МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

**Инженерно-технологическая академия**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра Математического обеспечения и применения ЭВМ**

**ОТЧЁТ**

по лабораторной работе 2

по курсу «РиОВПП»

Выполнили:

ст. группы КТмо1-3

Шепель И. О.

Куприянова А. А.

Проверил:

преподаватель МОП ЭВМ

Пирская Л. В.

Оценка

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2017 г.

Таганрог 2017

**Цель работы:**

С помощью библиотеки maven для сборки проектов под Apache Storm создать топологию для подсчета количества слов.

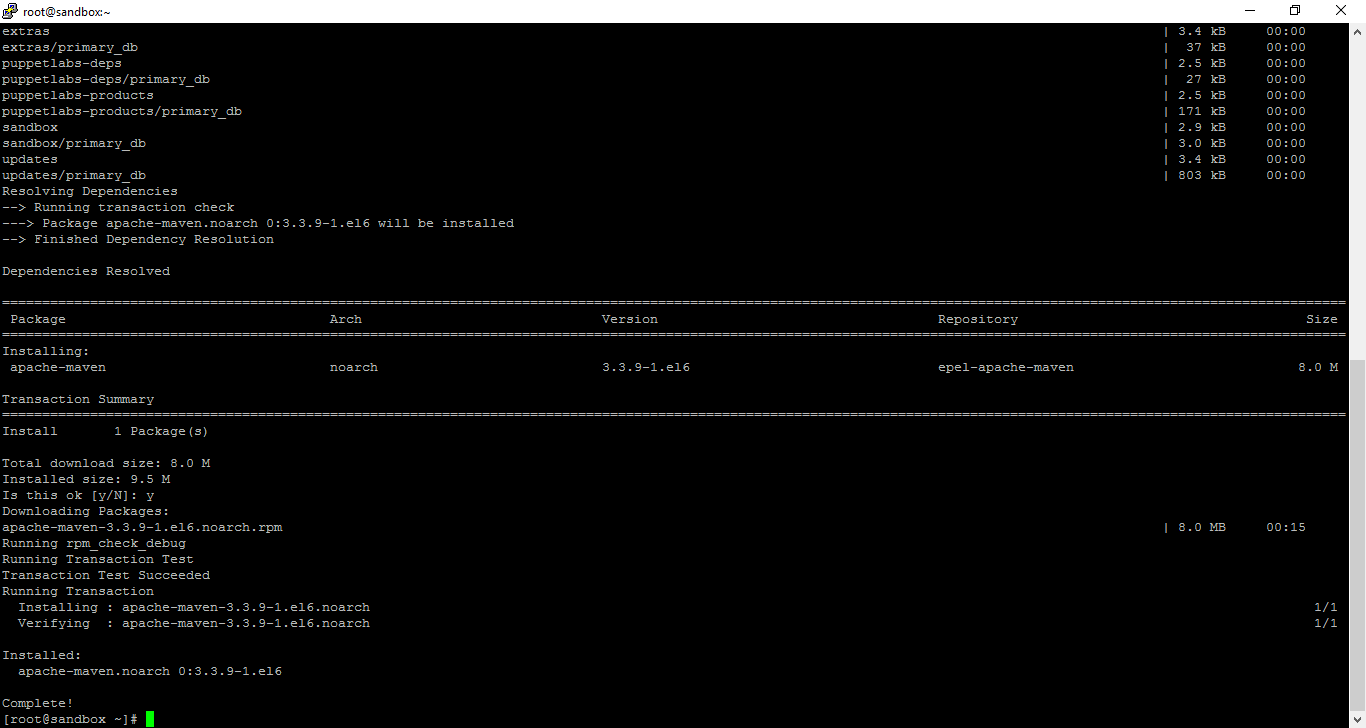
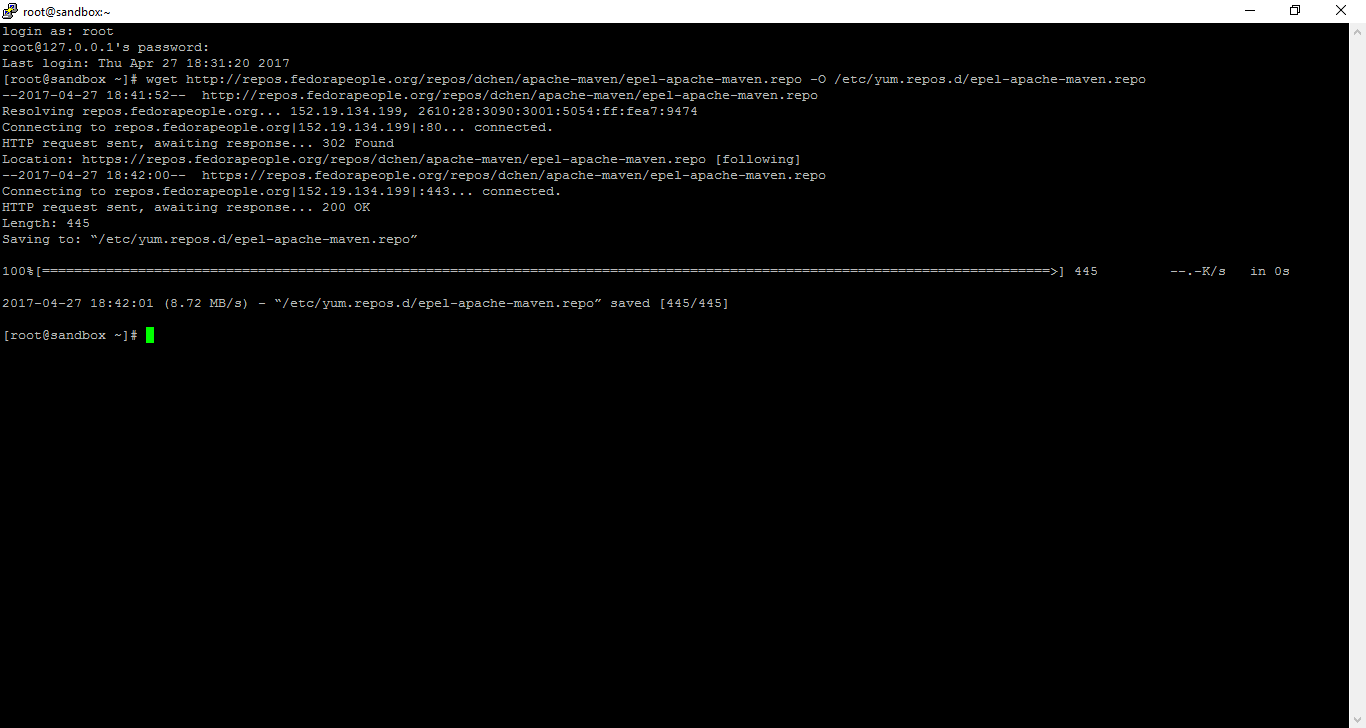
А. Предложения генерируются случайно с помощью класса RandomSentenceSpout;

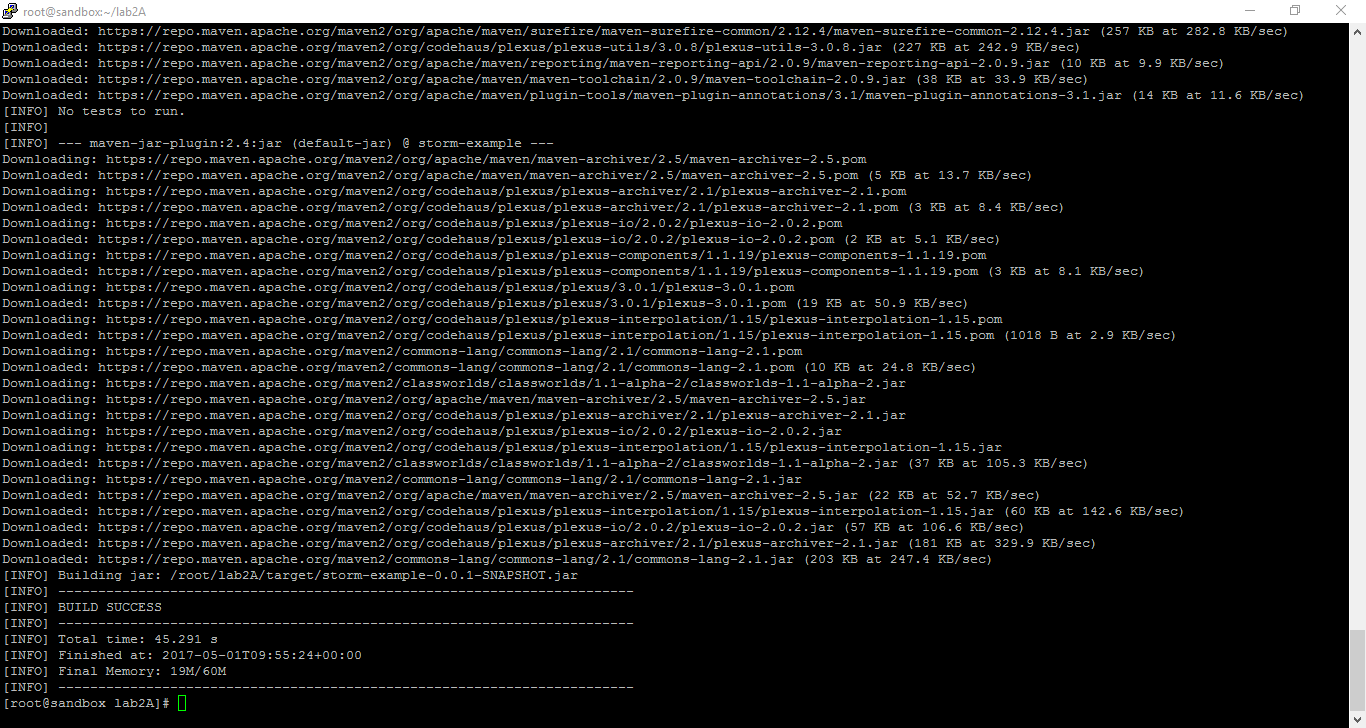
B. Данные подаются с файлом data.txt.

C. Добавить обработчик удаляющий стоп слова, и переводящий в нижний регистр.

D. Добавить вывод статистики по количеству слов через заданный интервал для топ N слов.

**Выполнение:**

Сперва загрузим и установим maven.

Сборка осуществляется с помощью команды mvn clean package, ее результат выглядит следующим образом:

A) Приведем код файла TopWordFinderTopologyPartA.java:

public class TopWordFinderTopologyPartA {

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

TopologyBuilder builder = new TopologyBuilder();

Config config = new Config();

config.setDebug(true);

config.setMaxTaskParallelism(3);

builder.setSpout("spout", new RandomSentenceSpout(), 5);

builder.setBolt("split", new SplitSentenceBolt(), 5).shuffleGrouping("spout");

builder.setBolt("count", new WordCountBolt(), 5).fieldsGrouping("split", new Fields("word"));

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("word-count", config, builder.createTopology());

//wait for 60 seconds and then kill the topology

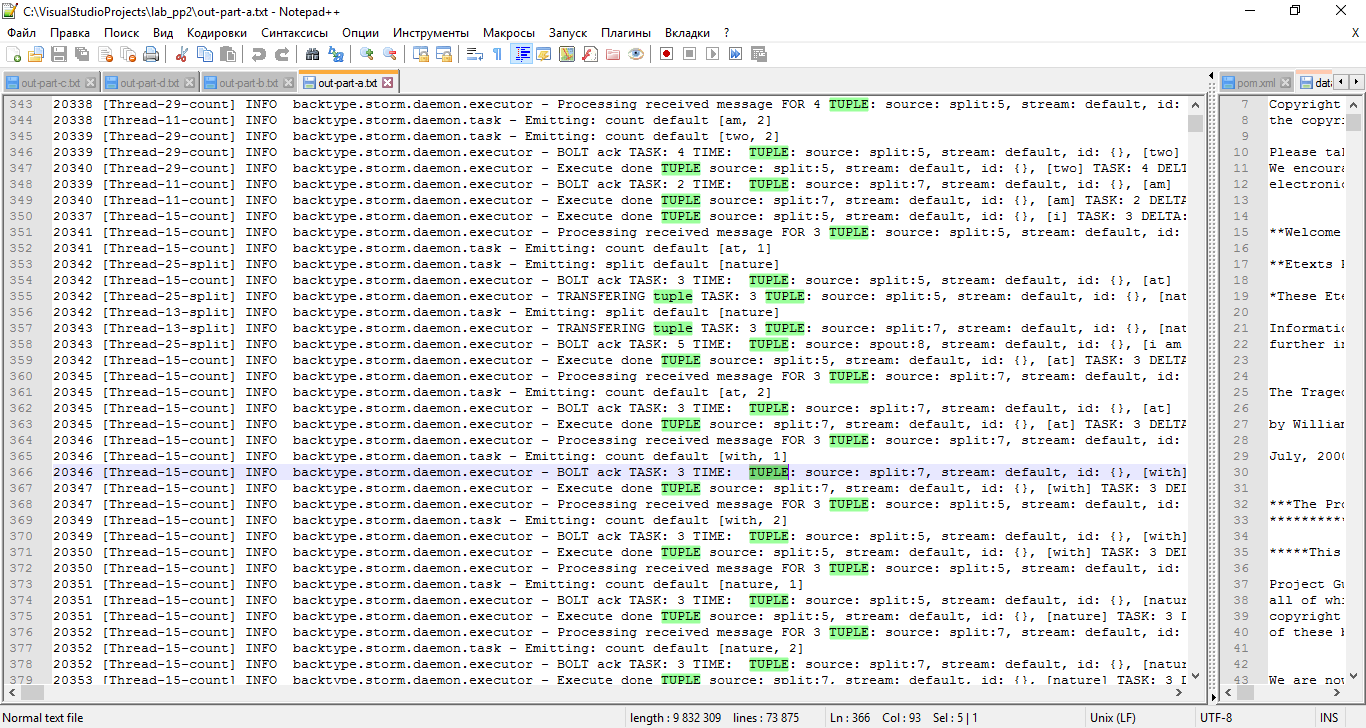
Thread.sleep(60 \* 1000);

cluster.shutdown();

}

}

В методе main мы формируем нашу топологию. Сначала указывается spout – источник слов для топологии, то, на чем она будет строится. Затем bolt – элемент топологии, в данном случае для разделения и подсчета слов.

Результат:

В файл выводятся строки, последовательно указывающие на порядок операций.

B) Приведем код файла TopWordFinderTopologyPartB.java:

public class TopWordFinderTopologyPartB {

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

TopologyBuilder builder = new TopologyBuilder();

Config config = new Config();

config.setDebug(true);

config.setMaxTaskParallelism(3);

config.put("in", args[0]);

config.put(Config.TOPOLOGY\_MAX\_SPOUT\_PENDING, 1);

builder.setSpout("spout", new FileReaderSpout(), 1);

builder.setBolt("split", new SplitSentenceBolt(), 5).shuffleGrouping("spout");

builder.setBolt("count", new WordCountBolt(), 5).fieldsGrouping("split", new Fields("word"));

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("word-count", config, builder.createTopology());

//wait for 2 minutes and then kill the job

Thread.sleep( 2 \* 60 \* 1000);

cluster.shutdown();

}

}

И код файла FileReaderSpout.java:

public class FileReaderSpout implements IRichSpout {

private SpoutOutputCollector \_collector;

private TopologyContext context;

private FileReader reader;

private boolean end = false;

@Override

public void open(Map conf, TopologyContext context,

SpoutOutputCollector collector) {

String file = conf.get("in").toString();

try

{

reader = new FileReader(file);

}

catch (IOException e)

{

}

this.context = context;

this.\_collector = collector;

}

@Override

public void nextTuple() {

if (end)

{

try

{

Thread.sleep(1000);

}

catch (InterruptedException e)

{

}

}

BufferedReader buffer = new BufferedReader(reader);

String line;

try

{

while ((line = buffer.readLine()) != null)

{

line = line.trim();

if (line.length() > 0)

{

\_collector.emit(new Values(line));

}

}

}

catch (IOException e)

{

}

finally

{

end = true;

}

}

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("word"));

}

@Override

public void close() {

try

{

reader.close();

}

catch (IOException e)

{

}

}

@Override

public void activate() {

}

@Override

public void deactivate() {

}

@Override

public void ack(Object msgId) {

}

@Override

public void fail(Object msgId) {

}

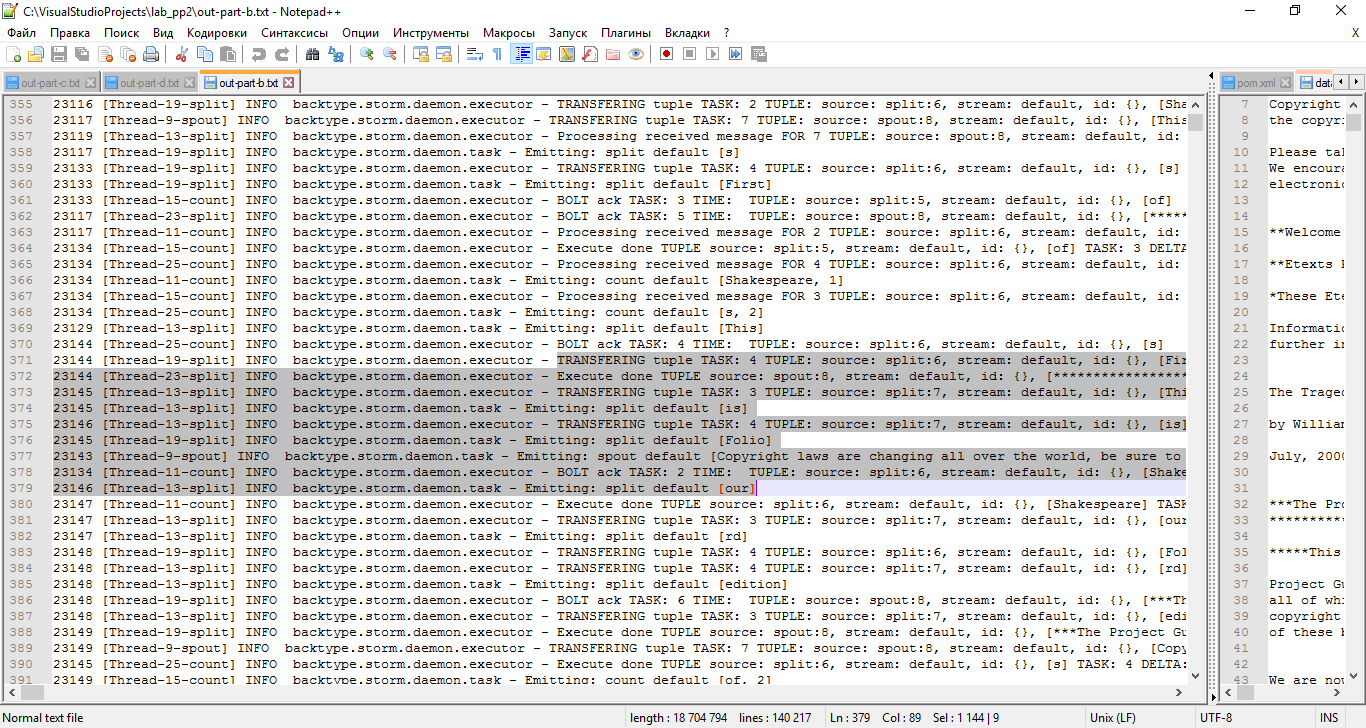
@Override

public Map<String, Object> getComponentConfiguration() {

return null;

}

}

Результат:

C) Приведем код файла TopWordFinderTopologyPartC.java:

public class TopWordFinderTopologyPartC {

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

TopologyBuilder builder = new TopologyBuilder();

Config config = new Config();

config.setDebug(true);

config.setMaxTaskParallelism(3);

config.put("in", args[0]);

config.put(Config.TOPOLOGY\_MAX\_SPOUT\_PENDING, 1);

builder.setSpout("spout", new FileReaderSpout(), 1);

builder.setBolt("split", new SplitSentenceBolt(), 5).shuffleGrouping("spout");

builder.setBolt("normalize", new NormalizerBolt(), 5).shuffleGrouping("split");

builder.setBolt("count", new WordCountBolt(), 5).fieldsGrouping("normalize", new Fields("word"));

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("word-count", config, builder.createTopology());

//wait for 2 minutes then kill the job

Thread.sleep(2 \* 60 \* 1000);

cluster.shutdown();

}

}

И код класса NormalizerBolt.java, отвечающий за обработку входящих последовательностей:

public class NormalizerBolt extends BaseBasicBolt {

private List<String> commonWords = Arrays.asList("the", "be", "a", "an", "and",

"of", "to", "in", "am", "is", "are", "at", "not", "that", "have", "i", "it",

"for", "on", "with", "he", "she", "as", "you", "do", "this", "but", "his",

"by", "from", "they", "we", "her", "or", "will", "my", "one", "all", "s", "if",

"any", "our", "may", "your", "these", "d" , " ", "me" , "so" , "what" , "him" );

@Override

public void execute(Tuple tuple, BasicOutputCollector collector) {

String w = tuple.getString(0).toLowerCase();

if (!commonWords.contains(w))

{

collector.emit(new Values(w));

}

}

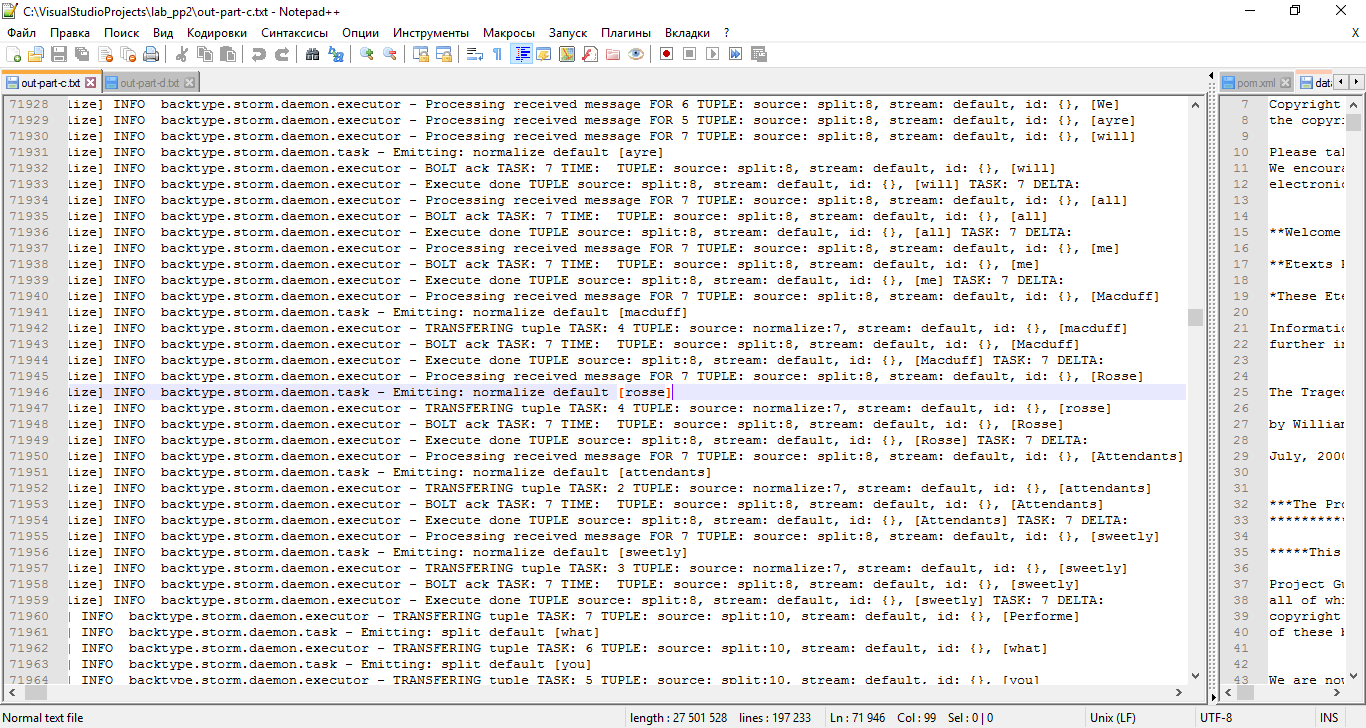
@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("word"));

}

}

Результат:

Как мы видим, добавился пункт “normalize” после которого слова с большим регистром букв (“Rosse”) приводятся к малому регистру.

D) Приведем код файла TopWordFinderTopologyPartD.java:

public class TopWordFinderTopologyPartD {

private static final int N = 10;

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

TopologyBuilder builder = new TopologyBuilder();

Config config = new Config();

config.setDebug(true);

config.setMaxTaskParallelism(3);

config.put("in", args[0]);

config.put(Config.TOPOLOGY\_MAX\_SPOUT\_PENDING, 1);

builder.setSpout("spout", new FileReaderSpout(), 1);

builder.setBolt("split", new SplitSentenceBolt(), 5).shuffleGrouping("spout");

builder.setBolt("normalize", new NormalizerBolt(), 5).shuffleGrouping("split");

builder.setBolt("count", new WordCountBolt(), 5).fieldsGrouping("normalize", new Fields("word"));

builder.setBolt("top-n", new TopNFinderBolt(N)).globalGrouping("count");

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("word-count", config, builder.createTopology());

//wait for 2 minutes and then kill the job

Thread.sleep(2 \* 60 \* 1000);

cluster.shutdown();

}

}

И код TopNFinderBolt.java:

public class TopNFinderBolt extends BaseBasicBolt {

private HashMap<String, Integer> currentTopWords = new HashMap<String, Integer>();

private int N;

private long intervalToReport = 20;

private long lastReportTime = System.currentTimeMillis();

public TopNFinderBolt(int N) {

this.N = N;

}

@Override  
 public void execute(Tuple tuple, BasicOutputCollector collector) {  
  
 String s = tuple.getString(0);  
 Integer i = tuple.getInteger(1);  
 Integer cnt = map.get(s);  
 if (cnt == null)  
 {  
 map.put(s, i);  
 }  
 else  
 {  
 map.put(s, i + cnt);  
 }  
 //reports the top N words periodically  
 if (System.currentTimeMillis() - lastReportTime >= intervalToReport) {  
 Vector<Map.Entry<String, Integer>> vec = new Vector<Map.Entry<String, Integer>>(map.entrySet());  
 Collections.sort(vec, new Comparator<Map.Entry<String, Integer>>(){  
 public int compare(Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2){  
 return o1.getValue().compareTo(o2.getValue());  
 } });  
 int l = Math.max(vec.size() - N, 0);  
 for(int ii = l ; ii < vec.size(); ii++)  
 {  
 currentTopWords.put(vec.get(ii).getKey(), vec.get(ii).getValue());  
 }  
 collector.emit(new Values(printMap()));  
 lastReportTime = System.currentTimeMillis();  
 currentTopWords.clear();  
 }  
 }

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("top-N"));

}

public String printMap() {

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.append("top-words = [ ");

for (String word : currentTopWords.keySet()) {

stringBuilder.append("(" + word + " , " + currentTopWords.get(word) + ") , ");

}

int lastCommaIndex = stringBuilder.lastIndexOf(",");

stringBuilder.deleteCharAt(lastCommaIndex + 1);

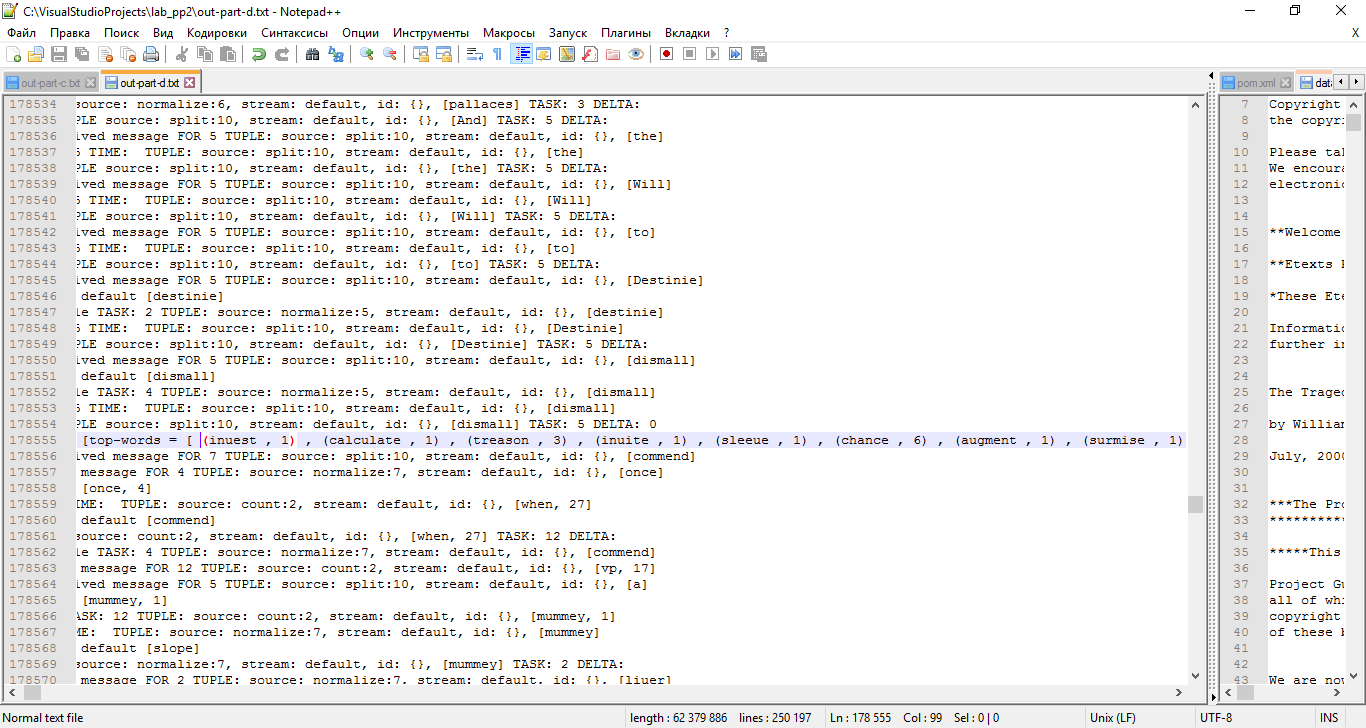
stringBuilder.deleteCharAt(lastCommaIndex);

stringBuilder.append("]");

return stringBuilder.toString();

}

}

Результат:

В файл выводятся строки с подсчитанным количеством самых частых слов.

**Вывод:**

В ходе выполнения лабораторной работы был изучен алгоритм построения топологий для анализа текстов с помощью класса TopologyBuilder.

**Список использованных источников.**

1. Документация Apache Storm класса Topology Builder <https://storm.apache.org/releases/1.0.3/javadocs/org/apache/storm/topology/TopologyBuilder.html>