# Lab 7 - *Identifying Bridges in Graphs*

***Due:** Wed, Mar 31st, 2021 @ 23:59*

Parallel and Sequential Data Structures and Algorithms 15-210 (Spring 2021)

## 1 Introduction

You have been recently hired as a researcher for BakedTuber, a Doha-based startup that specializes in delivering baked potatoes through pneuomatic tubes in the age of social distancing, using a much more elaborate version of a set-up like in Figure 1. Using their $30 billion in venture capital, they hired the best students from the nearby Cranberry Melon University in Qatar, who have excelled in CMUQ's legendary 15-210 course, you included. Your job at BakedTuber is to help write the graph algorithms that will let BakedTuber discover weaknesses in its pneumatic tube network to avoid in disruption of service to their hungry customers.

Figure 1. Pneumatic baked potato delivery network

In this lab you will be using DFS to solve different graph problems. The main coding section will involve using DFS to solve the "bridges" problem. Because this will be the first time that you work with graphs in 15-210, we highly advise you to start the lab *early* and seek out help at office hours *before* the due date.

## 2 Files

After downloading the assignment tarball from Autolab, extract the files by running:

```
tar -xvf bridgelab-handout.tgz
```

from a terminal window on a unix/linux system. Some of the files worth looking at are listed below. You should only modify the files denoted by *, as these will be the only ones that will be handed in terms of code

1. * `MkBridges.sml`
2. `Sandbox.sml`
3. `Tests.sml`

Additionally, you should create a file called:

```
written.pdf
```

which contains the answers to the written parts of the assignment. Your written answers should be composed in a suitable word processor such as LaTeX, Word, OpenOffice, etc. *Handwritten (and scanned in) answers will NOT be accepted.* **Make sure your name is also in the `written.pdf` file.**

## 3 Submission

To submit your assignment to Autolab in Diderot, generate a handin archive `handin.tgz` with the command

```
make package
```

You can then submit your `handin.tgz` file on Diderot. You should submit the answers to the written questions in *written.pdf* to Gradescope at `https://www.gradescope.com/courses/225303`.

Note that your code submitted to Autolab may not be graded until after the due date. It is your responsibility to test your code thoroughly before submitting.

### 3.1 Important points

1. Before submission, always, *always* verify that the files you will submit are indeed the versions that you intended to submit. Be very careful to NOT submit the wrong lab's `written.pdf`, blank files, or old versions of the files. **We will not entertain "*Oh no, I submitted the wrong file!*" emails or Diderot posts after the due date**. We will adopt a *zero-tolerance* approach to this, and you will receive the grade of your most recent submission (which may be subject to late submission penalties!) *You have been warned.*

2. If the homework has a written component, the answers to each written question in *written.pdf* should be on one page and each answer should fit on one page. If you decide not to answer one of the questions, still have the empty page in the file. This will make everyone's life easier in Gradescope.

3. If the written part has a question that asks you to show that your code has a certain complexity bound, **you should include a copy of your code in your answer and annotate it as suitable with relevant complexity bounds using comments and then argue or compute the resulting bounds. Answers to such question without a copy of the code you submitted will not be accepted.**

4. The questions below ask you to organize your solutions in a number of modules written almost from scratch. Your modules must be named exactly as stated in the handout. Correct code inside an incorrectly named structure, or a structure that does not ascribe to the specified signatures, *will not receive any credit*.

5. You may not modify any of the signatures or other library code that we give you. We will test your code against the signatures and libraries that we hand out, so if you modify the signatures your code will not compile and you will not receive credit.

6. This assignment requires the use of library code spread over several files. The compilation of this is orchestrated by the compilation manager through several `.cm` files. Instructions on how to use CM can be found in the file `cm.pdf` under Resources on Diderot.

7. Style grading for this assignment will be evaluated on a 5 point scale: -2 through +2. Style points will be added to your final grade from the rest of your homework. **You should review the full style guide available on the Diderot under Resources.**

8. **You should submit your solutions on or before the deadline on the top of the first page. Failure to do so will cause substantial deductions of your grade. Late submissions of up to a day will scale your homework grade by 70% (so if you were to get 100% as your grade,**

**you will now get 70%). Late submissions of up to 2 days will scale your homework grade by 50%. Beyond 2 days, you will receive 0 as your homework grade. This scaling will be handled outside Autolab.**

## 4 Bridges

While inspecting the network of tubes that BakedTuber uses to deliver potatoes, you discover that there are certain "critical" tubes that can disconnect a part of the network if broken. If one of these critical tubes goes down, a potato may be lost while in transit and the customer may write an angry review on BakedTuber's app, or worse yet ... switch to a BakedTuber's competitor, TuberExpress! Because BakedTuber is a fast-paced operation that can't afford downtime, it is absolutely essential that they can figure out which of the tubes are "critical".

### 4.1 Implementation

Let $G = (V, E)$ be some **unweighted, undirected**, simple (no self-loops; at most one edge between any two vertices) graph. $G$ is **not** necessarily connected. An edge $(u, v) \in E$ is a *bridge* if it is not contained in any cycles (equivalently, if a bridge is removed there will no longer be a path which connects its endpoints). Your task is to find all bridges in $G$.

**Task 4.1** (10 pts). In `MkBridges.sml`, define the type `ugraph` representing an undirected graph and implement the following function.

```
val makeGraph : edges -> ugraph
```

Given a sequence $E$ representing the edges of a graph $G$ as described above, `makeGraph` $E$ returns that same graph under your `ugraph` representation. For full credit, `makeGraph` must have $O(|E| \log |V|)$ work and $O(\log^2 |V|)$ span.

$E$ contains no duplicate elements; that is to say, for any two vertices $u$ and $v$, at most one of $\{(u, v), (v, u)\}$ is in the sequence $E$. Vertices are labeled from 0 to $|V| - 1$, where $|V|$ is the maximum vertex label in the edge sequence, plus one. You may assume that all vertices in the graph follow this convention, and that they all also have at least one neighbor.

**Task 4.2** (75 pts). In `MkBridges.sml`, implement the following function.

```
val findBridges : ugraph -> edges
```

Given an undirected graph $G$, `findBridges` $G$ returns a sequence containing exactly the edges which are bridges of $G$. For full credit, `findBridges` must have $O(|V| + |E|)$ work and span. The edges need not be ordered in any way, but for any edge $(u, v)$, at most one of $\{(u, v), (v, u)\}$ should appear in the output sequence. Our solution is around 40 lines with comments.

You should make use of the argument structure ascribing to `ST_SEQUENCE`, assuming the costs for `MkSTSequence` and `ArraySequence` in the library documentation.

*Hint:* you might want to make use of DFS numberings to solve this problem. Think about how we can use vertex numbers to determine if an edge is a bridge.

## 5 Testing

There are two ways to test your code.

1. In `Sandbox.sml`, write whatever testing code you'd like. You can then access the sandbox at the REPL:

   ```
   - CM.make "sandbox.cm";
   ...
   - open Sandbox;
   ```

2. In `Tests.sml`, add test cases according to the instructions given. Then run the autograder:

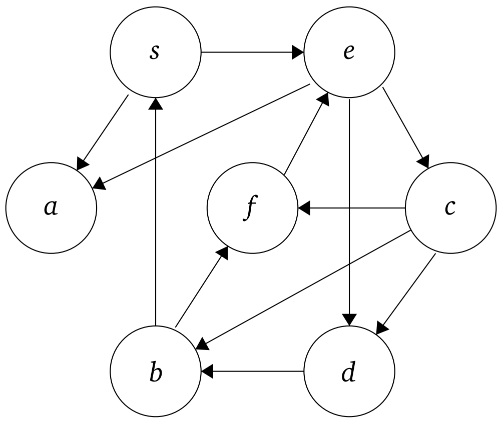
   ```
   - CM.make "autograder.cm";
   ...
   - Autograder.run ();
   ```

## 6 Written

Having solved the bridges problem, you helped Qatar Post identify which tubes in its network are critical and may require additional redundancy. The management of Qatar Post is grateful for your contribution to their operation and gives you a large bonus. Feeling satisfied with the work you have done at Qatar Post, you decide to go back to Cranberry Melon University in Qatar to do more work on graph algorithms.

### 6.1 Refresher

DFS numbering is a handy way to identify different types of edges in a graph. For the following two problems, refer to the graph below:



**Task 6.1** (10 pts). Draw the DFS tree for the graph, starting at node $s$. If there are multiple possible nodes to expand at a step, expand the nodes in alphabetical order. Also give the DFS numbering (discover time, finish time) for each node, for example $s : (0, 13)$. Include only the *tree edges* in your DFS tree.[1]

**Task 6.2** (5 pts). List all the forward edges, back edges, and cross edges in the graph.

For any directed graph $G$, we define its transpose, $G^T$, to be the graph that contains the same set of vertices with all of its edges reversed. That is, $(u, v)$ exists in $G \longleftrightarrow (v, u)$ exists in $G^T$. We also define $G^A$, the autotranspose graph of $G$, as follows: $V(G^A) = V(G), E(G^A) = E(G) \cup E(G^T)$.

**Task 6.3** (7 pts). Give an example of a graph $G$ with at most 4 vertices, such that there exists a *cross edge* $e = (u, v)$ in the DFS tree of $G$, but $e$ (not its reverse) is a *back edge* in the DFS tree of $G^A$. Label your vertices alphabetically from "a". The DFS tree should start from the node labelled "a". If there are multiple possible nodes to expand at a step, do so in alphabetical order. Label the edge $e$ clearly.

**Task 6.4** (8 pts). Give an example of a graph $H$ with at most 4 vertices, such that there exists a *tree edge* $e = (u, v)$ in the DFS tree of $H$, but $e$ is a *back edge* in the DFS tree of $H^A$. Label your vertices alphabetically from "a". The DFS tree should start from the node labelled "a". If there are multiple possible nodes to expand at a step, do so in alphabetical order. Label the edge $e$ clearly.

[1]Use `http://madebyevan.com/fsm/` to make your graphs look nice in LaTeX.

## 6.2 Connected Components

In an undirected graph, a connected component is a set of vertices such that for every pair vertices in the set, there exists a path between the two.

**Task 6.5** (5 pts). Give an algorithm that uses DFS to count the number of connected components in a undirected graph. You may give your answer in English or pseudocode. Your algorithm should run in $O(n+m)$ work and span.

## 6.3 Bipartite Graphs

Define a directed bipartite graph to be a graph where the vertices can be split into two sets $U$ and $V$ such that every edge has one endpoint in $U$ and the other endpoint in $V$. Specifically, for every directed edge $(x, y)$, either ($x \in U$ and $y \in V$) or ($x \in V$ and $y \in U$).

**Task 6.6** (5 pts). It turns out that it is possible to partition a bipartite graph into the two sets of nodes using BFS. Give a high level algorithm, in English or pseudocode, or SML, that uses BFS for to determine the two subsets of the vertices. You may assume that the graph is a strongly connected, directed, bipartite graph.

**Task 6.7** (5 pts). Analyze the work and span for *detecting* if a graph is bipartite using DFS with Single-Threaded Sequences. Give your answers in terms of $n = |V| andm = |E|$. For us to better understand your reasoning, you may want to give a high-level description of your algorithm in English or pseudocode, or SML.