In [171]:

```
# Définir une fonction qui prend comme argument une liste d'utilisateurs triés
# par ordre croissant (users) et retourne une liste de couples ordonnés d'utilis
ateurs

from pyspark.sql.functions import udf
from pyspark.sql.types import *

def parse_string(users):
    results = []
    i = 0
    for i in range(len(users)-1):
        for j in range(i+1, len(users)):
            results.append([users[i],users[j]])
    print(results)
    return results

parse_string_udf = udf(parse_string, ArrayType(StringType()))
```

In [191]:

```
# Implémentation de l'algorithme de calcul de triangles

from pyspark.sql.functions import collect_list, sort_array

#Map1 - cours
#Prendre en considération uniquement les couples ordonnés (trackId, track1) (tra
ckId < track1)
trackOrd = trackTrack.filter(col("trackId") < col("track1")).drop(col("norm_coun
t"))

#Reduce 1 - cours: Construire pour chaque trackId la liste de couples ordonnés d
e ses voisins
# a) regrouper les lignes par trackId en construisant la liste de voisins triés
 par ordre
#    croissant (utiliser sort_array)
neighbors = trackOrd.groupBy("trackId") \
           .agg(collect_list("track1").alias("neighbors"))
neighbors = neighbors.withColumn("sorted_neigh", sort_array(neighbors["neighbor
s"]))\
           .select(col("trackId"), col("sorted_neigh"))


# b) utiliser la fonction définie précédemment pour retourner la liste de couple
s de voisins
couples=neighbors.select(col("trackId"), parse_string_udf("sorted_neigh").alias(
"neigh_couples"))\
      .withColumn("neigh_couple", explode('neigh_couples')).drop(col("neigh_coup
les"))

# Map2 + Reduce 2 - cours
# prendre en considération uniquement les lignes telles que les couples de voisi
ns
# construits précédemment existent également dans le graphe
from pyspark.sql.functions import concat, lit, count, desc
liste = trackOrd.withColumn("couple", concat(lit("["), col("trackId"), lit(", "
), col("track1"), lit("]")))\
        .drop(col("trackId")).drop("track1")

# Calculer le nombre de triangles pour chaque utilisateur et
# trier le résultat par le nombre de triangles décroissant
triangles = couples.join(liste, col("neigh_couple")==col("couple")).drop(col("ne
igh_couple"))\
            .groupBy(col("trackid")).agg(count(col("couple")).alias("nb_triangle
s"))\
            .orderBy(desc("nb_triangles"))

triangles.show()
```

```
+-------+------------+
|trackid|nb_triangles|
+-------+------------+
| 808082|          62|
| 808082|          50|
| 825174|          48|
| 800288|          47|
| 831005|          47|
| 815388|          47|
| 800288|          44|
| 800288|          44|
| 800288|          42|
| 825174|          41|
| 806854|          40|
| 806854|          39|
| 825174|          39|
| 806854|          38|
| 825174|          38|
| 843219|          37|
| 817399|          37|
| 813969|          37|
| 810775|          37|
| 811513|          37|
+-------+------------+
only showing top 20 rows
```