Machine Problem 4: Apache Storm

1 Overview

Welcome to the Storm machine practice. The final goal of this assignment is to build a topology that finds the top N words in one of Shakespeare's articles. We will build the assignment step by step on top of the topology in the tutorial.

2 General Requirements

All these assignments are designed to work on the **Docker image** that we provide.

Java submission

If you choose to do this assignment in Python, skip this part and go to "Python submission"** part below.

1 Overview and Requirements

This assignment is going to build on **Tutorial 4: Introduction to Storm (Java)**. It is highly recommended that you practice that tutorial before starting this assignment. This assignment will be graded based on **JDK 8**

2 Set up the environment

Step 1: Start the "default" Docker machine that you created when following the "Tutorial: Docker installation" in week 4, run:

```
docker-machine start default
docker-machine env
# follow the instruction to configure your shell: eval $(...)
```

Step 2: Download the Dockerfile and related files for this MP, change the current folder, build, and run the docker image, run:

```
git clone https://github.com/UIUC-public/Docker_MP4_java.git
cd Docker_MP4_java
docker build -t docker_mp4_java .
docker run -it docker_mp4_java bin/bash
```

3 Procedures

Step 3: Download the Java templates and change the current folder, run:

```
git clone https://github.com/UIUC-public/MP4_java.git
cd MP4_java
```

Step 4: Finish the exercises by editing the provided template files. All you need to do is complete the parts marked with **TODO**.

- Each exercise has a Java code template. All you need to do is edit this file.
- Each exercise should be implemented in one file only. Multiple file implementation is not allowed.

Step 5: After you are done with the assignment, submit the zip file containing all your output files (output-part-a.txt, output-part-b.txt. output-part-c.txt, output-part-d.txt). Further submission instructions will be found on the submission page.

Exercise A: Simple Word Count Topology

In this exercise, you are going to build a simple word counter that counts the words a random sentence spout generates. This first exercise is similar to **Tutorial 4: Introduction to Storm**.

In this exercise, we are going to use the "RandomSentenceSpout" class as the spout, the "SplitSentenceBolt" class to split sentences into words, and "WordCountBolt" class to count the words. These components are exactly the same as in the tutorial. You can find the implementation of these classes in **MP4_java/src.**

All you need to do for this exercise is to wire up these components. build the topology, and submit the topology, which is exactly the same as in the tutorial. To make things easier, we have provided a boilerplate for building the topology in the file: **src/TopWordFinderTopologyPartA.**

All you have to do is complete the parts marked as "TODO". Note that this topology will run for 60 seconds and automatically gets killed after that.

NOTE: When connecting the component in the topology (using builder.setSpout() and builder.setBolt()), make sure to use the following names for each component. You might not get full credit if you don't use these names accordingly:

Component	Name
RandomSentanceSpout	"spout"
SplitSentenceBolt	"split"
WordCountBolt	"count"

After completing the implementation of this file, you have to build and run the application using the command below from the **"MP4_java"** directory:

```
mvn clean package
storm jar target/storm-example-0.0.1-SNAPSHOT.jar TopWordFinderTopologyPartA >
  output-part-a.txt
```

Here are **several parts** of a sample output of this application:

```
7179 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - Processing
received message FOR 2 TUPLE: source: split:7, stream: default, id: {}, [cow]
7179 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.task - Emitting: count
default [cow, 1]
7179 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - BOLT ack TASK: 2
TIME: -1 TUPLE: source: split:7, stream: default, id: {}, [cow]
7179 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - Execute done TUPLE
source: split:7, stream: default, id: {}, [cow] TASK: 2 DELTA: -1


7187 [Thread-30-spout-executor[9 9]] INFO  o.a.s.d.task - Emitting: spout
default [the cow jumped over the moon]
7187 [Thread-30-spout-executor[9 9]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 6 tuple: source: spout:9, stream: default, id: {}, [the cow jumped over
the moon]]


7189 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - Processing
received message FOR 6 TUPLE: source: spout:9, stream: default, id: {}, [the cow
jumped over the moon]
7189 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [the]
7189 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 3 tuple: source: split:6, stream: default, id: {}, [the]]
7189 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [cow]
7189 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 2 tuple: source: split:6, stream: default, id: {}, [cow]]
7189 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [jumped]
7189 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 2 tuple: source: split:6, stream: default, id: {}, [jumped]]
7190 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [over]
7190 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 3 tuple: source: split:6, stream: default, id: {}, [over]]
7190 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [the]
7190 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 3 tuple: source: split:6, stream: default, id: {}, [the]]
7190 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [moon]
7190 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 3 tuple: source: split:6, stream: default, id: {}, [moon]]
```

This is just a sample. Data may not be accurate.

Note that you can view the output of the program in the “output-part-a.txt” file. You will be graded based on the content of “output-part-a.txt” file.

Exercise B: Input Data from a File

As can be seen, the spout used in the topology of Exercise A is generating random sentences from a predefined set in the spout’s class. However, we want to count words from one of Shakespeare’s articles. Thus, in this exercise, you are going to create a new spout that reads data from an input file and emits each line as a tuple.

To make the implementation easier, we have provided a boilerplate for the spout needed in the following file: **src/FileReaderSpout.java.**

After finishing the implementation of **FileReaderSpout** class, you have to wire up the topology with this new spout.

To make the implementation easier, we have provided a boilerplate for the topology needed in the following file: **src/TopWordFinderTopologyPartB.java.**

Note that this topology will run for 2 minutes and automatically gets killed after that. There is a chance that you might not process all the data in the input file during this time. However, that is fine and incorporated in the grader.

All you need to do is to make the necessary changes in above files by implementing the sections marked as “TODO”.

NOTE: When connecting the component in the topology (using builder.setSpout() and builder.setBolt()), make sure to use the following names for each component. You might not get full credit if you don’t use these names accordingly:

Component	Name
FileReaderSpout	“spout”
SplitSentenceBolt	“split”
WordCountBolt	“count”

NOTE: You probably want to set the number of executors of the spout to “1” so that you don’t read the input file more than once. However, that depends on your implementation.

We are going to test this application on a Shakespeare article, which is stored in the file “data.txt”. When you are done with the implementation, you should build and run the application again using the following command, from the **"MP4_java"** directory:

```
1  mvn clean package
2  storm jar target/storm-example-0.0.1-SNAPSHOT.jar TopWordFinderTopologyPartB
     data.txt > output-part-b.txt
```

Note that this command assumes you are giving the input file name as an input argument. If needed, you can change the command accordingly.

Here are **several parts** of a sample output of this application:

```
6973 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - Processing
received message FOR 6 TUPLE: source: spout:8, stream: default, id: {}, [***The
Project Gutenberg's Etext of Shakespeare's First Folio***]
6974 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default []
6975 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 3 tuple: source: split:6, stream: default, id: {}, []]
6975 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [The]
6975 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 4 tuple: source: split:6, stream: default, id: {}, [The]]
6975 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [Project]
6975 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 2 tuple: source: split:6, stream: default, id: {}, [Project]]
6975 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [Gutenberg]
6975 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 2 tuple: source: split:6, stream: default, id: {}, [Gutenberg]]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [s]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 4 tuple: source: split:6, stream: default, id: {}, [s]]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [Etext]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 3 tuple: source: split:6, stream: default, id: {}, [Etext]]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [of]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 3 tuple: source: split:6, stream: default, id: {}, [of]]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [Shakespeare]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 2 tuple: source: split:6, stream: default, id: {}, [Shakespeare]]
6976 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [s]
6977 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 4 tuple: source: split:6, stream: default, id: {}, [s]]
6977 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [First]
6977 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 4 tuple: source: split:6, stream: default, id: {}, [First]]
6977 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split
default [Folio]
6977 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 4 tuple: source: split:6, stream: default, id: {}, [Folio]]
6977 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - BOLT ack TASK: 6
TIME: -1 TUPLE: source: spout:8, stream: default, id: {}, [***The Project
Gutenberg's Etext of Shakespeare's First Folio***]
6977 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - Execute done TUPLE
source: spout:8, stream: default, id: {}, [***The Project Gutenberg's Etext of
Shakespeare's First Folio***] TASK: 6 DELTA: -1


6979 [Thread-18-spout-executor[8 8]] INFO  o.a.s.d.task - Emitting: spout
default [Copyright laws are changing all over the world, be sure to check]
6979 [Thread-18-spout-executor[8 8]] INFO  o.a.s.d.executor - TRANSFERING tuple
[dest: 5 tuple: source: spout:8, stream: default, id: {}, [Copyright laws are
changing all over the world, be sure to check]]


7283 [Thread-24-count-executor[3 3]] INFO  o.a.s.d.executor - Processing
received message FOR 3 TUPLE: source: split:6, stream: default, id: {},
[included]
7283 [Thread-24-count-executor[3 3]] INFO  o.a.s.d.task - Emitting: count
default [included, 2]
```

This is just a sample. Data may not be accurate.

Note that you can view the output of the program in the “output-part-b.txt” file. You will be graded based on the content of “output-part-b.txt” file.

Exercise C: Normalizer Bolt

The application we developed in Exercise B counts the words “Apple” and “apple” as two different words. However, if we want to find the top N words, we have to count these words the same. Additionally, we don’t want to take common English words into consideration.

Therefore, in this part we are going to normalize the words by adding a normalizer bolt that gets the words from the splitter, normalizes them, and then sends them to the counter bolt. The responsibility of the normalizer is to:

1. Make all input words lowercase.

2. Remove common English words.

To make the implementation easier, we have provided a boilerplate for the normalizer bolt in the following file: **src/NormalizerBolt.java.**

There is a list of common words to filter in this class, so please make sure you use this exact list to in order to receive the maximum points for this part. After finishing the implementation of this class, you have to wire up the topology with this bolt added to the topology.

To make the implementation easier, we have provided a boilerplate for the topology needed in the following file: **src/TopWordFinderTopologyPartC.java.**

Note that this topology will run for 2 minutes and automatically gets killed after that. There is a chance that you might not process all the data in the input file during this time. However, that is fine and incorporated in the grader.

All you need to do is to make the necessary changes in above files by implementing the sections marked as "TODO".

NOTE: When connecting the component in the topology (using builder.setSpout() and builder.setBolt()), make sure to use the following names for each component. You might not get full credit if you don't use these names accordingly:

Component	Name
FileReaderSpout	"spout"
SplitSentenceBolt	"split"
WordCountBolt	"count"
NormalizerBolt	"normalize"

When you are done with the implementation, you should build and run the application again using the following command, from the **"MP4_java"** directory:

```
mvn clean package
storm jar target/storm-example-0.0.1-SNAPSHOT.jar TopWordFinderTopologyPartC
  data.txt > output-part-c.txt
```

Here is **a part** of a sample output of this application:

```
6949 [Thread-28-normalize-executor[6 6]] INFO  o.a.s.d.executor - Processing
received message FOR 6 TUPLE: source: split:9, stream: default, id: {}, [Etext]
6950 [Thread-28-normalize-executor[6 6]] INFO  o.a.s.d.task - Emitting:
normalize default [etext]
```

Note that you can view the output of the program in the "output-part-c.txt" file. You will be graded based on the content of "output-part-c.txt" file.

Exercise D: Top *N* Words

In this exercise, we are going to find the top *N* words. To complete this part, we have to build a topology that reads from an input file, splits it into words, normalizes the words, and counts the number of occurrences of each word. In this exercise, we are going to use the output of the count bolt to keep track of and periodically report the top *N* words.

For this purpose, you have to implement a bolt that keeps count of the top *N* words. Upon receipt of a new count from the count bolt, it updates the top *N* words. Then, it reports the top *N* words periodically. To make the implementation easier, we have provided a boilerplate for the top-*N* finder bolt in the following file: **TopNFinderBolt.java.**

After finishing the implementation of this class, you have to wire up the topology with this bolt added to the topology.

To make the implementation easier, we have provided a boilerplate for the topology needed in the following file: **src/TopWordFinderTopologyPartD.java.**

Note that this topology will run for 2 minutes and automatically gets killed after that. There is a chance that you might not process all the data in the input file during this time. However, that is fine and incorporated in the grader.

All you need to do is to make the necessary changes in above files by implementing the sections marked as "TODO".

NOTE: When connecting the component in the topology (using builder.setSpout() and builder.setBolt()), make sure to use the following names for each component. You might not get full credit if you don't use these names accordingly:

Component	Name
FileReaderSpout	"spout"
SplitSentenceBolt	"split"
WordCountBolt	"count"
NormalizerBolt	"normalize"
TopNFinderBolt	"top-n"

When you are done with the implementation, you should build and run the application again using the following command, from the **"MP4_java"** directory:

```
mvn clean package
storm jar target/storm-example-0.0.1-SNAPSHOT.jar TopWordFinderTopologyPartD
  data.txt > output-part-d.txt

```

Here is **a part** of a sample output of this application:

```
7760 [Thread-20-top-n-executor[12 12]] INFO  o.a.s.d.executor - Processing
received message FOR 12 TUPLE: source: count:4, stream: default, id: {}, [plays,
1]
7760 [Thread-20-top-n-executor[12 12]] INFO  o.a.s.d.task - Emitting: top-n
default [top-words = [ (plays , 1) , (etext , 1) , (shakespeare , 1) , (macbeth
, 1) , (folio , 1) , (project , 1) , (index , 1) , (edition , 1) , (gutenberg ,
1) , (first , 1) ]]
```

Note that you can view the output of the program in the "output-part-d.txt" file. You will be graded based on the content of "output-part-d.txt" file.

Python submission

If you choose to do this assignment in Java, skip this part and go to **"Java submission" part above

0 Tutorial

This tutorial covers the quickstart of working with Streamparse to run Python code against real-time streams of data via Apache Storm. You will build your first project following the instructions.

step 1: Start the "default" Docker machine that you created when following the "Tutorial: Docker installation" in week 4, run:

```
docker-machine start default
docker-machine env
# follow the instruction to configure your shell: eval $(...)
```

Step 2: Download the Dockerfile and related files for this tutorial, change the current folder, build, and run the docker image, run:

```
git clone https://github.com/UIUC-public/Docker_MP4_py.git
cd Docker_MP4_py
docker build -t docker_mp4_py .
docker run -it docker_mp4_py bin/bash
```

Step 3: Set the LEIN_ROOT environment variable to true, run:

```
export LEIN_ROOT=true
```

Step 4: Create a project using the command-line tool, **sparse**, and enter the project folder:

```
sparse quickstart wordcount
cd wordcount
```

Step 6: Edit the project.clj file in the folder and change the Apache Storm library version from "1.0.2" to "1.0.6" as the local Storm version is 1.0.6 in our Docker image.

```
vim project.clj
```

Step 5: Try running your topology locally with:

```
sparse run
```

The quickstart project provides a basic wordcount topology example which you can examine and modify.

For more information about Streamparse, you may visit:

https://streamparse.readthedocs.io/en/stable/index.html

1 Requirements

This assignment will be graded based on **Python 2.7.** The Docker image built from provided Dockerfile will have JDK 8, lein, streamparse, and Apache Storm installed.

2 Set up the environment

Step 1: Start the "default" Docker machine that you created when following the "Tutorial: Docker installation" in week 4, run:

```
docker-machine env
# follow the instruction to configure your shell: eval $(...)
docker-machine start default
```

Step 2: Download the Dockerfile and related files for this MP, change the current folder, build, and run the docker image, run:

```
git clone https://github.com/UIUC-public/Docker_MP4_py.git
cd Docker_MP4_py
docker build -t docker_mp4_py .
docker run -it docker_mp4_py bin/bash
```

3 Procedures

Step 3: Download the Python templates and change the current folder, run:

```
git clone https://github.com/UIUC-public/MP4_py.git
cd MP4_py
```

step 4: Start storm nimbus, dev-zookeeper, and supervisor

```
storm nimbus &
storm dev-zookeeper &
storm supervisor &
```

You can check if you started Storm nimbus, dev-zookeeper, and supervisor successfully by running "jps" command. If you encounter "Error:KeeperErrorCode = NoNode for /storm/leader-lock" when starting dev-zookeeper, you may just ignore it. That should not affect the results of this MP.

Step 5: Start the OpenSSH (SSH) Server

```
service ssh start
```

Step 4: Finish the exercises by editing the provided template files. All you need to do is complete the parts marked with **TODO**.

Step 5: After you are done with the assignment, you will need to submit the zip file containing all your output files (output-part-a-count.txt, output-part-a-split.txt, output-part-a-spout.txt, output-part-b-count.txt, output-part-b-split.txt, output-part-b-spout.txt, output-part-c-normalize.txt, output-part-d-topn.txt). Further submission instructions will be found on the submission page.

Exercise A: Simple Word Count Topology

In this exercise, you are going to build a simple word counter that counts the words a random sentence spout generates. We are going to use the “RandomSentenceSpout” as the spout, the “SplitSentenceBolt” to split sentences into words, and “WordCountBolt” to count the words. You will find these files in **src/bolts** and **src/spouts**.

All you need to do for this exercise is to wire up these components. build the topology, and submit the topology. To make things easier, we have provided a boilerplate for building the topology in the file: **topologies/TopWordFinderTopologyPartA.py**.

All you have to do is complete the parts marked as “TODO”. Note that this topology will need to be killed manually later.

NOTE: When connecting the component in the topology, make sure to use the following names for each component. You might not get full credit if you don’t use these names accordingly:

Component	Name
RandomSentanceSpout	“spout”
SplitSentenceBolt	“split”
WordCountBolt	“count”

After completing the implementation of this file, you have to build and run the application using the command below from the **"MP4_py"** directory:

```
sparse submit -n TopWordFinderTopologyPartA
```

To check the list of running Topologies, run

```
storm list
```

To kill this Topology, run

```
storm kill TopWordFinderTopologyPartA
```

Topology logs are available in the directory: /var/log/storm/streamparse

To check the lines of content in the logs, run

```
wc -l /var/log/storm/streamparse/*
```

Save one of the nonempty count log files for this Topology as "output-part-a-count.txt". Save one of the nonempty split log files for this Topology as "output-part-a-split.txt". Save one of the nonempty spout log files for this Topology as "output-part-a-spout.txt". You will be graded based on the content of these files.

Here is **part** of a sample output "output-part-a-count.txt" of this application:

```
2018-04-03 23:56:20,876 - pystorm.component.count - INFO - - [pid=629] -
Processing received message [dwarfs]
2018-04-03 23:56:20,877 - pystorm.component.count - INFO - - [pid=629] -
Emitting: count [dwarfs,9621]
2018-04-03 23:56:20,929 - pystorm.component.count - INFO - - [pid=629] -
Processing received message [dwarfs]
2018-04-03 23:56:20,929 - pystorm.component.count - INFO - - [pid=629] -
Emitting: count [dwarfs,9622]
2018-04-03 23:56:20,930 - pystorm.component.count - INFO - - [pid=629] -
Processing received message [dwarfs]
2018-04-03 23:56:20,930 - pystorm.component.count - INFO - - [pid=629] -
Emitting: count [dwarfs,9623]
2018-04-03 23:56:20,949 - pystorm.component.count - INFO - - [pid=629] -
Processing received message [dwarfs]
2018-04-03 23:56:20,949 - pystorm.component.count - INFO - - [pid=629] -
Emitting: count [dwarfs,9624]

```

Here is **part** of a sample output "output-part-a-split.txt" of this application:

```
2018-04-03 23:43:33,675 - pystorm.component.split - INFO - - [pid=610] -
Processing received message [four score and seven years ago]
2018-04-03 23:43:33,675 - pystorm.component.split - INFO - - [pid=610] -
Emitting: split [four]
2018-04-03 23:43:33,675 - pystorm.component.split - INFO - - [pid=610] -
Emitting: split [score]
2018-04-03 23:43:33,676 - pystorm.component.split - INFO - - [pid=610] -
Emitting: split [and]
2018-04-03 23:43:33,676 - pystorm.component.split - INFO - - [pid=610] -
Emitting: split [seven]
2018-04-03 23:43:33,676 - pystorm.component.split - INFO - - [pid=610] -
Emitting: split [years]
2018-04-03 23:43:33,676 - pystorm.component.split - INFO - - [pid=610] -
Emitting: split [ago]
```

Here is **part** of a sample output "output-part-a-spout.txt" of this application:

```
2018-04-03 23:53:58,977 - pystorm.component.spout - INFO - - [pid=625] -
Emitting: spout [an apple a day keeps the doctor away]
2018-04-03 23:53:59,079 - pystorm.component.spout - INFO - - [pid=625] -
Emitting: spout [four score and seven years ago]
2018-04-03 23:53:59,180 - pystorm.component.spout - INFO - - [pid=625] -
Emitting: spout [snow white and the seven dwarfs]
```

This is just a sample. Data may not be accurate.

Exercise B: Input Data from a File

As can be seen, the spout used in the topology of Exercise A is generating random sentences from a predefined set in the spout's class. However, we want to count words from one of Shakespeare's articles. Thus, in this exercise, you are going to create a new spout that reads data from an input file and emits each line as a tuple.

To make the implementation easier, we have provided a boilerplate for the spout needed in the following file: **src/spouts/FileReaderSpout.py.**

After finishing the implementation of **FileReaderSpout**, you have to wire up the topology with this new spout.

To make the implementation easier, we have provided a boilerplate for the topology needed in the following file: **topologies/TopWordFinderTopologyPartB.py.**

All you have to do is complete the parts marked as "TODO". Note that this topology will need to be killed manually later.

NOTE: When connecting the component in the topology, make sure to use the following names for each component. You might not get full credit if you don't use these names accordingly:

Component	Name
FileReaderSpout	"spout"
SplitSentenceBolt	"split"
WordCountBolt	"count"

We are going to test this application on a Shakespeare article, which is stored in the file "data.txt". When you are done with the implementation, you should build and run the application again using the following command, from the **"MP4_py"** directory:

```
sparse submit -n TopWordFinderTopologyPartB
```

To kill this Topology, run

```
storm kill TopWordFinderTopologyPartB
```

Topology logs are available in the directory: /var/log/storm/streamparse

Save one of the nonempty count log files for this Topology as "output-part-b-count.txt". Save one of the nonempty split log files for this Topology as "output-part-b-split.txt". Save one of the nonempty spout log files for this Topology as "output-part-b-spout.txt". You will be graded based on the content of these files.

Here is **part** of a sample output "output-part-b-count.txt" of this application:

```
2018-04-04 00:37:01,397 - pystorm.component.count - INFO - - [pid=1356] -
Processing received message [tax]
2018-04-04 00:37:01,418 - pystorm.component.count - INFO - - [pid=1356] -
Emitting: count [tax,2]
2018-04-04 00:37:01,418 - pystorm.component.count - INFO - - [pid=1356] -
Processing received message [Gutenberg's]
2018-04-04 00:37:01,419 - pystorm.component.count - INFO - - [pid=1356] -
Emitting: count [Gutenberg's,3]
2018-04-04 00:37:01,443 - pystorm.component.count - INFO - - [pid=1356] -
Processing received message [of]
2018-04-04 00:37:01,443 - pystorm.component.count - INFO - - [pid=1356] -
Emitting: count [of,54]
2018-04-04 00:37:01,443 - pystorm.component.count - INFO - - [pid=1356] -
Processing received message [Tragedie]
2018-04-04 00:37:01,444 - pystorm.component.count - INFO - - [pid=1356] -
Emitting: count [Tragedie,4]

```

Here is **part** of a sample output "output-part-b-split.txt" of this application:

```
2018-04-04 00:36:58,794 - pystorm.component.split - INFO - - [pid=1347] -
Processing received message [Project Gutenberg Etexts are usually created from
multiple editions,
]
2018-04-04 00:36:58,794 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [Project]
2018-04-04 00:36:58,794 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [Gutenberg]
2018-04-04 00:36:58,794 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [Etexts]
2018-04-04 00:36:58,795 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [are]
2018-04-04 00:36:58,795 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [usually]
2018-04-04 00:36:58,795 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [created]
2018-04-04 00:36:58,795 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [from]
2018-04-04 00:36:58,795 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [multiple]
2018-04-04 00:36:58,795 - pystorm.component.split - INFO - - [pid=1347] -
Emitting: split [editions,]
```

Here is **part** of a sample output "output-part-b-spout.txt" of this application:

```
2018-04-04 00:36:58,765 - pystorm.component.spout - INFO - - [pid=1354] -
Emitting:spout [Please note:  neither this list nor its contents are final till
]
2018-04-04 00:36:58,849 - pystorm.component.spout - INFO - - [pid=1354] -
Emitting:spout [midnight of the last day of the month of any such announcement.
]
2018-04-04 00:36:58,850 - pystorm.component.spout - INFO - - [pid=1354] -
Emitting:spout [The official release date of all Project Gutenberg Etexts is at
]
2018-04-04 00:36:58,877 - pystorm.component.spout - INFO - - [pid=1354] -
Emitting:spout [Midnight, Central Time, of the last day of the stated month.  A
]
```

This is just a sample. Data may not be accurate.

Exercise C: Normalizer Bolt

The application we developed in Exercise B counts the words “Apple” and “apple” as two different words. However, if we want to find the top N words, we have to count these words the same. Additionally, we don’t want to take common English words into consideration.

Therefore, in this part we are going to normalize the words by adding a normalizer bolt that gets the words from the splitter, normalizes them, and then sends them to the counter bolt. The responsibility of the normalizer is to:

1. Make all input words lowercase.

2. Remove common English words.

To make the implementation easier, we have provided a boilerplate for the normalizer bolt in the following file: **src/bolts/NormalizerBolt.py.**

There is a list of common words to filter in this file, so please make sure you use this exact list to in order to receive the maximum points for this part. After finishing the implementation of this file, you have to wire up the topology with this bolt added to the topology.

To make the implementation easier, we have provided a boilerplate for the topology needed in the following file: **topologies/TopWordFinderTopologyPartC.py.**

All you have to do is complete the parts marked as “TODO”. Note that this topology will need to be killed manually later.

NOTE: When connecting the component in the topology, make sure to use the following names for each component. You might not get full credit if you don’t use these names accordingly:

Component	Name
FileReaderSpout	“spout”
SplitSentenceBolt	“split”
WordCountBolt	“count”
NormalizerBolt	“normalize”

When you are done with the implementation, you should build and run the application again using the following command, from the **"MP4_py"** directory:

```
1  sparse submit -n TopWordFinderTopologyPartC
```

To kill this Topology, run

```
1  storm kill TopWordFinderTopologyPartC
```

Topology logs are available in the directory: /var/log/storm/streamparse

Save one of the nonempty normalize log files for this Topology as "output-part-c-normalize.txt". You will be graded based on the content of this file. Please note that common English words should not appear in this file.

Here is **part** of a sample output of this application:

```
2018-04-04 01:46:33,851 - pystorm.component.normalize - INFO - - [pid=585] -
Processing received message [Etexts,]
2018-04-04 01:46:33,855 - pystorm.component.normalize - INFO - - [pid=585] -
Processing emitting: normalize [etexts,]
2018-04-04 01:46:33,859 - pystorm.component.normalize - INFO - - [pid=585] -
Processing received message [month.]
2018-04-04 01:46:33,868 - pystorm.component.normalize - INFO - - [pid=585] -
Processing emitting: normalize [month.]
2018-04-04 01:46:33,876 - pystorm.component.normalize - INFO - - [pid=585] -
Processing received message [Information]
2018-04-04 01:46:33,884 - pystorm.component.normalize - INFO - - [pid=585] -
Processing emitting: normalize [information]
```

This is just a sample. Data may not be accurate.

Exercise D: Top *N* Words

In this exercise, we are going to find the top *N* words. To complete this part, we have to build a topology that reads from an input file, splits it into words, normalizes the words, and counts the number of occurrences of each word. In this exercise, we are going to use the output of the count bolt to keep track of and periodically report the top *N* words.

For this purpose, you have to implement a bolt that keeps count of the top *N* words. Upon receipt of a new count from the count bolt, it updates the top *N* words. Then, it reports the top *N* words periodically. To make the implementation easier, we have provided a boilerplate for the top-*N* finder bolt in the following file: **src/bolts/TopNFinderBolt.py.**

After finishing the implementation of this file, you have to wire up the topology with this bolt added to the topology.

To make the implementation easier, we have provided a boilerplate for the topology needed in the following file: **topologies/TopWordFinderTopologyPartD.py.**

All you have to do is complete the parts marked as “TODO”. Note that this topology will need to be killed manually later.

NOTE: When connecting the component in the topology, make sure to use the following names for each component. You might not get full credit if you don’t use these names accordingly:

Component	Name
FileReaderSpout	“spout”
SplitSentenceBolt	“split”
WordCountBolt	“count”
NormalizerBolt	“normalize”
TopNFinderBolt	“top-n”

When you are done with the implementation, you should build and run the application again using the following command, from the **"MP4_py"** directory:

```
sparse submit -n TopWordFinderTopologyPartD
```

To kill this Topology, run

```
storm kill TopWordFinderTopologyPartD
```

Topology logs are available in the directory: /var/log/storm/streamparse

Save one of the nonempty top-n log files for this Topology as "output-part-d-topn.txt". You will be graded based on the content of this file.

Here is **part** of a sample output of this application:

```
1  2018-04-04 01:50:30,365 - pystorm.component.top-n - INFO -  - [pid=1292] -
   Processing received message [crowne,3]
2  2018-04-04 01:50:30,365 - pystorm.component.top-n - INFO -  - [pid=1292] -
   Processing received message [out,10]
3  2018-04-04 01:50:30,367 - pystorm.component.top-n - INFO - - [pid=1292] -
   Emitting: top-n [top-word = [(u'which', 2556), (u'vpon', 1711), (u'macd.',
   1711), (u'hath', 1378), (u'was', 1225), (u'enter', 1176), (u'like', 780),
   (u'must', 741), (u'come', 741), (u'would', 666)]]
4  2018-04-04 01:50:30,369 - pystorm.component.top-n - INFO -  - [pid=1292] -
   Processing received message [two-fold,1]
5  2018-04-04 01:50:30,369 - pystorm.component.top-n - INFO -  - [pid=1292] -
   Processing received message [so.,4]
6  2018-04-04 01:50:30,371 - pystorm.component.top-n - INFO -  - [pid=1292] -
   Processing received message [blood-bolter'd,1]
```

This is just a sample. Data may not be accurate.

mp