PROOF OF CONCEPT 2

FIT 5202 – GROUP ASSIGNMENT

Wongnaret Khantuwan
That Bao Thu Ton
Thi Bich Ngoc Hoang
Lam Wai Chun

Group 1

Semester 2 - 2019

Code explanation

1. Choice of Spark API

The spark API will be imported to support the graph processing and result displaying. The usage of each API is listed as below

API	Usages
org.apache.spark.graphx._	Extends the Spark RDD by introducing a new Graph abstraction
org.apache.spark.rdd.RDD	Support RDD processing
scala.util.MurmurHash	Hash function to create unique vertices ID
org.apache.spark.SparkContext._	Create Spark context
org.apache.spark.sql.SQLContext(sc)	Create SQL context to support converting to data frames
sqlContext.implicits._	Support using SQL methods to transpose data frames

Figure 1: Choice of SparkAPI table

2. Code flow

a. Assumption

As there might be different definitions in terms of "shortest distance", our group primarily focuses on our own assumptions below.

"Each flight, which has the same route from one airport to another airport, might have difference distances."

Therefore, we use the averaged distance from one airport to another airport as the measure for the distance in our report. In other words, the code attempts to find the 10 shortest averaged distances between 2 airports.

b. Assumption

Figure 2 illustrates the code flow to display the 10 shortest distances between two airports.

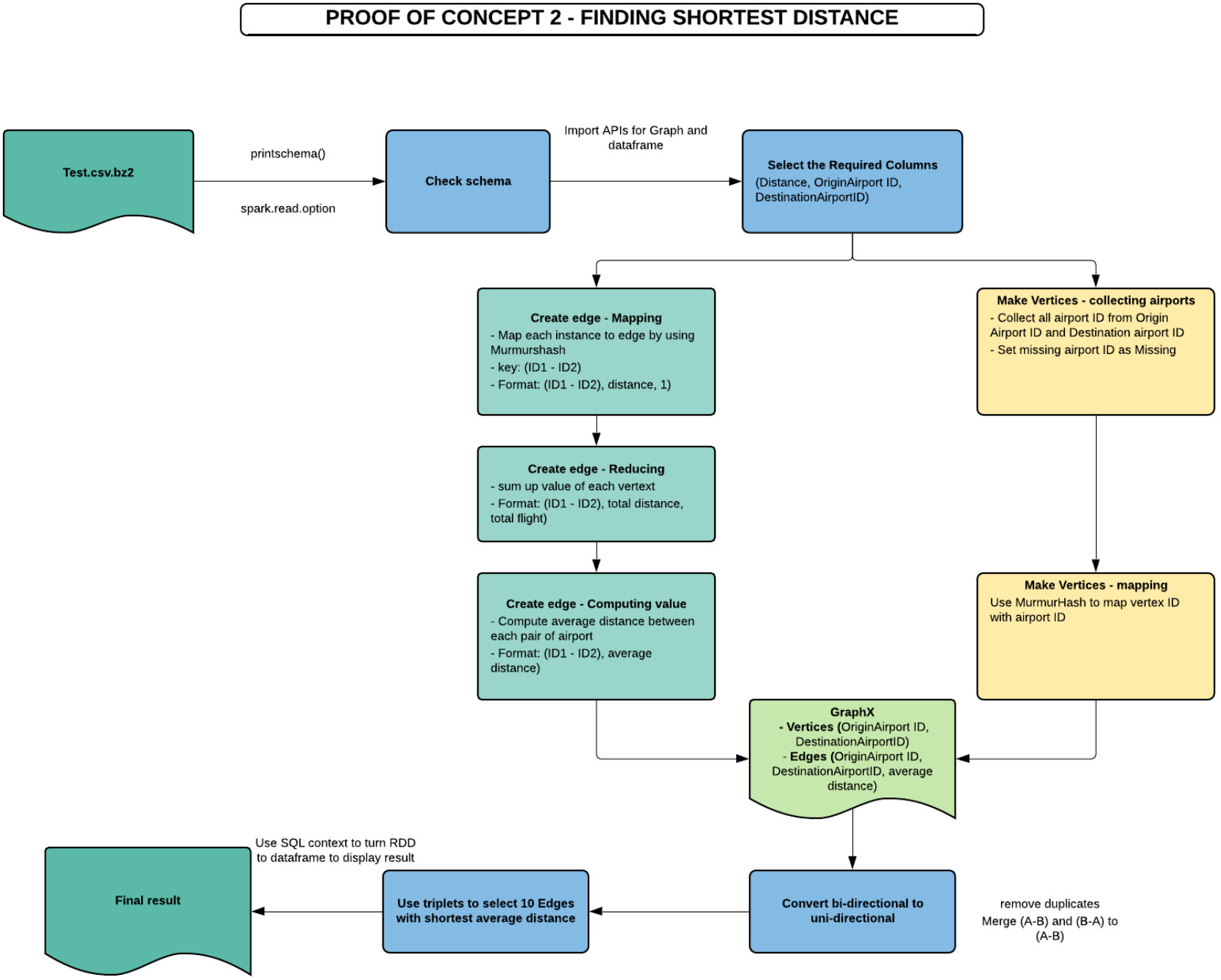

Figure 2: Code Flow

Step 1: Read data and import necessary APIs

First of all, we use "spark.read.option" to import data and check the schema of the file (Figure 3). Next, all necessary APIs listed in Section 1 is loaded to be ready for the next steps.

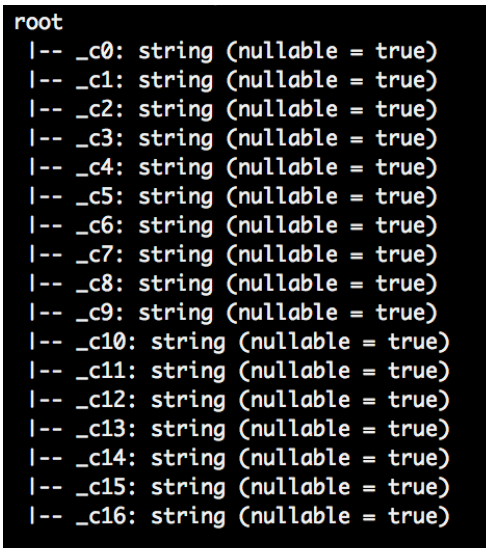

Figure 3: Original Data Schema

Step 2: Create the graph

In this step, we will create the graph, where airport ID is stored in the vertexes and the average distance will be stored in the edge of the graph.

2.1 Create Vertices

- All unique airport IDs are collected from OriginAirportID and DestinationAirportID columns
- Set default airport as "Missing"
- Create vertices ID using MurmurHash function

2.2 Create Edges

- Map distance of each airport and the vertices ID under the form ((VertexID1, VertexID2), distance, 1). It is noted that each flight is assigned with the weight of 1.
- Implement reducing by summing up all the distance and the weight. As a result, the edge now has the format ((VertexID1, VertexID2), total_distance, total_flight).
- Compute average distance and store it in the edge. The average distance is calculated by dividing total_distance by total_flight.

Step 3: Displaying the graph

In the dataset, we can see two-way flights between the same airports (i.e., from A to B and from B to A). This step will handle with this problem and remove one in two ways. Finally, we just display 10 unique distances.

- Converts bi-directional edges into uni-directional by using graph.convertToCanonicalEdges
- Sort all the edge in ascending order by using graph2.triplets.sortBy(_.attr, ascending=true).
- Convert the result to a data frame, rename columns, and put the limit to display only 10 shortest distances.

Final result

From airport	To airport	Average Distance
14256	15841	31.0
10930	10157	56.0
12335	14520	67.0
13930	13342	67.0
15024	14843	68.0
12945	11193	70.0
11292	11109	73.0
12266	11049	74.0
11433	12884	74.0
14633	13487	76.0

Figure 4: Final result