Data Structures

Directions

- DO NOT add new import statements.
- DO NOT add any new class attributes
- DO NOT change any of the method's signatures.
- DO NOT modify the given constructor.

To complete this Lab, you will implement the **FlightPathGraph()** constructor, **addEdge()**, and **removeEdge()** methods for the graph of cities (strings). Each vertex in this graph representation is a City, which is a simple linked list node with a name and a next.

The method signatures are already given, do not modify these or add any new ones.

There is one class attribute provided for you in FlightPathGraph.java:

- "**flightPaths**" which is a reference to an adjacency list for the graph
    - This is **not** initialized. You will initialize this in the constructor with the given vertices, and then you will use this in addEdge() and removeEdge().

FlightPathGraph(String[] cities) *(constructor)*

This method initializes the graph with the given list of String names. Each name is a unique vertex which you should add to the graph.

First, initialize the flightPaths adjacency list array to the same size as the parameter cities array. Then, for each index in flightPaths, initialize it to a new City node, with the corresponding name from the same index in the input array.

In the FlightPathGraphTest.java file, you are provided a completed test for this method.

addEdge(String departure, String arrival)

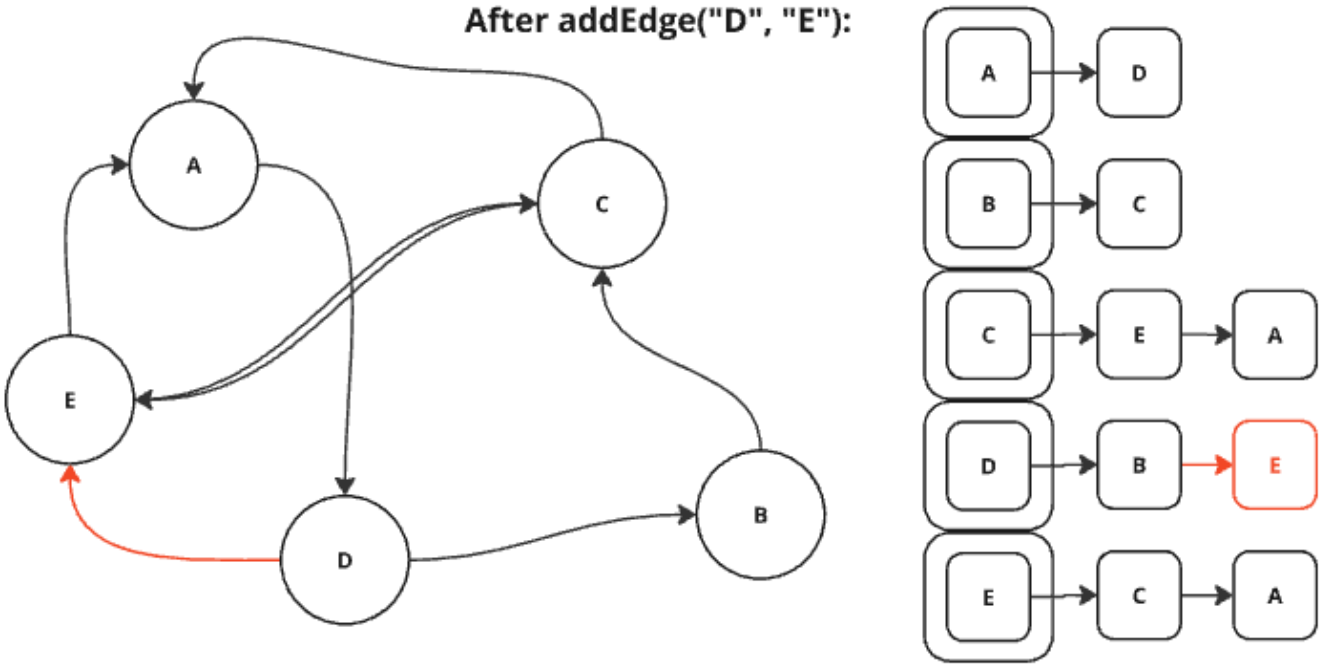
This method adds a new directed edge from the departure vertex to the arrival vertex. If an edge already exists, do nothing.

Do **NOT** add loops (edges from a vertex to itself, i.e. u -> u). You can't take a flight from an airport to the same airport. So addEdge("a", "a") should NOT add an edge.

To add an edge, you should first find the index in flightPaths with the departure City. Remember, that index is actually the head of a linked list. If the arrival city is NOT already in that linked list (aka edge list for the departure city), then add it to the end of the list.

Once you complete this method, the drivers addEdge button will start to function. Select two different cities, then add an edge to draw a line between those vertices. If your code correctly adds a directed edge, then the edges will appear as red. Undirected edges are represented by black lines, and will appear if you add a symmetric edge (i.e. u -> v AND v -> u).

Here is the graph from above after a call to addEdge("D","E")

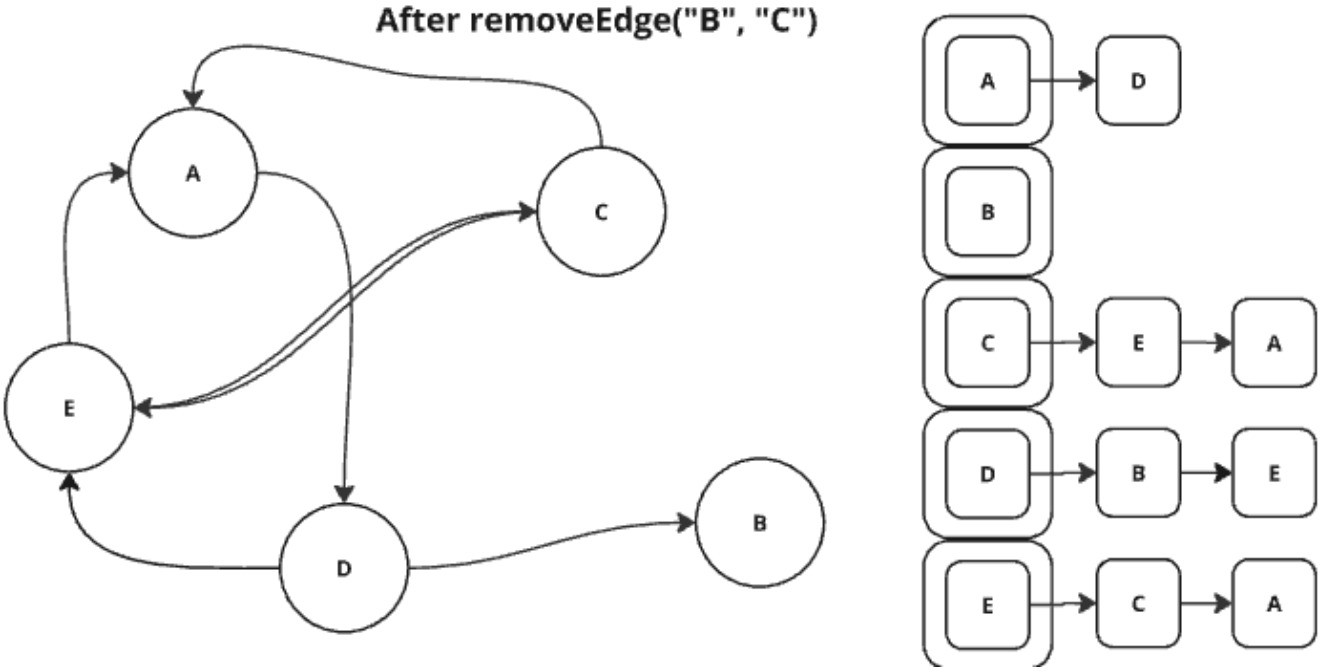


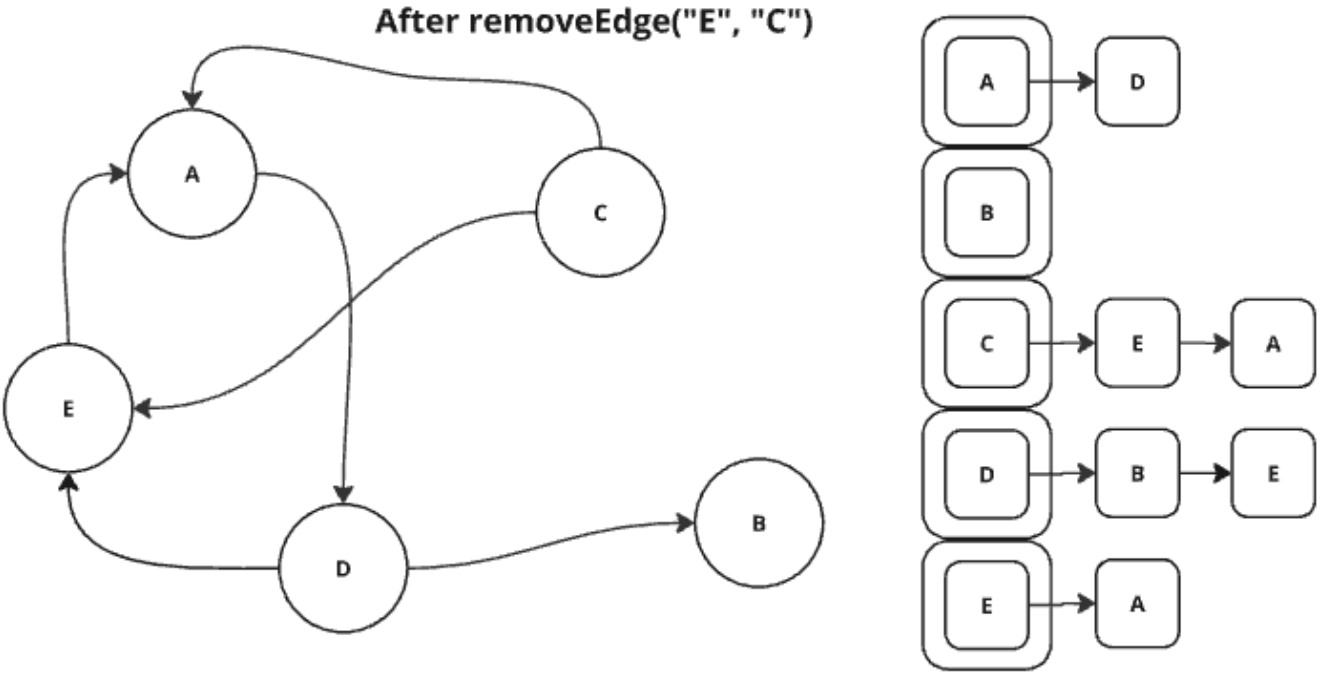
removeEdge(String departure, String arrival)

This method removes the DIRECTED edge between the departure vertex and arrival vertex. If neither edge exists, do nothing.

Search the adjacency list (flightPaths) for the departure city, and if you find it search its corresponding linked list for the arrival city (and remove that node if you find it).

Here is what the graph **AFTER** addEdge("D","E") would look like after performing removeEdge("B","C"), then removeEdge("E","C").




After removeEdge("E", "C")
A
C
E
D
B
A
D
B
C
E
A
D
B
E
E
A