CSCC24 & CSC324 Winter 2018 – Assignment 2
Due: Tuesday, February 26, midnight
This assignment is worth 10% of the course grade.

In this assignment, the problem is widely conjectured to be computationally intractible, so some brute-force search and backtracking is expected. Nonetheless, using lazy evaluation suitably, you can still have a high-level expression of the search, and it can still run in polynomial space; and using immutable data structures suitably, rolling back to "previous states" as required by backtracking can be both trivial to code and efficient to run. As usual, you should also aim for reasonably efficient algorithms and reasonably good and simple coding style.

The *Edge Geography Game* is played according to these rules:

- A simple directed graph G is given. There are two players taking turns to make their moves.

  The "simple" part means that there is no edge going from a vertex to itself (self loop).

- The first player's first move picks a vertex to visit. Afterwards (it will then be the second player's turn, then the first player's turn, etc.):

- In each player's turn, he/she makes a move by starting from the most recently visited vertex (call it v_i) of the opponent, choosing an unused edge going out of v_i (say, it goes from v_i to v_{i+1}), and using it to visit v_{i+1}.

  In the future, no one may use this edge again.

  Note that the two edges (u, v) and (v, u) are considered different in the context of directed graphs. Using (u, v) does not imply using (v, u).

- Put it another way, the players take turns to build an Eulerian walk—while each vertex may be visited as many times as you need, each edge may be followed at most once.

- Winning condition: The opponent is left with no unused edge to use.

The name of this game comes from two points:

- People usually like to build the graph using the English names of geographical places for vertices, and there is an edge from one name to another iff the first name ends in the same letter as the second name starts in. This is the "geography" part.

- The original Geography Game forbids re-visiting a vertex. The Edge Geography Game forbids re-visiting an edge, and allows re-visiting a vertex.

This question is about how to ensure that the first player wins (assuming that neither player makes any mistake) by choosing the first vertex carefully.

At any game state, the player P who makes the next move *has a winning strategy* iff there exists a next state s_1 that P may move the game into such that for all next-next states s_2 that the opponent may move the game into after s_1, P has a winning strategy.

Equivalently, at any game state, the player Q who made the last move *is destined to win* iff for all next states s_1 the opponent may move the game into, there exists a next-next state s_2 that Q may move the game into after s_1 such that Q is destined to win.

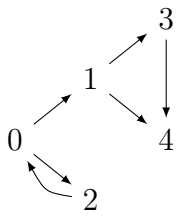
Note that these two definitions are:

- recursive (deliberately);
- terminating, since the game keeps removing choices for moves, and with a finite graph, we will hit one of:
  - "for all s in the empty set, ...", which is true (exercise: why?)
  - "there exists s in the empty set such that ...", which is false (exercise: why?)

  no matter what's in the ellipsis.

Your job is to implement an algorithm to compute the choices for the first vertex so the first player is destined to win; equivalently, the first moves of the winning strategies for the first player.

Example: Suppose the graph is:



Then the first player should choose 0, or 2, or 4 for the first vertex, but not the others.

Let's walk through why 0: The opponent will then move to 1 or 2, but either way the first player will win:

- If 1, then the first player can move to 4 and win. (Don't move to 3.)
- If 2, then the first player can move back to 0. Now the opponent must move to 1 (cannot re-use the edge from 0 to 2), and then the first player can move to 4 and win.

Here is why the first player should not choose 1 for the first vertex: Then the opponent could move to 4 and win.

Suggestions:

- The input format for the graph is patently inefficient to work on. Convert it to a suitable, efficient representation at the very beginning.

- Model the game state suitably and efficiently.
- Have a function that takes a game state and generates a lazy list of legal next states.
- Have a function that tests a game state for either "the player who makes the next move has a winning strategy" or "the player who made the last move is destined to win" (your choice).
- You can always extract first vertices from first states later.

Speed expectation: Although some brute-force search is unavoidable, there are still reasonable ways and unreasonable ways to go about it, e.g., inefficient representations of the graph and game states are unreasonable. To test against these, there will be fairly large test cases that each take my solution up to a few seconds, and your solution will be given only twice as much time.