Dechabhol Kotheeranurak Follow

May 11, 2020 · 6 min read · Listen

Save

EDA on flight delay prediction with Apache PySpark Graphframes

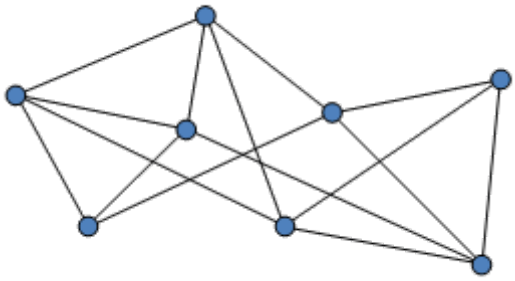
In this forum, I'll elaborate through what I've done to achieve answers for the following questions with Apache PySpark Graphframes:

- Read data into GraphFrame: Show vertices, edges.
- Find the airport with delay factor > 1.
- Find the most common airport.
- Find two airports with distances greater than 500 miles.

Disclaimer: This forum objective is to provide further information to the instructor of the 01219461 Big Data Platform course.

What are Graphframes?

Graphframes are simply one of Apache Spark libraries for graph processing which based upon Spark DataFrames. The structure of the graph is the same as the one that's in graph theory which contains vertices (V) and edges (E), so the complete graph would be G=(V, E). Moreover, Graphframes also provide with most of the common graph-related algorithms.



Load data into Dataframe

```
df = spark.read.csv('Jan_2020_ontime.csv', header = True,
inferSchema = True)
df.printSchema()
```

Output from printSchema()

```
root
 |-- DAY_OF_MONTH: integer (nullable = true)
 |-- DAY_OF_WEEK: integer (nullable = true)
 |-- OP_UNIQUE_CARRIER: string (nullable = true)
 |-- OP_CARRIER_AIRLINE_ID: integer (nullable = true)
 |-- OP_CARRIER: string (nullable = true)
 |-- TAIL_NUM: string (nullable = true)
 |-- OP_CARRIER_FL_NUM: integer (nullable = true)
 |-- ORIGIN_AIRPORT_ID: integer (nullable = true)
```

```
|-- DEP_DEL15: double (nullable = true)
|-- DEP_TIME_BLK: string (nullable = true)
|-- ARR_TIME: integer (nullable = true)
|-- ARR_DEL15: double (nullable = true)
|-- CANCELLED: double (nullable = true)
|-- DIVERTED: double (nullable = true)
|-- DISTANCE: double (nullable = true)
|-- _c21: string (nullable = true)
```

Data pre-processing

To construct the graph, I've to manually separate data into two parts (vertices and edges). For vertices, I've select unique *ORIGIN* and *ORIGIN_AIRPORT_ID* to be the columns then renamed *ORIGIN* to *id*. Next, for edges, I've only renamed *ORIGIN* and *DEST* to *src* and *dst*, respectively.

```
df_vertices = df.select('ORIGIN', 'ORIGIN_AIRPORT_ID')
.dropDuplicates(['ORIGIN']).withColumnRenamed('ORIGIN', 'id')
```

	id	ORIGIN_AIRPORT_ID
0	BGM	10577
1	INL	12343
2	PSE	14254
3	MSY	13495
4	PPG	14222
5	GEG	11884

df_vertices table

```
df_edges = df.withColumnRenamed('ORIGIN', 'src')
.withColumnRenamed('DEST', 'dst')
```

	DAY_OF_MONTH	DAY_OF_WEEK	OP_UNIQUE_CARRIER	OP_CARRIER_AIRLINE_ID	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID
0	1	3	EV	20366	EV	N48901	4397	13930
1	1	3	EV	20366	EV	N16976	4401	15370
2	1	3	EV	20366	EV	N12167	4404	11618

Now the data is ready to be load into GraphFrame. Initialize GraphFrame using the code below.

```
g = GraphFrame(df_vertices, df_edges)
```

To ensure that the data is successfully loaded into GraphFrame, print g.edges for edges and g.vertices for vertices, to show the columns and data types of each DataFrame.

Find the airport with delay factor > 1

I've assumed that the question is asking for the airports that have departure and arrival delay more than 15 minutes as they are only the columns related to the delay factor.

First, I've filtered out the airports that have value *DEP_DEL15* = 1 then use distinct to get the non-duplicate airport. After that, I've select containing *src* and *DEP_DEL15* columns. The step for arrival delay is the same as the departure, just change the *DEP_DEL15 to ARR_DEL15* and *src* to *dst*.

```
factor_more_than_1 = g.edges.filter('DEP_DEL15 == 1')
df3 = factor_more_than_1.select('src', 'DEP_DEL15')
.distinct().orderBy('src', ascending=False)

factor_more_than_12 = g.edges.filter('ARR_DEL15 == 1')
df4 = factor_more_than_12.select('dst', 'ARR_DEL15')
.distinct().orderBy('dst', ascending=False)
```

Finally, I use an inner join to the airports that have both *DEP_DEL15* = 1 and *ARR_DEL15* = 1.

```
inner = df4.join(df3, df3.src == df4.dst)
inner.show()
```


```
|BGM|      1.0|BGM|      1.0|
|INL|      1.0|INL|      1.0|
|PSE|      1.0|PSE|      1.0|
|MSY|      1.0|MSY|      1.0|
|PPG|      1.0|PPG|      1.0|
|DRT|      1.0|DRT|      1.0|
|GEG|      1.0|GEG|      1.0|
|BUR|      1.0|BUR|      1.0|
|SNA|      1.0|SNA|      1.0|
|GRB|      1.0|GRB|      1.0|
|GTF|      1.0|GTF|      1.0|
|IDA|      1.0|IDA|      1.0|
|GRR|      1.0|GRR|      1.0|
|LWB|      1.0|LWB|      1.0|
|JLN|      1.0|JLN|      1.0|
|PVU|      1.0|PVU|      1.0|
|EUG|      1.0|EUG|      1.0|
|PSG|      1.0|PSG|      1.0|
|ATY|      1.0|ATY|      1.0|
|GSO|      1.0|GSO|      1.0|
+---+---------+---+---------+
only showing top 20 rows
```

Find the most common airport

In this question, it can be interpreted in a lot of ways. I've provided three solutions to this problem.

The most common airport w.r.t arrivals

Simply just group by *dst* then count the number of occurrences of the airports.

```
g.edges.groupBy('dst').count()
.orderBy('count', ascending=False).show()
```

```
+---+-----+
|dst|count|
+---+-----+
|ATL|32187|
|ORD|25687|
|DFW|24354|
```

The most common airport w.r.t departures

Simply just group by *src* then count the number of occurrences of the airports.

```
+---+-----+
|src|count|
+---+-----+
|ATL|32190|
|ORD|25661|
|DFW|24339|
```

FYI: The above two solutions provide the same result as inDegrees and outDegrees.

The most common airport w.r.t arrivals & departures

The definition of degrees is just how many incoming edges and outgoing edges from the vertices. This is just the summation of the above solutions.

```
g.degrees.orderBy('degree', ascending=False).show(3)
```

```
+---+------+
| id|degree|
+---+------+
|ATL| 64377|
|ORD| 51348|
|DFW| 48693|
+---+------+
```

Find two airports with distances greater than 500 miles

I've select *src*, *dst*, and *DISTANCE* in order to depict the table of two airports and their distances. Then filter DISTANCE > 500 and use distinct once again to remove the redundant airports. Lastly, orderBy() to sort DISTANCE in descending order.

```
distance_greater = g.edges.select('src', 'dst', 'DISTANCE')
.filter('DISTANCE > 500').distinct()
.orderBy('DISTANCE', ascending=False)
distance_greater.show()
```

```
|HNL|BOS|  5095.0|
|BOS|HNL|  5095.0|
|HNL|JFK|  4983.0|
|JFK|HNL|  4983.0|
|EWR|HNL|  4962.0|
|HNL|EWR|  4962.0|
|HNL|IAD|  4817.0|
|IAD|HNL|  4817.0|
|ATL|HNL|  4502.0|
|HNL|ATL|  4502.0|
|DTW|HNL|  4475.0|
|HNL|DTW|  4475.0|
|ORD|HNL|  4243.0|
|HNL|ORD|  4243.0|
|OGG|ORD|  4184.0|
|ORD|OGG|  4184.0|
|MSP|HNL|  3972.0|
|HNL|MSP|  3972.0|
|IAH|HNL|  3904.0|
|HNL|IAH|  3904.0|
+---+---+--------+
only showing top 20 rows
```

However, this is not all two airports that have distances between them greater than 500 miles. Since there are some airports that can't directly fly through one flight, so the answer to this question would have to include no-direct connection airport which is the next question.

Find the max distance of all two airport connections A->B->C

The distance edges are created to remove unused columns and redundant *src* and *dst,* which also enhance the performance of motif finding.

```
from pyspark.sql.functions import desc
distance = g.edges.select('src', 'dst',
'DISTANCE').distinct().sort(desc('DISTANCE'))
distance.show()
```

I've initialized new GraphFrame with the old GraphFrame vertices and new edges distance. Inside the find function, the definition can be described as **airport a** has a direct flight to **airport b** then **airport b** has a direct flight to **airport c** where **airport a** has no direct flight to **airport c and airport c** is not the same as **airport a**.

max() function to find the furthest distance of two airport connections a->b->c.

Note: These no direct connection airports would be included in the previous question's answer (just add .filter("DISTANCE > 500")).

```
import pyspark.sql.functions as F
sub = GraphFrame(g.vertices, distance)

r = sub.find("(a)-[ab]->(b); (b)-[bc]->(c); !(a)-[]->(c)")
.filter('c.id != a.id')

r2 = r.withColumn("sum_distance", r.ab.DISTANCE + r.bc.DISTANCE)
.groupby('a.id','c.id').max('sum_distance')
.sort(desc('max(sum_distance)')).show()
```

```
+---+---+-----------------+
| id| id|max(sum_distance)|
+---+---+-----------------+
|JFK|EWR|           9945.0|
|EWR|JFK|           9945.0|
|GUM|BOS|           8896.0|
|BOS|GUM|           8896.0|
|GUM|JFK|           8784.0|
|JFK|GUM|           8784.0|
|EWR|GUM|           8763.0|
|GUM|EWR|           8763.0|
|GUM|IAD|           8618.0|
|IAD|GUM|           8618.0|
|ATL|GUM|           8303.0|
|GUM|ATL|           8303.0|
|GUM|DTW|           8276.0|
|DTW|GUM|           8276.0|
|ORD|GUM|           8044.0|
|GUM|ORD|           8044.0|
|ANC|BOS|           7872.0|
|BOS|ANC|           7872.0|
|MSP|GUM|           7773.0|
|GUM|MSP|           7773.0|
+---+---+-----------------+
only showing top 20 rows
```

To validate whether this is correct, I've created another GraphFrame to show the list of two airports connection with different input id. In this example, I want to find all of the connected airports from JFK to EW

 rmation

```
(c)").filter("a.id = 'JFK'").filter("c.id = 'EWR'")
result.show(100)
```

```
+------------+------------------+------------+------------------+------------+
|           a|                ab|           b|                bc|           c|
+------------+------------------+------------+------------------+------------+
|[JFK, 12478]| [JFK, BUF, 301.0]|[BUF, 10792]| [BUF, EWR, 282.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, SLC, 1990.0]|[SLC, 14869]|[SLC, EWR, 1969.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, PWM, 273.0]|[PWM, 14321]| [PWM, EWR, 284.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, STT, 1623.0]|[STT, 15024]|[STT, EWR, 1634.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, SYR, 209.0]|[SYR, 15096]| [SYR, EWR, 195.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, PBI, 1028.0]|[PBI, 14027]|[PBI, EWR, 1023.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, BNA, 765.0]|[BNA, 10693]| [BNA, EWR, 748.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, DCA, 213.0]|[DCA, 11278]| [DCA, EWR, 199.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, DTW, 509.0]|[DTW, 11433]| [DTW, EWR, 488.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, CLE, 425.0]|[CLE, 11042]| [CLE, EWR, 404.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, CHS, 636.0]|[CHS, 10994]| [CHS, EWR, 628.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, BOS, 187.0]|[BOS, 10721]| [BOS, EWR, 200.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, DEN, 1626.0]|[DEN, 11292]|[DEN, EWR, 1605.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, SRQ, 1041.0]|[SRQ, 14986]|[SRQ, EWR, 1034.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, RSW, 1074.0]|[RSW, 14635]|[RSW, EWR, 1068.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, ATL, 760.0]|[ATL, 10397]| [ATL, EWR, 746.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, SMF, 2521.0]|[SMF, 14893]|[SMF, EWR, 2500.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, SAV, 718.0]|[SAV, 14685]| [SAV, EWR, 708.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, BQN, 1576.0]|[BQN, 10732]|[BQN, EWR, 1585.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, ROC, 264.0]|[ROC, 14576]| [ROC, EWR, 246.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, DFW, 1391.0]|[DFW, 11298]|[DFW, EWR, 1372.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, PHX, 2153.0]|[PHX, 14107]|[PHX, EWR, 2133.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, IAH, 1417.0]|[IAH, 12266]|[IAH, EWR, 1400.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, SFO, 2586.0]|[SFO, 14771]|[SFO, EWR, 2565.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, HNL, 4983.0]|[HNL, 12173]|[HNL, EWR, 4962.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, JAX, 828.0]|[JAX, 12451]| [JAX, EWR, 820.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, BTV, 266.0]|[BTV, 10785]| [BTV, EWR, 266.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, CLT, 541.0]|[CLT, 11057]| [CLT, EWR, 529.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, FLL, 1069.0]|[FLL, 11697]|[FLL, EWR, 1065.0]|[EWR, 11618]|
|[JFK, 12478]| [JFK, RIC, 288.0]|[RIC, 14524]| [RIC, EWR, 277.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, SJC, 2569.0]|[SJC, 14831]|[SJC, EWR, 2548.0]|[EWR, 11618]|
|[JFK, 12478]|[JFK, JAC, 1894.0]|[JAC, 12441]|[JAC, EWR, 1874.0]|[EWR, 11618]|
```