CS310 Algorithms
Project 2: Graph Project

This is a **group** project. Each group should have 3-4 members. If you have difficulty in finding a group, please let me know as soon as possible. You could develop programs with any programming languages you are confident with.

Objectives

- To implement the Graph data structure in **adjacency-list** form.
- To implement a couple of DFS based algorithms using DFS.

Due on: Dec 11 23:55PM

Problems

Implement a program that reads an **undirected graph**, represented as pairs of node/vertex names (format below), one pair for each edge. It then will identify the connected components and, for each component, whether it is acyclic or not. It prints each connected component as a list of node names (without repeating), along with an indication of whether it contains a cycle.

```
Richmond,IN-Muncie,IN Richmond,IN-Oxford,OH Richmond,IN-Anderson,IN
Oxford,OH-Richmond,IN
Muncie,IN-Chicago,IL Muncie,IN-Fort_Wayne,IN Muncie,IN-Dayton,OH Muncie,IN-Richmond,IN
Indianapolis,IN-Dayton,OH Indianapolis,IN-Anderson,IN
Dayton,OH-Anderson,IN Dayton,OH-Indianapolis,IN Dayton,OH-Muncie,IN
Anderson,IN-Richmond,IN Anderson,IN-Indianapolis,IN Anderson,IN-Dayton,OH
Bloomington,IN-Cincinnati,OH Bloomington,IN-Fort_Wayne,IN
Fort_Wayne,IN-Muncie,IN Fort_Wayne,IN-Chicago,IL Fort_Wayne,IN-Bloomington,IN
Fort_Wayne,IN-Cincinnati,OH
Chicago,IL-Muncie,IN Chicago,IL-Fort_Wayne,IN
Cincinnati,OH-Fort_Wayne,IN Cincinnati,OH-Bloomington,IN
```

Figure 1: graphSmall1.in

Note: ','s are part of the name of the city. The first edge is between "Anderson,IN" and "Richmond,IN".

Implementation

You are required to implement this program using an adequate set of abstract data types. An adequate set of ADTs will include:

- a **type** for graph nodes which includes all of the per-node data your algorithm requires
- a **type** which will represent edges in the adjacency list representation of the graph

The **graph** itself (the adjacency list) can simply be represented as an array of **graph nodes**. By including all the per-node data in the graph node data type, you will avoid the need to implement these attributes as arrays parallel to the adjacency list representation of the graph.

For **edges**, you will likely only need:

- an **initialization** function

- two **selector** functions
  - one returns the index of the node at the other end of the edge
  - another returns a pointer to the next edge in the list

For the **nodes** in the graph, provide the following:

- A **data field** used to store the name of the city
- **Selector** functions: return the value of each data field
- **Mutator** functions: modify each data field of a node
- A function **firstEdge**: returns a pointer/reference to the first edge in the list of adjacent nodes
- A function **addEdge**: adds a new edge to the adjacent list of the node

For the graph, provide the following:

- A function **searchCity**: given the name of a city, it searches the graph for a node with that name, adds a new node to the graph with that name if there is no such node, and (in any case) returns the index of the node

In order to verify that your ADTs and the input function are working correctly, you should implement a **test program** that reads the input file, builds the adjacency list for the graph and traverses the structure as an array of graph vertices, listing each node by name and index along with the indexes of vertices it is adjacent to. (See Figure 2.)

Input
The input will be in a file containing pairs of city names. Each pair will consist of a singlestring with the form:**city1-city2**

$ ./graphPrint graphSmall1.in

```
Read 10 cities and 13 edges
Node: 0, Name: Richmond,IN
Edge: 3
Edge: 2
Edge: 1
Node: 1, Name: Muncie,IN
Edge: 0
Edge: 6
Edge: 5
Edge: 4
Node: 2, Name: Oxford,OH
Edge: 0
Node: 3, Name: Anderson,IN
Edge: 6
Edge: 7
Edge: 0
Node: 4, Name: Chicago,IL
Edge: 5
Edge: 1
Node: 5, Name: Fort_Wayne,IN
```

```
Edge: 9
Edge: 8
Edge: 4
Edge: 1
Node: 6, Name: Dayton,OH
Edge: 1
Edge: 7
Edge: 3
Node: 7, Name: Indianapolis,IN
Edge: 3
Edge: 6
Node: 8, Name: Bloomington,IN
Edge: 5
Edge: 9
Node: 9, Name: Cincinnati,OH
Edge: 8
Edge: 5
```

Figure 2: Dump of an adjacency list for graphSmall1.in

Here **city** 1 and **city** 2 are arbitrary strings which do not contain whites pace or '-'. The pair is separated by a single '-'. The file will contain a potentially large number of these pairs with each **pair** being separated by whitespace (one or more spaces, tabs or newlines). Each edge will appear twice—once in each direction. Figure 1 shows one of the development files. The first row gives edges between **Richmond,IN** and each of **Muncie,IN**, **Oxford,OH** and **Anderson,IN**. Your program should build the graph by reading pairs from the input file until end-of-file is reached.

Development Files

The following **input files** are provided on Moodle for your testing:

- graphOne.in — an example graph from class with "cities" named "one", "two", etc.
- graphSmall1.in — the same graph with real city names.
- graphSmall3.in — a graph of 20 cities and 19 edges.
- graphMed2.in — a graph of 87 cities and 110 edges.
- graphLarge1.in — a graph of 99 cities and 282 edges.

And some example **output files** are also on Moodle:

- graphOne.out
- graphSmall3.out
- graphMed2.out
- graphLarge1.out

Submission:

Place all the files in a single folder and tar and compress it into Project2.tar.gz.

(1) Source code

1. The source code for each of your **abstract data types**.
2. The source code of your program.
3. **All source code must be clearly documented.** Your source code must be commented in a way that makes it clear where each of the slots of the skeleton occur and what code you have inserted there in instantiating it for your algorithms.
4. A typescript of a session in which you compile (with -Wall) your program and run it on the test input I supply.

(2) Report.pdf

1. A **list** of group members and the **responsibility** for each member
2. **Design** of the project (including the structure of the project)
3. State what have been **finished** and what haven't.
4. **Future** work.

Make sure that you submit all the items listed above.

(This document is based on Jim Rogers' project 3 for CS310 Algorithms and Data Structures.)