Trabajo Práctico Final

El Paradigma No SQL – Bases de Datos de Grafos

Alumna:

Vilamowski, Abril 62495

Profesores:

Gómez, Leticia Irene

Vaisman, Alejandro Ariel

11 de diciembre de 2024

Índice

Parte A: Carga de los datos

Para compilar el proyecro con Maven, se debe correr

```
mvn clean install
```

Se debe utilizar el siguiente comando para realizar la carga de datos, tanto para el archivo air-routes.graphml, como para los graphml de testeo, example-a.graphml y example-b.graphml. Estos últimos se encuentran en la carpeta resources dentro del proyecto.

```
spark-submit --master yarn --deploy-mode=cluster --class
org.itba.grafos.tpe.Main --packages com.tinkerpop.blueprints:blueprints-
core:2.6.0 --jars hdfs://node1/user/[username]/graphframes-0.8.0-spark2.4-
s_2.11.jar hdfs://node1/user/[username]/original-tpe-1.jar
hdfs:///user/[username]/[file_name].graphml
```

Parte B: Procesamiento de las consultas

Los códigos con las implementaciones realizadas con GraphFrames se encuentran en el proyecto Maven junto con el código que permite guardar los resultados en un archivo con el formato pedido.

Parte C: Verificación de correctitud

Para comparar los resultados obtenidos con GraphFrames y JanusGraph, se pueden utilizar diferentes criterios:

- Verificar si cada par de queries devuelve la misma cantidad de filas, utilizando la función count() en cada uno de los casos.
- En caso de que las consultas devuelvan el mismo formato, se puede realizar una comparación uno a uno entre los resultados utilizando un script o un comando (por ejemplo, diff). En el caso de consultas ejecutadas en JanusGraph, es necesario guardar los resultados en un archivo antes de proceder. Es importante considerar el formato de salida de cada consulta, ya que si los resultados no están en el mismo formato, se debe realizar un proceso de formateo adecuado antes de la comparación. Si los resultados están en el mismo formato, también se puede observar el peso de los archivos para verificar si hay diferencias significativas en el tamaño. Además, dado que los resultados pueden no estar ordenados de la misma manera, se deben ordenar previamente para garantizar una comparación precisa.
- Modificar o eliminar ciertas condiciones de las consultas y comparar si los resultados obtenidos siguen siendo los mismos. Por ejemplo, en el ejercicio b1, se podría cambiar el aeropuerto de destino 'SEA' por otro aeropuerto y comparar los resultados. También, se podría eliminar la condición que exige que los aeropuertos tengan latitud y longitud. En el ejercicio b2, se podría limitar la cantidad de continentes considerados para verificar si los resultados siguen siendo consistentes.

Consultas de la parte B realizadas con Janusgraph

Indicar para **aquellos aeropuertos** que tengan valores de latitud y longitud negativos, cuáles van al aeropuerto SEA (Seattle) usando a lo sumo una escala, y cuál es esa forma de llegar.

```
graph.traversal().V()
  .hasLabel('airport').has('lat', lt(0)).has('lon', lt(0))
  .union(
    __.as('src').outE('route').inV().hasLabel('airport')
      .has('code', 'SEA')
      .where(neq('src'))
      .where(values('code').is(neq(select('src').values('code'))))
      .project('origin', 'destination')
      .by(select('src').values('code'))
      .by(values('code')),
    __.as('src').outE('route').inV().hasLabel('airport')
      .as('stop')
      .where(neq('src'))
      .where(values('code').is(neq(select('src').values('code'))))
      .outE('route')
      .inV()
      .hasLabel('airport')
      .has('code', 'SEA')
      .where(neq('src')).where(neq('stop'))
      .where(values('code').is(neq(select('stop').values('code'))))
      .project('origin', 'stop', 'destination')
      .by(select('src').values('code'))
      .by(select('stop').values('code'))
      .by(values('code'))
  )
  .map {
    def path = it.get();
    path.containsKey('stop') ?
      path.origin + " -> " + path.stop + " -> " + path.destination :
      path.origin + " -> " + path.destination
  }
  .dedup()
```

Como Gremlin no permite ingresar queries multilínea, adjunto la query en una única línea

```
graph.traversal().V().hasLabel('airport').has('lat', lt(0)).has('lon',
lt(0)).union(__.as('src').outE('route').inV().hasLabel('airport').has('code',
'SEA').where(neq('src')).where(values('code').is(neq(select('src').values('code
'')))).project('origin',
'destination').by(select('src').values('code')).by(values('code')),
__.as('src').outE('route').inV().hasLabel('airport').as('stop').where(neq('src'
)).where(values('code').is(neq(select('src').values('code')))).outE('route').in
V().hasLabel('airport').has('code',
'SEA').where(neq('src')).where(neq('stop')).where(values('code').is(neq(select(
'stop').values('code')))).project('origin', 'stop',
'destination').by(select('src').values('code')).by(select('stop').values('code'
)).by(values('code'))).map { def path = it.get(); path.containsKey('stop') ?
path.origin + " -> " + path.stop + " -> " + path.destination : path.origin + "
-> " + path.destination }.dedup()
```

Listar por cada **continente y país**, la lista de valores de las elevaciones de sus aeropuertos. Debe aparecer una sola tupla por cada continente y país con la agrupación de los valores de las elevaciones registradas.

```
graph.traversal().V()
    .hasLabel('continent').as('continent')
    .outE('contains').inV()
    .hasLabel('airport').as('airport')
    .inE('contains').outV()
    .hasLabel('country').as('country')
    .group()
    .by(
        project('continent', 'country')
            .by(select('continent').values('desc'))
            .by(select('country').valueMap('code', 'desc'))
    )
    .by(select('airport').values('elev').order().fold())
    .unfold()
    .order().by(keys, Order.asc)
    .map {
        def key = it.get().key
        def continentDesc = key['continent']
        def countryCode = key['country']['code'][0]
        def countryDesc = key['country']['desc'][0]
        def elevations = it.get().value
        continentDesc + " " + countryCode + " (" + countryDesc + ") " +
elevations
    }
```

Como Gremlin no permite ingresar queries multilínea, adjunto la query en una única línea

```
graph.traversal().V().hasLabel('airport').has('lat', lt(0)).has('lon',
lt(0)).union(__.as('src').outE('route').inV().hasLabel('airport').has('code',
'SEA').where(neq('src')).where(values('code').is(neq(select('src').values('code
')))).project('origin',
'destination').by(select('src').values('code')).by(values('code')),
__.as('src').outE('route').inV().hasLabel('airport').as('stop').where(neq('src'
)).where(values('code').is(neq(select('src').values('code')))).outE('route').in
V().hasLabel('airport').has('code',
'SEA').where(neq('src')).where(neq('stop')).where(values('code').is(neq(select(
'stop').values('code')))).project('origin', 'stop',
'destination').by(select('src').values('code')).by(select('stop').values('code'
)).by(values('code'))).map { def path = it.get(); path.containsKey('stop') ?
path.origin + " -> " + path.stop + " -> " + path.destination : path.origin + "
-> " + path.destination }
```