04.04.2022 Programming assignments. Due by 14.04.2022 at 10 a.m.

These practical programming assignments will be graded and will account for 15% of your final score. Just like in the programming exercises, solve them using Spark 2.3.2 and Scala 2.11.12.

Assignment 1:

Given a directed Graph $G = (V, E)$, use a map-reduce chain to compute the set of vertices that are reachable in *exactly 4 hops* from each vertex. For example, in a graph with vertices $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$, the vertex e is reachable in exactly four hops from a. To achieve this, we will use the following well known result:

If A is the adjacency matrix of the directed graph G, then the matrix A^n (i.e., the matrix product of n copies of A) has an interesting interpretation: the element (i, j) gives the number of (directed or undirected) walks of length n from vertex i to vertex j.

Thus $A^4[i, j] > 0$ will indicate the existence of a path of length 4 from i to j. We consider the matrix multiplication implementation of `Week-3, exercise 4` with the following modifications.

Input

Since the adjacency matrix should be multiplied to itself, the `matrixMultiply` function takes as input only a single matrix. The input to the function are the edges of the graph instead of the adjacency matrix given by `RDD[(Int, Int)]` where (i, j) corresponds to a directed edge from $i \rightarrow j$.

Output

The `matrixMultiply` function must output the product matrix also as a graph i.e. `RDD[(Int, Int)]`. For example, if $A^2[i, j] > 0$, the edge (i, j) must be included in the output. If $A^2[i, j] = 0$, the edge (i, j) is dropped from the output. This allows us to reuse the same function twice for evaluating A^4 as `matrixMultiply(matrixMultiply(A))` and output all four hop neighbors.

Task

The code template is provided at https://gitlab.epfl.ch/sacs/cs449-2022-sds-public/exercises/week7 (directory `task1`). You must provide the implementation of four functions: (1) `countVertices`, (2) `mmMapper`, (3) `mmReducer` and (4) `matrixMultiply`. The `matrixMultiply` function must make use of `mmMapper` and `mmReducer`. Further instructions regarding these functions can be found as comments on top of each function.

You can run the code locally as well as on `iccluster028.iccluster.epfl.ch`. Once inside the `task1` directory, do `sbt run` to run the main code.

To help you debug and give you a flavour of automated tests that we will run for grading, a subset of them have been provided in `src/tests/scala/CorrectnessTests.scala`. To execute all tests, do `sbt test` inside `task1` directory. If you want to run specific tests, you can temporarily disable other tests by changing the `test` to `ignore` as shown in the following example. This will ignore the corresponding test (*"Counting Number of Vertices"* in the example) when `sbt test` is called. However, remember to restore all tests before submitting.

```
//test 1
test("Counting Number of Vertices") {...}

//test 1
ignore("Counting Number of Vertices") {...}
```

Assignment 2:

You work at a tech-startup *SimplyStore* that provides storage services. For their new product, you are tasked with deploying a large scale peer-to-peer key-value storage service. Looking at the experimental costs, you decide to create a simulator using classes and objects to test the performance of your next-gen system.

In this task, you will implement the algorithms for storing and retrieving key-value pair on a pool of peers connected via an overlay network. Each participating node is identified by an *id*, has a limited *memory* (number of keys,value pairs the node can store), knows the *id*s of its neighbours, and has access to the *router*. Any node can however ask other nodes for the *ids* of their current neighbours at any time. A node can talk to other nodes by sending messages to other nodes through the *router* by calling the `router.sendMessage` function. There is no other way to contact other nodes. Moreover, you cannot store more than the *memory* supports.

A USER of the service can ask to STORE or RETRIEVE (key, value) pairs to any node. For simplicity, the first node to which USER sends the query message is called as HOST in further discussion. The *key* that the user requests to STORE may be stored on a different node depending on the *memory* used on the HOST. Similarly, on the RETRIEVE request, the value associated to the key must be returned, even if the key is stored on a node other than HOST.

The message contains the *id* of the originator of the message, a *messageType* field and *data* field. Whenever a node or user sends message to another node, the `onReceive` function is called at the destination node to process the query. **This is the only function you need to complete.**

Since this simulator should represent the real world, new nodes may join at any time (for simplicity, nodes only join when no query is being processed). Moreover, up to 4 nodes can have a disk failure (lose all previous data). This means that these nodes will still be able to process the queries, and add new (key,value) pairs, but their previous keys are lost. Your service should still be able to function and correctly respond to the USER's queries (still return the correct value of a previously inserted key).

The code template is provided at https://gitlab.epfl.ch/sacs/cs449-2022-sds-public/exercises/week7/-/tree/main/task2. You must provide the implementation of the function: `onReceive` in `task2/src/main/scala/MyNode.scala`. Further instructions regarding these functions can be found as comments on top of each function. If you are unfamiliar with classes and inheritance, please brush up your knowledge of inheritance in Scala before attempting the task.

Sub-Task 1

The basic code to store keys on the HOST is already present. Mentioned as TODO: task 2.1 in `task2/src/main/scala/MyNode.scala`, in this task you need to implement the searching of the keys in other nodes. Consequently, any node should be able to retrieve the key stored on any other node in the service. For example, if USER sends a STORE query to u_i to insert $key_i -> value_i$, a RETRIEVE query on u_j for key_i should return $value_i$.
Remember: all communication happens through messages via the `router.sendMessage` function.

Sub-Task 2

Moving on with the previous sub-task, assume that now there are multiple queries coming in. The HOST node's *memory* is exhausted. Implement the code to ask another node to store the incoming *key* when the HOST can't store any more. This sub-task is mentioned as TODO: task2.2 in `task2/src/main/scala/MyNode.scala`

Sub-Task 3

Implement fault tolerance in your system to continue responding with correct values even if up to 4 nodes lose their stored keys. This sub-task is mentioned as TODO: task2.3 in `task2/src/main/scala/MyNode.scala`

Constraints

- You must use the APIs provided for the `Node`, `Message` and `Router`. Do not change any existing function in the template (even the *Task2.scala* file while submitting).
- Example queries are given in Task2.scala. We will follow a similar structure of the queries in the final tests.
- For debugging purposes: each line of the overlay file represents *id memory* [*neighboursIDs*].
- For each STORE query, there will be at least 6 nodes who can accommodate the key in the tests. All keys will be distinct.
- The user's id will always be USER.
- There are no self-loops in the overlay graph
- Once stored, the USER shall not request the removal of the key.
- USER will use the same message format in the template for STORE, RETRIEVE and GET_NEIGHBOURS. The response messages to the USER should also follow the provided format.

- For the final submission, do not use `print` statements anywhere in `MyNode`.
- Since this is a simulator, everything executes sequentially. Remember to avoid any infinite loops.
- A node can always store at least one key ($memory \geq 1$).

User queries

There are three types of user queries. The Message format corresponding to each of them is:

1. Message("USER", "STORE", "key->value")
2. Message("USER", "RETRIEVE", "key")
3. Message("USER", "GET_NEIGHBOURS")

Responses

For a user query, the following can be the responses:

1. Successful Store: Message("HOST_ID", "STORE_SUCCESS")
2. Failed Store: Message("HOST_ID", "STORE_FAILED")
3. Successful Retrieve: Message("HOST_ID", "RETRIEVE_SUCCESS", value)
4. Failed Retrieve: Message("HOST_ID", "RETRIEVE_FAILED")
5. Neighbours IDs response: Message("HOST_ID", "NEIGHBOURS_RESPONSE", <space separated string of neighbour ids>)

Instructions

- For all assignments, you have a code template at
  https://gitlab.epfl.ch/sacs/cs449-2022-sds-public/exercises/week7.
- On top of each source file, replace `YOUR_FULL_NAME_HERE` by your full name.
- The source files include the expression `???` (it throws a `NotImplementedError`). You should replace them with your code.
- You are free to add as many functions as you wish, but you cannot change class names or signature of the functions that already are in the template (their names, parameters and return types). Moreover, do not edit the file `build.sbt`. We will use automated tests to grade the assignments.
- Remember to treat degenerate cases (empty sets, division by zero, identifiers not found). Do not throw exceptions.
- Do not hesitate to comment your code. In case your results diverge from what is expected, we may consider your comments in the evaluation.

- Even if you write your code elsewhere (*e.g.*, your laptop), you must be sure that it runs with `sbt run` on our cluster gateway (`iccluster028.iccluster.epfl.ch`). Your work will be evaluated over there.
- You are free to discuss with your classmates about how to solve the assignments, but each student will deliver their own solution: beware of plagiarism. We are going to compare your source files to detect similarities among different solutions. In case of reasonable doubt, all people involved will be invited to make oral presentations of their solutions and answer to the examiner's questions.

Deliverables

Let us suppose that your SCIPER number is 999999. We expect you to upload a single archive file named `999999.tgz` or `999999.zip` at https://moodle.epfl.ch/mod/assign/view.php?id=1144455. You can possibly do it like this:

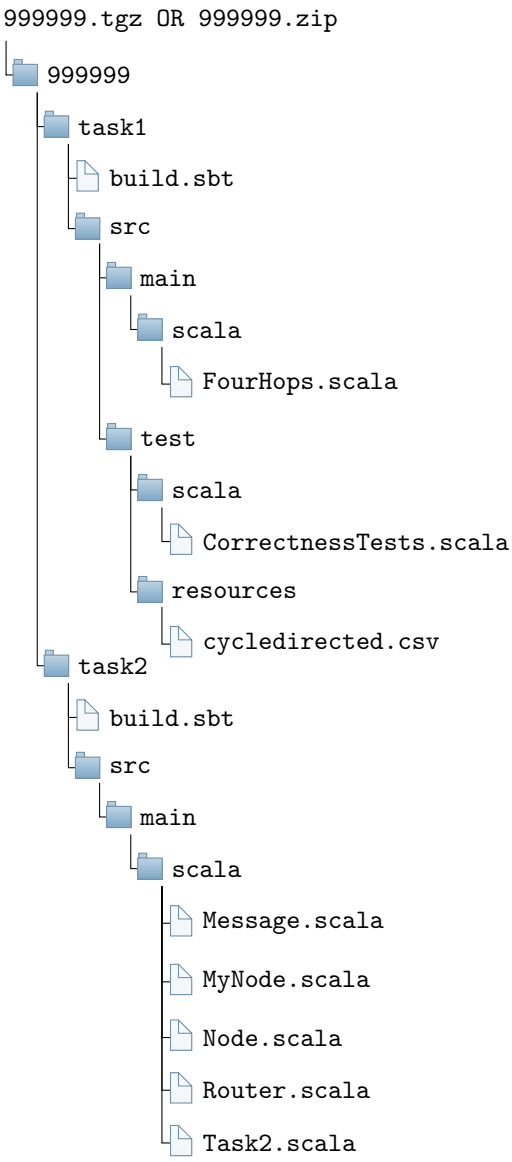
```
$ git clone https://gitlab.epfl.ch/sacs/cs-449-sds-public/exercises/practical-assignment.git 999999

# Solve the assignments by updating the required source files in 999999/task[1, 2]/src/main/scala/
# Run and test your code. You will probably have some auto-generated folders: project and target.
# Remove them before generating the submission archive file.

$ rm -rf 999999/task1/{project,target}
$ rm -rf 999999/task2/{project,target}
$ tar czf 999999.tgz 999999/

# Upload 999999.tgz to Moodle / Week 7 / Programming assignment
```

The internal directory structure of your archive file must follow the structure given on the next page. (Replace 999999 by your SCIPER number):


999999.tgz OR 999999.zip
999999
task1
build.sbt
src
main
scala
FourHops.scala
test
scala
CorrectnessTests.scala
resources
cycledirected.csv
task2
build.sbt
src
main
scala
Message.scala
MyNode.scala
Node.scala
Router.scala
Task2.scala