### FAQs

- **PA1**

---

### Programming Assignment 1

- **Creating N-gram profile for a Wikipedia Corpus**
- **Due:** 5:00PM Sept 30 2019
- **Objectives**
  - Basic features of Hadoop distributed file system and MapReduce
  - Creating N-Gram profiles using Hadoop MapReduce

---

### N-Gram

- A contiguous sequence of N items from a given sequence of text or speech
- "We analyze large datasets"
- 1-grams (aka unigram)
  - We, analyze, large, datasets
- 2-grams (aka bigram)
  - (__, We), (We, analyze), (analyze, large), (large, datasets), (datasets, __)
  - [https://books.google.com/ngrams/info](https://books.google.com/ngrams/info)

---

### Requirements

1. extracting all distinct unigrams
2. computing the frequency of each unigram per article and also over the corpus
3. ranking the unigrams based on these frequencies

---

### Structure of PA1: Profile 1

- Extract unigrams
- Mapper
- Shuffle and sort
- Join
- Reduce
- Output
Programming Assignment 1

Profile 1
A list of unigrams that occurred at least once in the entire corpus (1G dataset). The unigrams must be sorted in (ascending) alphabetical order. You should eliminate duplicates. The output should be generated using MapReduce. You may store the output in multiple files.

Profile 2
A list of unigrams and their frequencies within the target article. Your software must generate this profile per article. Your list should be grouped by the Document ID (see page 3), and sorted (in descending order) on the frequency of the unigram within the article. This output should be generated using MapReduce. Output may be stored in multiple files.

Profile 3
A list of unigrams and their frequencies within the target corpus. The list of unigrams should be sorted (in descending order) on the frequency of the unigram within the corpus (1G dataset). This output should be generated using MapReduce. Output may be stored in multiple files.

Input Data

• Format

<table>
<thead>
<tr>
<th>Title of Article-1</th>
<th>Document-ID-1</th>
<th>Text of Article-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redirect:424342</td>
<td>1</td>
<td>The Battle of Hormozdgān is fought. Ardeshir I defeats and kills Artabanus V effectively ending the Parthian Empire.357 – Emperor Constantius II enters Rome for the first time to celebrate his victory over Magnus Maximus.1330 – Assassination of Conrad of Montferrat (Conrad II), King of Jerusalem, in Tyre, three days after his title to the throne is confirmed by election. The killing is carried out by Hashshashin.1293 – Nichiren, a Japanese Buddhist monk, propounds Namu Myōhō Renge Kyō for the very first time and declares it to be the essence of Buddhism, in effect founding Nichiren Buddhism.</td>
</tr>
<tr>
<td>USA</td>
<td>353673</td>
<td>The United States of America (USA), commonly known as the United States (U.S.) or America, is a federal republic composed of 50 states, a federal district, five major self-governing territories, and various possessions. At 3.8 million square miles (9.8 million km²) and with over 325 million people, the United States is the world's third- or fourth-largest country by total area and the third-most populous. The capital is Washington, D.C., and the largest city by population is New York City. Forty-eight states and the capital's federal district are contiguous and located in North America between Canada and Mexico. The state of Alaska is in the northwest corner of North America, bordered by Canada to the east and across the Bering Strait from Russia to the west.</td>
</tr>
</tbody>
</table>

Handling input data

• Ignore tense, and gender of words
  • “He” and “She” should be different unigrams
  • “was” and “is” should be different unigrams
• Do not distinct the plural words
  • “name” and “names” should be considered as different unigrams
• Hyphenated words
  • “well-described” should be considered as 1 unigram
• Convert all of the upper cases to the lower case first, if needed
  • Include “a” and “the” in your unigram profiles

Output data format

• Profile 1
  Ngram-A NEWLINE
  Ngram-B NEWLINE
  Ngram-C NEWLINE
• Profile 2
  Document-ID TAB ngram TAB frequency NEWLINE
• Profile 3
  Document-ID TAB ngram TAB frequency NEWLINE

Topics

• MapReduce Design Pattern IV. Join Patterns
Join Patterns

- Data is all over the place
- "Joins" allow users to create a smaller reference set or filter out or select dataset to discover interesting relationships across datasets
- Joining a terabyte of data onto another terabyte dataset could require up to two terabytes of bandwidth!
  - That’s before any actual join logic can be done!

1. Reduce Side Join Pattern
2. Replicated Join Pattern
3. Composite Join Pattern
4. Cartesian Product Pattern

Replicated Join

- Special type of join operation between one large and (many) small data set(s) that can be performed on the map-side
- Mapper
  - Reads all files from the distributed cache during the setup phase
  - Sorting them in to in-memory lookup tables
  - Performs mapper process
  - Joining data
  - If the foreign key is not found in the in-memory structure?
    - The record is either omitted or output (based on the join type)
- No combiner/partitioner/reducer needed

Structure of the replicated join pattern

Hadoop DistributedCache

- Provided by the Hadoop MapReduce Framework
- Caches read only text files, archives, jar files etc.
- Once a file is cached for a job using Distributed cache
  - Data will be available on each data node where map/reduce tasks are running
Using `DistributedCache` for replicated join

- A small file is pushed to all map tasks using `DistributedCache`
- Useful for join between a small set and a large set of data
  - e.g. user information vs. transaction records, user information vs. comment history

Mapper Code

```java
public static class ReplicatedJoinMapper extends Mapper<Object, Text, Text, Text> {
    private static final Text EMPTY_TEXT = new Text;

    public void setup(Context context) throws IOException, InterruptedException {
        Path[] files = context.getConfiguration().getLocalCacheFiles;
        context.getConfiguration();
    }

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        BufferedReader rdr = new BufferedReader(new InputStreamReader(new FileInputStream(new File(p.toString()))));
        String line = null;
        while ((line = rdr.readLine) != null) {
            // Put each record in the user file
            userIdToInfo.get(userId); // Map the user ID to the record
            userInformation; // output the record with an empty value
            context.write(userId, line);
        }
    }
}
```

Size of `DistributedCache` in Hadoop

- Size
  - Default size of the Hadoop distributed cache is 10GB
  - Configurable in `mapred-site.xml`
- Data consistency
  - Hadoop Distributed Cache tracks the modification of timestamps of the cache file
- Overhead
  - Object serialization

Mapper Code

```java
public void setup(Context context) throws IOException, InterruptedException {
    Path[] files = DistributedCache.getLocalCacheFiles;
    context.getConfiguration();
}
```

Mapper Code

```java
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Join (Comment, user, on userID = 'Id');
    userInformation; // output the record with an empty value
    context.write(userId, line);
}
```

Working with `DistributedCache`

- Make sure
  - Your file is available and accessible via http:// or hdfs://
- Setup the application's `JobConf` in your `Driver` class
  ```java
  DistributedCache.addFileToClasspath(new Path("/usr/datafile/XML")
  ```

```java
for (Path p : files) {
    // Read all files in the datafile
    BufferedReader rdr = new BufferedReader(new InputStreamReader(new FileInputStream(new File(p.toString()))));
    String line = null;
    while ((line = rdr.readLine) != null) {
        // If the user information is not null, then output
        userIdToInfo.get(userId); // Map the user ID to the record
        userInformation; // output the record with an empty value
        context.write(userId, line);
    }
}
```
MapReduce Design Patterns IV: Join Patterns

3. Composite Join

- Joins very large datasets together
  - And if the datasets are sorted by foreign key
  - No shuffle and sort needed
- Each input dataset must be partitioned and sorted in a specific way and divided into the same number of partitions

Structure of the composite join pattern

Joining process (Inner Join) - Within a group

Joining process (Inner Join) - Between groups

Composite Join

- Joins very large datasets together
  - And if the datasets are sorted by foreign key
- No shuffle and sort needed
- Each input dataset must be partitioned and sorted in a specific way and divided into the same number of partitions

Structure of the composite join pattern

The number of output part files is equivalent to the number of map tasks.
Joining process (Inner Join)

For A in the left dataset and B in the right dataset:

1. Use approach without sorting and exact matching partitioning: takes O(NM)
2. With the composite join, it takes O(N)

### Example: Composite user comment join

**Preconditions**
- User and comment datasets should be preprocessed by MapReduce
- CompositeInputFormat will be used
- Output should use TextOutputFormat
  - Key: userID
  - Value: userName or comment XML
- Hadoop KeyValueTextOutputFormat can parse these information

**Note:** Since CompositeInputFormat uses Text objects as the key for comparisons
- Align your keys as TEXT objects (rather than LongWritable)
- E.g. "12345" comes before "2"

```java
class CompositeJoinDriver implements Driver {
  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(CompositeJoinDriver.class);
    conf.setJarByClass(CompositeJoinDriver.class);
    conf.setJobName("Join Usage Example");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);
    conf.setInputFormat(CompositeInputFormat.class);
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);
    conf.setNumReduceTasks(0);
    conf.setMapperClass(CompositeMapper.class);
    conf.setReducerClass(CompositeReducer.class);
    conf.setMapred.join.expr("((userID = userXML) AND (cardType = cardTypeXML))");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);
    conf.setMapred.join.outputClass(TextOutputFormat.class);
    conf.setMapred.join.outputDir(outputDir);
    RunningJob runningJob = JobClient.runJob(conf);
    conf.setMapred.join.outputPath(outputDir);
    JobClient.runJob(conf);
    System.exit(0);
  }
}
```

```java
class CompositeMapper extends MapReduceBase {
  public static void map(Text key, Text value, OutputCollector output, Reporter reporter) throws IOException {
    // Get the first two elements in the tuple and output them
    output.collect((Text)value.get(0),
                    (Text)value.get(1));
  }
}
```

**Driver Code**

```java
public static void main(String[] args) throws Exception {
  Path userPath = new Path(args[0]);
  Path commentPath = new Path(args[1]);
  Path outputDir = new Path(args[2]);
  String joinType = args[3];
  JobConf conf = new JobConf("CompositeJoinDriver");
  conf.setJarByClass(CompositeJoinDriver.class);
  conf.setMapperClass(CompositeMapper.class);
  conf.setReducerClass(CompositeReducer.class);
  conf.setNumReduceTasks(0);
  conf.setJoinType(joinType);
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(Text.class);
  conf.setMapred.join.expr("((userID = userXML) AND (cardType = cardTypeXML))");
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(Text.class);
  conf.setMapred.join.outputClass(TextOutputFormat.class);
  conf.setMapred.join.outputDir(outputDir);
  conf.setInputFormat(CompositeInputFormat.class);
  conf.setMapred.join.outputPath(outputDir);
  RunningJob runningJob = JobClient.runJob(conf);
  System.exit(runningJob.isSuccessful() ? 0 : 1);
}
```

**Mapper Code**

```java
public static class CompositeMapper extends MapReduceBase {
  public void map(Text key, TupleInputWritable value, OutputCollector output, Reporter reporter) throws IOException {
    // Get the first two elements in the tuple and output them
    Text userID = (Text)value.get(0);
    Text cardType = (Text)value.get(1);
    output.collect(userID, cardType);
  }
}
```

### MapReduce Design Patterns IV: Join Patterns

4. Cartesian Product
Cartesian Product

- Pairs every record from multiple inputs
- Applicability
  - You want to analyze relationships between all pairs of records
  - e.g. Correlation analysis

Structure of the Cartesian Product

<table>
<thead>
<tr>
<th>Input</th>
<th>Split</th>
<th>Dataset A</th>
<th>Input</th>
<th>Split</th>
<th>Dataset A</th>
<th>Input</th>
<th>Split</th>
<th>Dataset A</th>
<th>Input</th>
<th>Split</th>
<th>Dataset B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

Performance Analysis

- Cartesian product requires a massive data space
- To compute a Cartesian product between N records and M records
  - N x M data points are created
  - Requires many Map slots and a very long time

Example: Cartesian Product

- Comments comparison
  - Inspects a pair of comments and determines how similar they are
  - If they are similar enough, the pair is stored in the output file
- During the setup phase
  - `getInputSplits`
    - Creates the cross-product of the input splits and sets into a list of `CompositeInputSplits`

HDFS Block Size vs. Hadoop Input Split

- Block Size
  - Physical Location where the data been stored (default size of the HDFS block is 128MB)
  - All blocks of the file are of the same size except the last block
  - HDFS distributes these blocks (chunks) across the cluster
  - Achieves more parallelism and fault tolerance (with replication)

- Input Split
  - Logical representation of block
  - More or less than a Block Size
  - Used during data processing in MapReduce program or other processing techniques
  - InputSplit does not contain actual data but a reference to the data
  - Split acts as a broker between the block and mapper
    - During MapReduce execution, Hadoop scans through the blocks and create InputSplits and assign them to individual mapper

Example of Using Input Split

- 1.28GB file
  - Divided into 10 blocks (128MB)
  - `InputFormat.getSplits()` is responsible for generating the input splits
  - `Default` creates one input split for each HDFS block

- Case 1. If input split is not specified
  - HDFS block size will be the split size
  - 10 mappers are initialized to load the file
  - Each mapper loads one block

- Case 2. If the start and end positions of the records are not in the same block?
  - InputSplit provides the Start and End positions (offsets)
  - Mapper is going to load the block of data according to start and end offset values
Example of Using Input Split

- 1.28GB file
- Divided into 10 blocks (128MB)
- InputFormat.getSplits() is responsible for generating the input splits
- Default: creates one input split for each HDFS block.

Case 3. If the specified split size was not correct
- The whole file will form one input split and processed by one map
- Requires processing very large file

Case 4. If your resource is limited and you want to limit the number of maps?
- Specify the Split size as 256MB
- Only 5 maps will be executed

From Input Splits to Mapper tasks

- Hadoop's Interface InputFormat<K, V>
- RecordReader<K, V>
- InputSplit[]

- Validate the input specification of the job
- Split-up the input file(s) into logical InputSplits
- Assign to an individual Mapper
- Provide the RecordReader implementation to the user to glean input records from the logical InputSplit for processing Mapper

RecordReader<K, V>

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>close()</td>
</tr>
<tr>
<td>K</td>
<td>createKey()</td>
</tr>
<tr>
<td>V</td>
<td>createValue()</td>
</tr>
<tr>
<td>long</td>
<td>getPos()</td>
</tr>
<tr>
<td>float</td>
<td>getProgress()</td>
</tr>
<tr>
<td>boolean</td>
<td>next(K key, V value)</td>
</tr>
</tbody>
</table>

InputSplit

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>getLength()</td>
</tr>
<tr>
<td>String[]</td>
<td>getLocations()</td>
</tr>
</tbody>
</table>

Input Format Code

```java
public static class CartesianInputFormat extends FileInputFormat {
    public static final String LEFT_INPUT_FORMAT = "cart.left.inputformat";
    public static final String LEFT_INPUT_PATH = "cart.left.path";
    public static final String RIGHT_INPUT_FORMAT = "cart.right.inputformat";
    public static final String RIGHT_INPUT_PATH = "cart.right.path";

    public static void setLeftInputInfo(JobConf job, Class<? extends FileInputFormat> inputFormat, String inputPath) {
        job.set(LEFT_INPUT_FORMAT, inputFormat.getCanonicalName());
        job.set(LEFT_INPUT_PATH, inputPath);
    }

    public static void setRightInputInfo(JobConf job, Class<? extends FileInputFormat> inputFormat, String inputPath) {
        job.set(RIGHT_INPUT_FORMAT, inputFormat.getCanonicalName());
        job.set(RIGHT_INPUT_PATH, inputPath);
    }

    public InputSplit[] getSplits(JobConf conf, int numSplits) throws IOException {
        // Get the input splits from both the left and right datasets
        InputSplit[] leftSplits = getInputSplits(conf, conf.get(LEFT_INPUT_FORMAT), conf.get(LEFT_INPUT_PATH), numSplits);
        InputSplit[] rightSplits = getInputSplits(conf, conf.get(RIGHT_INPUT_FORMAT), conf.get(RIGHT_INPUT_PATH), numSplits);
        // Create our CompositeInputSplits, size equal to
        // left.length * right.length
        CompositeInputSplit[] returnSplits = new CompositeInputSplit[leftSplits.length * rightSplits.length];
        return returnSplits;
    }
}
```
```java
private InputSplit[] getLeftSplits(JobConf conf, String inputFormatClass, String inputPath, int numSplits) throws ClassNotFoundException, IOException {
    // Create a new instance of the input format
    FileInputFormat inputFormat = (FileInputFormat) ReflectionUtils.newInstance(Class.forName(inputFormatClass), conf);
    // Set the input path for the left data set
    inputFormat.setInputPaths(inputPath);
    // Get the left input splits
    return inputFormat.getSplits(conf, inputPath);
}
```

```java
public CartesianInputSplit[] getLeftSplits(JobConf conf, String inputFormatClass, String inputPath, int numSplits) throws ClassNotFoundException, IOException {
    // Create a new instance of the input format
    FileInputFormat inputFormat = (FileInputFormat) ReflectionUtils.newInstance(Class.forName(inputFormatClass), conf);
    // Set the input path for the left data set
    inputFormat.setInputPaths(inputPath);
    // Get the left input splits
    return inputFormat.getSplits(conf, inputPath);
}
```

```java
public CartesianInputSplit[] getRightSplits(JobConf conf, String inputFormatClass, String inputPath, int numSplits) throws IOException {
    // Create a new instance of the right record reader
    this.rightRR = null;
    // Create right record reader
    FileInputFormat rightFIF = (FileInputFormat) ReflectionUtils.newInstance(Class.forName(inputFormatClass), conf);
    rightFIF.getRecordReader(rightConf, rightReporter);
    // Create key value pairs for parsing
    leftRR.createKey(leftFIF.getRecordReader(leftConf, leftReporter));
    rightRR.createKey(rightFIF.getRecordReader(rightConf, rightReporter));
    // Create key value pairs for parsing
    leftRR.createValue(leftFIF.getRecordReader(leftConf, leftReporter));
    rightRR.createValue(rightFIF.getRecordReader(rightConf, rightReporter));
    // Create a new composite input split composing of the two
    return new CompositeInputSplit(leftSplits, rightSplits);
}
```

```java
public CartesianRecordReader(CartesianInputSplit split, JobConf conf, Reporter reporter) throws IOException {
    // Create right record reader
    rightRR = (CartesianInputFormat.ReflectionUtils.newInstance(Class.forName(split.rightInputFormat), conf);
    rightFIF = rightRR.getRecordReader(split.rightConf, split.rightReporter);
    // Create key value pairs for parsing
    key = (K1)split.leftReader.createKey();
    value = (V1)split.leftReader.createValue();
    key = (K2)split.rightReader.createKey();
    value = (V2)split.rightReader.createValue();
}
```
public static class CartesianMapper extends MapReduceBase implements Mapper < Text, Text, Text, Text > {
    private Text outkey = new Text();
    private Text outvalue = new Text();
    public void map(Text key, Text value, OutputCollector < Text, Text > output, Reporter reporter) throws IOException, InterruptedException {
        if (sameWordCount > 2) {
            output.collect(outkey, outvalue);
        } else {
            // Otherwise, this right data set is complete
            // and we should go to the next left pair
            goToNextLeft = true;
        }
    }
}

public static void main(String [] args) throws IOException, InterruptedException {
    JobConf conf = new JobConf(CartesianProduct.class);
    conf.setJarByClass(CartesianProduct.class);
    conf.setInputFormat(CartesianInputFormat.class);
    conf.setOutputFormat(CartesianOutputFormat.class);
    conf.setNumReduceTasks(0);
    CartesianInputFormat.setLeftInputInfo(conf, new Path(args[0]));
    CartesianOutputFormat.setOutputPath(conf, new Path(args[1]));
    CartesianInputFormat.setRightInputInfo(conf, new Path(args[2]));
    CartesianOutputFormat.setOutputPath(conf, new Path(args[3]));
    JobClient.runJob(conf);
    System.exit(0);
}

public boolean next(Text key, Text value) throws IOException, InterruptedException {
    // If we are to go to the next left key/value pair
    if (goToNextLeft) {
        // Read the next key value pair
        while (!leftRR.alldone) {
            // If no more, then this task is nearly finished
            goToNextLeft = !leftRR.goToNextLeft()
        }
        return false;
    } else {
        // Otherwise, this right data set is complete
        // and we should go to the next left pair
        goToNextLeft = true;
    }
    // This loop will continue if we finished reading key/value
    // pairs from the right data set
    while (goToNextLeft)
    // Return true if a key/value pair was read, false otherwise
    return !alldone;
}

public static void main(String [] args) throws IOException, InterruptedException {
    JobConf conf = new JobConf(CartesianProduct.class);
    conf.setJarByClass(CartesianProduct.class);
    conf.setInputFormat(CartesianInputFormat.class);
    conf.setOutputFormat(CartesianOutputFormat.class);
    conf.setNumReduceTasks(0);
    conf.setOutputValueClass(Text.class);
    conf.setOutputKeyClass(Text.class);
    conf.setJobName("Cartesian Product");
    RunningJob job = JobClient.runJob(conf);
    while (!job.isSuccessful()) {
        Thread.sleep(1000);
    }
    System.exit(job.isSuccessful() ? 0 : 1);
}

for (String s : leftSet) {
    if (rightSet.contains(s)) {
        sameWordCount += 1;
        StringBuilder words = new StringBuilder();
        for (String w : rightSet) {
            if (w.equals(s)) {
                words.append(w + ",");
                ++sameWordCount;
            }
        }
    } else {
        System.out.println(s + " not in right set");
        ++sameWordCount;
    }
    if (sameWordCount > 2) {
        outputKey.setValue("\"" + key);
        output.collect(outputKey, value);
    }
}

public boolean next(Text key, Text value) throws IOException, InterruptedException {
    // If we aren't done, set the value to the key and set
    // our flags
    key.set(value.toString());
    goToNextLeft = !leftRR.alldone;
    // Reset the right record reader
    if (!leftRR.alldone) {
        // Read the next key value pair, false means no more pairs
        while (!leftRR.alldone)
        // If we aren't done, set the value to the key and set
        // our flags
        key.set(value.toString());
    } else {
        // Otherwise, this right data set is complete
        // and we should go to the next left pair
        goToNextLeft = true;
    }
    // This loop will continue if we finished reading key/value
    // pairs from the right data set
    while (goToNextLeft)
    // Return true if a key/value pair was read, false otherwise
    return !alldone;
}

public static class CartesianProduct implements Mapper < Text, Text, Text, Text > {
    private Text outkey = new Text();
    private Text outvalue = new Text();
    public void map(Text key, Text value, OutputCollector < Text, Text > output, Reporter reporter) throws IOException, InterruptedException {
        if (sameWordCount > 2) {
            output.collect(outkey, outvalue);
        } else {
            // Otherwise, this right data set is complete
            // and we should go to the next left pair
            goToNextLeft = true;
        }
    }
}

for (String s : leftSet) {
    if (rightSet.contains(s)) {
        sameWordCount += 1;
        StringBuilder words = new StringBuilder();
        for (String w : rightSet) {
            if (w.equals(s)) {
                words.append(w + ",");
                ++sameWordCount;
            }
        }
    } else {
        System.out.println(s + " not in right set");
        ++sameWordCount;
    }
    if (sameWordCount > 2) {
        outputKey.setValue("\"" + key);
        output.collect(outputKey, value);
    }
}
Questions?