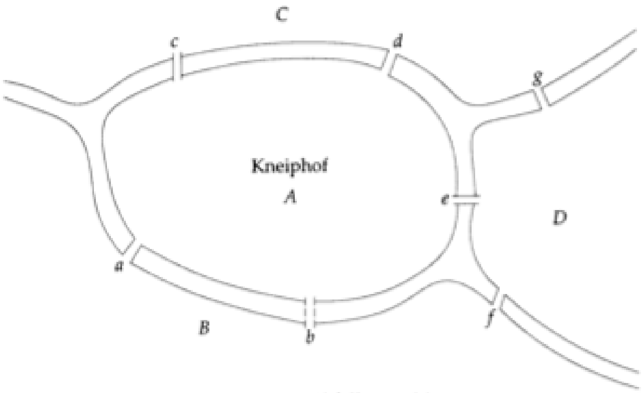
图的定义

张晓平

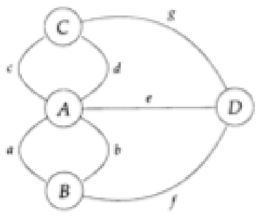
December 18, 2016

1 邻接矩阵

图论是数学的一个分支，最早可以追溯到1736年，数学家欧拉用图论方法解决了Königsberg七桥问题，此后七桥问题成为著名的数学经典。Königsberg城中的Pregel河围绕Kneiphof岛缓缓流过，分成两条支流。Pregel河把Königsberg城分割成四个区域，四个区域由七座桥连接。



(a) Königsberg 城中的 Pregel 河



(b) 七桥问题的欧拉图

四个区域用A, B, C, D标记，七座桥用a, b, c, d, e, f, g标记。七桥问题的提法是：从任何一个区域出发，跨过每座桥一次且一次，问最后能否回到出发的那个区域。

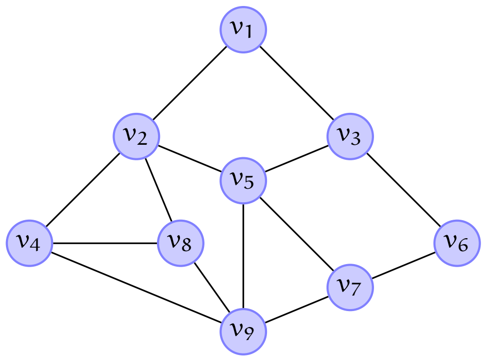
欧拉解决了这一问题，答案是不能。欧拉首先把这个问题描述为抽象的数学对象：图，用图的顶点表示区域，用图的边表示桥梁。图中顶点的度定义为与该顶点邻接的边数，欧拉证明了：如果从图中的一个顶点出发，经过图中所有边一次且仅一次，最后回到出发的顶点，那么当且仅当所有顶点的度都是偶数。后来为了纪念欧拉的发现，这样的回路称为欧拉回路。七桥问题之所以不存在欧拉回路，是因为所有顶点的度均为奇数。

2 图的定义与术语

图G由两个集合V, E组成，其中V是顶点的有限非空集合，E是顶点的二元组集合，顶点二元组称为边。$V(G)$和$E(G)$分别称为图G的顶点集和边集，以后也用$G = (V, E)$表示图。

注意：

- 线性表的数据元素叫元素，树的数据元素叫结点，图的数据元素称为顶点(vertex)。
- 线性表若无数据元素，称为空表。树中没有结点称为空树。但在图中，不允许没有顶点，即顶点集合V有限非空。



1、无向图

- 若顶点v_i到v_j之间的边没有方向，则称该边为无向边(edge)，用无序偶对(v_i, v_j)表示。
- 若图中任意两顶点之间的边都是无向边，则称该图为无向图(undirected graphs)。

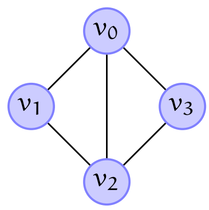


Figure 1: 无向图

2、有向图

- 若顶点v_i到v_j之间的边有方向，则称这条边为有向边，也称为弧(arc)，用有序偶对$\langle v_i, v_j \rangle$表示，其中v_i称为弧尾(tail)，v_j称为弧头(head)。
- 若图中任意两顶点之间的边都是有向边，则称该图为有向图(directed graphs)。

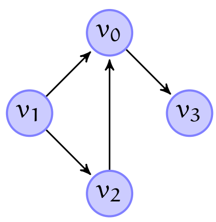


Figure 2: 有向图

3、简单图：若不存在顶点到自身的边，且同一条边不重复出现，则称这样的图为简单图。本章只讨论简单图。

4、无向完全图：在无向图中，若任意两顶点之间都存在边，则称该图为无向完全图。含有n个顶点的无向完全图有$\frac{n(n-1)}{2}$条边。

5、有向完全图：在有向图中，若任意两顶点之间都存在方向相反的两条弧，则称该图为有向完全图。含有n个顶点的有向完全图有$n(n-1)$条弧。

对于具有n个顶点和e条边的图，无向图满足

$$0 \leqslant e \leqslant \frac{n(n-1)}{2},$$

有向图满足

$$0 \leqslant e \leqslant n(n-1).$$

6、稠密图与稀疏图：有很少条边或弧的图称为稀疏图，反之称为稠密图。

这里稀疏与稠密是模糊概念，都是相对而言的。

7、网络

Figure 3: 非简单无向图

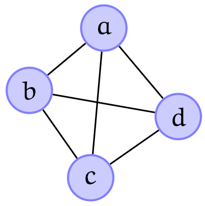


Figure 4: 无向完全图

- 有些图的边或弧具有与它相关的数字，这种与边或弧相关的数称为权(weight)。这些权可以表示从一个顶点到另一个顶点的距离或耗费。
- 这种带权的图称为网(network)。

8、子图：假设有两个图$G=(V,E)$和$G'=(V',E')$，如果$V'\subset V$且$E'\subset E$，则称G'为G的子图(subgraph)。

3 顶点与边的关系

1、无向图顶点的邻接、度
对于无向图$G=(V,E)$，

- 若边$(v,v')\in E$，则称v,v'相邻，称边(v,v')邻接顶点v和v'。
- 顶点v的度(degree)是和v邻接的边的数目，记为$D(v)$。
- 重要关系：

$$e=\frac{1}{2}\sum_{i=1}^{n}D(v_i).$$

2、有向图顶点的邻接、入度、出度和度
对于有向图$G=(V,E)$，

- 若弧$\langle v,v'\rangle\in E$，则称v指向v'，顶点v'发自v，弧$\langle v,v'\rangle$与v,v'邻接。
- 以v为头的弧的数目称为v的入度，记为$D_{in}(v)$；
  以v为尾的弧的数目称为v的出度，记为$D_{out}(v)$；
  顶点v的度为$D(v)=D_{in}(v)+D_{out}(v)$。

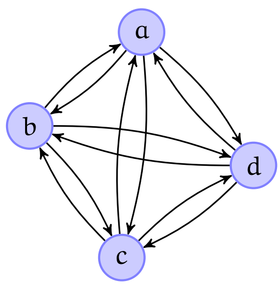


Figure 5: 有向完全图

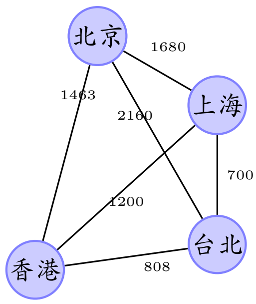


Figure 6: 网

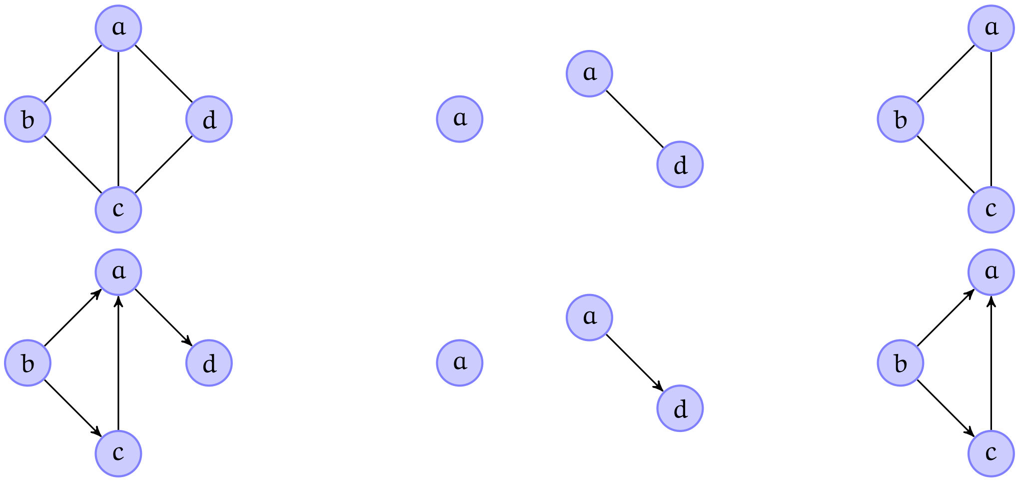


Figure 7: 子图

- 重要关系：

$$e = \sum_{i=1}^{n} D_{in}(v_i) = \sum_{i=1}^{n} D_{out}(v_i).$$

3、无向图的路径
对于无向图$G = (V, E)$，从顶点v到顶点v'的路径(path)是一个顶点序列$(v = v_{i_0}, v_{i_1}, \cdots, v_{i_m} = v')$，其中$(v_{i_{k-1}}, v_{i_k}) \in E$，$1 \leqslant k \leqslant m$。
4、有向图的路径
对于有向图$G = (V, E)$，其路径也是有向的，顶点序列应满足$(v_{i_{k-1}}, v_{i_k}) \in E$，$1 \leqslant k \leqslant m$。
5、路径长度
路径的长度是路径上的边或弧的数目。
6、回路或环
第一个顶点和最后一个顶点相同的路径称为回路或环(cycle)。
7、简单路径
序列中顶点不重复出现的路径称为简单路径。
8、简单回路或简单环 除第一个顶点和最后一个顶点外，其余顶点不重复的回路，称为简单回路或简单环。

4 图的连通性

4.1 无向图的连通性

1、连通图：在无向图中，

- 若顶点v到v'有路径，则称v与v'是连通的。
- 若对任意两个顶点$v_i, v_j \in E$，v_i和v_j都是连通的，则称G是连通图(connected graph)。

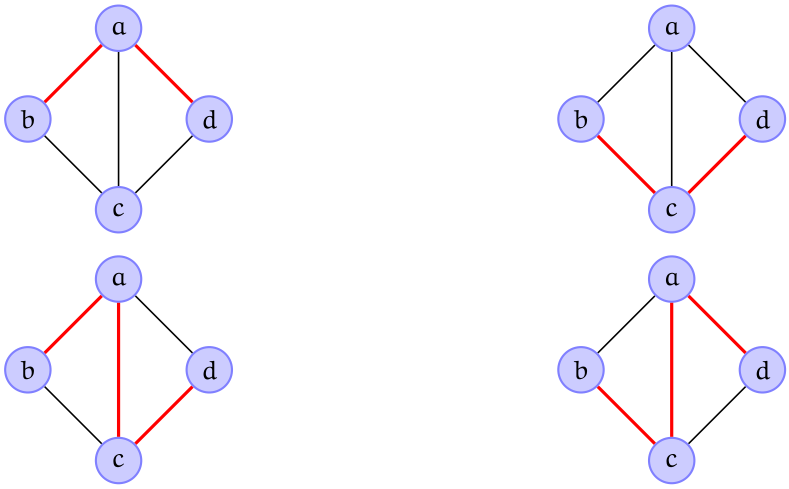


Figure 8: 顶点b到d有四条路径

Figure 9: 顶点b到d有两条路径

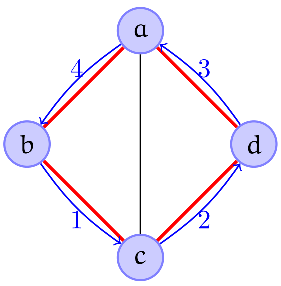


Figure 10: 简单回路

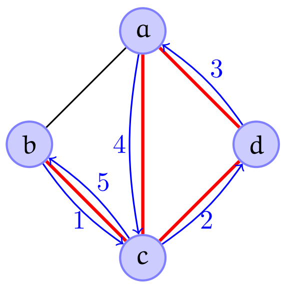


Figure 11: 非简单回路

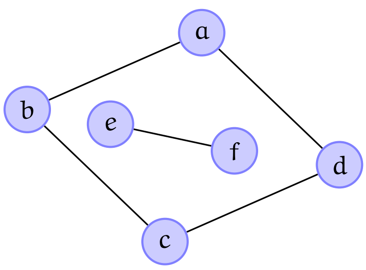


Figure 12: 非连通图

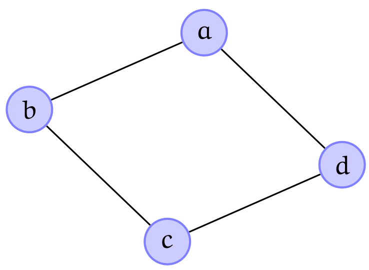


Figure 13: 连通图

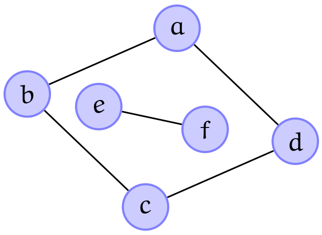


Figure 14: 非连通图

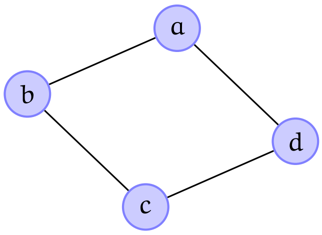


Figure 15: 连通分量

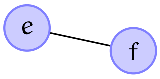


Figure 16: 连通分量

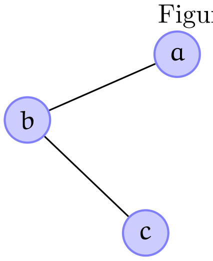


Figure 17: 不是连通分量

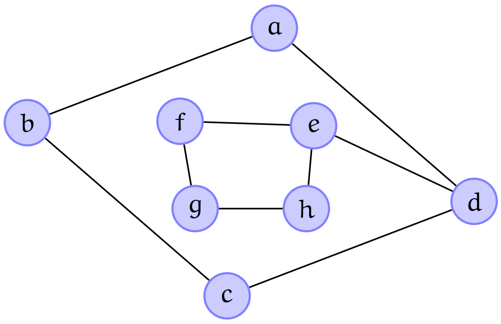


Figure 18: 普通无向图

2、连通分量： 无向图中的极大连通子图称为连通分量。
此概念强调

- 要是子图；
- 子图要是连通的；
- 连通子图含有极大顶点数；
- 具有极大顶点数的连通子图包含相关联于这些顶点的边。

3、连通图的生成树： 一个连通图的生成子树是一个极小的连通子图，它含有图中全部的n个顶点，但只有足以构成一棵树的$n-1$条边。
设无向图G有n个顶点，

- 若边数小于$n-1$，则G是非连通图；
- 若边数大于$n-1$，则G必定构成一个环；
- 但边数等于$n-1$，并不一定是生成树。

4.2 有向图的连通性

1、强连通图： 在有向图G中，若对$v_i, v_j \in V, v_i \neq v_j$，从$v_i$到$v_j$和从$v_j$到$v_i$存在路径，则称$G$是强连通图。有向图中的极大强连通子图称为有向图的强连通分量。
2、有向树的生成森林

- 如果一个有向图恰有一个顶点的入度为0，其余顶点的入度均为1，则它是一棵有向树。

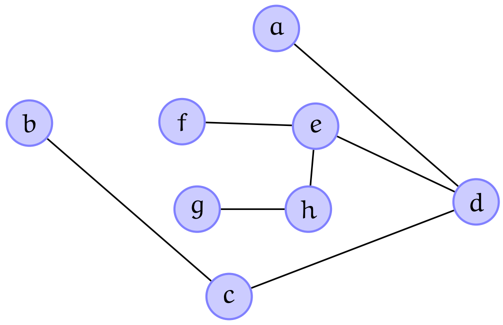


Figure 19: 无向图的生成树

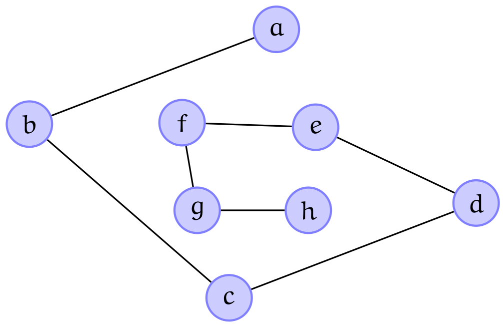


Figure 20: 无向图的生成树

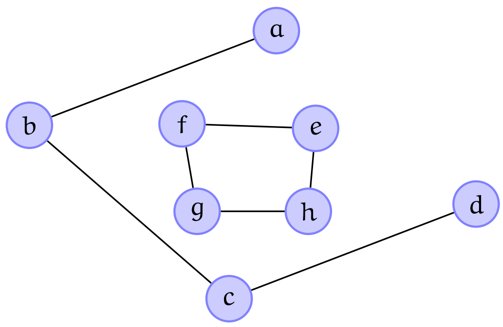


Figure 21: 不是生成树

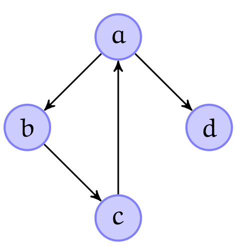


Figure 22: 非强连通图

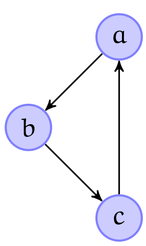


Figure 23: 强连通分量

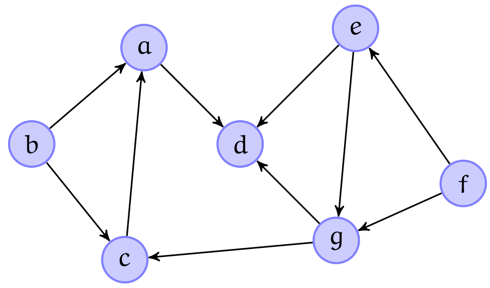


Figure 24: 普通有向树

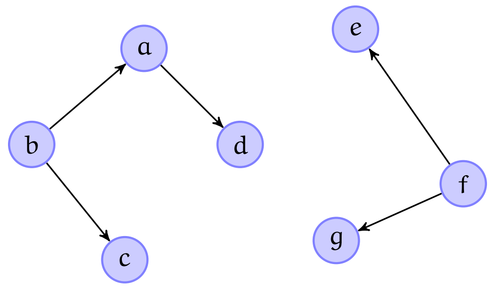


Figure 25: 有向树的生成森林

- 一个有向图的生成森林由若干棵有向树组成，含有图中全部顶点，但只有足以构成若干棵不相交的有向树的弧。

5 图的抽象数据类型

```
ADT Graph
数据对象：顶点的非空集合，由顶点二元组组成的边集
成员函数：
   Graph Create()                        return 空图
   Graph InsertVertex(graph, v)          在graph中插入v，然后return graph
   Graph InsertEdge(graph, v1, v2)       在graph中插入(v1,v2)，然后return graph
   Graph DeleteVertex(graph, v)          在graph中删除v及其邻接边，然后return graph
   Graph DeleteEdge(graph, v1, v2)       在graph中删除(v1,v2)，然后return graph
   Boolean IsEmpty(graph)                判断是否为空图
   List Adjacent(graph, v)               return与v相邻所有顶点的表
end Graph
```