Lab Assignment 4

Author: Vendramin Nicolò

The solution code required to implement the transactions specified in the task is the following:

```
In [13]: var ranks = edges.mapValues(v => 1.0)

def splitScoreThroughNeighbours = (score: Double, neighbours: Iterable[String]) => {
    val list = neighbours.toList
    val length = list.length
    val newScore = score/length
    var results : List[(String, Double)] = List()
    list.foreach{x => results = ((x, newScore) :: results)}
    results
}

for (i <- 1 to iter) {
    val outgoing_edges = ranks.join(edges)
    .map(v => v._2)
    val transitions = outgoing_edges.flatMap(x => splitScoreThroughNeighbours(x._1, x._2))
    .reduceByKey((a,b) => a+b)
    ranks = transitions.map(x => (x._1, 0.15 + 0.85*x._2))
}
```

Producing as a result the following Ids of the top-10 ranked nodes:

```
In [24]: ranks.sortBy(_._2,false).
         take(10).map(x => x._1).
         foreach(println)

6634
2625
15
2398
4037
5412
4335
1297
2066
7553
```

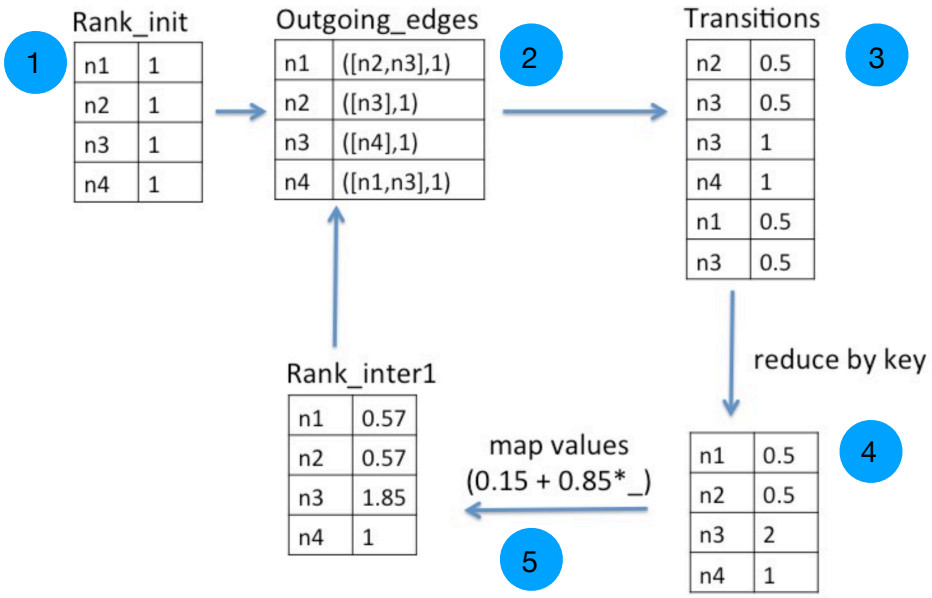
```
In [26]: ranks.sortBy(_._2, false).
         take(10).foreach(println)

(6634,2.8585544543148296)
(2625,2.6215622198034043)
(15,2.022126778253777)
(2398,1.8871610649967392)
(4037,1.848508491621217)
(5412,1.7455209978596167)
(4335,1.6822966318417538)
(1297,1.5686853085553163)
(2066,1.4989208771111784)
(7553,1.4958699460219822)
```

Respectively without and with the assigned scores. The Ids of the top ten ranked nodes in the considered data are:
6634, 2625, 15, 2398, 4037, 5412, 4335, 1297, 2066, 7553.

The code includes the following transformations:

- The vector with the ranks (initially set to be (nodeId, 1) for all the nodes) is joined to the edges vector to obtain the structure including (nodeId, (score, listOfNeighbours))
- The result is mapped discarding the nodeId and obtaining an RDD of (score, listOfNeighbours)
- A flat map is applied using the function splitScoreThroughNeighbours that counts the number of elements in listOfNeighbours, divides the score by that number (splittedScore = score / len(listOfNeighbours)) and returns one tuple for each of the neighbours in listOfNeighbours, containing (nodeId, splittedScore).

- This RDD (equivalent to the “transition” table present in the python notebook) is reduced by key summing the elements. In practice this aggregates all the scored-supports that the node received by other nodes at that step, and summer those scored-supports.
- Finally we update the ranks with the formula that was reported in the python notebook
- After repeating the procedure iter times we just operate on the resulting ranks RDD to sort it with descendent order, remove the score (in case we want to keep just node ids) and print the records.



With reference to this image, labelling the passages as reported above the code is mapped in this way:

```
var ranks = edges.mapValues(v => 1.0)

def splitScoreThroughNeighbours = (score: Double, neighbours: Iterable[String]) => {
    val list = neighbours.toList
    val length = list.length
    val newScore = score/length
    var results : List[(String, Double)] = List()
    list.foreach{x => results = ((x, newScore) :: results)}
    results
}

for (i <- 1 to iter) {
    val outgoing_edges = ranks.join(edges)
    .map(v => v._2)
    val transitions = outgoing_edges.flatMap(x => splitScoreThroughNeighbours(x._1, x._2))
    .reduceByKey((a,b) => a+b)
    ranks = transitions.map(x => (x._1, 0.15 + 0.85*x._2))
}
```

To resume, transitions is updated by splitting the rank of a node in equal parts among all the other nodes he links, and finally summing up for each nodes all those supports that he received. Ranks is simply updated at each round basing on transitions with the given formula (0.15 + 0.85*sum_of_supports).