All scripts require GraphFrames 0.1.0 and Spark 1.6.

================================================================

PREREQUISITE

Run the following commands in any new terminal window before executing a file. This ensures that python 2 is used instead of the default python 3.

export SPARK\_HOME=/usr/local/spark-1.6.2-bin-hadoop2.6

export PYSPARK\_PYTHON=/usr/bin/python2

export PYSPARK\_DRIVER\_PYTHON=/usr/bin/python2

pip install pandas

================================================================

degree.py

This contains the requested degreedist function.

Usage:

$SPARK\_HOME/bin/pyspark --packages graphframes:graphframes:0.1.0-spark1.6 degree.py [filename [large]]

Example:

$SPARK\_HOME/bin/pyspark --packages graphframes:graphframes:0.1.0-spark1.6 degree.py ./stanford\_graphs/amazon.graph.large large

$SPARK\_HOME/bin/pyspark --packages graphframes:graphframes:0.1.0-spark1.6 degree.py ./stanford\_graphs/amazon.graph.small

Notes:

degreedist takes a single argument, a GraphFrame where we want to

calculate the degree distribution. It returns a DataFrame with two

columns, 'degree' and 'count'. For each degree, the count of nodes

with that degree is provided.

degreedist relies on the included function simple(g) which generates

a simple graph (with bidirectional edges) from the provided

GraphFrame

If the program is executed without parameters, it will execute

the degreedist function on four random graphs generated by

networkx.

The program takes up to two parameters: an input file and the

optional parameter "large". An input file is expected to have two

node names on each row -- a source node and a destination node --

separated by a delimited. If "large" is given, it assumes that

the input file's first line reports the node and edge count, and

also assumes that the delimiter is a space. If the second argument

is absent (or anything other than "large") it's assumed that all

lines represent edges and that the delimiter is a comma.

centrality.py

This program contains the requested closeness function.

Usage:

$SPARK\_HOME/bin/pyspark --packages graphframes:graphframes:0.1.0-spark1.6 centrality.py

Notes:

closeness takes a single argument, a GraphFrame whose nodes we want

to calculate the closeness centrality of. The returned DataFrame

has columns for 'id' and 'closeness', and lists the nodes in order

of highest centrality to lowest.

When executed, this script will generate the graph given in the

assignment and calculate its nodes' closeness centrality.

articulation.py

This program contains the requested articulations function.

Usage:

$SPARK\_HOME/bin/pyspark --packages graphframes:graphframes:0.1.0-spark1.6 articulation.py [filename]

Example:

$SPARK\_HOME/bin/pyspark --packages graphframes:graphframes:0.1.0-spark1.6 articulation.py 9\_11\_edgelist.txt

Notes:

articulations takes a GraphFrame as an argument as well as an

optional argument named "usegraphframe," which defaults to False.

If True, the articulation code will use GraphFrames to calculate

connected components, iterating through nodes serially. If False,

node iteration will take place using Spark's RDD.map() function, and

connectedness calculations will take place using networkx.

The program takes one argument, a filename containing an edge list.

The file is assumed to have two node names on each line separated

by a comma. At execution completion the program will display

the articulation points of the graph.

The program will run the articulations function on

the graph twice, once with each approach to articulation

calculation as described above. It will time each execution to

demonstrate which execution is faster. For 9\_11\_edgelist.txt, the

non-GraphFrames version is faster.