**4.1 无向图**

图由一组顶点和一组能够将两个顶点相连的边组成。

一般用0到V-1来表示一张含有V个顶点的图中各个顶点。这样约定是为了方便使用数组的索引来编写能够高效访问各个顶点中信息的代码。

我们用v-w的记法来表示连接v和w的边，w-v是这条边的另一种表示方法。

我们的定义允许出现两种简单而特殊的情况：

自环，即一条链接一个顶点和其自身的边。

连接同一对顶点的两条边称为平行边。

数学家常常将含有平行边的图称为多重图，而将没有平行边或自环的图称为简单图。

一般来说，实现允许出现自环和平行边，但我们不会将它们作为示例。因此，我们用两个顶点就可以指代一条边。

**4.1.1 术语表**

当两个顶点通过一条边相连时，我们称这两个顶点是相邻的，并称这条边依附于这两个顶点。

某个顶点的度数即为依附于它的边的总数。

子图是由一幅图的所有边的一个子集以及它们所依附的所有顶点组成的图。

在图中，路径是由边顺序链接的一系列顶点。

简单路径是一条没有重复顶点的路径。

环是一条至少含有一条边且起点和终点相同的路径。

简单环是一条不含有重复顶点和边的环。

路径或者环的长度为其中所包含的边数。

大多数情况下，我们研究的是简单环和简单路径并会省略掉简单二字。

当两个顶点之间存在一条连接双方的路径时，我们称一个顶点和另一个顶点是连通的。

如果从任意一个顶点都存在一条路径到达另一个任意顶点，我们称这幅图是**连通图**。

一幅非连通的图由若干连通的部分组成，它们都是及其极大连通子图。

一般来说，要处理一张图就需要一个个地处理它的连通分量。

树是一幅无环连通图，互不相连的树组成的集合称为森林。

连通图的生成树是它的一幅子图，它含有图中的所有顶点且是一棵树。

图的生成树森林是它的所有连通子图的生成树的集合。

当且仅当一幅含有V个结点的图G满足下列5个条件之一时，它就是一棵树：

1，G有V-1条边且不含有环；

2，G有V-1条边且是连通的；

3，G是连通的，但删除任意一条边都会使它不在连通

4，G是无环图，但添加任意一条边都会产生一条环

5，G中的任意一对顶点之间仅存在一条简单路径

图的密度是指已经连接的顶点对占所有可能被连接的顶点对的比例。

在稀疏图中，被连接的顶点对很少，在稠密图中，只有少部分顶点对之间没有边连接。

二分图是一种能够将所有节点分为两部分的图，其中图的每条边所连接的两个顶点都分别属于不同的部分。

**4.1.2 表示无向图的数据类型**

参考文档：无向图的API

创建一个含有V个顶点但不含有边的图：Graph(int V)

从标准输入流in读入一幅图：Graph(In in)

顶点数：int V()

边数：int E()

向图中添加一条边v-w：addEdge(int v, int w)

和v相邻的所有顶点：Iterable<Integer>adj(int v)

对象的字符串表示：toString()

参考文档：最常用的图处理代码

计算v的度数：int degree(Graph G,int v)

计算所有顶点的最大度数：int maxDegree(Graph G)

计算所有顶点的平均度数：double avgDegree(Graph G)

计算自环个数:int numberOfSelfLoops(Graph G)

图的邻接表的字符串表示：String toString()

有三种表示图的方法：

邻接矩阵：使用一个V乘V的布尔矩阵，当顶点v和顶点w之间有相连接的边时，定义v行w列的元素值为true，否则为false。不能用于顶点很多的图。

边的数组：使用一个Edge类，它含有两个int实例变量。实现adj需要检查图中的所有边，不合适。

邻接表数组：使用一个以顶点为索引的列表数组，其中的每个元素都是和该顶点相邻的顶点列表。

编程：使用Bag实现Graph数据类型

参考文档：图处理算法的API：Search的接口

找到和起点s连通的所有顶点：Search(Graph G,int s)

v和s是连通的吗：boolean marked(int v)

与s连通的顶点总数：int count()

编程：使用深度搜索实现Search类

**4.1.3 深度优先搜索**

走迷宫：用迷宫代替图，通道代替边，路口代替顶点。

要探索迷宫中的所有通道，我们需要：

1，选择一条没有标记过的通道，在你走过的路上铺一条绳子；

2，标记所有你第一次路过的路口和通道；

3，当来到一个标记过的路口时（用绳子）回退到上个路口

4，当回退到的路口，没有可走的通道时继续回退

编程：深度优先搜索

深度优先搜索接口：

构造函数：DepthFirstSearch(Graph G, int s)

遍历路径：dfs(Graph G, int v)

是否连通：boolean marked(int w)

连接点数量：int count()

搜索连通图的经典递归算法，只需用一个递归方法来遍历所有顶点，在访问一个顶点时：

1，将它标记为已访问；

2，递归地访问它的所有没有被标记过的邻居顶点

这种方法称为深度优先搜索。

深度优先搜索标记与起点连通的所有顶点所需的时间和顶点的度数之和成正比。

代码中的方法的调用和返回机制对应迷宫中的绳子的作用。

深度优先搜索中每条边都会被访问两次，且第二次时总会法线这个顶点被访问过，这意味着深度优先搜索的轨迹可能会比你想象的长一倍。

单点路径：给定一幅图和一个起点s，回答“从s到给定目的顶点v是否存在一条路径？如果有，找出这条路径”

**4.1.4 寻找路径**

参考文档：路径的API

在G中找出所有起点为s的路径：Paths(Graph G,int s)

是否存在从s到v的路径：boolean hasPathTo(int v)

S到v的路径，如果不存在则返回null：Iterable<Integer> pathTo(int v)

编程：使用深度优先搜索查找图中的路径

深度优先搜索接口:

构造函数：DepthFirstPaths(Graph G, int s)

构造路径：dfs(Graph G, int v)

是否存在从s到v的路径：boolean hasPathTo(int v)

S到v的路径，如果不存在则返回null：Iterable<Integer> pathTo(int v)

使用深度优先搜索得到从给定起点到任意标记顶点的路径所需的时间与路径长度成正比。

**4.1.5 广度优先搜索**

单点最短路径，从s到给定目的顶点v是否存在一条路径，如果有找出其中最短的那条。

解决单点最短路径问题的经典方法叫做广度优先搜索。

要找到从s到v的最短路径，从s开始，在所有由一条边就可以到达的顶点中寻找v，如果找不到我们就继续在于s距离两条边的顶点中查找v，如此一直进行。

深度优先搜索就好像是一个人在走迷宫，广度优先搜索则好像是一组人在一起朝各个方向走这座迷宫，每个人都有自己的绳子。当出现岔路时，可以假设一个探索者可以分裂为更多的人来搜索它们，当两者相遇时，会合二为一。

编程：使用广度优先搜索查找图中的路径

广度优先搜索的接口同深度优先搜索

对于从s可达的任意顶点v，广度优先搜索都能找到一条从s到v的最短路径。

广度优先搜索所需的时间在最坏情况下和V+E成正比。

**4.1.6 连通分量**

深度优先搜索的下一个直接应用就是找出一幅图的所有连通分量。

参考文档：连通分量的API

预处理构造函数：CC(Graph G)

v和w连通吗：connected(int v, int w)

连通分量数：count()

V所在的连通分量的标识符：id(int v)

编程：使用深度优先搜索找出图中的所有连通分量

深度优先搜索的预处理使用的时间和空间与V+E成正比且可以在常数时间内处理关于图的连通性查询。

union-find比深度优先搜索更快，因为它不需要完整地构造并表示一幅图。

我们在完成只需要判断连通性或是需要完成有大量连通性查询和插入操作混合等类似的任务时，更倾向于使用union-find算法。

而深度优先搜索则更适合实现图的抽象数据类型。

编程：使用深度优先搜索处理图的其他示例。

G是无环图吗？

G是二分图吗？

能够用两种颜色将图的所有顶点着色，使得任意一条边的两个端点的颜色都不相同吗？这个问题也等价于：这是一幅二分图吗？

**4.1.7 符号图**

在典型应用中，图都是通过文件或者网页定义的，使用字符串而非整数来表示和指代顶点。

我们定义拥有以下性质的输入格式：

1，顶点名为字符串

2，用指定的分隔符来隔开顶点名

3，每一行都表示一组边的集合，每一条边都连接着这一行的第一个名称表示的顶点和其他名称所表示的顶点

4，顶点总数V和边的总数E都是隐式定义的

参考文档：用符号作为顶点名的图的API

构造函数：SymbolGraph(String filename,String delim)

Key是一个顶点吗：boolean contains(String key)

Key的索引：int index(String key)

索引v的顶点名：String name(int v)

隐藏的Graph对象：Graph G()

编程：符号图SymbolGraph的数据类型

图处理的一个经典问题就是，找到一个社交网络中两个人间隔的度数。

编程：间隔的度数

**4.1.8 总结**

参考文档：本节中得到解决的无向图处理问题