Homework #2 – Spark Report

Tharun Teja Mogili - 50559877

Introduction:

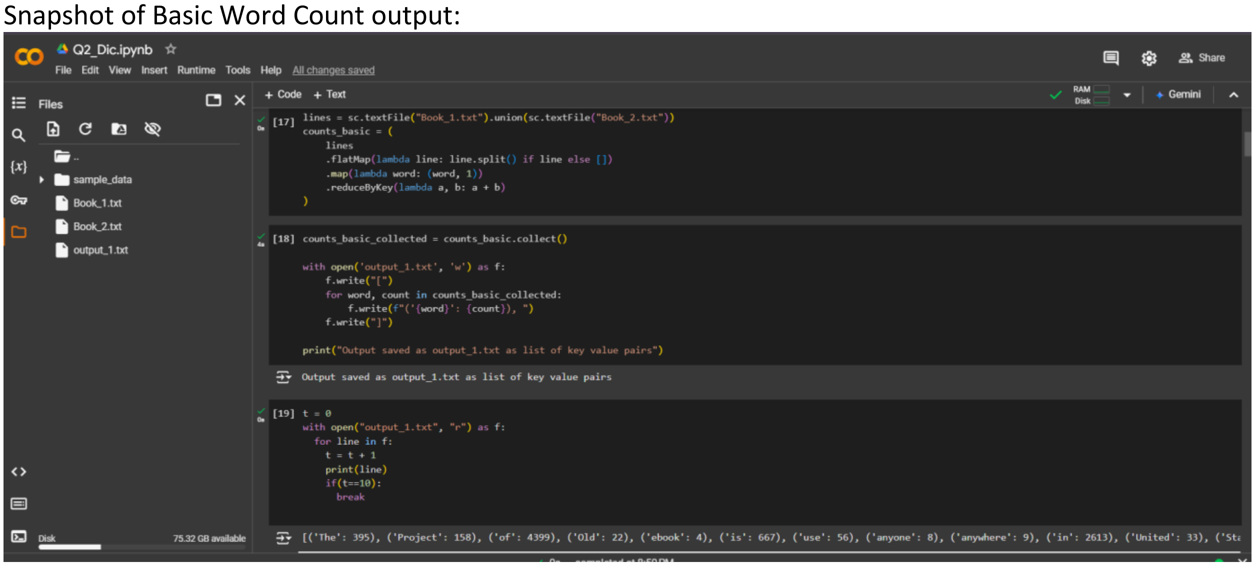
This report delves into the practical application of Spark, focusing on the implementation and analysis of fundamental graph and data processing algorithms. Spark, a powerful distributed computing framework, is widely used for handling big data efficiently.

The report is structured around three essential tasks: In the Word Count task, I applied data-cleaning steps like case normalization, punctuation removal, and stop-word filtering using the NLTK library, then sorted the word counts in descending order. For Dijkstra's algorithm, I calculated the shortest path between nodes, identifying the ones with the longest and shortest distances from a starting point. The final task used the PageRank algorithm on a simulated network of webpages, helping me analyze the highest- and lowest-ranked pages based on link structure. Throughout, I used the Ngrok library to visualize data stages in Spark's Directed Acyclic Graph (DAG).

Part -1 Implement and analyze word count.

Basic Word Count Implementation:
The code snippet reads two text files, "Book_1.txt" and "Book_2.txt," using Spark's textFile() function and combines them using the union() method. Each line of text is split into words, and a key-value pair is created for each word, initializing the count as 1. The reduceByKey() function then aggregates these counts to produce the total occurrences for each word. The results are collected and written into "output_1.txt" in a key-value format, where each entry is represented as a tuple ('word': count). This output is saved for further analysis.

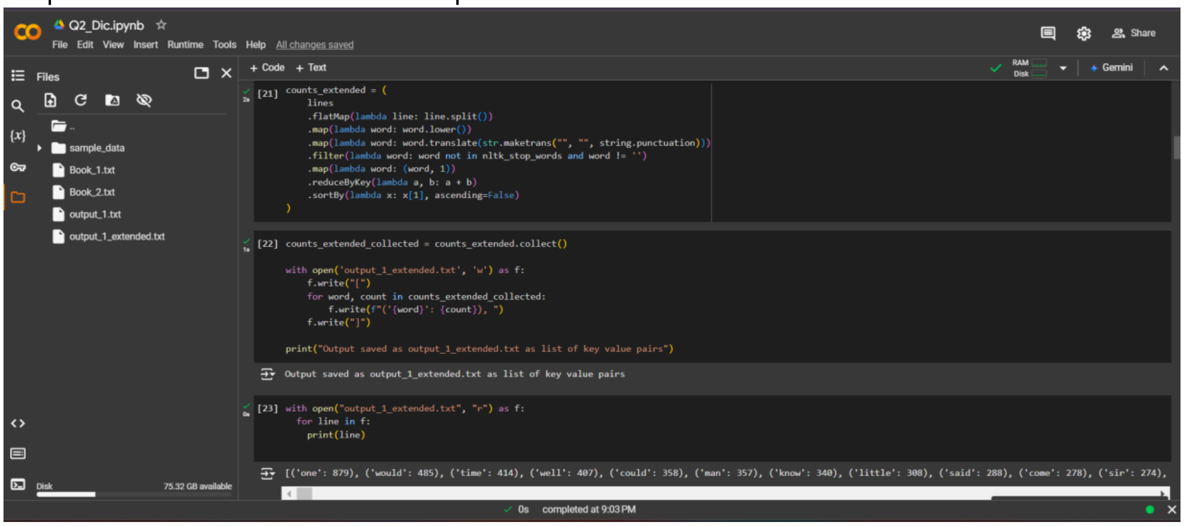
Snapshot of Basic Word Count output:



Extended Word Count Implementation:

The extended code enhances the basic word count by making it case insensitive, removing punctuation, filtering out stop words using NLTK, and sorting the output by word frequency in descending order. Words are first converted to lowercase to ensure uniformity. Punctuation is stripped using Python's translate() method, and common stop words from the NLTK library are excluded using the filter() method in PySpark. Finally, the reduceByKey() function aggregates the word counts, and sortBy() is used to order the words by their count, from highest to lowest. The output is saved in "output_1_extended.txt" as a list of key-value pairs.

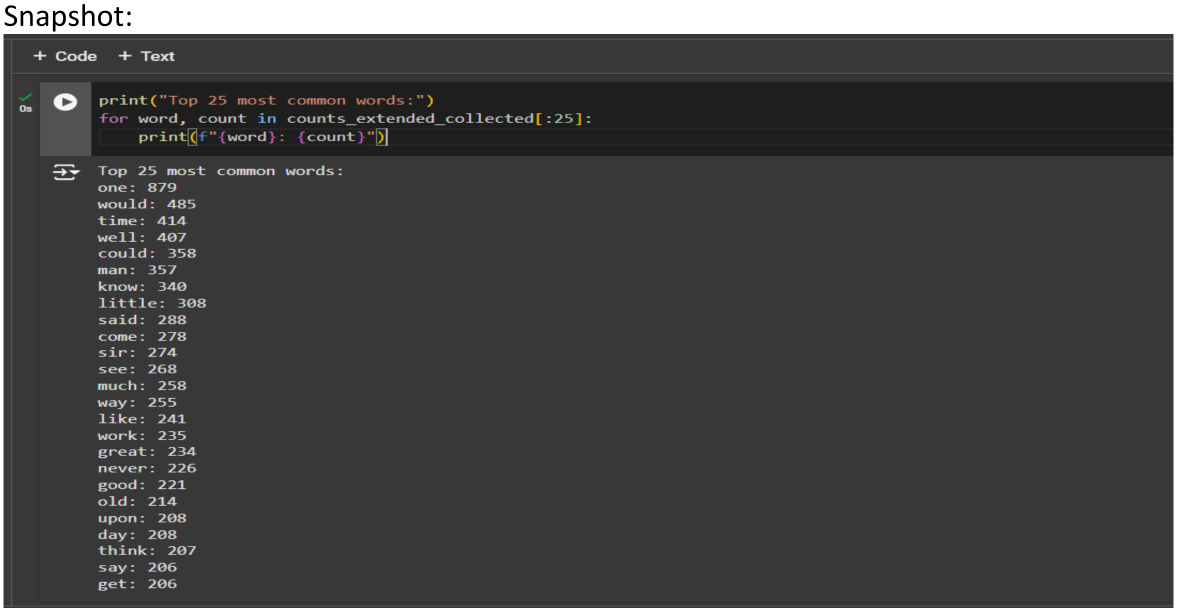
Snapshot of extended Word Count Implementation:



Analysis:

a. The code snippet prints the top 25 most frequent words from the sorted list of word counts. Since the words are already sorted in descending order, it directly displays them in "word: count" format.

Snapshot:

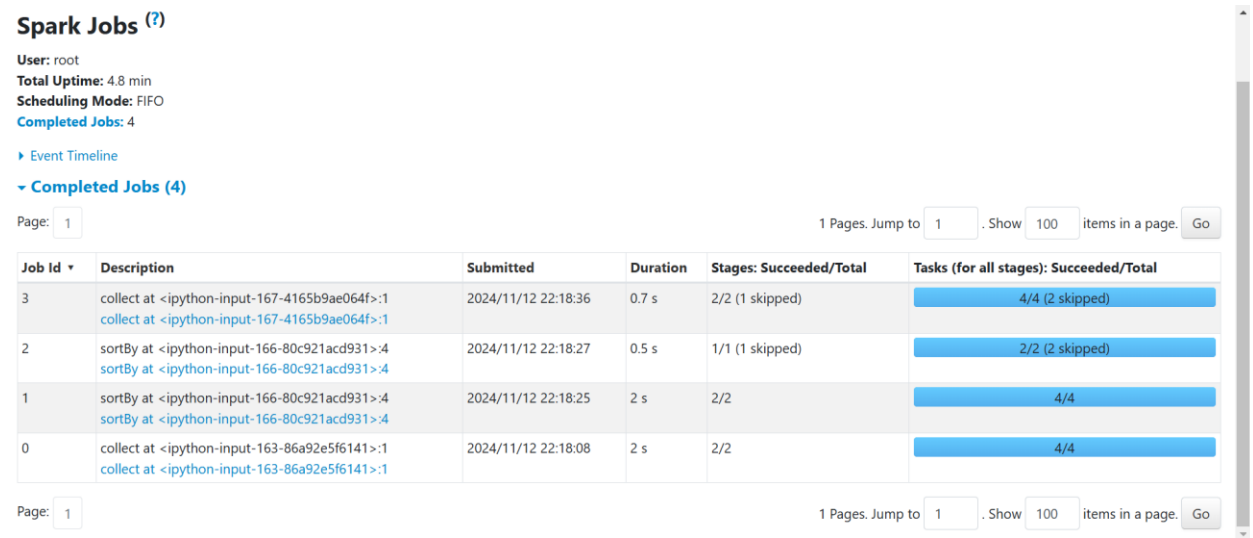


b. Spark breaks down jobs into multiple stages based on wide transformations like reduceByKey, sortBy, or partitionBy. These operations require shuffling data between different partitions, which introduces a boundary between stages. Narrow transformations (like map or filter) can be executed within a single stage because they don't require shuffling.

In this case:
- The union operation combines two RDDs (from two text files) in Stage 6.
- The partitionBy and mapPartitions operations are distributed across Stages 7 and 8, indicating that Spark needed to shuffle data between partitions, resulting in separate stages.

Snapshots of DAG visualizations:

1. Number of Jobs



Spark Jobs (?)

User: root
Total Uptime: 4.8 min
Scheduling Mode: FIFO
Completed Jobs: 4

Event Timeline

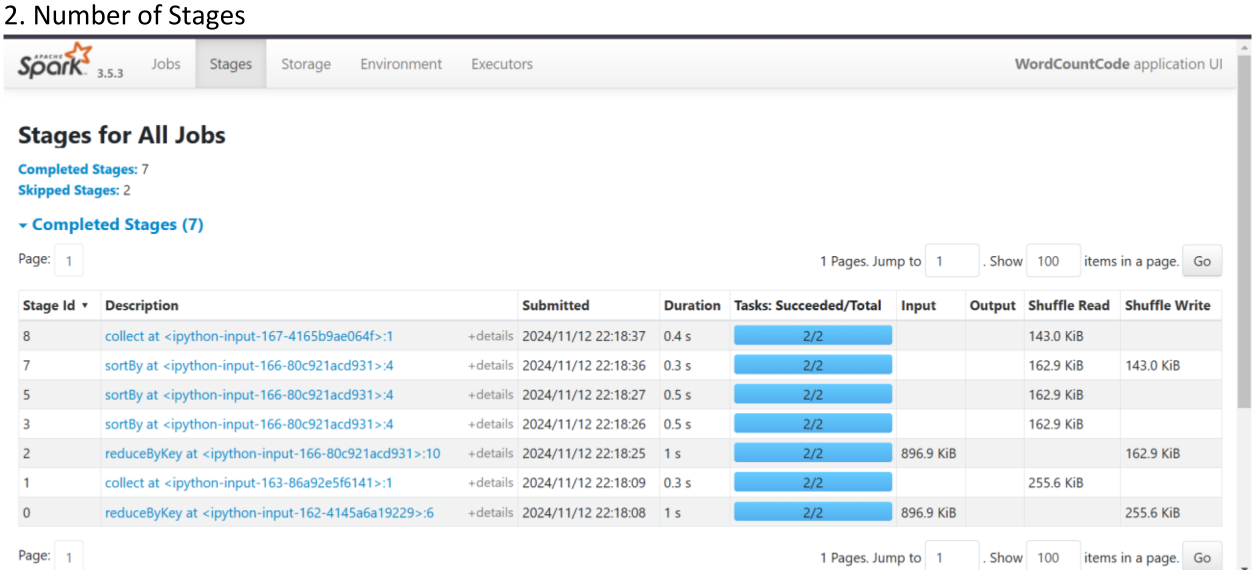
Completed Jobs (4)

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

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	collect at <ipython-input-167-4165b9ae064f>:1 collect at <ipython-input-167-4165b9ae064f>:1	2024/11/12 22:18:36	0.7 s	2/2 (1 skipped)	4/4 (2 skipped)
2	sortBy at <ipython-input-166-80c921acd931>:4 sortBy at <ipython-input-166-80c921acd931>:4	2024/11/12 22:18:27	0.5 s	1/1 (1 skipped)	2/2 (2 skipped)
1	sortBy at <ipython-input-166-80c921acd931>:4 sortBy at <ipython-input-166-80c921acd931>:4	2024/11/12 22:18:25	2 s	2/2	4/4
0	collect at <ipython-input-163-86a92e5f6141>:1 collect at <ipython-input-163-86a92e5f6141>:1	2024/11/12 22:18:08	2 s	2/2	4/4

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

2. Number of Stages



Spark 3.5.3 — Jobs | Stages | Storage | Environment | Executors — **WordCountCode** application UI

Stages for All Jobs

Completed Stages: 7
Skipped Stages: 2

Completed Stages (7)

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

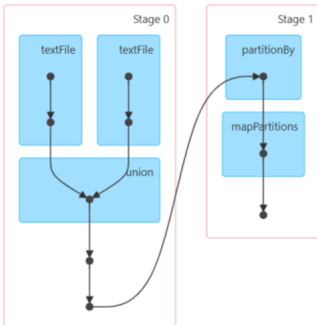
Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
8	collect at <ipython-input-167-4165b9ae064f>:1	+details	2024/11/12 22:18:37	0.4 s	2/2			143.0 KiB	
7	sortBy at <ipython-input-166-80c921acd931>:4	+details	2024/11/12 22:18:36	0.3 s	2/2			162.9 KiB	143.0 KiB
5	sortBy at <ipython-input-166-80c921acd931>:4	+details	2024/11/12 22:18:27	0.5 s	2/2			162.9 KiB	
3	sortBy at <ipython-input-166-80c921acd931>:4	+details	2024/11/12 22:18:26	0.5 s	2/2			162.9 KiB	
2	reduceByKey at <ipython-input-166-80c921acd931>:10	+details	2024/11/12 22:18:25	1 s	2/2	896.9 KiB			162.9 KiB
1	collect at <ipython-input-163-86a92e5f6141>:1	+details	2024/11/12 22:18:09	0.3 s	2/2			255.6 KiB	
0	reduceByKey at <ipython-input-162-4145a6a19229>:6	+details	2024/11/12 22:18:08	1 s	2/2	896.9 KiB			255.6 KiB

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

Details for Job 0

Status: SUCCEEDED
Submitted: 2024/11/12 22:18:08
Duration: 2 s
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization



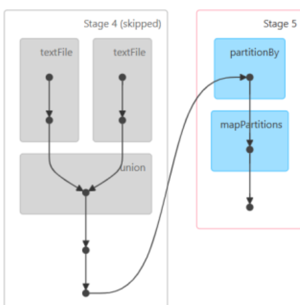
▾ **Completed Stages (2)**

Spark 3.5.3 | Jobs | Stages | Storage | Environment | Executors | **WordCountCode** application UI

Details for Job 2

Status: SUCCEEDED
Submitted: 2024/11/12 22:18:27
Duration: 0.5 s
Completed Stages: 1
Skipped Stages: 1

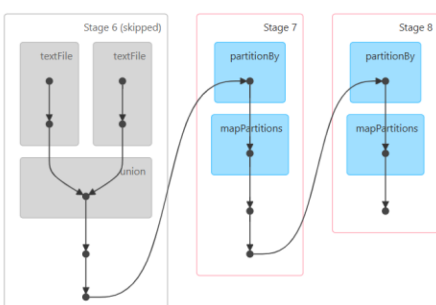
▸ Event Timeline
▾ DAG Visualization



Details for Job 3

Status: SUCCEEDED
Submitted: 2024/11/12 22:18:36
Duration: 0.7 s
Completed Stages: 2
Skipped Stages: 1

▸ Event Timeline
▾ DAG Visualization



▾ **Completed Stages (2)**

Page: 1

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

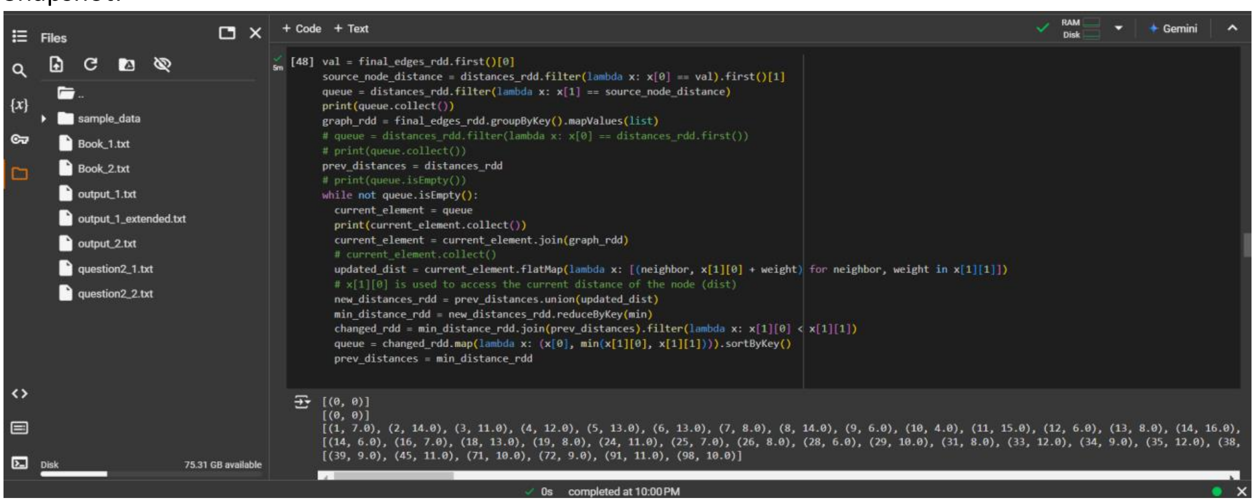
Part-2 Implement and analyze Dijakstra's Shortest Path algorithm.

Dijkstra's shortest path algorithm is a well-known graph traversal technique used to find the shortest path between a source node and all other nodes in a weighted graph. This implementation leverages PySpark to efficiently handle large-scale graph data and compute shortest paths in a distributed manner.

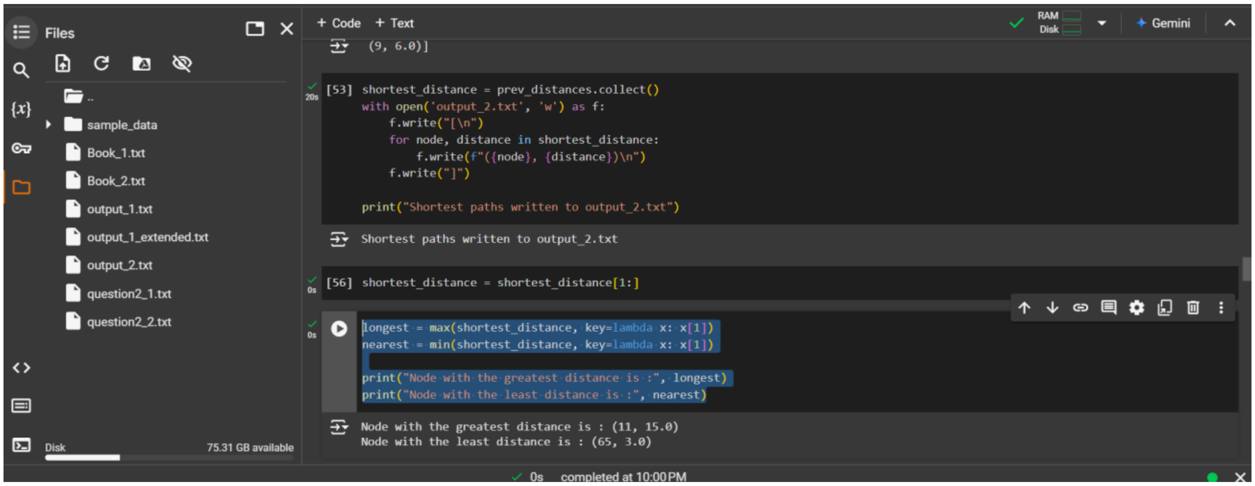
The code begins by reading edge data from two input text files (question2_1.txt and question2_2.txt) and parses the data into key-value pairs representing nodes and their respective edge weights. The edges are then combined using the union operation and added the weights which have same key's and sorted to ensure consistent processing. The algorithm initializes the source node distance to zero and sets all other nodes' distances to infinity. This allows the algorithm to iteratively compute the shortest paths by updating the distances of neighboring nodes as it traverses the graph.

The core of the algorithm works by processing each node in the queue, calculating the shortest distance to its neighbors, and updating the queue as necessary. The queue is dynamically modified to include only those nodes whose distances have changed. Once the algorithm completes, the final distances from the source node to all other nodes are saved in the output file output_2.txt.

Snapshot:



a) The output shows that the node with the greatest distance from the source node is node 11, with a distance of 15.0, while the node with the least distance is node 65, with a distance of 3.0. These results highlight the nodes that are the farthest and closest from the starting point, respectively.



b) The execution is broken up into 492 stages in total, as indicated by the DAG visualization and the Spark WebUI screenshots. Here's a breakdown of how this number is derived:

Completed Stages: 63
Skipped Stages: 430

- The DAG visualization shows multiple stages that are marked as "skipped." These stages are skipped due to Spark's optimization techniques like stage reuse and caching, where Spark avoids redundant recomputation.
- The Spark WebUI confirms that many tasks were skipped across various stages (as seen in the "Tasks" column), further supporting that a large portion of the stages were optimized out during execution.

The stages correspond to the logical execution plan in Spark, and each stage typically represents a set of operations that can be computed in parallel within the available partitions. The shuffling of data between partitions, required by wide transformations, often leads to more stages since it involves network communication and disk I/O.

The DAG visualization from Spark's WebUI provides a detailed view of the job execution plan, showing how Spark distributed tasks into stages and handled data shuffling, which is key to understanding the performance and optimizations in the job execution.

Snapshots:

```
# https://dashboard.ngrok.com/auth
authtoken = "2odaNVr4bHYfaumRhkJVaB01GUS_7fijEi5bsKRs1QQaENdKA"
ngrok.set_auth_token(authtoken)
spark = SparkSession.builder.master("local[*]").appName("DijkstraShortestPathCode").getOrCreate()
public_url = ngrok.connect(addr='http://localhost:4040')
print(f"Spark UI available at: {public_url}")
```

```
WARNING:pyngrok.process.ngrok:t=2024-11-13T03:25:25+0000 lvl=warn msg="can't bind default web address, trying alternatives" obj=we
Spark UI available at: NgrokTunnel: "https://13eb-35-234-0-59.ngrok-free.app" -> "http://localhost:4040"
```

```
[ ] sc.stop()
```

Part-3 Implement and analyze Page-rank algorithm.

Total Number of Stages

Spark 3.5.3 Jobs Stages Storage Environment Executors SQL / DataFrame DijkstraShortestPathCode application UI

Stages for All Jobs

Completed Stages: 63
Skipped Stages: 430

Completed Stages (63)

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
492	collect at <ipython-input-53-821f8cb0e1b0>:1 +details	2024/11/13 02:53:49	20 s	127/127			21.2 KiB	
473	collect at <ipython-input-52-1136a8cb8f18>:1 +details	2024/11/13 02:53:00	20 s	127/127			21.2 KiB	
454	collect at <ipython-input-51-aecaf27d4153>:1 +details	2024/11/13 02:52:01	22 s	127/127			21.2 KiB	
435	collect at <ipython-input-50-bb23bbbd923e>:1 +details	2024/11/13 02:50:56	21 s	127/127			21.2 KiB	
416	runJob at PythonRDD.scala:181 +details	2024/11/13 02:45:21	3 s	13/13			856.0 B	
396	runJob at PythonRDD.scala:181 +details	2024/11/13 02:45:20	0.8 s	4/4			234.0 B	
376	runJob at PythonRDD.scala:181 +details	2024/11/13 02:45:20	0.1 s	1/1			68.0 B	
375	sortByKey at <ipython-input-49-ef4fbbec79c8>:4 +details	2024/11/13 02:45:00	20 s	127/127			21.2 KiB	7.6 KiB
356	sortByKey at <ipython-input-49-ef4fbbec79c8>:4 +details	2024/11/13 02:44:39	20 s	127/127			21.2 KiB	
337	sortByKey at <ipython-input-49-ef4fbbec79c8>:4 +details	2024/11/13 02:44:18	21 s	127/127			21.2 KiB	
318	runJob at PythonRDD.scala:181 +details	2024/11/13 02:40:45	8 s	54/54				
298	runJob at PythonRDD.scala:181 +details	2024/11/13 02:40:30	15 s	100/100			11.9 KiB	
278	runJob at PythonRDD.scala:181 +details	2024/11/13 02:40:27	3 s	20/20			3.2 KiB	

Spark 3.5.3 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **DijkstraShortestPathCode** application UI

Spark Jobs (?)

User: root
Total Uptime: 1.0 h
Scheduling Mode: FIFO
Completed Jobs: 41

▸ Event Timeline

▾ **Completed Jobs (41)**

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

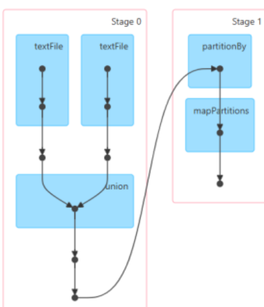
Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
40	collect at <ipython-input-53-821f8cb0e1b0>:1 collect at <ipython-input-53-821f8cb0e1b0>:1	2024/11/13 02:53:49	20 s	1/1 (18 skipped)	127/127 (566 skipped)
39	collect at <ipython-input-52-1136a8cb8f18>:1 collect at <ipython-input-52-1136a8cb8f18>:1	2024/11/13 02:53:00	20 s	1/1 (18 skipped)	127/127 (566 skipped)
38	collect at <ipython-input-51-aecaf27d4153>:1 collect at <ipython-input-51-aecaf27d4153>:1	2024/11/13 02:52:01	22 s	1/1 (18 skipped)	127/127 (566 skipped)
37	collect at <ipython-input-50-bb23bbbd923e>:1 collect at <ipython-input-50-bb23bbbd923e>:1	2024/11/13 02:50:56	21 s	1/1 (18 skipped)	127/127 (566 skipped)
36	runJob at PythonRDD.scala:181 runJob at PythonRDD.scala:181	2024/11/13 02:45:21	3 s	1/1 (19 skipped)	13/13 (693 skipped)
35	runJob at PythonRDD.scala:181 runJob at PythonRDD.scala:181	2024/11/13 02:45:20	0.8 s	1/1 (19 skipped)	4/4 (693 skipped)

Spark 3.5.3 | Jobs | Stages | Storage | Environment | Executors | SQL / DataFrame | **DijkstraShortestPathCode** application UI

Details for Job 0

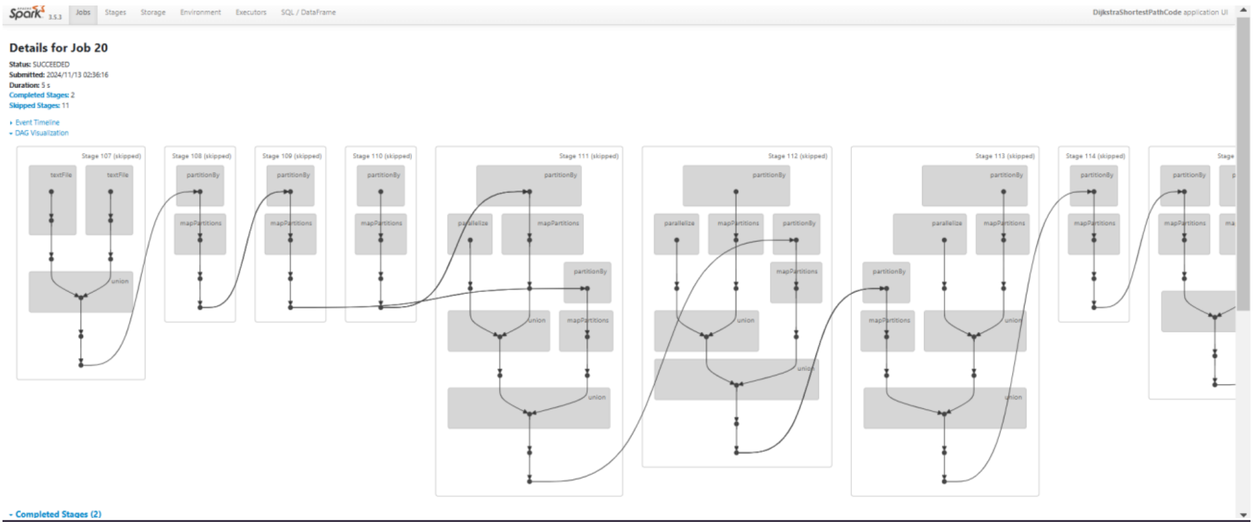
Status: SUCCEEDED
Submitted: 2024/11/13 02:34:49
Duration: 2 s
Completed Stages: 2

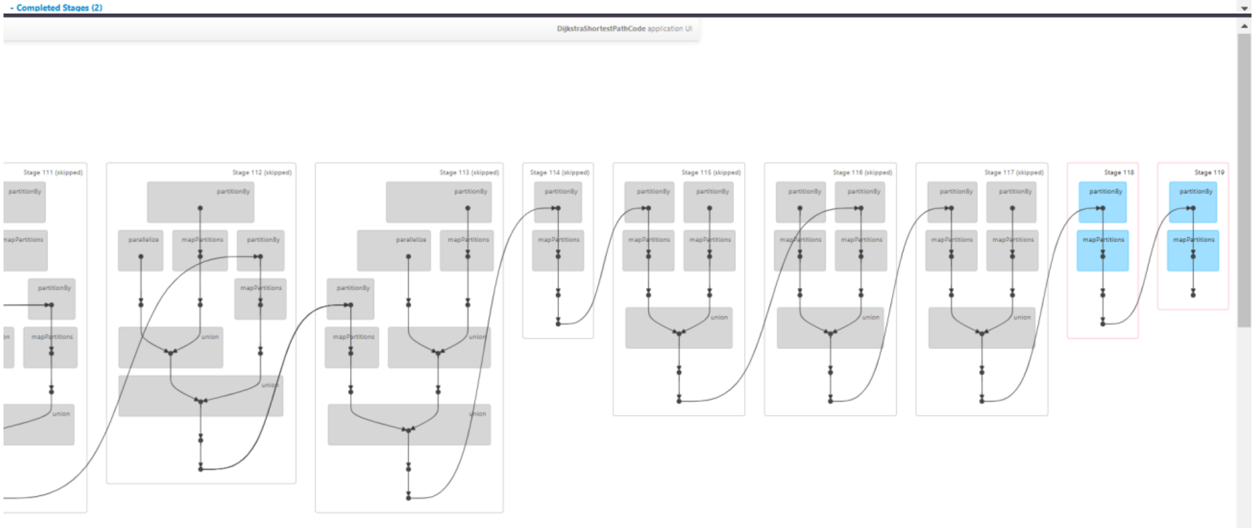
▸ Event Timeline
▾ DAG Visualization



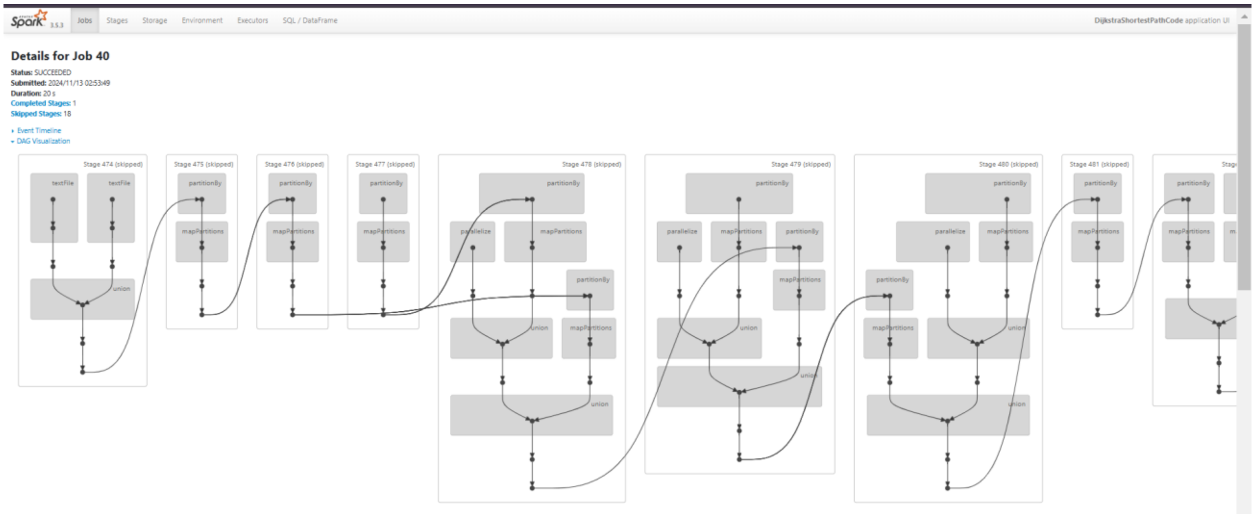
▾ **Completed Stages (2)**

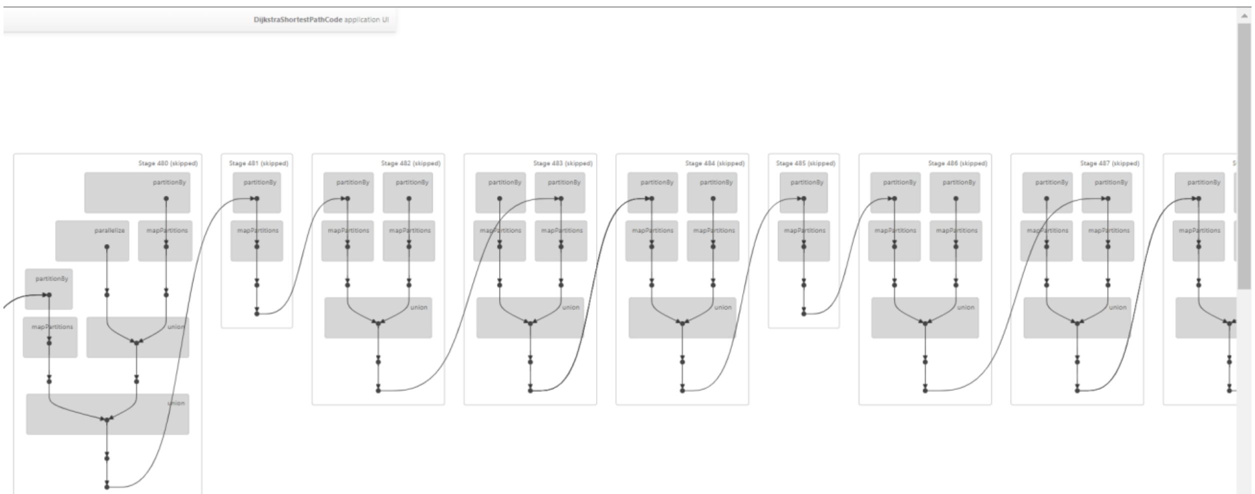
Mid

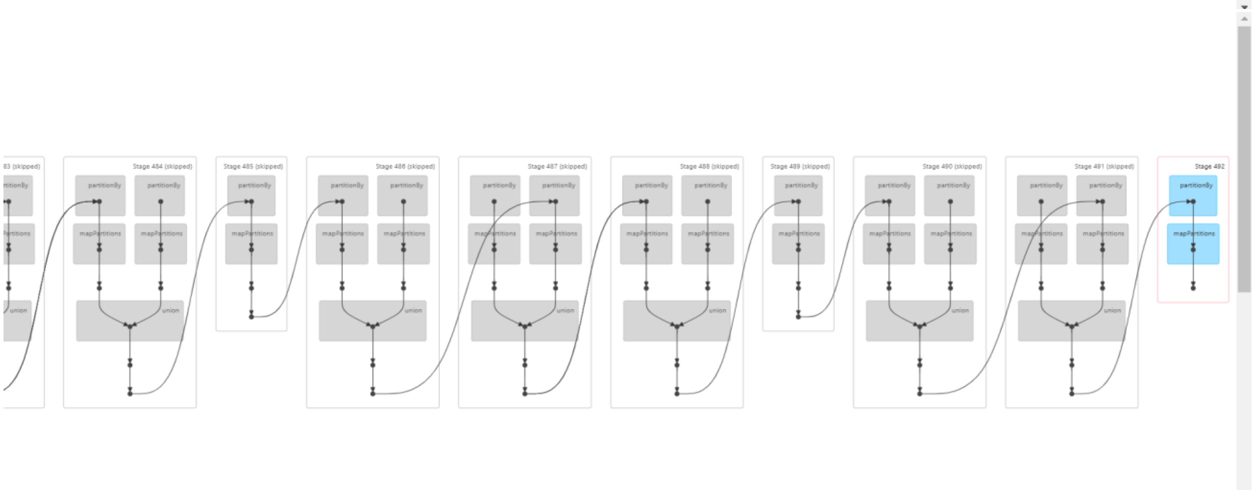

Spark 3.5.3
Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
DijkstraShortestPathCode application UI
Details for Job 20
Status: SUCCEEDED
Submitted: 2024/11/13 02:36:16
Duration: 5 s
Completed Stages: 2
Skipped Stages: 11
Event Timeline
DAG Visualization
Stage 107 (skipped)
Stage 108 (skipped)
Stage 109 (skipped)
Stage 110 (skipped)
Stage 111 (skipped)
Stage 112 (skipped)
Stage 113 (skipped)
Stage 114 (skipped)
Completed Stages (2)


DijkstraShortestPathCode application UI
Stage 111 (skipped)
Stage 112 (skipped)
Stage 113 (skipped)
Stage 114 (skipped)
Stage 115 (skipped)
Stage 116 (skipped)
Stage 117 (skipped)
Stage 118
Stage 119

Last


Spark 3.5.3
Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
DijkstraShortestPathCode application UI
Details for Job 40
Status: SUCCEEDED
Submitted: 2024/11/13 02:53:49
Duration: 20 s
Completed Stages: 1
Skipped Stages: 18
▸ Event Timeline
▾ DAG Visualization

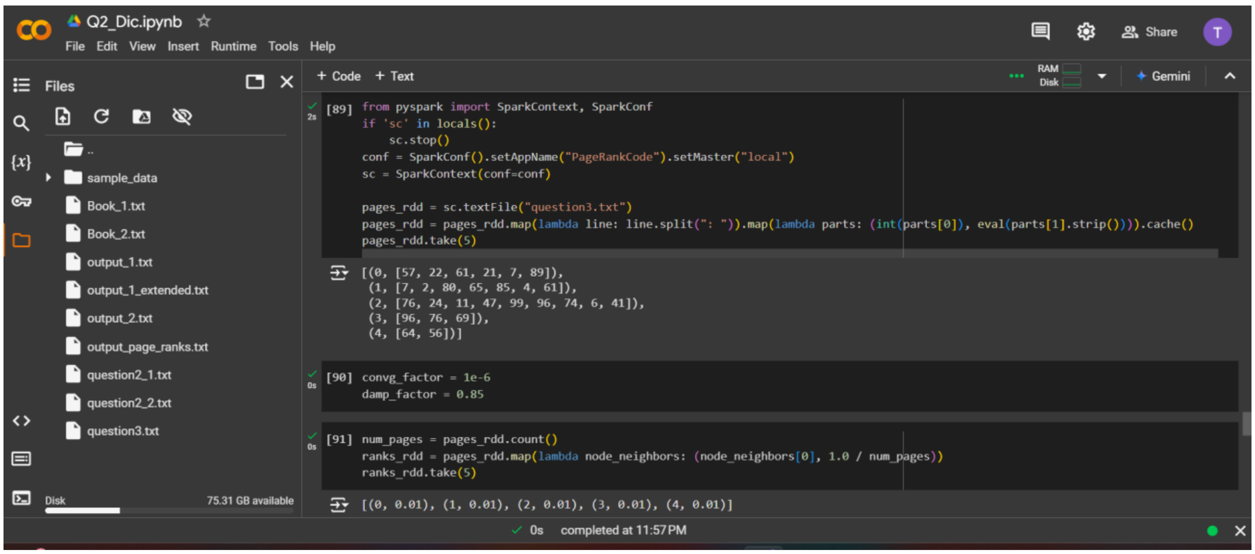

DijkstraShortestPathCode application UI

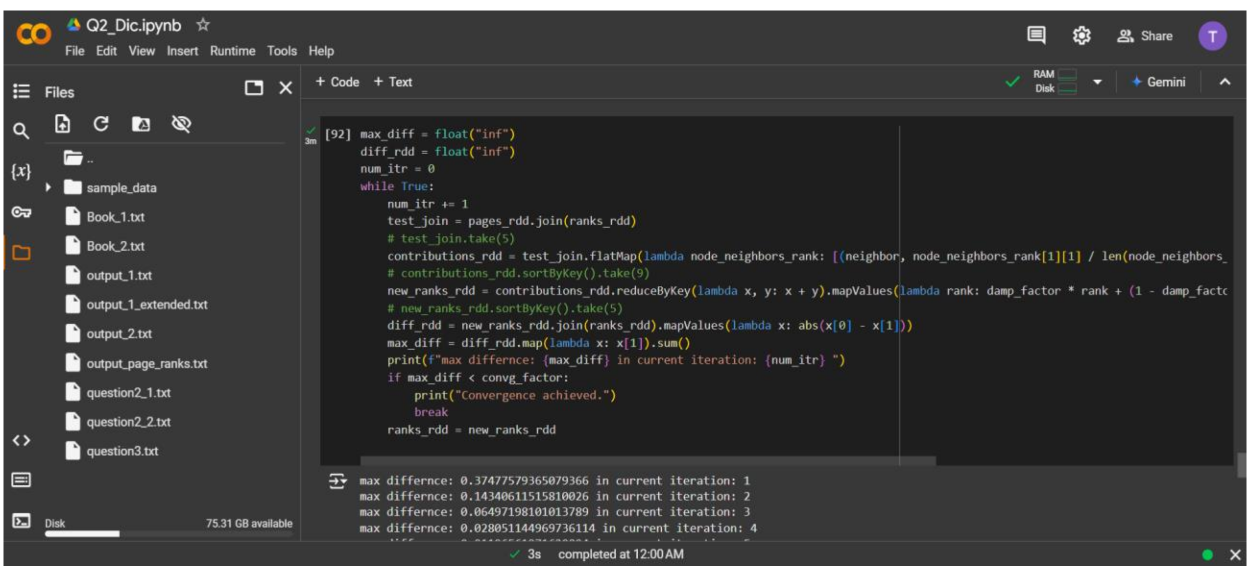


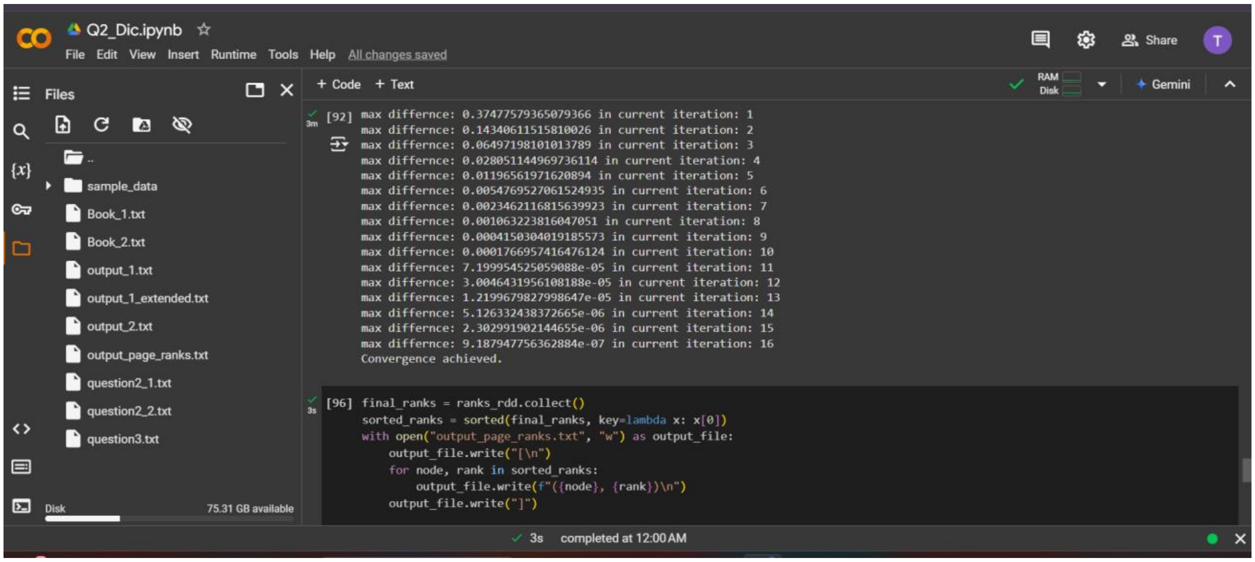
Part-3 Implement and analyze Page-rank algorithm.

The PageRank algorithm is a ranking system used to evaluate the importance of each page in a network, based on the links (hyperlinks) between them. This implementation calculates the PageRank for all webpages in a simulated network of 100 pages. It uses an iterative approach where each page's rank is updated by considering the ranks of the pages linking to it.

The algorithm starts by reading the network structure from the question3.txt file and initializing each page's rank to a uniform value (1/number of pages). It then iterates over the network, propagating rank values based on the damping factor and link structure until the rank differences between iterations fall below a predefined convergence threshold (1e-6). The damping factor (0.85) helps prevent ranks from being distributed too evenly.

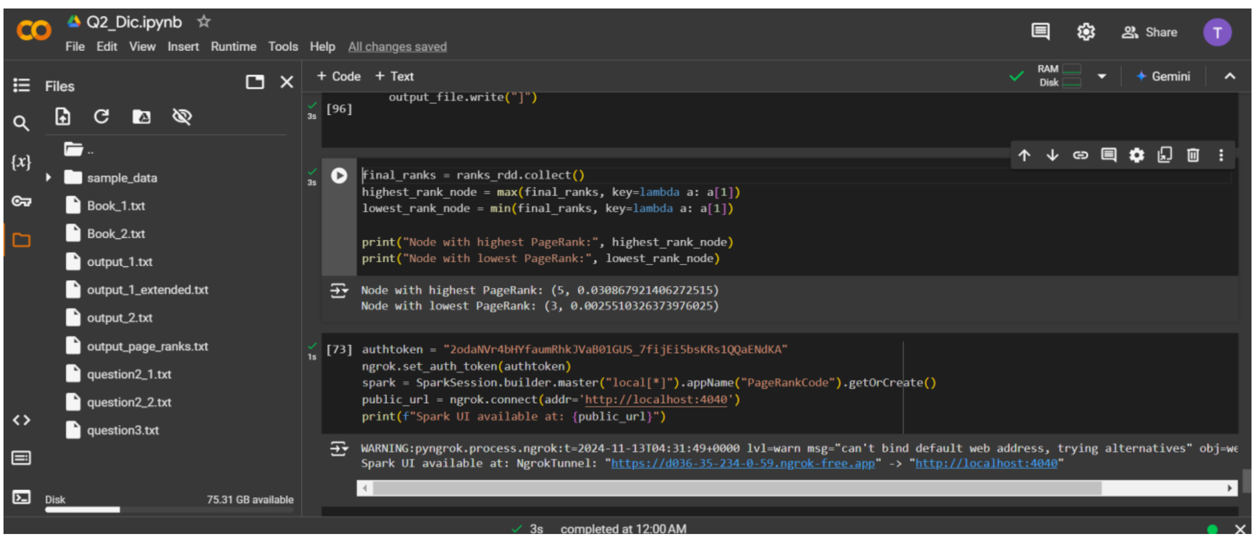




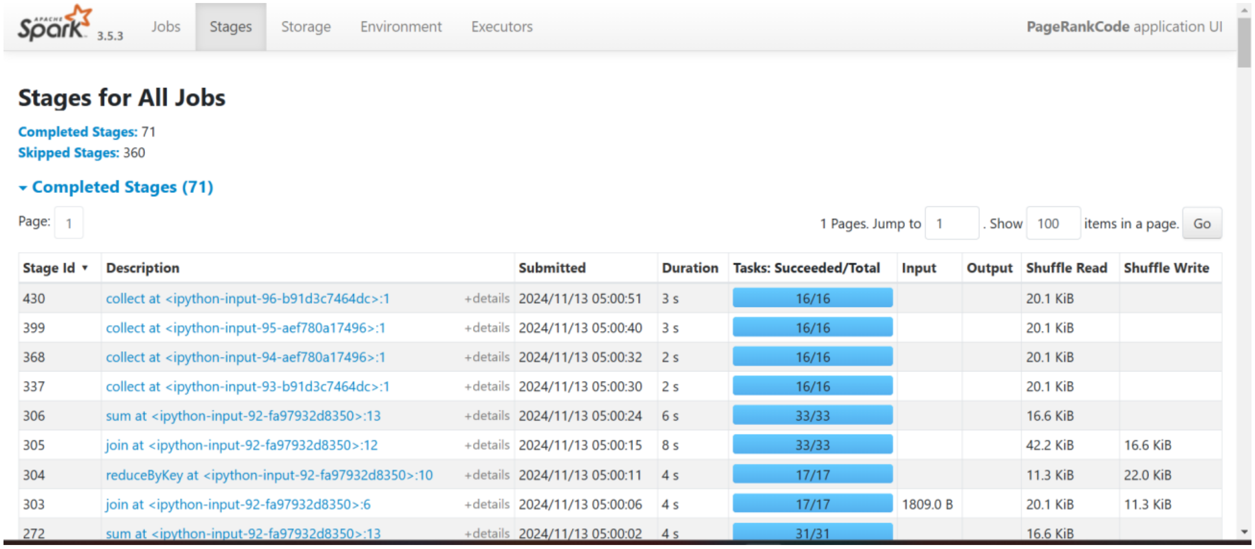


The code prints the iteration number and the maximum difference between PageRank values in each iteration until convergence is reached.

After convergence, the node with the highest and lowest PageRank values is identified.
Highest Rank node is 5 and lowest rank node is 3.



The execution of this PageRank algorithm is broken up into multiple stages across several jobs



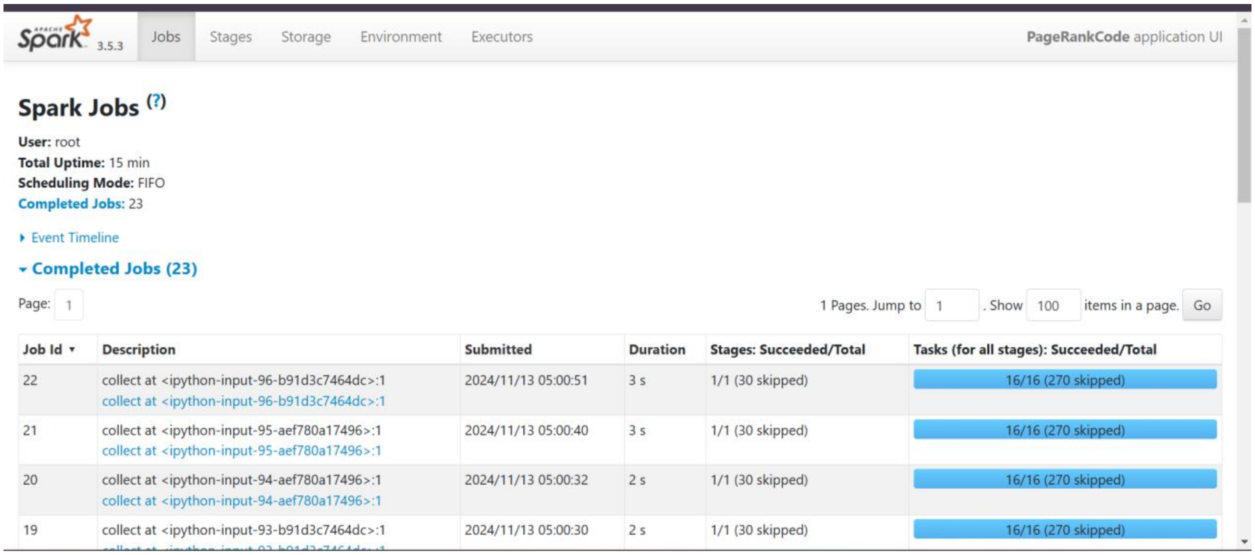
Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
430	collect at <ipython-input-96-b91d3c7464dc>:1	+details	2024/11/13 05:00:51	3 s	16/16			20.1 KiB	
399	collect at <ipython-input-95-aef780a17496>:1	+details	2024/11/13 05:00:40	3 s	16/16			20.1 KiB	
368	collect at <ipython-input-94-aef780a17496>:1	+details	2024/11/13 05:00:32	2 s	16/16			20.1 KiB	
337	collect at <ipython-input-93-b91d3c7464dc>:1	+details	2024/11/13 05:00:30	2 s	16/16			20.1 KiB	
306	sum at <ipython-input-92-fa97932d8350>:13	+details	2024/11/13 05:00:24	6 s	33/33			16.6 KiB	
305	join at <ipython-input-92-fa97932d8350>:12	+details	2024/11/13 05:00:15	8 s	33/33			42.2 KiB	16.6 KiB
304	reduceByKey at <ipython-input-92-fa97932d8350>:10	+details	2024/11/13 05:00:11	4 s	17/17			11.3 KiB	22.0 KiB
303	join at <ipython-input-92-fa97932d8350>:6	+details	2024/11/13 05:00:06	4 s	17/17	1809.0 B		20.1 KiB	11.3 KiB
272	sum at <ipython-input-92-fa97932d8350>:13	+details	2024/11/13 05:00:02	4 s	31/31			16.6 KiB	

The execution of the PageRank algorithm in this Spark application is broken up into 71 completed stages, with 360 stages being skipped. The large number of stages reflects the complex and iterative nature of the PageRank computation. Each stage in the DAG (Directed Acyclic Graph) corresponds to a sequence of transformations or actions applied to the RDDs (Resilient Distributed Datasets).

Reasons for Multiple Stages such as

- Iterative Algorithm: The PageRank algorithm runs in a while loop until convergence. Each iteration creates new stages for the various transformations.
- Multiple Transformations: Within each iteration, there are several transformations like join, flatMap, reduceByKey, and mapValues. Each of these, especially the wide transformations like join and reduceByKey, typically result in new stages.
- Actions Triggering Jobs: Actions like take(5) and count() trigger new jobs, each potentially consisting of multiple stages.
- Caching: The cache() operation on pages_rdd creates an additional stage to persist the data.
- Shuffle Operations: Operations like join and reduceByKey involve data shuffling, which necessitates stage boundaries.

The screenshot of the DAG will illustrate the breakdown of stages, showcasing the transformation sequence and the interdependencies among them.



Spark 3.5.3 | Jobs | Stages | Storage | Environment | Executors | **PageRankCode** application UI

Spark Jobs (?)

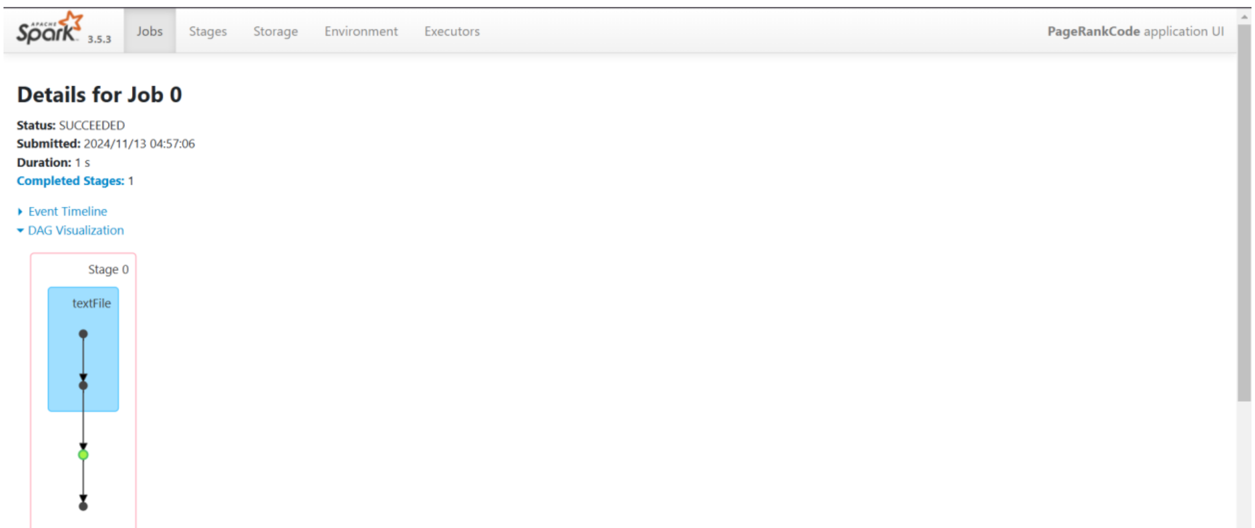
User: root
Total Uptime: 15 min
Scheduling Mode: FIFO
Completed Jobs: 23

▸ Event Timeline

▾ **Completed Jobs (23)**

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

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
22	collect at <ipython-input-96-b91d3c7464dc>:1 collect at <ipython-input-96-b91d3c7464dc>:1	2024/11/13 05:00:51	3 s	1/1 (30 skipped)	16/16 (270 skipped)
21	collect at <ipython-input-95-aef780a17496>:1 collect at <ipython-input-95-aef780a17496>:1	2024/11/13 05:00:40	3 s	1/1 (30 skipped)	16/16 (270 skipped)
20	collect at <ipython-input-94-aef780a17496>:1 collect at <ipython-input-94-aef780a17496>:1	2024/11/13 05:00:32	2 s	1/1 (30 skipped)	16/16 (270 skipped)
19	collect at <ipython-input-93-b91d3c7464dc>:1	2024/11/13 05:00:30	2 s	1/1 (30 skipped)	16/16 (270 skipped)

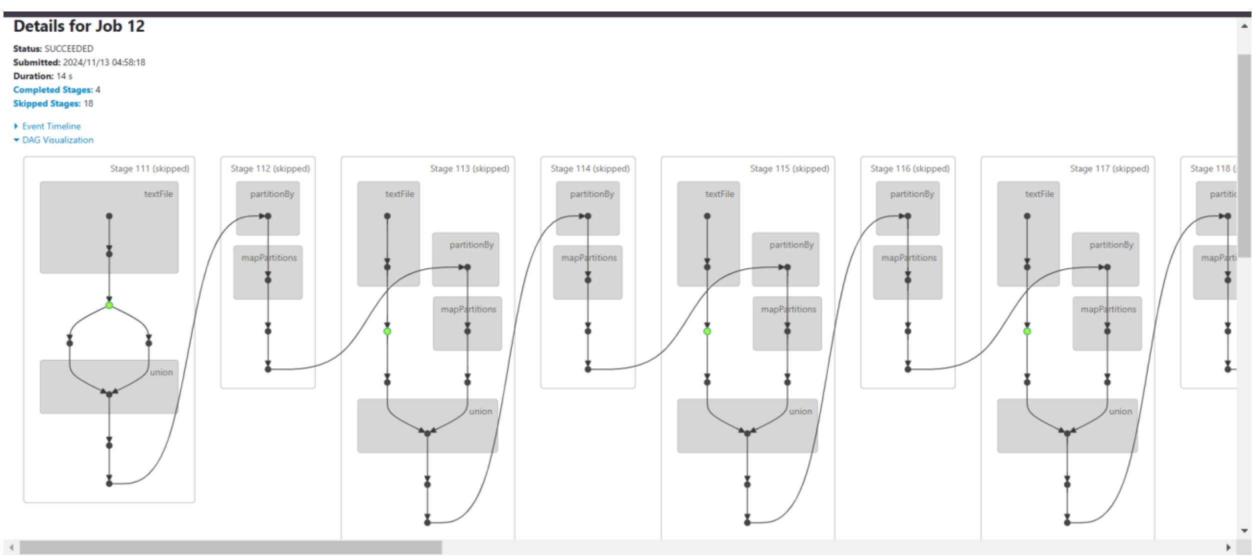


Spark 3.5.3 | Jobs | Stages | Storage | Environment | Executors | **PageRankCode** application UI

Details for Job 0

Status: SUCCEEDED
Submitted: 2024/11/13 04:57:06
Duration: 1 s
Completed Stages: 1

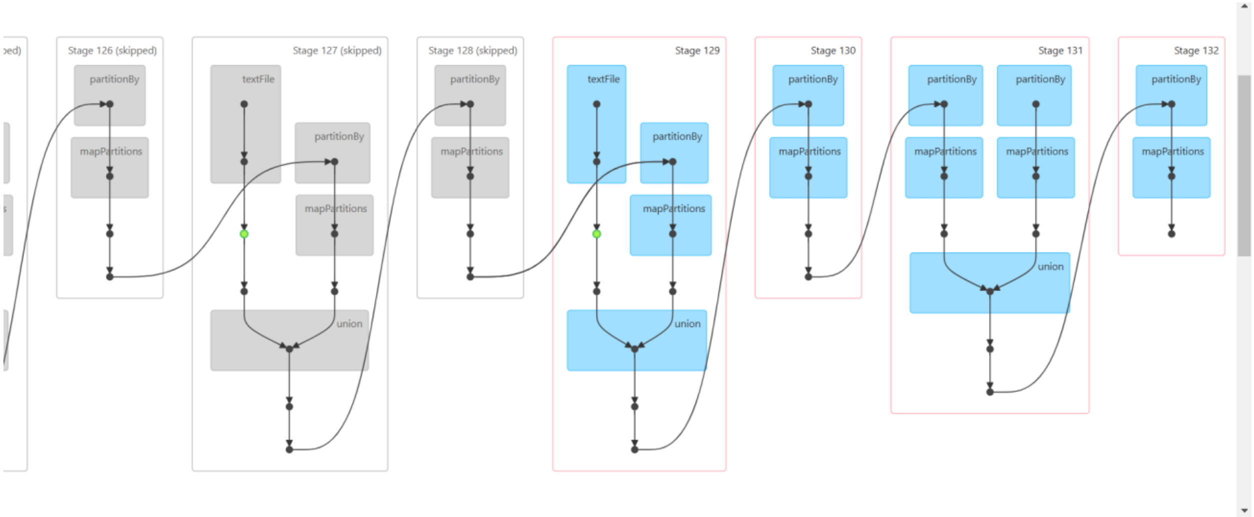
▸ Event Timeline
▾ DAG Visualization

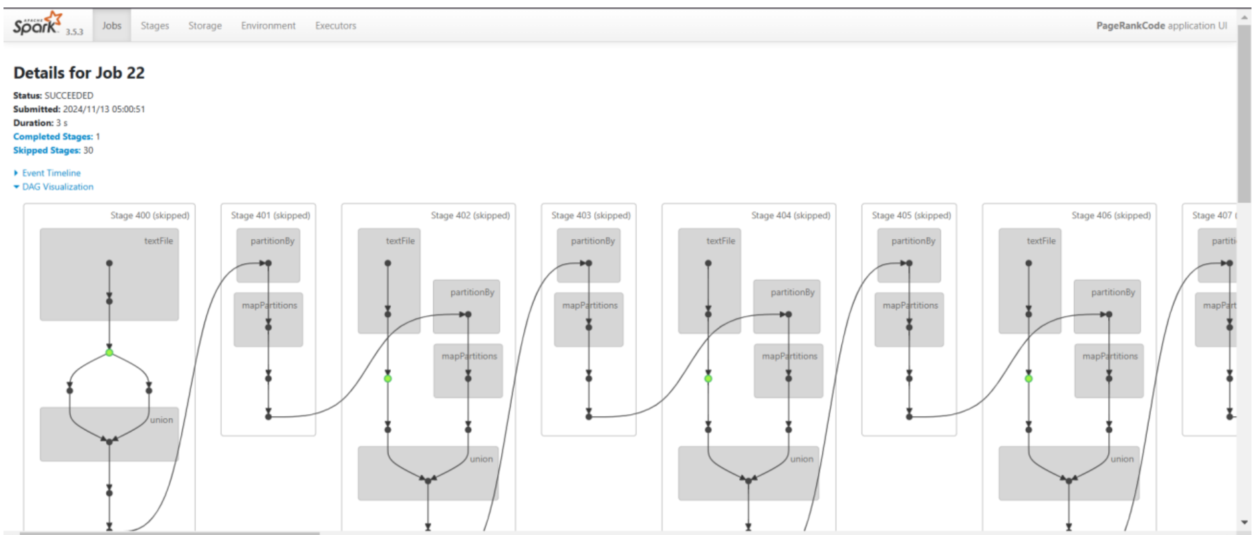


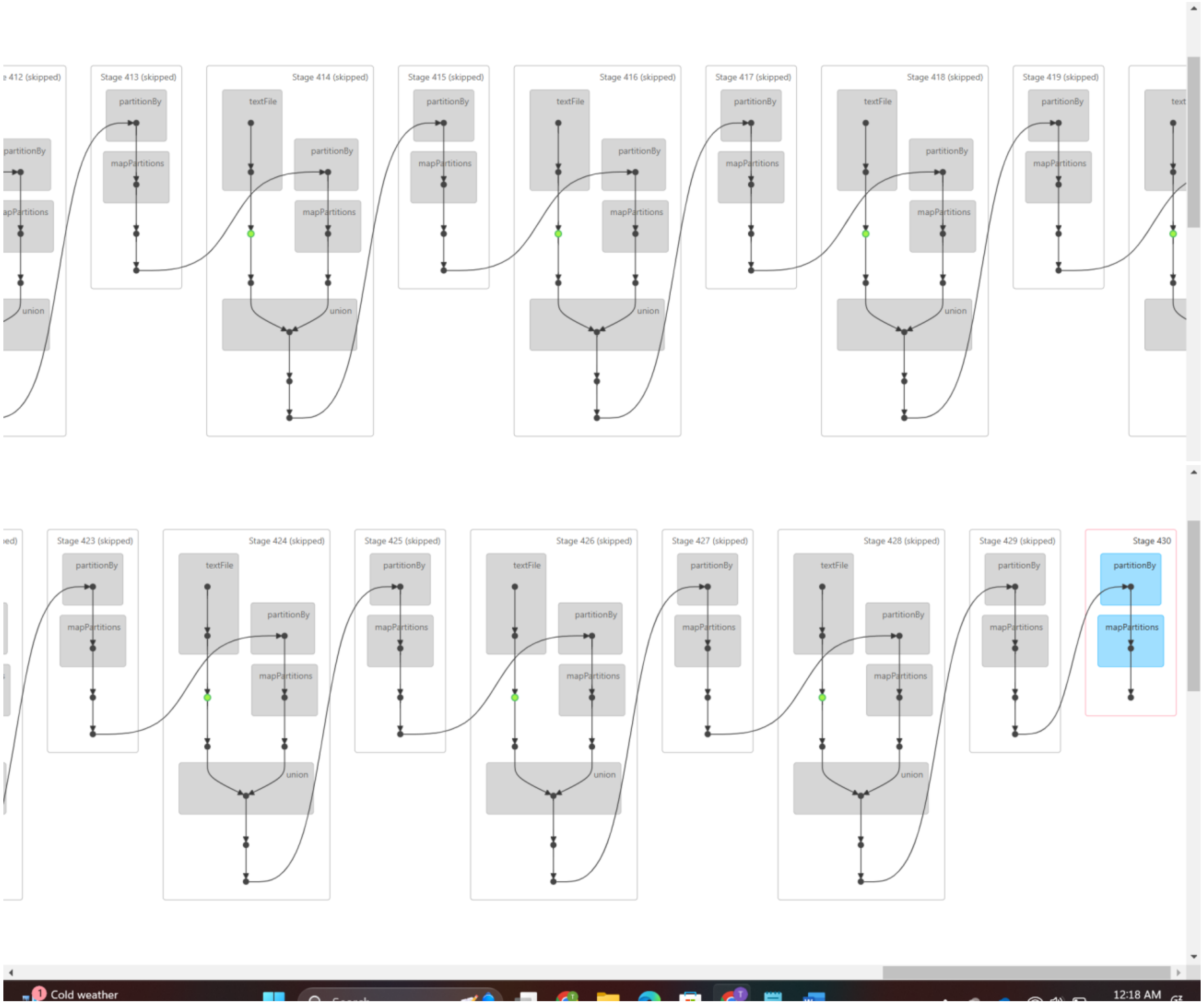
Details for Job 12

Status: SUCCEEDED
Submitted: 2024/11/13 04:58:18
Duration: 14 s
Completed Stages: 4
Skipped Stages: 18

▸ Event Timeline
▾ DAG Visualization


Stage 126 (skipped)
Stage 127 (skipped)
Stage 128 (skipped)
Stage 129
Stage 130
Stage 131
Stage 132
partitionBy
mapPartitions
textFile
union


Spark 3.5.3
Jobs
Stages
Storage
Environment
Executors
PageRankCode application UI
Details for Job 22
Status: SUCCEEDED
Submitted: 2024/11/13 05:00:51
Duration: 3 s
Completed Stages: 1
Skipped Stages: 30
Event Timeline
DAG Visualization
Stage 400 (skipped)
Stage 401 (skipped)
Stage 402 (skipped)
Stage 403 (skipped)
Stage 404 (skipped)
Stage 405 (skipped)
Stage 406 (skipped)
textFile
partitionBy
mapPartitions
union



Stage 413 (skipped)
partitionBy
mapPartitions
Stage 414 (skipped)
textFile
partitionBy
mapPartitions
union
Stage 415 (skipped)
partitionBy
mapPartitions
Stage 416 (skipped)
textFile
partitionBy
mapPartitions
union
Stage 417 (skipped)
partitionBy
mapPartitions
Stage 418 (skipped)
textFile
partitionBy
mapPartitions
union
Stage 419 (skipped)
partitionBy
mapPartitions
Stage 423 (skipped)
partitionBy
mapPartitions
Stage 424 (skipped)
textFile
partitionBy
mapPartitions
union
Stage 425 (skipped)
partitionBy
mapPartitions
Stage 426 (skipped)
textFile
partitionBy
mapPartitions
union
Stage 427 (skipped)
partitionBy
mapPartitions
Stage 428 (skipped)
textFile
partitionBy
mapPartitions
union
Stage 429 (skipped)
partitionBy
mapPartitions
Stage 430
partitionBy
mapPartitions
Cold weather
12:18 AM