ECE 420 Parallel and Distributed Programming
Lab 5: Exercises for Basic Apache Spark

Winter 2022

In this lab, we use the spark-shell for Scala. All your solutions should be commands in the interactive shell. To launch the spark shell, type the following command in a terminal:
$ spark-shell

The input data will be a text file named "data_input" which represents a sparse graph with directed links. Every line represents a directed link with the first entry being the source node and the second entry being the destination node. The original data is in "web-Stanford.txt" and a script "datatrim.c" is provided to fetch a data subset of a specified size, generating the input file "data_input".

1 Tasks

In this lab, there are four exercises to be performed on the same data set. For every task, you will be asked to generate some output stored in a variable named "result". For each task, once you have generated the output, please save your command line input in a text file named "taskX.txt" (where "X" is the task index). To verify the correctness of your solution, you may start with a small input (e.g. graph with 4 nodes), from which you can easily generate the intended output for comparison. Due to the "lazy evaluation" of spark, the computation will only be kicked off if *action* operation is engaged. Therefore, for the tasks that require an RDD result, you may first add some action method, such as "take", in order to print out your result. Once the correctness is confirmed, remember to remove the *action* method in order to produce an RDD result.

1.1 Task 1

For each node, there could be links going into it or coming out of it. We call them, respectively, *inlinks* and *outlinks*. For a node, the number of *inlinks* is its *indegree*, while the number of *outlinks* is its *outdegree*.

Task: Return a variable named "result" storing the **maximum** *indegree* among all the nodes.

1.2 Task 2

Following the same definitions in the former task, we define the *neighbor* of a node as some other node connected to it via either an inlink or an outlink.

Task: Return a key-value pair RDD named "result" with the keys being the node IDs, and the corresponding value of each key (node) storing a list of all its *neighbors*.

1.3 Task 3

Task: Return an RDD named "result" containing the node ID(s) which have the largest *outdegree*.

1.4 Task 4

In a directed graph, there may exist two nodes, node A and node B, with no direct link from A to B. However, there may exist a node C, such that there is a link from A to C and another link from C to B. In this case, there is an indirect link from A to B via node C. We call such indirect links via one intermediate node as *two-hop links*.

Task: Return a key-value pair RDD named "result" containing BOTH all the original one-hop links and all the *two-hop links*, in which each key stores the source node of a link and the corresponding value stores the destination node of the link.

To help understand this question, consider a graph with 4 nodes and 3 links, where the input data looks like:

```
2 3
1 2
```

3 4.

The desired output should be something similar to:

```
1 2
1 3
2 3
2 4
3 4.
```

2 Submission Instructions

Each team is required to submit a zip file to eClass by the submission deadline. The zip file should be named “StudentID.zip”, where “StudentID” is the Student ID of **one** of your group members (it does not matter which member, though this should be consistent for all submissions throughout the course).

The zip file should contain the following two folders:

1. “Members”: this folder should contain a single text file *named* “members.txt”, listing the student IDs of ALL group members, with each student ID occupying one line.
2. “Code”: this folder should contain four text files, each of which is the solution script to one of the four tasks. They should be named as “taskX.txt” with “X” being the index of the task;

DO NOT include the “datatrim.c” file or the “web-Stanford.txt”.

Note: you MUST use the filenames suggested above. File names are case-sensitive. You MUST generate the required zip file by directly compressing all the above files, rather than compressing a folder containing those files.

For testing purposes, we would expect your scripts to work once we put a “data_input” file in the same path where the spark-shell is started. We will then execute your script line-by-line in the spark-shell.

A Appendix: Marking Guideline

Task 1:	1
Task 2:	1
Task 3:	1
Task 4:	2
Total:	**5**