**Distributed Computing with Big Data Counting Triangles**

**Introduction**

This report shows the implementation of a distributed algorithm in PySpark for counting triangles. Computation of the total number of triangles in the graph and calculating the average number of triangles per node is done. The outcome of this analysis offers valuable insights into the graph's community structure and cohesion. The report provides an overview of the analysis and also briefly addresses challenges encountered during implementation and concludes with a summary of findings and potential future directions.

**Challenges**

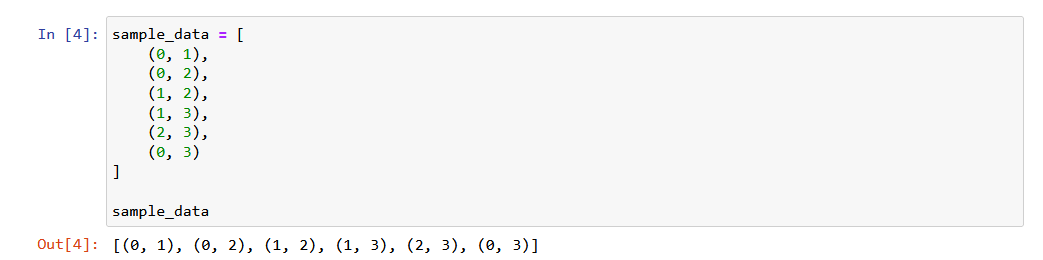
Some challenges were encountered during the analysis. One of the main difficulties involved ensuring the accuracy of the triangle-counting algorithm, as even small discrepancies in results could affect the overall analysis. Interpreting the results required an understanding of the graph's structure which could sometimes be intricate in the data.

**Enjoyable Aspects of the Analysis**

One of the most satisfying aspects was uncovering hidden patterns within the network in the datasets. Observing how the nodes connected and formed triangles showed the graph's relationships and structure. Interpreting these patterns was a rewarding part of the analysis. It offered a sense of accomplishment and a deeper understanding of the network's behaviour.

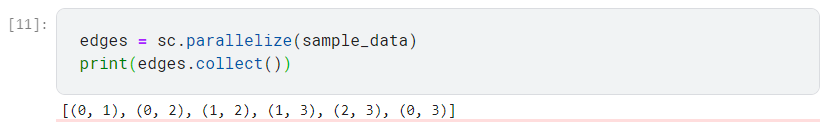
**Loading the data**

The sample data was loaded into the Spark environment, which represents an undirected graph. This data consisted of edge pairs denoting connections between nodes.



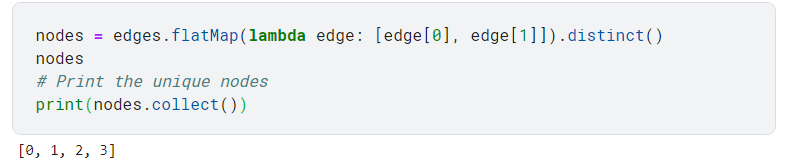
**Create an RDD from the data**

An RDD is created from the sample data each line representing an edge.



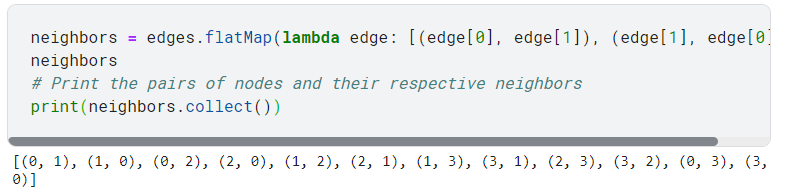
**Create an RDD of Unique Nodes**

An RDD containing the unique nodes present in the dataset is generated in order to facilitate graph analysis. This will help identify the nodes involved in triangles and calculate the average number of triangles per node.



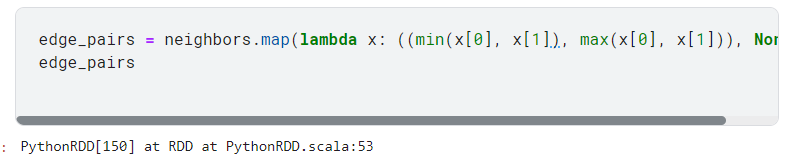
**Generate an RDD containing pairs of nodes and their respective neighbours**

For triangle counting, an RDD was created to pair each node with its neighbouring nodes. This process involved applying a flat map transformation to the edges RDD resulting in pairs representing nodes and their neighbours.



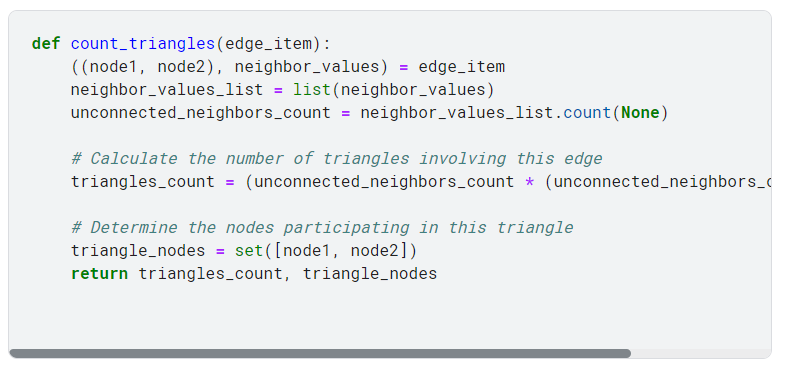
**Generate an RDD containing pairs ((u, v), w) with 'u' less than 'v'**

As per the triangle counting algorithm's requirements, an RDD was constructed to hold pairs in the format ((u, v), w), where u and v represent nodes while w is the placeholder value. U should always be less than V condition is used to ensure the uniqueness of these pairs and prevent redundancy.



**Define a function to count triangles and collect nodes**

A custom function plays a pivotal role in both counting triangles and collecting the nodes involved. It takes grouped edge pairs as input and performs distinct tasks. First it extracts the potential edge (u, v) and placeholder values from the grouped pairs. It then computes the number of triangles encompassing these edges by assessing the count of values serving as indicators of common neighbours.

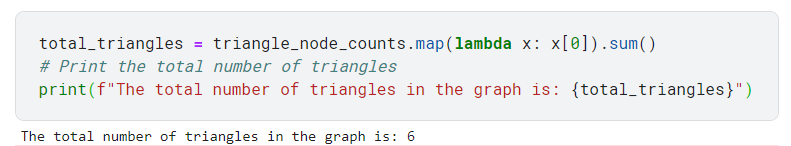


**Calculate triangles and collect nodes for each edge**

The process of identifying triangles and collecting nodes within the graph was initiated by applying the custom count\_triangles function to each edge within the grouped\_edges RDD.



**Calculate the total number of triangles**



**Calculate the average nodes involved in triangles for each node**

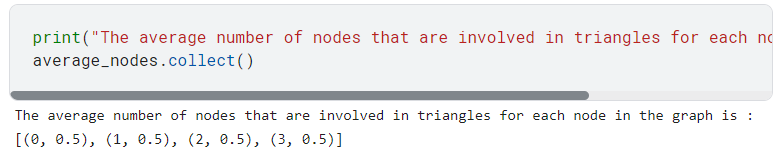


**Print the total number of triangles**



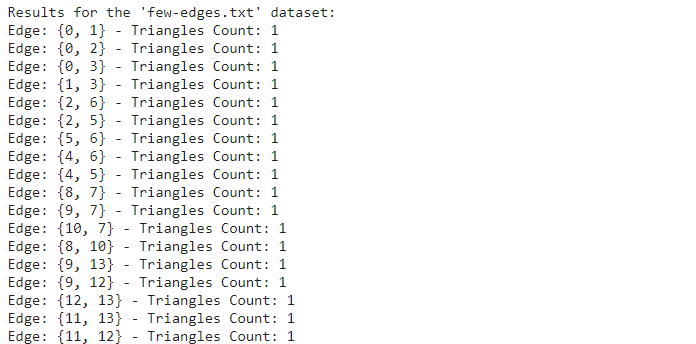
**Average Nodes Involved in Triangles**

This step involves calculating and printing the average number of nodes that each node shares a triangle within the graph.



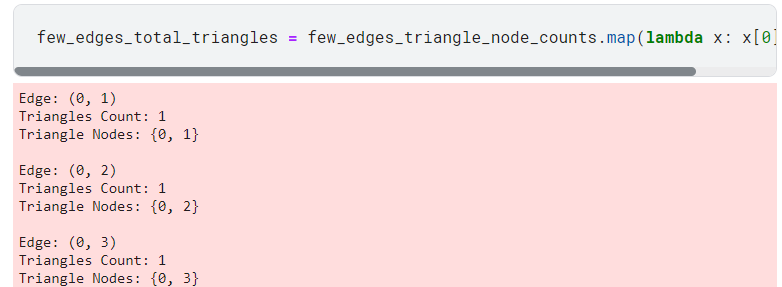
**Test in a Sample Input File (few-edges.txt)**

To validate the triangle counting algorithm and ensure its scalability to larger datasets, a test using the sample input file few-edges.txt is conducted.

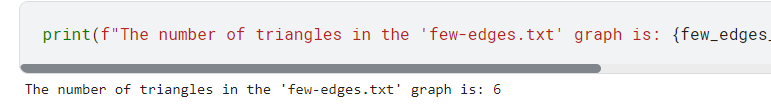


The analysis of the few-edges dataset showed insights into the graph's structure. Each edge within the dataset was examined for its participation in triangles. Every edge shared a connection with at least one triangle, as indicated by a triangle count of 1. This suggests a relatively interconnected graph, with triangles forming around most edges. The associated nodes for each triangle were identified. This provides valuable initial observations about the graph's connectivity.

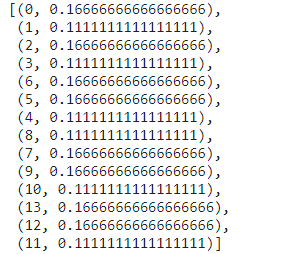
**Calculate the total number of triangles**



**Print the total number of triangles**



**Print the average nodes involved in triangles for each node**



The average nodes involved in triangles for each node were calculated. The results revealed varying degrees of node participation in triangular structures. Nodes 0, 2, 6, 5, 7, 9, 13 and 12 showed relatively higher averages indicating their central roles in connecting with other nodes through triangles. Conversely, nodes 1, 3, 4, 8, 10, and 11 exhibited lower averages, suggesting their comparatively peripheral positions within the network.

**Conclusion**

The analysis of the graph revealed a highly interconnected network where every edge participated in at least one triangle. The calculation of average nodes involved in triangles showed variations in node centrality, with some nodes acting as central connectors while others assume more peripheral roles. These findings provided valuable insights into the graph's structure, suggesting the presence of hubs and less-connected nodes.

**References**

Arjun, Subramanyam, Varalakshmi., Chong, Wang., Christoph, F., Eick. (2019). Fast proximity graph generation with spark. Doi: 10.1145/3356999.3365465

Newman, M. E. J. (2020). Networks: An Introduction. Oxford University Press.

Wang, D., Cui, P., Zhu, W., & Zhu, J. (2020). Structural deep network embedding. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1225-1234).