Big Data Management

Project nr 3: Analysing Flight Interconnected Data

Team members: Anna Maria Tammin, Maria Anett Kaha

Provided dataset

For analysing flight interconnected data, we used the 2009.csv file provided by the course organizers in Moodle. The dataset contains records of flights between various airports. Each row represents a direct flight connection between two airports, along with basic information such as the source and destination.

Data Transformation

Each row of the provided dataset represented an edge on the flight graphframe. After reading in the data, we created a dataframe of all the vertices (nodes) in the graph by selecting all source and destination airports from the flight data. Each node represents an airport. We also created a separate dataframe for the edges by renaming the origin and destination columns in the flight dataframe to “src” and “dst” as is needed to create the GraphFrame in Spark. Having created the necessary dataframes, we then combined them using the GraphFrame function to create our flight graph which could then be analysed.

Queries

Query 1 - Compute different statistics: in-degree, out-degree, total degree and triangle count.

We calculated the in-degree, out-degree and total degree of each node by counting how many times it appeared as either the source (out-degree) or destination (in-degree) in the edges (both for degree). We used the built-in functions to validate our results and joined them to our results table. All statistics matched the built-in functions.

In-degree results.

id	our_inDegree	inDegree
IAH	182088	182088
JAX	28813	28813
ABQ	35577	35577
IND	38198	38198
BOS	110463	110463
GRR	13970	13970
MEM	71721	71721
PBI	25496	25496
XNA	13764	13764
LBB	8004	8004
BTV	6021	6021
VPS	6958	6958
SYR	9330	9330
JFK	119571	119571
MBS	3443	3443
SBN	4527	4527
PDX	52251	52251
RDD	1433	1433
LNK	2765	2765
HPN	10661	10661

only showing top 20 rows

Out-degree results.

id	our_outDegree	outDegree
IAH	182097	182097
JAX	28810	28810
ABQ	35582	35582
IND	38201	38201
GRR	13973	13973
LBB	8002	8002
MEM	71713	71713
BTV	6028	6028
BOS	110460	110460
PBI	25500	25500
XNA	13755	13755
VPS	6959	6959
SYR	9336	9336
JFK	119574	119574
MBS	3444	3444
SBN	4526	4526
PDX	52242	52242
RDD	1433	1433
LNK	2765	2765
HPN	10657	10657

only showing top 20 rows

Degree results.

id	our_degree	degree
IAH	364185	364185
JAX	57623	57623
ABQ	71159	71159
IND	76399	76399
BOS	220923	220923
GRR	27943	27943
LBB	16006	16006
MEM	143434	143434
BTV	12049	12049
PBI	50996	50996
XNA	27519	27519
VPS	13917	13917
SYR	18666	18666
JFK	239145	239145
MBS	6887	6887
SBN	9053	9053
PDX	104493	104493
RDD	2866	2866
LNK	5530	5530
HPN	21318	21318

only showing top 20 rows

To find how many triangles each node is part of, we first converted the directed graph into an undirected graph by adding reversed edges (src ⟷ dst). We also removed all duplicate edges between pairs of nodes to find only distinct triangles. To find all triangles in the graph we first look for paths a → b and b → c and then select triangles where there is an edge c → a. After finding all unique triangles in the graph, we counted how many times each node was seen in a triangle. We validated these results by comparing the triangle count of each node to the built-in function and displayed all the node counts that differed. The resulting display was empty meaning all triangle counts were correct.

id	count
IAH	1338
JAX	342
ABQ	311
IND	612
GRR	96
LBB	23
MEM	1105
BTV	34
BOS	860
PBI	168
XNA	97
VPS	10
SYR	82
JFK	942
MBS	6
SBN	13
PDX	413
RDD	1
LNK	38
HPN	36

only showing top 20 rows

Query 2 - Compute the total number of triangles in the graph.

We used the same logic from the previous query to find the total number of triangles in the graph. The only difference is that we just returned the count of distinct triangles, not the count of node appearances. We implemented this query separately, but also added a version that uses the results from the previous query (count for each node). In this variation we sum all the individual node triangle counts and divided it by three since each triangle is included three times (once for each of its 3 vertices). We performed this calculation on the built-in function as well to validate our results. All the mentioned queries gave the same result: there are 16015 distinct triangles in the flight data graph.

Precomputation for query 3 and 4:

Since queries 3 and 4 need similar data as inputs, we decided to do some precomputation to get the data in the desired format. First we compute a lookup table for nodes, where each node's id is mapped to its index. Secondly we compute the adjacency matrix, where a value 1 in row i column j shows that a path from node j to node i exists. Similarly we compute a distance matrix, where the value in row i column j shows the actual distance between node j and node i. In the distance matrix, value 0 represents a missing direct flight.

Query 3 - Compute a centrality measure of your choice natively on Spark using Graphframes.

For this query, we chose to compute **closeness centrality** as a measure of how "close" an airport is to all other airports in the network. We calculated closeness with two definitions of a shortest path. First one being the number of edges between two nodes, second being the actual distance between two nodes. It helps to identify key airports that can reach other airports with the shortest paths.

We defined a function for the BFS (Breadth-first search). This function calculates the shortest path to each node from one specific starting node and returns the closeness measure of that node, which is defined to be $(n-1)/sum(distances)$ where n is the number of nodes in the graph and distances is a list of distances to every node from the starting node. This is done for both definitions of shortest paths.

Results:

id	centrality_edges	centrality_dist
IAH	0.5983773	7.851529E-4
JAX	0.47504026	6.9578097E-4
ABQ	0.50687283	7.6872966E-4
IND	0.5139373	8.692603E-4
GRR	0.47049442	8.364784E-4
LBB	0.40466392	7.721947E-4
MEM	0.5662188	8.7641896E-4
BTV	0.43382353	6.0750963E-4
BOS	0.53636366	5.937573E-4
PBI	0.45807454	6.000716E-4
XNA	0.47580644	8.764086E-4
VPS	0.42753622	7.3864305E-4
SYR	0.43768546	6.703997E-4
JFK	0.5462963	6.488807E-4
MBS	0.41143653	8.0058404E-4
SBN	0.43703705	8.5602776E-4
PDX	0.5305755	5.75475E-4
RDD	0.36019537	5.204612E-4
LNK	0.45807454	8.191714E-4
HPN	0.43255132	6.062599E-4

Query 4 - Implement the PageRank algorithm natively on Spark using Graphframes.

Pagerank is an iterative algorithm which finds the importance of a node (airport in our case). The formula for calculating rank is $R(u) = (1 - d)/N + d * sum(incoming)/sum(outdegree)$. Initially, each airport is given a rank of 1. First, the contribution of each edge is calculated. Then all of the contributions are summed up, based on the node where the edge goes to (giving us the part sum(incoming)/sum(outdegree) in the formula). Finally the new rank is calculated based on the formula.

Results:

id	pagerank	rank
IAH	5.662321605756482	5.662321605756484
JAX	1.6077915704615784	1.6077915704615786
ABQ	1.7429124686716673	1.7429124686716668
IND	1.9850210090574687	1.9850210090574683
GRR	0.836374678808676	0.8363746788086759
LBB	0.5372659782130557	0.5372659782130556
MEM	4.676034097666703	4.676034097666703
BTV	0.5764612714461971	0.5764612714461971
BOS	2.8138753782688175	2.8138753782688166
PBI	1.1467079819343051	1.1467079819343051
XNA	0.89689132633325	0.8968913263332497
VPS	0.40682553897292073	0.4068255389729206
SYR	0.7563288719817197	0.7563288719817195
JFK	3.508955992981467	3.5089559929814675
MBS	0.3536217424004337	0.35362174240043376
SBN	0.41375417295624806	0.41375417295624806
PDX	2.3823714398188938	2.3823714398188933
RDD	0.2915821901287276	0.2915821901287276
LNK	0.5806182004896793	0.5806182004896793
HPN	0.6290724318775694	0.6290724318775694

Query 5 - Find the group of the most connected airports.

To find the group of the most connected airports (largest connected component in the graph) we aimed to replicate the connectedComponents() function and then extract the component with the largest number of nodes in it. To find all connected components, we first initialized the components so that each node belongs to a separate component. After this, we added all nodes that are reachable from the initial node to the respective components. This process is repeated until the results converge. In this specific case, it takes 4 iterations. Finally, the largest connected component is identified by selecting the component with the largest size.
After each iteration we tracked how many distinct components remain by counting the unique labels. Below we visualized how the component groups developed during the 4 iterations. Before the first iterations, all the nodes belonged to separate components.

Iteration 1.

```
+-----+----+
|label|size|
+-----+----+
|  ATL| 124|
|  ABQ|  28|
|  DEN|  21|
|  DFW|  17|
|  ANC|  14|
|  DTW|  13|
|  ABE|   8|
|  SLC|   7|
|  MSP|   7|
|  LAX|   7|
|  JNU|   5|
|  ALB|   4|
|  ACV|   3|
|  JFK|   2|
|  ACK|   2|
|  AUS|   2|
|  PHX|   2|
|  ORD|   2|
|  PDX|   2|
|  CVG|   2|
+-----+----+
only showing top 20 rows
```

Iteration 2.

```
+-----+----+
|label|size|
+-----+----+
|  ABE| 200|
|  ABQ|  57|
|  ABI|  23|
|  ADK|   9|
|  ANC|   5|
|  ACK|   1|
|  ALB|   1|
+-----+----+
```

Iteration 3.

```
+-----+----+
|label|size|
+-----+----+
|  ABE| 292|
|  ABQ|   4|
+-----+----+
```

Iteration 4.

```
+-----+----+
|label|size|
+-----+----+
|  ABE| 296|
+-----+----+
```

The results of the query show that the graph contains one large connected component meaning that there is a path between all the nodes.