Quiz 12 (Dec 11)

By taking this quiz, you agree to adhere to the honor code of the class.

Name: netid:

Name: netid:

Write a method `ArrayList<Vertex<V>> shortestPath(Vertex<V> start, Vertex<V> end)` for `EdgeListGraph` that returns a shortest path (shortest in number of edges) between start and end if a path exists. In terms of n (number of vertices), m (number of edges), and d (number of maximum degree), your algorithm should be O(m + n). Justify the time.

Simplified from github

```
public class EdgeListGraph<V, E>{
        public class Vertex<V> {
                V element;
        }
        public class Edge<E, V>{
                E element;
                Vertex<V>[] endpoints;
        }
        DoublyLinkedList<Vertex<V>> vertices;
        DoublyLinkedList<Edge<E, V>> edges;
        int n_vertices;
        int n_edges;
        List<Edge<E, V>> outgoingEdges(Vertex<V> v);
        List<Edge<E, V>> incomingEdges(Vertex<V> v);
        Edge<E, V> getEdge(Vertex<V> from, Vertex<V> to);
        Vertex<V>[] endVertices(Edge<E, V> e);
        Vertex<V> opposite(Vertex<V> v, Edge<E, V> e);
}
```

You may need other data structures in the class. You can make up names for them, for instance, HashSet, as long as it's clear what data structures you intend to use.

```
ArrayList<Vertex<V>> shortestPath(Vertex<V> start, Vertex<V> end) {
        HashSet<Vertex<V>> visited = new HashSet<>();
        HashMap<Vertex<V>, Vertex<V>> parent = new HashMap<>();
        Queue<Vertex<V>> queue = new Queue();
        queue.enqueue(start);
        visited.add(start);
        while (!queue.isEmpty()) {
                Vertex<V> node = queue.dequeue();
                for (Edge<E, V> e : node.outgoingEdges()) {
                        Vertex<V> other = opposite(e, node);
                        if (!visited.contains(other)) continue; // skip if visited
                        parent.put(other, node);
                        visited.add(other);
                        if (other == end) {
                                ArrayList<Vertex<V>> ans = new ArrayList<Vertex<V>>();
                                // Define a variable to traverse back to find the path
                                Vertex<V> t = end;
                                while (t != start) {
                                        ans.addFirst(t);
                                        t = parent.get(t);
                                }
                                ans.addFirst(start);
                                return ans;
                        }
                        queue.enqueue(other);
        }
        return null;
}
```
This is a BFS, so the time complexity is O(m+n).