Complete the solution to the HPAir problem. The input to the program consists of three text files, as follows:

cityFile

flightFile

requestFile

Each line contains the name of a city that HPAir serves. The names are in alphabetical order.

Each line contains a pair of city names that represent the origin and destina- tion of one of HPAir’s flights.

Each line contains a pair of city names that represent a request to fly from some origin to some destination.

You can make the following assumptions:

* Each city name contains at most 15 characters. Pairs of city names are separated by a comma.
* HPAir serves at most 20 cities.
* The input data is correct.

For example, the input files could appear as

cityFile:

flightFile:

requestFile:

Albuquerque Chicago San Diego

Chicago,

Chicago,

Albuquerque,

Albuquerque, Albuquerque, San Diego,

San Diego Albuquerque Chicago

San Diego Paris Chicago

For this input, the program should produce the following output:

Request is to fly from Albuquerque to San Diego. HPAir flies from Albuquerque to San Diego.

Request is to fly from Albuquerque to Paris. Sorry. HPAir does not serve Paris.

Request is to fly from San Diego to Chicago. Sorry. HPAir does not fly from San Diego to Chicago.

Begin by implementing the ADT flight map as the C++ class Map. Use the stack version of isPath. Because getNextCity is the primary operation that the search algorithm performs on the flight map, you should choose an implementation that will efficiently determine which cities are adjacent to a given city. If there are *n* cities numbered 1, 2, . . ., *n*, you can use *n* chains of linked nodes to represent the flight map. You place a node on list *i* for city *j* if and only if there is a directed path from city *i* to city *j*. Such a data structure is called an adjacency list; Figure 6-14 illustrates an adjacency list for the flight map in Figure 6-6. Chapter 20 discusses adjacency lists further when it presents ways to represent graphs. At that time, you will learn why an adjacency list is a good choice for the present program.

To simplify reading the input text files, define a class that includes the following methods:

// *Returns a name from the next line in a text file.* +getName(): string

// *Returns a pair of two names from the next line in a text file.* +getNamePair(): Pair

In the implementation of the HPAir problem (see the previous programming problem), the search for the next unvisited city adjacent to a city *i* always starts at the beginning of the *i*th chain in the adjacency list. This approach is actually a bit inefficient, because once the search visits a city, the city can never become unvisited. Modify the program so that the search for the next city begins where the last search left off. That is, maintain an array of try-next pointers into the adjacency list.