doubletons.size() + " doubletons");
        System.out.println("Now sorting...");
        ArrayList<Map.Entry<String, Integer>> sorted_doubletons = new 
ArrayList(doubletons.entrySet());
        Collections.sort(sorted_doubletons, new Comparator<Map.Entry<String, 
Integer>>(){
            public int compare(Map.Entry<String, Integer>pair_one, 
Map.Entry<String, Integer>pair_two){
                return pair_two.getValue().compareTo(pair_one.getValue());
            }
        });
        BufferedWriter bw = new BufferedWriter(new 
FileWriter("output.txt"));
        int i = 1;
        for(Map.Entry<String, Integer>pair: sorted_doubletons){
            bw.write(pair.getKey() + "\t" + pair.getValue() / num + "\n");
            if(i >= 40){
                break;
            }
            i++;
        }
        bw.close();
    }
}
```

Top 40 frequent doubletons:

```
thou,thy	0.06308675976879298
thee,thou	0.05183536180851885
thee,thy	0.04240065091781316
```

```
art,thou	0.031937506811253936
act,scene 0.030257309335147555
enter,scene	0.026785818202209863
act,enter	0.025395528854852331
act,exeunt	0.022841374757043235
hast,thou 0.022823387756354438
o,thou	0.02252713127442132
exeunt,scene	0.021656348829310768
enter,exeunt	0.02125111228438083
good,lord	0.0204237102526962
now,thou 0.01955927616076993
shall,thou	0.019199536146994003
duke,gloucester	0.019144517086063566
dost,thou 0.016816787585160514
o,thy	0.01680091670219981
lord,thou 0.016728968699444624
come,thou	0.016511008573450975
lord,shall 0.016432712217511507
come,shall	0.016362880332484418
good,sir	0.016289874270865184
shall,thy	0.016085668910104146
good,thou	0.016069798027143443
duke,lord	0.015850779842285747
more,thou	0.015750264250201297
enter,here	0.015506910711470522
sir,thou	0.015323866527990419
shall,thee	0.014642476619544252
good,shall	0.014512335379266491
enter,lord	0.014397006963085384
i'll,thee	0.014209730544149092
love,thou 0.014084879598191566
lord,now	0.014079589303871331
enter,now	0.014072182891823003
o,thee	0.014065834538638723
now,thy	0.013891254826070993
king,thou	0.013883848414022666
henry,v	0.013872209766518151
```

(b)

Code for SONPhase1.java:

```java
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.Hashtable;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
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.util.LineReader;

public class SONPhase1{
    private static int CHUNKSIZE = 200000;
    private static double s = 0.005;
    private static double num = 945127;
    public static class NewInputFormat extends TextInputFormat{
        @Override
        public RecordReader<LongWritable, Text> 
createRecordReader(InputSplit split, TaskAttemptContext context){
            return new NewRecordReader();
        }
    }
    public static class NewRecordReader extends RecordReader<LongWritable, 
Text>{
        private LineReader in;
        private LongWritable key;
        private Text value = new Text();
        private long start = 0;
        private long end = 0;
        private long pos = 0;
        private int maxLineLength;

        @Override
        public void close() throws IOException{
            if(in != null){
                in.close();
            }
        }
        @Override
        public LongWritable getCurrentKey() throws IOException,
InterruptedException{
            return key;
        }
        @Override
        public Text getCurrentValue() throws IOException,
InterruptedException{
            return value;
        }
        @Override
        public float getProgress() throws IOException, InterruptedException{
            if(start == end){
                return 0.0f;
            }
            else{
                return Math.min(1.0f, (pos - start) / (float)(end - start));
            }
        }
        @Override
        public void initialize(InputSplit genericSplit, TaskAttemptContext
context) throws IOException, InterruptedException{
            FileSplit split = (FileSplit) genericSplit;
            final Path file = split.getPath();
            Configuration conf = context.getConfiguration();
            this.maxLineLength =
conf.getInt("mapred.linerecordreader.maxlength", Integer.MAX_VALUE);
            FileSystem fs = file.getFileSystem(conf);
            start = split.getStart();
            end = start + split.getLength();
            boolean skipFirstLine = false;
            FSDataInputStream filein = fs.open(split.getPath());

            if(start != 0){
                skipFirstLine = true;
                start--;
                filein.seek(start);
            }
            in = new LineReader(filein, conf);
            if(skipFirstLine){
                start += in.readLine(new Text(), 0,
(int)Math.min((long)Integer.MAX_VALUE, end - start));
            }
            this.pos = start;
        }
        @Override
        public boolean nextKeyValue() throws IOException,
InterruptedException{
            if(key == null){
                key = new LongWritable();
            }
            key.set(pos);
            if(value == null){
                value = new Text();
            }
            value.clear();
            final Text endline = new Text("\n");
            int newSize = 0;
            for(int i = 0; i < CHUNKSIZE; i++){
                Text v = new Text();
                while(pos < end){
                    newSize = in.readLine(v, maxLineLength,
Math.max((int)Math.min(Integer.MAX_VALUE, end - pos), maxLineLength));
                    value.append(v.getBytes(), 0, v.getLength());
                    value.append(endline.getBytes(), 0, endline.getLength());
                    if(newSize == 0){
                        break;
                    }
                    pos += newSize;
                    if(newSize < maxLineLength){
                        break;
                    }
                }
            }
            if(newSize == 0){
                key = null;
                value = null;
                return false;
            }
            else{
                return true;
            }
        }
    }
    public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable>{
        public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException{
            String lines = value.toString();
            String[] lineArr = lines.split("\n");
            int lcount = lineArr.length;
            double chunks = s * lcount / num;
            Hashtable<String, Integer>singletons = new Hashtable<String,
Integer>();
            Hashtable<String, Integer>doubletons = new Hashtable<String,
Integer>();
            for(String line: lineArr){
                StringTokenizer words = new StringTokenizer(line);
                while(words.hasMoreTokens()){
                    String word = words.nextToken();
                    if(singletons.containsKey(word)){
                        singletons.put(word, singletons.get(word) + 1);
                    }
                    else{
                        singletons.put(word, 1);
                    }
                }
            }
            for(Iterator<Entry<String, Integer>>it =
singletons.entrySet().iterator(); it.hasNext(); ){
                Entry<String, Integer>temp = it.next();
                if((double)temp.getValue() / lcount < s){
                    it.remove();
                }
            }
            for(String line: lineArr){
                StringTokenizer words = new StringTokenizer(line);
                List<String> candidates = new ArrayList<String>();
                while(words.hasMoreTokens()){
                    String word = words.nextToken();
                    if(singletons.containsKey(word)){
                        candidates.add(word);
                    }
                }
                for(String keyOne: candidates){
                    for(String keyTwo: candidates){
                        if(keyOne.compareTo(keyTwo) < 0){
                            if(doubletons.containsKey(keyOne+","+keyTwo)){
                                doubletons.put(keyOne+","+keyTwo,
doubletons.get(keyOne+","+keyTwo) + 1);
                            }
                            else{
                                doubletons.put(keyOne+","+keyTwo, 1);
                            }
                        }
                    }
                }
            }
            for(Iterator<Entry<String, Integer>>it =
doubletons.entrySet().iterator(); it.hasNext(); ){
                Entry<String, Integer>temp = it.next();
                if((double)temp.getValue() / lcount < s){
                    it.remove();
                }
            }
            ArrayList<Entry<String, Integer>>sorted_doubletons = new
ArrayList(doubletons.entrySet());
            Collections.sort(sorted_doubletons, new Comparator<Entry<String,
Integer>>(){
                public int compare(Entry<String, Integer>pairOne,
Entry<String, Integer>pairTwo){
                    return pairTwo.getValue().compareTo(pairTwo.getValue());
                }
            });
            for(Entry<String, Integer>pair: sorted_doubletons){
                context.write(new Text(pair.getKey()), new
IntWritable(pair.getValue()));
            }
        }
    }
    public static void main(String[] args) throws Exception{
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "SONPhaseOne");
        job.setJarByClass(SONPhase1.class);
        job.setInputFormatClass(NewInputFormat.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reducer.class);
        job.setReducerClass(Reducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

Code for SONPhase2.java:

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.StringTokenizer;
import java.util.HashSet;
import java.util.Set;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
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.util.LineReader;

public class SONPhase2{
    private static int CHUNKSIZE = 200000;
    private static double s = 0.005;
    private static double num = 945127;
    public static class NewInputFormat extends TextInputFormat{
        @Override
        public RecordReader<LongWritable, Text>createRecordReader(InputSplit
split, TaskAttemptContext context){
            return new NewRecordReader();
        }
    }
    public static class NewRecordReader extends RecordReader<LongWritable, 
Text>{
        private LineReader in;
        private LongWritable key;
        private Text value = new Text();
        private long start = 0;
        private long end = 0;
        private long pos = 0;
        private int maxLineLength;

        @Override
        public void close() throws IOException{
            if(in != null){
                in.close();
            }
        }
        @Override
        public LongWritable getCurrentKey() throws IOException, 
InterruptedException{
            return key;
        }
        @Override
        public Text getCurrentValue() throws IOException, 
InterruptedException{
            return value;
        }
        @Override
        public float getProgress() throws IOException, InterruptedException{
            if(start == end){
                return 0.0f;
            }
            else{
                return Math.min(1.0f, (pos - start) / (float)(end - start));
            }
        }
        @Override
        public void initialize(InputSplit genericSplit, TaskAttemptContext 
context) throws IOException, InterruptedException{
            FileSplit split = (FileSplit) genericSplit;
            final Path file = split.getPath();
            Configuration conf = context.getConfiguration();
            this.maxLineLength = 
conf.getInt("mapred.linerecordreader.maxlength", Integer.MAX_VALUE);
            FileSystem fs = file.getFileSystem(conf);
            start = split.getStart();
            end = start + split.getLength();
            boolean skipFirstLine = false;
            FSDataInputStream filein = fs.open(split.getPath());
            
            if(start != 0){
                skipFirstLine = true;
                start--;
                filein.seek(start);
            }
            in = new LineReader(filein, conf);
            if(skipFirstLine){
                start += in.readLine(new Text(), 0, 
(int)Math.min((long)Integer.MAX_VALUE, end - start));
            }
            this.pos = start;
        }
        @Override
        public boolean nextKeyValue() throws IOException, 
InterruptedException{
            if(key == null){
                key = new LongWritable();
            }
            key.set(pos);
            if(value == null){
                value = new Text();
            }
            value.clear();
            final Text endline = new Text("\n");
            int newSize = 0;
            for(int i = 0; i < CHUNKSIZE; i++){
                Text v = new Text();
                while(pos < end){
                    newSize = in.readLine(v, maxLineLength, 
Math.max((int)Math.min(Integer.MAX_VALUE, end - pos), maxLineLength));
                    value.append(v.getBytes(), 0, v.getLength());
                    value.append(endline.getBytes(), 0, endline.getLength());
                    if(newSize == 0){
                        break;
                    }
                    pos += newSize;
                    if(newSize < maxLineLength){
                        break;
                    }
                }
            }
            if(newSize == 0){
                key = null;
                value = null;
                return false;
            }
            else{
                return true;
            }
        }
    }
    public static class Map extends Mapper<LongWritable, Text, Text, 
IntWritable>{
        public void map(LongWritable key, Text value, Context context) 
throws IOException, InterruptedException{
            Configuration conf = context.getConfiguration();
            String[] candidatesString = conf.get("candidates").split("\\s+");
            Set<String> candidates = new HashSet<String>();
            for(String candidate: candidatesString){
                candidates.add(candidate);
            }
            String lines = value.toString();
            String[] lineArr = lines.split("\n");
            int lcount = lineArr.length;
            for(String line: lineArr){
                String[] words = line.split("\\s+");
                for(String wordOne: words){
                    for(String wordTwo: words){
                        if(wordOne.compareTo(wordTwo) < 0 && 
candidates.contains(wordOne+","+wordTwo)){
                            context.write(new Text(wordOne+","+wordTwo), new 
IntWritable(1));
                        }
                    }
                }
            }
        }
    }
    public static class Reduce extends Reducer<Text, IntWritable, Text, 
IntWritable>{
        public void reduce(Text key, Iterable<IntWritable> values, Context 
context) throws IOException, InterruptedException{
            int sum = 0;
            for(IntWritable val: values){
                sum += val.get();
            }
            if(sum / num >= s){
                context.write(key, new IntWritable(sum));
            }
        }
    }
    public static void main(String[] args) throws Exception{
        BufferedReader bf = new BufferedReader(new 
FileReader("candidates"));
        String candidates = "";
        String line;
        while((line = bf.readLine()) != null){
            String[] parts = line.split("\\s+");
            candidates += parts[0];
            candidates += " ";
        }
        Configuration conf = new Configuration();
        conf.set("candidates", candidates);
        Job job = Job.getInstance(conf, "SONPhaseTwo");
        job.setJarByClass(SONPhase2.class);
        job.setInputFormatClass(NewInputFormat.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setCombinerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

Code for Sorter.java:

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.Hashtable;
```

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;

public class Sorter{
    public static void main(String[] args) throws IOException{
        Hashtable<String, Integer> doubletons = new Hashtable<String,
Integer>();
        BufferedReader bf = new BufferedReader(new
FileReader("doubletons"));
        String line;
        while((line = bf.readLine()) != null){
            String[] parts = line.split("\\s+");
            doubletons.put(parts[0], Integer.parseInt(parts[1]));
        }
        ArrayList<Map.Entry<String, Integer>> sortedDoubletons = new
ArrayList(doubletons.entrySet());
        Collections.sort(sortedDoubletons, new Comparator<Map.Entry<String,
Integer>>(){
            public int compare(Map.Entry<String, Integer>pairOne,
Map.Entry<String, Integer>pairTwo){
                return pairTwo.getValue().compareTo(pairOne.getValue());
            }
        });
        BufferedWriter bw = new BufferedWriter(new FileWriter("SONOutput"));
        int i = 1;
        double num = 945127;
        for(Map.Entry<String, Integer>pair: sortedDoubletons){
            bw.write(pair.getKey() + "\t" + pair.getValue() / num + "\n");
            if(i >= 40){
                break;
            }
            i++;
        }
        bw.close();
    }
}
```

Code for the script:

```bash
#!/bin/bash
```

```
javac -classpath "$CLASSPATH" -d . SONPhase1.java
jar -cf SONPhase1.jar SONPhase1*.class;
javac -classpath "$CLASSPATH" -d . SONPhase2.java
jar -cf SONPhase2.jar SONPhase2*.class;
javac Sorter.java

rm candidates doubletons SONOutput


hadoop dfs -rm -R /user/1155062041/candidates;
hadoop dfs -rm -R /user/1155062041/doubletons;
hadoop jar SONPhase1.jar SONPhase1 /user/1155062041/shakespeare-basket 
/user/1155062041/candidates
hadoop dfs -copyToLocal /user/1155062041/candidates/part-r-00000
mv part-r-00000 candidates
hadoop jar SONPhase2.jar SONPhase2 /user/1155062041/shakespeare-basket 
/user/1155062041/doubletons
hadoop dfs -copyToLocal /user/1155062041/doubletons/part-r-00000
mv part-r-00000 doubletons
java Sorter
```

Output of 2b:

```
thou,thy	0.06308675976879298
thee,thou	0.05183536180851885
thee,thy	0.042400650917813164
art,thou	0.031937506811253936
act,scene	0.03025730933514755
enter,scene	0.026785818202209863
act,enter	0.02539552885485231
act,exeunt	0.022841374757043235
hast,thou	0.022823387756354438
o,thou	0.02252713127442132
exeunt,scene	0.021656348829310768
enter,exeunt	0.02125111228438083
good,lord	0.0204237102526962
now,thou	0.01955927616076993
shall,thou	0.019199536146994003
duke,gloucester	0.019144517086063566
dost,thou	0.016816787585160514
o,thy	0.01680091670219981
lord,thou	0.016728968699444624
come,thou	0.016511008573450975
lord,shall	0.016432712217511507
come,shall	0.016362880332484418
```

```
good,sir	0.016289874270865184
shall,thy	0.016085668910104146
good,thou	0.016069798027143443
duke,lord	0.015850779842285747
more,thou	0.015750264250201297
enter,here	0.015506910711470522
sir,thou	0.015323866527990419
shall,thee	0.014642476619544252
good,shall	0.014512335379266491
enter,lord	0.014397006963085384
i'll,thee	0.014209730544149092
love,thou	0.014084879598191566
lord,now	0.014079589303871331
enter,now	0.014072182891823003
o,thee	0.014065834538638723
now,thy	0.013891254826070993
king,thou	0.013883848414022666
henry,v	0.013872209766518151
```

Comparison of time:

A<B

(c)

Code for Apriori with randomness:

```java
import java.io.IOException;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.StringTokenizer;
import java.util.Hashtable;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;

public class AprioriRandom{
    public static void main(String[] args) throws IOException{
        double s = 0.005;
        double num = 94512;
        Hashtable<String, Integer>singletons = new Hashtable<String, 
Integer>();
```

```
        Hashtable<String, Integer>doubletons = new Hashtable<String, 
Integer>();
        BufferedReader bf = new BufferedReader(new FileReader("shakespeare-
basket"));
        String line;
        List<String> record = new ArrayList<String>();
        System.out.println("Now recording");
        while((line = bf.readLine()) != null){
            record.add(line);
        }
        Collections.shuffle(record);
        List<String> selected = new ArrayList<String>();
        selected = record.subList(0, (int)num);
        System.out.println("Now generating singletons...");
        for(String temp: selected){
            StringTokenizer words = new StringTokenizer(temp);
            while(words.hasMoreTokens()){
                String word = words.nextToken();
                if(singletons.containsKey(word)){
                    singletons.put(word, singletons.get(word)+1);
                }
                else{
                    singletons.put(word, 1);
                }
            }
        }
        System.out.println("In total " + singletons.size() + " candidates");
        System.out.println("Now cutting singletons...");
        for(Iterator<Map.Entry<String, Integer>>it = 
singletons.entrySet().iterator(); it.hasNext(); ){
            Map.Entry<String, Integer>temp = it.next();
            if(temp.getValue()/num < s){
                it.remove();
            }
        }
        System.out.println("In total " + singletons.size() + " singletons");
        System.out.println("Now generating doubletons...");
        for(String temp: selected){
            StringTokenizer words = new StringTokenizer(temp);
            List<String> candidates = new ArrayList<String>();
            while(words.hasMoreTokens()){
                String word = words.nextToken();
                if(singletons.containsKey(word)){
                    candidates.add(word);
                }
            }
            for(String key_one: candidates){
                for(String key_two: candidates){
                    if(key_one.compareTo(key_two) < 0){
                        if(doubletons.containsKey(key_one+","+key_two)){
                            doubletons.put(key_one+","+key_two, 
doubletons.get(key_one+","+key_two)+1);
                        }
                        else{
                            doubletons.put(key_one+","+key_two, 1);
                        }
                    }
                }
            }
        }
        System.out.println("In total " + doubletons.size() + " candidates");
        System.out.println("Now cutting doubletons...");
        for(Iterator<Map.Entry<String, Integer>>it = 
doubletons.entrySet().iterator(); it.hasNext(); ){
            Map.Entry<String, Integer>temp = it.next();
            if(temp.getValue()/num < s){
                it.remove();
            }
        }
        System.out.println("In total " + doubletons.size() + " doubletons");
        System.out.println("Now sorting...");
        ArrayList<Map.Entry<String, Integer>> sorted_doubletons = new 
ArrayList(doubletons.entrySet());
        Collections.sort(sorted_doubletons, new Comparator<Map.Entry<String, 
Integer>>(){
            public int compare(Map.Entry<String, Integer>pair_one, 
Map.Entry<String, Integer>pair_two){
                return pair_two.getValue().compareTo(pair_one.getValue());
            }
        });
        BufferedWriter bw = new BufferedWriter(new 
FileWriter("outputRandomFull.txt"));
        int i = 1;
        for(Map.Entry<String, Integer>pair: sorted_doubletons){
            bw.write(pair.getKey() + "\t" + pair.getValue() / num + "\n");
        }
        bw.close();
    }
```

```
}
```

Output of AprioriRandom (only the top 40):

```
thou,thy	0.06293380734721517
thee,thou	0.05148552564753682
thee,thy	0.04254486202810225
art,thou	0.03198535635686474
act,scene	0.030038513627899104
enter,scene	0.026240054173015066
act,enter	0.02483282546131708
act,exeunt	0.023044692737430168
hast,thou	0.02267436939224649
o,thou	0.022431014051125783
exeunt,scene	0.021880819366852888
enter,exeunt	0.02154223802268495
good,lord	0.02058997799221263
shall,thou	0.01957423395970882
duke,gloucester	0.01926739461655663
now,thou	0.01904520060944642
good,sir	0.017309971220585745
o,thy	0.01698197054342306
shall,thy	0.016876163873370577
lord,thou	0.016685711867276114
dost,thou	0.016675131200270865
good,thou	0.016664550533265616
come,thou	0.016273065854071442
more,thou	0.016146097850008465
duke,lord	0.015976807177924497
come,shall	0.015818097172845776
enter,here	0.0156911291687828
sir,thou	0.015648806500761808
lord,shall	0.015490096495683088
shall,thee	0.015204418486541391
i'll,thee	0.014442610462163535
lord,now	0.014421449128153038
king,thou	0.014347384459116303
o,thee	0.014262739123074318
enter,now	0.014262739123074318
love,thou	0.01425215845606907
good,shall	0.014220416455053326
henry,v	0.01398764178093787
sir,well	0.013850093109869645
enter,lord	0.013744286439817167
```

Code for Check.java:

```
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import java.io.BufferedReader;
import java.io.FileReader;

public class Check{
    public static void main(String[] args) throws IOException{
        BufferedReader outputFull = new BufferedReader(new
FileReader("outputFull.txt"));
        BufferedReader outputRandomFull = new BufferedReader(new
FileReader("outputRandomFull.txt"));
        HashSet<String>trueDoubletons1 = new HashSet<String>();
        HashSet<String>trueDoubletons2 = new HashSet<String>();
        HashSet<String>randomDoubletons1 = new HashSet<String>();
        HashSet<String>randomDoubletons2 = new HashSet<String>();
        String line;
        while((line = outputFull.readLine()) != null){
            String[] parts = line.split("\\s+");
            trueDoubletons1.add(parts[0]);
            trueDoubletons2.add(parts[0]);
        }
        while((line = outputRandomFull.readLine()) != null){
            String[] parts = line.split("\\s+");
            randomDoubletons1.add(parts[0]);
            randomDoubletons2.add(parts[0]);
        }
        trueDoubletons1.removeAll(randomDoubletons1);
        randomDoubletons2.removeAll(trueDoubletons2);
        System.out.println("The number of false negative items is
"+trueDoubletons1.size());
        System.out.println("The number of false positive items is
"+randomDoubletons2.size());
    }
}
```

Comparison of running time:

C<A<B

Result after checking:

```
The number of false negative items is 17
The number of false positive items is 33
```

(d)

Code for SONPhase1.java:

```java
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.Hashtable;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
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.util.LineReader;

public class SONPhase1{
    private static int CHUNKSIZE = 200000;
    private static double s = 0.0025;
    private static double num = 945127;
    public static class NewInputFormat extends TextInputFormat{
        @Override
        public RecordReader<LongWritable, Text> 
createRecordReader(InputSplit split, TaskAttemptContext context){
            return new NewRecordReader();
        }
    }
    public static class NewRecordReader extends RecordReader<LongWritable, 
Text>{
        private LineReader in;
        private LongWritable key;
        private Text value = new Text();
        private long start = 0;
        private long end = 0;
        private long pos = 0;
        private int maxLineLength;

        @Override
        public void close() throws IOException{
            if(in != null){
                in.close();
            }
        }
        @Override
        public LongWritable getCurrentKey() throws IOException, 
InterruptedException{
            return key;
        }
        @Override
        public Text getCurrentValue() throws IOException, 
InterruptedException{
            return value;
        }
        @Override
        public float getProgress() throws IOException, InterruptedException{
            if(start == end){
                return 0.0f;
            }
            else{
                return Math.min(1.0f, (pos - start) / (float)(end - start));
            }
        }
        @Override
        public void initialize(InputSplit genericSplit, TaskAttemptContext 
context) throws IOException, InterruptedException{
            FileSplit split = (FileSplit) genericSplit;
            final Path file = split.getPath();
            Configuration conf = context.getConfiguration();
            this.maxLineLength = 
conf.getInt("mapred.linerecordreader.maxlength", Integer.MAX_VALUE);
            FileSystem fs = file.getFileSystem(conf);
            start = split.getStart();
            end = start + split.getLength();
```

```java
            boolean skipFirstLine = false;
            FSDataInputStream filein = fs.open(split.getPath());

            if(start != 0){
                skipFirstLine = true;
                start--;
                filein.seek(start);
            }
            in = new LineReader(filein, conf);
            if(skipFirstLine){
                start += in.readLine(new Text(), 0,
(int)Math.min((long)Integer.MAX_VALUE, end - start));
            }
            this.pos = start;
        }
        @Override
        public boolean nextKeyValue() throws IOException,
InterruptedException{
            if(key == null){
                key = new LongWritable();
            }
            key.set(pos);
            if(value == null){
                value = new Text();
            }
            value.clear();
            final Text endline = new Text("\n");
            int newSize = 0;
            for(int i = 0; i < CHUNKSIZE; i++){
                Text v = new Text();
                while(pos < end){
                    newSize = in.readLine(v, maxLineLength,
Math.max((int)Math.min(Integer.MAX_VALUE, end - pos), maxLineLength));
                    value.append(v.getBytes(), 0, v.getLength());
                    value.append(endline.getBytes(), 0, endline.getLength());
                    if(newSize == 0){
                        break;
                    }
                    pos += newSize;
                    if(newSize < maxLineLength){
                        break;
                    }
                }
            }
            if(newSize == 0){
                key = null;
                value = null;
                return false;
            }
            else{
                return true;
            }
        }
    }
    public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable>{
        public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException{
            String lines = value.toString();
            String[] lineArr = lines.split("\n");
            int lcount = lineArr.length;
            double chunks = s * lcount / num;
            Hashtable<String, Integer>singletons = new Hashtable<String,
Integer>();
            Hashtable<String, Integer>doubletons = new Hashtable<String,
Integer>();
            Hashtable<String, Integer>tripletons = new Hashtable<String,
Integer>();
            for(String line: lineArr){
                StringTokenizer words = new StringTokenizer(line);
                while(words.hasMoreTokens()){
                    String word = words.nextToken();
                    if(singletons.containsKey(word)){
                        singletons.put(word, singletons.get(word) + 1);
                    }
                    else{
                        singletons.put(word, 1);
                    }
                }
            }
            for(Iterator<Entry<String, Integer>>it =
singletons.entrySet().iterator(); it.hasNext(); ){
                Entry<String, Integer>temp = it.next();
                if((double)temp.getValue() / lcount < s){
                    it.remove();
                }
            }
            for(String line: lineArr){
                StringTokenizer words = new StringTokenizer(line);
                List<String> candidates = new ArrayList<String>();
                while(words.hasMoreTokens()){
                    String word = words.nextToken();
                    if(singletons.containsKey(word)){
                        candidates.add(word);
                    }
                }
                for(String keyOne: candidates){
                    for(String keyTwo: candidates){
                        if(keyOne.compareTo(keyTwo) < 0){
                            if(doubletons.containsKey(keyOne+","+keyTwo)){
                                doubletons.put(keyOne+","+keyTwo,
doubletons.get(keyOne+","+keyTwo) + 1);
                            }
                            else{
                                doubletons.put(keyOne+","+keyTwo, 1);
                            }
                        }
                    }
                }
            }
            for(Iterator<Entry<String, Integer>>it =
doubletons.entrySet().iterator(); it.hasNext(); ){
                Entry<String, Integer>temp = it.next();
                if((double)temp.getValue() / lcount < s){
                    it.remove();
                }
            }
            for(String line: lineArr){
                StringTokenizer words = new StringTokenizer(line);
                List<String> candidates = new ArrayList<String>();
                while(words.hasMoreTokens()){
                    String word = words.nextToken();
                    if(singletons.containsKey(word)){
                        candidates.add(word);
                    }
                }
                for(String keyOne: candidates){
                    for(String keyTwo: candidates){
                        for(String keyThree: candidates){
                            if(keyOne.compareTo(keyTwo) < 0 &&
keyOne.compareTo(keyThree) < 0 && keyTwo.compareTo(keyThree) < 0 &&
doubletons.containsKey(keyOne+","+keyTwo) &&
doubletons.containsKey(keyOne+","+keyThree) &&
doubletons.containsKey(keyTwo+","+keyThree)){

if(tripletons.containsKey(keyOne+","+keyTwo+","+keyThree)){

tripletons.put(keyOne+","+keyTwo+","+keyThree,
tripletons.get(keyOne+","+keyTwo+","+keyThree) + 1);
                                }
                                else{

tripletons.put(keyOne+","+keyTwo+","+keyThree, 1);
                                }
                            }
                        }
                    }
                }
            }
            for(Iterator<Entry<String, Integer>>it =
tripletons.entrySet().iterator(); it.hasNext(); ){
                Entry<String, Integer>temp = it.next();
                if((double)temp.getValue() / lcount < s){
                    it.remove();
                }
            }
            for(Iterator<Entry<String, Integer>>it =
tripletons.entrySet().iterator(); it.hasNext(); ){
                Entry<String, Integer>temp = it.next();
                context.write(new Text(temp.getKey()), new
IntWritable(temp.getValue()));
            }
        }
    }
    public static void main(String[] args) throws Exception{
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "SONPhaseOne");
        job.setJarByClass(SONPhase1.class);
        job.setInputFormatClass(NewInputFormat.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reducer.class);
        job.setReducerClass(Reducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

Code for SONPhase2.java:

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.StringTokenizer;
import java.util.HashSet;
import java.util.Set;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
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.util.LineReader;

public class SONPhase2{
    private static int CHUNKSIZE = 200000;
    private static double s = 0.0025;
    private static double num = 945127;
    public static class NewInputFormat extends TextInputFormat{
        @Override
        public RecordReader<LongWritable, Text>createRecordReader(InputSplit 
split, TaskAttemptContext context){
            return new NewRecordReader();
        }
    }
    public static class NewRecordReader extends RecordReader<LongWritable, 
Text>{
        private LineReader in;
        private LongWritable key;
        private Text value = new Text();
        private long start = 0;
        private long end = 0;
        private long pos = 0;
        private int maxLineLength;

        @Override
        public void close() throws IOException{
            if(in != null){
                in.close();
            }
        }
        @Override
        public LongWritable getCurrentKey() throws IOException, 
InterruptedException{
            return key;
        }
        @Override
        public Text getCurrentValue() throws IOException, 
InterruptedException{
            return value;
        }
        @Override
        public float getProgress() throws IOException, InterruptedException{
            if(start == end){
                return 0.0f;
            }
            else{
                return Math.min(1.0f, (pos - start) / (float)(end - start));
            }
        }
        @Override
        public void initialize(InputSplit genericSplit, TaskAttemptContext 
context) throws IOException, InterruptedException{
            FileSplit split = (FileSplit) genericSplit;
            final Path file = split.getPath();
            Configuration conf = context.getConfiguration();
            this.maxLineLength = 
conf.getInt("mapred.linerecordreader.maxlength", Integer.MAX_VALUE);
            FileSystem fs = file.getFileSystem(conf);
            start = split.getStart();
            end = start + split.getLength();
            boolean skipFirstLine = false;
            FSDataInputStream filein = fs.open(split.getPath());

            if(start != 0){
                skipFirstLine = true;
                start--;
                filein.seek(start);
            }
            in = new LineReader(filein, conf);
            if(skipFirstLine){
                start += in.readLine(new Text(), 0,
(int)Math.min((long)Integer.MAX_VALUE, end - start));
            }
            this.pos = start;
        }
        @Override
        public boolean nextKeyValue() throws IOException,
InterruptedException{
            if(key == null){
                key = new LongWritable();
            }
            key.set(pos);
            if(value == null){
                value = new Text();
            }
            value.clear();
            final Text endline = new Text("\n");
            int newSize = 0;
            for(int i = 0; i < CHUNKSIZE; i++){
                Text v = new Text();
                while(pos < end){
                    newSize = in.readLine(v, maxLineLength,
Math.max((int)Math.min(Integer.MAX_VALUE, end - pos), maxLineLength));
                    value.append(v.getBytes(), 0, v.getLength());
                    value.append(endline.getBytes(), 0, endline.getLength());
                    if(newSize == 0){
                        break;
                    }
                    pos += newSize;
                    if(newSize < maxLineLength){
                        break;
                    }
                }
            }
            if(newSize == 0){
                key = null;
                value = null;
                return false;
            }
            else{
                return true;
            }
        }
    }
    public static class Map extends Mapper<LongWritable, Text, Text, 
IntWritable>{
        public void map(LongWritable key, Text value, Context context) 
throws IOException, InterruptedException{
            Configuration conf = context.getConfiguration();
            String[] candidatesString = conf.get("candidates").split("\\s+");
            Set<String> candidates = new HashSet<String>();
            for(String candidate: candidatesString){
                candidates.add(candidate);
            }
            String lines = value.toString();
            String[] lineArr = lines.split("\n");
            int lcount = lineArr.length;
            for(String line: lineArr){
                String[] words = line.split("\\s+");
                for(String wordOne: words){
                    for(String wordTwo: words){
                        for(String wordThree: words){
                            if(wordOne.compareTo(wordTwo) < 0 && 
wordOne.compareTo(wordThree) < 0 && wordTwo.compareTo(wordThree) < 0 && 
candidates.contains(wordOne+","+wordTwo+","+wordThree)){
                                context.write(new 
Text(wordOne+","+wordTwo+","+wordThree), new IntWritable(1));
                            }
                        }
                    }
                }
            }
        }
    }
    public static class Reduce extends Reducer<Text, IntWritable, Text, 
IntWritable>{
        public void reduce(Text key, Iterable<IntWritable> values, Context 
context) throws IOException, InterruptedException{
            int sum = 0;
            for(IntWritable val: values){
                sum += val.get();
            }
            if(sum / num >= s){
                context.write(key, new IntWritable(sum));
            }
        }
    }
    public static void main(String[] args) throws Exception{
        BufferedReader bf = new BufferedReader(new 
FileReader("candidates"));
        String candidates = "";
        String line;
        while((line = bf.readLine()) != null){
            String[] parts = line.split("\\s+");
            candidates += parts[0];
            candidates += " ";
        }
        Configuration conf = new Configuration();
        conf.set("candidates", candidates);
        Job job = Job.getInstance(conf, "SONPhaseTwo");
        job.setJarByClass(SONPhase2.class);
        job.setInputFormatClass(NewInputFormat.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setCombinerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);
    }
}
```

Code for Sorter.java:

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.Hashtable;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
```

```java
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;

public class Sorter{
    public static void main(String[] args) throws IOException{
        Hashtable<String, Integer> doubletons = new Hashtable<String,
Integer>();
        BufferedReader bf = new BufferedReader(new
FileReader("tripletons"));
        String line;
        while((line = bf.readLine()) != null){
            String[] parts = line.split("\\s+");
            doubletons.put(parts[0], Integer.parseInt(parts[1]));
        }
        ArrayList<Map.Entry<String, Integer>> sortedDoubletons = new
ArrayList(doubletons.entrySet());
        Collections.sort(sortedDoubletons, new Comparator<Map.Entry<String,
Integer>>(){
            public int compare(Map.Entry<String, Integer>pairOne,
Map.Entry<String, Integer>pairTwo){
                return pairTwo.getValue().compareTo(pairOne.getValue());
            }
        });
        BufferedWriter bw = new BufferedWriter(new FileWriter("SONOutput"));
        int i = 1;
        double num = 945127;
        for(Map.Entry<String, Integer>pair: sortedDoubletons){
            bw.write(pair.getKey() + "\t" + pair.getValue() / num + "\n");
            if(i >= 20){
                break;
            }
            i++;
        }
        bw.close();
    }
}
```

Code for script:

```bash
#!/bin/bash

javac -classpath "$CLASSPATH" -d . SONPhase1.java
jar -cf SONPhase1.jar SONPhase1*.class;
```

```
javac -classpath "$CLASSPATH" -d . SONPhase2.java
jar -cf SONPhase2.jar SONPhase2*.class;
javac Sorter.java

rm candidates doubletons SONOutput


hadoop dfs -rm -R /user/1155062041/candidates;
hadoop dfs -rm -R /user/1155062041/doubletons;
hadoop dfs -rm -R /user/1155062041/tripletons;
hadoop jar SONPhase1.jar SONPhase1 /user/1155062041/shakespeare-basket 
/user/1155062041/candidates
hadoop dfs -copyToLocal /user/1155062041/candidates/part-r-00000
mv part-r-00000 candidates
hadoop jar SONPhase2.jar SONPhase2 /user/1155062041/shakespeare-basket 
/user/1155062041/tripletons
hadoop dfs -copyToLocal /user/1155062041/tripletons/part-r-00000
mv part-r-00000 tripletons
java Sorter
```

Output of 2d:

```
act,enter,scene 0.025198729906139597
thee,thou,thy	0.022302822795243392
act,exeunt,scene	0.021588633062011774
enter,exeunt,scene	0.018079051809968397
act,enter,exeunt	0.01807164539792007
art,thou,thy	0.011495809557868943
art,thee,thou	0.010888483769906055
hast,thou,thy	0.008915203988458693
o,thou,thy	0.00812906625247189
database,george,mason	0.007849738712363524
george,mason,university 0.007848680653499477
domain,public,references	0.007845506476907337
open,shakespeare,source 0.007843390359179242
domain,filelocalhost,references	0.00784127424145115
code,database,program	0.007645533351602483
code,database,george	0.007644475292738436
database,george,university	0.007642359175010343
database,mason,university	0.007642359175010343
filelocalhost,public,references	0.007641301116146295
mason,texts,university	0.007640243057282249
```

Q3:

(a)

Elements	S1	S2	S3	S4
H1	5	1	1	1
H2	2	2	2	2
H3	0	1	4	2

(b)

Only the third one is a true permutation.

(c)

	S1, S2	S1, S3	S1, S4	S2, S3	S2, S4	S3, S4
True	0	0	1/4	0	1/4	1/4
Estimated	1/3	1/3	1/3	2/3	2/3	2/3