In [1]:

```
import warnings
warnings.simplefilter("ignore")

import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

spark = SparkSession.builder.appName("HW Session 3").getOrCreate()
```

```
21/11/01 17:30:13 WARN Utils: Your hostname, yousri-Lenovo-Legion-5-15
IMH05H resolves to a loopback address: 127.0.1.1; using 192.168.1.105
instead (on interface wlp0s20f3)
21/11/01 17:30:13 WARN Utils: Set SPARK_LOCAL_IP if you need to bind t
o another address
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform
(file:/opt/spark/jars/spark-unsafe_2.12-3.0.1.jar) to constructor jav
a.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apac
he.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illeg
al reflective access operations
WARNING: All illegal access operations will be denied in a future rele
ase
21/11/01 17:30:23 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicab
le
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.p
roperties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
```

- To open port : nc -l -p portnumber
- To open Localhost : nc localhost portnumber

Task 1

Streaming from TCP Socket

In [3]:

```
df = spark.readStream.format("socket") \
    .option("host", "localhost") \
    .option("port", 12345) \
    .load()
```

```
21/11/01 16:13:43 WARN TextSocketSourceProvider: The socket source sho
uld not be used for production applications! It does not support recov
ery.
```

Print Schema

In [4]:

```
df.printSchema()
```

```
root
 |-- value: string (nullable = true)
```

After processing, you can write the DataFrame to console.

In [9]:

```
query = df.writeStream.outputMode("append") \
    .format("console") \
    .start()

query.awaitTermination()
```

```
21/11/01 16:18:56 WARN StreamingQueryManager: Temporary checkpoint l
ocation created which is deleted normally when the query didn't fai
l: /tmp/temporary-0d1c23ba-4e6e-4fc6-987c-1bb2bbdd273f. If it's requ
ired to delete it under any circumstances, please set spark.sql.stre
aming.forceDeleteTempCheckpointLocation to true. Important to know d
eleting temp checkpoint folder is best effort.

-------------------------------------------
Batch: 0
-------------------------------------------
+-----+
|value|
+-----+
+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+-----+
|value|
```

Task 2

let's create a streaming DataFrame that represents text data received from a server listening on localhost:9999, and transform the DataFrame to calculate word counts.

Create DataFrame representing the stream of input lines from connection to localhost:9999

In [2]:

```
lines = spark.readStream.format('socket')\
        .option("host",'localhost')\
        .option('port', 9999)\
        .load()
```

```
21/11/01 16:30:54 WARN TextSocketSourceProvider: The socket source sho
uld not be used for production applications! It does not support recov
ery.
```

Split the lines into words

In [3]:

```
words = lines.select(explode(split(lines.value," ")).alias('word'))
```

Generate running word count

In [4]:

```
wordCounts = words.groupBy("word").count()
```

Start running the query that prints the running counts to the console

In [6]:

```
checkpointDir = 'chkpnt'
streamingQuery = (wordCounts.writeStream
                  .format('console')
                  .outputMode('complete')
                  .trigger(processingTime='1 second')
                  .option('checkpointLocation', checkpointDir)
                  .start())
streamingQuery.awaitTermination()
```

```
21/11/01 16:31:37 WARN ProcessingTimeExecutor: Current batch is fallin
g behind. The trigger interval is 1000 milliseconds, but spent 4001 mi
lliseconds

-------------------------------------------
Batch: 0
-------------------------------------------
+----+-----+
|word|count|
+----+-----+
+----+-----+


21/11/01 16:31:52 WARN ProcessingTimeExecutor: Current batch is fallin
g behind. The trigger interval is 1000 milliseconds, but spent 1887 mi
lliseconds

-------------------------------------------
Batch: 1
-------------------------------------------
+---------+-----+
|     word|count|
+---------+-----+
|    write|    1|
|      Try|    1|
|something|    1|
|       to|    1|
+---------+-----+


21/11/01 16:33:11 WARN ProcessingTimeExecutor: Current batch is fallin
g behind. The trigger interval is 1000 milliseconds, but spent 1743 mi
lliseconds

-------------------------------------------
Batch: 2
-------------------------------------------
+---------+-----+
|     word|count|
+---------+-----+
|    again|    1|
|    write|    2|
|      Try|    1|
|something|    2|
|      try|    1|
|       to|    2|
+---------+-----+


21/11/01 16:34:20 WARN ProcessingTimeExecutor: Current batch is fall
ing behind. The trigger interval is 1000 milliseconds, but spent 186
```

```
4 milliseconds

-------------------------------------------
Batch: 3
-------------------------------------------
+------------+-----+
|        word|count|
+------------+-----+
|query.stop()|    1|
|       again|    1|
|       write|    2|
|         Try|    1|
|   something|    2|
|         try|    1|
|          to|    2|
+------------+-----+


---------------------------------------------------------------------------
-------
KeyboardInterrupt                         Traceback (most recent cal
l last)
/tmp/ipykernel_47821/575588763.py in <module>
      6                     .option('checkpointLocation', checkpointDi
r)
      7                     .start())
----> 8 streamingQuery.awaitTermination()

/opt/spark/python/pyspark/sql/streaming.py in awaitTermination(self,
timeout)
    101             return self._jsq.awaitTermination(int(timeout *
1000))
    102         else:
--> 103             return self._jsq.awaitTermination()
    104
    105     @property

/opt/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py in __
call__(self, *args)
   1301             proto.END_COMMAND_PART
   1302
-> 1303         answer = self.gateway_client.send_command(command)
   1304         return_value = get_return_value(
   1305             answer, self.gateway_client, self.target_id, sel
f.name)

/opt/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py in se
nd_command(self, command, retry, binary)
   1031         connection = self._get_connection()
   1032         try:
-> 1033             response = connection.send_command(command)
   1034             if binary:
   1035                 return response, self._create_connection_gua
rd(connection)

/opt/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py in se
nd_command(self, command)
   1198
   1199         try:
-> 1200             answer = smart_decode(self.stream.readline()[:-1
])
```

```
   1201             logger.debug("Answer received: {0}".format(answe
r))
   1202             if answer.startswith(proto.RETURN_MESSAGE):

/usr/lib/python3.8/socket.py in readinto(self, b)
    667         while True:
    668             try:
--> 669                 return self._sock.recv_into(b)
    670             except timeout:
    671                 self._timeout_occurred = True

KeyboardInterrupt:
```

In [7]:

```
streamingQuery.stop()
```

Read csv file "test1.csv"

In [39]:

```
from pyspark.sql.types import (StructType, StructField,
                               StringType, IntegerType)

recordSchema = StructType([StructField('Name', StringType(), True),
                           StructField('Departments', StringType(), True),
                           StructField('Salary', IntegerType(), True)])
```

In [40]:

```
df = spark.readStream.format("csv") \
    .schema(df_schema) \
    .load("./")
```

In [41]:

```
df.printSchema()
```

```
root
 |-- Name: string (nullable = true)
 |-- Departments: string (nullable = true)
 |-- Salary: integer (nullable = true)
```

Writing Spark Streaming to Console

In [51]:

```
query = df.writeStream.outputMode("append") \
    .format("console") \
    .option("truncate", False) \
    .option("numRows", 10) \
    .start()
query.awaitTermination()
```

```
-------------------------------------------
Batch: 0
-------------------------------------------
+----------------------------+-----------+------+
|Name                        |Departments|Salary|
+----------------------------+-----------+------+
|{                           |null       |null  |
| "cells": [                 |null       |null  |
|  {                         |null       |null  |
|   "cell_type": "code"      |null       |null  |
|   "execution_count": 1     |null       |null  |
|   "metadata": {}           |null       |null  |
|   "outputs": [             |null       |null  |
|    {                       |null       |null  |
|     "name": "stderr"       |null       |null  |
|     "output_type": "stream"|null       |null  |
+----------------------------+-----------+------+
only showing top 10 rows
```

Task 3

Creat GraphFrames

Users can create GraphFrames from vertex and edge DataFrames.

Vertex DataFrame: A vertex DataFrame should contain a special column named "id" which specifies unique IDs for each vertex in the graph. Edge DataFrame: An edge DataFrame should contain two special columns: "src" (source vertex ID of edge) and "dst" (destination vertex ID of edge). Both DataFrames can have arbitrary other columns. Those columns can represent vertex and edge attributes.

id	name	age
a	Alice	34
b	Bob	36
c	Charlie	30
d	David	29
e	Esther	32
f	Fanny	36
g	Gabby	60

src	dst	relationship
a	b	friend
b	c	follow
c	b	follow
f	c	follow
e	f	follow
e	d	friend
d	a	friend
a	e	friend

In [2]:

```
# Create a Vertex DataFrame with unique ID column "id"
vert = spark.createDataFrame([('a','Alice',34),
                              ('b','Bob',36),
                              ('c','Charlie',30),
                              ('d','David',29),
                              ('e','Esther',32),
                              ('f','Fanny',36),
                              ('g','Gabby',30)],
                             ["id", "name", "age"])
vert.show()
```

```
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  a|  Alice| 34|
|  b|    Bob| 36|
|  c|Charlie| 30|
|  d|  David| 29|
|  e| Esther| 32|
|  f|  Fanny| 36|
|  g|  Gabby| 30|
+---+-------+---+
```

In [3]:

```
Edg = spark.createDataFrame([('a','b','friend'),
                             ('b','c','follow'),
                             ('c','b','follow'),
                             ('f','c','follow'),
                             ('e','f','follow'),
                             ('e','d','friend'),
                             ('d','a','friend'),
                             ('a','e','friend')
                             ],
                            ["src", "dst", "relationship"])
Edg.show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  a|  b|      friend|
|  b|  c|      follow|
|  c|  b|      follow|
|  f|  c|      follow|
|  e|  f|      follow|
|  e|  d|      friend|
|  d|  a|      friend|
|  a|  e|      friend|
+---+---+------------+
```

Create a graph from these vertices and these edges:

Command: cp filename //opt/spark/jars

In [4]:

```
from graphframes import*

gf = GraphFrame(vert,Edg)
```

Display vertices

In [5]:

```
gf.vertices
```

Out[5]:

```
DataFrame[id: string, name: string, age: bigint]
```

In [6]:

```
gf.vertices.show()
```

```
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  a|  Alice| 34|
|  b|    Bob| 36|
|  c|Charlie| 30|
|  d|  David| 29|
|  e| Esther| 32|
|  f|  Fanny| 36|
|  g|  Gabby| 30|
+---+-------+---+
```

Display edges

In [7]:

```
gf.edges.show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  a|  b|      friend|
|  b|  c|      follow|
|  c|  b|      follow|
|  f|  c|      follow|
|  e|  f|      follow|
|  e|  d|      friend|
|  d|  a|      friend|
|  a|  e|      friend|
+---+---+------------+
```

Display inDegrees

In [8]:

```
gf.inDegrees.show()
```

```
+---+--------+
| id|inDegree|
+---+--------+
|  f|       1|
|  e|       1|
|  d|       1|
|  c|       2|
|  b|       2|
|  a|       1|
+---+--------+
```

Display the outgoing degree of the vertices:

In [9]:

```
gf.outDegrees.show()
```

```
+---+---------+
| id|outDegree|
+---+---------+
|  f|        1|
|  e|        2|
|  d|        1|
|  c|        1|
|  b|        1|
|  a|        2|
+---+---------+
```

Display the degree of the vertices:

In [10]:

```
gf.degrees.show()
```

```
+---+------+
| id|degree|
+---+------+
|  f|     2|
|  e|     3|
|  d|     2|
|  c|     3|
|  b|     3|
|  a|     3|
+---+------+
```

Find the age of the youngest person in the graph

In [11]:

```
gf.vertices.select(min('age')).show()
```

```
+--------+
|min(age)|
+--------+
|      29|
+--------+
```

Count the number of 'follow' relationships in the graph:

In [13]:

```
gf.edges.filter('relationship = "follow"').count()
```

Out[13]:

4

Motif finding

Search for pairs of vertices with edges in both directions between them.

In [14]:

```
# Search for pairs of vertices with edges in both directions between them.
motifs = gf.find("(a)-[e]->(b); (b)-[e2]->(a)")
motifs.show()
```

```
+----------------+--------------+----------------+--------------+
|               a|             e|               b|            e2|
+----------------+--------------+----------------+--------------+
|[c, Charlie, 30]|[c, b, follow]|    [b, Bob, 36]|[b, c, follow]|
|    [b, Bob, 36]|[b, c, follow]|[c, Charlie, 30]|[c, b, follow]|
+----------------+--------------+----------------+--------------+
```

find all the reciprocal relationships in which one person is older than 30:

In [15]:

```
# More complex queries can be expressed by applying filters.
motifs.filter("b.age > 30").show()
```

```
+----------------+--------------+------------+--------------+
|               a|             e|           b|            e2|
+----------------+--------------+------------+--------------+
|[c, Charlie, 30]|[c, b, follow]|[b, Bob, 36]|[b, c, follow]|
+----------------+--------------+------------+--------------+
```

Explore some patterns from your choice using Motifs

In [16]:

```
motifs_3 = gf.find("(a)-[e]->(b); (b)-[e2]->(c); (c)-[e3]->(a)")
motifs_3.show()
```

```
+---------------+--------------+---------------+--------------+-------
--------+--------------+
|              a|             e|              b|            e2|
c|            e3|
+---------------+--------------+---------------+--------------+-------
--------+--------------+
| [d, David, 29]|[d, a, friend]| [a, Alice, 34]|[a, e, friend]|[e, Est
her, 32]|[e, d, friend]|
| [a, Alice, 34]|[a, e, friend]|[e, Esther, 32]|[e, d, friend]| [d, Da
vid, 29]|[d, a, friend]|
|[e, Esther, 32]|[e, d, friend]| [d, David, 29]|[d, a, friend]| [a, Al
ice, 34]|[a, e, friend]|
+---------------+--------------+---------------+--------------+-------
--------+--------------+
```

In [ ]:

```
```