# ID2221 : Data-Intensive Computing

Lab 4 - GraphX Hands on

**Solution code:**

## I. Reading the data file

*The code for the 1st part of the program is as follows:*

```
val iter=20
val lines = sc.textFile("data/wiki-Vote.txt")

// Construct outgoing edges from each vertex
val edges = lines.map{ s =>
    val parts = s.split("\\s+")
    (parts(0), parts(1))
  }.distinct().groupByKey().cache()

// For each edge, assign initial rank 1.0
var page_rank = edges.mapValues(v => 1.0)
```

The 2nd line of the code reads the input data file and produce a dataset of strings with each line in the file being one entire string within the RDD.In the following line of the code, the split command generates for each line (one entire string) an array with two elements. In the next line each of the two elements of the array are accessed and then used to produce a key/value pair. The last line in the code applies the groupByKey command on the key/value pair RDD to produce the edges RDD, which is also a key/value pair. This creates "page_rank" - a key/value pair RDD by taking the key from the edges RDD and assigning the value = 1.0 to it. page_rank is the initial ranks RDD and it is populated with the seed number 1.

## II. **Iterations and calculating ranks.**

The main part of the code is as follows:

```
val resprob: Double = 0.15
val damp_factor: Double = 0.85
for (i <- 1 to iter) {
  page_rank = edges
        // Associate the degree with each vertex
      .join(page_rank)
      .map{case(k,(out,page_rank))=>(out,page_rank/out.size) } //the ranks are
distributed equally amongst the various nodes.
      .flatMap{case(out,shares) =>out.map(v =>(v,shares))}
```

```
      .reduceByKey(_ + _) // use total value received from neighbours and have ranks for next iteration.
      .map{case(vid,total_vertices)=>(vid,resprob+damp_factor*total_vertices)} // Re-calculates vertex ranks based on neighbor contributions.

}

//Collects all vertex ranks and dump them in sorted order selecting top 10
page_rank.sortBy(_._2,false).take(10).foreach(v=>println(v))
```

This part is the core of the PageRank algorithm. In each iteration, the contributions are calculated and the ranks are recalculated based on those contributions. The algorithm has 4 steps:

1- Start the algorithm with each page at rank 1
2- Calculate node contribution: contribution = rank/size
3- Set each node new rank = 0.15 + 0.85 x contribution
4- Iterate to step 2 with the new rank

Firstly, the edges RDD and the page_rank RDD are joined together to form RDD1. In following line, RDD2 is flatmapped to generate the contribution RDD. Referring to the image given, in the first iteration, for example node1 references node2 and node3 so it contribution is 1/2 = 0.5 for each of the nodes it references. The rank is now calculated using 0.15 + 0.85 x contribution and then passed to next contribution cycle. After 20 iterations, it converges to following output:

```
(6634, 2.8542546924060384)
(2625, 2.6200812971121277)
(15,   2.020797230766344)
(2398, 1.8857540222427247)
(4037, 1.8472492089942458)
(5412, 1.7443111694374216)
(4335, 1.6812276862337627)
(1297, 1.5678663810839784)
(2066, 1.4979638628665695)
(7553, 1.494939884967576)
```

**********