BIG DATA PROGRAMING

ASSIGNMENT-4

1. The Page rank algorithm:

Page rank algorithm in general used to find the rank of web pages such that the most ranked pages are suggested when search engine used to search for the content on the browser.

The Page rank algorithm is a iterative algorithm and In each iteration rank values are computes and updated with previous values until the change in values from present and previous iteration is around 10^-6, i.e. the values are very similar with minute error.

Based on the given graph the rank values are computed using the standard page rank formula. And the generalized version for computing ranks is shown below.

$$\mathbf{r} = c\mathbf{P}^\top\mathbf{r} + (1 - c)/n$$

In each iteration we take the updated page rank RDD and use it for following iterations. This type of iteration is also called as power iteration

$$\mathbf{r}^t = c \cdot \mathbf{P}^\top \cdot \mathbf{r}^{t-1} + (1 - \mathrm{c})\mathbf{1}/n$$

r_i^t : PageRank value of node i at iteration t

$$\begin{aligned}
r_1^t &= c \cdot r_4^{t-1} \cdot p_{4,1} + (1 - c)/5 \\
r_2^t &= c \cdot r_1^{t-1} \cdot p_{1,2} + (1 - c)/5 \\
r_3^t &= c \cdot r_2^{t-1} \cdot p_{2,3} + c \cdot r_5^{t-1} \cdot p_{5,3} + (1 - c)/5 \\
r_4^t &= c \cdot r_2^{t-1} \cdot p_{2,4} + c \cdot r_3^{t-1} \cdot p_{3,4} + (1 - c)/5 \\
r_5^t &= c \cdot r_4^{t-1} \cdot p_{4,5} + (1 - c)/5
\end{aligned}$$

2. Implementation of Page rank algorithm in pyspark. As a first step the adjacent list text file provided is read into the spark RDD. In which each line of the text file is stored as String in Adjlist1.

Adjlist1 consists of ['1 2', '2 3 4', '3 4', '4 1 5', '5 3'] Here clearly see that each line of adjacency list is read as a string and all the lines are strings of lines.

Further this RDD is used to produce another RDD named Adjlist2 in which the RDD element i.e each line is tokenized using MAP and split operations after tokenizing all the lines. After this each tokenized line is used to compute transition probability of the nodes that are present each index of the adjacency list (1/total number of nodes = transition probability). The transition probability is one divided by number of outward edges from a node. And the node that pointing to other nodes are stored along with their transition probability separately using flap map transformation as show below

[(2, [1.0, 1.0]), (3, [2.0, 0.5]), (4, [2.0, 0.5]), (4, [3.0, 1.0]), (1, [4.0, 0.5]), (5, [4.0, 0.5]), (3, [5.0, 1.0])] the Adjlist 2

In each tuple above the first element is the node the is used to compute the rank of it as shown above the second list first element is the edge connection to this node and second element is the transition probability or weight on the edge.

For example:

Rank of node2 = c*rank of node 1 * transition probability (node 1 to 2) + (1-c)/number of nodes

As a Next step we Group the RDD elements in second step by using keys as the first element and sort them such that all the element with same node key are stored in the list contain that with its edges connections and transition probabilities.

The output will look like as shown below in Adjlist3 and from here the number of nodes mentioned in the above formulae are stores in a variable using count method of RDD.

[[1, [[4.0, 0.5]]], [2, [[1.0, 1.0]]], [3, [[2.0, 0.5], [5.0, 1.0]]], [4, [[2.0, 0.5], [3.0, 1.0]]], [5, [[4.0, 0.5]]]]

Initial page rank values are assigned as 0.2 and this a RDD is created such that it is used for updating after every iteration in the page rank algorithm and this is one of the main process of power iteration.

And this RDD is created in such a way that it uses the adjlist3 and Map values operation with a value of 0.2 for each map. And during this the index are taken from the above step i.e. the first element of the adjlist3, the output looks like as shown below.

[(1, 0.2), (2, 0.2), (3, 0.2), (4, 0.2), (5, 0.2)]

The RDD in strep 3 is used to rearrange in such a way that we replace the index or the first value with the index values in each list such that using flat map the output looks like

```
#[[4.0, [0.5, 1]], [1.0, [1.0, 2]], [2.0, [0.5, 3]], [5.0, [1.0, 3]], [2.0, [0.5,
 4]], [3.0, [1.0, 4]], [4.0, [0.5, 5]]]
```

And this RDD is used for further for the entire iteration process so it is persisted for fast processing.

In the iteration for loop there are three main step joining the page ranks rdd with above rdd , and the result look like below, the last element is the rank

```
    #[(4.0, ([0.5, 1], 0.2)), (4.0, ([0.5, 5], 0.2)), (2.0, ([0.5, 3], 0.2)), (2.
0, ([0.5, 4], 0.2)), (1.0, ([1.0, 2], 0.2)), (5.0, ([1.0, 3], 0.2)), (3.0, ([1.0,
 4], 0.2))]
```

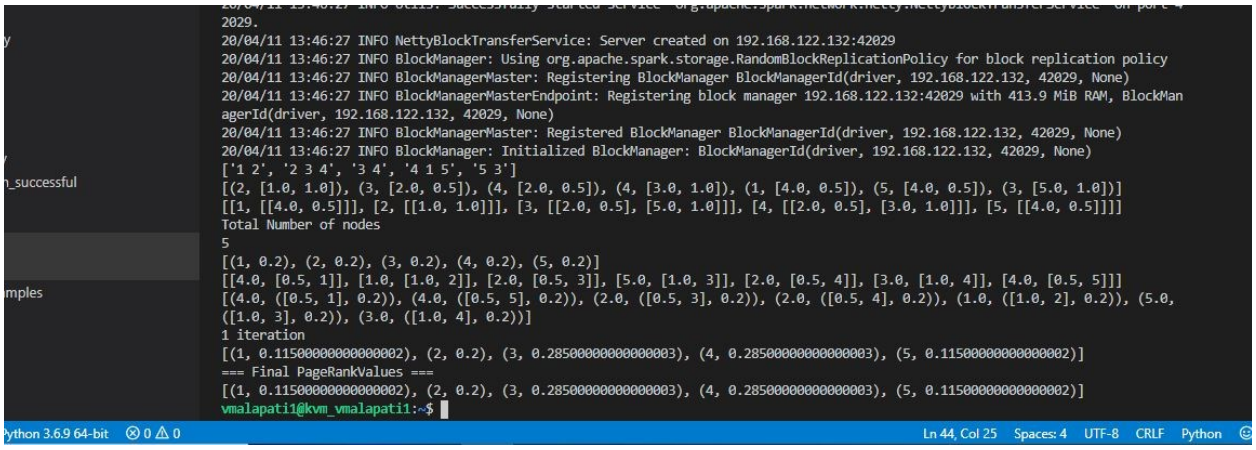
This above RDD is used to compute the new page rank values and this process keeps on iterating until the errors are less than 0.0000001 from the last iteration

The computation of new page rank is done similar to the process of Map Reduce in Hadoop. As of it primarily step is to compute the product of page rank of previous iteration with transition probability and then using reduced by key method to sum these product values of same key and finally these summed values are multiplied by constant c and add with (1-c)/number of nodes. And finally sort by key to get the page rank values a shown below.

The final page rank values are as shown below

[(1, 0.11500000000000002), (2, 0.2), (3, 0.28500000000000003), (4, 0.28500000000000003), (5, 0.11500000000000002)]

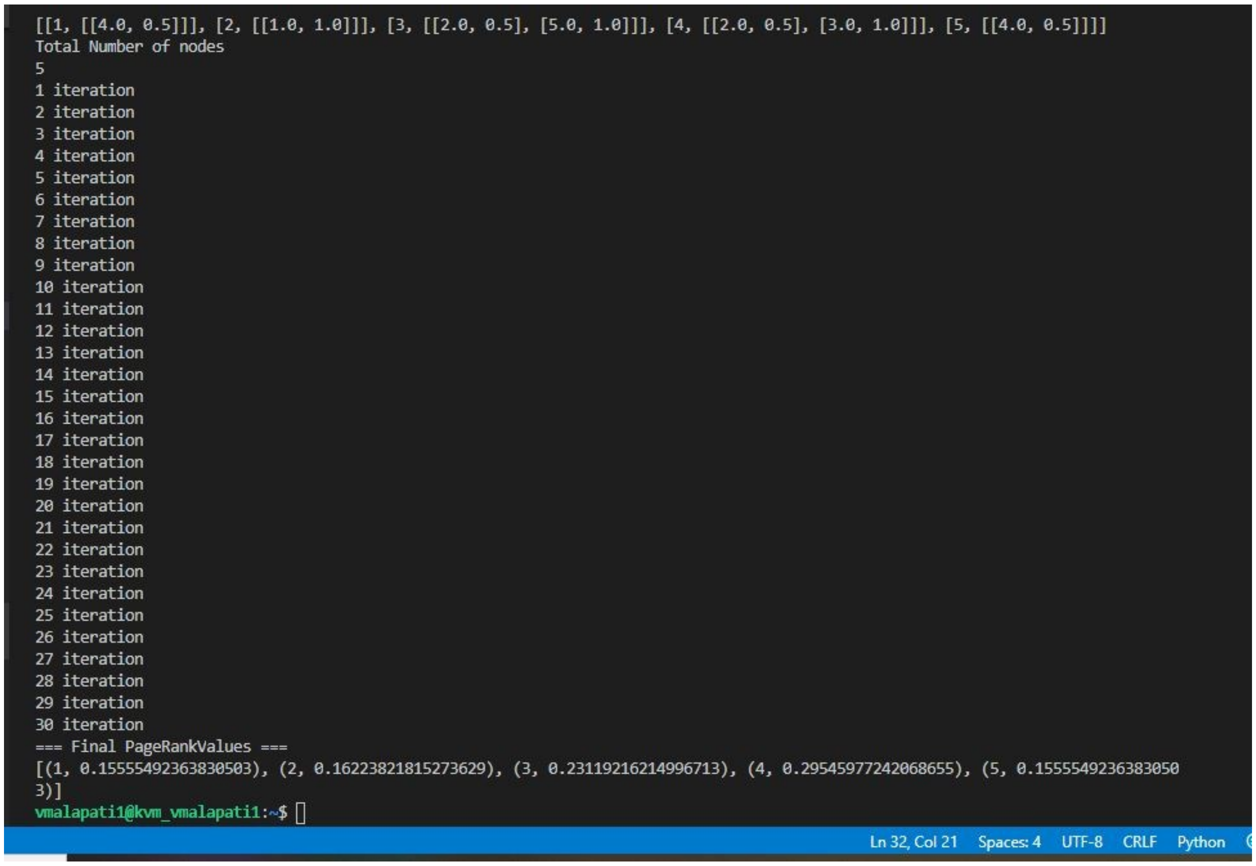
2. The screens shots of output of each step after iteration-1 are shown below



```
2029.
20/04/11 13:46:27 INFO NettyBlockTransferService: Server created on 192.168.122.132:42029
20/04/11 13:46:27 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
20/04/11 13:46:27 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.122.132, 42029, None)
20/04/11 13:46:27 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.122.132:42029 with 413.9 MiB RAM, BlockMan
agerId(driver, 192.168.122.132, 42029, None)
20/04/11 13:46:27 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.122.132, 42029, None)
20/04/11 13:46:27 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 192.168.122.132, 42029, None)
['1 2', '2 3 4', '3 4', '4 1 5', '5 3']
[(2, [1.0, 1.0]), (3, [2.0, 0.5]), (4, [2.0, 0.5]), (4, [3.0, 1.0]), (1, [4.0, 0.5]), (5, [4.0, 0.5]), (3, [5.0, 1.0])]
[[1, [[4.0, 0.5]]], [2, [[1.0, 1.0]]], [3, [[2.0, 0.5], [5.0, 1.0]]], [4, [[2.0, 0.5], [3.0, 1.0]]], [5, [[4.0, 0.5]]]]
Total Number of nodes
5
[(1, 0.2), (2, 0.2), (3, 0.2), (4, 0.2), (5, 0.2)]
[[4.0, [0.5, 1]], [1.0, [1.0, 2]], [2.0, [0.5, 3]], [5.0, [1.0, 3]], [2.0, [0.5, 4]], [3.0, [1.0, 4]], [4.0, [0.5, 5]]]
[(4.0, ([0.5, 1], 0.2)), (4.0, ([0.5, 5], 0.2)), (2.0, ([0.5, 3], 0.2)), (2.0, ([0.5, 4], 0.2)), (1.0, ([1.0, 2], 0.2)), (5.0,
([1.0, 3], 0.2)), (3.0, ([1.0, 4], 0.2))]
1 iteration
[(1, 0.11500000000000002), (2, 0.2), (3, 0.28500000000000003), (4, 0.28500000000000003), (5, 0.11500000000000002)]
=== Final PageRankValues ===
[(1, 0.11500000000000002), (2, 0.2), (3, 0.28500000000000003), (4, 0.28500000000000003), (5, 0.11500000000000002)]
vmalapati1@kvm_vmalapati1:~$
```

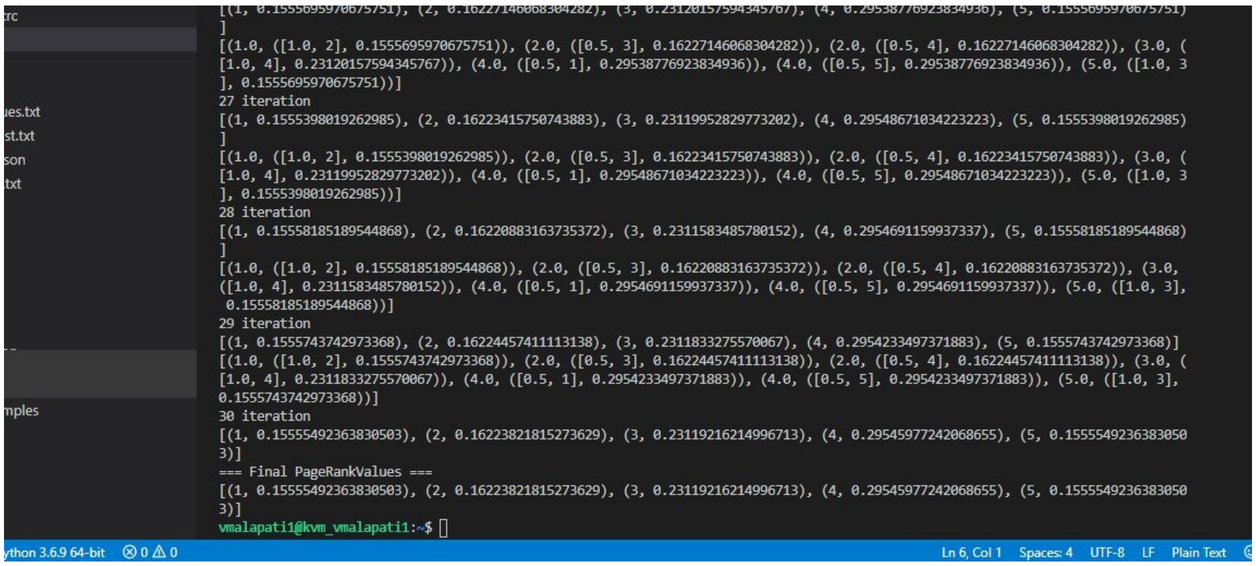
_successful
mples
Python 3.6.9 64-bit ⊗ 0 ⚠ 0
Ln 44, Col 25 Spaces: 4 UTF-8 CRLF Python

The Output computed for 30 iterations are shown below:

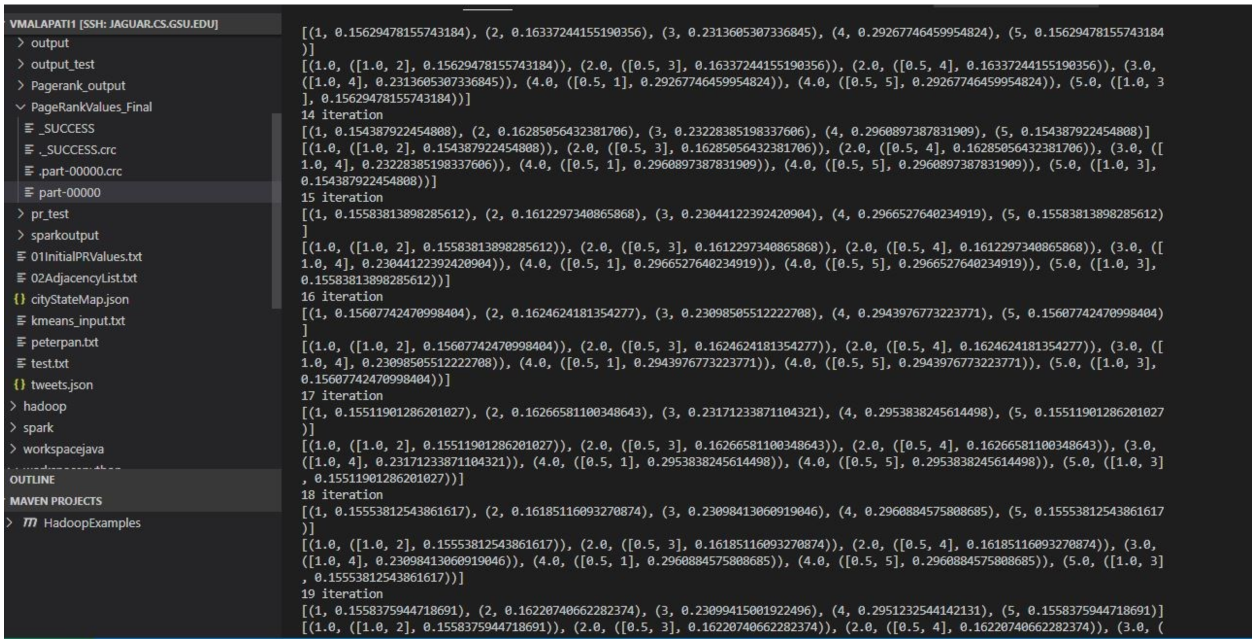


```
[[1, [[4.0, 0.5]]], [2, [[1.0, 1.0]]], [3, [[2.0, 0.5], [5.0, 1.0]]], [4, [[2.0, 0.5], [3.0, 1.0]]], [5, [[4.0, 0.5]]]]
Total Number of nodes
5
1 iteration
2 iteration
3 iteration
4 iteration
5 iteration
6 iteration
7 iteration
8 iteration
9 iteration
10 iteration
11 iteration
12 iteration
13 iteration
14 iteration
15 iteration
16 iteration
17 iteration
18 iteration
19 iteration
20 iteration
21 iteration
22 iteration
23 iteration
24 iteration
25 iteration
26 iteration
27 iteration
28 iteration
29 iteration
30 iteration
=== Final PageRankValues ===
[(1, 0.15555492363830503), (2, 0.16223821815273629), (3, 0.23119216214996713), (4, 0.29545977242068655), (5, 0.1555549236383050
3)]
vmalapati1@kvm_vmalapati1:~$ 
```

The results after 30 iteration with using print statements after each and every step

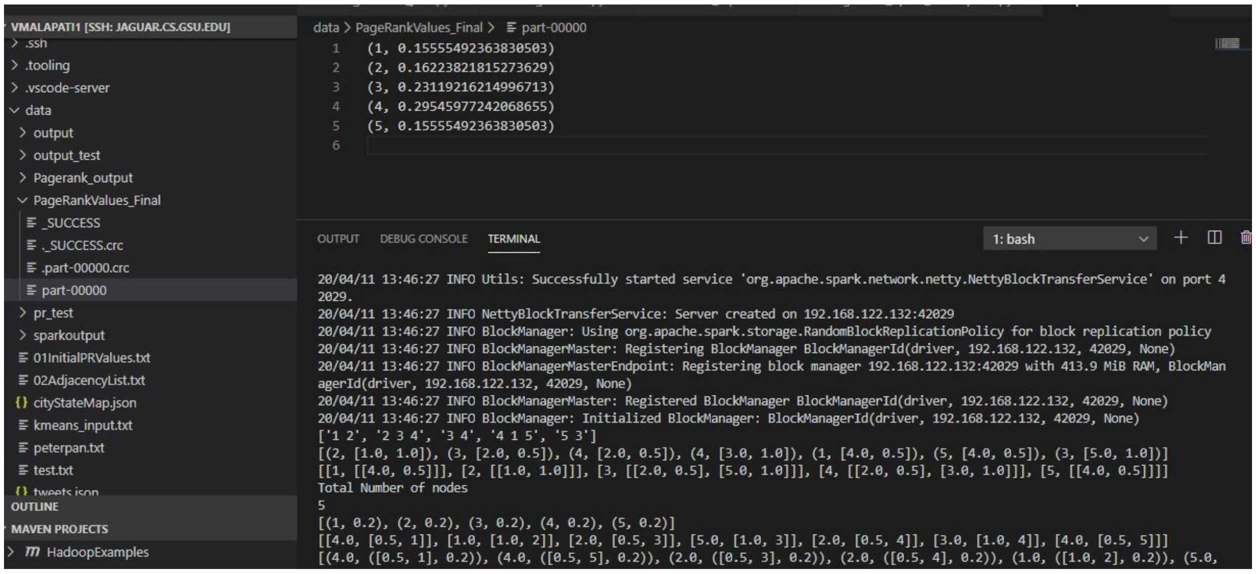


```
[(1, 0.1555695970675751), (2, 0.16227146068304282), (3, 0.23120157594345767), (4, 0.29538776923834936), (5, 0.1555695970675751)
]
[(1.0, ([1.0, 2], 0.1555695970675751)), (2.0, ([0.5, 3], 0.16227146068304282)), (2.0, ([0.5, 4], 0.16227146068304282)), (3.0, (
[1.0, 4], 0.23120157594345767)), (4.0, ([0.5, 1], 0.29538776923834936)), (4.0, ([0.5, 5], 0.29538776923834936)), (5.0, ([1.0, 3
], 0.1555695970675751))]
27 iteration
[(1, 0.1555398019262985), (2, 0.16223415750743883), (3, 0.23119952829773202), (4, 0.29548671034223223), (5, 0.1555398019262985)
]
[(1.0, ([1.0, 2], 0.1555398019262985)), (2.0, ([0.5, 3], 0.16223415750743883)), (2.0, ([0.5, 4], 0.16223415750743883)), (3.0, (
[1.0, 4], 0.23119952829773202)), (4.0, ([0.5, 1], 0.29548671034223223)), (4.0, ([0.5, 5], 0.29548671034223223)), (5.0, ([1.0, 3
], 0.1555398019262985))]
28 iteration
[(1, 0.15558185189544868), (2, 0.16220883163735372), (3, 0.2311583485780152), (4, 0.2954691159937337), (5, 0.15558185189544868)
]
[(1.0, ([1.0, 2], 0.15558185189544868)), (2.0, ([0.5, 3], 0.16220883163735372)), (2.0, ([0.5, 4], 0.16220883163735372)), (3.0,
([1.0, 4], 0.2311583485780152)), (4.0, ([0.5, 1], 0.2954691159937337)), (4.0, ([0.5, 5], 0.2954691159937337)), (5.0, ([1.0, 3],
 0.15558185189544868))]
29 iteration
[(1, 0.1555743742973368), (2, 0.16224457411113138), (3, 0.2311833275570067), (4, 0.2954233497371883), (5, 0.1555743742973368)]
[(1.0, ([1.0, 2], 0.1555743742973368)), (2.0, ([0.5, 3], 0.16224457411113138)), (2.0, ([0.5, 4], 0.16224457411113138)), (3.0, (
[1.0, 4], 0.2311833275570067)), (4.0, ([0.5, 1], 0.2954233497371883)), (4.0, ([0.5, 5], 0.2954233497371883)), (5.0, ([1.0, 3],
0.1555743742973368))]
30 iteration
[(1, 0.15555492363830503), (2, 0.16223821815273629), (3, 0.23119216214996713), (4, 0.29545977242068655), (5, 0.1555549236383050
3)]
=== Final PageRankValues ===
[(1, 0.15555492363830503), (2, 0.16223821815273629), (3, 0.23119216214996713), (4, 0.29545977242068655), (5, 0.1555549236383050
3)]
vmalapati1@kvm_vmalapati1:~$ 
```



```
[(1, 0.15629478155743184), (2, 0.16337244155190356), (3, 0.2313605307336845), (4, 0.29267746459954824), (5, 0.15629478155743184
)]
[(1.0, ([1.0, 2], 0.15629478155743184)), (2.0, ([0.5, 3], 0.16337244155190356)), (2.0, ([0.5, 4], 0.16337244155190356)), (3.0,
([1.0, 4], 0.2313605307336845)), (4.0, ([0.5, 1], 0.29267746459954824)), (4.0, ([0.5, 5], 0.29267746459954824)), (5.0, ([1.0, 3
], 0.15629478155743184))]
14 iteration
[(1, 0.154387922454808), (2, 0.16285056432381706), (3, 0.23228385198337606), (4, 0.2960897387831909), (5, 0.154387922454808)]
[(1.0, ([1.0, 2], 0.154387922454808)), (2.0, ([0.5, 3], 0.16285056432381706)), (2.0, ([0.5, 4], 0.16285056432381706)), (3.0, ([
1.0, 4], 0.23228385198337606)), (4.0, ([0.5, 1], 0.2960897387831909)), (4.0, ([0.5, 5], 0.2960897387831909)), (5.0, ([1.0, 3],
0.154387922454808))]
15 iteration
[(1, 0.15583813898285612), (2, 0.1612297340865868), (3, 0.23044122392420904), (4, 0.2966527640234919), (5, 0.15583813898285612)
]
[(1.0, ([1.0, 2], 0.15583813898285612)), (2.0, ([0.5, 3], 0.1612297340865868)), (2.0, ([0.5, 4], 0.1612297340865868)), (3.0, ([
1.0, 4], 0.23044122392420904)), (4.0, ([0.5, 1], 0.2966527640234919)), (4.0, ([0.5, 5], 0.2966527640234919)), (5.0, ([1.0, 3],
0.15583813898285612))]
16 iteration
[(1, 0.15607742470998404), (2, 0.1624624181354277), (3, 0.23098505512222708), (4, 0.2943976773223771), (5, 0.15607742470998404)
]
[(1.0, ([1.0, 2], 0.15607742470998404)), (2.0, ([0.5, 3], 0.1624624181354277)), (2.0, ([0.5, 4], 0.1624624181354277)), (3.0, ([
1.0, 4], 0.23098505512222708)), (4.0, ([0.5, 1], 0.2943976773223771)), (4.0, ([0.5, 5], 0.2943976773223771)), (5.0, ([1.0, 3],
0.15607742470998404))]
17 iteration
[(1, 0.15511901286201027), (2, 0.16266581100348643), (3, 0.23171233871104321), (4, 0.2953838245614498), (5, 0.15511901286201027
)]
[(1.0, ([1.0, 2], 0.15511901286201027)), (2.0, ([0.5, 3], 0.16266581100348643)), (2.0, ([0.5, 4], 0.16266581100348643)), (3.0,
([1.0, 4], 0.23171233871104321)), (4.0, ([0.5, 1], 0.2953838245614498)), (4.0, ([0.5, 5], 0.2953838245614498)), (5.0, ([1.0, 3]
, 0.15511901286201027))]
18 iteration
[(1, 0.15553812543861617), (2, 0.16185116093270874), (3, 0.23098413060919046), (4, 0.2960884575808685), (5, 0.15553812543861617
)]
[(1.0, ([1.0, 2], 0.15553812543861617)), (2.0, ([0.5, 3], 0.16185116093270874)), (2.0, ([0.5, 4], 0.16185116093270874)), (3.0,
([1.0, 4], 0.23098413060919046)), (4.0, ([0.5, 1], 0.2960884575808685)), (4.0, ([0.5, 5], 0.2960884575808685)), (5.0, ([1.0, 3]
, 0.15553812543861617))]
19 iteration
[(1, 0.1558375944718691), (2, 0.16220740662282374), (3, 0.23099415001922496), (4, 0.2951232544142131), (5, 0.1558375944718691)]
[(1.0, ([1.0, 2], 0.1558375944718691)), (2.0, ([0.5, 3], 0.16220740662282374)), (2.0, ([0.5, 4], 0.16220740662282374)), (3.0, (
```

The Page rank values after 30 iterations are stored in the output file are shown below:



```
(1, 0.15555492363830503)
(2, 0.16223821815273629)
(3, 0.23119216214996713)
(4, 0.29545977242068655)
(5, 0.15555492363830503)
```

```
20/04/11 13:46:27 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 4
2029.
20/04/11 13:46:27 INFO NettyBlockTransferService: Server created on 192.168.122.132:42029
20/04/11 13:46:27 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
20/04/11 13:46:27 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.122.132, 42029, None)
20/04/11 13:46:27 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.122.132:42029 with 413.9 MiB RAM, BlockMan
agerId(driver, 192.168.122.132, 42029, None)
20/04/11 13:46:27 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.122.132, 42029, None)
20/04/11 13:46:27 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 192.168.122.132, 42029, None)
['1 2', '2 3 4', '3 4', '4 1 5', '5 3']
[(2, [1.0, 1.0]), (3, [2.0, 0.5]), (4, [2.0, 0.5]), (4, [3.0, 1.0]), (1, [4.0, 0.5]), (5, [4.0, 0.5]), (3, [5.0, 1.0])]
[[1, [[4.0, 0.5]]], [2, [[1.0, 1.0]]], [3, [[2.0, 0.5], [5.0, 1.0]]], [4, [[2.0, 0.5], [3.0, 1.0]]], [5, [[4.0, 0.5]]]]
Total Number of nodes
5
[(1, 0.2), (2, 0.2), (3, 0.2), (4, 0.2), (5, 0.2)]
[[4.0, [0.5, 1]], [1.0, [1.0, 2]], [2.0, [0.5, 3]], [5.0, [1.0, 3]], [2.0, [0.5, 4]], [3.0, [1.0, 4]], [4.0, [0.5, 5]]]
[(4.0, ([0.5, 1], 0.2)), (4.0, ([0.5, 5], 0.2)), (2.0, ([0.5, 3], 0.2)), (2.0, ([0.5, 4], 0.2)), (1.0, ([1.0, 2], 0.2)), (5.0,
```

The Final output results are same the given result in the assignment file and from these we can say that for the given graph page 4 and page 3 has more rank these will be given high priority when searched in a search engine.

Note: I have commented the code for storing the final iteration results and also uncomment it, I have also implemented code for writing the output as txt final which is also commented.