Lab4_python_2

December 1, 2022

[4]:

```
import findspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession

findspark.init()
spark = SparkContext.getOrCreate(SparkConf().setMaster("local[4]"))
spark = SparkSession(spark)
```

[5]:

```
from pyspark.sql.types import *
from graphframes import *
from pyspark.sql import functions as F
import pandas as pd
```

[6]:

```
from graphframes.lib import AggregateMessages as AM
from pyspark.sql import functions as F
```

[7]:

```
def create_transport_graph():
    node_fields = [
        StructField("id", StringType(), True),
        StructField("latitude", FloatType(), True),
        StructField("longitude", FloatType(), True),
        StructField("population", IntegerType(), True)
    ]
    nodes = spark.read.csv("/home/spark/lab04/task2/transport-nodes.csv",␣
 ↪header=True,
                           schema=StructType(node_fields))

    rels = spark.read.csv("/home/spark/lab04/task2/transport-relationships.
 ↪csv", header=True)
    reversed_rels = (rels.withColumn("newSrc", rels.dst)
                     .withColumn("newDst", rels.src)
                     .drop("dst", "src")
                     .withColumnRenamed("newSrc", "src")
                     .withColumnRenamed("newDst", "dst")
                     .select("src", "dst", "relationship", "cost"))
    relationships = rels.union(reversed_rels)
    return GraphFrame(nodes, relationships)
```

[8]:

```
add_path_udf = F.udf(lambda path, id: path + [id], ArrayType(StringType()))
```

[10]:

```
def sssp(g, origin, column_name="cost"):
    vertices = g.vertices \
        .withColumn("visited", F.lit(False)) \
        .withColumn("distance",
            F.when(g.vertices["id"] == origin, 0).otherwise(float("inf"))) \
        .withColumn("path", F.array())
    cached_vertices = AM.getCachedDataFrame(vertices)
    g2 = GraphFrame(cached_vertices, g.edges)

    while g2.vertices.filter('visited == False').first():
        current_node_id = g2.vertices.filter('visited == False').
↪sort("distance").first().id

        msg_distance = AM.edge[column_name] + AM.src['distance']
        msg_path = add_path_udf(AM.src["path"], AM.src["id"])
        msg_for_dst = F.when(AM.src['id'] == current_node_id, F.
↪struct(msg_distance, msg_path))
        new_distances = g2.aggregateMessages(
            F.min(AM.msg).alias("aggMess"), sendToDst=msg_for_dst)

        new_visited_col = F.when(
            g2.vertices.visited | (g2.vertices.id == current_node_id), True).
↪otherwise(False)
        new_distance_col = F.when(new_distances["aggMess"].isNotNull() &
                                  (new_distances.aggMess["col1"] < g2.vertices.
↪distance),
                                  new_distances.aggMess["col1"]) \
                            .otherwise(g2.vertices.distance)
        new_path_col = F.when(new_distances["aggMess"].isNotNull() &
                              (new_distances.aggMess["col1"] < g2.vertices.
↪distance),
                              new_distances.aggMess["col2"].
↪cast("array<string>")) \
                        .otherwise(g2.vertices.path)

        new_vertices = g2.vertices.join(new_distances, on="id",␣
↪how="left_outer") \
            .drop(new_distances["id"]) \
            .withColumn("visited", new_visited_col) \
            .withColumn("newDistance", new_distance_col) \
            .withColumn("newPath", new_path_col) \
            .drop("aggMess", "distance", "path") \
            .withColumnRenamed('newDistance', 'distance') \
            .withColumnRenamed('newPath', 'path')
        cached_new_vertices = AM.getCachedDataFrame(new_vertices)
        g2 = GraphFrame(cached_new_vertices, g2.edges)

    return g2.vertices \
                .withColumn("newPath", add_path_udf("path", "id")) \
                .drop("visited", "path") \
                .withColumnRenamed("newPath", "path")
```

[11]:
```
g = create_transport_graph()
```

[12]:
```
via_udf = F.udf(lambda path: path[1:-1], ArrayType(StringType()))
```

[13]:
```
result = sssp(g, "Amsterdam", "cost")
(result
 .withColumn("via", via_udf("path"))
 .select("id", "distance", "via")
 .sort("distance")
 .show(truncate=False))
```

```
+----------------+--------+-----------------------------------------------------
--------+
|id              |distance|via
|
+----------------+--------+-----------------------------------------------------
--------+
|Amsterdam       |0.0     |[]
|
|Utrecht         |46.0    |[]
|
|Den Haag        |59.0    |[]
|
|Gouda           |81.0    |[Utrecht]
|
|Rotterdam       |85.0    |[Den Haag]
|
|Hoek van Holland|86.0    |[Den Haag]
|
|Felixstowe      |293.0   |[Den Haag, Hoek van Holland]
|
|Ipswich         |315.0   |[Den Haag, Hoek van Holland, Felixstowe]
|
|Colchester      |347.0   |[Den Haag, Hoek van Holland, Felixstowe, Ipswich]
|
|Immingham       |369.0   |[]
|
|Doncaster       |443.0   |[Immingham]
|
|London          |453.0   |[Den Haag, Hoek van Holland, Felixstowe, Ipswich,
Colchester]|
+----------------+--------+------------------------------------------------------
--------+
```

[ ]: