P155-P157

# 第11章 图和图算法

网络研究已经成为本世纪最棒的科学温床之一，虽然数学家和其他人已经研究了几百年网络。科学技术（例如，互联网）以及社会理论的近期发展（社交网络，随着“六度分隔理论”概念而普及），更不用提社交媒体，都将焦点放在了网络研究上。

本章我们将看到网络是怎样使用图来建模的。我们将定义什么是图，怎样用JavaScript来表示图，以及如何实现重要的图算法。我们还将讨论使用图时选择正确数据展现的重要性，因为图算法的效率大部分依赖用于表示图的数据结构。

## 图的定义

***图***（graph）由顶点的集合和边的集合组成。想像一下美国的一个州的地图。每个镇都是通过某种类型的道路与其他镇相连。地图就是一种图，其中每个镇都是一个***顶点***（vertex），连接两个镇之间的每条道路都是一条***边***（edge）。边使用(v1,v2)顶点对定义，其中v1和v2是图的两个顶点。顶点还可以有权重（weight），有时候也被称为开销（cost）。顶点对有序的图被称为***有向图***（directed graph），或者digraph。有向图中的顶点对有顺序时，会从一个顶点到另一个顶点画一个箭头。有向图指出了从顶点到顶点之间的流动方向。计算机程序中指示计算方向的流程图就是一个有向图的实例。图11-1展示了一个有向图：

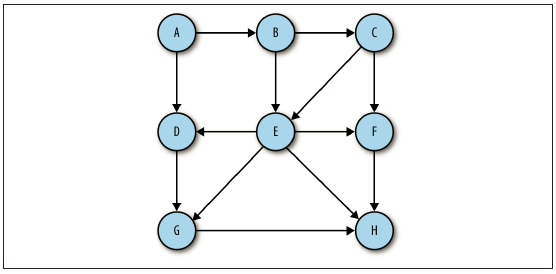


图11-1.有向图（digraph）

如果一个图的顶点对没有顺序，就被称为无向图，或者图。图11-2展示了一个无向图的实例：

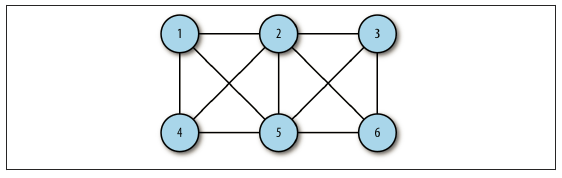


图11-2.无向图

***路径***（path）是图中顶点的序列，路径中的所有顶点都通过边连接。路径的长度是从路径中第一个顶点到最后一个顶点所经过的边的数量。路径也可以是一个指向自己的顶点，称为***环***（loop），环的长度为0。

***圈***（cycle）是至少有一条边且起点和终点相同的路径。对于有向图和无向图，没有重复边和重复顶点的圈叫***简单圈***（simple cycle）。除了第一个顶点和最后一个顶点有其他顶点重复的路径被称为***平凡圈***（general cycle）。

如果从一个点到另一个点有路径，则认为这两个点是***强连通***（strongly connected）的，反之亦然。如果一个图是有向图，而且图的所有顶点都是强连通的，那么这个有向图也被认为是强连通的。

## 用图建模的真实系统

现实世界中很多不同类型的系统都使用图来建模。交通流就是一个例子。顶点代表了街道的交叉口，边代表街道。边的权值可用于表示速度限制或者车道数量。建模者可以用该系统确定最佳路径以及最有可能陷入交通阻塞的街道。

任何类型的交通系统都可以使用图来建模。例如，航空公司可以用图来为其飞行系统建模。每个机场是一个点，从一个点到另一个点的每条航线是一条边。边的权值可以表示从一个机场到另一个机场的飞行成本，或者从一个机场到另一个机场的距离，这取决于建模对象。

计算机网络，包括局域网和更加广域的网络，例如互联网，也常使用图来建模。另一个可以使用图来建模的真实系统实例是消费者市场，其中点代表机构（供应商）和消费者。

## 图类

初看起来，图很像一棵树或者二叉树，你很可能会像创建树一样来创建图类，使用节点来代表顶点。然而，像这样使用基于对象的方法是有问题的，因为图可能增长得非常大。仅使用对象来表示图很快会变得低效，因此我们使用一种不同的模式来表示顶点和边。

## 顶点的表示

创建图类的第一步是创建一个保存图顶点的Vertex类。这个类和链表以及二叉搜索树的Node类有相同的职责。Vertex类需要两个数据成员：一个用来确定点，另一个用布尔类型来标记这个点是否被访问过。这两个数据成员分别命名为label和wasVisited。Vertex类只需要一个构造函数，来允许我们为点的数据成员设值。这是Vetex类的代码：

Function Vertext(lable) {

this.lable = label;

}

我们把点的列表存到数组中，并在Graph类中通过它们在数组中的位置来引用。

P158-

## 边的表示

图的真实信息存储在边里，因为边描述了图的结构。正如我们前面提到的，将图作为二叉树表示充满诱惑，但这样做是错误的。二叉树的表示特别固定，因为一个父亲节点只能有两个孩子节点，而图的结构提供了更多的灵活性。例如，可以有多条边或者仅仅一条边连接到一个单独的顶点。

我们用于表示图边的方法被称为***邻接表***（adjacency list）或者邻接表数组。有了这种方法，边会被存储为以顶点做索引，由顶点的相邻顶点列表（数组）组成的数组。使用这种方案，在程序中引用一个顶点时，我们可以高效地访问这个点连通的所有点的列表。例如，如果顶点2连通了顶点0、1、3和4，并且存储在数组中的位置2，访问这个元素让我们可以访问在数组位置2存储的由顶点0、1、3、4组成的数组。这是我们在本章选择使用的表示方法，如图11-3所示：

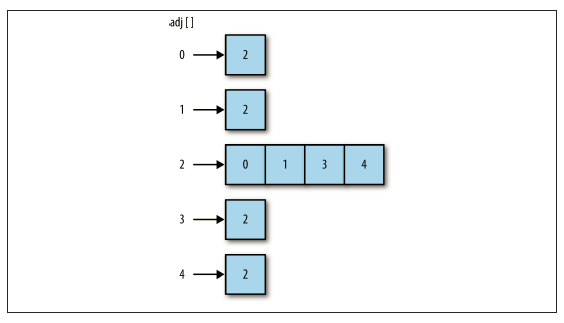


图11-3：邻接表

另一种表示图边的方法被称为邻接矩阵（adjacency matrix）。这是一个二维数组，数组中的元素代表在两个顶点之间是否有边。

## 构建一个图

一旦决定怎样用代码表示图，创建一个表示图的类就非常直观了。这里有Graph类的第一个定义：

function Graph(v) {

this.vertices = v;

this.edges = 0;

this.adj = [];

for (var i = 0; I < this.vertices; ++i) {

this.adj[i] = [];

this.adj[i].push(“”);

}

this.addEdge = addEdge;

this.toString = toString;

}

这个类会记录图中表示了多少条边，以及顶点的数量，利用了一个长度与图中顶点数相同的数组。在数组中的每个元素，for循环会添加一个子数组来存储所有的相邻点，并将每个元素初始化为空字符串。

addEdge()函数定义如下：

function addEdge(v, w) {

this.ajd[v].push(w);

this.adj[w].push(v);

this.edges++;

}

使用两个顶点A和B调用该函数时，函数会查找顶点A的邻接表并将顶点B加到列表中，在查找顶点B的邻接表并将A加入列表中。最终，这个函数会将边的数量加1。

showGraph()函数通过展示所有顶点的列表以及顶点的相邻顶点来显示图：

function showGraph() {

for (var I = 0; i < this.vertices; ++i) {

putstr(i + ‘>’);

for (var j = 0; j < this.vertices; ++j) {

if (this.adj[i][j] != undefined)

putstr(this.adj[i][j] + ‘ ’);

}

print();

}

}

示例11-1：显示Graph类的完整定义

P159