Çankaya University
Department of Computer Engineering

CENG 595 Distributed Data Processing and Analysis
Spring 2017

Homework 2

Due by Apr 4 (Tuesday) 6pm

Subject: Spark and graph problem

1) Download the following graph data graph.tsv from course web page.
   Each line includes a triple of the form <*source_url destination_url weight*> (tab separated).
2) For each url compute the out degree (number of outgoing edges) and output (node, count) pairs in sorted order by node (**outdegree**.scala or .py program).
3) For each url compute the sum of weights of incoming edges and output (node, weight_sum) pairs in sorted order by node (**weight**.scala or .py program).
4) For each node X, find a list of all other nodes Y such that there is an (X,Y) edge in the graph and a (Y,X) edge in the graph, and output (X, [Y1, Y2, …, Yn]) pairs in order sorted by X (**pairs**.scala or .py program).

   Develop the above graph processing algorithms in Apache Spark using Scala or Python languages.

   Submit your programs and output files as a single zipped file with your name (**asg2_lastname_firstname.zip**) via webonline. Output files should be named the same as the program, for example **outdegree.txt**.

Resources:

- Developing Spark programs in Eclipse: http://freecontent.manning.com/wp-content/uploads/how-to-start-developing-spark-applications-in-eclipse.pdf
- Spark download and quick start: http://spark.apache.org/docs/latest/index.html
- Spark programming guide: http://spark.apache.org/docs/latest/programming-guide.html
- Spark SQL: http://spark.apache.org/docs/latest/sql-programming-guide.html
- Mastering Apache Spark 2 Book: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/
  - https://github.com/jaceklaskowski/mastering-apache-spark-book

Solutions:

(2) For each url compute the out degree (number of outgoing edges) and output (node, count) pairs in sorted order by node.

```
var t = sc.textFile("../graph.tsv")
// graph nodes' outgoing edge count
t.map(line => (line.split("\t")(0),1)).reduceByKey(_+_)
  .sortBy(-_._2)
  .take(100).foreach(println)
```

```
Pseudo code algorithm:

map(k,v) {
  // input: k,v="src\t des\t weight"
  k'=v.split('\t')[0]  // src
  v'=1
  emit(k',v')
}
// in mapper combining is possible

// combine is possible

reduce(k,v[]) {
  // input: k:src, v=[1,1,...]
  k'=k
  sum=0
  for each x in v[]
    sum += x
  v'=sum
  emit(k',v')
}
```

(3) For each url compute the sum of weights of incoming edges and output (node, weight_sum) pairs in sorted order by node.

```
// graph nodes' incoming total weight
t.map(line => (line.split("\t")(1),line.split("\t")(2).toInt))
      .reduceByKey(_+_).sortBy(-_._2).take(100).foreach(println)

Pseudo code algorithm:

map(k,v) {
      // input: k,v="src\t des\t weight"
      // src --> des
  k'=v.split('\t')[1]  // dest
  v'=v.split('\t')[2]  // weight
  emit(k',v')
}

// in-mapper combining and combine possible

reduce(k,v[]) {
  // input: k:des, v=[2,6,5,...] (weight list)
  k'=k
  sum=0
  for each x in v[]
    sum += x
  v'=sum
  emit(k',v')
}
```

(4) For each node X, find a list of all other nodes Y such that there is an (X,Y) edge in the graph and a (Y,X) edge in the graph, and output (X, [Y1, Y2, …, Yn]) pairs in order sorted by X.

```
Pseudo code algorithm:
```

```
map(k,v) {
  // input: k,v="src\t des\t weight"
  // src -w-> des
  k'=v.split('\t')[0]  // src
  v'=v.split('\t')[1]  // des
  // in mapper combine
  emit(k',v')
  emit(v',k')
}

// in-mapper combining and combine are not possible

reduce(k,v[]) {
  // input: k:node, v=[1,1,3,4,4] (weight list)
  v=sort(v);
  k'=k
  for (int i=1; i<size(v); i++) {
     if (v[i]==v[i-1]) {
        v'=v[i]
        emit(k',v')
     }
  }
}
```