Danmarks Tekniske Universitet

01415

Computational Tools for Big Data

Week 67 - Exercise

Anonymous

October 2016

1 Week 5

Exercise 5.1

Assignment :
Go to GraphStory.com and make a new account (you get 14 days for free). Create a graph database. Go to the Neo4j Web UI for your new graph. Log in with the autogenerated username and password from Graph Story.
Solution :

We created account on GraphStory and imported all data. Example query and data output can be seen on following figure.

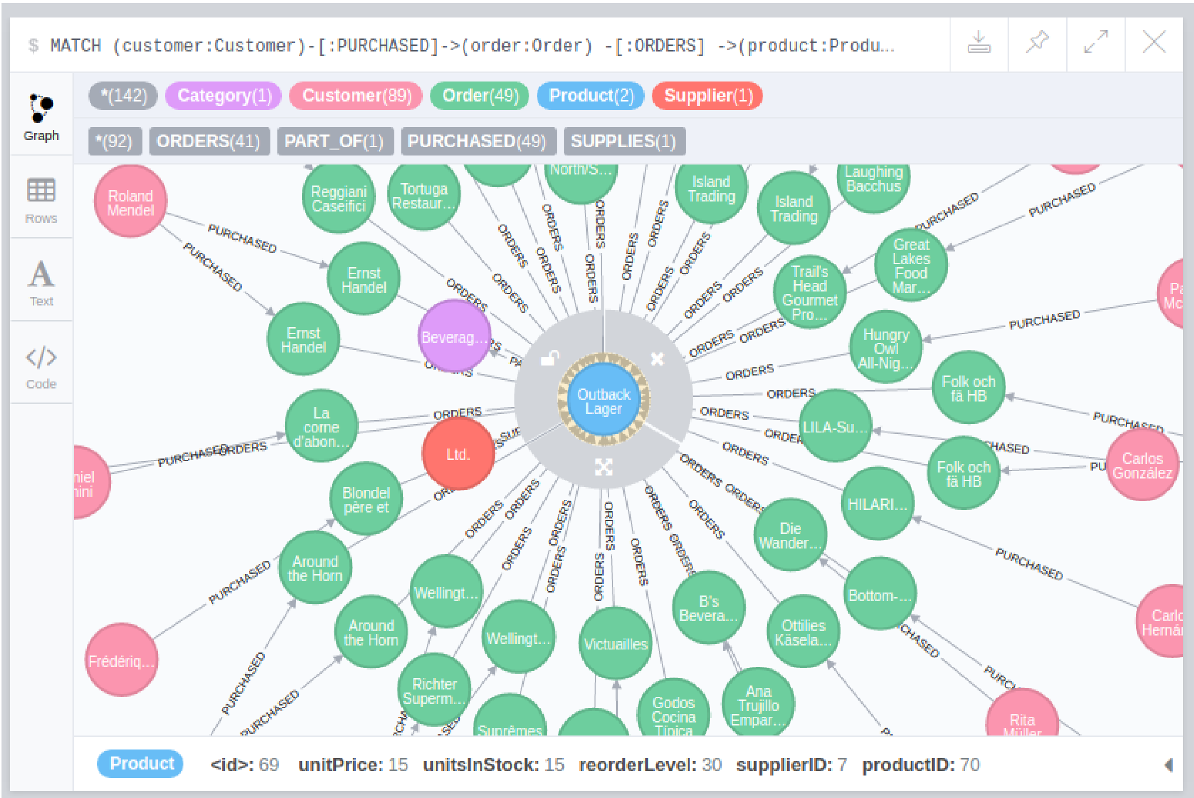


Figure 1: Example call on GraphStory.com

Exercise 5.2

Assignment :
The customer with customerID ALFKI has made a number of orders containing some products. Return all the orders made by ALFKI and the products they contain.
Solution :
We selected relation between customers-orders-products and then filtered entries from customer 'ALFKI'.

```
MATCH (customer:Customer)-[:PURCHASED]->(order:Order) -[:ORDERS] ->(product:Product)
WHERE customer.customerID = "ALFKI"
RETURN order.orderID,product.productID, product.productName
```

Output:

order.orderID	product.productID	product.productName
11011	58	Escargots de Bourgogne
11011	71	Flotemysost
10952	6	Grandma's Boysenberry Spread
10952	28	Rössle Sauerkraut
10835	59	Raclette Courdavault
10835	77	Original Frankfurter grüne Soße
10692	63	Vegie-spread
10702	3	Aniseed Syrup
10702	76	Lakkalikööri
10643	39	Chartreuse verte
10643	46	Spegesild
10643	28	Rössle Sauerkraut

Exercise 5.3

Assignment :
The customer with customerID ALFKI has made a number of orders containing some products. Return orders made by ALFKI that contain at least 2 products. Also return the products.
Solution :
We selected relation between customers-orders and filtered entries from customer 'ALFKI'.Then we counted number of products in each order and selected just those which contain at least 2 products.

```
MATCH (customer:Customer)-[:PURCHASED]->(order:Order)
WHERE customer.customerID = "ALFKI"
MATCH (order)-[r:ORDERS]->(:Product)
WITH order, count(r) AS count
WHERE count => 2
MATCH (order)-[r:ORDERS]->(product:Product)
RETURN order.orderID,product.productID,product.productName,count;
```

Output:

order.orderID	product.productID	product.productName	count
10835	59	Raclette Courdavault	2
10835	77	Original Frankfurter grüne Soße	2
10643	39	Chartreuse verte	3
10643	46	Spegesild	3
10643	28	Rössle Sauerkraut	3
11011	58	Escargots de Bourgogne	2
11011	71	Flotemysost	2
10702	3	Aniseed Syrup	2
10702	76	Lakkalikööri	2
10952	6	Grandma's Boysenberry Spread	2
10952	28	Rössle Sauerkraut	2

Exercise 5.4

Assignment :
Determine how many and who has ordered "Uncle Bob's Organic Dried Pears" (productID 7).
Solution :
We selected relation between product-order-customer and then removed all entries which do not contain productID = 7. We sum quantity and return customers ids.

```
MATCH (product:Product)<-[r:ORDERS]-(order:Order)<-[:PURCHASED]-(customer:Customer)
WHERE product.productID = "7"
RETURN customer.customerID, SUM(r.quantity)
```

Output:
20 Customers

customer.customerID	SUM(r.quantity)
VAFFE	10
BOTTM	20
BONAP	58
SPLIR	10
GOURL	3
OCEAN	6
ERNSH	18
LILAS	15
SAVEA	45
QUICK	135
FOLKO	60
OTTIK	55
RATTC	76
LACOR	5
BSBEV	34
EASTC	100
SANTG	12
VICTE	26
FOLIG	35
REGGC	40

Exercise 5.5

Assignment :
How many different and which products have been ordered by customers who have also ordered “Uncle Bob’s Organic Dried Pears”?
Solution :
Similar to solution from Exercise 5.4, we selected customers which ordered productID = 7 and then on these customers we match all products orders they made and returned just distinct names.

```
MATCH (prod:Product)<-[r:ORDERS]-(Order)<-[:PURCHASED]-(customer:Customer)
WHERE prod.productID = "7"
MATCH (customer)-[:PURCHASED]->(ord:Order) -[:ORDERS] ->(product:Product)
RETURN DISTINCT(product.productName)
```

Output:
76 Products
Grandma’s Boysenberry Spread, Chef Anton’s Cajun Seasoning, Northwoods Cranberry Sauce, Uncle Bob’s Organic Dried Pears, Aniseed Syrup, Chang, Pavlova, Sir Rodney’s Marmalade , Tunnbröd , Mascarpone Fabioli , Ikura, Queso Manchego La Pastora, Konbu, Tofu , Pâté chinois , Camembert Pierrot, Wimmers gute Semmelknödel, Louisiana Hot Spiced Okra, Chartreuse verte , Jack’s New England Clam Chowder, Spegesild, Filo Mix , Original Frankfurter grüne Soße, Rhönbräu Klosterbier , Röd Kaviar , Guaraná Fantástica , Tarte au sucre , Queso Cabrales , Côte de Blaye, Gnocchi di nonna Alice , Alice Mutton , NuNuCa Nuß-Nougat-Creme, Schoggi Schokolade , Flotemysost, Gorgonzola Telino, Lakkalikööri , Perth Pasties, Raclette Courdavault , Louisiana Fiery Hot Pepper Sauce , Nord-Ost Matjeshering, Escargots de Bourgogne , Chai , Ipoh Coffee, Mozzarella di Giovanni , Chef Anton’s Gumbo Mix , Tourtière, Ravioli Angelo , Outback Lager, Maxilaku , Rössle Sauerkraut, Sir Rodney’s Scones, Gumbär Gummibärchen, Longlife Tofu, Gudbrandsdalsost , Teatime Chocolate Biscuits , Genen Shouyu , Geitost, Steeleye Stout , Thüringer Rostbratwurst, Gustaf’s Knäckebröd, Manjimup Dried Apples, Vegie-spread , Zaanse koeken, Sirop d’érable , Gula Malacca , Scottish Longbreads, Singaporean Hokkien Fried Mee, Carnarvon Tigers , Boston Crab Meat , Rogede sild, Mishi Kobe Niku, Inlagd Sill, Valkoinen suklaa , Sasquatch Ale, Chocolade, Gravad lax

2 Week 7

Exercise 7.1

Assignment :
Define and implement a MapReduce job to count the occurrences of each word in a text file. Document that it works with a small example.
Solution :

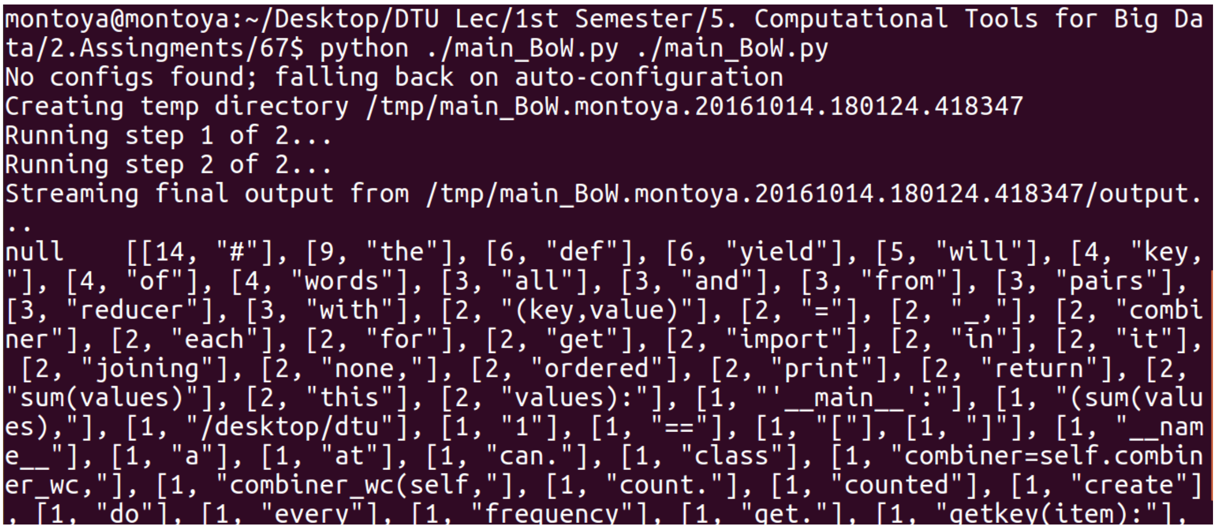
We have implemented this function with MrJob using one single step, consisting of a mapper, a combiner and a reducer. Each line of the file goes to a mapper that divides the line into words and yields a (key,value) pair containing ("word",1). The combiners will receive a set of these tuples and for each key, they will add up their values. The reducer performs the same task, but in this case it receives all the (key,value) pair, thus finishing the job.

Then we output the ("word",N) pairs as values, with no key, so that the next job will receive them all under the same key. We have improved a little the system and created a next step in which a reducer orders the words in descending order according to the number of occurrences of the word.
The next image shows the partial output of the first step. The file used as input is the python code file itself.

```
montoya@montoya:~/Desktop/DTU Lec/1st Semester/5. Computational Tools for Big Da
ta/2.Assingments/67$ python ./main_BoW.py ./main_BoW.py
No configs found; falling back on auto-configuration
Creating temp directory /tmp/main_BoW.montoya.20161014.175740.231398
Running step 1 of 1...
Streaming final output from /tmp/main_BoW.montoya.20161014.175740.231398/output.
..
null    [14, "#"]
null    [1, "'__main__':"]
null    [2, "(key,value)"]
null    [1, "(sum(values),"]
null    [1, "/desktop/dtu"]
null    [1, "1"]
null    [2, "="]
null    [1, "=="]
null    [1, "["]
null    [1, "]"]
null    [2, "_,"]
null    [1, "__name__"]
```

Figure 2: Step 1 Word Counter

The next image shows the partial output of the second step, with the ordered words. We can see that each word is a tuple with the number of occurrences and the the word itself.



```
montoya@montoya:~/Desktop/DTU Lec/1st Semester/5. Computational Tools for Big Da
ta/2.Assingments/67$ python ./main_BoW.py ./main_BoW.py
No configs found; falling back on auto-configuration
Creating temp directory /tmp/main_BoW.montoya.20161014.180124.418347
Running step 1 of 2...
Running step 2 of 2...
Streaming final output from /tmp/main_BoW.montoya.20161014.180124.418347/output.
..
null    [[14, "#"], [9, "the"], [6, "def"], [6, "yield"], [5, "will"], [4, "key,
"], [4, "of"], [4, "words"], [3, "all"], [3, "and"], [3, "from"], [3, "pairs"],
[3, "reducer"], [3, "with"], [2, "(key,value)"], [2, "="], [2, "_,"], [2, "combi
ner"], [2, "each"], [2, "for"], [2, "get"], [2, "import"], [2, "in"], [2, "it"],
 [2, "joining"], [2, "none,"], [2, "ordered"], [2, "print"], [2, "return"], [2,
"sum(values)"], [2, "this"], [2, "values):"], [1, "'__main__':"], [1, "(sum(valu
es),"], [1, "/desktop/dtu"], [1, "1"], [1, "=="], [1, "["], [1, "]"], [1, "__nam
e__"], [1, "a"], [1, "at"], [1, "can."], [1, "class"], [1, "combiner=self.combin
er_wc,"], [1, "combiner_wc(self,"], [1, "count."], [1, "counted"], [1, "create"]
, [1, "do"], [1, "every"], [1, "frequency"], [1, "get."], [1, "getkey(item):"],
```

Figure 3: Step 2 Word Counter

2.1 main_BoW.py

```python
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRWordWC(MRJob):

    def mapper_WC(self, _, line):
        # Each mapper will create the (key,value) pairs of words
        # for every line they get.
        words = line.split()
        for word in words:
            yield word.lower(), 1

    def combiner_WC(self, key, values):
        # Each combiner will get a set of (key,value) pairs
        # And it will join the ones it can.
        yield key, sum(values)

    def reducer_WC(self, key, values):
        # The reducer will do the same joining with all the joining
        # from the combiner

        # Instead of yielding key-value pairs with yield key, sum(values)
        # We just yield values, with the words and their count.
        # This way, the next reducer will get all values at once
#         print list(key)
#         print list(values)
        yield None, (sum(values), key)

    def reducer_orderedWords(self, _, word_count_pairs):
        # This reducer gets all the counted words and yields
        # then in order of frequency
        def getKey(item):
            return item[0]

        ordered = sorted(word_count_pairs, key= getKey, reverse=True)
        yield None, ordered

    def steps(self):
        return [
```

```python
            MRStep(mapper=self.mapper_WC,
                   combiner=self.combiner_WC,
                   reducer=self.reducer_WC),
            MRStep(reducer=self.reducer_orderedWords)
        ]

if __name__ == '__main__':
    MRWordWC.run()
```

Exercise 7.2

Assignment :
Define and implement a MapReduce job that determines if a graph has an Euler tour (all vertices have even degree) where you can assume that the graph you get is connected.

This file https://www.dropbox.com/s/usdi0wpsqm3jb7f/eulerGraphs.txt?dl=0 has 5 graphs – for each graph, the first line tells the number of nodes N and the number of edges E. The next E lines tells which two nodes are connected by an edge. Two nodes can be connected by multiple edges.
It is fine if you split the file into 5 different files. You do not need to keep the node and edge counts in the top of the file.
Solution :

We have splitted the file into 5 different files to make the code more reusable. From Graph Theory we know that there is an Euler Path if the number of edges of each node is even, or, there are exactly 2 nodes with and odd number of edges. To solve this exercise we have create 2 steps.

The **first step** calculates if the number of edges of each node is an even or odd number. For doing so, for each vertex, its mapper creates ("Node", 1) pairs, then the combiner and the reducer add the values for each node, and the reducer also applies modulus 2 operator. Then the **second step** is a reducer that checks if the number of odd degree nodes is 0 or 2 and outputs the solution.

The next image shows the partial output of the first step using the second graph given in the assignment. For every node, it yields 0 if the number of edges is even and 1 if it is odd. The key is the same for all nodes and it is "None".

```
montoya@montoya:~/Desktop/DTU Lec/1st Semester/5. Computational Tools for Big Da
ta/2.Assingments/67$ python ./main_EULER.py ./data/eulerGraphs2.txt
No configs found; falling back on auto-configuration
Creating temp directory /tmp/main_EULER.montoya.20161015.144559.363265
Running step 1 of 1...
Streaming final output from /tmp/main_EULER.montoya.20161015.144559.363265/outpu
t...
null    0
null    0
null    0
null    0
null    0
null    0
null    0
null    0
null    0
null    0
null    0
null    0
Removing temp directory /tmp/main_EULER.montoya.20161015.144559.363265...
```

Figure 4: Euler path Step 1

The next image shows the output of the second step, it just the result of weather or not there exists an Euler path.

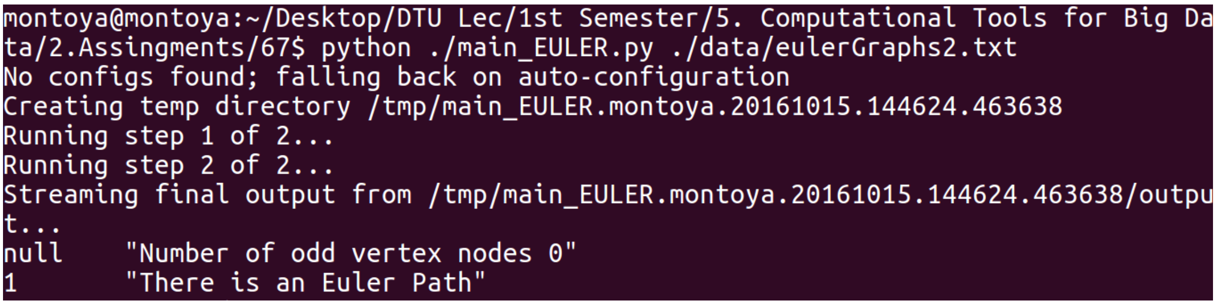


Figure 5: Euler path Step 2

2.2 main_EULER.py

```python
from mrjob.job import MRJob
from mrjob.step import MRStep
# This Job is meant to determine if a graph has an Euler tour
# This happends if all vertices have even degree, or only 2 vertex
# have an even degree

class MRWordVC(MRJob):

    def mapper_VC(self, _, line):
        # This mapper assigns per every vertex, a 1 to every node in the vertex.
        nodes_vertex = line.split()
        if (len(nodes_vertex) == 2):
            yield nodes_vertex[0], 1
            yield nodes_vertex[1], 1

    def combiner_VC(self, key, values):
        # Each combiner will get a set of (key,value) pairs
        # And it will join the ones it can.
        yield key, sum(values)

    def reducer_VC(self, key, values):
        # The reducer will do the same joining with all the joining
        # from the combiner. And it will also say if the number of
        # vertex is odd or even for every node
        yield None, sum(values)%2

    def reducer_oddVertex(self, _, vertex_count_nodes):
        # This reducer gets all the counted words and yields
        # then in order of frequency
        suma_odd = sum(vertex_count_nodes)
        yield None, "Number of odd vertex nodes " + str(suma_odd)

        if (suma_odd == 0 or suma_odd == 2):
            yield 1, "There is an Euler Path"
        else:
            yield 0, "There is no Euler Path"

    def steps(self):
        return [
            MRStep(mapper=self.mapper_VC,
                   combiner=self.combiner_VC,
                   reducer=self.reducer_VC),
            MRStep(reducer=self.reducer_oddVertex)
        ]

if __name__ == '__main__':
```

```
    MRWordVC.run()
```

Exercise 7.3

Make a MapReduce job which counts the number of triangles in a graph. Use the following graph http://www.cise.ufl.edu/research/sparse/matrices/SNAP/roadNet-CA.html. The "Matrix Market" format is the preferred one as it has the same format as the Facebook data. (Remember to remove the header in the beginning of file)
Document that it works using a small example and run it on the large file to see that you get the right results.
Solution :

We have solved this problem using two different approaches, first we will show the most efficient one for this problem and then we will explain the second implementation.

First implementation :

In the first solution, we have used the Algortihm 3 MR-NodeIterator++(V,E) described in the document "Counting Triangles and the Curse of the Last Reducer" by Siddharth Suri Sergei and Vassilvitskii. In this implementation we have 3 steps:

- The first step calculates the adjacency matrix for all nodes. That is, for every node, it stores its neighbours in a sparse form. Each node is a row of the adjacency matrix and is stored as a dictionary, where each entry is a neighbour of that node and contain the value 1. The reducer of the method yields all the nodes (rows of the adjacency matrix) with the same key "A" and a tuple ("Node", DictionaryOfNeighbours) so that the next step has access to the whole adjacency matrix.
- The second step contains only a reducer that yields all the possible 2-node paths. For every node, and for each couple of neighbours of the node, it yields one of the nodes and the neighbours of the other node. It is optimized so that it does not yield each pair of neighbours twice. The yielded 2 node paths are composed by a key containing the first node and the two neighbours, and a value containing the first neighbour and the neighbours of the second neighbour.
- The third step contains a mapping that calculates for every 2-node paths, if there exists the triangle. For doing so, it just checks if the first node received is in the neighbourhood of the second node received. If it is, it yields a 1. Then the reducer sums all the yielded values, and divided by 3 to get the number of triangles. It is divided by 3 because we are gonna check the same path 3! times but since we optimize it so that for a given node, it does not output the same 2 neighbour nodes twice, we only have to divide by 3.

The next image shows the output for the first step using the third graph given in the assignment. We can appreciate that it yields all the nodes under the same key value "A". The nodes are rows of the adjacency matrix and we can see the tuples ("Node", DictionaryOfNeighbours).

```
montoya@montoya:~/Desktop/DTU Lec/1st Semester/5. Computational Tools for Big Da
ta/2.Assingments/67$ python ./main_Triangle.py ./data/eulerGraphs3.txt
No configs found; falling back on auto-configuration
Creating temp directory /tmp/main_Triangle.montoya.20161015.145318.260322
Running step 1 of 1...
Streaming final output from /tmp/main_Triangle.montoya.20161015.145318.260322/ou
tput...
"A"     ["0", {"1": 1, "98": 1, "2": 1, "99": 1}]
"A"     ["1", {"99": 1, "0": 1, "3": 1, "2": 1}]
"A"     ["10", {"11": 1, "8": 1, "12": 1, "9": 1}]
"A"     ["11", {"9": 1, "10": 1, "13": 1, "12": 1}]
"A"     ["12", {"11": 1, "10": 1, "13": 1, "14": 1}]
"A"     ["13", {"11": 1, "12": 1, "15": 1, "14": 1}]
"A"     ["14", {"13": 1, "12": 1, "15": 1, "16": 1}]
"A"     ["15", {"13": 1, "14": 1, "17": 1, "16": 1}]
"A"     ["16", {"18": 1, "15": 1, "14": 1, "17": 1}]
"A"     ["17", {"19": 1, "18": 1, "15": 1, "16": 1}]
"A"     ["18", {"19": 1, "20": 1, "17": 1, "16": 1}]
"A"     ["19", {"18": 1, "20": 1, "21": 1, "17": 1}]
```

Figure 6: Counting Triangles step 1

The next image shows the output for the second step. We can see that the each yielded item contains in its value the first neighbour and the neighbours of the second neighbour.

```
Running step 2 of 2...
Streaming final output from /tmp/main_Triangle.montoya.20161015.150611.136503/ou
tput...
"24_25_26"      ["25", {"24": 1, "25": 1, "27": 1, "28": 1}]
"24_25_22"      ["25", {"24": 1, "20": 1, "21": 1, "23": 1}]
"24_25_23"      ["25", {"24": 1, "25": 1, "21": 1, "22": 1}]
"24_26_22"      ["26", {"24": 1, "20": 1, "21": 1, "23": 1}]
"24_26_23"      ["26", {"24": 1, "25": 1, "21": 1, "22": 1}]
"24_22_23"      ["22", {"24": 1, "25": 1, "21": 1, "22": 1}]
"25_24_26"      ["24", {"24": 1, "25": 1, "27": 1, "28": 1}]
"25_24_27"      ["24", {"25": 1, "26": 1, "28": 1, "29": 1}]
"25_24_23"      ["24", {"24": 1, "25": 1, "21": 1, "22": 1}]
"25_26_27"      ["26", {"25": 1, "26": 1, "28": 1, "29": 1}]
"25_26_23"      ["26", {"24": 1, "25": 1, "21": 1, "22": 1}]
"25_27_23"      ["27", {"24": 1, "25": 1, "21": 1, "22": 1}]
"26_24_25"      ["24", {"24": 1, "26": 1, "27": 1, "23": 1}]
"26_24_27"      ["24", {"25": 1, "26": 1, "28": 1, "29": 1}]
```

Figure 7: Counting Triangles step 2

The next image shows the output for the third step. It just consists of the number of triangles found.

```
Streaming final output from /tmp/main_Triangle.montoya.20161015.145201.833106/ou
tput...
"NT"    100
Removing temp directory /tmp/main_Triangle.montoya.20161015.145201.833106...
```

Figure 8: Counting Triangles step 3

For the big Graph roadNet problem, the obtained number of triangles is 120676 and in Intel(R) Core(TM) i7-4712MQ CPU @ 2.30GHz processor, along with other operating applications, the time required is lower than 8min30sec.

```
montoya@montoya:~/Desktop/DTU Lec/1st Semester/5. Computational Tools for Big Da
ta/2.Assingments/67$ time time python main_Triangle.py ./data/roadNet-CA.mtx
No configs found; falling back on auto-configuration
Creating temp directory /tmp/main_Triangle.montoya.20161015.160600.265625
Running step 1 of 3...
Running step 2 of 3...
Running step 3 of 3...
Streaming final output from /tmp/main_Triangle.montoya.20161015.160600.265625/ou
tput...
"NT"    120676
Removing temp directory /tmp/main_Triangle.montoya.20161015.160600.265625...

real    8m24.405s
user    8m0.068s
sys     0m12.776s
```

Figure 9: Counting Triangles roadNet problem

2.3 main_Traingle.py

```python
from mrjob.job import MRJob
from mrjob.step import MRStep

# This Job determines the number of triangle sin a graph using the algorithm
# Algortihm 3 MR-NodeIterator++(V,E) described in the document
#"Counting Triangles and the Curse of the Last Reducer"
# by Siddharth Suri Sergei and Vassilvitskii
```

```

class MRNodesNT(MRJob):

    #####################################
    ## Job to create the adjacency matrix
    #####################################
    def mapper_AGeneration(self, _, line):
        # This mapper assigns per every vertex, a 1 to every nodein the vextex.
        nodes_vertex = line.split()
        if (len(nodes_vertex) == 2):
            yield nodes_vertex[0], (nodes_vertex[1],1)
            yield nodes_vertex[1], (nodes_vertex[0],1)

    def combiner_AGeneration(self, key, values):
        # Combines the adjacencies of the nodes.
        yield key, list(values)

    def reducer_AGeneration(self, key, values):
        # Combines the adjacencies of the nodes again, all of them.
        final_list = []
        for value in values:
            final_list.extend(list(value))
        final_dict = dict(final_list)

        row_elem = [key, final_dict]
        # Each row contains on its data, the number of the row and the dictionary
        # of values in sparse way. [Nrow, Dict of cols]

        yield "A", row_elem  # So that the next reducer get

    #####################################
    ## Generate the 2-length paths
    #####################################
    # We need to have access to all rows of the adjacency matrix to create
    # the tasks so we implement a new returcer to join them
    def reducer_inter(self, key, values):
        val_list = list(values)
        Matrix_A = dict(val_list) # We make another dictionary with the cols
        A_keys = Matrix_A.keys()

        # Output all the pairs !! And their neighbours
        for k in A_keys:
            neighs = Matrix_A[k]
            neighs_keys = neighs.keys()
            Nneigh = len(neighs_keys)
            for i in range(Nneigh):
                for j in range(i,Nneigh):
                    nk1 = neighs_keys[i]
                    nk2 = neighs_keys[j]
                    if (nk1 != nk2):
                        yield str(k) + "_" + str(nk1)+"_"+str(nk2), [nk1, Matrix_A[nk2]]

    ##################################################
    ## Check if you can form a triangle with every pair
    ##################################################
    def mapper_2Path(self, key, values):
        # This mapper outputs pairs of neighbours for each node.
        # So that latter the mapping can
        lista = list(values)
#          print lista
        if (lista[0] in lista[1].keys()):
```

```
                yield None,1

        def reducer_2Path(self, key, values):
            yield "NT", sum(values)/3

        def steps(self):
            return [

                MRStep(mapper=self.mapper_AGeneration,
                       combiner=self.combiner_AGeneration,
                       reducer=self.reducer_AGeneration),

                MRStep(reducer=self.reducer_inter),

                MRStep(mapper=self.mapper_2Path,
                       reducer=self.reducer_2Path),
            ]

if __name__ == '__main__':
    MRNodesNT.run()
```

Second approach :

Before we implemented the above described algorithm, we found on the Internet that the number triangles in the Graph could be calculated as $tr(A^3/6)$ so we decided to implement a job that gets the diagonal of A^3. It ended up being a little tricky to do with our limited knowledge but we will try to explain how we implemented it.

- As always, we first have a step to create the Adjacency Matrix. It is stored in a sparse way, as a dictionary, in which each row is an entry (referenced by the number of the node). Each row is described by another dictionary that contains for every non-zero elements, the entry (node, value). It is just a way of describing the non-zero elements of the Adjacency matrix in a sparse form.
- Next task is to perform AA multiplication in a sparse way so we created a another step consisting of only one reducer that yields a (key, value) pair for every element of the output matrix to calculate. The key consists of the position of the matrix to be calculated "Nodei_Nodej" and the values is the tuple [Row_i, Col_j], lucky enough, the A matrix is symmetric so the rows and the columns are the same.
- The next step calculates the elements of the AA matrix, first, its mapper performs the sparse multiplication in an efficient way and yields a (key, value) pair consisting on ("Row_i", "Col_j":Value) in order to construct the AA in the same sparse row-oriented way as matrix A. The combiner and the reducer collects all the outputs and finally create matrix. During all the stages, we had to propagate matrix A using a special key, in order to be able to use in the future to calculate AAA.
- The next step just combines both sparse matrices AA and A under the same key so that we can have access to both of them simultaneously.
- Step number 5 is a reducer similar to the number 2, it yields a (key, value) pair for every element of the output matrix to calculate AAA. But since we only need the diagonal, it only yields those values.
- Step number 6 performs the calculation of the elements of the output matrix in the same way as step 3, but its reducer just yields the value of the diagonal elements (if they are non-zero)
- The final step is a reducer that just adds all the values and divides by 6 to finally obtain the output.

2.4 main_Traingle2.py

```
from mrjob.job import MRJob
from mrjob.step import MRStep

# This Job determines the number of triangle sin a graph using
```

```
# the trace of the A^3 (A = Adjacency matrix)

class MRNodesNT(MRJob):
    ########################################
    ## Job to create the adjacency matrix
    ########################################
    def mapper_AGeneration(self, _, line):
        # This mapper assigns per every vertex, a 1 to every nodein the vextex.
        nodes_vertex = line.split()
        if (len(nodes_vertex) == 2):
            yield nodes_vertex[0], (nodes_vertex[1],1)
            yield nodes_vertex[1], (nodes_vertex[0],1)

    def combiner_AGeneration(self, key, values):
        # The combiner does need to do anything, the nodes will have all
        # its nodes joined already
        yield key, list(values)

    def reducer_AGeneration(self, key, values):
        # Same here
        final_list = []
        for value in values:
            final_list.extend(list(value))
        final_dict = dict(final_list)

        row_elem = [key, final_dict]
        # Each row contains the number of the row and the dictionary
        # of values in sparse way. [Nrow, Dict of cols]

        yield "Matrices", row_elem  # So that the next reducer get

    ########################################
    ## Matrix multiplication
    ## We are gonna multiply the matrix A*A in sparse way,
    ########################################
    def reducer_MatrixMult(self, key, values):
        # This reducer output a par of (row_1, col_2) to multiply
        # by the combiner so that it outputs one element of the
        # output matrix
        # SINCE THE MATRIX IS SYMETRIC
        # We also want to obtain the number of nodes using the keys!
        # For knowing how many multiplications

        val_list = list(values)
        Matrix_A = dict(val_list) # We make another dictionary with the cols
        # First we yield the A matrix to perform A (A^2) later
        yield "Matrices", val_list

        keys_all = Matrix_A.keys()
        nt = len(keys_all)
        c = 1   # Just for having a sense of how much time it will take.
        for i in keys_all:
            if (c % 50 == 0):
                print str(c)+" / "+ str(nt) + " getting"
            c+= 1
            for j in keys_all:
                key_aux = i + "_" + j
                yield key_aux, [Matrix_A[i], Matrix_A[j]]

    ########################################
    ## MAPPERS are gonna do multiplications
```

```python
    #####################################
    def mapper_MatrixMult(self, key, values):
        # This mapper expects the 2 vectors to multiply.
        # The key contains the position
        # Multiply the vectors and output the value only if it is nonzero
        # The output key will be the row only.
        if (key == "Matrices"):
            yield key, list(values)  # The first time we do not do [0]  TODO

        else:
            val_list = list(values)
            v1 = val_list[0]
            v2 = val_list[1]
            #### Sparse multiplication of the vectors
            # Get the intersection of elements
            InterKeys = list(set(v1.keys()) & set(v2.keys()))

            if (len(InterKeys) > 0):  # If they have same keys in common
                total = 0
                for key_i in InterKeys:
                    total += v1[key_i] * v2[key_i]

                i = key.split("_")[0]
                j = key.split("_")[1]

                yield i, (j,total)  # Yield the result

    def combiner_AGeneration_2(self, key, values):
        # Just combine the cells to create the output matrix
        if (key == "Matrices"):
            yield key,dict(list(values)[0])
        else:
            yield key, list(values)

    def reducer_AGeneration_2(self, key, values):
        # Same here, more combinations to create the matrix
        if (key == "Matrices"):
            yield key,list(values)[0]
        else:
            # Same here
            final_list = []
            for value in values:
                final_list.extend(list(value))
            final_dict = dict(final_list)
            row_elem = [key, final_dict]
            yield "AA", row_elem  # So that the next reducer get

    #####################################
    ## JOIN MATRIX AA
    #####################################
    def reducer_AGeneration_3(self, key, values):
        if (key == "Matrices"):
            Matrix_A = list(values)[0]
            yield "Matrices",Matrix_A

        elif(key == "AA"):
            val_list = list(values)
            Matrix_AA = dict(val_list) # Make another dictionary with the cols
            yield "Matrices",Matrix_AA

    ###################################################################
```

```
    ## MULTIPLY AGAIN
    #################################################################

    def reducer_MatrixMult2(self, key, values):
        final_list = list(values)
        Matrix_A = final_list[0]
        Matrix_AA = final_list[1]
        # We are only interested in the diagonal elements so...
        # we only calculate those. We do not need to calculate the whole matrix
        print Matrix_AA.keys()
        for i in Matrix_AA.keys():
                key_aux = i + "_" + i
                yield key_aux, [Matrix_AA[i], Matrix_A[i]]

    ###################################
    #### Finally get the trace of elements
    ###################################
    def reducer_trace_rows(self, key, values):
        # Get the diagonal
        lista = list(values)[0]
        dictionary = dict(lista)

        if key in dictionary:  # Get the trace if it is non-zero
            yield None, dictionary[key]
        else:
            yield None, 0

    def reducer_trace_rows_2(self, key, values):
        # Sum t
        Trace = sum(values)
        print "Trace " + str(Trace)
        N_triangles = Trace/6

        print "Number of triangles " + str(N_triangles)
        yield "NT", N_triangles

    def steps(self):
        return [
            # Get the Adjacency Matrix, the key is the row, and it is
            # eliminated in the reduction so that  we can handle all the rows
            # in the next reduction
            MRStep(mapper=self.mapper_AGeneration,
                   combiner=self.combiner_AGeneration,
                   reducer=self.reducer_AGeneration),

#             Generate the vector multiplications
            MRStep(reducer=self.reducer_MatrixMult),
#
            # Multiply the matrices and obtain the result,
            # While storing the initial matrix
            MRStep(mapper=self.mapper_MatrixMult,
                   combiner=self.combiner_AGeneration_2,
                   reducer=self.reducer_AGeneration_2),
#
            # We join the AA matrix and the A matrix
            MRStep(reducer=self.reducer_AGeneration_3),
            MRStep(reducer=self.reducer_MatrixMult2),
#
            # We multiply them again
            MRStep(mapper=self.mapper_MatrixMult,
                   combiner=self.combiner_AGeneration_2,
```

```
                        reducer = self.reducer_trace_rows),

                # Add all the traces and divide by 6
                MRStep(reducer=self.reducer_trace_rows_2),
            ]

if __name__ == '__main__':
    MRNodesNT.run()
```