Exercise 3: PageRank

**How to Run the code:**

Programme file name is PageRank.java

Javac PageRank.java

Java PageRank input\_path\_file, output\_path, [iteration times]

**Argument Declaration:**

The first argument(input\_path\_file) is for the input file.

The second argument(output\_path) is for the directory of the output, not the file.

The third argument is optional, and square brackets are not included.

If the third argument is used, this number determines iterations of PageRank calculation. If this argument is not typed, the calculation will stop when the distance is small enough, I use 0.000001 as the ending distance.

**The method how to implement the programme.**

Firstly, Hadoop is used to read the original file to reduces the lines of the input. Put the all the toNodeIds in oneline led by a fromNodeId.

So the donation to each toNodeId can calculate easily, which is 1 / (length of array split by “ ” – 1), because the first element of this array is a fromNodeId.

Use a SortedMap<Integer, Map<Integer, Double>> to make a triple for a sparse matric.

One Integer class to store toNodeIds, as rows in an adjacency matrix, another one is to store fromNodesIds, as columns in an adjacency matrix. The double is to store the value as the value in an adjacency matrix. So that the we use Map data structure like this, Map<ToNodeId, Map<FromNodeId, donated value>>

To avoid dead-end, if the a fromNodeId is not appeared in the row/toNodeIds, this id will be added with an empty SortedMap. It will be used in calculation.

We don’t need to add toNodeId in fromNodeIds, because it won’t affect the result.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 |
| 1 | 0 | 0 | d | f |
| 2 | a | 0 | 0 | 0 |
| 3 | 0 | b | 0 | 0 |
| 4 | a | b | 0 | f |
| 5 | a | 0 | d | 0 |
| 6 | 0 | 0 | 0 | 0 |

Illustration for how to implement the sparse matrix:

Using Map<K, V> in a Map<K, V> can represent this table,

<1, <4, d>>, <1, <6, f>>,

<2, <1, a>>,

<3, <2, b>>,

<4, <1, a>>, <4, <2, b>>, <4, <6, f>>,

<5, <1, a>>, <5, <4, d>>.

Node 3 and 5 won’t go to anywhere, which are a dead end.

And there is not any node going to 6, which will be zero unless we use a factor to increase its PR value.

We assign same value to each node, which is 1/N, N equals to the amount of the nodes.

It can be represented by a Map<ToNodeId, 1/N> R .

**Calculation**

In calculation, we can use an iterator of Map to finish the multiplication.

For each row, column or FromNodeId can be used as a key to look for the PR value that need to be multiplied in R. And we use map instead of an array is that the ToNodeId could not a consecutive number, Array may waste lots of memory.

The reason why we traverse Map instead of Map R is to reduce the calculation scale. Only those nodes those have a value need to be calculated, which the number of multiplication is less than N or map.size(). If we traverse array R, we will calculate N times, as long as the R.size().

The sum of those results of multiplications is the new PR value for that toNodeId/ row.

And the fromNodeId also bind the key of the Map R, so we don’t need worry about calculating wrong.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 |
| 1 | 0 | 0 | d | f |
| 2 | a | 0 | 0 | 0 |
| 3 | 0 | b | 0 | 0 |
| 4 | a | b | 0 | f |
| 5 | a | 0 | d | 0 |
| 6 | 0 | 0 | 0 | 0 |

|  |  |
| --- | --- |
| K | V |
| 1 | 1/6 |
| 2 | 1/6 |
| 3 | 1/6 |
| 4 | 1/6 |
| 5 | 1/6 |
| 6 | 1/6 |

We use FromNodeId to allocate the key of the R.

In the 2 tables on the left, there is not 3 or 5 in the column(fromNodeId), we don’t need to deal with it. It is like Node 1 and 2, 0 multiply anything is zero.

**Formular**

To avoid the effect from spider trap and dead end. We add a Damping Factor D which equal to 0.85 and use another factor (1-D)/N to increase the pr value of those without any inlinks.

So that, the new value can be calculated by this formular:

.

For the distance of two values can be calculated by this formular:

The ending value we choose is 0.000001.

Once the distance is less than this value, we image the value is stable.