

## CSE 4/587 Assignment 2 - Spring 2023

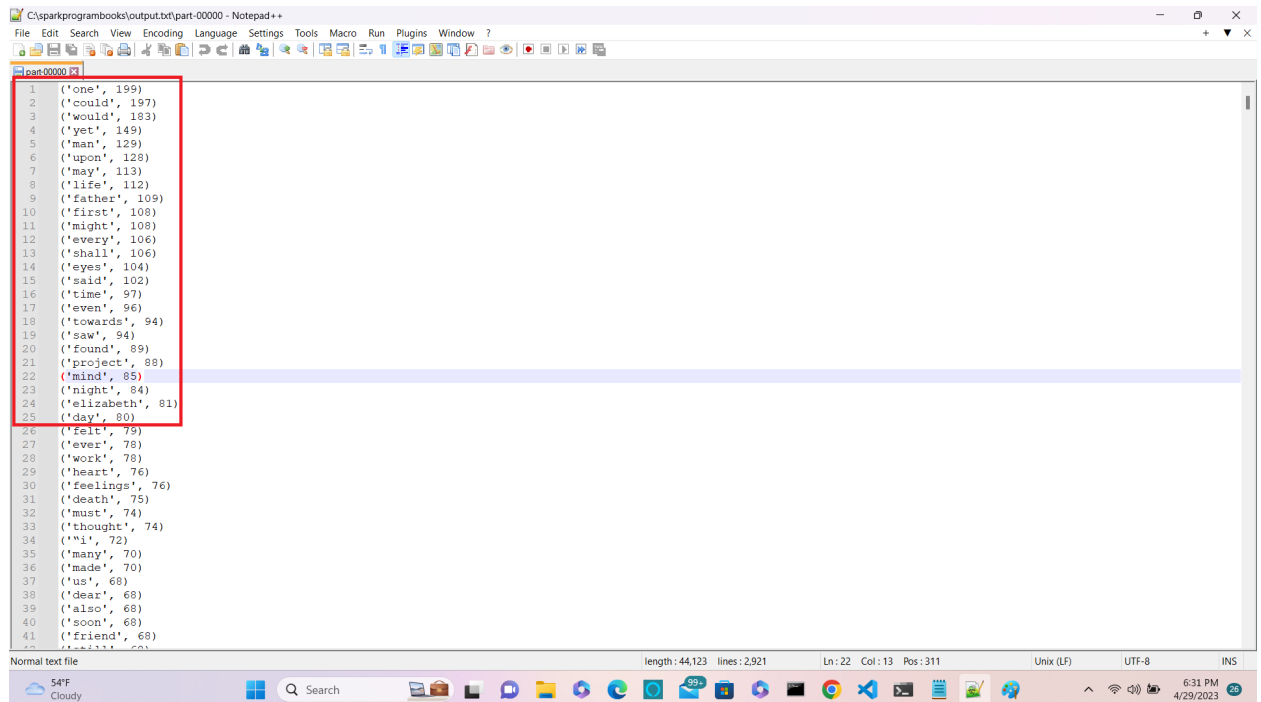
Homework #2 - Spark  
Manikanta Kalyan Gokavarapu  
50465129  
[mgokavar@buffalo.edu](mailto:mgokavar@buffalo.edu)

1. In the PySpark REPL, run your basic word count program on a single text file.
  - a. What are the 25 most common words? Include a screenshot of program output to back-up your claim.

### Answer:

('one', 199) ('could', 197) ('would', 183) ('yet', 149) ('man', 129) ('upon', 128), ('may', 113)  
('life', 112) ('father', 109) ('first', 108) ('might', 108) ('every', 106) ('shall', 106) ('eyes', 104)  
('said', 102) ('time', 97) ('even', 96) ('towards', 94) ('saw', 94) ('found', 89) ('project', 88)  
('mind', 85) ('night', 84) ('elizabeth', 81) ('day', 80)

### Screenshot:



```
C:\sparkprogrambooks\output.txt\part-00000 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
length: 44,123 lines: 2,921 Ln: 22 Col: 13 Pos: 311 Unix (LF) UTF-8 INS
Normal text file

1 ('one', 199)
2 ('could', 197)
3 ('would', 183)
4 ('yet', 149)
5 ('man', 129)
6 ('upon', 128)
7 ('may', 113)
8 ('life', 112)
9 ('father', 109)
10 ('first', 108)
11 ('might', 108)
12 ('every', 106)
13 ('shall', 106)
14 ('eyes', 104)
15 ('said', 102)
16 ('time', 97)
17 ('even', 96)
18 ('towards', 94)
19 ('saw', 94)
20 ('found', 89)
21 ('project', 88)
22 ('mind', 85)
23 ('night', 84)
24 ('elizabeth', 81)
25 ('day', 80)
26 ('felt', 79)
27 ('ever', 78)
28 ('work', 78)
29 ('heart', 76)
30 ('feelings', 76)
31 ('death', 75)
32 ('must', 74)
33 ('thought', 74)
34 ('i', 72)
35 ('many', 70)
36 ('made', 70)
37 ('us', 68)
38 ('dear', 68)
39 ('also', 68)
40 ('soon', 68)
41 ('friend', 68)
```

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.

### Answer:

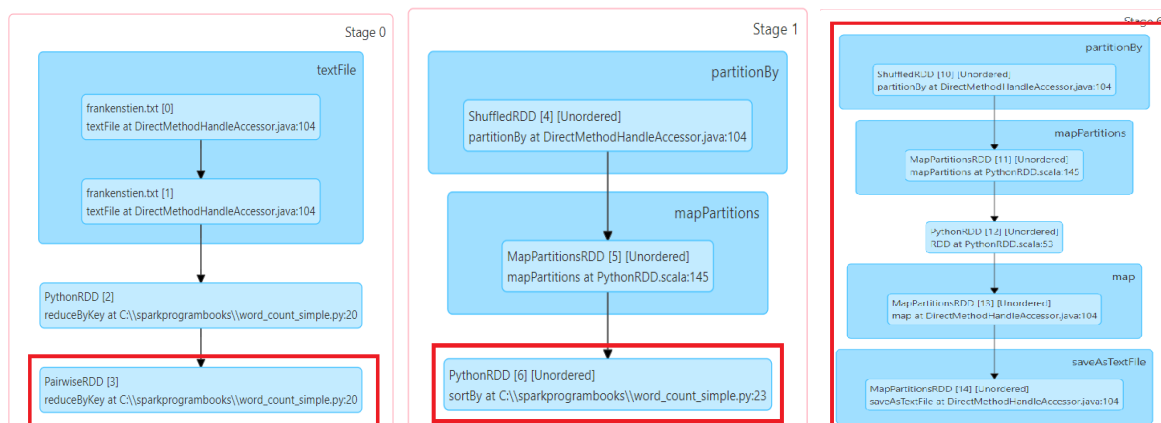
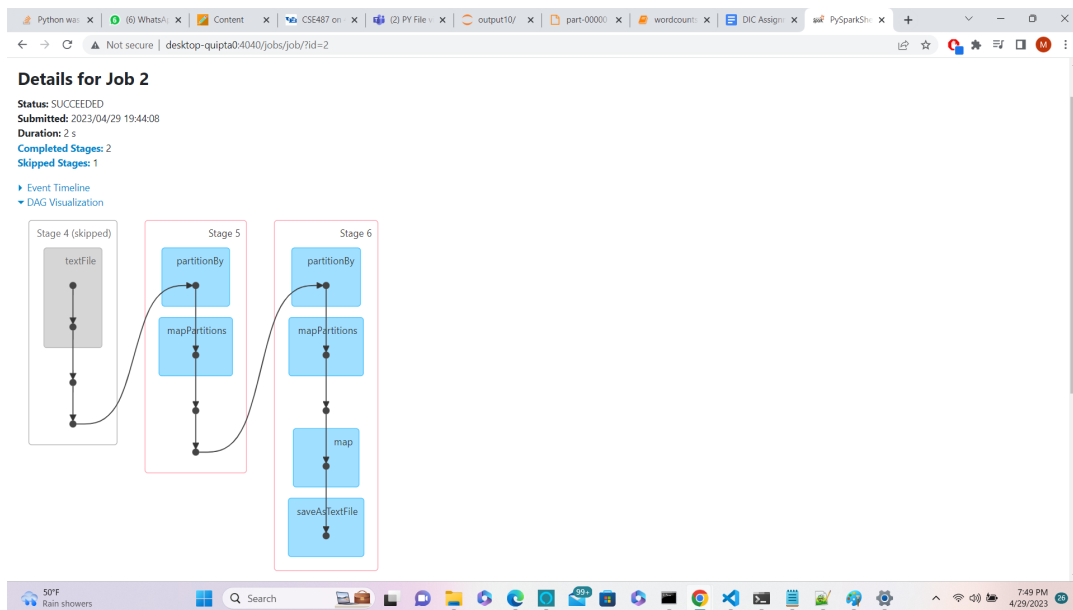
The Execution is broken up into **3 stages**.

### Explanation:

Stages are formed whenever there is a wide dependency which requires shuffling of data across the network.

1. In my code **reduceByKey()** transformation is a wide dependency so it forms the first stage boundary
2. **sortBy()** transformation is a wide dependency so it forms the second stage boundary.
3. Finally, **saveAsTextFile()** action occurs in the third stage of DAG

### DAG Visualizations Screen shot:



2. In the PySpark REPL, run your extended word count program on all 10 text files.
- What are the 25 most common words? Include a screenshot of program output to back-up your claim

**Answer:**

('said', 7672) ('one', 4938) ('would', 4720) ('mr', 3854) ('could', 3080) ('like', 2817) ('little', 2588) ('mrs', 2544) ('man', 2385) ('must', 2141) ('know', 2139) ('see', 2106) ('much', 2095) ('well', 2082) ('never', 2074) ('time', 1968) ('go', 1907) ('good', 1869) ('come', 1775) ('say', 1752) ('made', 1750) ('old', 1700) ('think', 1686) ('shall', 1643) ('might', 1640)

**Screenshots:**

```
C:\sparkprogrambooks\outputmany.txt:part-00000 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window Help
part-00000
1 ('said', 7672)
2 ('one', 4938)
3 ('would', 4720)
4 ('mr', 3854)
5 ('could', 3080)
6 ('like', 2817)
7 ('little', 2588)
8 ('mrs', 2544)
9 ('man', 2385)
10 ('must', 2141)
11 ('know', 2139)
12 ('see', 2106)
13 ('much', 2095)
14 ('well', 2082)
15 ('never', 2074)
16 ('time', 1968)
17 ('go', 1907)
18 ('good', 1869)
19 ('come', 1775)
20 ('say', 1752)
21 ('made', 1750)
22 ('old', 1700)
23 ('think', 1686)
24 ('shall', 1643)
25 ('might', 1640)
26 ('may', 1594)
27 ('two', 1565)
28 ('us', 1531)
29 ('thought', 1528)
30 ('d'artagnan', 1522)
31 ('upon', 1456)
32 ('first', 1419)
33 ('make', 1414)
34 ('away', 1371)
35 ('without', 1365)
36 ('young', 1354)
37 ('great', 1334)
38 ('way', 1331)
39 ('even', 1293)
40 ('jo', 1292)
41 ('nothing', 1261)
42
```

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.

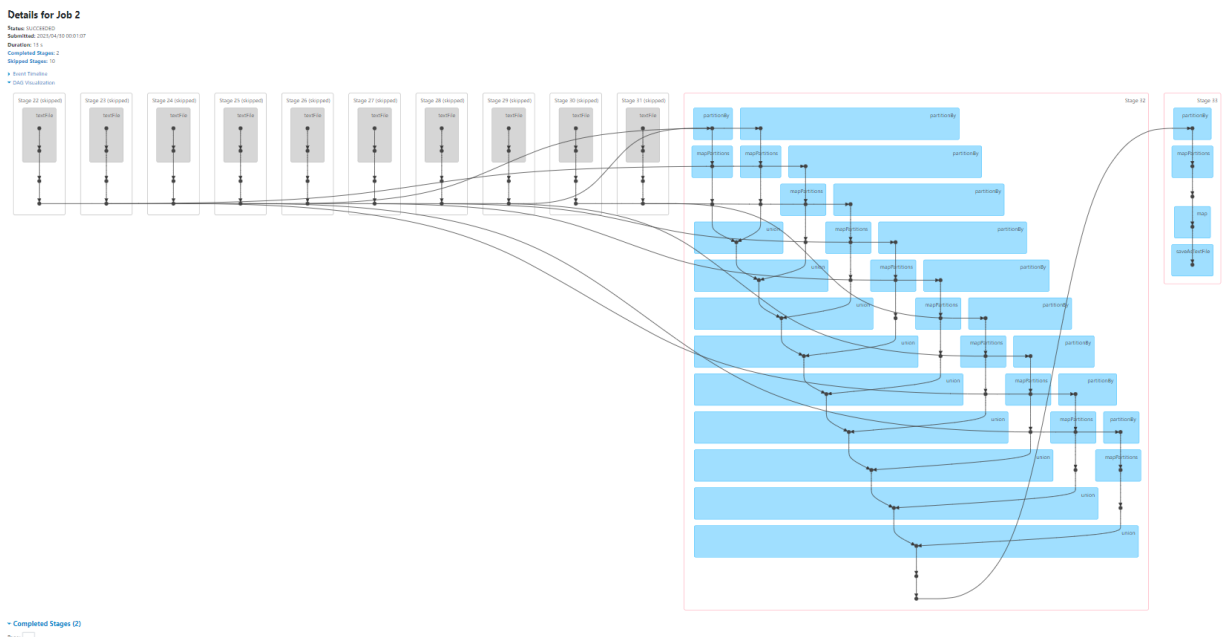
### Answer:

The Execution is broken into 12 stages.

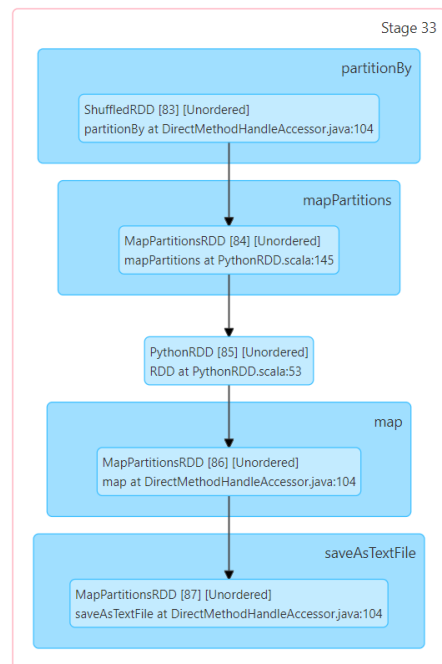
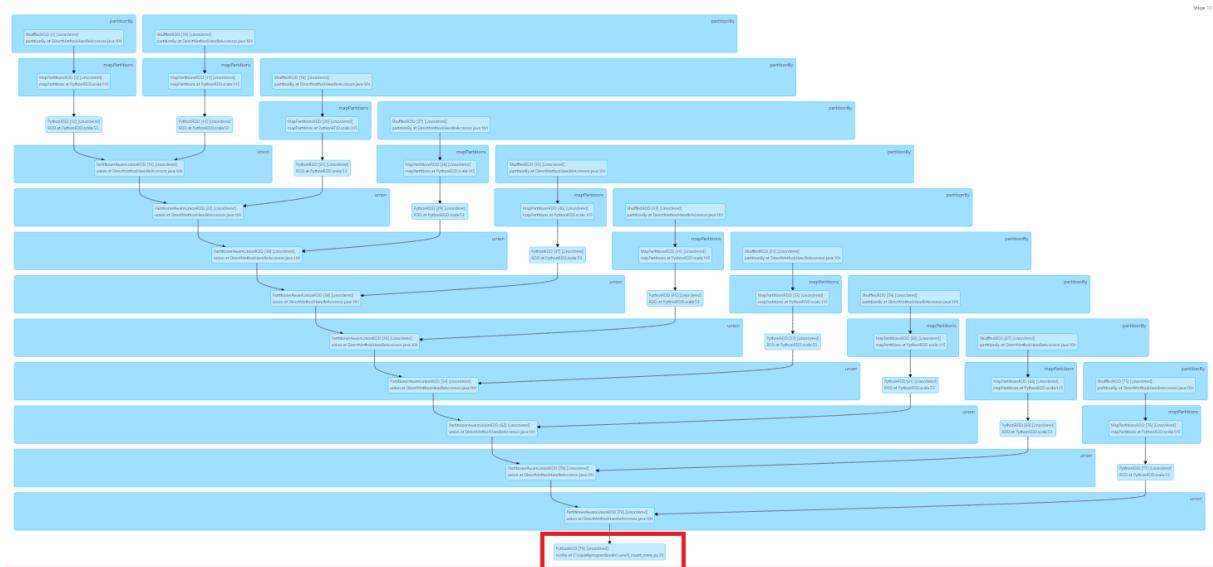
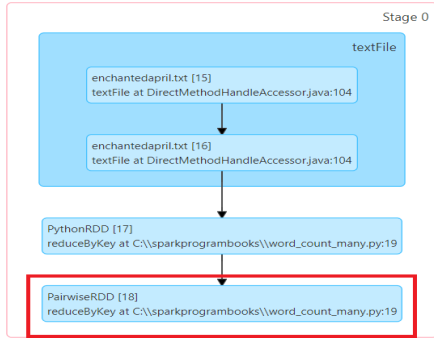
### Explanation:

- In my code for each text file I am using **reduceByKey transformation** so there are 10 text files which form **10 stages** with reduceByKey as stage boundary for each.
- All the resultant RDD's obtained are unionized using union transformation which is a narrow dependency and we don't get new stages.
- Then in the code we do the sortBy transformation to sort all the word counts which is a wide dependency and **sortBy** becomes the **11th stage boundary**.
- Then we execute the **saveAsTextFile()** action in the **12th stage**

### DAG Visualizations:



1st stage DAG, similarly we will have 10 such stages for 10 text files:



3. Your WordCount application should compute the same results as your WordCount application from Homework #1. Answer the following based on your knowledge of both MapReduce and Spark:

a. If you were running your WordCount programs in a large cluster or cloud environment, and one of the nodes you were running on died mid computation, how would your MapReduce and Spark programs handle this?

**Answer:**

- In Mapreduce (MR) if a node dies in the middle of computation it will do re-execution which means the job tracker will detect the failure and assign the task to other available nodes in the cluster. The dead node will be marked as unavailable until it comes back online and no new tasks are assigned to it.
- In Spark, if a node dies, the spark framework uses RDD Lineage information to reconstruct the lost partition on another node in the cluster. In order to recover lost partitions, Spark uses the already tracked actions and transformations and it just replays the transformations on a fresh node.
- In summary, Fault-tolerance in hadoop is achieved by replicating blocks of data. If a node goes down, the data can be found on another node but in spark fault tolerance is achieved by storing the chain of transformations. If data is lost, the chain of transformations can be recomputed on the original data.

b. Explain one concrete benefit you experienced when writing the Spark version of WordCount compared to the MapReduce version.

**Answer:**

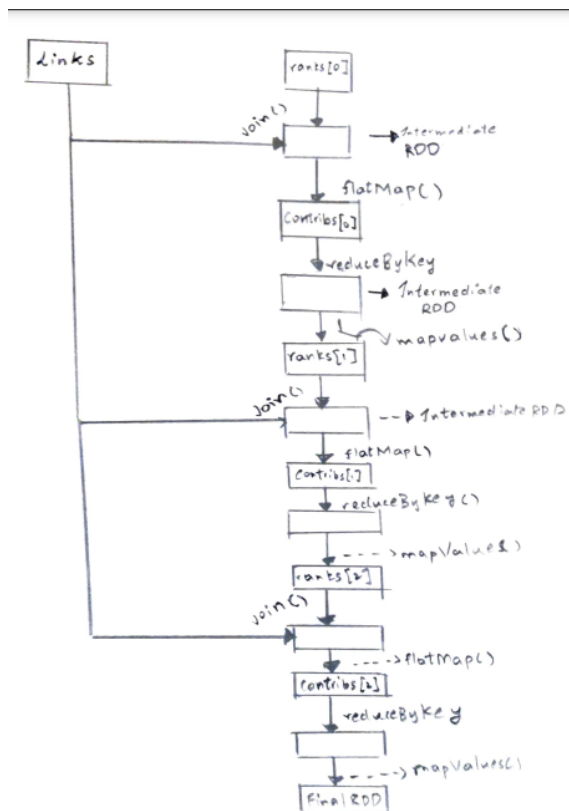
- Benefit I experienced while writing spark version of word count program is, I felt spark is a much higher **level abstraction language** than map reduce because in MR we need to specify all operations in detail like java classes, configurations, mapper and reducer logics etc. But while writing the wordcount program in spark I felt it is much more **streamlined** and also has a good amount of **predefined, lambda functions** which are easy to use. And also I felt it is easier to integrate libraries and data in spark than in MR.
- In Addition to that Spark version of wordcount program is fast when compared to mapreduce version because of its **in-memory processing** which keeps data as **RDD's** whereas in MR intermediate results are written to disk which sometimes causes I/O bottlenecks.

4. Given the above spark application, draw the lineage graph DAG for the RDD ranks on line 12 when the iteration variable i has a value of 2. Include nodes for all intermediate RDDs, even if they are unnamed.

**Answer:**

As the i has a value of two we will have three iterations.

**Lineage graph DAG:**

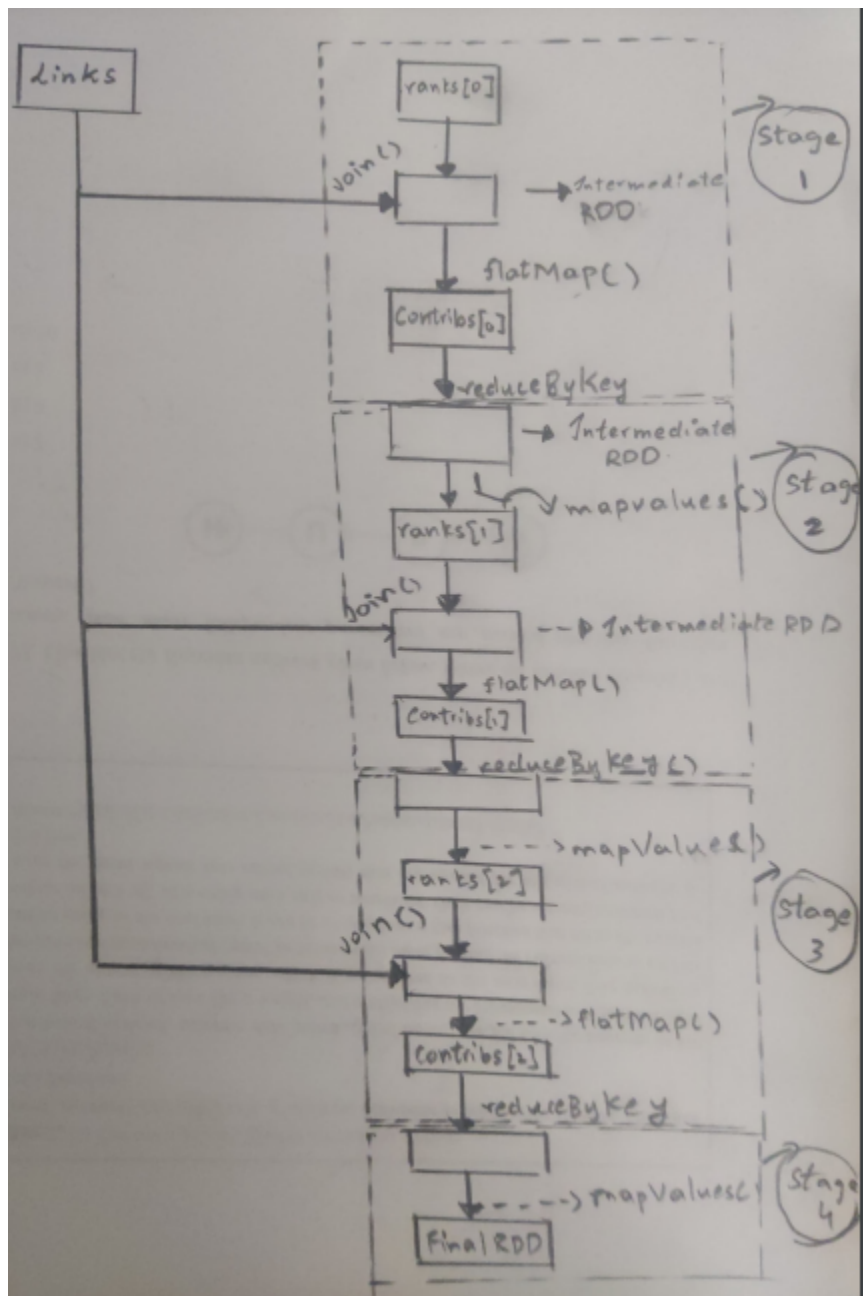


5. How many stages will the above DAG be broken into? Give the number of stages AND draw stage boundaries on your diagram.

**Answer:**

The DAG is broken into **4 stages**. Three iterations triggers three times reducebyKey (stage boundary) so 3 stages get created and the last reducebyKey will generate the final stage so  $3 + 1 = 4$ .

State boundary diagram:





6. Identify in the above code (by function name AND line number) one instance of:

**Answer:**

a. A transformation that results in a wide dependency

groupByKey() - line 3 (or) reduceByKey - line 12

b. A transformation that results in a narrow dependency

map() - line 6 (or) flatMap - line 10.

c. A transformation that may result in a narrow dependency or a wide dependency

join() - line 9

d. An action

count() - line 5

7. How many "jobs" will the above code run if iters has value 10?

**Answer:**

One because of the "count()" action.

8. What algorithm is the above code an implementation of?

**Answer:**

The code is an implementation of the PageRank algorithm.

**References:**

- Professor Eric Mikida Lecture slides on Spark.
- Spark demo code on simple and many word count programs by Prof. Eric Mikida.