Homework #2 - Spark

Due: 5/1/23

Content Covered
Spark

Homework Overview

For this homework you will get a bit of experience writing and running Spark code, and performing some follow-up analysis. In general, you will install and set up Spark, write a basic word count program, and then answer some questions about data flow in Spark applications.

General Homework Requirements

1. **Work Environment:** This homework must be written in Python.
2. **Programming:** Relevant tutorials and resources are linked below to aid in the programming portion of this homework.
3. **Academic Integrity: You will get an automatic F for the course if you violate the academic integrity policy.**
4. **Teams:** This homework is **an individual assignment**. You are not permitted to work with anyone else on this assignment. All work submitted must be yours and yours alone.

Submission Format

Your submission should be a single zip file that contains a src/ directory for your code, and a single PDF report. Submissions must be made on UBLearns by 5/1/23 @ 11:59PM.

- Your src/ directory should contain your code files and your input dataset (10 txt files from project gutenberg), and your list of stopwords.
- Your report should contain the answers to each of the questions below, including any required screenshots of your application output.

Setup

To prepare your development environment for this homework you must first install and set up PySpark. **Make sure to start this process as early as possible, issues WILL come up.** To install PySpark, follow the instructions here:
https://spark.apache.org/docs/latest/api/python/getting_started/install.html

Once you have Spark setup, you must also get the data you will be using for the homework. Go to Project Gutenberg (https://www.gutenberg.org/ebooks/) and download 10 different books as plain text files. Each of the files you download must be at least 100 kB in size, but the larger the book the more interesting your results may be. **You may use the same books you used from HW #1.**

Part 1 - Implement and Analyze Word Count [14/30 points]

Basic Word Count

For the first part of this homework, you must write a basic word count program that reads in a single text file, counts the number of occurrences of each word in that text file, 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.

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

1. It must be case insensitive (see `lower()` in Python)
2. It must ignore all punctuation (see, for example, `translate()` in Python)
3. It must ignore stop words (see `filter()` in Spark)
4. 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.

Any examples from class and any official Spark documentation can be used as reference material. Cite any code you use directly as necessary. 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

Note: To make later parts of this HW easier, you should write your word count code in a function that takes the Spark context as an argument so you can call your function from the interactive PySpark REPL. See the in-class demo code as an example.

Extending Your Application

Once you have your basic word count working for a single text file as described above, you will now extend it to work for a list of text files, combining the results from each file into a single RDD of key-value pairs, where the key is the word and the value is the number of times that word appeared across all passed in text files. The result should only be sorted and written to file once after all text files have been processed.

Note: This is different from the in class example where we combined results using **`join()`**. See instead **`union()`** in Spark.

Analysis

Answer the following questions based on your WordCount application.

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.
    [1 point] Screenshot of 25 words without any stop words or punctuation (if a smart quote or occasional stop word shows up that is OK)
    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.
    [1 point] for a valid number of stages (should be 3 but if the explanation and screenshot backup a slightly different number that is fine too)
    [1 point] for explanation of number of stages (should mention that stages are dictated by wide dependencies and reduceByKey and sort will cause wide dependencies)
    [1 point] for a screenshot that aligns with their answers

2. In the PySpark REPL, run your extended word count program on all 10 text files.
    a. What are the 25 most common words? Include a screenshot of program output to back-up your claim.
    [1 point] Screenshot of 25 words without any stop words or punctuation (if a smart quote or occasional stop word shows up that is OK)
    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.
    [1 point] for a valid number of stages (should be 3 or 12 depending on if they did union before or after reduce by key, but if the explanation and screenshot backup a slightly different number that is fine too)
    [1 point] for explanation of number of stages (should mention that stages are dictated by wide dependencies and reduceByKey and sort will cause wide dependencies)
    [1 point] for a screenshot that aligns with their answers

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?
    [1.5 points] mentions that MapReduce would get lost data from a node where that data was replicated
    [1.5 points] mentions that in Spark, the lineage graph (or DAG) would allow the program to recompute the data that was lost
    b. Explain one concrete benefit you experienced when writing the Spark version of WordCount compared to the MapReduce version.
    [3 points] Most likely looking for something related to productivity: Higher level abstractions made it easier to use, less code to write, etc. Other answers are potentially valid.

Part 2 - Code Analysis [16/30 points]

Read through the following PySpark code snippet and answer the questions following:

```
 1  lines = sc.textFile(file)
 2  links = lines.map(lambda urls: parseNeighbors(urls)) \
 3               .groupByKey()
 4               .cache()
 5  N = links.count()
 6  ranks = links.map(lambda u: (u[0], 1.0/N))
 7
 8  for i in range(iters):
 9    contribs = links.join(ranks) \
10         .flatMap(lambda u: computeContribs(u[1][0], u[1][1]))
11
12    ranks = contribs.reduceByKey(lambda a,b: a+b) \
13         .mapValues(lambda rank: rank * 0.85 + 0.15*(1.0/N))
14  return ranks
```

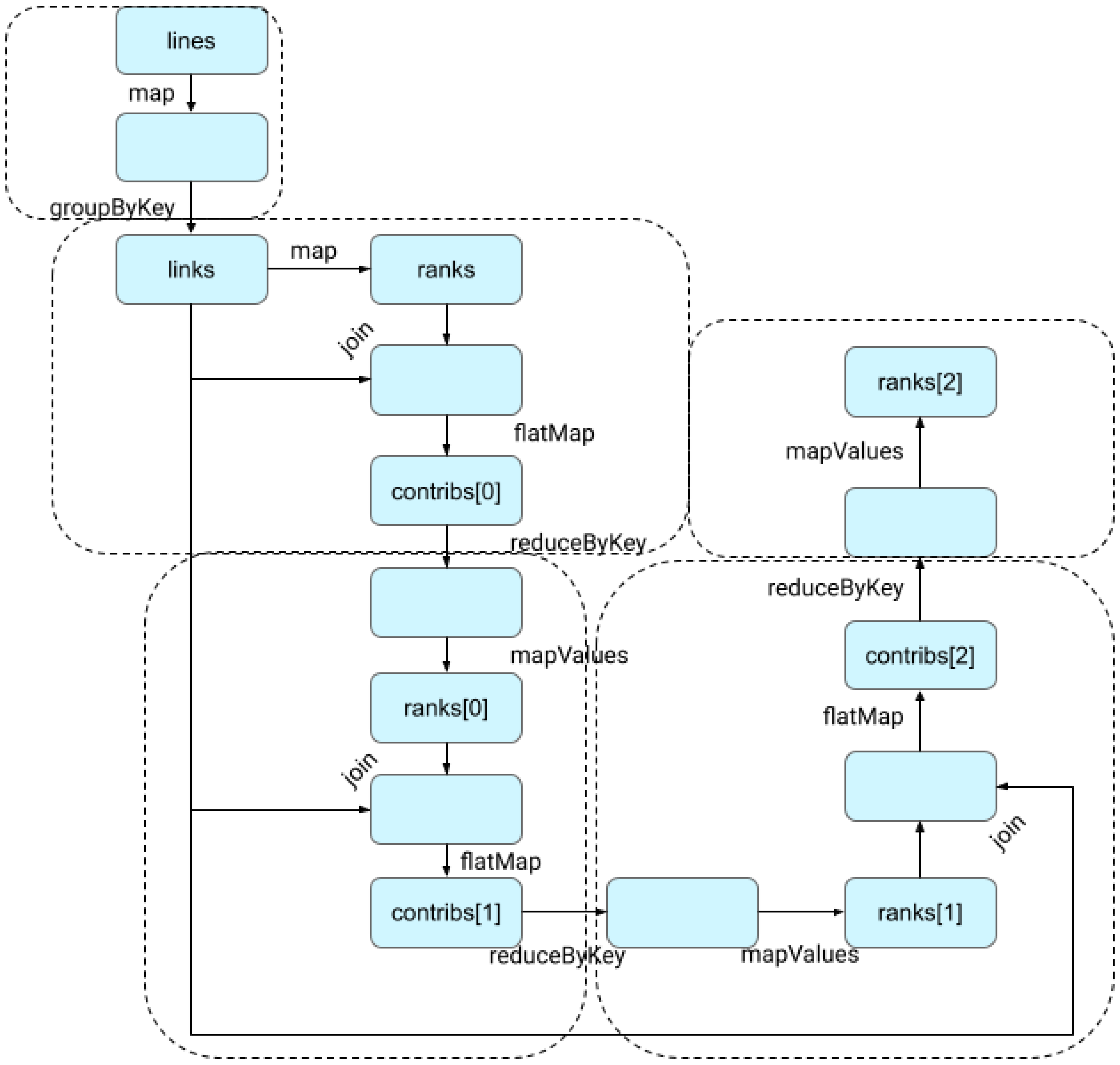
4. Given the above spark application, draw the lineage graph DAG for the RDD **ranks** on line 12 when the iteration variable **i** has a value of 2. Include nodes for all intermediate RDDs, even if they are unnamed.
   See answer diagram on last page (ignore stage boundaries for this Q)
   Deduce 1 point for any of the following mistakes:
   - **Didn't include intermediate RDDs (the unlabeled boxes)**
   - **Didn't include the right number of iterations (3 iterations + setup, so 4 boxes corresponding to "ranks")**
   - **DAG doesn't end at ranks**
   - **DAG doesn't include lineage for links and/or the initial ranks RDD**
     - **Specifically, lines is read from file, and mapped and grouped to get links, and links is mapped to get initial ranks**
   - **1 point off for each instance where the arrows are incorrectly labeled or lead to the wrong RDD**

5. How many stages will the above DAG be broken into? Give the number of stages AND draw stage boundaries on your diagram.
   [2 points] Correct number of stages (either 5 or 8 if the joins are wide), deduct 1 point if they are only off by one stage
   [2 points] For correct stage boundaries (can draw them as shown in the diagram below, or hierarchically. As long as the boundaries are drawn at the groupByKey and reduceByKey transformations, and optionally at the joins if they treated joins as wide)

6. Identify in the above code (by function name AND line number) one instance of:
    a. A transformation that results in a wide dependency
    [1 point] groupByKey line 2 OR reduceByKey line 12
    b. A transformation that results in a narrow dependency
    [1 point] map line 2 OR line 6 OR flatMap line 10 OR mapValues line 13
    c. A transformation that may result in a narrow dependency OR a wide dependency
    [1 point] join line 9
    d. An action
    [1 point] count line 5

7. How many "jobs" will the above code run if `iters` has value 10?
[2 points] 1 job will be run

8. What algorithm is the above code an implementation of?
[2 points] PageRank


lines
map
groupByKey
links
map
ranks
join
flatMap
contribs[0]
reduceByKey
mapValues
ranks[0]
join
flatMap
contribs[1]
reduceByKey
mapValues
ranks[1]
join
flatMap
contribs[2]
reduceByKey
mapValues
ranks[2]