Graph Exercises

Download starter code

Implement the following on the Graph class. The Node and Graph constructor has been provided for you in ***graph.js***

addVertex / addVertices

This function should add a node to the graph.

```
let graph = new Graph()
let a = new Node("A")
let b = new Node("B")
let c = new Node("C")
graph.addVertices([a,b])
graph.addVertex(c)
graph.nodes.has(a) // true
graph.nodes.has(b) // true
graph.nodes.has(c) // true
```

addEdge

This function should add an edge between two nodes in the graph and place each value of the nodes in each array for the value of the node in the adjacency list.

```
let graph = new Graph()
let a = new Node("A")
let b = new Node("B")
let c = new Node("C")
let d = new Node("D")
graph.addVertices([a, b, c, d])
graph.addEdge(a, b)
graph.addEdge(a, c)
graph.addEdge(b, d)
graph.addEdge(c, d)

a.adjacent // contains b and c
b.adjacent // contains a and d
c.adjacent // contains a and d
d.adjacent // contains b and c
```

removeEdge

This function should accept two nodes and remove the edge between them. It should modify the adjacency list to ensure that both values are not in each array for the two nodes which no longer contain the edge.

```
let graph = new Graph()
let a = new Node("A")
let b = new Node("B")
let c = new Node("C")
let d = new Node("D")
graph.addVertices([a, b, c, d])
graph.addEdge(a, b)
graph.addEdge(a, c)
graph.addEdge(b, d)
graph.addEdge(c, d)

graph.removeEdge(b,a)
graph.removeEdge(c,d)


a.adjacent // does not contain b
b.adjacent // does not contain a

c.adjacent // does not contain d
d.adjacent // does not contain c
```

removeVertex

This function should remove the node in the array of nodes and also remove all edges that the removed node previously contained.

```
let graph = new Graph()
let a = new Node("A")
let b = new Node("B")
let c = new Node("C")
let d = new Node("D")
graph.addVertices([a, b, c, d])
graph.addEdge(a, b)
graph.addEdge(a, c)
graph.addEdge(b, d)
graph.addEdge(c, d)

graph.removeVertex(c)
graph.removeVertex(d)

graph.nodes.has(a) // true
graph.nodes.has(b) // true
graph.nodes.has(c) // false
graph.nodes.has(d) // false
```

depthFirstSearch

This function should return an array of nodes visited using DFS. You can do this iteratively (using a stack) or recursively, but note the order of the results will be different. Try to solve this without consulting the lecture notes!

```
let graph = new Graph()
let S = new Node('S');
let P = new Node('P');
let U = new Node('U');
let X = new Node('X');
let Q = new Node('Q');
let Y = new Node('Y');
let V = new Node('V');
let R = new Node('R');
let W = new Node('W');
let T = new Node('T');

graph.addVertices([S,P,U,X,Q,Y,V,R,W,T])

graph.addEdge(S, P);
graph.addEdge(S, U);

graph.addEdge(P, X);
graph.addEdge(U, X);

graph.addEdge(P, Q);
graph.addEdge(U, V);

graph.addEdge(X, Q);
graph.addEdge(X, Y);
graph.addEdge(X, V);

graph.addEdge(Q, R);
graph.addEdge(Y, R);

graph.addEdge(Y, W);
graph.addEdge(V, W);

graph.addEdge(R, T);
graph.addEdge(W, T);

// this is one option:
graph.depthFirstSearch(S) // ["S", "P", "U", "X", "Q", "V", "Y", "R", "W", "T"]
```

breadthFirstSearch

This function should return an array of vertices visited using BFS. Try to solve this without consulting the lecture notes!

```
let graph = new Graph()
let S = new Node('S');
let P = new Node('P');
let U = new Node('U');
let X = new Node('X');
let Q = new Node('Q');
let Y = new Node('Y');
let V = new Node('V');
let R = new Node('R');
let W = new Node('W');
let T = new Node('T');

graph.addVertices([S,P,U,X,Q,Y,V,R,W,T])

graph.addEdge(S, P);
graph.addEdge(S, U);

graph.addEdge(P, X);
graph.addEdge(U, X);

graph.addEdge(P, Q);
graph.addEdge(U, V);

graph.addEdge(X, Q);
graph.addEdge(X, Y);
graph.addEdge(X, V);

graph.addEdge(Q, R);
graph.addEdge(Y, R);

graph.addEdge(Y, W);
graph.addEdge(V, W);

graph.addEdge(R, T);
graph.addEdge(W, T);

// this is one option:
graph.depthFirstSearch(S) // ["S", "U", "V", "W", "T", "R", "Q", "Y", "X", "P"]
```

Further Study

shortest path

Write a function which accepts a graph, a source vertex and target vertex and returns the shortest path. You can assume your graph is unweighted and undirected.

course-schedule

https://leetcode.com/problems/course-schedule

numberOfIslands

https://leetcode.com/problems/number-of-islands/

course-schedule-ii

https://leetcode.com/problems/course-schedule-ii/

cloneGraph

https://leetcode.com/problems/clone-graph/

hasCycle

Write a function on the graph class which returns true if the graph contains a cycle

shortestReach

https://www.hackerrank.com/challenges/bfsshortreach/problem

graphValidTree

https://leetcode.com/problems/graph-valid-tree

roadsAndLibraries

https://www.hackerrank.com/challenges/torque-and-development/problem

evenTree

https://www.hackerrank.com/challenges/even-tree/problem

Solution

View our solutions