



Paradigma NoSQL: Bases de Datos de  
Grafos

**Trabajo Práctico Especial**


Gomez, Lucas  
60408

# Introducción

El presente trabajo tiene como objetivo el procesamiento de estructuras geométricas en grafos cuyos vértices sean coordenadas del plano. Más precisamente, encontrar todos los cuadriláteros simples.

## Configuración de Spark

Inicialmente se creó la configuración de Spark con el objetivo de interactuar con el cluster de Spark

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Java and configures a Spark session.

```
SparkConf sparkConf = new SparkConf().setAppName("Geometrias");  
JavaSparkContext sparkContext = new JavaSparkContext(sparkConf);  
SparkSession sparkSession = SparkSession.builder()  
    .sparkContext(sparkContext.sc())  
    .getOrCreate();
```

Se creó un *JavaSparkContext* y a partir de este, se crea una *SparkSession* la cual se es utilizada para trabajar con dataframes en Spark.

## Carga de datos

El archivo de entrada se encuentra en formato *.csv*, el cual contiene información sobre las aristas del grafo. Las columnas corresponden al archivo son *x1*, *y1*, *x2*, *y2*. Donde las primeras dos columnas representan las coordenadas del origen de una arista y las últimas dos columnas representan las coordenadas del punto destino.

La información del archivo de entrada se almacena en un *Dataset* y se procede a filtrar valores inválidos, tales como bucles en un mismo vértice y valores negativos

```

Dataset<Row> dataset = sparkSession.read().option("header", "true").csv(inputPath);

dataset = dataset.select(
    dataset.col("x1").cast("int"),
    dataset.col("y1").cast("int"),
    dataset.col("x2").cast("int"),
    dataset.col("y2").cast("int")
).dropDuplicates();

// Remove self-loops
dataset = dataset.filter(
    dataset.col("x1").notEqual(dataset.col("x2"))
    .or(dataset.col("y1").notEqual(dataset.col("y2")))
);

//Drop negative coords
dataset = dataset.filter(
    dataset.col("x1").geq(0).and(dataset.col("y1").geq(0))
    .and(dataset.col("x2").geq(0)).and(dataset.col("y2").geq(0))
);

```

## Asignación de identificadores a los vértices

Una vez filtrados los valores inválidos, se procede a agregar un identificador a cada uno de los vértices, para esto, se utilizó la función de **Cantor pairing**, la cual se encarga de asignar un identificador a partir de los valores (x, y) correspondientes a sus coordenadas.

$$\pi(k_1, k_2) = \frac{1}{2}(k_1 + k_2)(k_1 + k_2 + 1) + k_2$$

```

// Cantor pairing
dataset = dataset.withColumn("id1", cantorPairing(dataset.col("x1"), dataset.col("y1")))
    .withColumn("id2", cantorPairing(dataset.col("x2"), dataset.col("y2")));
dataset.show();

```

```

private static Column cantorPairing(Column x, Column y) {
    // Formula modificada: (x^2 + x + 2xy + 3y + y^2) / 2
    return expr("((( " + x + " + " + y + " ) / 2) * (( " + x + " + " + y + " ) + 1)) + " + y);
}

```

## Creación de vértices

Una vez identificados los vértices, se procede a crear un *Dataset* donde se incluyen los vértices origen y destino de cada arista.

```
Dataset<Row> vertices1 = dataset.select(
    dataset.col("id1").alias("id"),
    dataset.col("x1").alias("x"),
    dataset.col("y1").alias("y")
);
Dataset<Row> vertices2 = dataset.select(
    dataset.col("id2").alias("id"),
    dataset.col("x2").alias("x"),
    dataset.col("y2").alias("y")
);
Dataset<Row> vertices = vertices1.union(vertices2).distinct();
vertices.show();
```

## Creación de aristas

Análogamente al paso anterior, se crean las aristas que se encuentran definidas en el archivo de entrada.

```
Dataset<Row> edges = dataset.selectExpr("id1 as src", "id2 as dst");
Dataset<Row> reversedEdges = edges.selectExpr("dst as src", "src as dst");
edges = edges.union(reversedEdges).dropDuplicates();
edges.show();
```

Es importante notar que *GraphFrame* define grafos dirigidos, por lo que es necesario definir las aristas en ambos sentidos.

## Creación de grafo

Con los datasets previamente definidos, se procede a instanciar un nuevo grafo utilizando *GraphFrame*.



```
GraphFrame graphFrame = new GraphFrame(vertices, edges);
```

## Búsqueda de cuadriláteros simples

Se implementó la función *findQuads*, encargada de realizar todo el procesamiento y filtrado de los cuadriláteros simples que se encontraban definidos en el archivo de entrada.

```
private static Dataset<Row> findQuads(GraphFrame graphFrame) {  
    Dataset<Row> quads = graphFrame.find("(A)-[ab]->(B); (B)-[bc]->(C); (C)-[cd]->(D); (D)-[da]->(A)")  
        .filter("A != B AND A != C AND A != D AND B != C AND B != D AND C != D")  
        .filter(row -> {  
            Row A = row.getAs("A");  
            Row B = row.getAs("B");  
            Row C = row.getAs("C");  
            Row D = row.getAs("D");  
  
            return !segmentsIntersect(A, B, C, D);  
        })  
        .filter(row -> {  
            Row A = row.getAs("A");  
            Row B = row.getAs("B");  
            Row C = row.getAs("C");  
            Row D = row.getAs("D");  
  
            return antiClockwise(A,B,C,D);  
        })  
        .withColumn("sorted_ids", sort_array(array(col("A.id"), col("B.id"), col("C.id"),  
col("D.id"))))  
        .orderBy("A.id")  
        .dropDuplicates("sorted_ids")  
        .select(  
            col("A.id").alias("id1"),  
            col("A.x").alias("x1"),  
            col("A.y").alias("y1"),  
            col("B.id").alias("id2"),  
            col("B.x").alias("x2"),  
            col("B.y").alias("y2"),  
            col("C.id").alias("id3"),  
            col("C.x").alias("x3"),  
            col("C.y").alias("y3"),  
            col("D.id").alias("id4"),  
            col("D.x").alias("x4"),  
            col("D.y").alias("y4"),  
            col("A.id").alias("id5"),  
            col("A.x").alias("x5"),  
            col("A.y").alias("y5")  
        );  
    return quads.orderBy("id1");  
}
```

En las siguientes secciones se procederá a describir cada uno de los pasos realizados

## Búsqueda de aristas conectadas

Mediante un simple patrón se encontraron todos subgrafos de 4 aristas conectadas

```
graphFrame.find("(A)-[ab]->(B); (B)-[bc]->(C); (C)-[cd]->(D); (D)-[da]->(A)")
```

## Evitar vértices repetidos

Para evitar que el recorrido de las cuatro aristas conectadas vuelva a pasar por un nodo ya visitado, se filtraron solo las que poseían todos sus vértices diferentes entre sí.

```
.filter("A != B AND A != C AND A != D AND B != C AND B != D AND C != D")
```

## Eliminar intersecciones entre aristas

Para asegurar que los cuadriláteros encontrados son simples, se validó que los segmentos opuestos entre sí (es decir,  $\overline{AB}$  y  $\overline{CD}$ ;  $\overline{BC}$  y  $\overline{DA}$ ) no se intersectan.

Con este objetivo, se creó un filtro donde se aplica la función *segmentsIntersect* que valida la condición previamente descrita.

```
.filter(row -> {  
  Row A = row.getAs("A");  
  Row B = row.getAs("B");  
  Row C = row.getAs("C");  
  Row D = row.getAs("D");  
  
  return !segmentsIntersect(A, B, C, D) && !segmentsIntersect(B, C, D, A);  
})
```

```

private static boolean segmentsIntersect(Row A, Row B, Row C, Row D) {
    int o1 = orientation(A, B, C);
    int o2 = orientation(A, B, D);
    int o3 = orientation(C, D, A);
    int o4 = orientation(C, D, B);

    return o1 != o2 && o3 != o4;
}

```

## Algoritmo de intersección

La función *orientation* es utilizada para determinar la orientación de tres vértices. La cual puede tener tres resultados posibles

- Colineales: Los puntos están en la misma línea y el resultado es 0.
- Sentido horario: Los puntos giran en sentido horario y el resultado es mayor a 0.
- Sentido antihorario: Los puntos giran en sentido antihorario y el resultado es menor a 0.

La fórmula utilizada para determinar esto es:

$$val = (q_y - p_y)(r_x - q_x) - (q_x - p_x)(r_y - q_y)$$

Donde:

- $(p_x, p_y)$  son las coordenadas del punto p.
- $(q_x, q_y)$  son las coordenadas del punto q.
- $(r_x, r_y)$  son las coordenadas del punto r.

Para que los segmentos  $\overline{AB}$  y  $\overline{CD}$  se intersectan se debe cumplir que:

- Las orientaciones de A,B,C y las de A,B,D son diferentes
- Las orientaciones de C,D,A y las de C,D,B son diferentes

Esto significa que los puntos C y D están en lados opuestos de la línea formada por A y B; o equivalentemente, los puntos A y B están en lados opuestos de la línea formada por C y D.

Finalmente, la función *orientation* se define de la siguiente manera:

```

public static int orientation(Row p, Row q, Row r) {
    int px = p.getAs("x");
    int py = p.getAs("y");
    int qx = q.getAs("x");
    int qy = q.getAs("y");
    int rx = r.getAs("x");
    int ry = r.getAs("y");

    double val = (qy - py) * (rx - qx) - (qx - px) * (ry - qy);
    if (val == 0) {
        return 0;
    }
    return (val > 0) ? 1 : -1;
}

```

## Ordenar vértices de forma anti-horaria

Haciendo uso de la función de orientación previamente definida, se agregó un nuevo filtro a los cuadriláteros:

```

.filter(row -> {
    Row A = row.getAs("A");
    Row B = row.getAs("B");
    Row C = row.getAs("C");
    Row D = row.getAs("D");

    return antiClockwise(A,B,C,D);
})

```

La función *antiClockwise* se define como:



```
private static boolean antiClockwise(Row A, Row B, Row C, Row D) {  
    return orientation(A, B, C) == -1 &&  
        orientation(B, C, D) == -1 &&  
        orientation(C, D, A) == -1;  
}
```

## Ordenar por el menor ID del primer nodo

Finalmente, para eliminar los cuadriláteros que se encuentran duplicados, pero listados comenzando por distintos nodos, se ordenaron las columnas para que el nodo A sea el de menor ID. Al estar ordenados, todos los cuadriláteros equivalentes tendrán los mismos valores en cada columna del dataset, por lo que se procede a eliminar los duplicados:

```
.withColumn("sorted_ids", sort_array(array(col("A.id"), col("B.id"), col("C.id"), col("D.id"))))  
.dropDuplicates("sorted_ids")
```

## Obtener los valores de salida

A fines de obtener el archivo de salida como se muestra en la consigna, se obtienen todos los valores necesarios para armar el archivo final.

```
.select(
    col("A.id").alias("id1"),
    col("A.x").alias("x1"),
    col("A.y").alias("y1"),
    col("B.id").alias("id2"),
    col("B.x").alias("x2"),
    col("B.y").alias("y2"),
    col("C.id").alias("id3"),
    col("C.x").alias("x3"),
    col("C.y").alias("y3"),
    col("D.id").alias("id4"),
    col("D.x").alias("x4"),
    col("D.y").alias("y4"),
    col("A.id").alias("id5"),
    col("A.x").alias("x5"),
    col("A.y").alias("y5")
);
```

## Almacenamiento de resultados

Por último, se guardan los resultados en un archivo .csv

```
private static String saveResults(Dataset<Row> results, String inputPath) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd_HH:mm:ss");
    String timestamp = sdf.format(new Date());

    String directory = new File(inputPath).getParent() + "/" + timestamp;

    results.write()
        .option("header", "true")
        .csv(directory);
    return directory;
}
```

# Anexo

## Uso de dependencias

### Spark

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.12</artifactId>
  <version>3.0.0</version>
  <scope>provided</scope>
</dependency>
```

### GraphFrames

```
<dependency>
  <groupId>graphframes</groupId>
  <artifactId>graphframes</artifactId>
  <version>0.8.0-spark2.4-s_2.11</version>
</dependency>
```

Salida de archivos de ejemplo

data\_1.csv:

id1	x1	y1	id2	x2	y2	id3	x3	y3	id4	x4	y4	id5
0.0	0	0	11425.0	50	100	31575.0	50	200	1325.0	0	50	0.0
0.0	0	0	5050.0	100	0	20200.0	100	100	5150.0	0	100	0.0
4.0	1	1	7.0	2	1	12.0	2	2	8.0	1	2	4.0
55.0	10	0	465.0	30	0	1295.0	30	20	485.0	10	20	55.0
210.0	20	0	820.0	40	0	3280.0	40	40	1870.0	20	40	210.0
475.0	20	10	830.0	30	10	1565.0	30	25	840.0	20	20	475.0

data\_2.csv:

id1	x1	y1	id2	x2	y2	id3	x3	y3	id4	x4	y4	id5
0.0	0	0	20100.0	200	0	80400.0	200	200	20300.0	0	200	0.0
0.0	0	0	11425.0	50	100	31575.0	50	200	1325.0	0	50	0.0
0.0	0	0	5050.0	100	0	20200.0	100	100	5150.0	0	100	0.0
210.0	20	0	820.0	40	0	3280.0	40	40	1870.0	20	40	210.0

## Bibliografía

<https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/>