Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Отчёт по практической работе № 2

По дисциплине: «Распределенная обработка информации»

Тема: «MapReduce: BiGram»

Выполнил:
студент гр. МГ-101
Лукошкин В.Ю.

Проверил:
Профессор Кафедры ВС
Курносов М.Г.

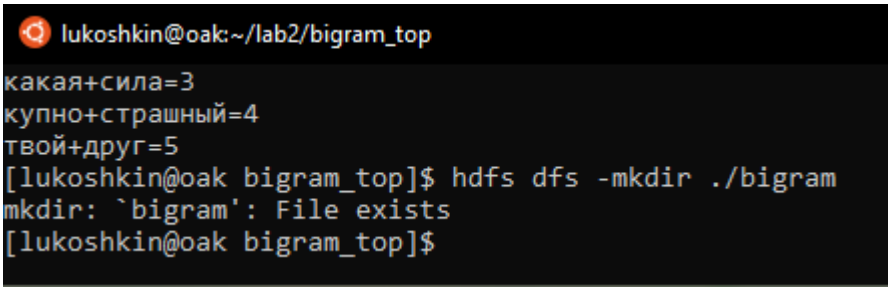
Новосибирск 2021

Задание.

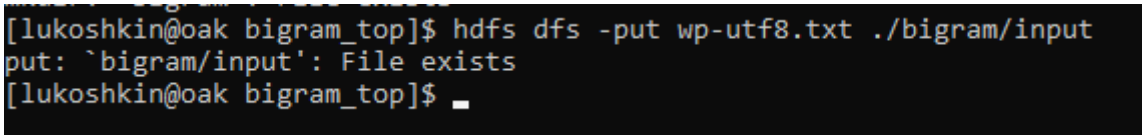
С помощью Hadoop MapReduce подсчитать количество последовательностей из двух соседних элементов (биграмм) в заданном входном наборе.

Ход работы.

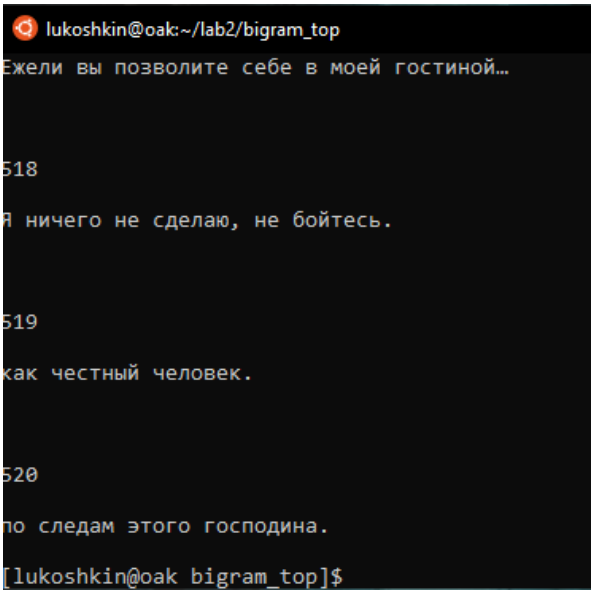
Создаем каталог в HDFS:



Для подсчета биграмм будет использоваться файл *wp-utf8.txt* из директории /home/pub/hadoop/. Копируем файл в HDFS:

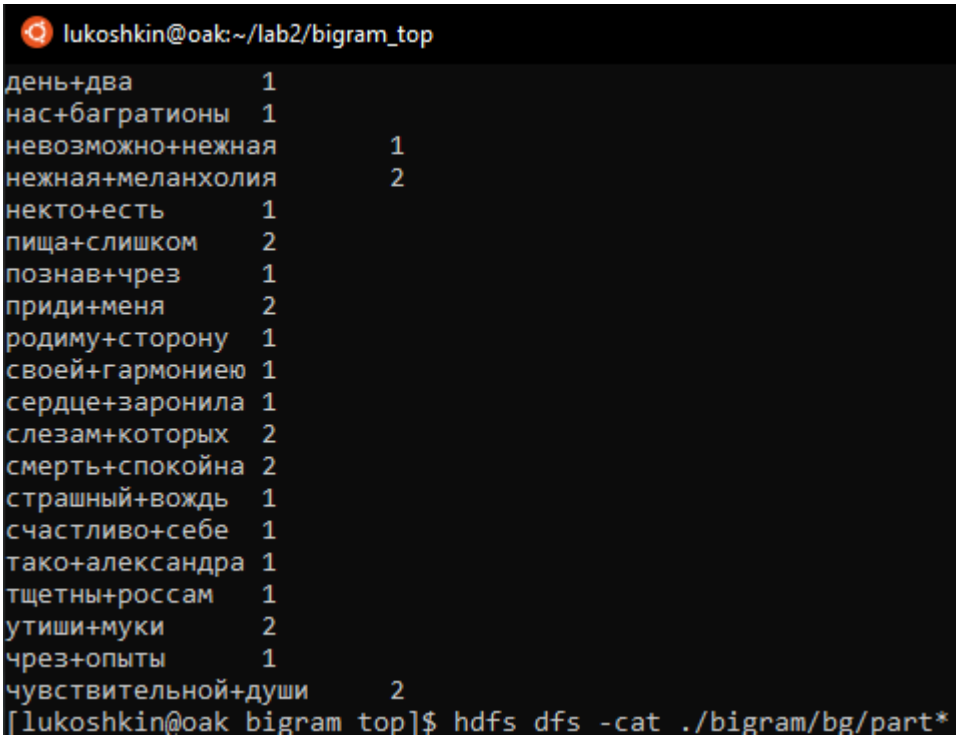


Проверим содержимое файла:



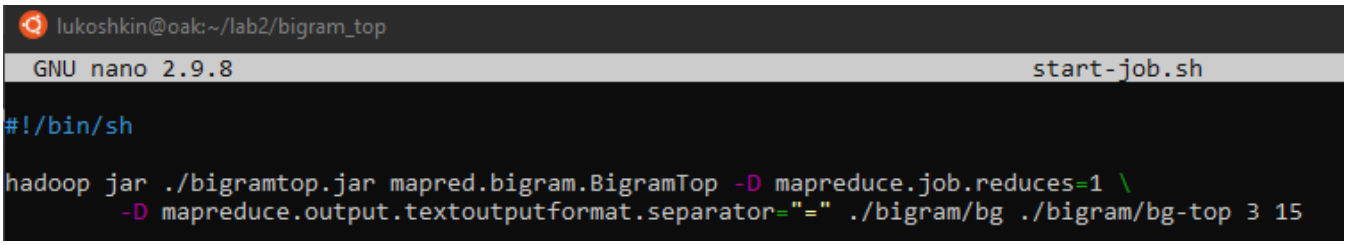
Копируем пример из /home/pub/hadoop/lab2/bigram_count, в домашнюю директорию, компилируем BigramCount.java через команду ./build.sh и запускаем задание ./start-job.sh.

После выполнения будет создан каталог ./bigram/bg в HDFS. Проверим содержимое файлов командой hdfs dfs -cat ./bigram/bg/part*:



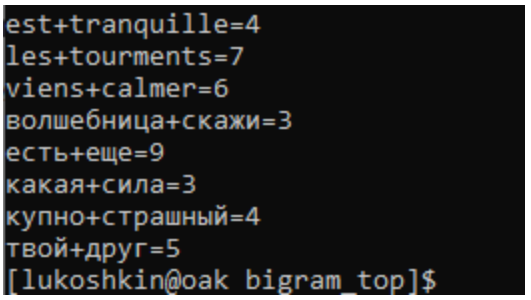
В результате получаем биграммы и их количество в тексте. Так как максимальное количество биграмм не превышало двух, был изменен выходной файл ./bigram/bg/part*: для вычисления TOP-биграмм.

Затем компилируем BigramTop.java и запускаем задание ./start-job.sh .



После выполнения будет создан каталог ./bigram/bg-top, где будет выведено <=15 биграмм с числом повторений биграмм >= 3.

Проверим получившиеся результаты:



В итоге получилось 8 биграмм с числом повторений >=3.

Исходный код программы

BigramCount.java

```
package mapred.bigram;

import java.io.BufferedReader;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.StringTokenizer;

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.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.io.WritableUtils;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.StringUtils;
```

```
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class BigramCount extends Configured implements Tool {

    /*
     * Custom data type for bigram
     */
    public static class Bigram implements WritableComparable<Bigram> {

        private String first;
        private String second;

        public void set(String first, String second) {
            this.first = first;
            this.second = second;
        }

        public String getFirst() {
            return first;
        }

        public String getSecond() {
            return second;
        }

        @Override
        public void readFields(DataInput in) throws IOException {
            first = Text.readString(in);
            second = Text.readString(in);
        }

        @Override
        public void write(DataOutput out) throws IOException {
```

```
            Text.writeString(out, first);
            Text.writeString(out, second);
        }

        @Override
        public int compareTo(Bigram o) {
            if (!first.equals(o.first)) {
                return first.compareTo(o.first);
            } else {
                return second.compareTo(o.second);
            }
        }

        @Override
        public String toString() {
            return first + "+" + second;
        }

        @Override
        public int hashCode() {
            return toString().hashCode();
        }

    }

    /*
     * Custom key comparator
     */
    public static class BigramComparator extends WritableComparator {

        private static final Text.Comparator TEXT_COMPARATOR = new
Text.Comparator();

        public BigramComparator() {
```

```
            super(Bigram.class);
        }

        /*
         * RawComparator implementation for speed
         * (compare binary representations without deserializing keys)
         */
        @Override
        public int compare(byte[] b1, int s1, int l1, byte[] b2, int
s2, int l2) {
            try {
                int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) +
readVInt(b1, s1);
                int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) +
readVInt(b2, s2);
                int cmp1 = TEXT_COMPARATOR.compare(b1, s1, firstL1, b2,
s2, firstL2);
                if (cmp1 != 0) {
                    return cmp1;
                } else {
                    int secondL1 =
WritableUtils.decodeVIntSize(b1[s1+firstL1]) + readVInt(b1, s1+firstL1);
                    int secondL2 =
WritableUtils.decodeVIntSize(b2[s2+firstL2]) + readVInt(b2, s2+firstL2);
                    return TEXT_COMPARATOR.compare(b1, s1+firstL1,
secondL1, b2, s2+firstL2, secondL2);
                }
            } catch (IOException e) {
                throw new IllegalArgumentException(e);
            }
        }

        @Override
        public int compare(WritableComparable a, WritableComparable b)
```

```
{
                if (a instanceof Bigram && b instanceof Bigram) {
                    return ((Bigram)a).compareTo((Bigram)b);
                }
                return super.compare(a, b);
            }
        }



        /*
         * MAPPER
         */
        public static class CountMapper extends Mapper<Text, Text, Bigram,
IntWritable> {

            static enum Counters {
                INPUT_WORDS
            }

            private List<String> skipList = new ArrayList<String>();
            private long numRecords = 0;
            private Map<String, Map<String, Integer>> results = new
HashMap<String, Map<String, Integer>>();
            private final Bigram bg = new Bigram();
            private final IntWritable bgCount = new IntWritable();

            @Override
            protected void setup(Context context) {
                String skipListFile =
context.getConfiguration().get("wordcount.skip-list");
                if (skipListFile != null) {
                    loadSkipListFile(skipListFile);
                }
```

```
            }

            public void map(Text key, Text value, Context context) throws
IOException, InterruptedException {
                String text = value.toString().toLowerCase();

                for (String pattern : skipList) {
                    text = text.replaceAll(pattern, " ");
                }

                StringTokenizer tokenizer = new StringTokenizer(text);
                if (tokenizer.hasMoreElements()) {
                    String prev = tokenizer.nextToken();
                    String curr;
                    while (tokenizer.hasMoreTokens()) {
                        curr = tokenizer.nextToken();
                        if (prev.length() > 2 && curr.length() > 2) {
                            addResult(prev, curr);
                        }
                        prev = curr;

context.getCounter(Counters.INPUT_WORDS).increment(1);
                    }
                }

                if ((++numRecords % 1000) == 0) {
                    context.setStatus("Finished processing " + numRecords +
" records");
                    emitResults(context);
                }
            }

            private void loadSkipListFile(String skipListFile) {
                    BufferedReader fis = null;
```

```
                try {
                    fis = new BufferedReader(new FileReader(skipListFile));
                    String pattern = null;
                    while ((pattern = fis.readLine()) != null) {
                        skipList.add(pattern);
                    }
                } catch (IOException ioe) {
                    System.err.println("Caught exception while loading skip
file '" + skipListFile + "' : "
                            + StringUtils.stringifyException(ioe));
                } finally {
                    if (fis != null) {
                            try {
                                                fis.close();
                                        } catch (IOException ioe) {

System.err.println("Caught exception while closing skip file '" +
skipListFile + "' : "
                                        +
StringUtils.stringifyException(ioe));
                                        }
                    }
                }
            }

            private void addResult(String w1, String w2) {
                Map<String, Integer> counts = results.get(w1);
                if (counts == null) {
                    counts = new HashMap<String, Integer>();
                    results.put(w1, counts);
                }
                Integer count = counts.get(w2);
                if (count == null) {
                    counts.put(w2, 1);
```

```
                } else {
                    counts.put(w2, ++count);
                }
            }

            private void emitResults(Context context) throws IOException,
InterruptedException {
                for (Entry<String, Map<String, Integer>> counts :
results.entrySet()) {
                    String w1 = counts.getKey();
                    for (Entry<String, Integer> count :
counts.getValue().entrySet()) {
                        bg.set(w1, count.getKey());
                        bgCount.set(count.getValue());
                        context.write(bg, bgCount);
                    }
                }
                results.clear();
            }

            @Override
            protected void cleanup(Context context) throws IOException,
InterruptedException {
                emitResults(context);
            }
        }

        /*
         * REDUCER
         */
        public static class CountReducer extends Reducer<Bigram,
IntWritable, Bigram, IntWritable> {
            private IntWritable result = new IntWritable();
```

```
            public void reduce(Bigram key, Iterable<IntWritable> values,
Context context) throws IOException,
                    InterruptedException {
                int sum = 0;
                for (IntWritable val : values) {
                    sum += val.get();
                }
                result.set(sum);
                context.write(key, result);
            }
        }

        /*
         * APPLICATION
         */
        public int run(String[] args) throws Exception {
            Configuration conf = getConf();
            if (args.length != 3) {
                System.err.println("Usage: BigramCount <input_path>
<output_path> <skip_list_path>");
                System.exit(2);
            }

            File skipFile = new File(args[2]);
            conf.set("wordcount.skip-list", skipFile.getName());
            conf.set("tmpfiles", "file://" + skipFile.getAbsolutePath());

            Job job = new Job(conf);
            job.setJarByClass(BigramCount.class);
            job.setJobName("bigram count");

            job.setMapperClass(CountMapper.class);
            job.setMapOutputKeyClass(Bigram.class);
            job.setMapOutputValueClass(IntWritable.class);
```

```
        job.setCombinerClass(CountReducer.class);
        job.setSortComparatorClass(BigramComparator.class);

        job.setReducerClass(CountReducer.class);
        job.setOutputKeyClass(Bigram.class);
        job.setOutputValueClass(IntWritable.class);

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

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

        boolean success = job.waitForCompletion(true);
        return success ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int ret = ToolRunner.run(new BigramCount(), args);
        System.exit(ret);
    }
}
```

BigramTop.java

```
package mapred.bigram;

import java.io.BufferedReader;
import java.io.DataInput;
import java.io.DataOutput;
```

```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.StringTokenizer;

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.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.io.WritableUtils;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.StringUtils;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class BigramTop extends Configured implements Tool {

    /*
     * MAPPER
     */
```

```
        public static class TopMapper extends Mapper<Text, Text, Text,
IntWritable> {

            static Integer bg_repeat;
            private Map<Text, IntWritable> results = new HashMap<Text,
IntWritable>();

            @Override
            protected void setup(Context context) {
                bg_repeat =
Integer.parseInt(context.getConfiguration().get("bigramtop.bg-repeat"));
            }

            public void map(Text key, Text value, Context context) throws
IOException, InterruptedException {

                Integer val = Integer.parseInt(value.toString());
                if(val >= bg_repeat) {
                    context.write(key, new IntWritable(val));
                }
            }
        }

        /*
         * REDUCER
         */
        public static class TopReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {
            static Integer lines_count;
            private IntWritable result = new IntWritable();

            @Override
            protected void setup(Context context) {
                lines_count =
```

```
Integer.parseInt(context.getConfiguration().get("bigramtop.lines-count"));
            }

            public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException,
                    InterruptedException {

                if (lines_count-- > 0) {
                    int sum = 0;
                    for (IntWritable val : values) {
                        sum += val.get();
                    }

                    result.set(sum);
                    context.write(key, result);
                }
            }
        }

        /*
         * APPLICATION
         */
        public int run(String[] args) throws Exception {
            Configuration conf = getConf();
            if(args.length != 4) {
                System.err.println("Usage: BigramTop <input_path>
<output_path> <bg_repeat> <lines_count>");
                System.exit(2);
            }

            conf.set("bigramtop.bg-repeat", args[2]);
            conf.set("bigramtop.lines-count", args[3]);

            Job job = new Job(conf);
```

```
        job.setJarByClass(BigramTop.class);
        job.setJobName("bigram topn");

        job.setMapperClass(TopMapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setCombinerClass(TopReducer.class);

        job.setReducerClass(TopReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

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

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

        boolean success = job.waitForCompletion(true);
        return success ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int ret = ToolRunner.run(new BigramTop(), args);
        System.exit(ret);
    }
}
```