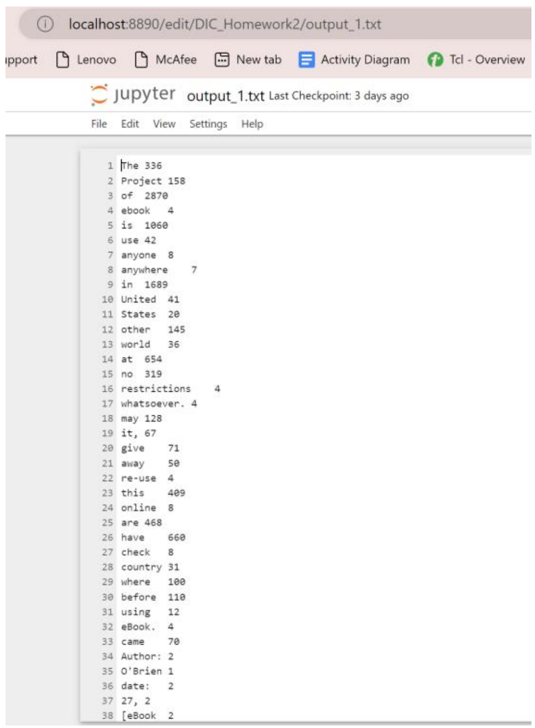
Homework #2 – Spark

Sahithya Arveti Nagaraju (50559752)

Part -1 Implement and analyze word count.

Implementation of basic word count. You must write a basic word count program that reads two text files, counts the number of occurrences of each word in the text files, and outputs the result back to a text file as a list of key-value pairs, where the key is the word, and the value is the number of times the word occurred in the text file named output_1.txt. Provide screen shot of your output.

Output of output_1.txt after implementing basic word count program:




```
The 336
Project 158
of 2870
ebook 4
is 1060
use 42
anyone 8
anywhere 7
in 1689
United 41
States 20
other 145
world 36
at 654
no 319
restrictions 4
whatsoever. 4
may 128
it, 67
give 71
away 50
re-use 4
this 409
online 8
are 468
have 660
check 8
country 31
where 100
before 110
using 12
eBook. 4
came 70
Author: 2
O'Brien 1
date: 2
27, 2
[eBook 2
```

In addition to basic word counting, extend your code, which must also do the following:

- **It must be case insensitive (see lower() in Python)**
- **It must ignore all punctuation (see, for example, translate() in Python)**
- **It must ignore stop words (see filter() in Spark)**
- **The output must be sorted by count in descending order (see sortBy() in Spark)**

To accomplish this, you will use a combination of basic Python and RDD operations in PySpark or Scala. The following programming guide goes over basics of getting started with Spark and should contain everything you need to complete this part of the homework: https://spark.apache.org/docs/latest/rdd-programming-guide.html

Analysis

Answer the following questions based on your WordCount application. [2X10=20 marks] 1. 1. In the PySpark REPL, run your extended word count program on a single text file.

a. **What are the 25 most common words? Include a screenshot of the program output to back up your claim.**

```
[10]: top_25=sorted_by_count_rdd.take(25)
```

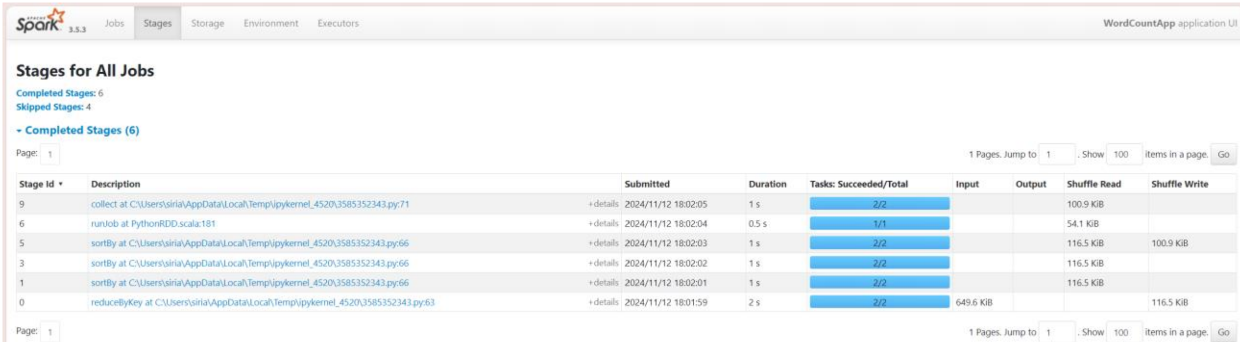
The 25 most common words and program output are shown below,

```
[12]: # Print the first 10 results to verify
      print("Top 25 Words:")
      for word, count in top_25:
          print(f'{word}: {count}')
```

```
Top 25 Words:
one: 452
said: 424
would: 420
mrs: 397
grace: 296
like: 270
mr: 262
never: 252
man: 241
much: 239
know: 217
little: 214
time: 212
walter: 211
well: 201
say: 197
could: 193
see: 193
young: 179
miss: 179
mordaunt: 177
think: 174
project: 166
mother: 162
dont: 160
```

b. **How many stages is execution broken up into? Explain why. Include a screenshot of the DAG visualization from Spark's WebUI to back-up your claim.**

There are 6 complete stages created after executing extended word count code as shown in below screenshot,



Spark 3.5.3 Jobs Stages Storage Environment Executors WordCountApp application UI

Stages for All Jobs

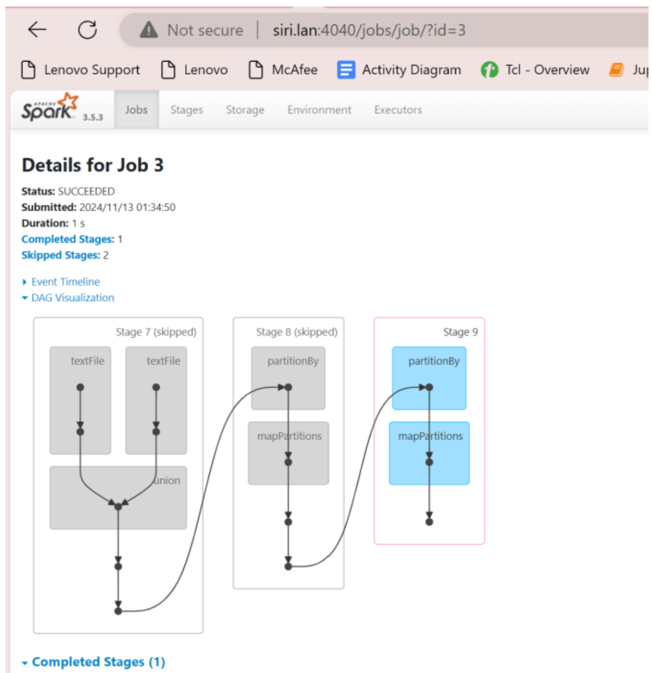
Completed Stages: 6
Skipped Stages: 4

Completed Stages (6)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

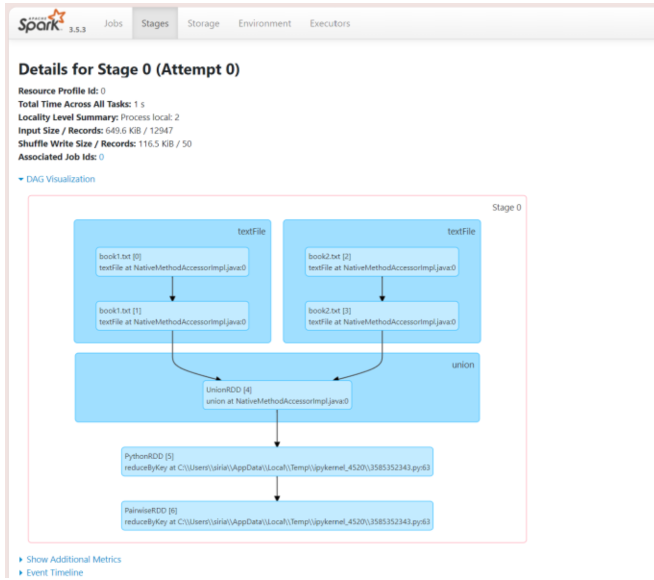
Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	collect at C:\Users\siria\AppData\Local\Temp\ipykernel_4520\3585352343.py:71	+details	2024/11/12 18:02:05	1 s	2/2			100.9 KiB	
6	runJob at PythonRDD.scala:181	+details	2024/11/12 18:02:04	0.5 s	1/1			54.1 KiB	
5	sortBy at C:\Users\siria\AppData\Local\Temp\ipykernel_4520\3585352343.py:66	+details	2024/11/12 18:02:03	1 s	2/2			116.5 KiB	100.9 KiB
3	sortBy at C:\Users\siria\AppData\Local\Temp\ipykernel_4520\3585352343.py:66	+details	2024/11/12 18:02:02	1 s	2/2			116.5 KiB	
1	sortBy at C:\Users\siria\AppData\Local\Temp\ipykernel_4520\3585352343.py:66	+details	2024/11/12 18:02:01	1 s	2/2			116.5 KiB	
0	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_4520\3585352343.py:63	+details	2024/11/12 18:01:59	2 s	2/2	649.6 KiB			116.5 KiB

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

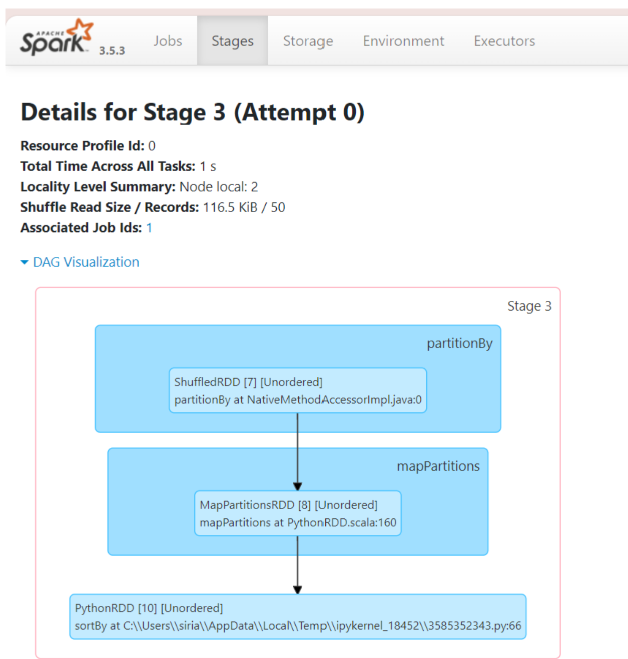


Stages Explanation:

- **Stage Id 0**: This stage involves reading the text files (book1.txt and book2.txt) and performing the union operation to combine the RDDs. Additionally, the initial transformation (flatMap) to split the text into words takes place and filters the combined RDD by removing stop words. Then, maps each filtered word to a key-value pair using map transformation. The reduceByKey transformation aggregates word counts by key, leading to a shuffle operation. DAG visualization of Stage 0 is as shown below,

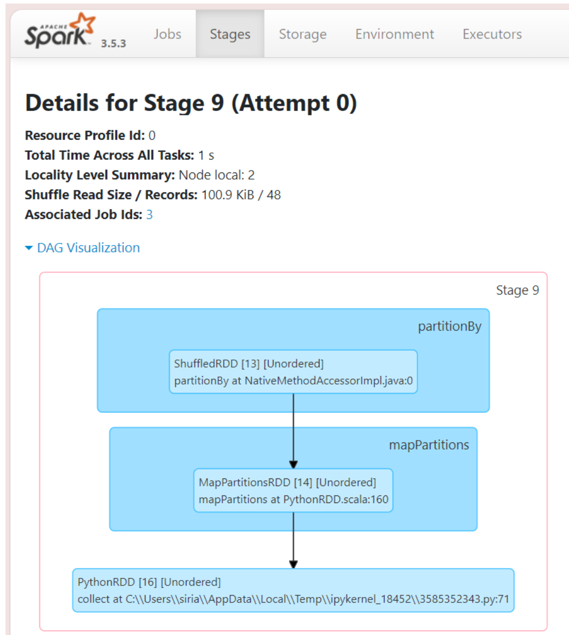


- **Stage Id 1, 3, 5**: Here, sorting is applied to arrange the word counts in descending order using the sortBy transformation, which involves another shuffle. DAG visualization for these Stages is shown below,



- **Stage Id 6**: This is a general stage for running jobs.

- **Stage Id 9**: Finally, the collect action triggers the computation to gather results, finalizing the workflow. DAG visualization for this Stage is shown below,



Shuffle and Partitioning: Data is shuffled between partitions during the key-value pairing and aggregation stages to ensure correct grouping of word counts.

Part-2 Implement and analyze Dijakstra's Shortest Path algorithm.

1. You must write a basic Dijakstra's shortest path algorithm for the two text files that were generated (question2_1.txt and question2_2.txt). Where the first column of each row is the initial node, the second column of each row is the destination, and the third column is the weight associated with the connection.

2. The algorithm should read both the files and compute the shortest path between the first node to all the other nodes and save them in a text file named output_2.txt.

3. Find the nodes with the greatest and the least distance from the starting node and provide a screenshot of the same.

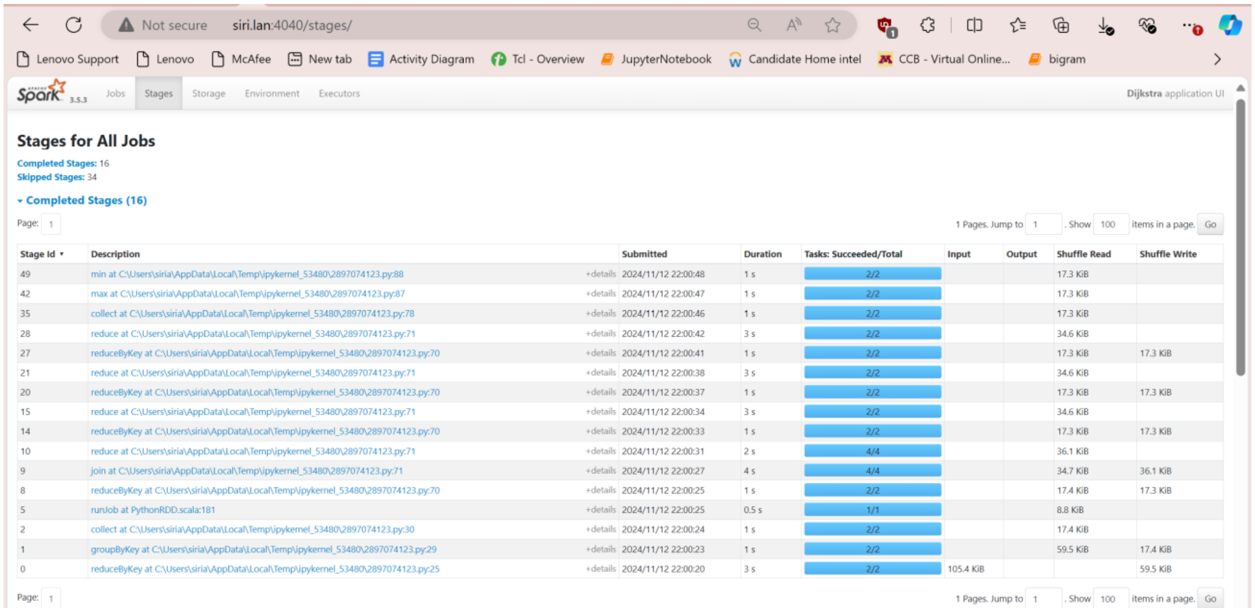
Node with greatest distance from source node 0 to 44 is: 14

Node with least distance from source node 0 to 2 is: 3

```
Adjacency list size: 99
Source node:  0
[(0, 0), (2, 3), (4, 9), (6, 11), (8, 8), (10, 10), (12, 13), (14, 10), (16, 9), (18, 11), (20, 10), (22, 4), (24, 5), (26, 10), (28, 7), (30, 11), (32,
12), (34, 9), (36, 8), (38, 7), (40, 9), (42, 10), (44, 14), (46, 4), (48, 10), (50, 11), (52, 11), (54, 9), (56, 8), (58, 10), (60, 6), (62, 9), (64,
8), (66, 9), (68, 5), (70, 9), (72, 9), (74, 6), (76, 8), (78, 8), (80, 9), (82, 7), (84, 4), (86, 9), (88, 10), (90, 9), (92, 8), (94, 6), (96, 7), (9
8, 7), (1, 12), (3, 8), (5, 8), (7, 9), (9, 6), (11, 12), (13, 10), (15, 13), (17, 13), (19, 11), (21, 5), (23, 12), (25, 8), (27, 9), (29, 8), (31, 1
0), (33, 8), (35, 4), (37, 9), (39, 4), (41, 7), (43, 13), (45, 11), (47, 7), (49, 10), (51, 8), (53, 8), (55, 7), (57, 7), (59, 7), (61, 8), (63, 8),
(65, 11), (67, 9), (69, 6), (71, 9), (73, 6), (75, 4), (77, 8), (79, 6), (81, 9), (83, 10), (85, 6), (87, 6), (89, 8), (91, 9), (93, 7), (95, 5), (97,
8), (99, 9)]
Node with greatest distance from source node  0 to  44 is : 14
Node with least distance from source node  0 to  2 is : 3
```

4. How many stages is execution broken up into? Explain why. Include a screenshot of the DAG visualization from Spark's WebUI to back-up your claim.

The execution broke up into 16 complete stages totally and below is the screenshot of those stages,

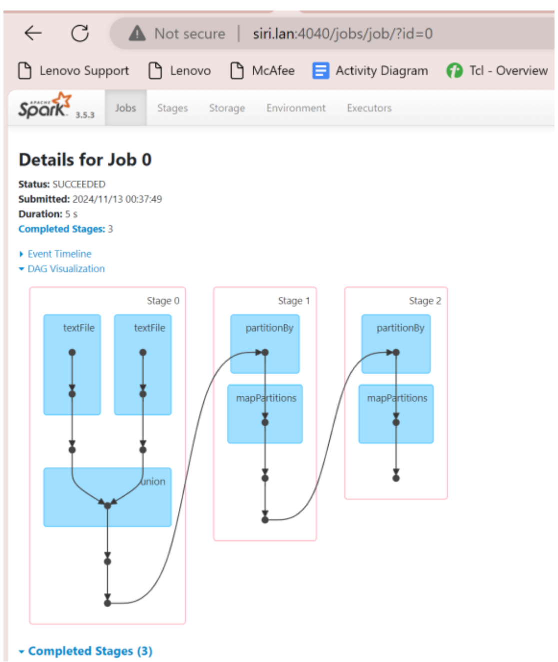


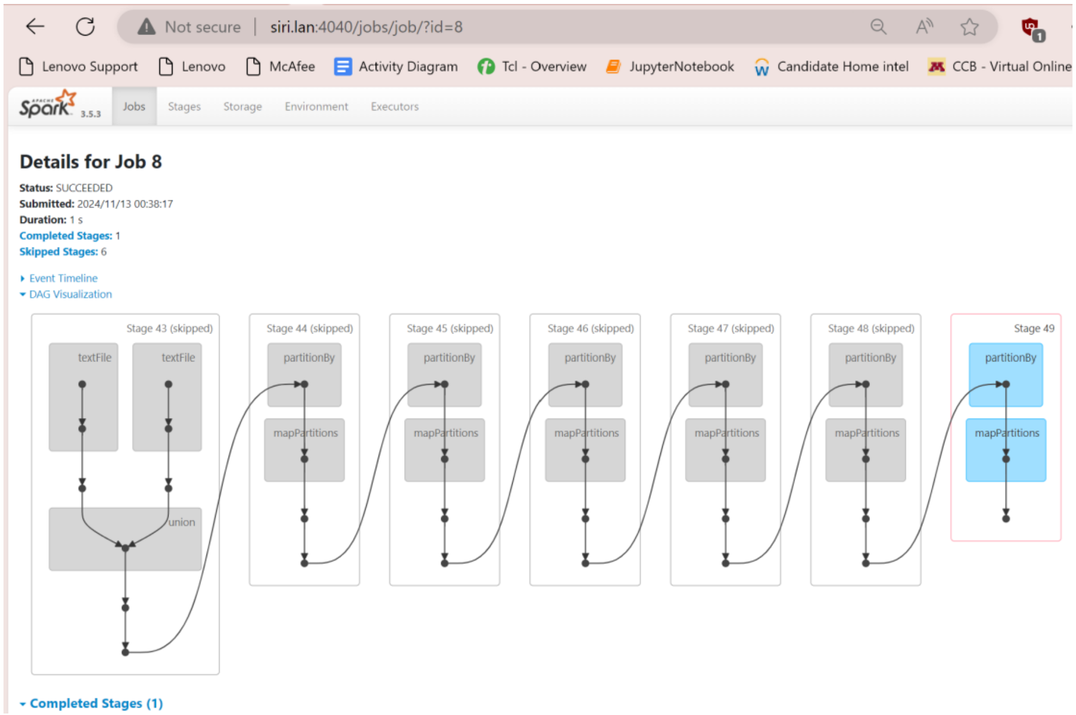
Stages for All Jobs

Completed Stages: 16
Skipped Stages: 34

Completed Stages (16)

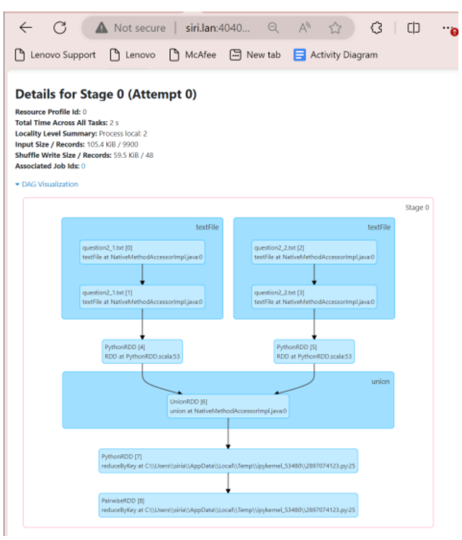
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
49	min at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:88 +details	2024/11/12 22:00:48	1 s	2/2			17.3 KiB	
42	max at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:87 +details	2024/11/12 22:00:47	1 s	2/2			17.3 KiB	
35	collect at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:78 +details	2024/11/12 22:00:46	1 s	2/2			17.3 KiB	
28	reduce at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:71 +details	2024/11/12 22:00:42	3 s	2/2			34.6 KiB	
27	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:70 +details	2024/11/12 22:00:41	1 s	2/2			17.3 KiB	17.3 KiB
21	reduce at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:71 +details	2024/11/12 22:00:38	3 s	2/2			34.6 KiB	
20	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:70 +details	2024/11/12 22:00:37	1 s	2/2			17.3 KiB	17.3 KiB
15	reduce at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:71 +details	2024/11/12 22:00:34	3 s	2/2			34.6 KiB	
14	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:70 +details	2024/11/12 22:00:33	1 s	2/2			17.3 KiB	17.3 KiB
10	reduce at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:71 +details	2024/11/12 22:00:31	2 s	4/4			36.1 KiB	
9	join at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:71 +details	2024/11/12 22:00:27	4 s	4/4			34.7 KiB	36.1 KiB
8	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:70 +details	2024/11/12 22:00:25	1 s	2/2			17.4 KiB	17.3 KiB
5	runJob at PythonRDD.scala:181 +details	2024/11/12 22:00:25	0.5 s	1/1			8.8 KiB	
2	collect at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:30 +details	2024/11/12 22:00:24	1 s	2/2			17.4 KiB	
1	groupByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:29 +details	2024/11/12 22:00:23	1 s	2/2			59.5 KiB	17.4 KiB
0	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_53480\2897074123.py:25 +details	2024/11/12 22:00:20	3 s	2/2	105.4 KiB			59.5 KiB



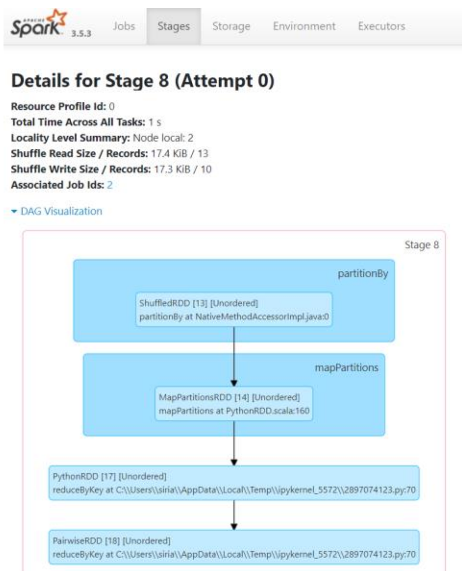


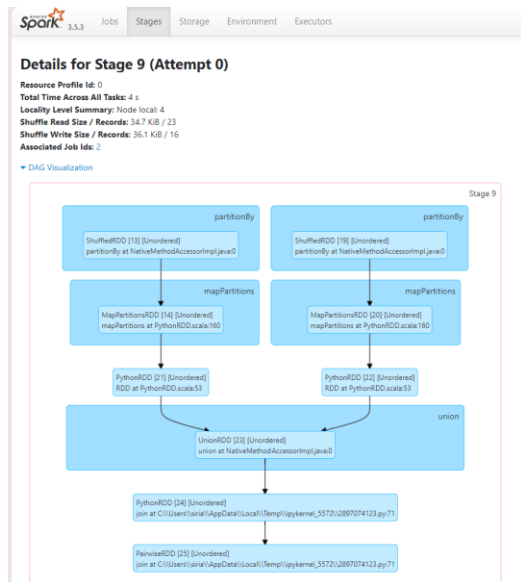
Stages Explanation:

- **Stage Id 0**: This stage involves reading the text files (question2_1.txt and question2_1.txt) and performs the union operation to combine the RDDs and then performs addition of weights if source and destination nodes of two files match. Then formatted the node to (node, (neighbor, weight)) using reduceByKey transformation as shown below,

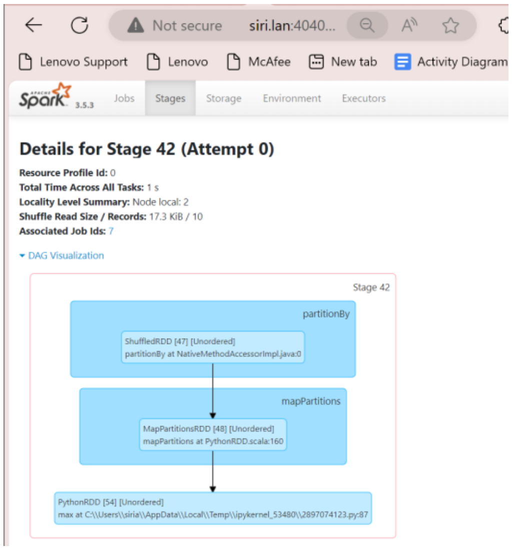


- **Stage Id 1, 2, 5**: Creates an adjacency list representation of a graph by grouping edges by their source node and converting the grouped edges into lists of neighbor-weight pairs and prints the adjacency list using collect().
- **Stage Id 8, 9, 10, 14, 15, 20, 21, 27, 28**: These stages runs Dijkstra's algorithm using MapReduce, where each iteration updates node distances (Map phase) and aggregates the minimum distances (Reduce phase). It repeats until convergence, meaning no further distance updates occur. DAG visualization for these Stages is shown below,





- **Stage Id 35**: Used collect() to collect data to store it in output_2.txt file.
- **Stage Id 42**: Finds the node with the maximum distance
- **Stage Id 49**: finds the node with the minimum non-zero distance (since source node to itself is zero).



Part-3 Implement and analyze Page-rank algorithm.

1. You must write a basic page-rank algorithm considering the text file that is generated (question3.txt). It is a simulated network of 100 pages and its hyperlink . The algorithm should take the network provided and evaluate the page rank for all the webpages or nodes.

2. Find the node with the highest and the lowest page rank and provide a screenshot of the same.

Page with the lowest rank: (31, 0.0028219032914927742)

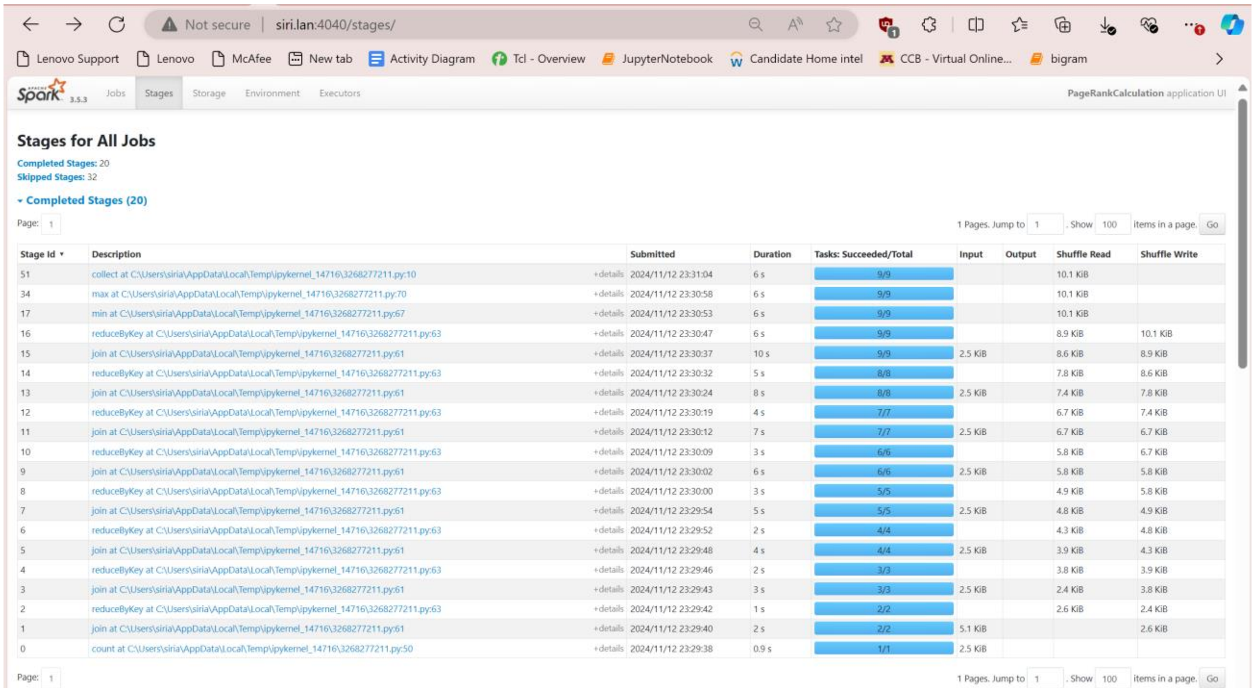
Page with the highest rank: (60, 0.026501353833270597)

Screenshot is shown below,

```
Page with the lowest rank: (31, 0.002821903291492742)
Page with the highest rank: (60, 0.02650135383327059)
```

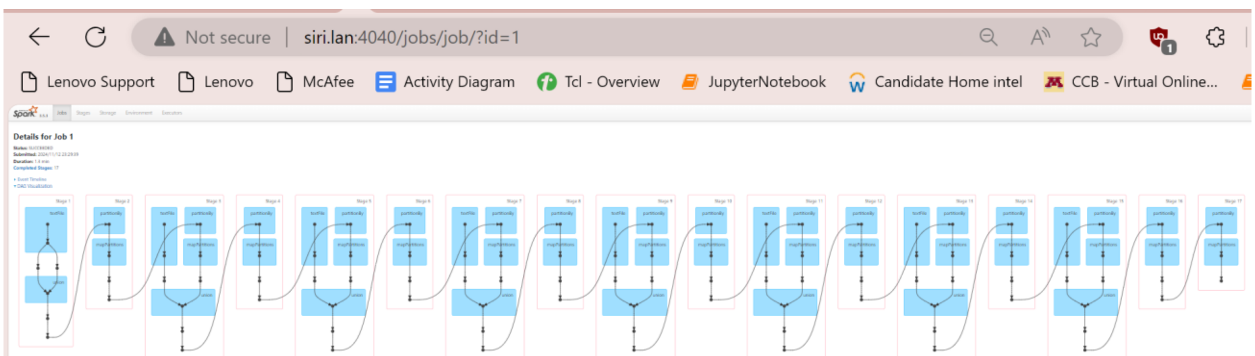
3. How many stages is execution broken up into? Explain why. Include a screenshot of the DAG visualization from Spark's WebUI to back-up your claim

There are in total 20 completed stages



Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
51	collect at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:10 +details	2024/11/12 23:31:04	6 s	9/9			10.1 KiB	
34	max at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:70 +details	2024/11/12 23:30:58	6 s	9/9			10.1 KiB	
17	min at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:67 +details	2024/11/12 23:30:53	6 s	9/9			10.1 KiB	
16	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:63 +details	2024/11/12 23:30:47	6 s	9/9			8.9 KiB	10.1 KiB
15	join at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:61 +details	2024/11/12 23:30:37	10 s	9/9	2.5 KiB		8.6 KiB	8.9 KiB
14	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:63 +details	2024/11/12 23:30:32	5 s	8/8			7.8 KiB	8.6 KiB
13	join at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:61 +details	2024/11/12 23:30:24	8 s	8/8	2.5 KiB		7.4 KiB	7.8 KiB
12	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:63 +details	2024/11/12 23:30:19	4 s	7/7			6.7 KiB	7.4 KiB
11	join at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:61 +details	2024/11/12 23:30:12	7 s	7/7	2.5 KiB		6.7 KiB	6.7 KiB
10	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:63 +details	2024/11/12 23:30:09	3 s	6/6			5.8 KiB	6.7 KiB
9	join at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:61 +details	2024/11/12 23:30:02	6 s	6/6	2.5 KiB		5.8 KiB	5.8 KiB
8	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:63 +details	2024/11/12 23:30:00	3 s	5/5			4.9 KiB	5.8 KiB
7	join at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:61 +details	2024/11/12 23:29:54	5 s	5/5	2.5 KiB		4.8 KiB	4.9 KiB
6	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:63 +details	2024/11/12 23:29:52	2 s	4/4			4.3 KiB	4.8 KiB
5	join at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:61 +details	2024/11/12 23:29:48	4 s	4/4	2.5 KiB		3.9 KiB	4.3 KiB
4	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:63 +details	2024/11/12 23:29:46	2 s	3/3			3.8 KiB	3.9 KiB
3	join at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:61 +details	2024/11/12 23:29:43	3 s	3/3	2.5 KiB		2.4 KiB	3.8 KiB
2	reduceByKey at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:63 +details	2024/11/12 23:29:42	1 s	2/2			2.6 KiB	2.4 KiB
1	join at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:61 +details	2024/11/12 23:29:40	2 s	2/2	5.1 KiB			2.6 KiB
0	count at C:\Users\siria\AppData\Local\Temp\ipykernel_14716\3268277211.py:50 +details	2024/11/12 23:29:38	0.9 s	1/1	2.5 KiB			

DAG Visualization:

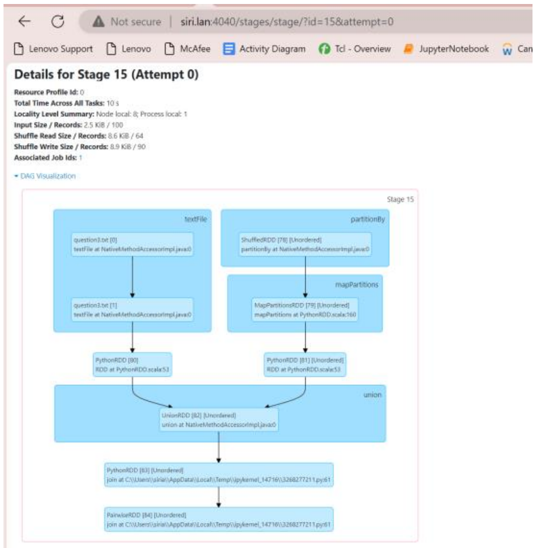


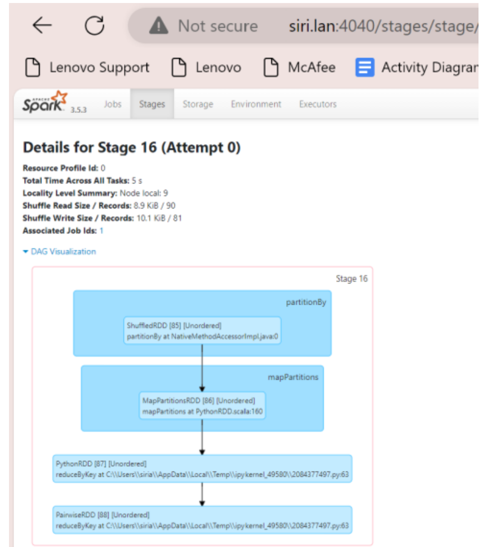
Stages Explanation:

- **Stage Id 0**: This counts the total number of pages in the edges_rdd.
- **Stage Id 1**: The map transformation to assign an initial rank to each page.
- **Stage Id 2 to 16**: PageRank Iterations (one for each iteration) - Each iteration involves two stages:

  contribution computation: The join operation between edges_rdd and page_rank_initial, followed by flatMap to compute contributions. This requires a shuffle.

  rank update: The reduceByKey to sum contributions, followed by mapValues to apply the damping factor. This also requires a shuffle.





- **Stage Id 34 and 51:** The min and max operations on the final ranks to find the nodes with the lowest and highest rank.