# Page Ranking Algorithm

By Samruddhi Pasari (001306800)

# What is Page Ranking Algorithm?


- **Page Rank is an Algorithm used by Google search.**
- **PageRank calculate the rank of web pages based on the probability of the number of visits done by a random web surfer.**
- **It is an iterative graph processing algorithm.**

# Dataset

- The dataset includes Web graph from the Google programming contest, 2002.
- It contains 2 columns nodes and directed edges. Nodes represent web pages and directed edges represent hyperlinks between them.
- https://snap.stanford.edu/data/web-Google.html

| Column | Number of entries |
|---|---|
| Nodes | 875713 |
| Edges | 5105039 |

# Implementation

Using MapReduce for each functionality


- Creating Adjacency List
- Calculating Page Rank
- Handling Dangling nodes
- Finding Top 10 web pages

# Adjacency List

1. Creating vertex object to store page rank and adjacency list of each node
2. Emitting outgoing links to create adjacency list of each node from Mapper class
3. Calculating initial page rank and assigning it to all the vertices

```java
public class Vertex {

    private List<String> edgeList;
    private Double pageRank;

    public Vertex() {
    }

    public Vertex(List<String> edgeList, Double pageRank) {
        this.edgeList = edgeList;
        this.pageRank = pageRank;
    }
```

```
99841   1.0E-4  []
99844   1.0E-4  []
9985    1.0E-4  [103999]
99855   1.0E-4  [5005]
99859   1.0E-4  [8756]
9986    1.0E-4  [803800]
99861   1.0E-4  [133]
99868   1.0E-4  []
9987    1.0E-4  [849843,36292,120166,267090,290635,451024,564385,570790,684404]
99874   1.0E-4  [5942]
9988    1.0E-4  [834531,724601,659936,524670,410035,284441,264636,104746,73480,7074]
999     1.0E-4  [837478,915573]
9990    1.0E-4  [661878,819629,853185,196794]
9991    1.0E-4  [330937,510393,658533,738364,21682,177050]
99911   1.0E-4  [7646]
99916   1.0E-4  [4627,7431]
9992    1.0E-4  [807915,913376,470716]
99922   1.0E-4  []
99923   1.0E-4  []
99926   1.0E-4  []
9993    1.0E-4  [697731]
```

# Page Ranking

1. **Calculating new page rank, in Mapper, for all the outgoing links of a node by equally dividing the page rank of that node among the outgoing links**

   **p = N.pageRank / N.adjacencyList.size()**

1. **Calculating page rank in Reducer using the following formula**

- $P(n) = (1-\alpha)\frac{1}{|V|} + \alpha \sum_{m \in L(n)} \frac{P(m)}{C(m)}$

The variables in the formula have the following meaning:

- |V| is the number of pages (vertices) in the Web graph considered.
- $\alpha$ is the probability of the surfer following a link; 1-$\alpha$ the probability of making a random jump.
- L(n) is the set of all pages in the graph linking to n.
- P(m) is the PageRank of another page m.
- C(m) is the out-degree of page m, i.e., the number of links on that page.

# Limitations?

- **What will happen to the nodes with no outgoing links?**
- **Does these nodes affect the page rank calculation?**
- **Such nodes are called dangling nodes and they do affect the page rank**

# Handling Dangling Nodes

1. **Adding dangling nodes with empty adjacency list in Stage 1 Mapper and emitting page rank values of those nodes for delta calculation with dummy key**

2. **Calculating delta by adding all the page rank value of dangling nodes and summation of incoming page ranks in Stage 1 Reducer**

3. **Recalculating page ranks in Stage 2 Mapper (Map only job) with delta set as Counter in DriverClass**

$$P(n) = (1-\alpha)\frac{1}{|V|} + \alpha\left(\frac{\delta}{|V|} + \sum_{m \in L(n)} \frac{P(m)}{C(m)}\right)$$

# Analysis

# TopK Algorithm

- Used TopK Algorithm to calculate top 10 web pages based on the page rank we calculated.
- Out of two Convergence Criteria for calculating page ranking:
  - Iterating until page rank value convergence
  - Iterating until page rank ranking unchanged
  - Fixed Iterations
- This analysis is based on Fixed iteration

For 3 iterations

```
hadoopusr@ubuntu:~$ hdfs dfs -cat /Project/pageRank/topK/part-r-00000
20/08/14 09:47:15 WARN util.NativeCodeLoader: Unable to load native-hadc
0.14686566865683623     7314
0.11759180977156147     8316
0.040142025617284686    1536
0.030158119261188453    3054
0.027431929656477432    9810
0.026676122001262736    3170
0.02482882751101009     9533
0.024117484023687512    1532
0.023574856016006486    6236
0.019038140924691596    280
```

For 10 iterations

```
hadoopusr@ubuntu:~$ hdfs dfs -cat /Project/pageRank/topK/part-r-00000
20/08/14 09:59:57 WARN util.NativeCodeLoader: Unable to load native-ha
0.0668477486397055      7314
0.054582511164657795    8316
0.01840043578532789     1536
0.015189470226104295    3054
0.015058302619399572    6236
0.01410764155372648     3170
0.012781427043296358    9810
0.01154250112158667     9533
0.011233407495713816    1532
0.00960421230442097     894399
hadoopusr@ubuntu:~$
```

# Future Scope

- **The Algorithm can be executed on AWS with more resources to support bigger web graphs.**
- **Implementing the algorithm in one stage instead of two stages**
- **Implementing alternative convergence criterias:**
  - **Iterating until page rank value convergence**
  - **Iterating until page rank ranking unchanged**

# Conclusion

The Page Ranking Algorithm was implemented using 2 Stage Mapreduce. It is observed that for Fixed iteration method we achieved better convergence with higher iteration counts.

# Thank You!!