**Ronaldlee Ejalu**

# CSC 555 Mining Big Data

Assignment 5

**Due Sunday, November 7th**

1) Identify which type of database/data processing system you would choose (Key-Value store, Column-oriented store, Document-oriented store, Graph database, Relational database, Streaming engine) in each scenario below.

   a) Highly structured multi-table data that requires enforcing data constraints.
      - Relational database

   b) Stock market data ticker with decisions that must be made in real time.
      - Streaming engine.

   c) LinkedIn type data with interconnected nodes where much of the information resides in the links between nodes.
      - Graph database

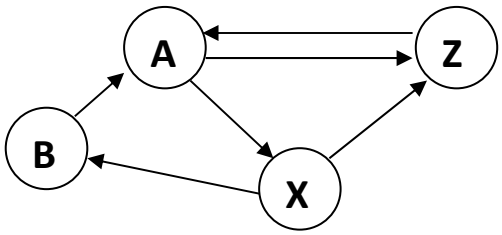
   d) An image storage system that allows lookup images by file name.
      - Key-Value store

   e) A collection of JSON objects (e.g., tweets).
      - Document-oriented store.

   f) Data that is stored in large sparse tables that are continuously growing (new rows/columns).
      - Column-oriented store.

2)

   a) Consider the following graph




   Compute the page rank for the nodes in this graph. If you are multiplying matrices manually, you may stop computing after 5 steps. If you use a tool (e.g., Matlab, python) for matrix multiplication, you should get your answer to converge.

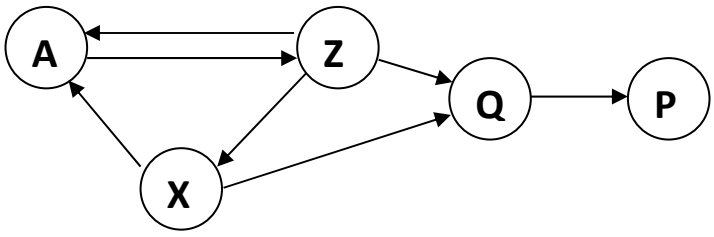
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | A | * | 1/4 | = | 52429/131072 | |
| 0 | 0 | 1/2 | 0 | B | | 1/4 | | 52429/524288 | |
| 1/2 | 0 | 0 | 0 | X | | 1/4 | | 104857/524288 | |
| 1/2 | 0 | 1/2 | 0 | Z | | 1/4 | | 78643/262144 | |
| A | B | X | Z | | | | | | |

```
C: > Users > rejalu1 > OneDrive - Henry Ford Health System > CSC555MiningBigData > python files > pageRankPartA.py > ...
1  import numpy as np
2  import fractions
3  np.set_printoptions(formatter={'all':lambda x: str(fractions.Fraction(x).limit_denominator())})
4
5  m = [[0, 1, 0, 1], [0, 0, 1/2, 0], [1/2, 0, 0, 0], [1/2, 0, 1/2, 0]]
6  M = np.array(m)
7  # print(M.shape)
8
9  v = [[1/4], [1/4], [1/4], [1/4]]
10 v = np.array(v)
11 # print(v)
12
13 mult1 = np.dot(M, v)
14 mult2 = np.dot(M, mult1)
15
16 for num in range(1, 100):
17     if np.allclose(mult1, mult2) == True:
18         break # Once they converge, break
19     else:
20         mult1 = mult2
21         mult2 = np.dot(M, mult1)
22 print('The ranks converge at the %d iteration with the following ranks %s' %(num, mult2))
23
```

```
PROBLEMS 2   OUTPUT   TERMINAL

DEBUG CONSOLE

TERMINAL

PS C:\Users\rejalu1> & C:/ProgramData/Anacoda32019/python.exe "c:/Users/rejalu1/OneDrive
 - Henry Ford Health System/CSC555MiningBigData/python files/pageRankPartA.py"
The ranks converge at the 32 iteration with the following ranks [[52429/131072]
 [52429/524288]
 [104857/524288]
 [78643/262144]]

PS C:\Users\rejalu1>
```

It converges at the 32 step

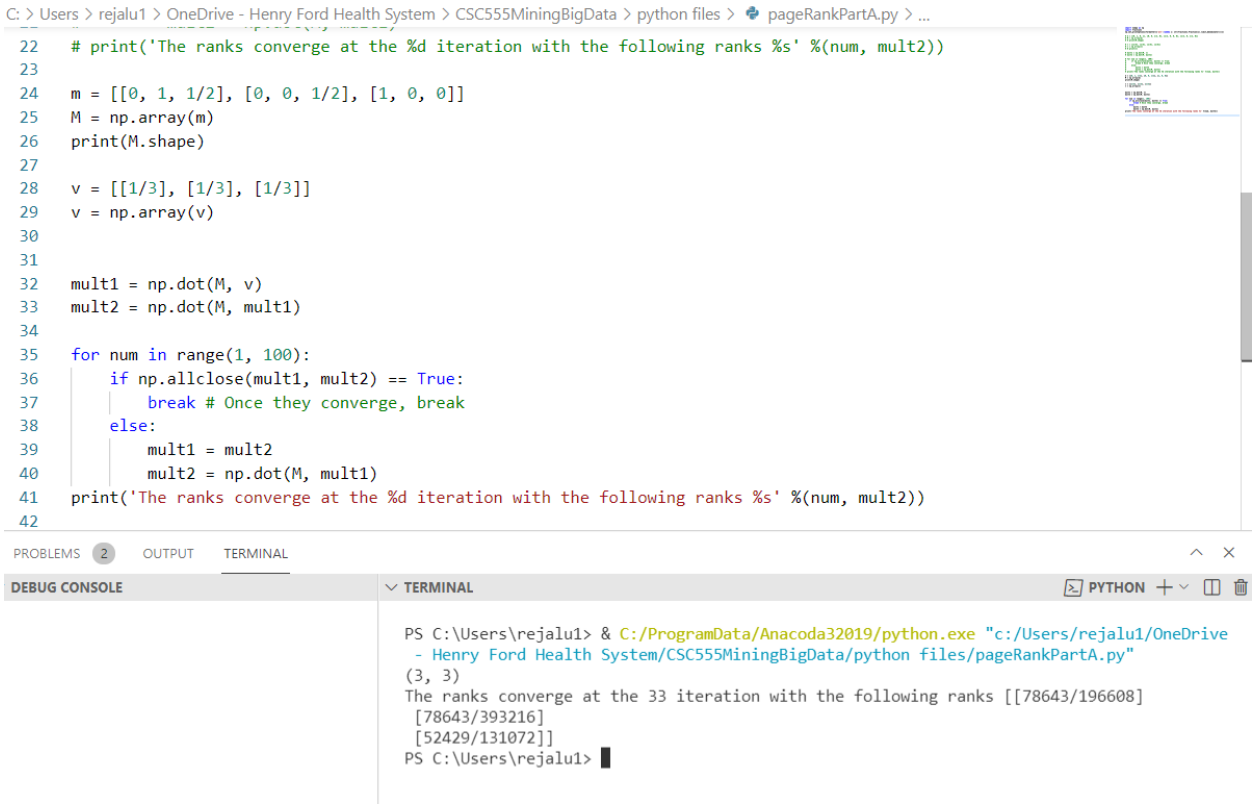
b) Now consider a graph with dead-end nodes Q and P:




What is the page rank of Q?
What is the page rank of P?
Removing the dead ends P and Q:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1/2 | A | | 1/3 | | 78643/196608 |
| 0 | 0 | 1/2 | X | | 1/3 | = | 78643/393216 |
| 1 | 0 | 0 | Z | * | 1/3 | | 52429/131072 |
| A | X | Z | | | | | |

It converges at the 33 iteration as shown below:




The page rank of :

Q = 0 * (78643/196608) + (½) * (78643/393216) + 1/3 * (52429/131072) = 0.23

P = 1 * Q = 1 * 0.23 = 0.23

c) Exercise 5.1.6 from Mining of Massive Datasets

Q

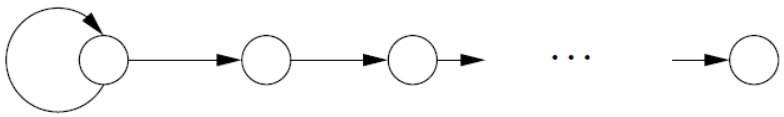


Figure 5.9: A chain of dead ends

**Exercise 5.1.6:** Suppose we recursively eliminate dead ends from the graph, solve the remaining graph, and estimate the PageRank for the dead-end pages as described in Section 5.1.4. Suppose the graph is a chain of dead ends, headed by a node with a self-loop, as suggested in Fig. 5.9. What would be the PageRank assigned to each of the nodes?

Looking at figure 5.9, all the successive dead end nodes are ranked ½ since they have one successor and predecessor. Removing the dead ends leaves us with only the root node, which has a link to itself, therefore its rank is 1. The first dead end and the root node itself make up the root node's successor, which means that the first dead end has a page rank of ½.

3) Given the following data: QQQQQZZZZAAAANNN and assuming that each letter takes 1 byte (8 bits). These are not trick questions – the goal of this exercise is to quantify compression effects.

a) What is the storage size of the uncompressed string?
Since each letter takes 1 byte:
Number of given letters = 16
Size of uncompressed string = 1byte * 16
⇨ 16 bytes.

b) Suppose you apply Run Length Encoding compression (e.g., replace QQQQQ by Q5). What is the size of the RLE-compressed string? You can assume that 5 also requires 1 byte just as Q.
(Q,5; Z,4; A,4; N, 3)
Q takes 1 byte
5 takes 1 byte
Z takes 1 byte
4 takes 1 byte
A takes 1 byte
4 takes 1 byte
N takes 1 byte

3 takes 1 byte

⇨ The RLE-compressed string takes 8 bytes.

c) Suppose you build a dictionary that represents each letter with a 5-bit code. What is the size of the dictionary-compressed string (where each letter is replaced by a 5 bit code)?

Size of the RLE compressed string = 8 bytes

⇨ 8 * 5 = 40 bits

d) Repeat the computation in 3-b and 3-c using QQBQQZZUZAAANN string (which only has 14 characters and thus takes 14 bytes uncompressed)

The size of the RLE-compressed string:
QQBQQZZUZAAANN

⇨ (Q, 4; B, 1; Z, 3; U,1; A, 3; N, 2)

⇨ 12 * 8 byte = 96 bits

The size of the dictionary-compressed string:
Number of letters = 14

⇨ Size = 12 * 5

⇨ 60 bits

4) Given the input data [(1pm, $6), (2pm, $15), (3pm, $16), (4pm, $29), (5pm, $10), (6pm, $20), (7pm, $20), (8pm, $21), (9pm, $23), (10pm, $28), (11pm, $26), (12am, $30)].

a) What will the Hive query "compute average price" return? (yes, this question is as obvious as it seems, asked for comparison with part-b and part-c)
Average price = (6 + 15 + 16 + 29 + 10 + 20 + 20 + 21 + 23 + 28 + 26 + 30)/12

⇨ 244/12

⇨ 20.3

b) What will a Storm streaming query "compute average price per each 3 hour window" return? (tumbling, i.e., non-overlapping window of tuples). For example, the first window

would 1pm-4pm. Second window would be 4pm-7pm. If you are wondering about overlap, I recommend defaulting to [1pm-4pm) [4pm-7pm). (e.g., 1pm-3:59pm). Also note that with this type of processing you have to ask such questions.

- [1pm – 4pm) = (6 + 15 + 16) / 3 = 37/3 = 12.3
- [4pm – 7pm) = (29 + 10 + 20) / 3 = 59/3 = 19.7
- [7pm – 10pm) = (20 + 21 + 23)/ 3 = 64/3 = 21.3
- [10pm – 1am) = (28 + 26 + 30) / 3 = 84/3 = 28

c) What will a Storm query "compute average price per each 3 hour window" return? (sliding, i.e. overlapping window of tuples, moving the window forward 2 hours each time). First window is 1pm-4pm, second window is 3pm-6pm and so on.

**1st Window 1pm – 4pm:**
⇨ (6 + 15 + 16)/3
⇨ 37/3 = 12.3

**2nd Window 3pm – 6pm:**
⇨ (16 + 29 + 10)/3
⇨ 55/3 = 18.3

**3rd Window 5pm – 8pm:**
⇨ (10 + 20 + 20)/3
⇨ 50/3 = 16.7

**4th Window 7pm -10pm:**
⇨ (20 + 21 + 23)/3
⇨ 64/3 = 21.3

**5th Window 9pm – 12pm:**
⇨ (23 + 28 + 26)/3
⇨ 77/3 = 25.7

**NOTE**: when Storm does not have a full window, you cannot output anything until the window fills with data.

5) Run another custom MapReduce job, implementing a solution for the following query:
For Employee(EID, EFirst, ELast, Extension) and Customer(CID, CFirst, CLast, Address), find everyone with the same name using MapReduce:

SELECT EFirst, ELast, Address, Extension
FROM Employee, Customer
WHERE EFirst = CFirst AND ELast = CLast

You can use this input data:
http://cdmgcsarprd01.dpu.depaul.edu/CSC555/employee.txt
http://cdmgcsarprd01.dpu.depaul.edu/CSC555/customer.txt

Be sure to submit your python code, the command line and the screenshot of successful execution of your code. We discussed a join example and there is a posted sample code in Week6.

# joinMapper.py

```
# JoinMapper.py
#!/usr/bin/python
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    line = line.strip()
    split = line.split('|')
    #if split[0][:3]== 'EMP':
        #if split[1] == 'Brendan' and split[2] == 'Anastasio':
        #       print(split[1] + '\t' + split[2] + '\t' + split[3]  + '\t' + 'Emp
loyees')
    if split[0][:3]== 'EMP':
        if len(split[1]) > 0 or len(split[2]) > 0  or len(split[3]) > 0:
            print(split[1] + '\t' + split[2] + '\t' + split[3] + '\t' + 'Employee
s')
    else:
        if len(split[3]) > 0:
            print(split[1] + '\t' + split[2] + '\t' + split[3] + '\t' + 'Customer
s')
```

# joinReducer.py

```
#joinReducer.py
#!/usr/bin/python
import sys

key = ''
currentKey = None
empFirstName = None  # declare variables to be used
empLastName = None
extension = None
custAdd = None
# input comes from STDIN (standard input)
for line in sys.stdin:
#for line in listOfWords:
    line = line.strip()
    split = line.split('\t')
    key = split[0] + '|' + split[1]

    value = '\t'.join(split[2:])

    if currentKey == key: # same key
        if value.endswith('Employees'):
            empFirstName = split[0]     # assign the string first name to the var
iable
            empLastName = split[1]      # assign the string last name to the vari
able
            extension = split[2]        # assign the extension number to the vari
able

        if value.endswith('Customers'):
            custAdd = split[2]              # assign the address to the variable

    else:
        if currentKey:                      # when the current key is done
            lenExtension = len(extension)  # derive the length of the variables e
xtension and CustAdd to be used to perform the join
            lenCustAdd = len(custAdd)

            if (lenExtension * lenCustAdd) > 0: # for this to  act as a join   ro
ws must exist on both sides.
                print(empFirstName + '\t' + empLastName + '\t' + extension + '\t'
 + custAdd)
        # reset the variables
        empFirstName = ''
        empLastName = ''
        extension = ''
```

Screenshots:

```
[ec2-user@ip-172-31-16-126 hadoop-2.6.4]$ hadoop jar hadoop-streaming-2.6.4.jar -D stream.num.map.output.key.fields=2 -input /user/ec2-user/empcust -output /data/empcust/out -mapper joinMapper.py -reducer joinReducer.py -file joinMapper.
py -file joinReducer.py
21/11/06 02:41:00 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [joinMapper.py, joinReducer.py, /tmp/hadoop-unjar7319519773940514522/] [] /tmp/streamjob3941967491472928973.jar tmpDir=null
21/11/06 02:41:01 INFO client.RMProxy: Connecting to ResourceManager at /172.31.16.126:8032
21/11/06 02:41:01 INFO client.RMProxy: Connecting to ResourceManager at /172.31.16.126:8032
21/11/06 02:41:02 INFO mapred.FileInputFormat: Total input paths to process : 2
21/11/06 02:41:02 INFO mapreduce.JobSubmitter: number of splits:3
21/11/06 02:41:02 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1636159689426_0005
21/11/06 02:41:02 INFO impl.YarnClientImpl: Submitted application application_1636159689426_0005
21/11/06 02:41:02 INFO mapreduce.Job: The url to track the job: http://ip-172-31-16-126.us-east-2.compute.internal:8088/proxy/application_1636159689426_0005/
21/11/06 02:41:02 INFO mapreduce.Job: Running job: job_1636159689426_0005
21/11/06 02:41:06 INFO mapreduce.Job: Job job_1636159689426_0005 running in uber mode : false
21/11/06 02:41:06 INFO mapreduce.Job:  map 0% reduce 0%
21/11/06 02:41:21 INFO mapreduce.Job:  map 33% reduce 0%
21/11/06 02:41:22 INFO mapreduce.Job:  map 100% reduce 0%
21/11/06 02:41:26 INFO mapreduce.Job:  map 100% reduce 100%
21/11/06 02:41:26 INFO mapreduce.Job: Job job_1636159689426_0005 completed successfully
21/11/06 02:41:26 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=6848677
                FILE: Number of bytes written=14138115
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=6311978
                HDFS: Number of bytes written=10785
                HDFS: Number of read operations=12
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Killed map tasks=1
                Launched map tasks=3
                Launched reduce tasks=1
                Data-local map tasks=3
                Total time spent by all maps in occupied slots (ms)=39725
                Total time spent by all reduces in occupied slots (ms)=2704
                Total time spent by all map tasks (ms)=39725
                Total time spent by all reduce tasks (ms)=2704
                Total vcore-milliseconds taken by all map tasks=39725
                Total vcore-milliseconds taken by all reduce tasks=2704
                Total megabyte-milliseconds taken by all map tasks=40678400
                Total megabyte-milliseconds taken by all reduce tasks=2768896
        Map-Reduce Framework
                Map input records=110000
                Map output records=110000
                Map output bytes=6628671
                Map output materialized bytes=6848689
                Input split bytes=321
                Combine input records=0
                Combine output records=0
                Reduce input groups=14225
                Reduce shuffle bytes=6848689
                Reduce input records=110000
                Reduce output records=188
                Spilled Records=220000
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=530
```

```
                GC time elapsed (ms)=530
                CPU time spent (ms)=4240
                Physical memory (bytes) snapshot=796205056
                Virtual memory (bytes) snapshot=8445886464
                Total committed heap usage (bytes)=506671104
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=6311657
        File Output Format Counters
                Bytes Written=10785
21/11/06 02:41:26 INFO streaming.StreamJob: Output directory: /data/empcust/out
```

One file was written:

```
[ec2-user@ip-172-31-16-126 hadoop-2.6.4]$ hadoop fs -ls /data/empcust/out
Found 2 items
-rw-r--r--   2 ec2-user supergroup          0 2021-11-06 02:41 /data/empcust/out/_SUCCESS
-rw-r--r--   2 ec2-user supergroup      10785 2021-11-06 02:41 /data/empcust/out/part-00000
[ec2-user@ip-172-31-16-126 hadoop-2.6.4]$
```

A screenshot of the file contents:

```
[ec2-user@ip-172-31-16-126 hadoop-2.6.4]$ hadoop fs -cat /data/empcust/out/part-00000
Brendan Anastasio       70      613 Devon Court, West Orange, NJ 07052
Brendan Berenbaum       76      343 Franklin Street, Fort Walton Beach, FL 32547
Brendan Bosque  79      783 8th Avenue, Elkton, MD 21921
Brendan Cashin  71      742 Beechwood Drive, Fairfax, VA 22030
Brendan Lembke  32      926 Olive Street, Fort Wayne, IN 46804
Brendan Mabe    41      761 Route 5, Chandler, AZ 85224
Brendan Maynor  80      693 Orchard Street, Algonquin, IL 60102
Brendan Mcdougle        24      228 Homestead Drive, Aiken, SC 29803
Brendan Mullican        40      992 Oxford Court, Tewksbury, MA 01876
Brendan Platt   45      232 Old York Road, Englewood, NJ 07631
Brendan Read    66      222 Sycamore Lane, Garden City, NY 11530
Brendan Tyrrell 50      455 Warren Street, Wyandotte, MI 48192
Brendan Walpole 38      949 Maple Street, Oakland Gardens, NY 11364
Francoise       Anastasio       73      52 Augusta Drive, Clayton, NC 27520
Francoise       Berenbaum       46      957 Liberty Street, Satellite Beach, FL 32937
Francoise       Bosque  34      228 Dogwood Drive, Melrose, MA 02176
Francoise       Cashin  80      80 Dogwood Drive, Fairhope, AL 36532
Francoise       Hartley 81      687 Cedar Street, Elgin, IL 60120
Francoise       Lembke  72      55 Forest Avenue, Media, PA 19063
Francoise       Mabe    52      655 Park Avenue, Waynesboro, PA 17268
Francoise       Maynor  62      128 Summit Avenue, Cranston, RI 02920
Francoise       Mcdougle        80      761 Route 5, Chandler, AZ 85224
Francoise       Mullican        61      274 Meadow Street, El Paso, TX 79930
Francoise       Platt   34      803 Cedar Lane, Essex, MD 21221
Francoise       Read    33      589 Bridge Street, Fort Worth, TX 76110
Francoise       Tyrrell 39      969 Valley View Road, Deland, FL 32720
Francoise       Walpole 47      957 Liberty Street, Satellite Beach, FL 32937
Freeda  Anastasio       69      480 Strawberry Lane, South Lyon, MI 48178
Freeda  Berenbaum       48      176 Warren Street, Piqua, OH 45356
Freeda  Bosque  70      687 Cedar Street, Elgin, IL 60120
Freeda  Cashin  73      1 Homestead Drive, Willoughby, OH 44094
Freeda  Hartley 32      187 Hickory Lane, Raleigh, NC 27603
Freeda  Lembke  36      228 Dogwood Drive, Melrose, MA 02176
Freeda  Mabe    37      797 Cedar Street, Muskegon, MI 49441
Freeda  Maynor  80      516 Essex Court, Adrian, MI 49221
Freeda  Mcdougle        58      783 8th Avenue, Elkton, MD 21921
Freeda  Mullican        50      517 Andover Court, Naugatuck, CT 06770
Freeda  Platt   24      187 Hickory Lane, Raleigh, NC 27603
Freeda  Read    66      949 Maple Street, Oakland Gardens, NY 11364
Freeda  Tyrrell 29      480 Strawberry Lane, South Lyon, MI 48178
Freeda  Walpole 24      688 Main Street West, Alexandria, VA 22304
Hosea   Anastasio      68      295 Hillcrest Drive, Green Bay, WI 54302
Hosea   Berenbaum      23      720 Route 20, Los Banos, CA 93635
Hosea   Bosque  64      187 Magnolia Avenue, Maryville, TN 37803
Hosea   Cashin  65      957 Liberty Street, Satellite Beach, FL 32937
Hosea   Hartley 25      635 Cross Street, Monsey, NY 10952
Hosea   Lembke  34      655 Park Avenue, Waynesboro, PA 17268
Hosea   Mabe    47      274 Meadow Street, El Paso, TX 79930
Hosea   Maynor  58      705 Main Street South, Anaheim, CA 92806
Hosea   Mcdougle        28      187 Magnolia Avenue, Maryville, TN 37803
Hosea   Mullican        75      295 Hillcrest Drive, Green Bay, WI 54302
Hosea   Platt   23      455 Warren Street, Wyandotte, MI 48192
Hosea   Read    55      455 Warren Street, Wyandotte, MI 48192
Hosea   Tyrrell 71      800 Rosewood Drive, Soddy Daisy, TN 37379
Hosea   Walpole 28      477 Pine Street, Neenah, WI 54956
Isidro  Anastasio       25      55 Forest Avenue, Media, PA 19063
Isidro  Berenbaum       24      687 Cedar Street, Elgin, IL 60120
Isidro  Bosque  63      128 Summit Avenue, Cranston, RI 02920
Isidro  Cashin  26      705 Main Street South, Anaheim, CA 92806
Isidro  Hartley 38      253 Route 2, Sun Prairie, WI 53590
Isidro  Mabe    71      187 Magnolia Avenue, Maryville, TN 37803
```

6) In this section you will run an implementation of the page rank algorithm. Unfortunately, newer versions of Mahout (machine learning library that runs on Hadoop and Spark, which we will use for a couple of other examples later) removed their MapReduce page rank implementation. So we will run a publically available implementation from GitHub – an opportunity to build a custom Java job (while it is written in Java, you do not need to know Java to run it).

Download the Stanford graph dataset (this is a graph of page links from Stanford.edu, originally from here – https://snap.stanford.edu/data/)

**wget http://cdmgcsarprd01.dpu.depaul.edu/CSC555/web-Stanford.txt.gz**

**gunzip web-Stanford.txt.gz**

a. Take a look at the file and report how many nodes and edges the web-Stanford.txt contains (it's on the third line).

- **There are 281,903 nodes and 2,312,497 edges.**

Download the PageRank implementation from here: https://github.com/danielepantaleone/hadoop-pagerank, as follows:

Install git (Git is a version control system for tracking changes in source code).

**sudo yum install git**

**git clone** https://github.com/danielepantaleone/hadoop-pagerank

**cd hadoop-pagerank/**

Edit the source file (that is the reducer of the preprocessing MapReduce job)

**nano src/it/uniroma1/hadoop/pagerank/job1/PageRankJob1Reducer.java**

to modify PageRank.NODES.size() to 281903 (number of nodes), like in the screenshot. // is a comment in Java, equivalent of # in python, so you do not need the blue line.

```
boolean first = true;
// String links = (PageRank.DAMPING / PageRank.NODES.size()) + "\t";
String links = (PageRank.DAMPING / 281903) + "\t";
```

This code is meant to count the total number of nodes, but it isn't working correctly (returning 0) and I have not yet figured out why. If you do not make this change, all of the initial values will compute be Infinity.

Set the classpath environment variable (you do not have to put it in .bashrc because we will only be using it to compile Java code in the next command). As always, note that this is a single line, no linebreaks

**export CLASSPATH="/home/ec2-user/hadoop-2.6.4/share/hadoop/common/*:/home/ec2-user/hadoop-2.6.4/share/hadoop/mapreduce/lib/*:/home/ec2-user/hadoop-2.6.4/share/hadoop/mapreduce/*"**

Compile the code:

**javac -sourcepath src/ src/it/uniroma1/hadoop/pagerank/PageRank.java**

Build the new jar file with your custom compiled code.

**cd src**

**jar cvf PageRank.jar it**

Congratulations, you have build a new custom PageRank.jar from Java code, similar to hadoop-streaming or hadoop-examples jar files that we have previously used.

Now, load the web Stanford data into HDFS:

**hadoop fs -mkdir /data/**

**hadoop fs -mkdir /data/webStanford**

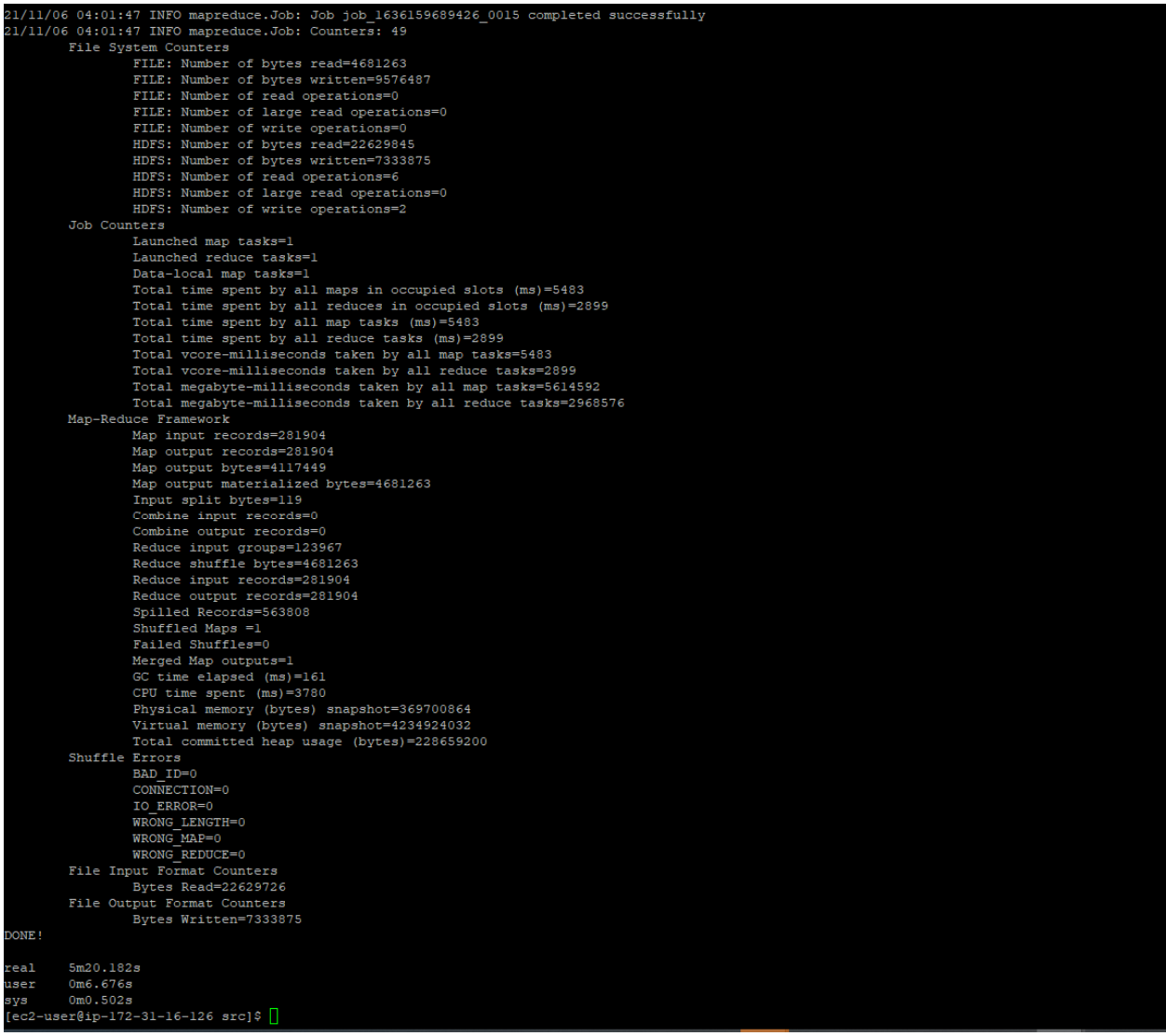
**hadoop fs -put ~/web-Stanford.txt /data/webStanford**

And, finally, run the new jar to execute page rank evaluation.

**time hadoop jar PageRank.jar it.uniroma1.hadoop.pagerank.PageRank –input /data/webStanford –output /data/prOutput –damping 90 –count 8**

Note that --damping 90 means a 10% chance of teleporting and --count of 8 means 8 iterations are computed.

b. Report the runtime (took about 5 minutes to run when I tested it)
   It took 5mins to run as shown below:




c. Submit a screenshot of the first page of nodes, e.g., by running **hadoop fs -cat /data/prOutput/result/part-r-00000 | more**

ec2-user@ip-172-31-16-126:~/hadoop-pagerank/src

```
0.15000000596046448     75380
0.15000000596046448     155237
0.15000000596046448     75378
0.15000000596046448     155226
0.15000000596046448     155222
0.15000000596046448     155221
0.15000000596046448     75372
0.15000000596046448     125860
0.15000000596046448     47294
0.15000000596046448     155216
0.15000000596046448     155204
0.15000000596046448     155203
0.15000000596046448     47303
0.15000000596046448     155116
0.15000000596046448     125881
0.15000000596046448     47316
0.15000000596046448     75340
0.15000000596046448     47317
0.15000000596046448     125872
0.15000000596046448     75358
0.15000000596046448     75357
0.15000000596046448     47327
0.15000000596046448     47336
0.15000000596046448     155122
0.15000000596046448     47357
0.15000000596046448     155140
0.15000000596046448     47360
0.15000000596046448     155137
0.15000000596046448     75347
0.15000000596046448     155129
0.15000000596046448     68099
0.15000000596046448     229465
0.15000000596046448     68079
0.15000000596046448     100022
0.15000000596046448     22977
0.15000000596046448     229439
0.15000000596046448     68199
0.15000000596046448     229784
0.15000000596046448     9995
0.15000000596046448     99949
0.15000000596046448     229792
0.15000000596046448     68078
0.15000000596046448     229796
0.15000000596046448     22941
0.15000000596046448     1335
0.15000000596046448     229811
0.15000000596046448     229401
0.15000000596046448     229388
0.15000000596046448     229387
0.15000000596046448     229831
0.15000000596046448     68068
0.15000000596046448     229369
0.15000000596046448     229842
0.15000000596046448     229846
0.15000000596046448     22936
0.15000000596046448     229358
0.15000000596046448     133522
0.15000000596046448     229855
0.15000000596046448     229351
0.15000000596046448     133529
0.15000000596046448     229341
0.15000000596046448     99924
--More--
```

Submit a single document containing your written answers. Be sure that this document contains your name and "CSC 555 Assignment 5" at the top.