# CSCE 221: Programming Assignment 6 The Final Challange

Due on Tuesday, April 30, 2017

*Dr. Leyk*

**Robert Quan**

**Abstract**
In this assignment we create a graph to represent the direct flights between different departure cities. By combining our previously studied Data Structures, we are able to find any path between any two cities in the least amount of flight connections necessary. The Data Structures we will use are the Graph, Dequeue, Adjacency List, Lists, Queues, Stacks and Unordered Sets ADTs. This is the final project for CSCE 221 Spring 2017.

# Problem 1

**A)** Describe your **Data Structure** used in your project

In this program we used the following Data Structures for the implementation: the **Graph** Data Structure which organized and builds the map between Verticies and their respective connections between them. The class contains a vector called **adj_list** which contains a list of Edge_List objects which are the connections to that between adjacent vertices. The Graph Data Strucutre also contains a vector of **Vertex** objects which store the individual vertex objects from our input data. Another Data Strucutre that I implemented in this class is the **Unordered Set** which is used for partitioning the groups for part two, when finding if the input cities can be partitioned into two disjoined sets.

This **Graph** Class functions will build the graph, display the graph and find the shortest path from a vertex A to a vertex B.

**B)** Describe the **Algorithms** used in your Project

The Bredth-First Search Algorithm (BFS) is used in find the shortest path from a vertex A to a vertex B in our graph. The algorithm works by implementing a Queue Data Structure that will store the initial starting vertex. It will tests and add each adjacent vertex untill the desired end destination. If the vertex is not found after visiting the first connecting flights, it will continue onto each new vertex that was enqueued in the queue and repeat the test to check the new connections. Once the desired end vertex is found, it will display the output from the shortest path from the start vertex to the end vertex. This is done by recording a parent_node member function for each adjacent vertex that we visit. Ultimately, this shortestPath(int A,int B) will find the shortest path from a starting Vertex A, to an ending Vertex B in our graph.

**C)** Evidence
I tested the input data 1, 2, and 3 to check whether the connecting flights could be partitioned into a disjoined set. The input data 1 and 3 were successfully partitioned since each adjacent vertex could be separated into two individual sets. Input data 2 failed because city number 2 was placed into both sets, and failed out test for a disjoined set.
Below are some screenshots of the testing of my code.

```
[rquan@fedoration A6]$ ./main input1.data
Number of Cities: 4      Number of Connections: 4
--------------------
0   ||  1   3
1   ||  2   0
2   ||  1   3
3   ||  0   2
--------------------
Group 1 is: 2 0
Group 2 is: 3 1
We can partition the cities correctly!
Please enter two vertices you would like to find the path two:
0
2
0->1->2
The Shortest path is: 2 connection.
```

Figure 1: Testing the code for correctness, input1

```
[rquan@fedoration A6]$ ./main input3.data
Number of Cities: 7      Number of Connections: 6
--------------------
0   ||  1   2
1   ||  0   3   4
2   ||  0   5   6
3   ||  1
4   ||  1
5   ||  2
6   ||  2
--------------------
Group 1 is: 6 0 5 3 4
Group 2 is: 2 1
We can partition the Citites correctly!
Please enter two vertices you would like to find the path two:
0
6

 The Shortest path is:
0->2->6
```

Figure 2: Testing the code for correctness, input3

```
[rquan@fedoration A6]$ ./main input3.data
Number of Cities: 7      Number of Connections: 6
--------------------
0   ||  1   2
1   ||  0   3   4
2   ||  0   5   6
3   ||  1
4   ||  1
5   ||  2
6   ||  2
--------------------
Group 1 is: 6 0 5 3 4
Group 2 is: 2 1
We can partition the cities correctly!
Please enter two vertices you would like to find the path two:
0
2
0->2
The Shortest path is: 1 connection.
```

Figure 3: Testing the code for correctness, input3