CAP 6318 Project - USA Flights Network Analysis

- Divya Selvaraj

Project Statement:

In this project, USA flight routes network graph has been created where the USA airports are the nodes and an edge will exist between the nodes if there is a direct connecting flight between these two airports. The edges are weighted based on the number of flights between the airports.

Directed and undirected network graphs are created by using Julia Graphs and GraphPlot packages. Interactive visualizations are created using Widgets (like Slider & dropdown) from Julia Interact package. Network analysis done in this project are categorized as below:

Part 1: USA Flights Network - Directed, Weighted

- 1.1 Complete directed, weighted network
- 1.2 Directed, weighted network - filter by edge weights
- 1.3 Identify Airports
  - 1.3a With high outdegree
  - 1.3b With high betweenness centrality

Part 2: USA Flights Network - Undirected, Weighted

- 2.1 Complete undirected, weighted network
- 2.2 Undirected, weighted network - filter by edge weights

Part 3: USA Flights Network - Map View

3.1 USA flights network by airport

3.2 USA flights network by airline

Part 4: Shortest path between 2 airports (dijkstra shortest path)

Note: More details provided in respective section.

Data Description:

There are two major datasets used for this project, one containing routes information and another containing airport information can be downloaded from https://openflights.org/data.html (https://openflights.org/data.html) Since these datasets have information for all countries, it has to be filtered to get USA flights information for this project.

They can also be accessed uing the direct links as below:

1. Routes Dataset:

Github link: https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes.dat (https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes.dat)

2. Airports Dataset:

Github link: https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat (https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat)

Datasets from links above does not have header/column information. Column names has been taken from same website (https://openflights.org/data.html (https://openflights.org/data.html))

Most important column names are the ones using "IATA" in their names which refers to unique codes assigned to airlines and airports. For example, Airline_IATA refers to unique airline code. Similarly, Source_IATA & Dest_IATA refers to unique airport codes which can be used to identify the source and destination airport respectively for each routes specified in routes dataset.

There are two more datasets used in this project which are:

Airlines Dataset:(It has various information about the airlines but it has been used in this project only to get the airlines name)

Github link: https://raw.githubusercontent.com/jpatokal/openflights/master/data/airlines.dat (https://raw.githubusercontent.com/jpatokal/openflights/master/data/airlines.dat)

USA map dataset:(JSON file that is used for US map view plot)

This dataset can be directly accessed from the VegaDataSets package. (no cleanup required, directly used in part 3 for map visualization)

Following code section includes downloading required datasets and cleanup.

Import Packages

In [1]:

```
# display setting
ENV["COLUMNS"] = 200
ENV["LINES"] = 5

# include packages
using CSV, DataFrames, HTTP
using WebIO
using Query
using Graphs, GraphPlot, SimpleWeightedGraphs
using Colors
using Interact
using VegaLite, VegaDatasets
```

Import Data Files & Cleanup

In [2]:

```
# function to read csv dataset into dataframe
read_csv_df(url) = DataFrame(CSV.File(HTTP.get(url).body, header = 0))
```

Out[2]:

```
read_csv_df (generic function with 1 method)
```

a) Routes Dataset

In [3]:

```
# read routes dataset
routes_url = "https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes
routes_df = read_csv_df(routes_url)
println(nrow(routes_df))

# assign column names to dataframe
colnames = ["Airline_IATA","Airline_ID","Source_IATA", "Source_ID","Dest_IATA", "Dest_I
rename!(routes_df, Symbol.(colnames))
```

67663

Out[3]:

67,663 rows × 9 columns

	Airline_IATA	Airline_ID	Source_IATA	Source_ID	Dest_IATA	Dest_ID	Codeshare	Stops	Eq
	String3	String7	String3	String7	String3	String7	String1?	Int64	S
1	2B	410	AER	2965	KZN	2990	*missing*	0	
2	2B	410	ASF	2966	KZN	2990	*missing*	0	
3	2B	410	ASF	2966	MRV	2962	*missing*	0	
4	2B	410	CEK	2968	KZN	2990	*missing*	0	
5	2B	410	CEK	2968	OVB	4078	*missing*	0	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

In [4]:

```
# drop columns that are not required
routes_df = select(routes_df, Not([:Codeshare, :Equipment]))

# filter by number of stops 0 for direct flights
routes_df = routes_df[( routes_df.Stops .== 0 ), :]
```

Out[4]:

67,652 rows × 7 columns

	Airline_IATA	Airline_ID	Source_IATA	Source_ID	Dest_IATA	Dest_ID	Stops
	String3	String7	String3	String7	String3	String7	Int64
1	2B	410	AER	2965	KZN	2990	0
2	2B	410	ASF	2966	KZN	2990	0
3	2B	410	ASF	2966	MRV	2962	0
4	2B	410	CEK	2968	KZN	2990	0
5	2B	410	CEK	2968	OVB	4078	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

b) Airports Dataset

In [5]:

```
# read airport dataset
airports_url = "https://raw.githubusercontent.com/jpatokal/openflights/master/data/airp
airports_df = read_csv_df(airports_url)
println(nrow(airports_df))

# assign column names to dataframe
colnames = ["Airport_ID", "Name", "City", "Country", "IATA", "ICAO", "latitude", "longi
            "Altitude", "Timezone", "DST", "Tzdatabase_time", "Type", "Source"]
rename!(airports_df, Symbol.(colnames))
```

7698

Out[5]:

7,698 rows × 14 columns

	Airport_ID	Name	City	Country	IATA	ICAO	latitude	longitude	Altitude
	Int64	String	String63?	String63	String3	String7	Float64	Float64	Int64
1	1	Goroka Airport	Goroka	Papua New Guinea	GKA	AYGA	-6.08169	145.392	5282
2	2	Madang Airport	Madang	Papua New Guinea	MAG	AYMD	-5.20708	145.789	20
3	3	Mount Hagen Kagamuga Airport	Mount Hagen	Papua New Guinea	HGU	AYMH	-5.82679	144.296	5388
4	4	Nadzab Airport	Nadzab	Papua New Guinea	LAE	AYNZ	-6.5698	146.726	239
5	5	Port Moresby Jacksons International Airport	Port Moresby	Papua New Guinea	POM	AYPY	-9.44338	147.22	146
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

In [6]:

```
# drop columns that are not required
airports_df = select(airports_df, Not([:ICAO, :Altitude, :Timezone, :DST, :Tzdatabase_t
first(airports_df, 5)
```

Out[6]:

5 rows × 7 columns

	Airport_ID	Name	City	Country	IATA	latitude	longitude
	Int64	String	String63?	String63	String3	Float64	Float64
1	1	Goroka Airport	Goroka	Papua New Guinea	GKA	-6.08169	145.392
2	2	Madang Airport	Madang	Papua New Guinea	MAG	-5.20708	145.789
3	3	Mount Hagen Kagamuga Airport	Mount Hagen	Papua New Guinea	HGU	-5.82679	144.296
4	4	Nadzab Airport	Nadzab	Papua New Guinea	LAE	-6.5698	146.726
5	5	Port Moresby Jacksons International Airport	Port Moresby	Papua New Guinea	POM	-9.44338	147.22

c) Airlines Dataset

In [7]:

```
# read airline dataset
airlines_url = "https://raw.githubusercontent.com/jpatokal/openflights/master/data/airl
airlines = read_csv_df(airlines_url)

# assign column names to dataframe
colnames = ["Airline_ID", "Name", "Alias", "IATA", "ICAO", "Callsign", "Country", "Acti
rename!(airlines, Symbol.(colnames))

# drop rows with null values
airlines_df = airlines[completecases(airlines), :]
#airlines_df = airlines[(airlines.Country .== "United States"), :]
```

Out[7]:

1,012 rows × 8 columns

	Airline_ID	Name	Alias	IATA	ICAO	Callsign	Country	Active
	Int64	String	String63?	String7?	String7?	String63?	String63?	String1
1	-1	Unknown	\\N	-	N/A	\\N	\\N	Y
2	3	1Time Airline	\\N	1T	RNX	NEXTIME	South Africa	Y
3	10	40-Mile Air	\\N	Q5	MLA	MILE-AIR	United States	Y
4	13	Ansett Australia	\\N	AN	AAA	ANSETT	Australia	Y
5	15	Abelag Aviation	\\N	W9	AAB	ABG	Belgium	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Filter airports and routes for USA

In [8]:

```
# filter USA airports and remove rows with null ("\\N") IATA
airports_usa = airports_df[(airports_df.Country .== "United States") .& (airports_df.IA
```

Out[8]:

1,251 rows × 7 columns

	Airport_ID	Name	City	Country	IATA	latitude	longitude
	Int64	String	String63?	String63	String3	Float64	Float64
1	3411	Barter Island LRRS Airport	Barter Island	United States	BTI	70.134	-143.582
2	3413	Cape Lisburne LRRS Airport	Cape Lisburne	United States	LUR	68.8751	-166.11
3	3414	Point Lay LRRS Airport	Point Lay	United States	PIZ	69.7329	-163.005
4	3415	Hilo International Airport	Hilo	United States	ITO	19.7214	-155.048
5	3416	Orlando Executive Airport	Orlando	United States	ORL	28.5455	-81.3329
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

In [9]:

```
# filter usa routes
routes_usa = filter(row -> row.Source_IATA ∈ airports_usa.IATA, routes_df) #filter sou
routes_usa = filter(row -> row.Dest_IATA   ∈ airports_usa.IATA, routes_usa) #filter de
```

Out[9]:

10,512 rows × 7 columns

	Airline_IATA	Airline_ID	Source_IATA	Source_ID	Dest_IATA	Dest_ID	Stops
	String3	String7	String3	String7	String3	String7	Int64
1	2O	146	ADQ	3531	KLN	7162	0
2	2O	146	KLN	7162	KYK	7161	0
3	3E	10739	BRL	5726	ORD	3830	0
4	3E	10739	BRL	5726	STL	3678	0
5	3E	10739	DEC	4042	ORD	3830	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Flight frequency between USA airports

In [10]:

```
# combine same route flights and get count/frequency
flight_freq = DataFrame(routes_usa |> @groupby({_.Source_IATA, _.Dest_IATA, _.Source_ID
                    count=length(_)}))

sort!(flight_freq, [:Source_IATA, :Dest_IATA])
```

Out[10]:

5,450 rows × 5 columns

	Source_IATA	Dest_IATA	Source_ID	Dest_ID	count
	String3	String3	String7	String7	Int64
1	ABE	ATL	4355	3682	3
2	ABE	CLT	4355	3876	2
3	ABE	DTW	4355	3645	1
4	ABE	MYR	4355	3515	1
5	ABE	ORD	4355	3830	1
⋮	⋮	⋮	⋮	⋮	⋮

Network Construction and Analysis

Generic graph function

In [11]:

```
function generategraph(df, src, dst, graphtype)

    # get a list of nodes from dataframe
    nodes = sort(unique([df[!, src] ; df[!, dst]]))
    nodes_len = size(nodes)[1]

    # assign node numbers
    nodes_int = [x for x in 1:nodes_len]
    nodes_dict = Dict(zip(nodes, nodes_int))

    # create directed/undirected graph based on input
    if graphtype == "directed"
        g = SimpleWeightedDiGraph(nodes_len)
    else
        g = SimpleWeightedGraph(nodes_len)
    end

    # add source, destination, weights & create edges
    ew = Int[]
    for i in 1:nrow(df)
        w = df.count[i]
        push!(ew, w)
        source = nodes_dict[df[!, src][i]]
        destination = nodes_dict[df[!, dst][i]]
        add_edge!(g, source, destination, w)
    end

return g, nodes
end
```

Out[11]:

```
generategraph (generic function with 1 method)
```

Part 1. USA Flights Network (SimpleWeightedDiGraph)

1.1 Complete directed, weighted network (All airports and routes)

The first flight network graph is a directed weighted graph which has all USA airports (549 nodes) and routes (5450 weighted edges)

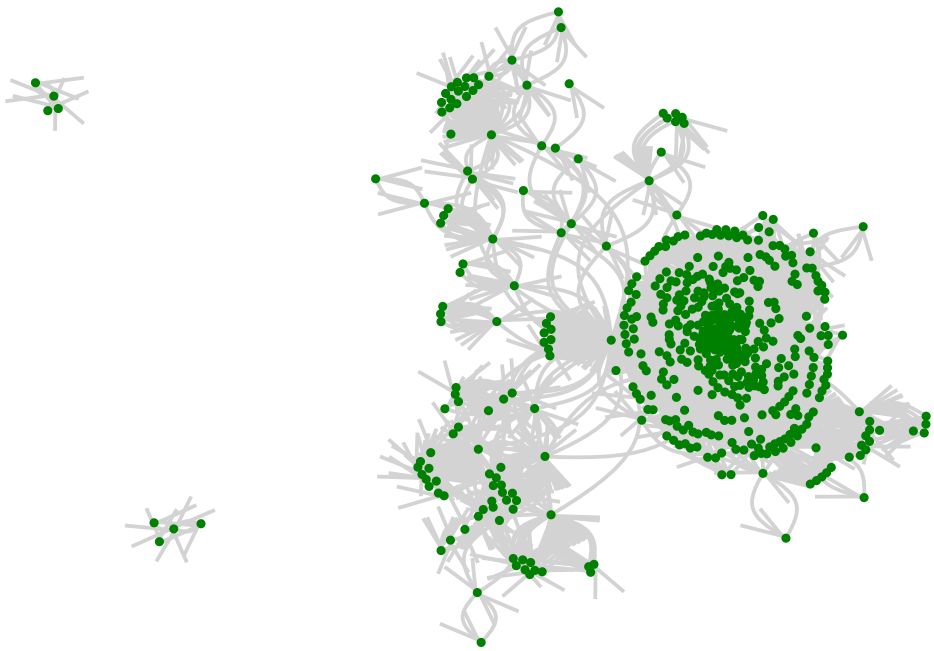
In [12]:

```
# get nodes and graph for initial USA flight network and plot
g1, nodes1 = generategraph(flight_freq, "Source_IATA", "Dest_IATA", "directed")
println("The number of edges in network is ", ne(g1))
println("The number of nodes in network is ", nv(g1))
printstyled("\n\t\t\tDirected Weighted Graph"; color=:bold)
gplot(g1, nodefillc = "green", linetype="curve", EDGELINEWIDTH = 0.5) # plot default sp
```

```
The number of edges in network is 5450
The number of nodes in network is 549
```

Directed Weighted Graph

Out[12]:



1.2 Directed, weighted network - filter by edge weights

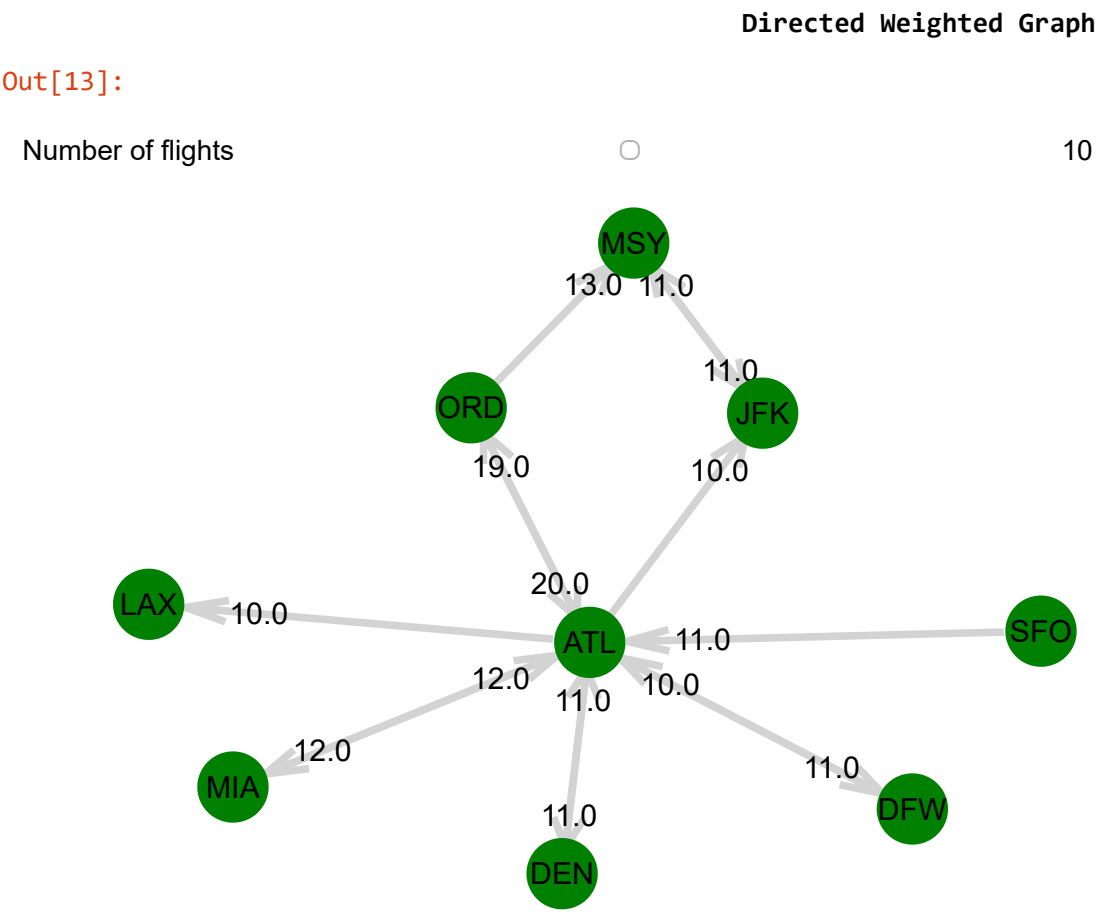
Since the initial network has high number of nodes & edges, this interactive slider can be used to filter the graph by edge weights/flight frequency to provide a better view of the network

*By default, the graph shows the network with edge weights greater than equal to 10, which means it shows the aiports between which the number of flights is greater than or equal to 10. We can see from the graph that there are **20** flights from **ORD->ATL** and **19** flights from **ATL->ORD** which are the highest values in the network.*

In [13]:

```
printstyled("\n\t\t\t\t\t\tDirected Weighted Graph"; color=:bold)
@manipulate for frequency = slider(1:1:20, show_value=true, label = "Number of flights'
    filter_flights = sort(flight_freq[(flight_freq.count .>= frequency[]), :]) # filter
    g2, nodes2 = generategraph(filter_flights, "Source_IATA", "Dest_IATA", "directed")
    weights2 = weight.(edges(g2))
    gplot(g2, nodefillc = "green", edgelabel = weights2, nodelabel = nodes2) # plot dej
end
```

Out[13]:



1.3 USA Flights Network (Identify major airports)

In this section, airports in the network with high values for outdegree and betweenness centrality are identified

1.3a Graph with node size based on outdegree

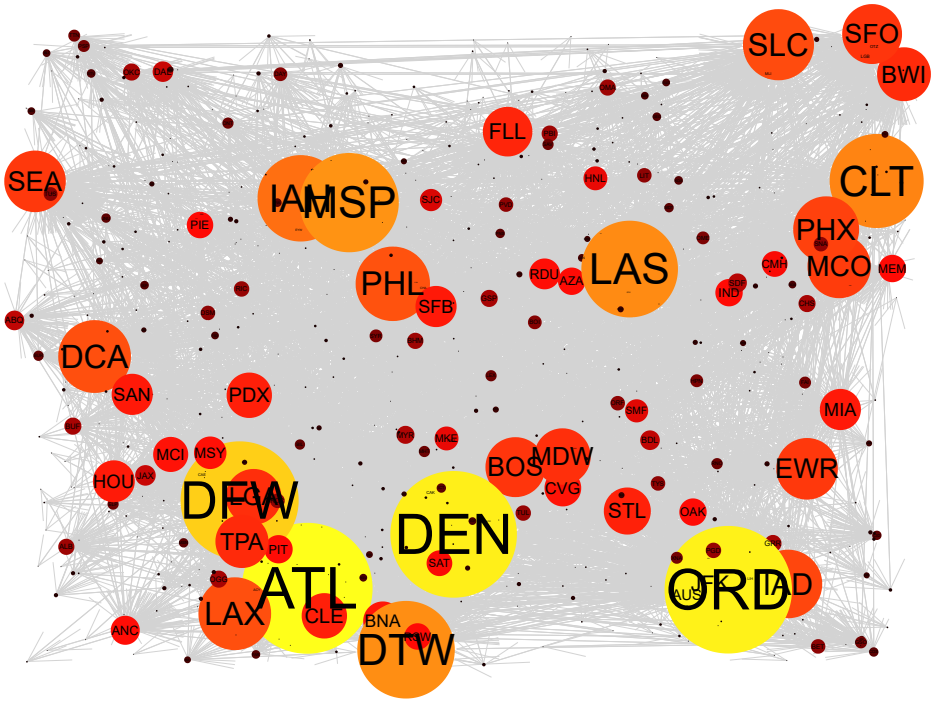
Graph has been plotted by assigning nodesize and color based on outdegree value. Nodes(Airports) in the network with high outdegree values are identified

In [14]:

```
# Use all airports and routes graph
# assign node size based on outdegree
nodesize3 = [outdegree(g1, v) for v in Graphs.vertices(g1)]

# assign color based on node size
size3 = nodesize3/maximum(nodesize3)
nodefillc3 = [RGB(i*5,i^2,i/10) for i in size3]

printstyled("\n\t\tNodes color/size by Outdegree"; color=:bold)

# plot on random layout
gplot(g1, layout = random_layout, nodefillc = nodefillc3, NODESIZE = nodesize3/1000,  r
        NODELABELSIZE = nodesize3/20, EDGELINEWIDTH = weight.(edges(g1))/50)
```

```
        Nodes color/size by Outdegree
```

Out[14]:



Airports with high outdegree values in graph:

ATL
ORD
DEN
DFW
MSP

In [15]:

```
# nodes with outdegree > 100 (airports that has over 100 destinations)
high_outdegree = findall(item -> item > 100, nodesize3)

printstyled("Airports with outdegree > 100\n\n"; color=:bold)
printstyled("Airport_IATA    Outdegree   Airport_Name\n", color = :bold)
for i in high_outdegree
    airport_name = reduce(vcat,[airports_usa[(airports_usa.IATA .== nodes1[i]) ,:].Name
    println("    ", nodes1[i],"\t\t  ", nodesize3[i] ,"       ", airport_name[])
end
```

Airports with outdegree > 100

Airport_IATA	Outdegree	Airport_Name
ATL	153	Hartsfield Jackson Atlanta International Airport
CLT	110	Charlotte Douglas International Airport
DEN	148	Denver International Airport
DFW	138	Dallas Fort Worth International Airport
DTW	114	Detroit Metropolitan Wayne County Airport
IAH	101	George Bush Intercontinental Houston Airport
LAS	113	McCarran International Airport
MSP	116	Minneapolis-St Paul International/Wold-Chamberla
in Airport		
ORD	149	Chicago O'Hare International Airport

1.3b Graph with node size based on betweenness centrality

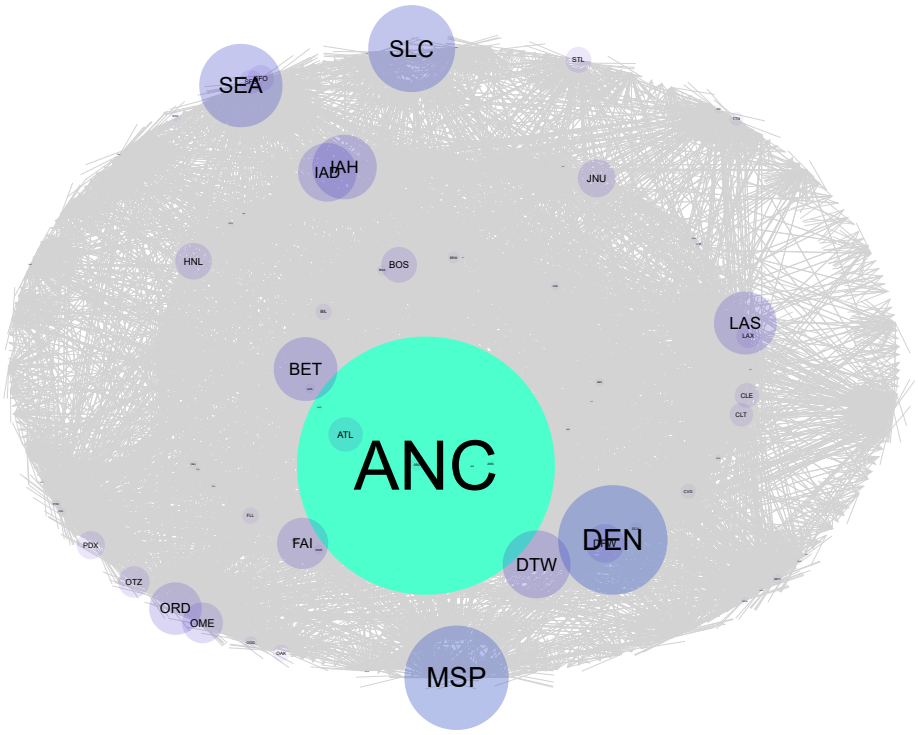
Graph has been plotted by assigning nodesize and color based on betweenness centrality value. Nodes(Airports) in the network with high betweenness centrality values are identified

In [16]:

```
# Use the all airports and routes graph
# assign node size based on betweenness centrality
nodesize4 =  betweenness_centrality(g1)

# assign color based on node size
size4 = nodesize4/maximum(nodesize4)
nodefillc4 = nodefillc = [RGBA(0.3,i,0.8,i) for i in size4] #[RGB(0.5,i^2,i/10) for i i

# define 3 shells for shell layout
nlist = Vector{Vector{Int}}(undef, 3)
nlist[1] = 1:100 # first shell
nlist[2] = 101:300 # second shell
nlist[3] = 301:nv(g1) # third shell
locs_x, locs_y = shell_layout(g1, nlist)

printstyled("\n\t\tNodes color/size by betweenness centrality"; color=:bold)

# plot on shell layout
gplot(g1, locs_x, locs_y, nodefillc = nodefillc4, NODESIZE = nodesize4,  nodelabel = nc
        NODELABELSIZE = nodesize4*30, EDGELINEWIDTH = weight.(edges(g1))/50)
```

Nodes color/size by betweenness centrality

Out[16]:



Airports with high betweenness centrality in graph:

ANC
DEN
MSP
SLC
SEA

In [17]:

```
# nodes with high betweenness centrality (centrality measure based on shortest paths)
high_bet = findall(item -> item > 0.08, nodesize4)

printstyled("Airports with betweenness centrality > 0.08\n\n"; color=:bold)
printstyled("Airport_IATA   betweenness\t Airport_Name\n", color = :bold)
for i in high_bet
    airport_name = reduce(vcat,[airports_usa[(airports_usa.IATA .== nodes1[i]) ,:].Name
    println("    ", nodes1[i],"\t       ", floor.(nodesize4[i], digits = 10), "       ",
end
```

Airports with betweenness centrality > 0.08

Airport_IATA	betweenness	Airport_Name
ANC	0.3015639047	Ted Stevens Anchorage International Airport
DEN	0.1279072637	Denver International Airport
MSP	0.1219985682	Minneapolis-St Paul International/Wold-Cham
berlain Airport		
SEA	0.0973650239	Seattle Tacoma International Airport
SLC	0.1016200246	Salt Lake City International Airport

Part 2. USA Flights Network (SimpleWeightedGraph)

In this section, an undirected weighted network is created. In order to do this, flights between same airports are combined. For example, the flight_freq dataframe will have two rows for flights between ATL & ORD airports (one for ATL->ORD and another for ORD->ATL). We will combine these two rows and add their weights to create undirected weighted flight network.

In [18]:

```
# create a new dataframe to combine flights between same airports ( a->b  & b->a)
flights_undirected = DataFrame(Airport1 = String[], Airport2 = String[], count = Int64[

for row in eachrow(flight_freq)
    src, dest, cnt = row.Source_IATA, row.Dest_IATA, row.count
    df_flights = flight_freq[(flight_freq.Source_IATA .== dest) .& (flight_freq.Dest_IA
    df_length = nrow(df_flights)

    if df_length == 1
        if isempty(flights_undirected[(flights_undirected.Airport2 .== src) .& (flights
            push!(flights_undirected, [src  dest cnt + df_flights.count[]])
        end
    elseif df_length == 0
        push!(flights_undirected, [src  dest cnt])
    end
end

sort!(flights_undirected, [:Airport1, :Airport2])
```

Out[18]:

2,787 rows × 3 columns

	Airport1	Airport2	count
	String	String	Int64
1	ABE	ATL	6
2	ABE	CLT	4
3	ABE	DTW	2
4	ABE	MYR	2
5	ABE	ORD	2
⋮	⋮	⋮	⋮

2.1 Complete undirected, weighted network (All flights & routes)

This flight network graph is an undirected weighted graph which has all USA airports (549 nodes) and routes(2787 weighted edges). Since the routes are combined between same airports, the number of edges is reduced here as compared to directed graph in part 1.1

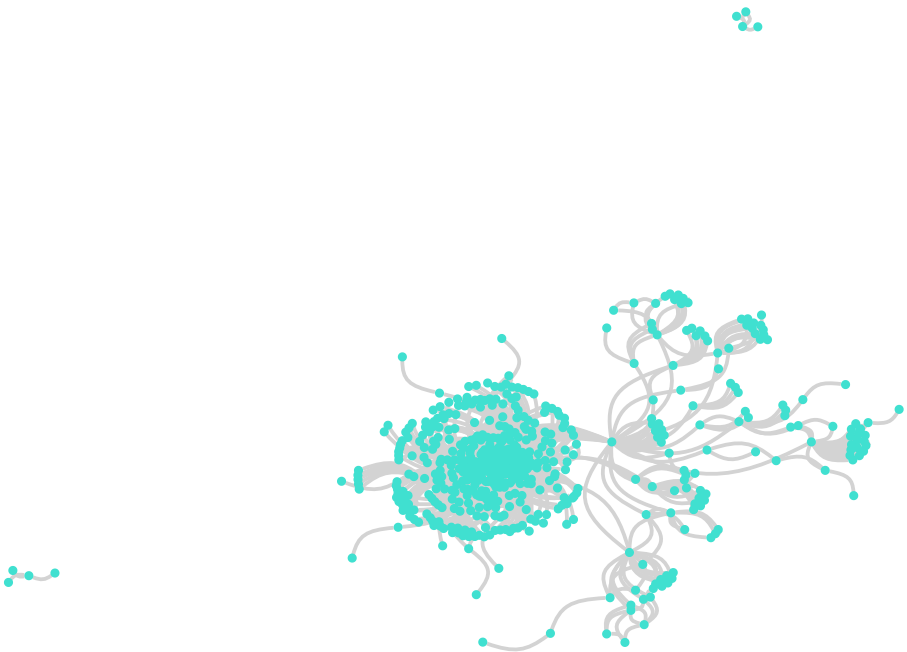
In [19]:

```
# get nodes and graph for USA flight undirected network and plot
g5, nodes5 = generategraph(flights_undirected, "Airport1", "Airport2", "undirected")
println("The number of edges in network is ", ne(g5))
println("The number of nodes in network is ", nv(g5))
printstyled("\n\t\t\tUndirected Weighted Graph"; color=:bold)
gplot(g5, linetype="curve", EDGELINEWIDTH = 0.5)
```

```
The number of edges in network is 2787
The number of nodes in network is 549
```

Undirected Weighted Graph

Out[19]:



2.2 Undirected, weighted network - filter by edge weights

Interactive slider can be used to filter the graph by edge weights/flight frequency.

By default, the graph shows the network with edge weights greater than equal to 20,

*which means it shows the aiports between which the number of flights is greater than or equal to 20. We can see from the graph that there are **39** flights between **ORD and ATL** airports which is the highest value in the network.*

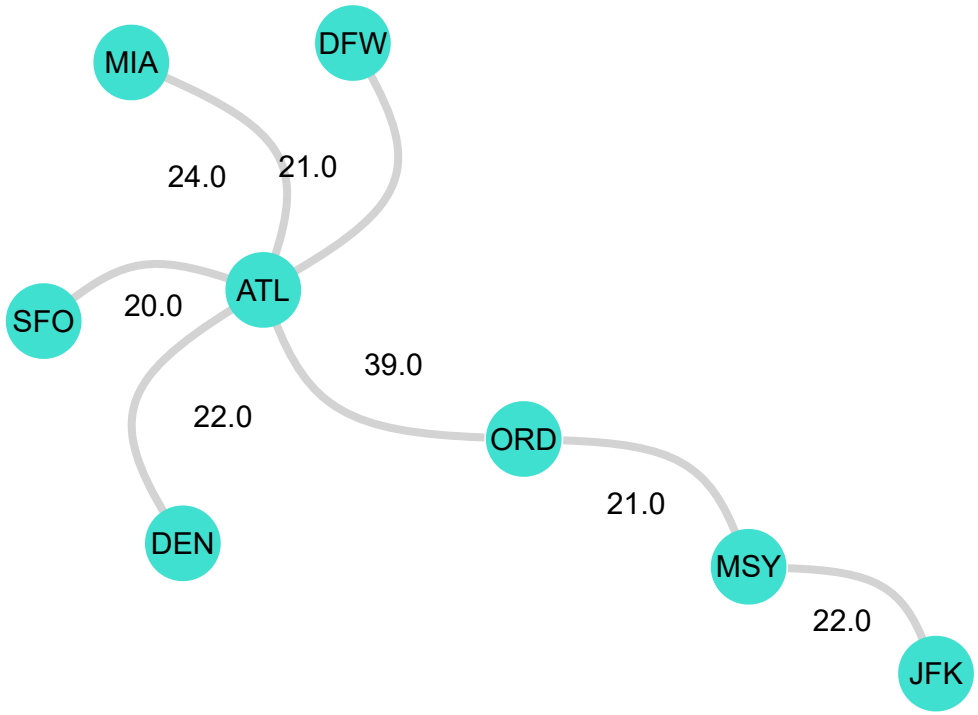
In [20]:

```
printstyled("\n\t\t\t\t\t\tUndirected Weighted Graph"; color=:bold)

@manipulate for frequency = slider(1:1:39, show_value=true, label = "Number of flights"
    filter_flights_undir = sort(flights_undirected[(flights_undirected.count .>= freque
    g6, nodes6 = generategraph(filter_flights_undir, "Airport1", "Airport2", "undirecte
    weights6 = weight.(edges(g6))
    gplot(g6, edgelabel = weights6, linetype="curve", nodelabel = nodes6)
end
```

```
                        Undirected Weighted Graph
```

Out[20]:

Number of flights ○ 20



Part 3. USA Flights Network (Map view)

In this section, a map visualization of the USA flights network has been created. The views are interactive and the network can be viewed based on the selected airport or airline

3.1. Flight Network by Source Airport

In [21]:

```
# import US map JSON dataset
us10m = dataset("us-10m")
```

Out[21]:

Vega JSON Dataset

In [22]:

```
# function to create a map view of flight network
function airportmap(airport_view)
    # plot size
    @vlplot(width = 800, height = 500) +

    # base map
    @vlplot(
        mark={
            :geoshape,
            fill=:white,
            stroke=:black
        },
        title = "Network by Source Airport - " * airports_usa[(airports_usa.IATA .== a
        data={
            values=us10m,
            format={
                type=:topojson,
                feature=:states
            }
        },
        projection={type=:albersUsa}) +

      # line marker connecting source and destination airports
    @vlplot(
        :rule,
        data=routes_usa,
        transform=[
            {filter={field=:Source_IATA,equal=airport_view}},
            {
                lookup=:Source_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "longitude"]
                },
                as=["origin_latitude", "origin_longitude"]
            },
            {
                lookup=:Dest_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "longitude"]
                },
                as=["dest_latitude", "dest_longitude"]
            }
        ],
        projection={type=:albersUsa},
        longitude="origin_longitude:q",
        latitude="origin_latitude:q",
        longitude2="dest_longitude:q",
        latitude2="dest_latitude:q",
        color={value=:lightblue}) +

    # point marker for destination aiports
    @vlplot(
        data = routes_usa,
        mark = :point,
        transform=[
```

```
            {filter={field=:Source_IATA,equal=airport_view}},
            {
                lookup=:Source_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "longitude"]
                }}],

        projection = {type=:albersUsa},
        longitude = "longitude:q",
        latitude="latitude:q",
        size={value=30},
        color={value=:green}) +

     # point marker for source aiport
    @vlplot(
        data = routes_usa,
        mark = :point,
        transform=[
            {filter={field=:Source_IATA,equal=airport_view}},
            {
                lookup=:Dest_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "longitude"]
                }}],

        projection = {type=:albersUsa},
        longitude = "longitude:q",
        latitude="latitude:q",
        size={value=10},
        color={value=:red}) +

    # text marker for displaying destination aiport name
    @vlplot(
        data = routes_usa,
        mark={
            type=:text,
            dy=-10
        },
        transform=[
            {filter={field=:Source_IATA,equal=airport_view}},
            {
                lookup=:Dest_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "IATA", "longitude"]
                }}],

        projection = {type=:albersUsa},
        longitude = "longitude:q",
        latitude= "latitude:q",
        size= {value=10},
        color= {value=:black},
        text = "IATA:n") +

     # text marker for displaying source aiport name
    @vlplot(
```

```
        data = routes_usa,
        mark={
            type=:text,
            dy=-10
        },
        transform=[
            {filter={field=:Source_IATA,equal=airport_view}},
            {
                lookup=:Source_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "IATA", "longitude"]
                }}],

        projection = {type=:albersUsa},
        longitude = "longitude:q",
        latitude= "latitude:q",
        size= {value=10},
        color= {value=:black},
        text = "IATA:n")
end
```

Out[22]:

```
airportmap (generic function with 1 method)
```

An interative dropdown has been created to view the USA flight network based on source airport selected. By default, the airport chosen is "DFW" (Dallas fort worth international airport). We can see that there are direct connecting flights from DFW to many other aiports specifically it is more in the east as compared to west. There are also some states in the northwest that has either no connectivity or very less connected aiports.

Note: Please wait for few seconds for the map to refresh after selecting the source airport from the dropdown

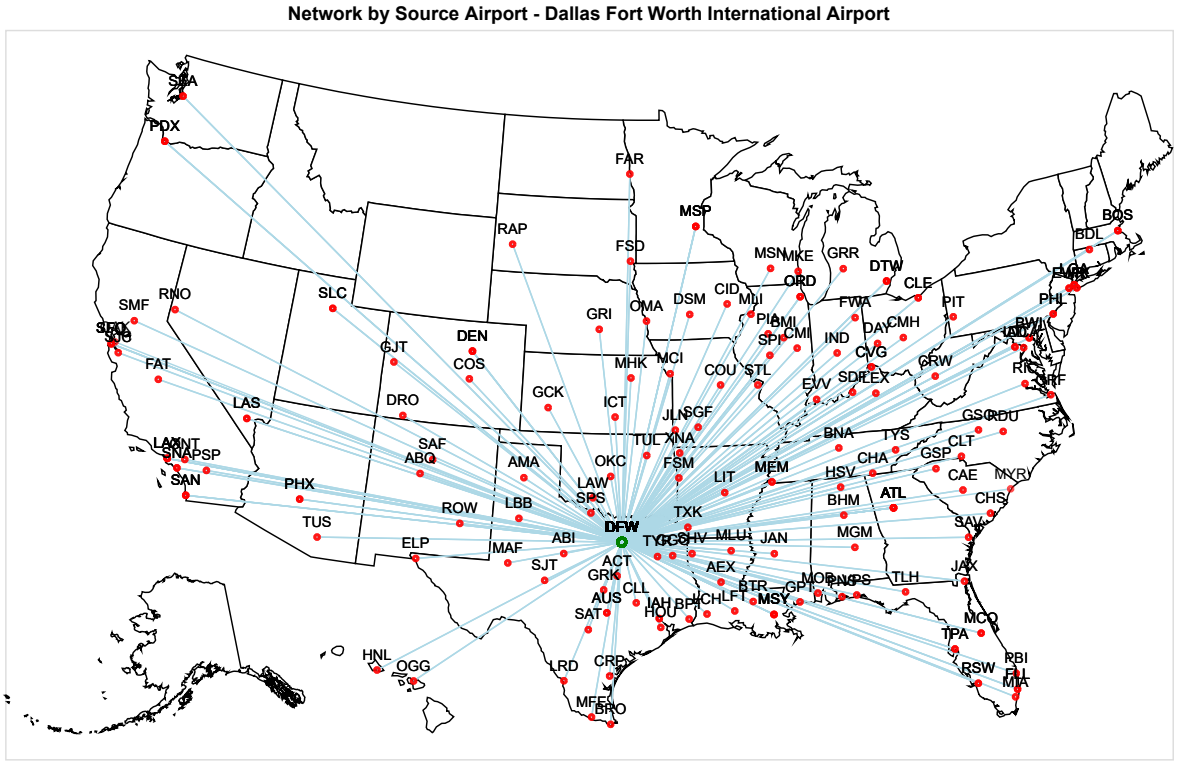
In [23]:

```
# dropdown widget (choose airport from dropdown to view network for different airport)
node_selected = dropdown(nodes1, show_value=true, label = "Select Source Airport", valu
@manipulate for src_airport = node_selected
    airportmap(src_airport)
end
```

Out[23]:

Select Source Airport

DFW



In [24]:

```
1 global_clustering_coefficient(g1)
```

Out[24]:

```
0.07757585522633605
```

3.2. Flight Network by Airlines

In [25]:

```
# function to create a map view of flight network
function airlinemap(airline_view, airline_name)
    # plot size
    @vlplot(width = 800, height = 500) +

    # base map
    @vlplot(
        mark={
            :geoshape,
            fill=:white,
            stroke=:black
        },
        title = "Network by Airlines - " * airline_name ,
        data={
            values=us10m,
            format={
                type=:topojson,
                feature=:states
            }
        },
        projection={type=:albersUsa}) +

    # point marker for aiports
    @vlplot(
        data = routes_usa,
        mark = :point,
        transform=[
            {filter={field=:Airline_IATA,equal=airline_view}},
            {
                lookup=:Dest_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "longitude"]
                }}],

        projection = {type=:albersUsa},
        longitude = "longitude:q",
        latitude="latitude:q",
        size={value=10},
        color={value=:red}) +

    # text marker for displaying destination aiport name
    @vlplot(
        data = routes_usa,
        mark={
            type=:text,
            dy=-10
        },
        transform=[
            {filter={field=:Airline_IATA,equal=airline_view}},
            {
                lookup=:Dest_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "IATA", "longitude"]
                }}],

```

```
        projection = {type=:albersUsa},
        longitude = "longitude:q",
        latitude= "latitude:q",
        size= {value=10},
        color= {value=:black},
        text = "IATA:n") +

    # line marker connecting source and destination airports
    @vlplot(
        :rule,
        data=routes_usa,
        transform=[
            {filter={field=:Airline_IATA,equal=airline_view}},
            {
                lookup=:Source_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "longitude"]
                },
                as=["origin_latitude", "origin_longitude"]
            },
            {
                lookup=:Dest_IATA,
                from={
                    data=airports_usa,
                    key=:IATA,
                    fields=["latitude", "longitude"]
                },
                as=["dest_latitude", "dest_longitude"]
            }
        ],
        projection={type=:albersUsa},
        longitude="origin_longitude:q",
        latitude="origin_latitude:q",
        longitude2="dest_longitude:q",
        latitude2="dest_latitude:q",
        color={value=:lightblue})

end
```

Out[25]:

```
airlinemap (generic function with 1 method)
```

In [26]:

```
# list of airlines having flights
airlines_list = unique(routes_usa[!, "Airline_IATA"];)

# dict to get airlines name
airlines_dict = Dict()
for i in airlines_list
    airlines_name = airlines_df[(airlines_df.IATA .== i), :].Name
    if isempty(airlines_name)
        airlines_dict[i] = i # K3 airlines - no airline map, display airline IATA
    elseif length(airlines_name) == 1
        airlines_dict[i] = airlines_name[] # DL, AA - unique name
    elseif length(airlines_name) > 1
        airlines_dict[i] = airlines_df[(airlines_df.IATA .== i), :].Name[1] # K5 - m
    end
end
```

An interative dropdown has been created to view the USA flight network based on Airline selection. By default, the airlines chosen is "DL" (Delta airlines). We can see that there are direct connecting flights between many airports. We can also see that certain airports appear to be hubs for delta airlines like ATL, MSP, SLC, JFK, etc.

Note: Please wait for few seconds for the map to refresh after selecting airline from the dropdown

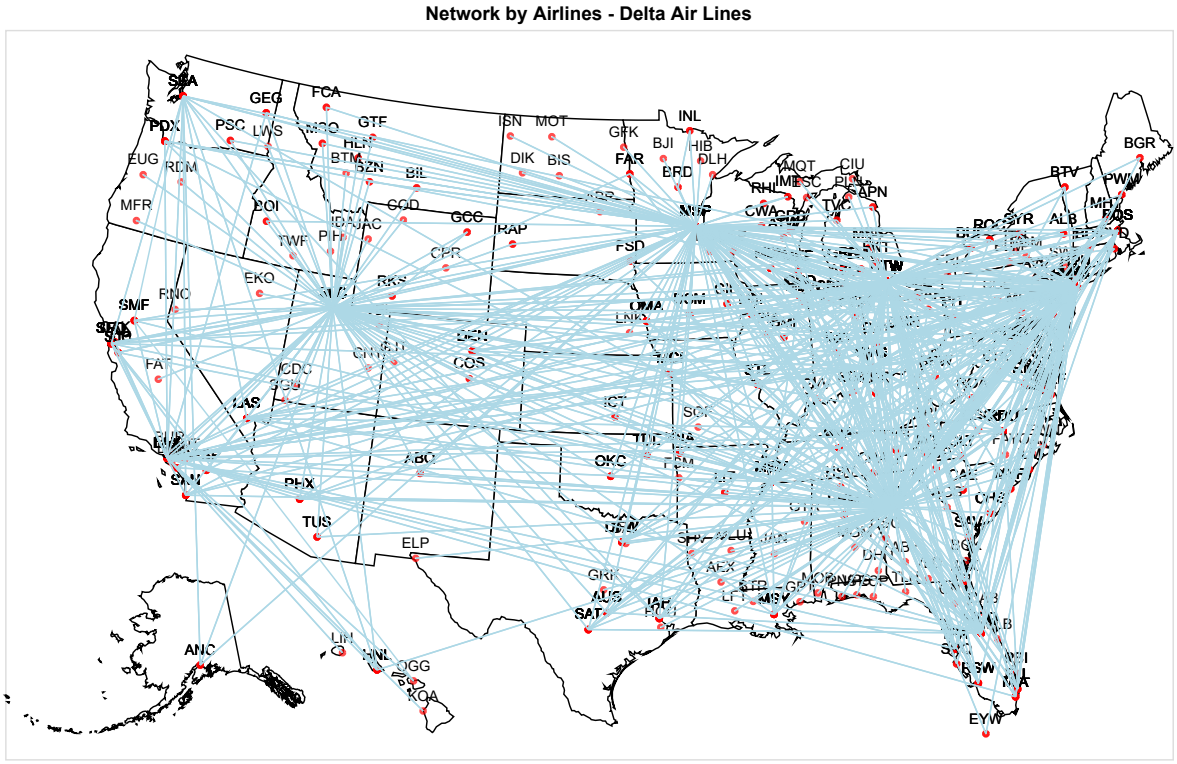
In [27]:

```
# interactive dropdown (choose airport from dropdown to view network for different aril
airline_selected = dropdown(airlines_list, show_value=true, label = "Select Airlines",
@manipulate for al = airline_selected
    airline_name = airlines_dict[al]
    airlinemap(al, airline_name)
end
```

Out[27]:

Select Airlines

DL



Part 4. Shortest path (select source and destination)

In this section, shortest path between two airports can be identified by choosing a source and destination airport from the respective dropdowns. If there is a direct connecting flight, source and destination airports chosen will be displayed. If not, the airports through which source and destination can be connected by shortest distance calculated using dijkstra algorithm will be displayed.

In [28]:

```
# Update weights to one to find shortest path
flights_airports = DataFrame(routes_usa |> @groupby({_.Source_IATA, _.Dest_IATA, _.Sour
                   count=1}))

# create graph
g7, nodes7 = generategraph(flights_airports, "Source_IATA", "Dest_IATA", "undirected")
```

Out[28]:

```
({549, 2787} undirected simple Int64 graph with Float64 weights, String3["AB
E", "ABI", "ABL", "ABQ", "ABR", "ABY", "ACK", "ACT", "ACV", "ACY"  …  "WNA",
"WRG", "WRL", "WSN", "WTK", "XNA", "YAK", "YKM", "YNG", "YUM"])
```

In [29]:

```
nodes_int = [x for x in 1:size(nodes7)[1]]
nodes_IATA_dict = Dict(zip(nodes7, nodes_int))

nodes_ids_dict = Dict(zip(nodes_int, nodes7))

src_selected = dropdown(nodes1, show_value=true, label = "Choose Source Airport", value
dst_selected = dropdown(nodes1, show_value=true, label = "Choose Destination Airport",

@manipulate for src = src_selected, dst = dst_selected
    shortest_path = (enumerate_paths(dijkstra_shortest_paths(g7, nodes_IATA_dict[src]),
    println("Shortest path between ",src_selected[], " and ", dst_selected[])
    for i in shortest_path
        print(nodes_ids_dict[i],"\t")
    end
    println("")
end
```

```
Shortest path between JFK and MSP
JFK     MSP
```

Out[29]:

Choose Source Airport

ILI

Choose Destination Airport

MSP

```
Shortest path between ILI and MSP
ILI     ANC     MSP
```