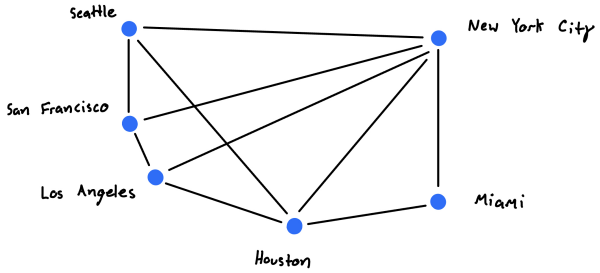
Name: _____ **ESMI Applied Math**
Due: 06/20 **Worksheet 9**

Problem 1. Learn about a different network centrality method other than the two we covered in class (we covered degree centrality and eigenvalue centrality) and describe the algorithm below. Some options can be Katz centrality, PageRank centrality, or Betweeness centrality, but you are not limited to these options. Additionally write pseudocode (that you can understand) for the algorithm below. If you type in "network centrality" or "network centrality algorithms" to Google then you will get lots of good options that you can choose from.

Problem 2. Learn about another community detection algorithm other than the Girvan-Newman algorithm (which we covered in class). Some options can be Louvain community detection, walktrap community detection, or hierarchical aggolmerative clustering, but you are not limited to these options. Additionally write pseudocode (that you can understand) for the algorithm below. If you type in "community detection algorithms" or "clustering algorithms" to Google then you will get lots of good options that you can choose from.

Problem 3. (Extra credit, I know you're not getting a grade for this class but this is an excellent problem to practice converting pseudocode into python code.) From the pseudocode presented in lecture on 6/16, write a python function that finds the shortest path from node starting node s to ending node t. The function should take as input an adjacency list, the label of the starting node s, and the label of the ending node s and output a shortest path from s to t.

For example, consider the following unweighted graph



and it's corresponding adjacency list

```
flightNetUnweighted = {
    "Seattle": ["San Francisco", "Houston", "New York City"],
    "San Francisco": ["Seattle", "Los Angeles", "New York City"],
    "Los Angeles": ["San Francisco", "New York City", "Houston"],
    "Houston": ["Los Angeles", "Seattle", "New York City", "Miami"],
    "Miami": ["Houston", "New York City"],
    "New York City": ["Seattle", "San Francisco", "Los Angeles", "Houston", "Miami"]
}
```

Write the function described above then test it on the above adjacency list to find the shortest path from "Miami" to "San Francisco". If you are stuck, follow the pseudocode from lecture on 6/16 by hand first then try to start coding. Testing the code could look something like this:

```
path = shortestPath(flightNetUnweighted, "Miami", "San Francisco")
print(path)    # should print ["Miami", "New York City", "San Francisco"]
```