# Graphs and Graph Algorithms

The study of networks has become one of the great scientific hotbeds of this century, though mathematicians and other have been studying networks for many hundreds of years. Recent developments in computer technology (the Internet, for example) and in social theory (the social network, as popularized by the concept of "six degrees of separation"), not to mention social media, have put a spotlight on the study of networks.

In this chapter we look at how networks are modeled with *graphs*. We define what a graph is, how to represent them in JavaScript, and how to implement important graph algorithms. We also discuss the importance of choosing the correct data representation when working with graphs, since the efficiency of graph algorithms largely depends on the data structure used to represent a graph.

## Graph Definitions

A graph consists of a set of *vertices* and a set of *edges*. Think of a map of a U.S. state. Each town is connected with other towns via some type of road. A map is a type of graph where each town is a vertex and a road that connects two towns is an edge. Edges are defined as a *pair* (v1, v2), where v1 and v2 are two vertices in a graph. A vertex can also have a *weight*, which is sometimes called a *cost*.

A graph whose pairs are ordered is called a *directed graph*, or just a *digraph*. A directed graph is shown in Figure x.1. If a graph is not ordered, it is called an *unordered graph*, or just a graph. An example of an unordered graph is shown in Figure x.2.

A *path* is a sequence of vertices in a graph such that all vertices are connected by edges. The length of a path is the number of edges from the first vertex in the path to the last vertex. A path can also consist of a vertex to itself, which is called a *loop*. Loops have a length of 0.

A *cycle* is a path of at least 1 in a directed graph so that the beginning vertex is also the ending vertex. In a directed graph, the edges can be the same edge, but in an unordered graph, the edges must be distinct.

An unordered graph is considered *connected* if there is a path from every vertex to every other vertex. In a directed graph, this condition is called *strongly connected*. A directed graph that is not strongly connected, but is considered connected, is called *weakly connected*. If a graph has an edge between every set of vertices, it is said to be a *complete* graph.

## Real-World Systems Modeled by Graphs

Graphs are used to model many different types of real-world systems. One example is traffic flow. The vertices represent street intersections and the edges represent the streets. Weighted edges can be used to represent speed limits or the number of lanes. Modelers can use the system to determine the best routes and the streets most likely to suffer from traffic jams.

Any type of transportation system can be modeled using a graph. For example, an airline can model its flight system using a graph. Each airport is a vertex and each flight from one vertex to another is an edge. A weighted edge can represent the cost of a flight from one airport to another, or perhaps the distance from one airport to another, depending upon what is being modeled.

## The Graph Class

At first glance, a graph looks much like a tree or a binary tree, and you might be tempted to try to build a graph class like a tree. There are problems with using an object-based approach like that, however, so we will look at a different scheme for representing both vertices and edges.

### Representing Vertices

The first step in building a Graph class is to first build a Vertex class to store the vertices of a graph. This class has the same duties the Node class had with linked lists and binary search trees.

The Vertex class needs two data members – one for the data that identify the vertex and the other to store a Boolean value indicating whether or not the vertex has been visited. These members are named label and wasVisited, respectively.

The only function we need for the class is the constructor function that allows us to set the values for a vertex's data members. Here is the code for the Vertex class:

function Vertex(label) {

this.label = label;

this.wasVisited = false;

}