Assignment 4 – Chocolate Couriers

All submitted work must be *done individually* without consulting someone else's solutions in accordance with the University's Academic Dishonesty and Plagiarism policies.

Story

<A screen blinks to life showing a Belgian chocolate shop.>

Good evening, Mr. Hunt.

Our chocolate shop is seeking to expand its operations and we need to share our top secret recipes with our next location. For this we have worked long and hard to set up a network of trusted couriers. However, our Swiss competitors are keen to discover the secret to our success and are trying to convince some couriers to let them take a peek at our recipes.

<The screen changes to show a map of red and blue dots connected by line segments.>

We have marked our couriers on this map. The blue vertices are couriers we know we can trust, but the red ones are the ones we aren't sure about. The edges show which couriers can communicate with which other couriers.

Your mission, if you choose to accept, is to:

- Maintain our network of couriers by supporting operations to add new couriers, remove existing ones, and add and remove edges between them.
- After your colleagues investigate a courier, we may want to update whether they're trusted, so you should support this.
- Our main concern is finding a way to transport our recipe from a vertex s to a vertex t by going via at most one untrusted courier. Our encryption should be strong enough to handle one potentially compromised courier, but we aren't sure it'll handle more than that.
- We also want to check how secure our network is. We measure this by checking whether any edges connecting an untrusted and a trusted courier exist, such that the removal of a single one of these edges means that every path between s and t now needs to use at least one edge connecting two untrusted couriers.

Good luck, Mr. Hunt. The fate of the chocolate-loving world is in your hands.

This message will self-destruct in 5 seconds…

5…

4…

3…

2…

1...

<A puff of smoke comes out of a nearby device, but by that time we're long gone.>

Your Task

Maintain a **graph** where each vertex stores a boolean value `is_trusted` indicating whether it is a trusted courier. You can assume that the initial graph is undirected and simple and your operations should ensure it stays that way.

Send Message

You must also support the `can_send_message(s, t)` function, which returns a path from s to t that includes at most 1 untrusted vertex if such a path exists, and none otherwise. You can assume that s and t are distinct trusted vertices.

Example:

`A(trusted) - B(untrusted) - C(trusted) - D(untrusted) - E(trusted)`

`can_send_message(A, C)` returns [A, B, C]

`can_send_message(A, E)` returns `none`

Check Security

Your graph must support the `check_security(s,t)` operation that returns a set of edges. We define an untrusted edge as an edge connecting two untrusted vertices. A semi-trusted edge is an edge connecting a trusted to an untrusted vertex. When called, the `check_security(s,t)` operation returns the set S of semi-trusted edges such that removing any edge in S from the graph forces any s-t path to traverse an untrusted edge. You can assume that s and t are distinct trusted vertices.

Example:

```
 A(trusted) -  B(untrusted)
     |             |
 C(trusted)    D(untrusted)
     |             |
E(untrusted) - F(trusted)
```

`check_security(A,F)` returns {(C,E), (E,F)}, since only removing either of those edges forces every path between A and F to take at least one untrusted edge.

You should strive to make your implementation as efficient as possible. Note that while we don't explicitly test for the efficiency of your implementation, *using inefficient implementations may cause timeouts on Ed.*

TO IMPLEMENT:

You will need to implement these functions:

`vertex.py`

- `add_edge, remove_edge`
- `update_status`

`graph.py`

- `add_vertex, remove_vertex`
- `add_edge, remove_edge`
- `send_message`
- `check_security`

(Note, you can add additional functions and variables to the classes if required, so feel free to modify and extend those as long as you leave the existing function signatures and variables intact.)

Code

`vertex.py`

This file holds all information about the vertex in the graph.

Properties

- `is_trusted - boolean` Indicates whether the vertex is trusted or untrusted.
- `edges - List[Vertex]` The list of vertices connected to this vertex, forming the edges of the graph.

Functions

`add_edge(vertex)`

- Adds an edge to the vertex, does nothing if this would result in a graph that isn't simple.

`remove_edge(vertex)`

- Removes the edge to the vertex, if it exists.

`get_edges() (OR vertex.edges)`

- Returns the list of edges.

`update_status(is_trusted)`

- Updates `is_trusted`.

`get_is_trusted() (OR vertex.is_trusted)`

- Returns whether the vertex is trusted.

`graph.py`

The main graph file, holds all interaction with graphs and vertices.

Properties

- `vertices - List[Vertex]` The vertices of the graph.

Functions

`add_vertex(vertex)`

- Adds the vertex to the graph.

`remove_vertex(vertex)`

- Removes the vertex from the graph.

`add_edge(vertex_A, vertex_B)`

- Adds and edge between vertex A and vertex B, unless the resulting graph wouldn't be simple.

`remove_edge(vertex_A, vertex_B)`

- Removes the edge between vertex A and vertex B, if it exists.

`send_message(s, t)`

- Returns a path from s to t that contains at most one non-trusted vertex, if such a path exists. If no such path exists, return none.
- You can assume that s and t are distinct trusted vertices.

`check_security(s, t)`

- Returns the set S of semi-trusted edges such that removing any single edge (u,v) in S from the graph forces any s-t path to go via an untrusted edge.
- You can assume that s and t are distinct trusted vertices.

Important Information

- Be careful with `copy` and `deepcopy` as this may affect checks in the testcases. Please always return unmodified vertices when asked for a path.
- The list of vertices that form a path is ordered. Its first element should be s and its last should be t, with every pair of consecutive elements connected by an edge.

Marking

You will be marked using a range of public and hidden tests. There won’t be additional tests added after the due date.

All tests are between 1 to 3 marks each.