



Prueba Practica Sistemas Expertos Basados en casos.

Nombre: Fabian Armijos

Objetivo:

- Consolidar los conocimientos adquiridos en clase de los sistemas expertos basados en casos.

Enunciado:

Se desea generar un sistema de recomendación de películas, por tal motivo se va a utilizar una base de datos orientada a grafos para lograr esto se describe los pasos a seguir:

- 1) Con estos datos aplicar el algoritmo de KNN y Similitud de Coseno para la recomendación de películas, seguir el siguiente tutorial: <https://www.markhneedham.com/blog/2018/09/28/neo4j-graph-algorithms-cosine-game-of-thrones/> o <https://vladbatushkov.medium.com/one-month-graph-challenge-flags-5d30aec366a0>.
- 2) Finalmente realizar alguna interfaz para poder acceder a la recomendación e ingreso de datos y resultados de los procesos en python.

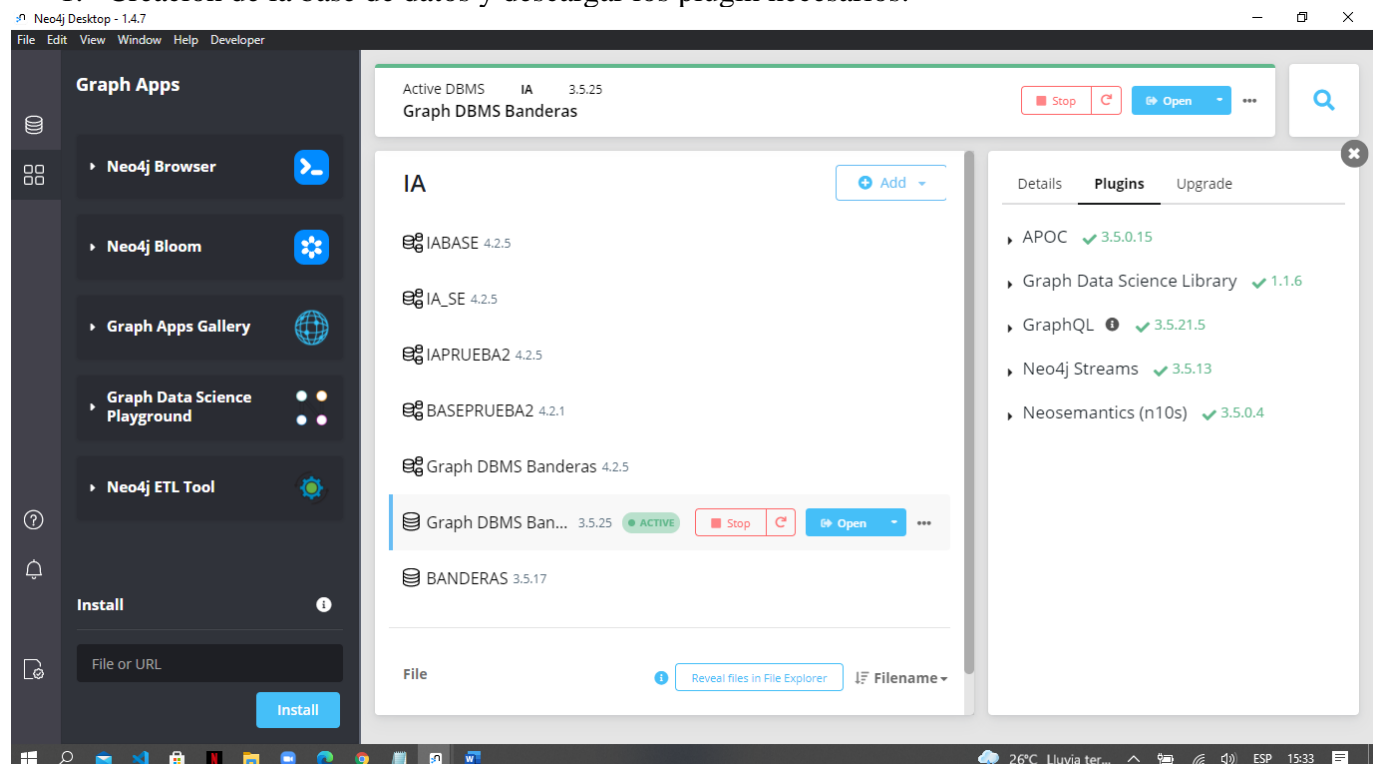
Generar el Informe en PDF y subir los scripts al repositorio Git para su evaluación.

Fecha de Entrega : **18/07/2021 - 23:55**

Criterios de Evaluación

- Neo4J Knn: 50%
- Informe y resultados: 20%
- GUI, programación y pruebas: 30%

1. Creación de la base de datos y descargar los plugin necesarios.





Sistemas Expertos

Tema: Basados casos

Prueba Practica Sistemas Expertos Basados en casos.

2. Ingresamos los datos a crear o unir en NEO4J de la siguiente manera y con match(n) return n; retornamos los valores ingresados.

The top screenshot shows the Neo4j Browser interface with the following Cypher query:

```
$
57 CREATE (f15:Flag { name: "Hungary" })
58 CREATE (f15)-[:CONTAINS { weight: 33 }]->(red)
59 CREATE (f15)-[:CONTAINS { weight: 33 }]->(white)
60 CREATE (f15)-[:CONTAINS { weight: 33 }]->(green)
61 CREATE (f16:Flag { name: "Romaina" })
62 CREATE (f16)-[:CONTAINS { weight: 33 }]->(red)
63 CREATE (f16)-[:CONTAINS { weight: 33 }]->(yellow)
64 CREATE (f16)-[:CONTAINS { weight: 33 }]->(blue)
65 CREATE (f17:Flag { name: "Austria" })
66 CREATE (f17)-[:CONTAINS { weight: 66 }]->(red)
67 CREATE (f17)-[:CONTAINS { weight: 33 }]->(white)
68 CREATE (f18:Flag { name: "Italy" })
```

The bottom screenshot shows the resulting graph visualization. The graph displays nodes for colors (Red, White, Green, Blue, Yellow) and flags (Hungary, Romaina, Austria, Italy). The relationships are labeled 'CONTAINS' with weights. The query executed is:

```
$ match (n) return n;
```

The graph shows a central 'Red' node connected to 'Hungary', 'Romaina', and 'Austria'. 'White' is connected to 'Hungary' and 'Austria'. 'Green' is connected to 'Hungary'. 'Blue' is connected to 'Romaina'. 'Yellow' is connected to 'Romaina'. 'Italy' is connected to 'Red'.

3. Como siguiente ingresamos las siguientes líneas para un pequeño filtrado y encontremos el color



Prueba Practica Sistemas Expertos Basados en casos.

más utilizado.

The top screenshot shows the Neo4j Browser interface with the following Cypher query:

```
$ MATCH (:Flag)-[c1:CONTAINS]→(:Color) WITH sum(c1.weight) as total MATCH (:Flag)-[c2:CONTAINS]→(cl:Color) W...
```

The result is displayed as a table with 6 records:

	colorName	percentage
1	"Red"	41
2	"White"	20
3	"Blue"	14
4	"Yellow"	11
5	"Green"	7
6	"Black"	4

The bottom screenshot shows the same interface with a graph visualization at the bottom. The graph shows a node labeled (112) connected to a node labeled CONTAINS(112). The graph also shows a node labeled (156) connected to a node labeled Color(12), which is connected to a node labeled Flag(44). The graph also shows a node labeled (112) connected to a node labeled CONTAINS(112).

- Y finalmente buscamos las banderas más similares de todos los países según el peso del color



Sistemas Expertos

Tema: Basados casos

Prueba Practica Sistemas Expertos Basados en casos.

neo4j@bolt://localhost:11003 - Neo4j Browser

```
$
```

```
1 MATCH (item:`Flag`), (category:`Color`)
2 OPTIONAL MATCH (item:`Flag`)-[rel:`CONTAINS`]->(category:`Color`)
3 WITH {item:id(item), weights: collect(coalesce(rel.`weight`, gds.util.NaN()))} as userData
4 WITH collect(userData) as data
5 WITH $config AS config, data
6 WITH config {.*, data: data} as config
7
8 ~ CALL gds.alpha.similarity.cosine.stream(config)
9
10 ~ YIELD item1, item2, similarity
11 RETURN gds.util.asNode(item1) AS from, gds.util.asNode(item2) AS to, similarity
12 ORDER BY similarity DESC
```

"from"	"to"	"similarity"
{"name": "Belarus"}	{"name": "Germany"}	1.0
{"name": "Belarus"}	{"name": "Estonia"}	1.0
{"name": "Russia"}	{"name": "Ukraine"}	1.0
{"name": "Russia"}	{"name": "Estonia"}	1.0
{"name": "Russia"}	{"name": "Sweden"}	1.0

Nota: Subir el sistema en un cuaderno de Python + scripts + PDF al Git Personal.