School of Information Technology and Electrical Engineering
INFS3208 – Cloud Computing

Programming Assignment Task III (10 Marks)

Task Description:

In this assignment, you are asked to write a piece of Spark code to **count occurrences of verbs in the collection of Shakespeare. The returned result should be the top 10 verbs that are most frequently used in Shakespeare's collection.** This assignment is to test your ability to use transformation and action operations in Spark RDD programming. You will be given the collection file (shakespeare.txt), a verb list (all_verbs.txt), a verb dictionary file (verb_dict.txt), and the programming environment (a docker-compose). You can choose either Scala or Python to program in the Jupyter Notebook. There are some technical requirements in your code submission as follows:

1. You should use an appropriate method to load the files into RDDs. You are NOT allowed to make changes to the collection file (`shakespeare.txt`), verb list file (`all_verbs.txt`), and verb dictionary file (`verb_dict.txt`).
2. To accurately count the verbs in the collection, you should use learned RDD operations to pre-process the text in the collection file:
    a. Remove empty lines;
    b. Remove punctuations that could attach to the verbs;
    E.g., "`work,`" and "`work`" will be counted differently, if you DO NOT remove the punctuation mark.
    c. Change the capitalization or case of text
    E.g., "`WORK`", "`Work`" and "`work`" will be counted as three different verbs, if you DO NOT make all of them in lower-case.
3. You should use learned RDD operations to find out used verbs in the collection (`shakespeare.txt`) by matching the verbs in the given verb list (`all_verbs.txt`).
4. A verb can have different forms: present tense, past tense, and future tense
    a. E.g., regular verb: "`work`" - `works`", "`worked`", and "`working`".
    b. E.g., irregular verb: "`begin`" - "`begins`", "`began`", and "`begun`".
    c. E.g., linking verb "`be`" and its various forms, including "`is`", "`am`", "`are`", "`was`", "`were`", "`being`" and "`been`".
    You should use the learned RDD operations to calculate the occurrences of all the verbs (listed in the given verb dictionary file) and merge the verbs that have different tenses by using the learned RDD operations to look up the verb dictionary file (`verb_dict.txt`).
    d. E.g., (`work, 100`), (`works,50`), (`working,150`) $\rightarrow$ (**`work, 300`**).
5. In the final result, you should return the **top 10 verbs** (in the base form, e.g., `work`) that are most frequently used in the collection of Shakespeare.

Preparation:

In this individual coding assignment, you will apply your knowledge of Spark and Spark RDD Programming (in Lectures 8 & 9). Firstly, you should read **Task Description** to understand what the task is and what the technical requirements include. Secondly, you should review all the transformation and action operations in Lectures 8 & 9. In the Appendix, there are some transformation and action

operations you could use in this assignment. Also, it would be very helpful to practise the RDD programming activity in Prac 7 (Week 8). Lastly, you need to write the code (Scala or Python) in the Jupyter Notebook. **All technical requirements need to be fully met to achieve full marks.**
You can either practise on the GCP's VM or your local machine with Oracle Virtualbox if you are unable to access GCP. Please read the **Example of writing Spark code** below to have more details.

Assignment Submission:

- You need to compress the Jupyter Notebook file.
- The name of the compressed file should be named "FirstName_LastName_StudentNo.zip".
- You must make an online submission to Blackboard before 1:00 PM 14/10/2022.
- Only one extension application could be approved due to medical conditions.

Example of writing Spark code:

Step 1:

Log in your VM and change to your home directory

Step 2:

Download docker-compose.yml and the data you required (`shakespeare.txt, all_verbs.txt, verb_dict.txt`).

```
git clone https://github.com/csenw/cca3.git && cd cca3

sudo chmod -R 777 nbs/
```

Step 3:

Run all the containers: `docker-compose up -d`

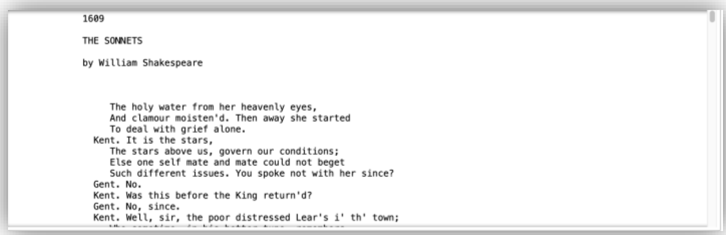
Step 4:

Open the Jupyter Notebook (`EXTERNAL_IP:8888`) and write the Spark code in it.

Step 5:

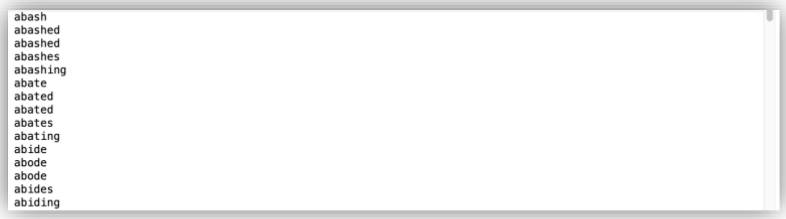
Use the learned method to load external files (`shakespeare.txt, all_verbs.txt, verb_dict`) into RDDs.

Output samples:

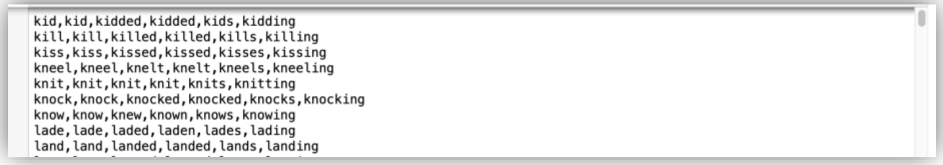


```
1609

THE SONNETS

by William Shakespeare



    The holy water from her heavenly eyes,
    And clamour moisten'd. Then away she started
    To deal with grief alone.
  Kent. It is the stars,
    The stars above us, govern our conditions;
    Else one self mate and mate could not beget
    Such different issues. You spoke not with her since?
  Gent. No.
  Kent. Was this before the King return'd?
  Gent. No, since.
  Kent. Well, sir, the poor distressed Lear's i' th' town;
```

shakespeare.txt


```
abash
abashed
abashed
abashes
abashing
abate
abated
abated
abates
abating
abide
abode
abode
abides
abiding
```

all_verbs.txt


```
kid,kid,kidded,kidded,kids,kidding
kill,kill,killed,killed,kills,killing
kiss,kiss,kissed,kissed,kisses,kissing
kneel,kneel,knelt,knelt,kneels,kneeling
knit,knit,knit,knit,knits,knitting
knock,knock,knocked,knocked,knocks,knocking
know,know,knew,known,knows,knowing
lade,lade,laded,laden,lades,lading
land,land,landed,landed,lands,landing
```

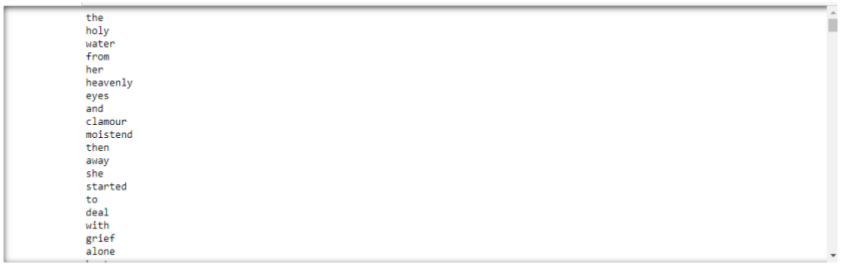
verb_dict.txt

Step 6:

Use learned RDD operations to pre-process the RDD that stores the text:

1. Remove empty lines
2. Remove punctuations that could attach to the verbs;
3. Change the capitalization or case of text

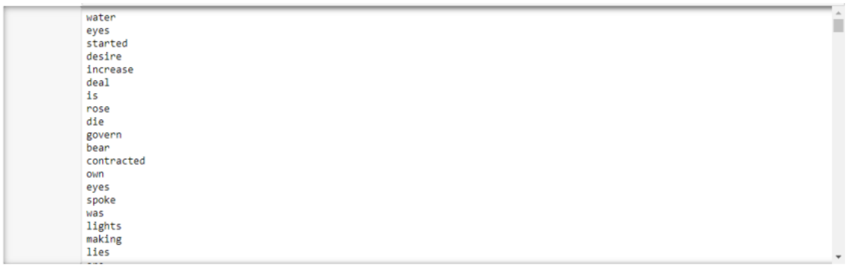
Output sample:


```
the
holy
water
from
her
heavenly
eyes
and
clamour
moistend
then
away
she
started
to
deal
with
grief
alone
```

Step 7:

Use learned RDD operations to keep all the verbs according to the `all_verbs.txt`.

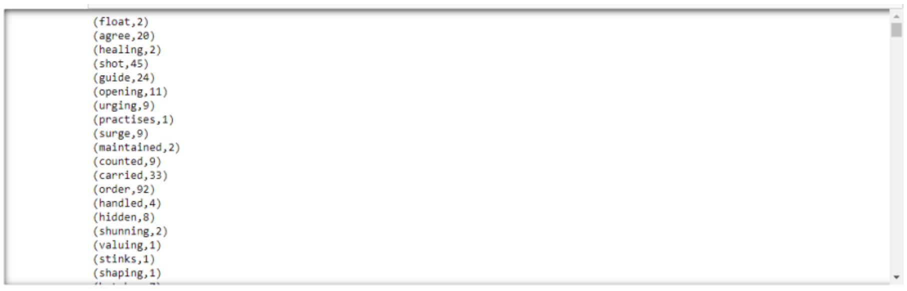
Output sample:


```
water
eyes
started
desire
increase
deal
is
rose
die
govern
bear
contracted
own
eyes
spoke
was
lights
making
lies
```

Step 8:

Use learned RDD operations to count the occurrences of the kept verbs:

Output sample:

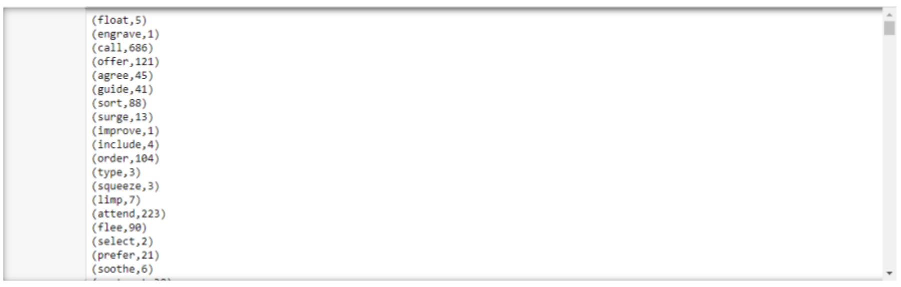


```
(float,2)
(agree,20)
(healing,2)
(shot,45)
(guide,24)
(opening,11)
(urging,9)
(practises,1)
(surge,9)
(maintained,2)
(counted,9)
(carried,33)
(order,92)
(handled,4)
(hidden,8)
(shunning,2)
(valuing,1)
(stinks,1)
(shaping,1)
```

Step 9:

Use learned RDD operations to merge the verb pairs that are from the same verb. E.g. (`work, 100`), (`works,50`), (`working,150`) → (**`work, 300`**).

Output sample:



```
(float,5)
(engrave,1)
(call,686)
(offer,121)
(agree,45)
(guide,41)
(sort,88)
(surge,13)
(improve,1)
(include,4)
(order,104)
(type,3)
(squeeze,3)
(limp,7)
(attend,223)
(flee,90)
(select,2)
(prefer,21)
(soothe,6)
```

Step 10:

Use learned RDD operations to return the top 10 that are most frequently used in the collection of Shakespeare.

Output sample:

```
(be,26727)
(have,7848)
(do,6416)
(come,3610)
(make,2892)
(go,2575)
(love,2501)
(let,2384)
(say,2356)
(know,2251)
```

Note that Steps 5 -10 are just recommended procedure, you can feel free to use your processing steps. However, your result should reasonably reflect the top 10 verbs that are most frequently used in the collection of Shakespeare.

Appendix:

Transformations:

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func*returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func*should return a Seq rather than a single item).
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numPartitions*]))	Return a new dataset that contains the distinct elements of the source dataset.
groupByKey([numPartitions])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numPartitions argument to set a different number of tasks.
reduceByKey(func, [numPartitions])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
sortByKey([ascending], [numPartitions])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.
join(otherDataset, [numPartitions])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through leftOuterJoin, rightOuterJoin, and fullOuterJoin.

Actions:

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset.
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(func)	Run a function func on each element of the dataset. This is usually done for side effects such as updating an Accumulator or interacting with external storage systems. Note: modifying variables other than Accumulators outside of the foreach() may result in undefined behavior. See Understanding closures for more details.