Homework 5

Due by **12:30pm, Friday, October 14, 2016**.

Make sure you follow all the homework policies (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/policies/hw-policy.html).

All submissions should be done via Autolab (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/autolab.html).

Sample Problem

The Problem

Extend the topological ordering algorithm we saw in class so that, given an input directed graph G, it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG, or (b) a cycle in G, thus establishing that G is not a DAG.

The running time of your algorithm should be $O(m+n)$ for a directed graph with n nodes and m edges.

Click here for the Solution

Submission

You will **NOT** submit this question. This is for you to get into thinking more about designing algorithms on graphs.

Question 1 (Programming Assignment) [40 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

The Problem

We are given an undirected graph G with n nodes. The graph is formatted as an adjacency list, meaning that for each node, u, we can access all of u's neighbors. The goal is to output a single cycle from G. If there is no cycle, then return an empty list.

Note

Note that as we mentioned in class, a cycle in an undirected graph (as is the case in this problem) needs to have **at least** three nodes in it.

Input

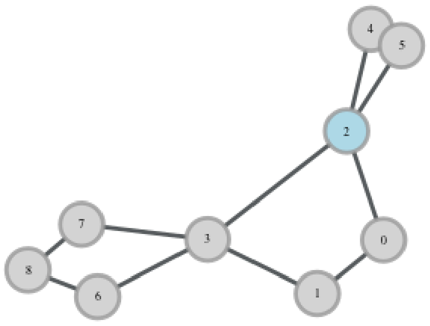
The input file is given as an adjacency list for graph G (we assume that the set of vertices is $\{0, 1, \dots, n-1\}$). The adjacency list assumes that the current node is the index of the line under consideration. For instance line 0 of the input file has the list of all nodes adjacent to node 0.

```
                u₁ u₄ u₆         <- All nodes that share edges with u₀
                u₃ uᵤ            <- All nodes that share edges with u₁
                u₀               <- All nodes that share edges with u₂
                .
                .
                .
                u₀ u₄ u₂ u₇      <- All nodes that share edges with uₙ₋₁
```

For example:

```
                1 2              <- Node 0 shares edges with nodes 1 & 2
                0 3              <- Node 1 shares edges with nodes 0 & 3
                0 3 4 5          <- Node 2 shares edges with nodes 0, 3, 4 & 5
                1 2 6 7          <- Node 3 shares edges with nodes 1, 2, 6 & 7
                2                <- Node 4 shares edges with node 2
                2                <- Node 5 shares an edge with node 2
                3 8              <- Node 6 shares an edge with nodes 3 & 8
                3 8              <- Node 7 shares edges with nodes 3 & 8
                6 7              <- Node 8 shares edges with nodes 6 & 7
```

If we draw the graph using the above input, it will look like this:



Output

The output is a list of nodes that will create a cycle. If our cycle is [u_0, u_1, ... u_m] where m is the legth of our cycle then we assume there exists an edge between every adjacent element in the list and that there is an edge between u_m and u_0.

```
                [u₀ u₁ u₂... uₘ]        <- u₀ shares an edge with uₘ & u₁
                                           u₁ shares an edge with u₀ & u₂
                                           u₂ shares an edge with u₁ & u₃
                                           .
                                           .
                                           .
                                           uₘ shares an edge with uₘ₋₁ & u₀
```

For example, using the example input from above:

```
                [0, 1, 3, 2]                <- Node 0 shares an edge with 1 & 3
                                               Node 1 shares an edge with 0 & 3
                                               Node 3 shares an edge with 1 & 2
                                               Node 2 shares an edge with 3 & 0
```

</> Note

There is **no** guarantee that the graph has only one connected component. Also the input graph might **not** have any cycle in it.

</> Note

It is possible that the input graph might have multiple cycles in which case the given outputs might not match yours (even if your algorithm is correct).

Hint

The best possible algorithm for this problem that we are aware of runs in time $O(m + n)$ where m in the total number of edges, and n is the total number of nodes.

! Note

Both the input and output parsers in each of the three languages are already written for you.
Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file).

! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Java Python C++

Download Java Skeleton Code (HW5Java.zip)

Directory Structure

```
├── Driver.java
├── HW5_Student.java
├── testcases/
│   ├── input1.txt
│   ├── input2.txt
│   ├── input5.txt
│   ├── output1.txt
│   ├── output2.txt
│   └── output5.txt
```

You are given two coding files: `Driver.java` and `HW5_Student.java`. `Driver.java` takes the input file, parses it and creates an instance of the class and prints result in output file. You only need to update the `HW5_Student.java` file. You may write your own helper methods and data structures in it.

The testcases folder has 3 input files and their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding. The output files provided are not the ONLY solutions. They only represent an example of a cycle from each of the inputs.

Method you need to write:

```
/**
 * This method must be filled in by you. You may add other methods and subclasses
 * but they must remain within the HW5_Student_Solution class.
 * @return A list of integers corresponding the unique nodes ids for every node i
 */
public ArrayList<Integer> outputDistances() {
  return null;
}
```

The `HW5_Student` class has 1 instance variables:

- `graph` which is of type `HashMap<Integer, ArrayList<Integer>>` where the key is the node id and the value is an arraylist of adjacent nodes.

The output is an arraylist of `int` s. In particular, the list of unique node ids such that these nodes create a cycle. If there is no cycle, output an empty arraylist.

Compiling and executing from command line:

Assuming you're in the same directory level as `Driver.java`. Run `javac Driver.java` to compile.

To execute your code on `input1.txt`, run `java Driver testcases/input1.txt`. The output array will be printed to `stdout`.

Submission

You only need to submit `HW5_Student.java` to Autolab.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

Question 2 (Helping out Kiran) [45 points]

The Problem

Kiran Rudra has recently been expressing interest in bugs especially moths and butterflies. On a recent field trip, Kiran collected n specimens and wants to identify each specimen as either a butterfly or a moth. Kiran is new to studying butterflies and moths so she cannot do this directly.

However, she can do the following: given a pair of specimen (i, j) for $i \neq j$, she can judge them as `same` (meaning she believes that both are either moths or both are butterflies) or `different` (meaning she believes exactly one of the two specimens is a moth and the other is a butterfly) or `ambiguous` (meaning she is not sure and does not want to judge them as same or different). Assume that there are m pairs of specimen that she can label as same or different (of course $m \leq \binom{n}{2}$). Kiran is too young to take the algorithms course so she comes to you for help: she wants to know whether her judgments are `consistent` : i.e. is there a labeling of the n specimen as either butterfly or moth that is consistent with her judgments and if you can design an algorithm for him that she can run to check whether her judgments are consistent.

More precisely, the input to your algorithm is the set of n specimen and m definitive judgments (a definitive judgment is a pair of specimen $i \neq j$ such that (i, j) is either judged as `same` or `different`)-- the pairs that are not definitive judgments are `ambigious` . Your algorithm must decide if there exists a labeling of the n specimen (i.e. each specimen is labeled as either `moth` or `butterfly`) that is consistent: i.e. for each definitive judgment (i, j) that says `same` , the labels of i and j must match and for each definitive judgment (i, j) that says `different` , the labels of i and j are different. (For a pair (i, j) that has a judgment of `ambiguous` , the labeling can assign i and j in any arbitrary manner.)

Kiran is still young so to make sure she does not get frustrated, design a linear $O(m + n)$ time algorithm that solves the above problem.

Hint

It *might* be useful to represent the input as a graph and run one of the connectivity algorithms we have seen in class. Also in one way of representing the "relationships," there might be two kinds of edges: i.e., it might be useful to "label" the edges. As usual, feel free to ignore this hint and come up with an algorithm from scratch.

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible. Also make sure that you preview your upload on Autolab to make sure it was uploaded correctly.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

1. `Algorithm idea`: `9` points.
2. `Algorithm details`: `9` points.
3. `Proof of correctness idea`: `9` points for arguing correctness of the algorithm.
4. `Proof details`: `9` points.
5. `Runtime analysis`: `9` points, i.e. show why it runs in $O(m + n)$ time.

! Note

If you do not have labeled and separated out proof idea, algorithm idea, proof details, algorithm details and runtime analysis you will get a zero(0) irrespective of the technical correctness of your solution.

Questions 3 (Listing Triangles) [15 points]

The Problem

A `triangle` in a graph $G = (V, E)$ is a 3-cycle; i.e. a triplet of vertices (u, v, w) such that $(u, v), (v, w), (u, w) \in E$. (Note that G is undirected.) In this problem you will design a series of algorithms that given a *connected* graph G as input, lists **all** the triangles in G. (It is fine to list one triangle more than once.) We call this the `triangle listing problem` (duh!). You can assume that as input you are given G in *both* the adjacency matrix and adjacency list format. *For this problem you can also assume that G is connected.*

1. Present an $O(n^3)$ time algorithm to solve the triangle listing problem.

Note

The trivial algorithm works. For this part only a correct algorithm idea is needed.

2. Let Δ denote the maximum degree, i.e.

$$\Delta = \max_u n_u.$$

Present an $O(n \cdot \min(m, \Delta^2))$ time algorithm to solve the triangle listing problem.

Hint

It might be useful to consider all the potential triangles that have one fixed vertex $u \in V$ as one of its vertices.

Note

Both algorithm idea and details are needed. Only proof idea (for correctness of the algorithm) is needed: no proof details are required. Runtime analysis is also needed.

3. Present an $O(m^{3/2})$ algorithm to solve the triangle listing problem.

Note

Only a valid algorithm idea and runtime analysis is required.

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible. Also make sure that you preview your upload on Autolab to make sure it was uploaded correctly.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

- Part 1:
  1. `Algorithm idea` : `2` points.
- Part 2:
  1. `Algorithm idea` : `2` points.
  2. `Algorithm details` : `2` points.
  3. `Proof idea` : `2` points.
  4. `Runtime analysis` : `2` points.
- Part 3:
  1. `Algorithm idea` : `2` points.
  2. `Runtime analysis` : `3` points.

! Note

If you do not have labeled separated out parts as mentioned in the grading rubric above, you will get a zero(0) irrespective of the technical correctness of your solution.