AP Computer Science — Project 35a - Was 2016
Dr. Paul Bailey — April 13, 2017

Due Sunday, April 23, at 11:59 P.M. Complete as much as you can and email a COMPLETE .ZIP of your ENTIRE NETBEANS PROJECT to `paul.bailey@basis.ed`. It is required that the entire project compiles in NETBEANS.

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. Adjacent vertices are called *neighbors*. 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 visited, 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.

A *trail* is a walk with distinct edges. A trail is *Eulerian* if it traverses every edge in the graph.

A *path* is a trail if which visits each vertex at most once, except possibly if the initial and terminal vertices are the same. A *circuit* is a walk whose initial vertex equals its terminal vertex. A *cycle* is a path which is a circuit.

A *tree* is a connected graph which does not admit a circuit. Note that every trail in a tree is simple. Given two vertices in a tree, there is exactly one trail from one to the other.

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.
- Create a program which detects whether a graph admits an Eulerian path.
- Create a program which detects whether a graph admits an Eulerian cycle.
- Create a program the finds the shortest path between two vertices in a graph (Dystra's Algorithm).

Program 1. Create the `Vertex`, `Edge`, and `Graph` classes are described and outlined during class.

```
public class Vertex
{
    private int id;
    public Vertex(int a)
    public int id()
    public String toString()
    public boolean equals(Vertex that)
}

public class Edge
{
    private Vertex v;
    private Vertex w;
    public Edge(Vertex v, Vertex w)
    public Edge(int a, int b)
    private void initialize(int a, int b)
    public Vertex v()
    public Vertex w()
    public boolean contains(Vertex u)
    public String toString()
    public boolean equals(Edge that)
}

public class Graph
{
    private ArrayList<Vertex> vertices = new ArrayList<Vertex>();
    private ArrayList<Edge> edges = new ArrayList<Edge>();
    public int sizeV()
    public int sizeE()
    public Vertex getV(int i)
    public Edge getE(int i)
    public boolean add(Vertex v)
    public boolean add(Edge e)
    public boolean contains(Vertex v)
    public boolean contains(Edge e)
    public int degree(Vertex v)
    public ArrayList<Vertex> neighbors(Vertex v)
    public void print()
}
```

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

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 nonzero and is zero if and only if the number of vertices is zero.

Program 4. Create a method `public boolean isTree()` which return `true` if the graph is a tree.

Program 5. Create a method `public boolean admitsEulerian(Vertex v)` which return `true` if the graph admits an Eulerian path starting at vertex v. Create another method `public boolean admitsEulerian()` which returns `true` if the graph admits any Eulerian path.

Program 6. Create a class `Walk` which models walks. Create methods `isTrail`, `isPath`, `isCircuit`, `isCycle` inside this class. Create a method `public Walk shortestPath(Vertex v, Vertex w)` to find the shortest path between two vertices.