Connecting Vertices with Edges

12 min

Since we can add vertices to our graph, we should be able to connect them together. We want to provide this functionality in the Graph class to add a layer of abstraction that will simplify adding edges, similar to how we abstracted vertex creation. This is where our Edge class in **Edge.js** will come in handy. Go ahead and take a look at the class.

The start and end properties mark the vertices that the edge connects. If the graph is directed, we can indicate the direction the edge points (towards the end vertex).

We will create an .addEdge()

Preview: Docs Loading link description

method

in the Vertex class that connects the vertices together by creating an Edge and adding it to the vertices' list of edges. When the Edge is created, it expects the two Vertex instances, which is how the Edge tracks the connection between the two vertices .

Then, we will use this method in the Graph's .addEdge() method to create edges going in both directions between the two given vertices. Even though this graph is undirected, we want to create two edges going in both directions so it is easier to traverse.

Instructions

1. Checkpoint 1 Passed

1.

In our Vertex class, create the .addEdge() method that expects a vertex parameter, which will represent the other end of the edge. It must be an instanceof a Vertex, otherwise we should throw an error. Then, create an Edge instance to represent the connection from this vertex to the ending vertex.

Add the Edge instance to the vertex's list of edges to open up our first connection from one vertex to another.

Hint

To verify that we can successfully create an edge between two vertices, create two vertices under the Vertex class with any value. Add an edge from the first vertex to the second vertex by calling .addEdge() on the first vertex (the vertex at the start of the edge), and giving it the second vertex (the vertex at the end of the edge).

Print out the resulting connection by calling the .print() method on the first vertex. We should see that the first vertex points to the second vertex. You can also call the .print() method on the second vertex to verify that it has no connections to the first vertex.

2. Checkpoint 2 Passed

2.

We're ready to connect vertices with edges through our Graph class. In the Graph class, create an .addEdge() method, which will create edges between the
parameters, vertexOne and vertexTwo.

If the parameters are both an instanceof a Vertex, use the vertices' .addEdge() method to create an edge between the other vertex. Remember to add edges between both vertices.

Otherwise, throw an error if either of them are not.

Hint

By using the vertices' .addEdge() method to create an edge with another vertex, the graph user avoids the need to understand how the Vertex class interacts with the Edge class.

Remember that if an error is thrown, we should not create the edges between the two vertices. To avoid creating edges when both vertices are not Vertex instances, you can put the edge creation inside an if block, and throw the error inside an else block.

3. Checkpoint 3 Passed

3.

Let's verify that we can successfully create an edge between two vertices through
the Graph class. Under the Graph class, there are
two Vertex instances: atlantaStation and newYorkStation.

Before the trainNetwork is printed, use the trainNetwork's .addEdge() method to create an edge between the two vertices. We should see Atlanta connect to New York, and New York connect to Atlanta.

Vertex.js

```
const Edge = require('./Edge.js');


class Vertex {
 constructor(data) {
  this.data = data;
  this.edges = [];
 }


 addEdge(vertex) {
```

```
    if (!(vertex instanceof Vertex)) {
      throw new Error('Parameter must be a Vertex instance');
    }
    const edge = new Edge(this, vertex);

    this.edges.push(edge);
  }

  print() {
    const edgeList = this.edges.map(edge =>
        edge.weight !== null ? `${edge.end.data} (${edge.weight})` : edge.end.data);

    const output = `${this.data} --> ${edgeList.join(', ')}`;
    console.log(output);
  }
}

const vertex1 = new Vertex(1);
const vertex2 = new Vertex(2);

vertex1.addEdge(vertex2);
vertex1.print();
vertex2.print();

module.exports = Vertex;
```

Graph.js

```
const Edge = require('./Edge.js');
const Vertex = require('./Vertex.js');

class Graph {
  constructor() {
    this.vertices = [];
  }

  addVertex(data) {
    const newVertex = new Vertex(data);
    this.vertices.push(newVertex);

    return newVertex;
  }

  removeVertex(vertex) {
    this.vertices = this.vertices.filter(v => v !== vertex);
  }

  addEdge(vertexOne, vertexTwo) {
  // Check if both parameters are instances of Vertex
  if (!(vertexOne instanceof Vertex) || !(vertexTwo instanceof Vertex)) {
    throw new Error('Both parameters must be instances of Vertex.');
  }

  // Create edges in both directions
  vertexOne.addEdge(vertexTwo);
  vertexTwo.addEdge(vertexOne);
```

```
}

  print() {
    this.vertices.forEach(vertex => vertex.print());
  }
}

const trainNetwork = new Graph();
const atlantaStation = trainNetwork.addVertex('Atlanta');
const newYorkStation = trainNetwork.addVertex('New York');

trainNetwork.addEdge(atlantaStation, newYorkStation);

trainNetwork.print();

module.exports = Graph
```

Edge.js

```
class Edge {
  constructor(start, end, weight = null) {
    this.start = start;
    this.end = end;
    this.weight = weight;
  }
}

module.exports = Edge;
```