Neo4j Graph Algorithms Library Extension

R07902043 易大中 R08921089 蘇彥齊 R08944019 王郁婷 R08946007 蔡昀達

Introduction

The objective of our project is to extend the neo4j graph algorithm library and contribute to the neo4j open source project. By this opportunity of developing some new features in the neo4j algorithm library, we are able to understand the entire structure and workflow of a production database.

Claim: Although it is our proposal to contribute to the neo4j project, we still emphasize that our engineering effort are way beyond others which no teams could compare. We will list our main effort and several difficulties we encountered.

- Neo4j has no documents for their internal database codes and graph library.
- Unlike numerous online tutorial for mysql storage engines, there are barely any online resources or examples for developing neo4j.
- The open source repository are not fully updated to the latest release such that we have to supplement some part of codes from scratch to fix the completeness of the database.
- To understand the workflow of neo4j, each of our team member spend more than **25** hours tracing codes in over **300** files and over **3 millions** lines from scratch before fulfilling the division of work.
- None of us are familiar with java and neo4j develop environment before this project.
- We spend more than a hundred hours on this project in total.

Neo4j Procedure Template

The neo4j procedure template are designed for customized code. It provides pom.xml file for Apache Maven to build the entire project as a jar-file plugin. The basic dependencies for connecting neo4j database are pre-build in the template.
However, even with the template, we had to understand the workflow and dataflow of the neo4j database before start to write custom codes.

Approximate Nearest Neighbors

- **Motivation**

The original procedure in neo4j graph algorithm library has a great drawback of re-calculating the nearest neighbor every time the graph updates with a high complexity of O(n^2d) and O(nkd) with approximation. The situation of frequently graph updating is common in many real word scenarios and re-calculating will cause high overhead cost. We define the problem as how to efficiently dynamic update the similarity graph.

- **Constraint**

The data structure for maintaining the nearest neighbor search will not be available after the procedure is called and only topk relationships will remain.

1. No data structures are available such as kd-tree or the R-tree that supports dynamic context.
2. The topk relationship is only an incomplete distance matrix which made most existing algorithms and data structures for nearest neighbor search unsuitable in this condition.

- **Solution**

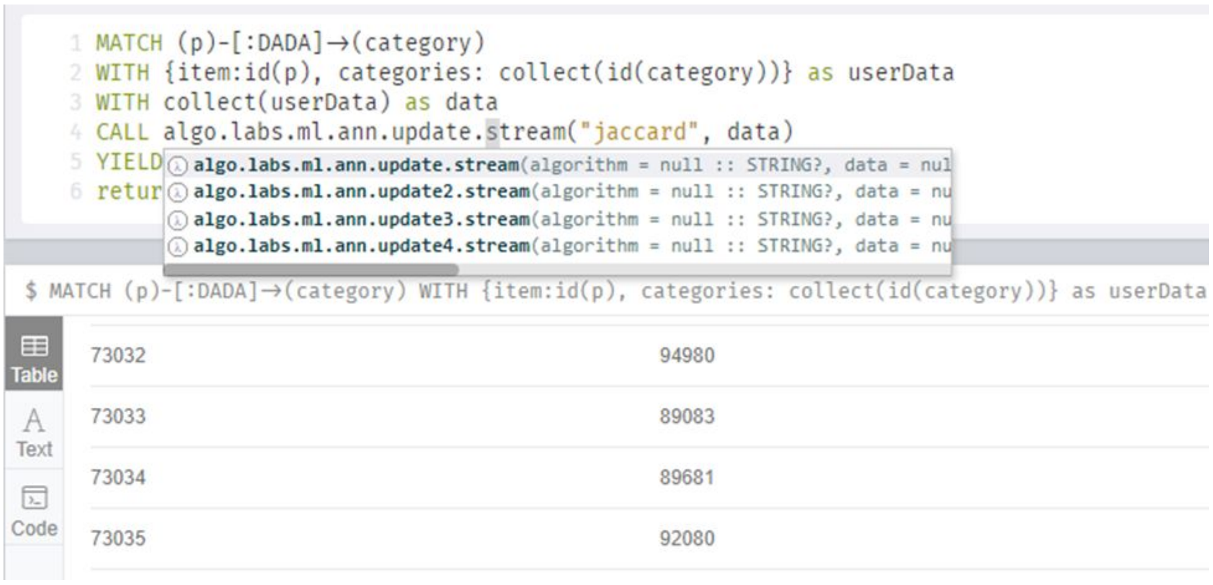
After a long brainstorming and paper survey, we are inspired by the HNSW[1] algorithm (2016) that based on greedy graph traversing which is considered the current state-of-the-art for the approximate nearest neighbors search on proximity graph. We ingeniously turn this constrained nearest neighbor search problem into a proximity neighborhood graphs problem. Instead of updating the entire graph, we build the existing similarity relationships into a proximity graph and perform approximate nearest neighbors search on the similarity relationship graph. The reduction of this algorithm perfectly meets the constraints and provides a near constant complexity O(mkd) where m << n. **We came up with this novel solution all by ourselves and we believe that we have achieved an optimal solution for this problem in both time and memory complexity**.

- **Implemetation**

The approximate nearest neighbor search is in the latest release of neo4j graph algorithm library which is not yet open-sourced and there are no open-sourced java implementations for HNSW algorithm. Therefore, we implement the approximate nearest neighbor algorithm and the HNSW algorithm all from scratch with over a thousand line of codes.

The interface and the parameters of our implementation are designed exactly the same as the original one in neo4j algorithm library. Users would need no effort to use the interface.

[1] Malkov, Yury; Yashunin, Dmitry (2016). "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs". arXiv:1603.09320



```
MATCH (p)-[:DADA]→(category)
WITH {item:id(p), categories: collect(id(category))} as userData
WITH collect(userData) as data
CALL algo.labs.ml.ann.update.stream("jaccard", data)
YIELD
retur
```

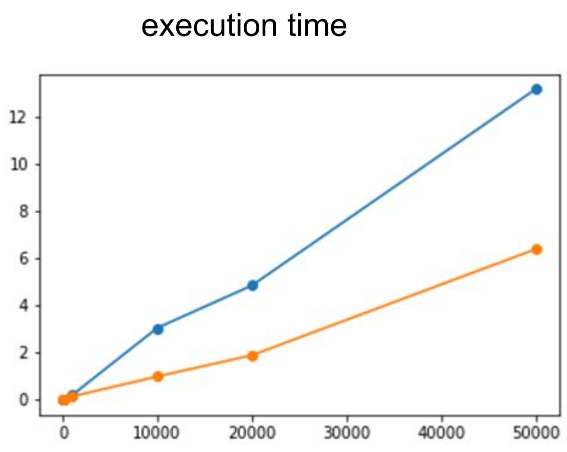
```
algo.labs.ml.ann.update.stream(algorithm = null :: STRING?, data = nul
algo.labs.ml.ann.update2.stream(algorithm = null :: STRING?, data = nu
algo.labs.ml.ann.update3.stream(algorithm = null :: STRING?, data = nu
algo.labs.ml.ann.update4.stream(algorithm = null :: STRING?, data = nu
```

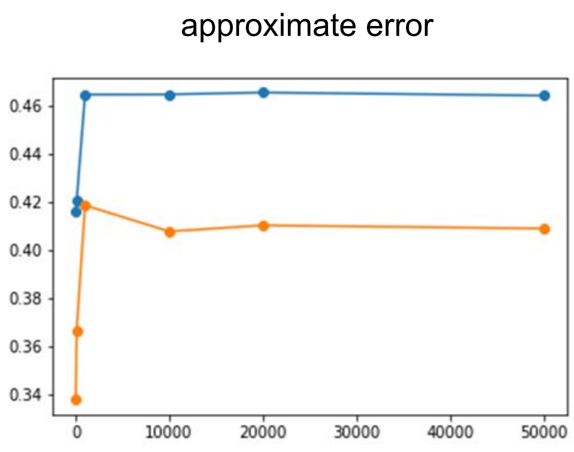
```
$ MATCH (p)-[:DADA]→(category) WITH {item:id(p), categories: collect(id(category))} as userData
```

73032	94980
73033	89083
73034	89681
73035	92080

- **Performance**

We test both our execution time and approximation error which both show improvements.





Group Centralization

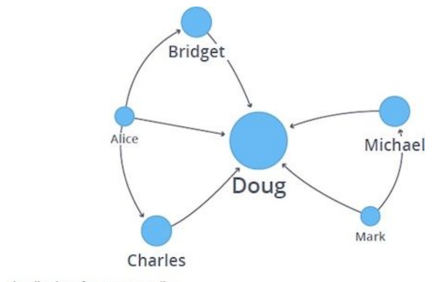
- **Why?**

Existing neo4j library of centrality allows comparison of node level within the same network but does not on a network / graph level. This becomes a problem when user tries to compare several node groups using different tag values.
With Group level measurements, comparing different networks could be done easily.

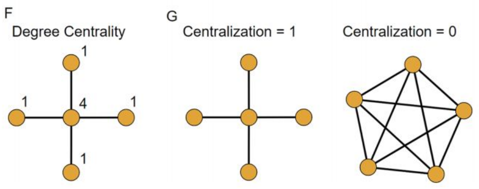
- **Group level index of centralization**

Group level index of centralization shows that the larger it is, the more likely it is that a single node is quite central, with the remaining nodes considerably less central.

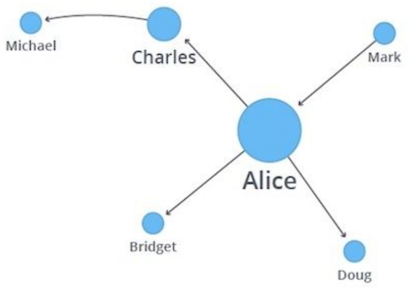
- **Group Degree Centralization**



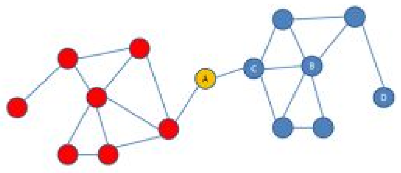
Visualization of Degree Centrality



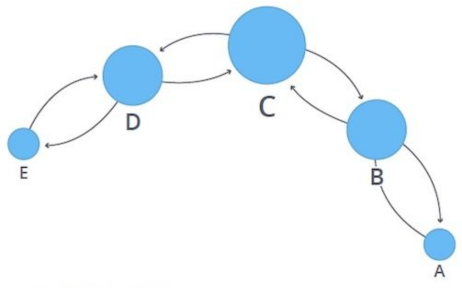
- **Group Betweenness Centralization**



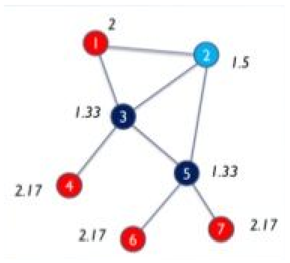
Visualization of Betweenness Centrality



- **Group Closeness Centralization**



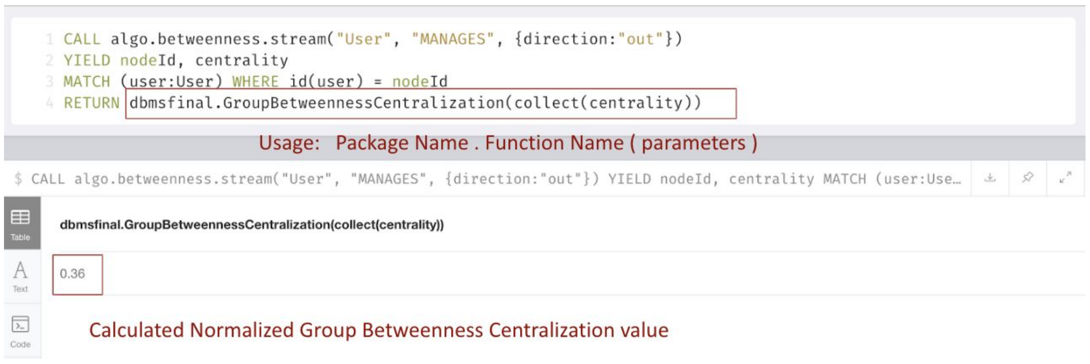
Visualization of Closeness Centrality



- **Group Centralization Usage**

The usage of our implemented user defined function is shown as the picture, calling the package name.function name(), and pass in the arguments needed. Here we pass in the the results of neo4j library centrality calculation of each node, and use them to calculate the according group centralization.

The number we return would be the normalized centralization value of the group betweenness centralization. The other two function usage should be similar.



```
CALL algo.betweenness.stream("User", "MANAGES", {direction:"out"})
YIELD nodeId, centrality
MATCH (user:User) WHERE id(user) = nodeId
RETURN dbmsfinal.GroupBetweennessCentralization(collect(centrality))
```

Usage: Package Name . Function Name (parameters)

```
$ CALL algo.betweenness.stream("User", "MANAGES", {direction:"out"}) YIELD nodeId, centrality MATCH (user:Use…
```

dbmsfinal.GroupBetweennessCentralization(collect(centrality))
0.36

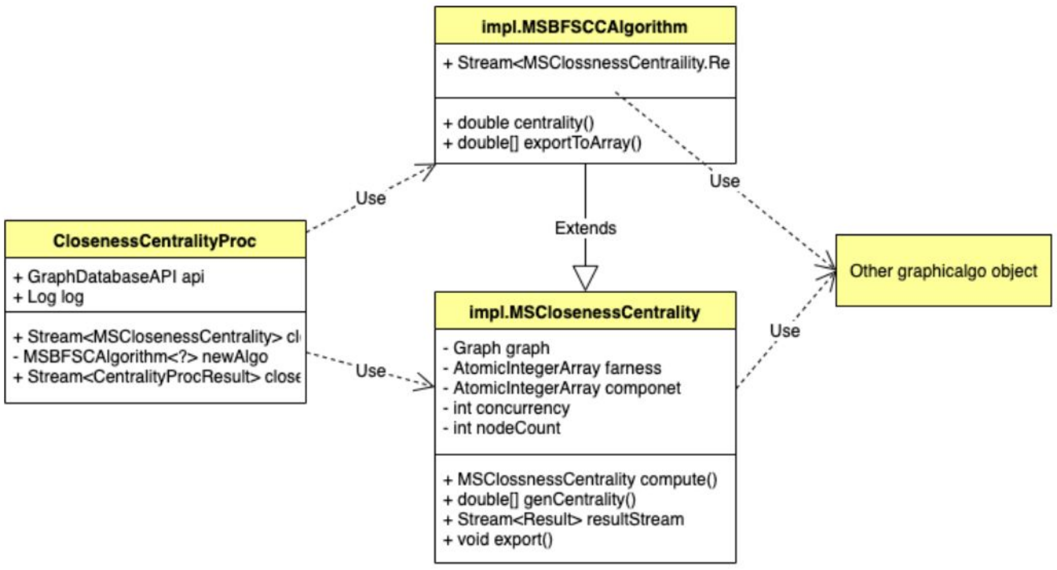
Calculated Normalized Group Betweenness Centralization value

With the 3 functions we implement, user will be able to compare the difference and similarity between networks under 3 most useful perspectives.

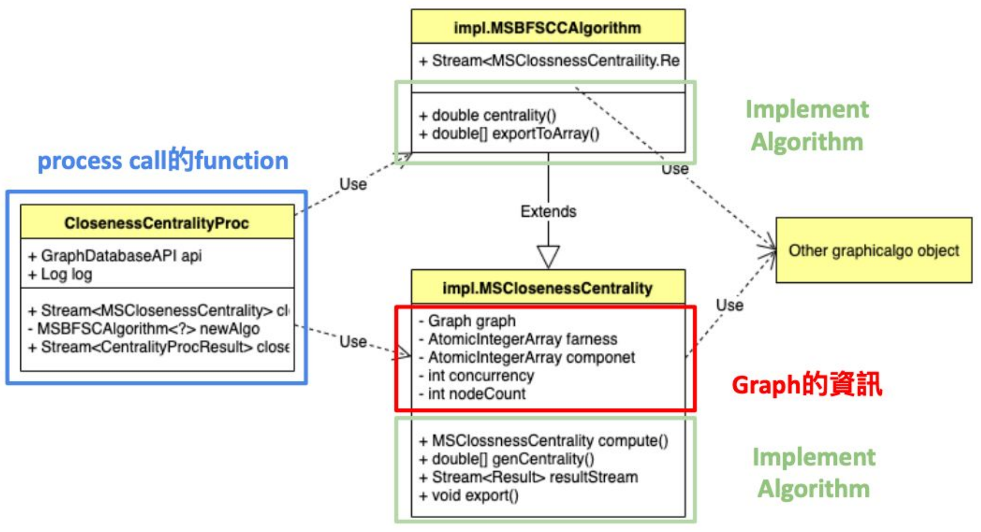
Neo4j Graph Algorithm Extension

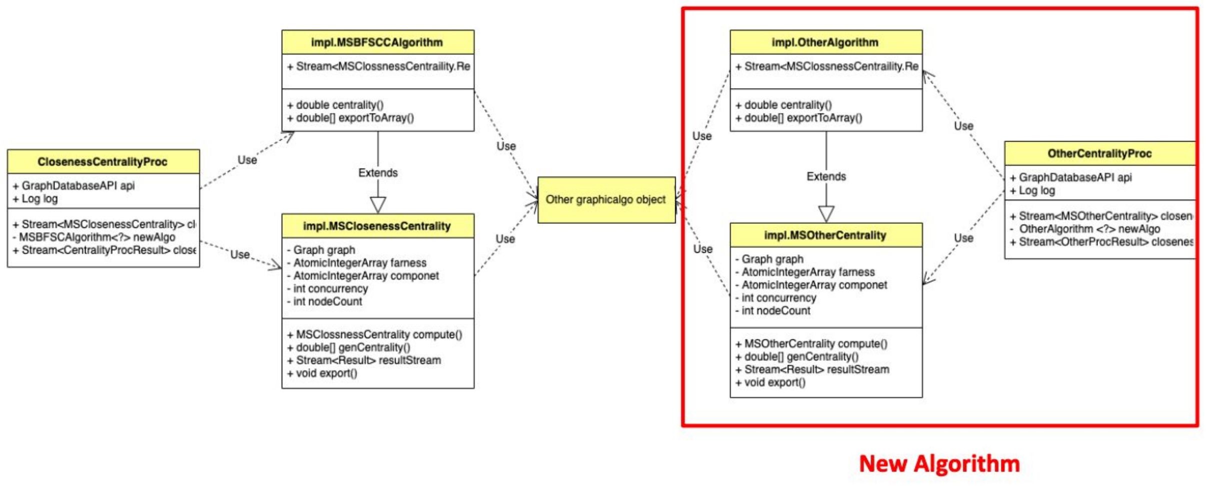
Not all algorithm can be implemented by procedure template, so we directly implement in Neo4j graph algorithm package.

- **Neo4j Graph Algorithm Extension:**



impl.MSBFSCCAlgorithm
+ Stream<MSClossnessCentraility.Re
+ double centrality()
+ double[] exportToArray()
Use
Use
Extends
ClosenessCentralityProc
+ GraphDatabaseAPI api
+ Log log
+ Stream<MSClosenessCentrality> cl
- MSBFSCAlgorithm<?> newAlgo
+ Stream<CentralityProcResult> close
Use
impl.MSClosenessCentrality
- Graph graph
- AtomicIntegerArray farness
- AtomicIntegerArray componet
- int concurrency
- int nodeCount
+ MSClossnessCentrality compute()
+ double[] genCentrality()
+ Stream<Result> resultStream
+ void export()
Use
Other graphicalgo object


impl.MSBFSCCAlgorithm
+ Stream<MSClossnessCentraility.Re
+ double centrality()
+ double[] exportToArray()
Implement Algorithm
process call的function
Use
Use
Extends
ClosenessCentralityProc
+ GraphDatabaseAPI api
+ Log log
+ Stream<MSClosenessCentrality> cl
- MSBFSCAlgorithm<?> newAlgo
+ Stream<CentralityProcResult> close
Use
impl.MSClosenessCentrality
- Graph graph
- AtomicIntegerArray farness
- AtomicIntegerArray componet
- int concurrency
- int nodeCount
Use
Other graphicalgo object
Graph的資訊
+ MSClossnessCentrality compute()
+ double[] genCentrality()
+ Stream<Result> resultStream
+ void export()
Implement Algorithm

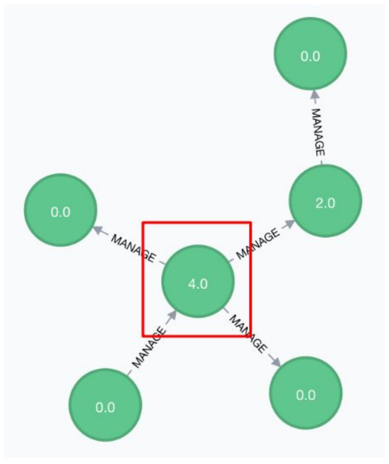

impl.MSBFSCCAlgorithm
+ Stream<MSClossnessCentraility.Re
+ double centrality()
+ double[] exportToArray()
ClosenessCentralityProc
+ GraphDatabaseAPI api
+ Log log
+ Stream<MSClosenessCentrality> cl
- MSBFSCAlgorithm<?> newAlgo
+ Stream<CentralityProcResult> close
Use
Extends
Use
Use
Use
impl.MSClosenessCentrality
- Graph graph
- AtomicIntegerArray farness
- AtomicIntegerArray componet
- int concurrency
- int nodeCount
+ MSClossnessCentrality compute()
+ double[] genCentrality()
+ Stream<Result> resultStream
+ void export()
Other graphicalgo object
impl.OtherAlgorithm
+ Stream<MSClossnessCentraility.Re
+ double centrality()
+ double[] exportToArray()
Use
Use
Extends
Use
Use
OtherCentralityProc
+ GraphDatabaseAPI api
+ Log log
+ Stream<MSOtherCentrality> closen
- OtherAlgorithm <?> newAlgo
+ Stream<OtherProcResult> closenes
impl.MSOtherCentrality
- Graph graph
- AtomicIntegerArray farness
- AtomicIntegerArray componet
- int concurrency
- int nodeCount
+ MSOtherCentrality compute()
+ double[] genCentrality()
+ Stream<Result> resultStream
+ void export()
New Algorithm

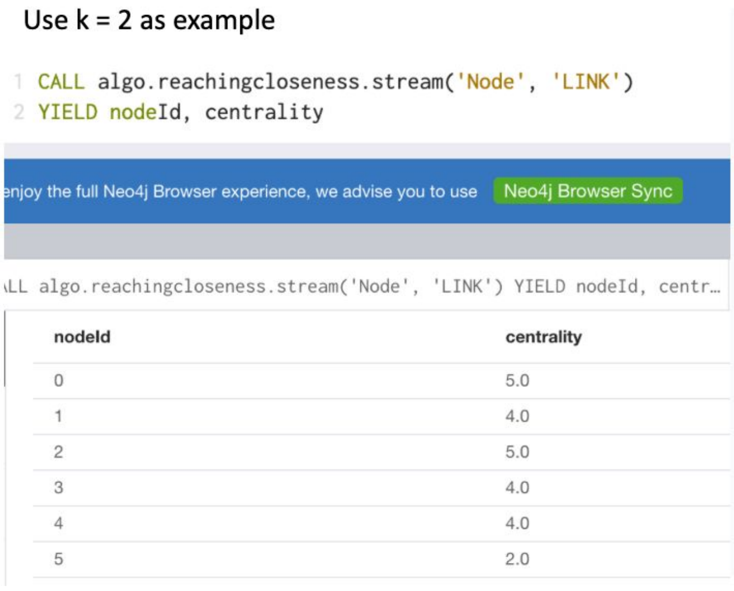
1. **Reach Centrality**

Counts the number of nodes each node can reach in k or less steps.

For this node:

- when k = 1, reaching centrality = 4.
- when k = 2, reaching centrality = 5.



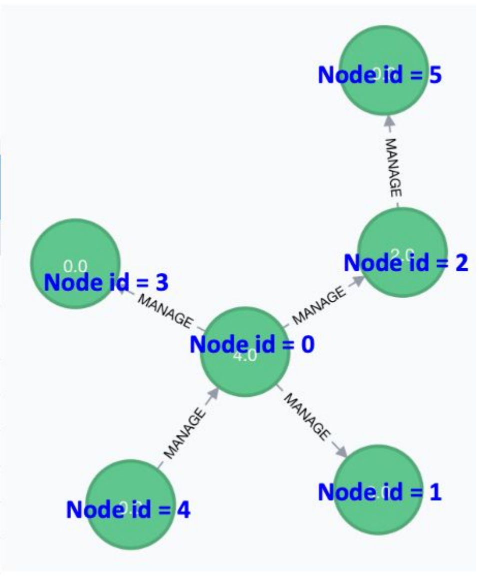

Use k = 2 as example

```
CALL algo.reachingcloseness.stream('Node', 'LINK')
YIELD nodeId, centrality
```

enjoy the full Neo4j Browser experience, we advise you to use
Neo4j Browser Sync

```
LL algo.reachingcloseness.stream('Node', 'LINK') YIELD nodeId, centr…
```

nodeId	centrality
0	5.0
1	4.0
2	5.0
3	4.0
4	4.0
5	2.0



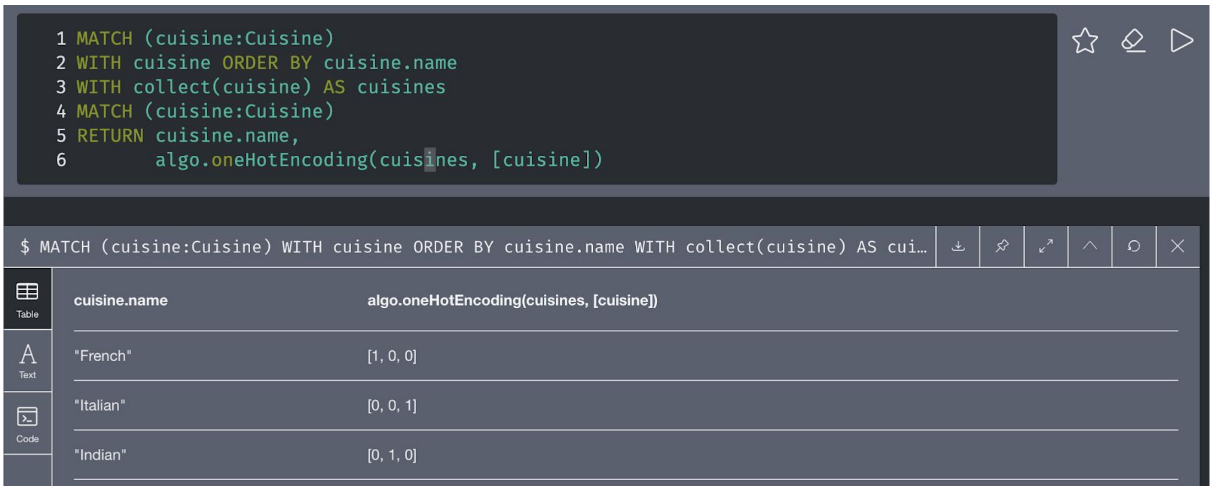
2. Encoding

Preprocessing functions and procedures in the Neo4j Graph Algorithm Library.

- **OneHot Encoding**

This method produces a vector with length equal to the number of categories in the data set. If a data point belongs to the i-th category then components of this vector are assigned the value 0 except for the i-th component, which is assigned a value of 1.

In this way one can keep track of the categories in a numerically meaningful way.



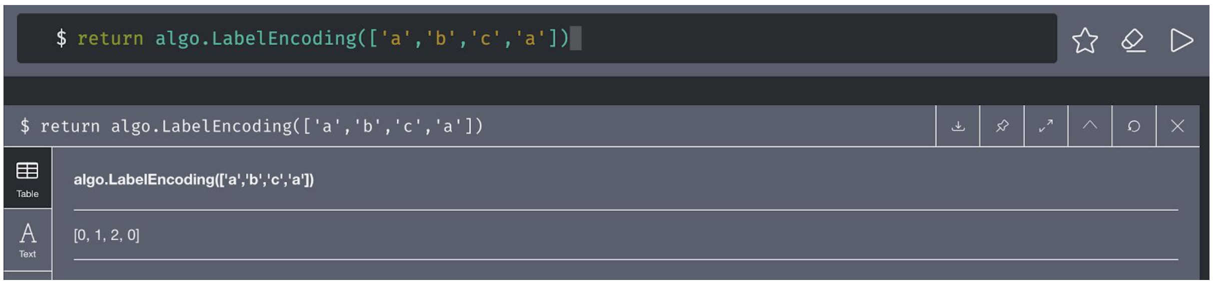
```
MATCH (cuisine:Cuisine)
WITH cuisine ORDER BY cuisine.name
WITH collect(cuisine) AS cuisines
MATCH (cuisine:Cuisine)
RETURN cuisine.name,
       algo.oneHotEncoding(cuisines, [cuisine])
```

```
$ MATCH (cuisine:Cuisine) WITH cuisine ORDER BY cuisine.name WITH collect(cuisine) AS cui…
```

cuisine.name	algo.oneHotEncoding(cuisines, [cuisine])
"French"	[1, 0, 0]
"Italian"	[0, 0, 1]
"Indian"	[0, 1, 0]

- **Label Encoding**

In this encoding each category is assigned a value from 1 through N (here N is the number of category for the feature).
There is no relation or order between these classes but algorithm might consider them as some kind of order or there is some kind of relationship.

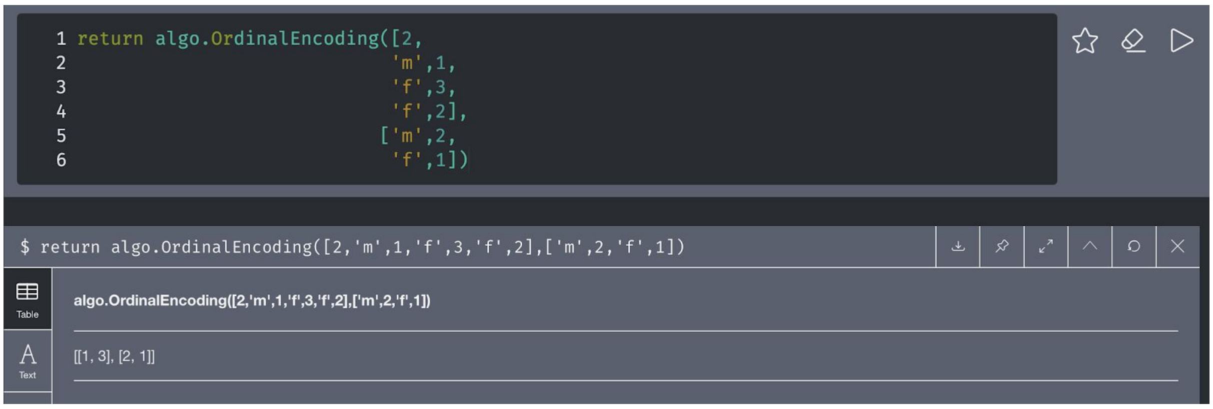


```
$ return algo.LabelEncoding(['a','b','c','a'])
```

```
$ return algo.LabelEncoding(['a','b','c','a'])
algo.LabelEncoding(['a','b','c','a'])
[0, 1, 2, 0]
```

- **Ordinal Encoding**

This encoding looks almost similar to Label Encoding but slightly different as Label coding would not consider whether variable is ordinal or not and it will assign sequence of integers.

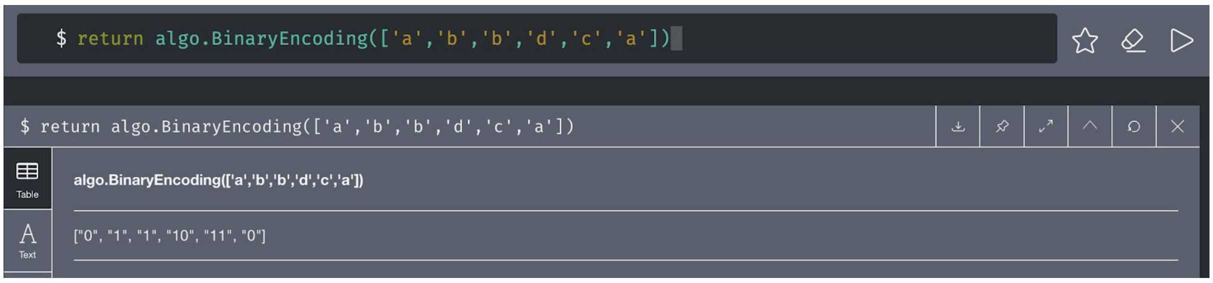


```
return algo.OrdinalEncoding([2,
                              'm',1,
                              'f',3,
                              'f',2],
                             ['m',2,
                              'f',1])
```

```
$ return algo.OrdinalEncoding([2,'m',1,'f',3,'f',2],['m',2,'f',1])
algo.OrdinalEncoding([2,'m',1,'f',3,'f',2],['m',2,'f',1])
[[1, 3], [2, 1]]
```

- **Binary Encoding**

Binary encoding convert a category into a binary digits. Each binary digit creates one feature column. If there are n unique categories, then binary encoding results in only $log\ n$ features.

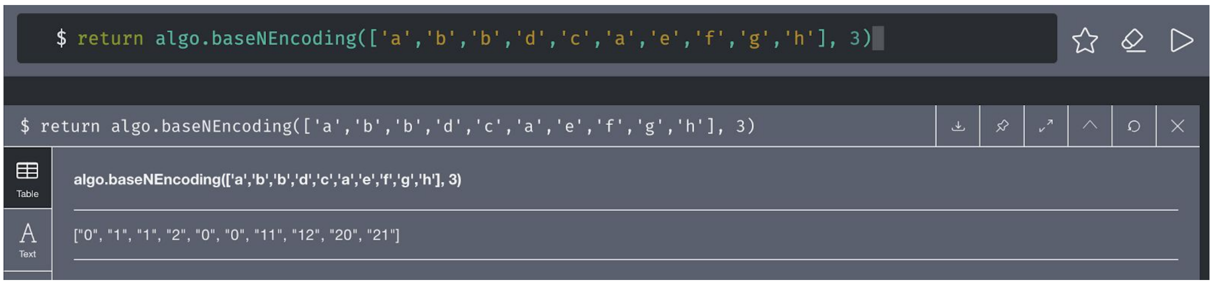


```
$ return algo.BinaryEncoding(['a','b','b','d','c','a'])
```

```
$ return algo.BinaryEncoding(['a','b','b','d','c','a'])
algo.BinaryEncoding(['a','b','b','d','c','a'])
["0", "1", "1", "10", "11", "0"]
```

- **BaseN Encoding**

BaseN Encoding is like Binary Encoding, but Binary Encoding is based on 2. Using BaseN Encoding, we can choose the base arbitrarily
In the following example, we choose 3 as the base.



```
$ return algo.baseNEncoding(['a','b','b','d','c','a','e','f','g','h'], 3)
```

```
$ return algo.baseNEncoding(['a','b','b','d','c','a','e','f','g','h'], 3)
algo.baseNEncoding(['a','b','b','d','c','a','e','f','g','h'], 3)
["0", "1", "1", "2", "0", "0", "11", "12", "20", "21"]
```

Conclusion and Future Work

What we have done:

1. Approximate Nearest Neighbors
2. Group Centralization
3. Reach Centrality
4. Encoding

Future work:

1. Extend more algorithms
2. Optimization:
    a. Use state-of-the-art algorithm
    b. Parallelism

Reference

1. Roussopoulos, N.; Kelley, S.; Vincent, F. D. R. (1995). "Nearest neighbor queries". Proceedings of the 1995 ACM SIGMOD international conference on Management of data – SIGMOD '95. p. 71. doi:10.1145/223784.223794. ISBN 0897917316.
2. Malkov, Yury; Yashunin, Dmitry (2016). "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs". arXiv:1603.09320
3. "New approximate nearest neighbor benchmarks"
4. "Approximate Nearest Neighbours for Recommender Systems"