# Oracle

**Due:** 11:59pm Saturday, February 1, 2020

## Stencil Files

We have provided stencil files for this assignment. They can be found in `/course/cs1950y/pub/oracle`

## But first! Collaboration Policy

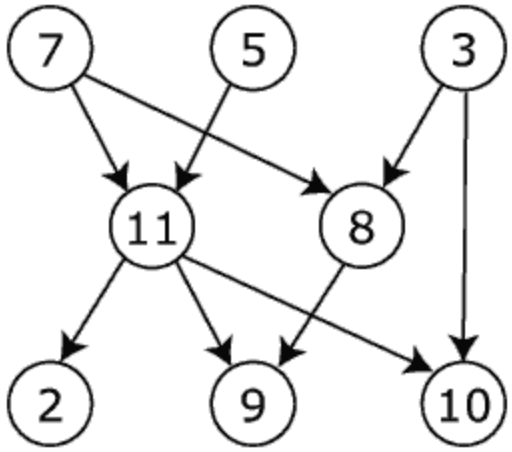
Before starting this assignment, please read through the course's collaboration policy and complete this quiz.

## Part 1 - Background Information

### Simple Directed Acyclic Graphs

In this assignment, we will be working with directed acyclic graph (DAG).

A directed graph contains edges that run from one node to another (like a one way street). An acyclic graph does not allow the edges in the graph to create a cycle (cannot reach a vertex from itself). Here is an example of a DAG:




### Topological Sort

Given this simple, directed acyclic graph `G = (V, E)`, where `V` are the vertices of the graph and `E` is the edges in the graph, a topological sort finds a linear ordering of vertices, such that for all edges `(a, b)` in `E`, `a` precedes `b` in the ordering.

An example of a topological sort is to determine the order in which a student can take classes given a list of prerequisites. Another example is the order of formula cell evaluation when recomputing formula values in spreadsheets is an application of topological sort. A concrete

example is in Part 3 of this assignment. Depending on the graph, **many valid topological sorts may exist** for each of these problems.

## Part 2 - Assignment Specifications

You will write two programs for this assignment: a topological sorting program (named `topsort`) and an oracle (named `oracle`).
Implement the topological sorting program using the following pseudocode:

```
function topsort(g): // g = directed acyclic graph
    L = empty list to store ordering
    S = set of vertices with no incoming edges
    while S is not empty:
        u = vertex removed from S
        append u to L
        for each vertex v where edge e = (u,v) in g:
            remove e from g
            if v has no other incoming edges in g:
                insert v in S
    return L
```

Your program should adhere to the input-output specification in Part 3 of this assignment. Being confident in your software's correctness involves more than just writing code and some unit tests. Many software companies (and computer scientists!) use automated testing to strengthen confidence. For this assignment, you will build an **automated testing oracle** for topological sorting programs.

Your oracle will consume a topological sorting program and the number of inputs to generate, `n`. Then, your oracle will generate `n` valid inputs and pass them to the provided program. Your oracle will check that each output is in fact a topological sort of the corresponding input and conforms to the specifications in Part 3 of the assignment.

If the provided program always produces a valid topological sort (up to some value of "always"), then your oracle should return `True`. Otherwise, your program should return `False`. The oracle should not write anything to standard output or standard error. Note that you may assume `n` is valid (a non-negative integer) and all input is well-formed.

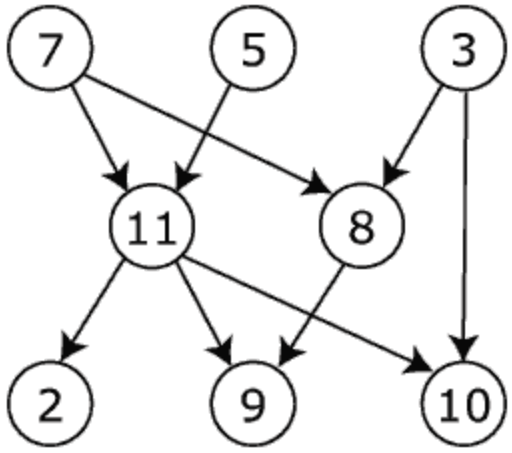
Using your oracle, check that your implementation is correct.

### Which language do I use?

This assignment can be completed in either Python or Java. If you are unsure of which one to use, we recommend Python. This online tutorial is a good starting place to become familiar with the language. There is also an optional available on the course website.

## Part 3 - Input-Output Specification

Consider the following directed acyclic graph (DAG):




One possible valid topological sorting for the above DAG is `[7, 5, 11, 3, 10, 8, 2, 9]` (there may be other valid orderings, though).

We will represent a DAG as a list of edges, where each edge is a tuple of two vertices (each vertex is just a string). In Python, this corresponds to the type `List[Tuple[str]]`. In Java, this is `List<String[]>`. Your topsort function should return a list of strings representing a valid topological sorting of the inputted DAG (in Python this is `List[str]` and in Java this is `List<String>`).

For the oracle, you should implement three functions: `generate_input`, `is_valid`, and `oracle` (in Java, they are named `generateInput`, `isValid`, and `oracle`). The `generate_input` function should take in an integer and return that many randomly generated DAGs to test your topsort function with.

The `is_valid` function should take in a DAG and an ordering of vertices and return true if the ordering of vertices is a valid topological sorting of the given DAG.

The `oracle` function should take in a topsort function (Topsort object in Java) and an integer and return true if the given function is a valid topological sort and false otherwise. Note that your oracle function should use `is_valid` and `generate_input` to determine if the program is a valid topsort function.

**Please make sure you adhere to the type signatures we provide for the functions**, as we will be checking all components of this assignment separately.

While we don't care about efficiency, in order for us to auto-grade your work, both of your topsort and oracle should not take more than 30 seconds to run. The input graph that we will be testing on your topsort will have no more than 200 vertices. When we test your oracle, `n` will be no more than 20.

## Part 4 - Testing

We expect you to test the functions that you write. In Python, you can do this using `assert` in Python or Java. Please make sure to write these tests in the spots we provide so as to not interfere with the autograder.

To test your oracle, it might be a good idea to write a few (subtly) incorrect topsort programs to see if you oracle is able to classify them as invalid topsorts.

If you are using Python, make sure that you are using version 3.7. Also please make sure that if you used the `pdb` library, that it is not present in your final submission as this causes problems with the autograder.

## Part 5 - Handing In

Run `cs1950y_handin oracle` from a directory holding your topological sorting program (named `topsort.py` or `Topsort.java`) and oracle (named `oracle.py` or `Oracle.java`), and any additional files you wish to submit. You should receive an email confirming that your handin has worked.

If you are not registered, you will not be able to run the above handin script. Please zip your files, name it `oracle.zip`, and send it to us at `cs1950ytas@lists.brown.edu`. Please only do so if you cannot register for the course by the due date of the assignment.