Aims

This exercise aims to get you to practice Spark GraphX Programming.

Background

The transformation and action functions examples are available at:

http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

The GraphX programming guide is at:

https://spark.apache.org/docs/latest/graphx-programming-guide.html

A tutorial of Scala 2 is available at:

http://docs.scala-lang.org/tutorials/?_ga=1.99469143.850382266.1473265612

Spark GraphX programming

Question 1. Create a graph from a file in GraphX (do this in spark shell).

Download the file tiny-graph.txt from https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68841. There are many ways of loading a graph from disk files. GraphX provides a function fromEdge () to create a graph from only an RDD of edges, which is the most suitable method for the give format. The steps are as below:

a. Import Graphx relevant classes: `import org.apache.spark.graphx._`
b. Load the file into an RDD named edges (use sc.textFile())
c. Split the lines in edges and transform each line to a tuple (srcId, dstId, attr), where srcId and dstId are of Long data type, and attr is of Double data type (use map())
d. Transform each tuple to an Edge (use map())
e. Use Graph.fromEdges () function to create a graph
f. Now you can use the created graph to do various tasks. For example, you can check the triplets by: `graph.triplets.collect()`

The detailed code can be seen at:

https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68844

Question 2. Compute the single source shortest distances.

Combine the codes in Question 1 and the codes provided in the lecture slides, and compute the shortest distances with source node 0. Your program should take two

parameters: the first is the file location and the second is the source node ID. The code template can be downloaded from:

https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68844

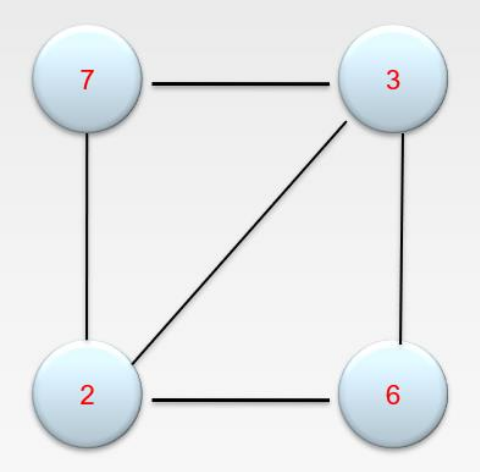
To package your project by sbt, please use the following configuration file:

```
name := "GraphX Project"
version := "1.0"
scalaVersion := "2.12.10"
libraryDependencies += "org.apache.spark" % "spark-core_2.12" %
"3.1.2"
libraryDependencies += "org.apache.spark" % "spark-graphx_2.12" %
"3.1.2"
```

Note that your jar file name now would be: graphx-project_2.12-1.0.jar. Submit your job to Spark by the following command:

```
$ spark-submit --class "comp9313.lab7.SSSPExample" ~/sparkapp/target/scala-
2.12/graphx-project_2.12-1.0.jar file:///home/comp9313/tiny-graph.txt 0
```

Question 3. Propagate the minimum value in an undirected connected graph.



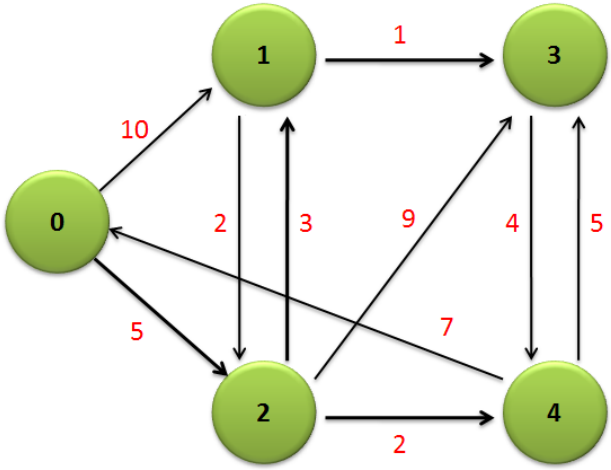
You can also first create the vertex RDD and the edge RDD and then build the graph from the two RDDs. The detailed code can be seen at:

https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68843.

Use the above template to finish the task. Please refer to the lecture notes of Chapter 5.2.

Question 4. Given a directed graph and a number k, find all vertices that can go back to themselves within k hops. Your program should take a file containing the graph and a value as input. You can use the code template of Question 4 for this question as well, and now the second parameter is the value of k.

For example, give the below example graph and k=2, the output is 1, 2, 3, and 4. 0 is not in the result because it needs 3 hops.



Hint: the idea is that, on each vertex, we use a set to store all the vertices that can reach it. After k iterations, if a vertex is in its own set, we know that this vertex is in the result.

a. Use Set[VertexId] as the vertex attribute
b. Initially, make the sets on all vertices empty
c. When a vertex receives a message (a set of vertices that can reach it), combine the set on the vertex and the message as the new attribute of this vertex
d. When sending out messages on each edge, merge the source vertex and the set on the source as the message to the destination vertex
e. When combining messages from different sources on a vertex, you just need to merge all the messages (sets)
f. To merge two sets *a* and *b*, you can use the ++ operator, and to add one vertex *v* to a set *a*, you can use the + operator
g. Set the maximum number of iterations to k, because we only need to run k rounds to get the results (why?)

Try to solve the problem on your own first. If you have no clue, you can use the template at:

https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/68842.

It is strongly recommended that you have a try. This question aims to make you have a better understanding of the Pregel operator, and thus to help you solve the second problem of Project 2.