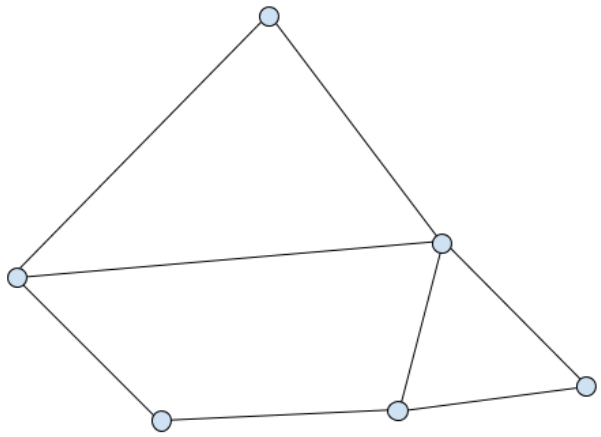
Higharc Computational Geometry Engineer Challenge

Thank you for taking the time to complete Higharc's Computational Geometry Engineering Challenge. We hope you can use this as an opportunity to showcase your technical skills.

Background

You're given a set of vertices connected by edges in the plane. The edges do not intersect except at their endpoints. Together, they form a collection of closed polygons. The edges form a connected component. Here's an example with 6 vertices and 8 edges:



This data structure is provided in the form of a collection of vertex positions and an edge set. Here's an example that defines a 2x2 rectangle with a diagonal interior edge:

```
{
  "vertices": [[0, 0], [2, 0], [2, 2], [0, 2]],
  "edges": [[0, 1], [1, 2], [0, 2], [0, 3], [2, 3]]
}
```

Basic requirements

1. *Algorithm 1:* Write an algorithm that finds all of the interior faces (polygons) of such a data structure. The output of the algorithm is up to you. Include tests (with text descriptions of the input data) demonstrating that it works. Comment your code with specifics about the computational complexity of your implementation.

2. *Algorithm 2:* Write an algorithm that processes the output of Algorithm 1 in order to find the neighboring faces of any face. That is, faces that share an edge with the query face. It should take the output of Algorithm 1 as input, unique identifier for the face and output an array of face identifiers. The face identifiers might be an integer or string. Include tests (with text descriptions of the input data) demonstrating that it works. Comment your code with specifics about the computational complexity of your implementation.

3. Create a simple HTML page that presents the output of algorithm 1. You should provide a way to import JSON in the vertex-edge format described above. Your page should then process and present the output. We will use this to test your project. You're allowed to use any browser API including SVG, Canvas, or WebGL. The page should display the faces using unique colors. This part of the challenge is simply to display that you've completed the project.

Commit all of this code to a public Github repository with instructions on how to pull and run the code. Also, provide a website endpoint where your code is deployed. All of the code should be implemented in JavaScript or TypeScript. You should not use any libraries, only browser API's.

Advanced requirements

If you find that you have extra time, pursue the Advanced requirements, don't embellish the UI.

1. *Algorithm 3:* Given a point and the output of Algorithm 1, find the face the point is contained within. Naturally, the point may not be inside of a face. Include tests (with text descriptions of the input data) demonstrating that it works. Comment your code with specifics about the computational complexity of your implementation.

2. *Algorithm 4:* Implement an algorithm that, given a start face, computes the neighboring faces, and then the neighbors of the neighbor faces and so on. The system should find "layers" of neighboring face sets. The output should be an array of arrays of face ids until

all faces have been visited. Include tests (with text descriptions of the input data) demonstrating that it works.