# MAC0328 GRAPH ALGORITHMS: PROGRAMMING ASSIGNMENT 2

# BICONNECTED COMPONENTS

MARCEL K. DE CARLI SILVA AND THIAGO LIMA OLIVEIRA

Due Date: 04/Nov/2021 at 11:59pm

## 1. Introduction

In this programming assignment, your task is to write a program that labels the edges of an input graph according to its biconnected components.

This assignment will be graded out of 100 marks.

Your code **must** be written in `C++14` and use the BGL library. As before, you must use the BGL **only** for the data structures (and corresponding accessor functions) that store a graph and the attributes for its vertices and edges. The use of any **BGL algorithm** is forbidden.

## 2. Definitions

We briefly recall some (adapted) definitions from Lecture 14.

Let $G = (V, E)$ be graph. Define the binary relation $\overset{C}{\sim}$ on $E$ by setting $e \overset{C}{\sim} f$ if $e = f$ or ($e \neq f$ and there is a cycle in $G$ that traverses both $e$ and $f$).

**Theorem.** The relation $\overset{C}{\sim}$ on the edge set of a graph is an equivalence relation.

The equivalence classes of $\overset{C}{\sim}$ are called the *biconnected components*[1] of $G$.

Lecture 14 described many intermediate steps towards computing the biconnected components of a graph. Those ideas can be combined with adapted notions from Tarjan's algorithm for the strong components of a digraph to label each edge of an input graph according to the biconnected components to which it belongs.

## 3. Test Cases and Grading

Your program should solve each test case in $O(n + m)$ time, where $n$ is the number of vertices and $m$ is the number of edges of the input graph $G$.

Each test case has the following format:

- The first line has an integer $d \geq 0$ by itself, which indicates a *debugging level*. This shall be better explained in Section 5.
- The second line has two integers, $n$ and $m$, the numbers of vertices and edges, respectively, such that $1 \leq n \leq 10^5$ and $1 \leq m \leq 10^5$.

Instituto de Matemática e Estatística, Universidade de São Paulo, R. do Matão 1010, 05508-090, São Paulo, SP

*E-mail address*: `{mksilva,thilio}@ime.usp.br`.

*Date*: October 13, 2021, git commit `3be9743`.

[1]In Lecture 14, we defined the term biconnected components are the **non-singleton** equivalence classes of $\overset{C}{\sim}$. For the purposes of this programming assignment, we override that previous definition and consider **all** equivalence classes of $\overset{C}{\sim}$ to be biconnected components.

- The next $m$ lines have the description of the edges. Each edge is represented by two integers in $[n]$.

The existing driver/template code already handles reading the graph and calling a prescribed function, as explained in Section 4.

If $d = 0$, the driver prints biconnected component labels for each edge. If $d > 0$, the driver enters the corresponding debugging level and prints either the cut vertices or the bridges of $G$; refer to Section 5.

## 4. Submission and Implementation Details

In the Google Drive folder of the assignment you will find 5 template files, which you must use in this assignment. We now describe how the files interact and which ones you should modify.

There is a `Makefile`, which you should adapt to the location of the `boost` library in your computer. It is not recommended to modify the other flags, since they will be used when building the program for grading.

There are two template header files: `asgt.h` and `graph.h`.

The header file `graph.h` declares the `Graph` type using bundled vertex and edge properties. Some of these properties are `cutvertex` for each vertex, and `bcc` and `bridge` for each edge. You may add other fields and methods to these bundled classes for use by your algorithm, but the fields mentioned above must be kept unchanged. These are the only changes that you are allowed to make on the `graph.h` header.

The other header file, `asgt.h`, **must not be modified**. (In particular, you will not submit it, and we will use the file we distributed to build the program for grading.) This file contains the following prototype:

```
void compute_bcc (Graph& g, bool fill_cutvxs, bool fill_bridges);
```

This function declared by `asgt.h` must be defined in your `asgt.cpp` file, which you will submit. Calling this function should fill in the `bcc` field[2] of each edge of the parameter `Graph g`. (Examples of how the field may be accessed can be seen in the template `main.cpp` and `asgt.cpp` files.) The field `bcc` should be filled with integers in the set $[b]$, where $b$ is the number of biconnected components of the input graph $G = (V, E)$, in such a way that, for each $i \in [b]$, the set[3] $\{\, e \in E : \texttt{bcc}[e] = i\}$ is a biconnected component of $G$.

The grading driver `main.cpp` interacts with your program **only through the interface specified at `asgt.h`**. For grading purposes, `main.cpp` need not be the same as the one that we distributed.

You should submit a compressed archive `NUSP.tar.gz` through Moodle, obviously with `NUSP` replaced by your university ID number. The compressed archive must have precisely two files, inside no directory, namely `asgt.cpp` and `graph.h`.

| |
|---|
| Failure to follow these instructions exactly will be penalized. |

[2]The parameters `fill_cutvxs` and `fill_bridges` are explained in Section 5.

[3]In writing the next set using precise mathematical notation, we get into a bit of a pickle. Namely, the field `bcc` of an edge (descriptor) $e$ is accessed in code using the lvalue `g[e].bcc`, whereas our usual mathematical notation in course notes and the book [1] is $\texttt{bcc}[e]$. Since we are already using mathematical notation to define a set, we went with a mathematical version of accessing the `bcc` field.

## 5. Fallback Grading

To avoid having an "all or nothing" marking scheme, we will allow your program to enter some debugging state. Correctly implementing this debugging functionality is **optional**. The first element in the input description from Section 3, namely the integer $d \geq 0$, describes the desired *debugging level* of your program execution; thus $d = 0$ indicates no debugging at all. This debugging level is used to set (at most) one of the parameters `fill_cutvxs` and `fill_bridges` in the function call to `compute_bcc` described in Section 4. When $d = 0$, both arguments are set to `false`.

When running your code on some input test case, if the desired answer is incorrect, we will run your code with debugging levels 1 and 2 (independently of each other, that is, we try to run level 2 even if the output to level 1 is correct) in order to assign partial credit.

5.1. **Debugging Level 1: Cut vertices.** If $d = 1$, then the main program will call the function `compute_bcc` with the `fill_cutvxs` argument set to `true` (and the `fill_bridges` argument set to `false`). Then the boolean field `cutvertex` for each vertex must be filled correctly.

5.2. **Debugging Level 2: Bridges.** If $d = 2$, then the main program will call the function `compute_bcc` with the `fill_bridges` argument set to `true` (and the `fill_cutvxs` argument set to `false`). Then the boolean field `bridge` for each edge must be filled correctly.

## References


[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. 2nd edition. MIT Press, Cambridge, MA; McGraw-Hill Book Co., Boston, MA, 2001, pages xxii+1180 (cited on page 2).