**INVERTED INDEX DOCUMENTATION**

Dhanush Bommavaram – SE20UCSE039

Sreevaatsav Bavana – SE20UARI147

Chaitanya Srikanth – SE20UARI038

M Pranav Reddy – SE20UARI090

Submission Structure

1. BigDataProject
   1. Inverted\_index.py
   2. Text\_files folder
      1. T1.txt
      2. T2.txt
      3. T3.txt
      4. T4.txt

The code demonstrates the concept of inverted indexing using Apache Spark’s RDD’s.

Paramters:

Firstly, we need to give the path of the text files directory

path(str): The file path to the directory containing the input text files

Outputs:

Op\_final (list): The final processed output in the form of tuples

This code demonstrates text processing operations using Apache Spark's RDD (Resilient Distributed Dataset) API.

Parameters:

path (str): The file path to the directory containing the input text files.

Returns:

op\_final (list): The final processed output in the form of a list of tuples.

Code Explanation:

# Create a SparkContext object

sc = SparkContext()

//Read the files and partition the data into 3 parts.

rdd = sc.wholeTextFiles(path, minPartitions=3)

# Step 1: Convert all lines in each file to lowercase and split them into individual words.

# Create key-value pairs (key -> file\_path,val-> word)

#Iterate through all the lines and split the words.

output0 = rdd.flatMapValues(lambda allLines: allLines.lower().split())

# Step 2: Modify the key-value pairs to have the format (word, file\_path).

output1 = output0.map(lambda keyVal: (keyVal[1], keyVal[0].replace("file:"+path+"/", "", 1)))

# Step 3: Concatenate the lists based on the same key value

#output : (word, [

input1 = output1.mapValues(lambda x: [x]).reduceByKey(lambda x, y: x + y)

# Step 4: Flatten the values (file\_paths) into individual elements and create key-value pairs with the format (word, (file\_path, 1)).

output2 = input1.flatMapValues(lambda fileList: fileList).mapValues(lambda file: (file, 1))

# Step 5: Modify each key-value pair to have the format ((word, file\_path), count),

# and sum the counts for the same (word, file\_path) tuples.

output3 = output2.map(lambda x: ((x[0], x[1][0]), x[1][1])).reduceByKey(lambda k1, k2: k1 + k2)

# Step 6: Modify the key-value pairs to have the format (word, [(file\_path, count)]).

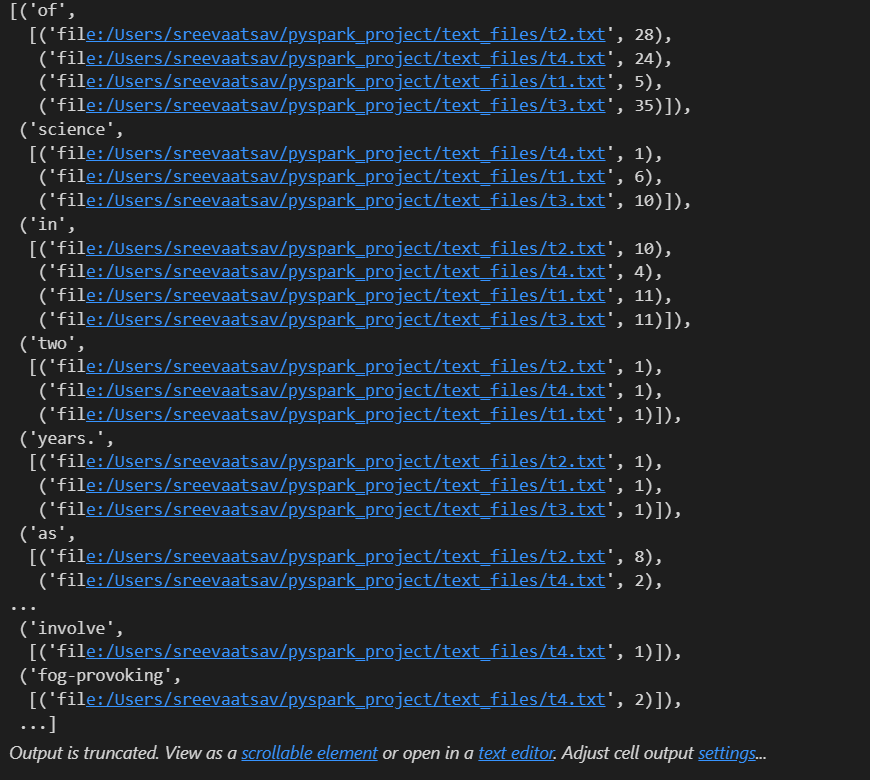
output4 = output3.map(lambda x: (x[0][0], (x[0][1], x[1]))).mapValues(lambda x: [x]).reduceByKey(lambda x, y: x + y)

#Use the .collect() to get the outputs as a list of tuples

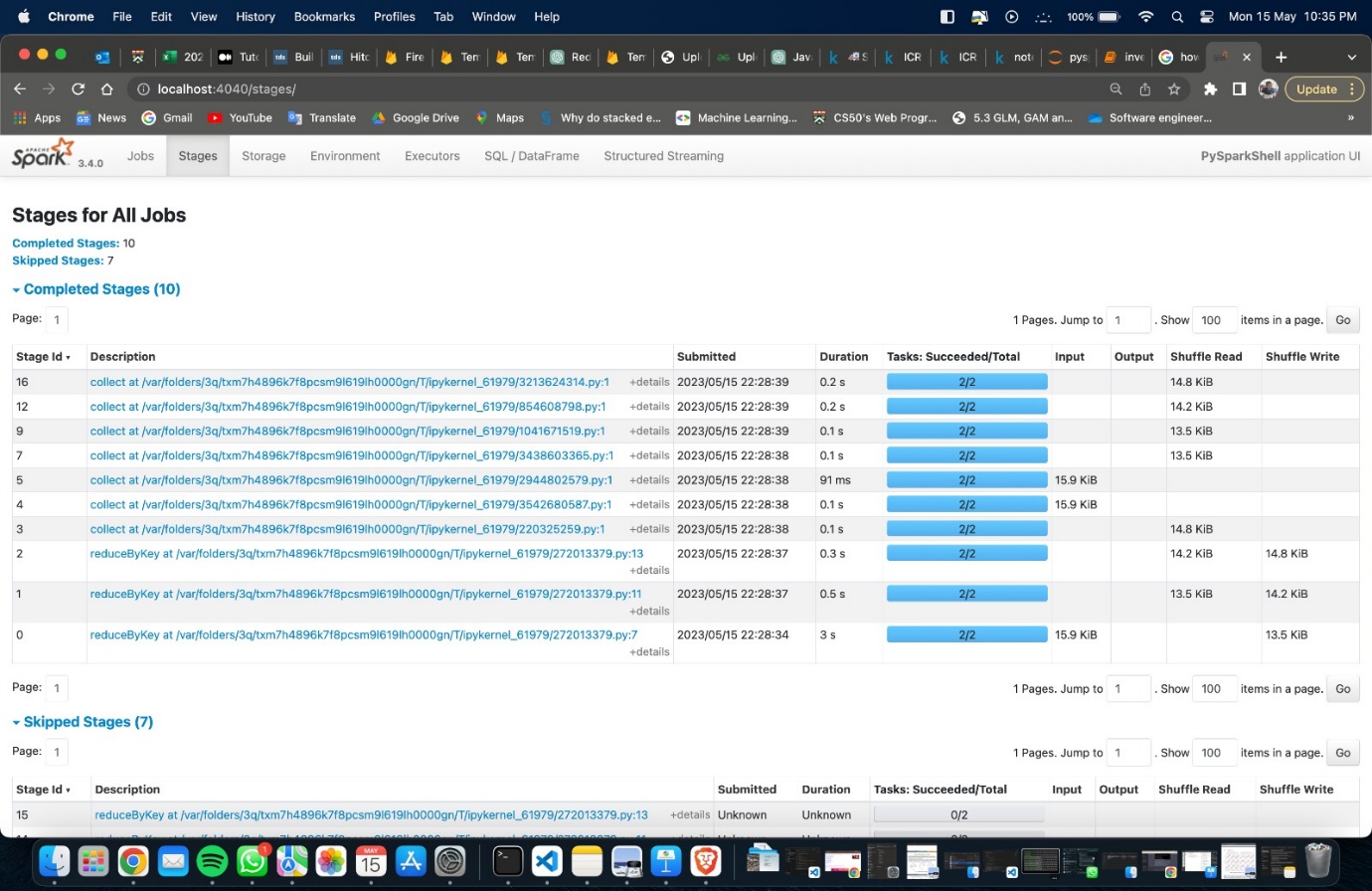
op\_final = output4.collect()

With the above code implemented, we get the outputs in the form of ([word, [(filepath, count)]).

Outputs:



**Functions on the RDD’s**



**Executor and tasks**

