# Big Data HW6

Tiago Antunes 2020280599

April 12, 2021

## 1 How to run the code

Create a folder called *checkpoints*. Then for both parts the code can be run by typing

```
spark-submit <filename> <input file>
```

in the terminal. Filename should be the respective part's filename. For part 1 it's word_count.py, for part 2 it's page_rank.py.

## 2 Part 1

For word count my solution is divided into the following steps:

1. Split each line into individual words
2. Flatten out these values to get 1 row per each word
3. Create (word, 1) pairs similar to map reduce
4. Reduce the values with a sum on the values, obtaining individual pairs (word, count)
5. Sort (the sorting key is -count because the sort is in increasing order)
6. Get the last 10 elements

The implementation is very straightforward. It takes 2.85s to run the file *pg100.txt*.

## 3 Part 2

I wrote the iterative version of page rank as the solution for this case. I start by generating the edges RDD, which required splitting and processing each line into (source, destination) with the values as integers. Then, to handle repeated

edges, I grouped them by the source and generated a unique list of destinations. This resulted in having a edges RDD with rows as

$$(source, (destination1, destination2, ...))$$

The number of nodes in the file is easily got by doing $edges.count()$ and this variable will be used in the computation part of the algorithm. The ranks is where the results of each iteration will be stored. It is initialized with an equal probability to each node, with the value $1.0/num\_nodes$

After this, a loop will repeat for the 100 iterations. This loop will compute the new scores for each node by:

1. Join the edges RDD with the ranks. This results in having each row with $(origin, ((dst1, dst2, ...), old_score))$
2. For each row calculate the score of the outgoing edge into each destination, resulting in a list of results (destination, new_score)
3. FlatMap the previous results and reduce them by key performing an addition. This will sum the values across all nodes thus getting the true real new score
4. Map the damping factor $0.8 * new\_score + 0.2/num\_nodes$ to each row on its score.
5. Assign new value to rank and repeat

In the end, the only requirement is to sort the values by the negative score, similar to Part 1. Then take the first 5 elements will give the top scores. During development I noticed that I was getting a stack overflow error, and it turns out it was due to the lineage being very long. To fix this, I make a checkpoint every 10 iterations which would cut the lineage by saving the RDD into disk, and avoids the problem.

The results for each file are presented in Table 1. They match the given example results in the pdf.

| File | Position | Node | Score | Time (s) |
|---|---|---|---|---|
| Small | 1 | 53 | 0.0357312 | 123.48 |
| | 2 | 14 | 0.0341709 | |
| | 3 | 40 | 0.0336301 | |
| | 4 | 1 | 0.0300059 | |
| | 5 | 27 | 0.0297201 | |
| Large | 1 | 4 | 0.0106669 | 501.99 |
| | 2 | 3 | 0.0094945 | |
| | 3 | 6 | 0.0062135 | |
| | 4 | 2 | 0.0054480 | |
| | 5 | 5 | 0.0053700 | |
| Full | 1 | 263 | 0.0020202 | 289.42 |
| | 2 | 537 | 0.0019433 | |
| | 3 | 965 | 0.0019254 | |
| | 4 | 245 | 0.0018526 | |
| | 5 | 187 | 0.0018273 | |

Table 1: Results for each file