CSCI 310 – 02 (Fall 2019)

Programming Foundations

Lab #25: Finding articulation points

DUE: Wed, Dec 11, 11:59pm (turnin time)

Specifications

As we have discussed previously, the following problem is an instance of an *articulation point* (or *cut vertex*) problem:

> *UVA Online Judge 10199 (Tourist Guide)*
> Rio de Janeiro is a very beautiful city. But there are so many places to visit that sometimes you feel a bit lost. But dont worry, because your friend Bruno promised you to be your tourist guide. The problem is that he is not a very good driver, as he cant see very well (poor Bruno).
>
> Because of his disabilities, Bruno have a lot of fines to pay and he doesnt want to have even more to pay, even though he promised you to be your tourist guide. What he would like to know, however, is where are all the cameras that help the police to fine the bad drivers, so he can drive more carefully when passing by them.
>
> Those cameras are strategically distributed over the city, in locations that a driver must pass through in order to go from one zone of the city to another. In order words, if there are two city locations A and B such that to go from one to another (A to B or B to A) a driver must pass through a location C, then C will have a camera.
>
> For instance, suppose that we have 6 locations (A, B, C, D, E and F) and that we have 7 routes (all of them bidirectonal) B-C, A-B, C-A, D-C, D-E, E-F and F-C. Theres a camera on C because to go from A to E, for instance, you must pass through C. In this configuration, theres only one camera (on C).
>
> Your task is to help Bruno (as he wants to be a musician and he doesnt want to get even close of computers) and write a program which will tell him where are all the cameras, given the map of the city, so he can be your tourist guide and avoid further fines.

We referred to the articulation point solution provided by *GeeksForGeeks* as a reference to our discussions. You will continue to do so to find a solution to this problem using your `Graph` class. Since we are instantiating the DFS–based articulation points algorithm to a particular problem, you will need to define two functions:

1. `findCameras()` – initializes all auxiliary data structures needed for the DFS–based articulation points algorithm and then calls the `APutil()` operation (see below); returns the information on which locations (potentially) have cameras installed; and
2. `APutil()` – DFS–based articulation points algorithms (hence `AP` in the name of the function).

The functions above somewhat correspond to the `AP()` and the `APUtil()` functions in the articulation point solution provided by *GeeksForGeeks*.

The driver for the program, provided in **turnin**, will be the following:

```
#include "Graph.h"
#include "lab25.h"

// To allow comparison via second of pair in a map

struct Compare {
    bool value;
    Compare(bool val) : value(val) {}
};
bool operator==(const pair<string,bool>&p, const Compare& c) {
    return c.value == p.second;
}
bool operator==(const Compare& c, const pair<string,bool>&p) {
    return c.value == p.second;
}

```

```
// Solution to UVA Online Judge Problem #10199: Tourist Guide (modified)

int main()
{
    unsigned mapCount=0; // number of city maps read
    unsigned R; // number of routes between cities in a map
    string src, dst; // source and destination cities

    cin >> R;
    while( R>0 ) // route information available
    {
        mapCount++;
        string name="City map #"+('0'+mapCount);
        Graph< string > cityMap( name , false ); // undirected graph

        // Read in each route/edge
        for( unsigned i=0 ; i<R ; i++ )
        {
            cin >> src >> dst;
            cityMap.add( src , dst );
        }

        // Display the city map
//        cout << cityMap << endl;

        // Find the cameras (articulation points) and report results
        map<string,bool> isArticulationPoint=findCameras( cityMap );
        cout << "City map #" << mapCount << ": "
             << count( isArticulationPoint.begin() ,
                       isArticulationPoint.end() ,
                       Compare(true) )
             << " camera(s) found" << endl;
        set<string> location=cityMap.getVertices();
        for( auto loc: location )
            if( isArticulationPoint[loc] )
                cout << " " << loc << endl;
        cout << endl;

        // Read number of routes for next city map
        cin >> R;
    }

    return 0;
}
```

Input

All input comes from standard input. The input will consist on an arbitrary number of city maps. Each city map will begin with a non–negative number, $0 \leq R \leq 1000$, representing the total number of routes of the city. This is followed by R lines with the routes. Each route will be represented by the name of both places that the route connects (remember that the routes are bidirectional). Each location name will have at least one and at most 30 characters (all of them will be lowercase alphabetic letters). Location names in route descriptions will always be valid and there will be no route from one place to itself. You must read standard input until $R = 0$, and this input should not be processed.

Sample Input

```
7
ipanema copacabana
copacabana sugarloaf
ipanema sugarloaf
maracana lapa
sugarloaf maracana
corcovado sugarloaf
lapa corcovado

4
guanabarabay sambodromo
downtown sambodromo
sambodromo botanicgarden
colombo sambodromo

13
alpha charlie
alpha delta
bravo charlie
charlie delta
charlie foxtrot
echo bravo
echo foxtrot
foxtrot golf
hotel echo
india echo
juliett kilo
hotel juliett
india kilo
```

You may opt to use your overloaded `operator<<` defined on `Graph` objects to confirm you are reading each city map correctly. You may also consider displaying the values of auxiliary variables to debug your code.

Output

For each city map you must print the line:

```
City map #d: c camera(s) found
```

where d stands for the city map number (starting from 1) and c stands for the total number of cameras found. This is followed by c lines with the location names (in alphabetic order) where are each camera is found. You should print a blank line between output sets.

Sample Output

Here is the output for the Sample Input provided above:

```
City map #1: 1 camera(s) found
 sugarloaf

City map #2: 1 camera(s) found
 sambodromo

City map #3: 3 camera(s) found
 charlie
 echo
 foxtrot
```

The third case above is from the graph we considered when we went through the DFS–based articulation points algorithm. Here is a portion of the output for the third case above with all the auxiliary variables displayed (you can confirm the values from your notes):

```
    vertex   visited discovery       low    parent        AP
     alpha      true         1         1               false
     bravo      true         3         2   charlie     false
   charlie      true         2         1     alpha      true
     delta      true        11         1   charlie     false
      echo      true         4         2     bravo      true
   foxtrot      true         5         2      echo      true
      golf      true         6         6   foxtrot     false
     hotel      true         7         4      echo     false
     india      true        10         4      kilo     false
   juliett      true         8         4     hotel     false
      kilo      true         9         4   juliett     false

City map #3: 3 camera(s) found
 charlie
 echo
 foxtrot
```

Submission

Your submission will consist of the following file(s), submitted using the **turnin** facility.

- `lab25.h` – implementation of the `findCameras()` function (see line #43 above) and `APutil()` function described in the Specifications
- `Graph.h` – header file for your `Graph` class
- `Graph.cpp` – implementation file for your `Graph` class