Big Data Mining - Assignment 2

Clayton Preston

Data Preparation

The aim of this project is to take data from CSV files and use them to create a graph database using neo4j. To assist in this process, I decided to use python programming language to manipulate the data before loading them into the graph db.

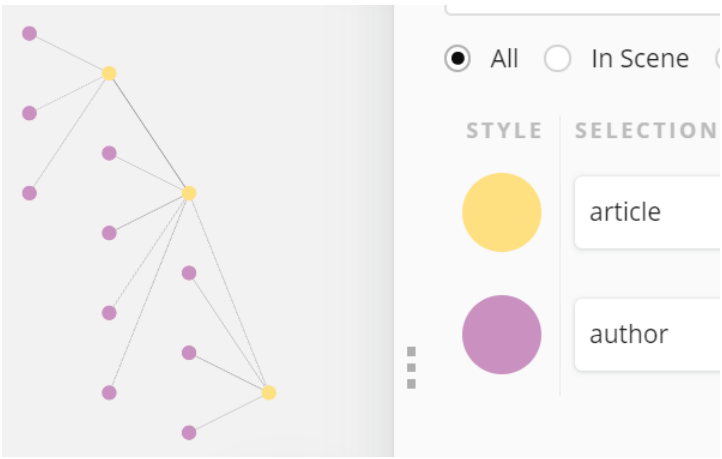
Being brand new to neo4j and the cypher language, I attempted several iterations of data loading before I could finally load them in successfully. One attempt, which I still cannot quite understand how it failed, involved looping through many CREATE statements one after another. This would result in some data duplication. Let's see an example of the erroneous data loading:

For instance: If I wanted to create a simple graph with ***node A***, ***node B*** and ***relationship A to B*** - running multiple CREATE statements would result in a single node A (no relationship), a single node B (no relationship) and a generic relationship "to" between two unidentified nodes.

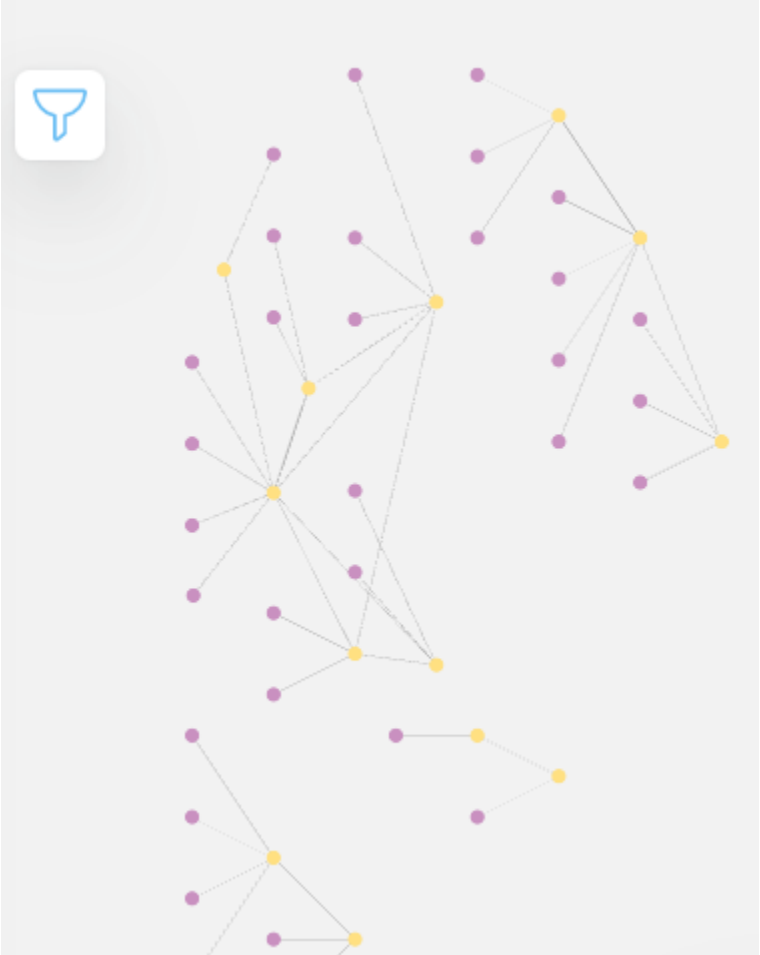
I finally was able to successfully load the data in with one single CREATE statement. However, after several memory errors attempting to load the full dataset, I gave up and decided to just take the first 10,000 rows from each CSV – Articles, Authors and Citations. Given more time, I would like to explore looping through CREATE statements or utilizing MERGE to load the full dataset as querying was limited by this factor which we will see further in the assignment.

Graph Concept

While there are a multitude of ways you could graph this information, we will proceed with the method which I believe will make the data retrieval as simple as possible. We will create two ***nodes*** and two *relationships*; ***articles*** and ***authors***, and *cites* and *writes* respectively. Articles can cite other articles while authors can write articles.



Example article and author nodes in neo4j bloom



A visual sample of the graph

Articles

I first loaded the articles csv and did some minor text clean up to insure there were no strange characters.

Articles

```
In [4]: art = pd.read_csv('Citation Dataset\\ArticleNodes.csv', header=None, nrows =10000)
art.columns = ['id','title','year','journal','abstract']
art['title'] = art['title'].str.replace('"','').replace("'",'').replace('`','').replace('{','').replace('}','').replace('\\','').
art['abstract'] = art['abstract'].str.replace('"','').replace("'",'').replace('`','').replace('{','').replace('}','').replace('\\
art.head() # 29555
```

Out[4]:

	id	title	year	journal	abstract
0	1001	Compactification Geometry and Duality: N=2	2000	NaN	These are notes based on lectures given at T...
1	1002	Domain Walls and Massive Gauged Supergravity P...	2000	Class.Quant.Grav.	We point out that massive gauged supergravit...
2	1003	Comment on Metric Fluctuations in Brane Worlds	2000	NaN	Recently- Ivanov and Volovich (hep-th/991224...
3	1004	Moving Mirrors and Thermodynamic Paradoxes	2000	Phys.Rev.	Quantum fields responding to moving mirrors...
4	1005	Bundles of chiral blocks and boundary conditio...	2000	NaN	Various aspects of spaces of chiral blocks a...

Articles CSV loaded into a pandas dataframe in python

Knowing that I would ultimately use one single string with CREATE at the beginning to load the data, I started to compile all the commands I was planning to run into a list. After several attempts, I continued to get errors from neo4j regarding the abstract attribute despite all the cleaning I did so in the interest of time, I left the abstract attribute out of my model.

```
In [5]: createlist = []
for index, row in art.iterrows():
    create = '(ID' + str(row['id']) + ":article { id: 'ID" + str(row['id']) + "', title: '" + str(row['title'])+ "', year: '" + s
    createlist.append(create)

createlist
```

```
Out[5]: ["(ID1001:article { id: 'ID1001', title: 'Compactification  Geometry and Duality: N=2', year: '2000', journal: 'nan'})",
 "(ID1002:article { id: 'ID1002', title: 'Domain Walls and Massive Gauged Supergravity Potentials', year: '2000', journal: 'C
lass.Quant.Grav.'})",
 "(ID1003:article { id: 'ID1003', title: 'Comment on  Metric Fluctuations in Brane Worlds ', year: '2000', journal: 'nan'})",
 "(ID1004:article { id: 'ID1004', title: 'Moving Mirrors and Thermodynamic Paradoxes', year: '2000', journal: 'Phys.Rev.'})",
 "(ID1005:article { id: 'ID1005', title: 'Bundles of chiral blocks and boundary conditions in CFT', year: '2000', journal: 'n
an'})",
 "(ID1006:article { id: 'ID1006', title: 'Questions in quantum physics: a personal view', year: '2000', journal: 'nan'})",
 "(ID1007:article { id: 'ID1007', title: 'Topological Defects in 3-d Euclidean Gravity', year: '2000', journal: 'nan'})",
 "(ID1008:article { id: 'ID1008', title: 'N=0 Supersymmetry and the Non-Relativistic Monopole', year: '2000', journal: 'Phys.
Lett.'})",
```

Creating a list of each CREATE query for all article nodes

Next, I loaded in the Citations dataset and created relationship queries from them. Here we name the relationship “cites” and would read the graph as “node X (an article) *cites* node Y (an article).”

Citation Relationships

```
cit = pd.read_csv('Citation Dataset\\Citations.csv', sep = '\t', header=None, nrows = 10000)
cit.columns = ['id1','id2']
cit['id1'] = cit['id1'].astype(str).str.strip()
cit['id2'] = cit['id2'].astype(str).str.strip()
cit.head()
```

	id1	id2
0	1001	9304045
1	1001	9308122
2	1001	9309097
3	1001	9311042
4	1001	9401139

Loading the Citations csv into a pandas dataframe in python

```
for index, row in cit.iterrows():
    rel = '(ID' + row['id1'] + ')-[:cites]->(ID' + row['id2'] + ')'
    createlist.append(rel)

createlist
```

```
["(ID1001:article { id: 'ID1001', title: 'Compactification  Geometry and Duality: N=2', year: '2000', journal: 'nan'})",
 "(ID1002:article { id: 'ID1002', title: 'Domain Walls and Massive Gauged Supergravity Potentials', year: '2000', journal: 'C
lass.Quant.Grav.'})",
 "(ID1003:article { id: 'ID1003', title: 'Comment on  Metric Fluctuations in Brane Worlds ', year: '2000', journal: 'nan'})",
 "(ID1004:article { id: 'ID1004', title: 'Moving Mirrors and Thermodynamic Paradoxes', year: '2000', journal: 'Phys.Rev.'})",
 "(ID1005:article { id: 'ID1005', title: 'Bundles of chiral blocks and boundary conditions in CFT', year: '2000', journal: 'n
an'})",
```

Creating cites relationships and appending them to the createlist

```
createlist[-1:]
```

```
['(ID9905006)-[:cites]->(ID9901042)']
```

An example of the citation query

The articles and their relationships have now been created. Finally, we will create author nodes and relate them to the works they have written. Here the id attribute given to an author is the article id which they have written.

Author

```
aut = pd.read_csv('Citation Dataset\\AuthorNodes.csv', header=None, nrows=10000)
aut = aut.reset_index()
aut.columns = ['autid','id','author']
aut['autid'] = aut['autid'].astype(str).str.strip()
aut['id'] = aut['id'].astype(str).str.strip()
aut.head()
```

	autid	id	author
0	0	1001	Paul S. Aspinwall
1	1	1002	C.N. Pope
2	2	1002	M. Cvetic
3	3	1002	H. Lu
4	4	1003	Y.S. Myung

```
for index, row in aut.iterrows():
    create = '(aID' + str(row['autid']) + ":author { id: 'ID" + str(row['id']) + "', author: '" + str(row['author']) + "'})"
    createlist.append(create)
```

Creating the author nodes from the pandas dataframe

```
createlist[-1:]
```

```
["(aID9999:author { id: 'ID107200', author: 'Simeon Hellerman'})"]
```

An example of an author node

```
for index, row in aut.iterrows():
    rel = '(aID' + row['autid'] + ')-[:writes]->(ID' + row['id'] + ')'
    createlist.append(rel)
```

Creating queries for the writes relationships between authors and articles

```
createlist[-1:]
```

```
['(aID9999)-[:writes]->(ID107200)']
```

An example of a writes relationship query

All the CQL statements we wish to execute in neo4j have been created and are now stored in a list called createlist. To run them in neo4j, we need them to all be in one single string according to the syntax as follows: 'CREATE (query), (query), … '

```
# List to string with commas
createstring = ','.join(createlist)
createstring = 'CREATE ' + createstring
createstring

# insert CREATE
```

```
"CREATE (ID1001:article { id: 'ID1001', title: 'Compactification  Geometry and Duality: N=2', year: '2000', journal: 'nan'}),
(ID1002:article { id: 'ID1002', title: 'Domain Walls and Massive Gauged Supergravity Potentials', year: '2000', journal: 'Cla
```

Joining all queries into one CREATE statement

With the CREATE query now in a string, we use the neo4j python library and GraphDatabase to open an instance of our database and run our query in it. After a while, this will populate the data into the desired graph format in neo4j.

```
import pandas as pd
from neo4j import GraphDatabase
```

```
uri = "neo4j://localhost:7687"
conn = GraphDatabase.driver(uri, auth=("neo4j", "Ass2DB"), max_connection_lifetime=10000)
```

```
with conn.session() as graphDB_Session:
    # Create nodes
    graphDB_Session.run(createstring)
```

Running the CREATE statement in neo4j

Querying the Data

We now turn from python and neo4j bloom to neo4j browser to run the queries.

1.

Which are the top 5 authors with the most citations (from other papers). Return author names and number of citations.

Here we find articles which cite articles, authors who write the latter articles and return those authors along with a count of the number of citations the articles which they have written have.

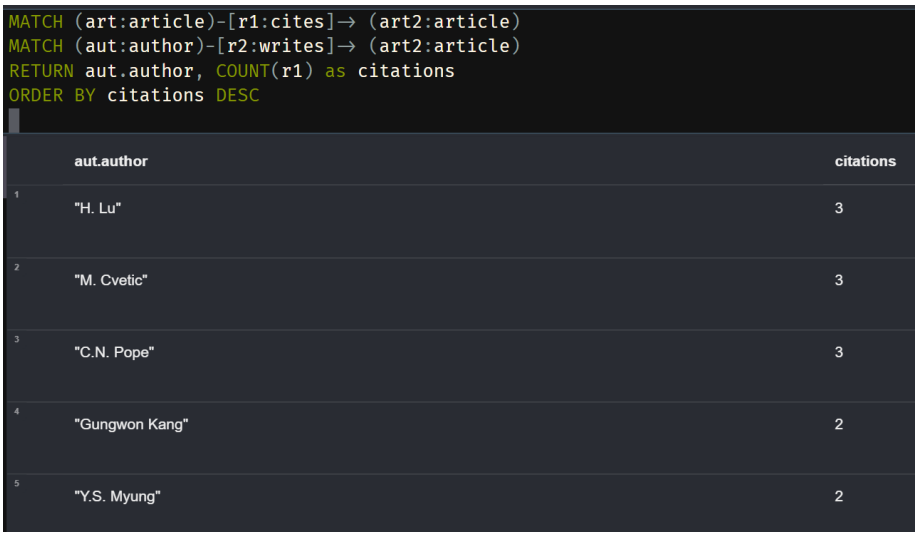
MATCH (art:article)-[r1:cites]-> (art2:article)

MATCH (aut:author)-[r2:writes]-> (art2:article)

RETURN aut.author, COUNT(r1) as citations

ORDER BY citations DESC

LIMIT 5



```
MATCH (art:article)-[r1:cites]→ (art2:article)
MATCH (aut:author)-[r2:writes]→ (art2:article)
RETURN aut.author, COUNT(r1) as citations
ORDER BY citations DESC
```

	aut.author	citations
1	"H. Lu"	3
2	"M. Cvetic"	3
3	"C.N. Pope"	3
4	"Gungwon Kang"	2
5	"Y.S. Myung"	2

2.

Which are the top 5 authors with the most collaborations (with different authors). Return author names and number of collaborations

Here we find authors who write articles and a count of all authors who have written the same article as our original author. This will tell us which authors have written articles with the most other authors.

MATCH (a:author)-[:writes]->(b)

, ()-[r:writes]->(b)

return a.author, count(r) as collabs

ORDER BY collabs DESC

Limit 5;

```
MATCH (a:author)-[:writes]→(b)
, ()-[r:writes]→(b)
return a.author, count(r) as collabs
ORDER BY collabs DESC
Limit 5;
```

	a.author	collabs
1	"M. Cvetic"	59
2	"H. Lu"	59
3	"C.N. Pope"	59
4	"Shinichi Nojiri"	45
5	"Sergei D. Odintsov"	42

3.

Which is the author who has written the most papers without collaborations. Return author name and number of papers.

We look for authors who write articles, and the number of authors who have also written that article. We restrict that number to 1 to ensure that this author is the only author.

MATCH (a:author)-[:writes]-(b:article),()-[r:writes]-(b)

with a,b,count(r) as otherauthors

where otherauthors <= 1

RETURN a.author, count(b)

order by count(b) desc

Limit 1

```
MATCH (a:author)-[:writes]-(b:article),()-[r:writes]-(b)
with a,b,count(r) as otherauthors
where otherauthors <= 1
RETURN a.author, count(b)
order by count(b) desc
Limit 1
```

	a.author	count(b)
1	"Sergei D. Odintsov"	11

4.

Which author published the most papers in 2001? Return author name and number of papers.

For this simple query, we find authors who have written articles, specifying only articles with the year 2001, and count the number of writes relationships; returning that author and the number of papers they have published in that year.

MATCH (a:author)-[r:writes]->(b:article)

where b.year = '2001'

RETURN a.author, count(r) as numbofpapers

order by numbofpapers desc

limit 1

```
MATCH (a:author)-[r:writes]→(b:article)
where b.year = '2001'
RETURN a.author, count(r) as numbofpapers
order by numbofpapers desc
limit 1
```

	a.author	numbofpapers
1	"Donam Youm"	9

5.

Which is the journal with the most papers about “gravity” (derived only from the paper title) in 1998. Return name of journal and number of papers.

Here we are looking for articles which have specific attributes; articles containing the word gravity in their title. 1998 did not yield any results, but 2001 did so we will examine both.

MATCH (b:article)

where b.year = '1998' and b.title contains 'gravity'

return b.journal, count(*) as numbjourns

order by numbjourns desc

```
MATCH (b:article)
where b.year = '1998' and  b.title contains 'gravity'
return b.journal, count(*) as numbjourns
order by numbjourns desc
```

(no changes, no records)

MATCH (b:article)

where b.year = '2001' and b.title contains 'gravity'

return b.journal, count(*) as numbjourns

order by numbjourns desc

```
MATCH (b:article)
where b.year = '2001' and  b.title contains 'gravity'
return b.journal, count(*) as numbjourns
order by numbjourns desc
```

	b.journal	numbjourns
1	"JHEP"	21

6.

Which are the top 5 papers with the most citations? Return paper title and number of citations

We look for articles which are cited and return those which have the highest number of cited relationships.

MATCH ()-[r:cites]->(b:article)

return b.title, count(r) as citations

order by citations desc

limit 5

```
MATCH ()-[r:cites]→(b:article)
return b.title, count(r) as citations
order by citations desc
limit 5
```

	b.title	citations
1	"Open strings and their symmetry groups"	20
2	"An Early Proposal of Brane World "	3
3	"Domain Walls and Massive Gauged Supergravity Potentials"	3
4	"Comment on Metric Fluctuations in Brane Worlds "	2
5	"Universal Aspects of Gravity Localized on Thick Branes"	2

7.

Which were the papers that use “holography” and “anti de sitter” (derived only from the paper abstract). Return authors and title.

As we weren’t able to include abstract in our dataset, this query cannot be executed, but this is the query which should be were it possible. We search for articles written by authors specifying the abstract of these articles to contain holography and anti de sitter.

MATCH (b:article)<-[:writes]-(a:author)

where b.abstract contains 'holography' and b.abstract contains 'anti de sitter'

return a.author, b.title

```
MATCH (b:article)←[:writes]-(a:author)
where b.abstract contains 'holography' and b.abstract contains 'anti de sitter'
return a.author, b.title
```

(no changes, no records)

8.

Find the shortest path between ‘C.N. Pope’ and ‘M. Schweda’ authors (use any type of edges). Return the path and the length of the path. Comment about the type of nodes and edges of the path.

Again, with the shortened dataset, C.N. Pope and M. Schweda did not give us any results. In this case, I found a pair of authors that were connected and decided to analyze them instead. We can interpret the *null* nodes as relationships which were created without nodes being specified.

MATCH p = shortestPath((c:author{author:'Yuri Shirman'})-[*]-(f:author{author:'H. Lu'}))

RETURN [n in nodes(p) | n.author] AS ShortestPath, length(p) as Length

```
MATCH p = shortestPath((c:author{author:'Yuri Shirman'})-[*]-(f:author{author:'H. Lu'}))
RETURN [n in nodes(p) | n.author] AS ShortestPath, length(p) as Length
```

	ShortestPath	Length
1	["Yuri Shirman", *null*, *null*, "H. Lu"]	3

9.

Run again the previous query (8) but now use only edges between authors and papers. Comment about the type of nodes and edges of the path. Compare the results with query 8.

Here we change the relationship type to specify only writes relationships (edges between authors and papers). As you can see we yielded no results; likely, again unfortunately, due to the shortened dataset.

MATCH p = shortestPath((c:author{author:'Yuri Shirman'})-[r:writes]-(f:author{author:'H. Lu'}))

RETURN [n in nodes(p) | n.author] AS ShortestPath, length(p) as Length

```
MATCH p = shortestPath((c:author{author:'Yuri Shirman'})-[r:writes]-(f:author{author:'H. Lu'}))
RETURN [n in nodes(p) | n.author] AS ShortestPath, length(p) as Length
```

(no changes, no records)

10.

Find all authors with shortest path lengths > 25 from author ‘Edward Witten’. The shortest paths will be calculated only on edges between authors and articles. Return author name, the length and the paper titles for each path.

MATCH p = (c:author{author:'Edward Witten'})-[:writes*25]-(f:author)

with f.author as author, length(p) as leng, f.title as title, relationships(p) as r

return author, leng, title