Removing Vertices

6 min

We also want our Graph to manage its own vertex removal, just like how it handles its own vertex creation.

We will use the .removeVertex()

Preview: Docs A method is a small piece of code, usually defined in a class, that can be used outside the class and in other parts of the program.

method

to look for the requested vertex and remove it from the list of vertices.

Instructions

1. Checkpoint 1 Passed

1.

Inside the Graph class, implement the .removeVertex() method that accepts the vertex to be removed as a parameter. Iterate through the list of vertices and remove the vertex that is strictly equal to the vertex given in the parameter.

Hint

To remove the requested vertex, you can use the filter iterator to only keep all the vertices except the requested one. Finding the requested vertex's index with findIndex, and then using splice at the vertex's index will also work.

Remember, equality with objects checks if the instance in the argument is **exactly** the instance in the array of vertices. When comparing objects, equality will check if they are located at the same memory address. This approach ensures that we are specifically removing the requested vertex, which comes in handy if there are two vertices with the same value and edge connections.

2. Checkpoint 2 Passed

2.

Underneath our Graph class, let's remove the Atlanta vertex we added in the previous exercise using the trainNetwork's .removeVertex() method. Remember to do it before the call to .print() so we can see what the resulting graph looks like.

We should see our graph with only the New York vertex remaining, and no edges.

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) {
    const index = this.vertices.findIndex(v => v === vertex);

    if (index !== -1) {
      this.vertices.splice(index, 1);
    }
  }

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

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

trainNetwork.removeVertex(atlantaStation);

trainNetwork.print();

module.exports = Graph;
```

Vertex.js

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

  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);
  }
}

module.exports = Vertex;
```

Edge.js

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

module.exports = Edge;
```