Weighted Graphs

8 min

The current implementation of our Graph class is unweighted, where there is no cost associated with the edge that connects the vertices together. Since we want our Graph to be flexible, we should give the option for weights to be added to the edge when a new edge is created.

Instructions

1. Checkpoint 1 Passed

1.

In the Graph class, add an isWeighted boolean parameter in the constructor for the user to designate that the graph is weighted. It should default to false if no argument is given.

Set the argument to the isWeighted class property.

2. Checkpoint 2 Passed

2.

In the Vertex class, add a second parameter for weight in the .addEdge() method. Pass the argument to the new Edge instance that will be created.

Hint

To check that the weight of the edge is set correctly, create two Vertex instances.

Call the first vertex's .addEdge() method and pass in the second vertex as well as a number for the weight. This is the weighted edge.

Then call the second vertex's .addEdge() method with the first vertex and nothing for the weight. This is the unweighted edge.

Print out both vertices using their .print() method, and you should see the first vertex with a weighted edge to the second vertex, and the second vertex with an unweighted edge to the first vertex.

3. Checkpoint 3 Passed

3.

Next, we should feed in the weight argument to the calls to Vertex's .addEdge() method from the Graph's .addEdge() method if the graph is weighted.

In the Graph class, add a third parameter for weight in the .addEdge() method. Create a variable edgeWeight, and set it to the weight argument if the graph is weighted, otherwise set it to null. Pass edgeWeight to the calls that create edges between the given vertices. Remember to do this for both calls.

4. Checkpoint 4 Passed

4.

Let’s verify that the Graph can add weights to the edges by adding an edge between atlantaStation and newYorkStation. In **Graph.js**, edit trainNetwork to be weighted. Then call the its .addEdge() method and pass in a value of 800 as an argument, to represent the number of miles between the two train stations.

When we call the trainNetwork’s .print() method to print out the resulting graph, we should see that the edge between Atlanta and New York has a value of 800.

Graph.js

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

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


class Graph {

 constructor(isWeighted = false) {

  this.vertices = [];

  this.isWeighted = isWeighted;

 }


 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, weight) {
  let edgeWeight = null;
  if (this.isWeighted) {
    edgeWeight = weight;
  }
  if (vertexOne instanceof Vertex && vertexTwo instanceof Vertex) {
    vertexOne.addEdge(vertexTwo, edgeWeight);
    vertexTwo.addEdge(vertexOne, edgeWeight);
  } else {
    throw new Error('Expected Vertex arguments.');
  }
 }

 removeEdge(vertexOne, vertexTwo) {
  if (vertexOne instanceof Vertex && vertexTwo instanceof Vertex) {
    vertexOne.removeEdge(vertexTwo);
    vertexTwo.removeEdge(vertexOne);
  } else {
    throw new Error('Expected Vertex arguments.');
  }
 }

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

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

```
trainNetwork.addEdge(atlantaStation, newYorkStation, 800);

trainNetwork.print();

module.exports = Graph;
```

Vertex.js

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

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

  addEdge(vertex, weight) {
    if (vertex instanceof Vertex) {
      this.edges.push(new Edge(this, vertex, weight));
    } else {
      throw new Error('Edge start and end must both be Vertex');
    }
  }

  removeEdge(vertex) {
    this.edges = this.edges.filter(edge => edge.end !== vertex);
  }

  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;
```