AP Computer Science　　Project 35a - Was 2017
Dr. Paul Bailey　　Monday, April 9, 2017

Due Thursday, April 26, 2018 at 1:59 PM. Zip and email entire netbeans project to `paul.bailey@basised.com` before that time.

This project involves the mathematical subject known as Graph Theory. We begin by defining the mathematical terms, outlining the underlying problems to be solved, and then outlining the programs to be built.

A *graph* is a pair $(V, \mathcal{E})$, where V is a set whose members are called *vertices*, and $\mathcal{E}$ is a set whose members are called *edges*, where an *edge* is a subset of V which contains exactly two (distinct) vertices.

We view the vertices as points and the edges as line segments between them; however, this viewpoint only models the abstract notion of graph.

Two vertices are *adjacent* is they are contained in the same edge. The *degree* of a vertex is the number of adjacent vertices.

A *subgraph* of $(V, \mathcal{E})$ is a graph $(W, \mathcal{F})$ such that $W \subset V$ and $\mathcal{F}\mathcal{E}$. Note that if $(W, \mathcal{F})$ is a subgraph of $(V, \mathcal{W})$, the edges in $\mathcal{F}$ must contain only vertices that come from W.

A *walk* in a graph $(V, \mathcal{E})$ is a finite sequence of at least three vertices such that their exists an edge between consecutive vertices. We say that the walk *visits* each of the vertices in the sequence, and that it *traverses* each edge between consecutive vertices. The *length* of the walk is the number of vertices in the sequence, minus one. The first vertex in the sequence is called the *initial* vertex of the walk, and the last is called the *terminal* vertex of the walk.

The graph is *connected* if their exists a walk between any two vertices. A *component* of a graph is a maximal connected subgraph. Two vertices are *associates* if they lie in the same component.

A walk is *Eulerian* if it traverses every edge in the graph.

A *trail* is a walk with distinct edges. A *path* is a walk with distinct vertices.

A *circuit* is a walk whose initial vertex equals its terminal vertex. A *cycle* is a circuit which visits each vertex at most once, except for the initial and terminal vertex.

A *tree* is a connected graph which does not admit a cycle. Given two vertices in a tree, there is exactly one trail from one to the other. A *forest* is a graph in which each component is a tree.

- A connected graph has an Eulerian circuit if and only if every vertex has even degree.
- A connected graph has an Eulerian trail if and only if exactly zero or two vertices have odd degree.

We wish to design a Java class `Graph` which models these definitions, and then build code to solve the following problems.

- Create a program which detects whether a graph is connected.
- Create a program which finds the number of distinct components of a graph.
- Create a program which detects if the graph is a tree or a forest.
- Create a program which returns components of a graph.
- Create a program which detects whether a graph admits an Eulerian trail.
- Create a program which detects whether a graph admits an Eulerian circuit.
- Create a program which finds an Eulerian circuit (Hierholzer's algorithm).
- Create a program which finds the shortest path between two vertices in a graph (Dijktra's Algorithm).

Program 0. Download the zip file containing the `GraphTheory` project from the webpage.

You are now tasked with completing as many of the following programs as you can. Add methods to the `Graph` class where appropriate.

Program 1. Create a method `public List<Vertex> adjacents(Vertex v)` returns a list of the vertices which share an edge with v (excluding v). Create another method `public List<Vertex> associates(Vertex v)` returns a list of vertices which may be visited by a walk initiating at v (including v).

Program 2. Create a method `public boolean isConnected()` which returns `true` if the graph is connected.

Program 3. Create a method `public int numberOfComponents()` which returns the number of components. This is nonnegative and is zero if and only if the number of vertices is zero.

Program 4. Create a method `public boolean isTree()` which returns `true` if the graph is a tree. Create another method `public boolean isForest()` which returns `true` if the graph is a forest.

Program 5. Create a method `public Graph component(Vertex v)` which returns the component containing v. Create another method `public List<Graph> components()` which returns a list of all the components of the graph.

Program 6. Create a method `public boolean admitsEulerianCircuit()` which return `true` if the graph admits an Eulerian circuit. Create another method `public boolean admitsEulerianTrail()` which returns `true` if the graph admits any Eulerian trail.

Program 7. Create a class `Walk` which models walks. Create methods `isTrail`, `isPath`, `isCircuit`, `isCycle`, `isEulerian` inside this class.

Program 8. Create a method `public Walk eulerianCircuit` which returns an Eulerian circuit, if one exists.

Program 9. Create a method `public Walk shortestTrail(Vertex v, Vertex w)` which returns a trail of minimal length between two vertices. Create another method `public int distance(Vertex v, Vertex w)` which returns the minimum number of edges required to walk from v to w.