

# 11

## Undirected Graphs

# 无向图

由一组节点和连接节点的无向边组成结构



从某种意义上说，数学是逻辑思想的诗歌。

***Mathematics is, in its way, the poetry of logical ideas.***

—— 阿尔伯特·爱因斯坦 (Albert Einstein) | 理论物理学家 | 1879 ~ 1955



- networkx.DiGraph() 创建有向图的类，用于表示节点和有向边的关系以进行图论分析
- networkx.draw\_networkx() 用于绘制图的节点和边，可根据指定的布局将图可视化呈现在平面上
- networkx.draw\_networkx\_edge\_labels() 用于在图可视化中绘制边的标签，显示边上的信息或权重
- networkx.get\_edge\_attributes() 用于获取图中边的特定属性的字典，其中键是边的标识，值是对应的属性值
- networkx.Graph() 创建无向图的类，用于表示节点和边的关系以进行图论分析
- networkx.MultiGraph() 创建允许多重边的无向图的类，可以表示同一对节点之间的多个关系
- networkx.random\_layout() 用于生成图的随机布局，将节点随机放置在平面上，用于可视化分析
- networkx.spring\_layout() 使用弹簧模型算法将图的节点布局在平面上，模拟节点间的弹簧力和斥力关系，用于可视化分析
- networkx.to\_numpy\_matrix() 用于将图表示转换为 NumPy 矩阵，方便在数值计算和线性代数操作中使用

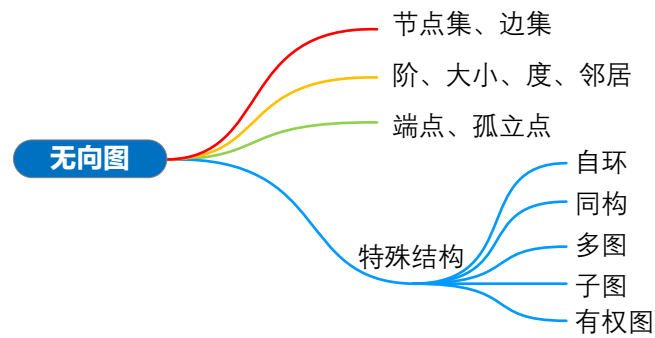
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



## 11.1 无向图：边没有方向

### 节点集、边集

将图 1 中的无向图记做  $G$ 。一个图有两个重要集合：(1) 节点集  $V(G)$ ；(2) 边集  $E(G)$ 。因此， $G$  也常常被写成  $G = (V, E)$ 。

以图 1 的图  $G$  为例， $G$  的节点集  $V(G)$  为：

$$V(G) = \{a, b, c, d\} \quad (1)$$

$G$  的边集  $E(G)$  为：

$$E(G) = \{ab, bc, bd, cd, ca\} = \{(a, b), (b, c), (b, d), (c, d), (c, a)\} \quad (2)$$

上式的第二种集合记法是为了配合 NetworkX 语法。

由于图 1 中图  $G$  是无向图，因此节点  $a$  到节点  $b$  的边  $ab$ ，和节点  $b$  到节点  $a$  边  $ba$ ，没有区别。但是，下一章介绍有向图时，我们就需要注意连接节点的先后顺序了。

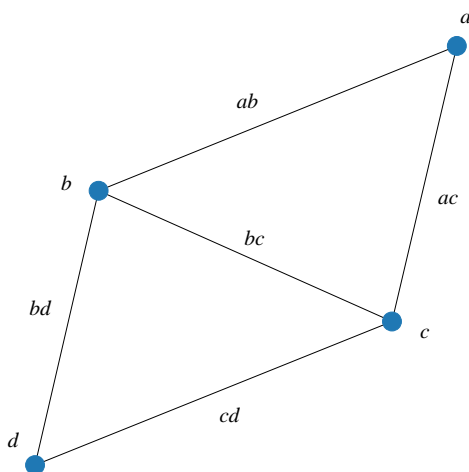


图 1.4 个节点，5 条边的无向图

图 1 是用 NetworkX 绘制，下面聊聊代码 1。

```

import matplotlib.pyplot as plt
import networkx as nx

a undirected_G = nx.Graph()
# 创建无向图的实例

b undirected_G.add_node('a')
# 添加单一顶点

c undirected_G.add_nodes_from(['b', 'c', 'd'])
# 添加多个顶点

d undirected_G.add_edge('a', 'b')
# 添加一条边

e undirected_G.add_edges_from([('b', 'c'),
                                ('b', 'd'),
                                ('c', 'd'),
                                ('c', 'a')])
# 增加一组边

f random_pos = nx.random_layout(undirected_G, seed=188)
# 设定随机种子，保证每次绘图结果一致

g pos = nx.spring_layout(undirected_G, pos=random_pos)
# 使用弹簧布局算法来排列图中的节点
# 使得节点之间的连接看起来更均匀自然

h plt.figure(figsize = (6,6))
nx.draw_networkx(undirected_G, pos = pos,
                  node_size = 180)
plt.savefig('G_4顶点_5边.svg')

```

代码 1. 用 NetworkX 绘制无向图 | Bk6\_Ch14\_01.ipynb

**a** 用 `networkx.Graph()` 创建一个空的无向图对象实例。在这个实例中，我们可以添加节点和边，进行图的各种操作和分析。

**b** 用 `add_node()` 方法增加单一节点 `a`。

**⚠** 注意，只有一个节点的图叫做**平凡图** (trivial graph)。

**c** 用 `add_nodes_from()` 方法增加另外三个节点，这三个节点以列表形式保存 `['b', 'c', 'd']`。

**d** 用 `add_edge()` 方法向图中添加一条连接节点 `'a'` 和 `'b'` 的无向边。

**e** 用 `add_edges_from()` 方法向图中添加一组无向边，连接 `'b'` 与 `'c'`，`'b'` 与 `'d'`，`'c'` 与 `'d'`，`'c'` 与 `'a'`。

**f** 用 `networkx.random_layout()` 设定随机种子值，以确保每次可视化采用相同的布局。

**g** 用 `networkx.spring_layout()` 弹簧布局算法来排列图中的节点。

**h** 用 `networkx.draw_networkx()` 绘制无向图，传入图 `undirected_G`、节点的位置信息 `pos`、节点的大小 `node_size`。

图 2 所示为供大家在 NetworkX 练习的几个无向图，请大家注意对每个图用不同的命名。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

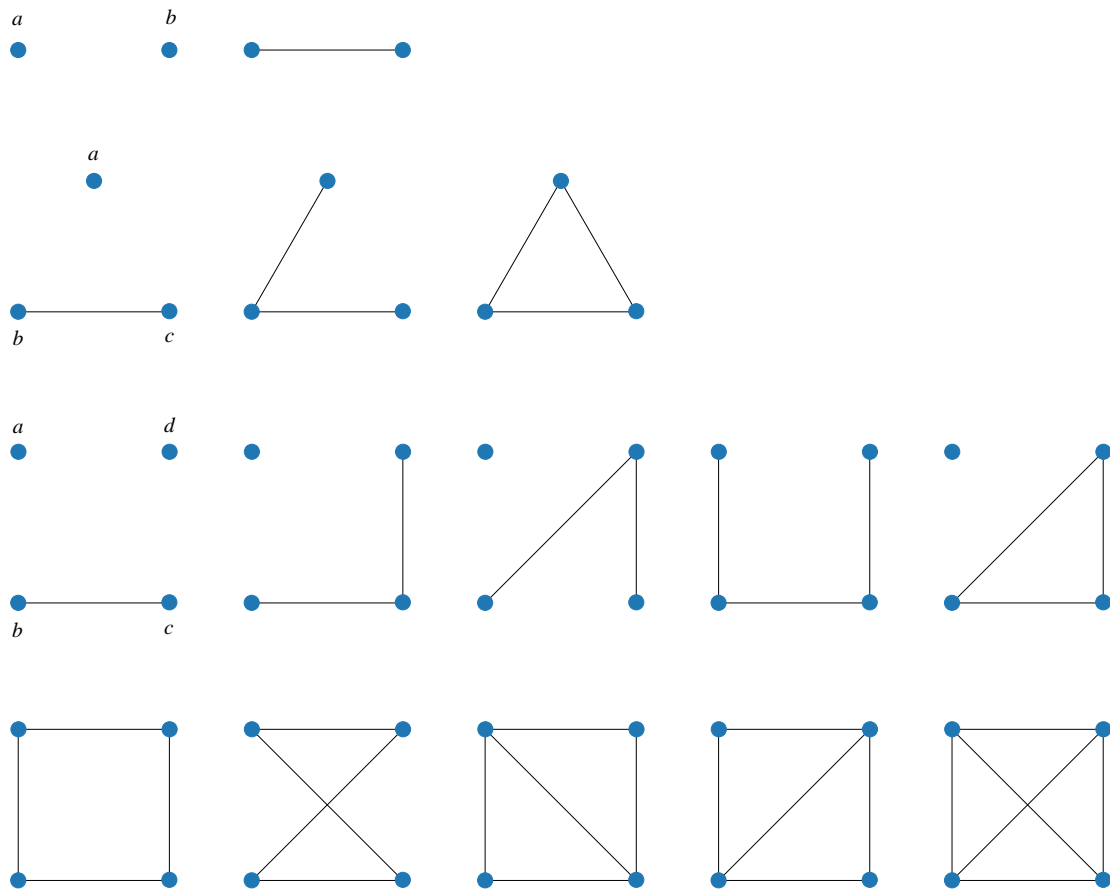


图 2. 供大家练习的几个无向图

### 空图

一个没有边的图叫做**空图** (empty graph)。也就是说，一个非空图至少要有一条边。图 3 所示为三个空图。

⚠ 注意，有些学术文献中，空图是指节点集和边集都是空集的图。

本书空图的定义参考如下两个来源：

<https://mathworld.wolfram.com/EmptyGraph.html>

[https://networkx.org/documentation/stable/reference/generated/networkx.generators.classic.empty\\_graph.html](https://networkx.org/documentation/stable/reference/generated/networkx.generators.classic.empty_graph.html)

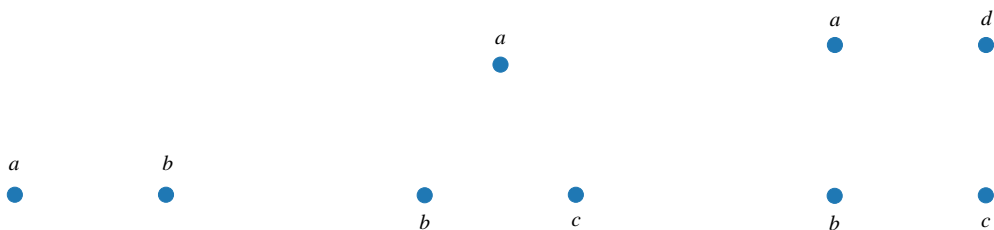


图 3. 三个空图

## 阶、大小、度、邻居

图  $G$  的节点数量叫做**阶** (order)，常用  $n$  表示。图 1 所示的图  $G$  的阶为 4 ( $n = 4$ )，也就是说  $G$  为 4 阶图。

图  $G$  的边的数量叫做图的**大小** (size)，常用  $m$  表示。图 1 所示的图  $G$  的大小为 5 ( $m = 5$ )。

对于无向图，一个节点的**度** (degree) 是与它相连的边的数量。

比如，如图 4 所示，图  $G$  中节点  $a$  的度为 2，记做  $\deg_G(a) = 2$ 。图  $G$  中节点  $b$  的度为 3，记做  $\deg_G(b) = 3$ 。

无向图中，给定一个节点的**邻居** (neighbors) 指的是与该节点通过一条边直接相连的其他节点。

简单来说，如果两个节点之间存在一条边，那么它们就互为邻居。

如图 4 所示，节点  $a$  有两个邻居—— $b$ 、 $c$ 。

如果一个图有  $n$  个节点，那么其中任意节点最多有  $n - 1$  个邻居，它的度最大值也是  $n - 1$ 。注意，这是在不考虑自环的情况下！本章马上介绍自环有关内容。

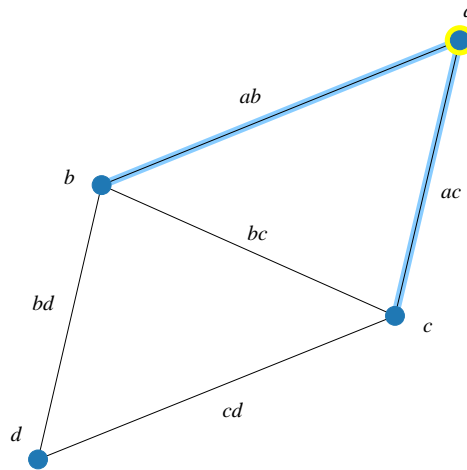


图 4. 节点  $a$  的度为 2，有 2 个邻居

在代码 1 基础上，代码 2 计算阶、大小、度、邻居等值。下面聊聊这段代码。

```

a undirected_G.order()
# 图的阶

b undirected_G.number_of_nodes()
# 图的节点数

c undirected_G.nodes
# 列出图的节点

d undirected_G.size()
# 图的大小

e undirected_G.edges
# 列出图的边

f undirected_G.number_of_edges()
# 图的边数

g undirected_G.has_edge('a', 'b')
# 判断是否存在ab边
# 结果为 True

h undirected_G.has_edge('a', 'd')
# 判断是否存在ad边
# 结果为 False

i undirected_G.degree()
# 图的度

j list(undirected_G.neighbors('a'))
# 邻居

```



代码 2. 用 NetworkX 计算无向图的阶、大小、度、邻居 | Bk6\_Ch14\_01.ipynb

- a 用 `order()` 方法计算无向图的阶，即图中节点总数。
- b 用 `number_of_nodes()` 方法计算无向图中节点数量，结果与阶相同。
- c 用 `nodes` 列出无向图所有节点。
- d 用 `size()` 方法计算了图的大小，即无向图中边数总和。
- e 用 `edges` 列出无向图所有的边。
- f 计算了无向图的边数。
- g 用 `has_edge()` 判断无向图中是否存在连接节点 a 和 b 的边，结果为 `True` 表示存在。
- h 用 `has_edge()` 判断无向图中是否存在连接节点 a 和 d 的边，结果为 `False` 表示不存在。
- i 用 `degree()` 方法计算无向图的度。结果列出所有节点的各自度。  
用 `dict(undirected_G.degree())` 可以将结果转化为字典 `dict`。  
也可以用 `undirected_G.degree('a')` 计算某个特定节点，比如 a，的度。
- j 用 `neighbors()` 方法查找特定节点的邻居，结果是可迭代键值对；用 `list()` 将结果转化为列表。

请大家也计算图 2 中每幅图的阶、大小、度、邻居等值。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

## 端点、孤立点

度数为 1 的节点叫**端点** (end vertex, end node)，如图 5 (a) 所示。

我们可以用 `remove_edge()` 方法删除 `ac` 这条边，比如 `undirected_G.remove_edge('c','a')`。

度数为 0 的节点叫**孤立点** (isolated node, isolated vertex)，如图 5 (b) 所示。

请大家指出图 2 中每幅图可能存在的端点和孤立点。

我们可以用 `undirected_G.remove_edges_from([('b','a'),('a','c')])` 方法删除两条边。

类似地，我们可以用 `undirected_G.remove_node('a')` 从 `undirected_G` 图上删除一个节点；或者用 `undirected_G.remove_nodes_from(['b','a'])` 删除若干节点。

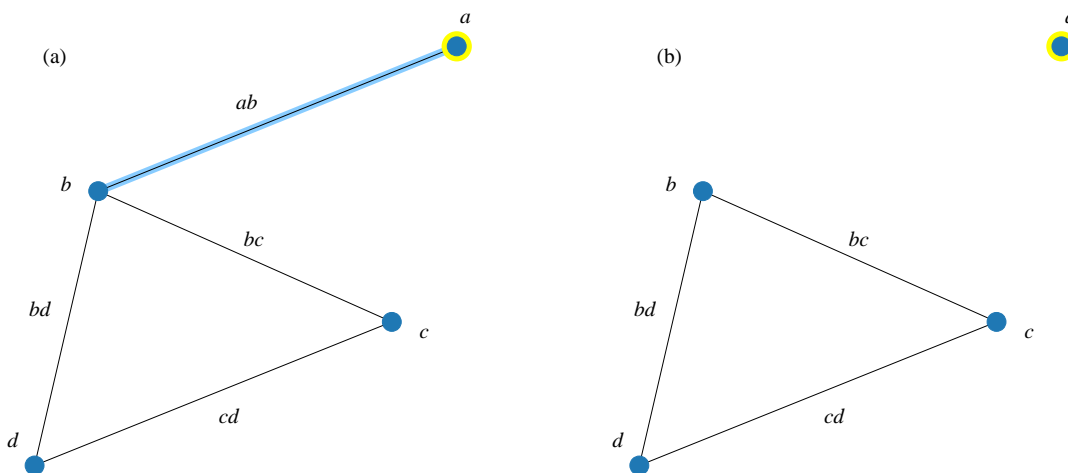


图 5. 端点和孤立点

## 11.2 自环：节点到自身的边

在图论中，一个节点到自身的边被称为**自环** (self-loop)，也叫自环边，也叫圈。简单来说，如图 6 所示，自环就是图中节点 `a` 与它自己之间存在一条边。

这时候，图 6 的大小变为 6；因为在原来 5 条边的基础上，又增加一条边 `aa`。

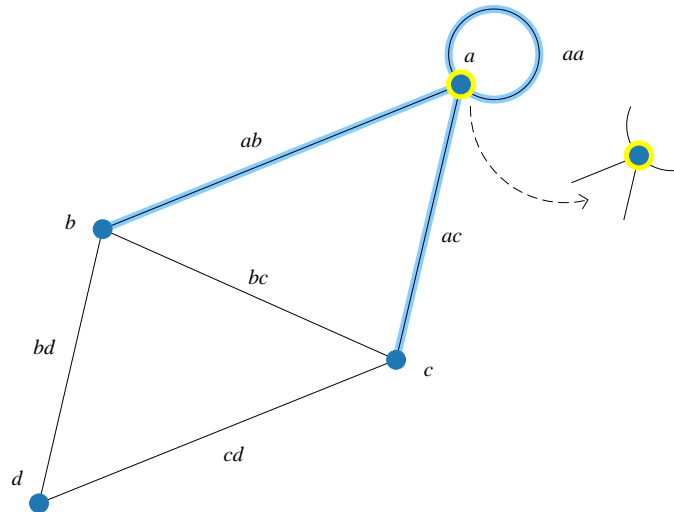
特别请大家注意，这时候节点 `a` 的度从 2 变成了 4。当某个节点增加自环时，它的度将增加 2，因为自环会导致节点与自己连接两次，每次连接都增加了节点的度。自环的存在使得节点的度增加了 2，而不是 1。

从图 6 图上来看，节点伸出了 4 根“触须”。

多了自环，节点 `a` 的邻居变成了 3 个——`a`、`b`、`c`。

也就是说，考虑自环的情况下，如果一个图有  $n$  个节点，那么其中任意节点最多有  $n$  个邻居，它的度最大值也是  $n + 1$ 。



图 6. 节点  $a$  增加自环

在前文代码基础上，代码 3 添加节点  $a$  的自环，并计算大小、度、邻居具体值。请大家自行分析这段代码，并运行结果。

```

a undirected_G.add_edge('a', 'a')
  # 添加一条自环

b # 可视化
  plt.figure(figsize = (6,6))
  nx.draw_networkx(undirected_G, pos = pos,
c                      node_size = 180)
  plt.savefig('G_4顶点_5边_a自环.svg')

d undirected_G.size()
  # 图的大小

e undirected_G.edges
  # 列出图的边

f undirected_G.degree('a')
  # 节点a的度

g list(undirected_G.neighbors('a'))
  # 邻居

```

代码 3. 节点  $a$  增加自环后计算无向图的大小、度、邻居 | Bk6\_Ch14\_01.ipynb

## 11.3 同构：具有等价关系的图

代码 1 在设定随机数种子时，`pos` 已经固定下来；如果没有传入 `pos`，每次运行可视化得到的图外观会随机变化（但是图的基本性质不变），具体如图 7 所示。

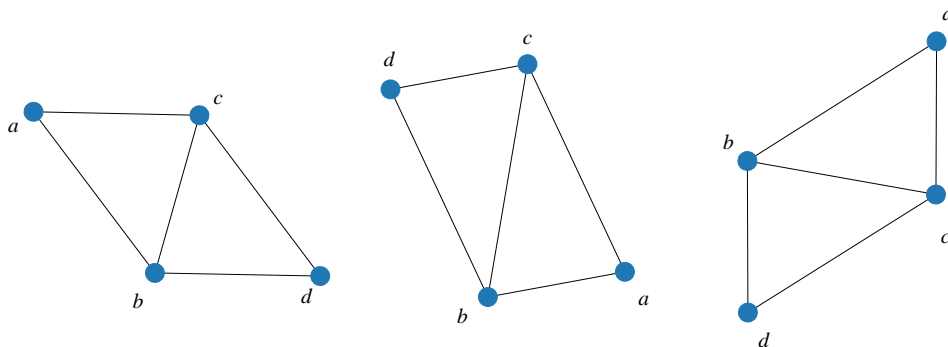


图 7. 图随机布局

图 7 也告诉我们，很多图外观看起来大有不同，节点名称、边名称都不同，但是图的本质完全一致；这种图叫做**同构图** (isomorphic graphs)。图 8 所示四幅图看上去完全不同，但是本质上四幅图展示的连接关系完全一致；因此，这四幅图同构，也就是等价。

观察图 8，我们会发现图中所有蓝色节点内部之间没有一条边；同样，所有黄色节点内部之间也没有一条边。所有的边都是介于蓝色、黄色节点之间。这种图叫做**二分图** (bipartite graph)，也叫二部图。

进一步仔细观察图 8，我们会发现，每个蓝色节点和每个黄色节点都存在一条边，这种特殊的二分图叫做**完全二分图** (complete bipartite graph)。

本书后续会介绍包括二部图在内的各种常见图类型。

NetworkX 有判断两个图是否同构的几个函数，请大家参考：

<https://networkx.org/documentation/stable/reference/algorithms/isomorphism.html>

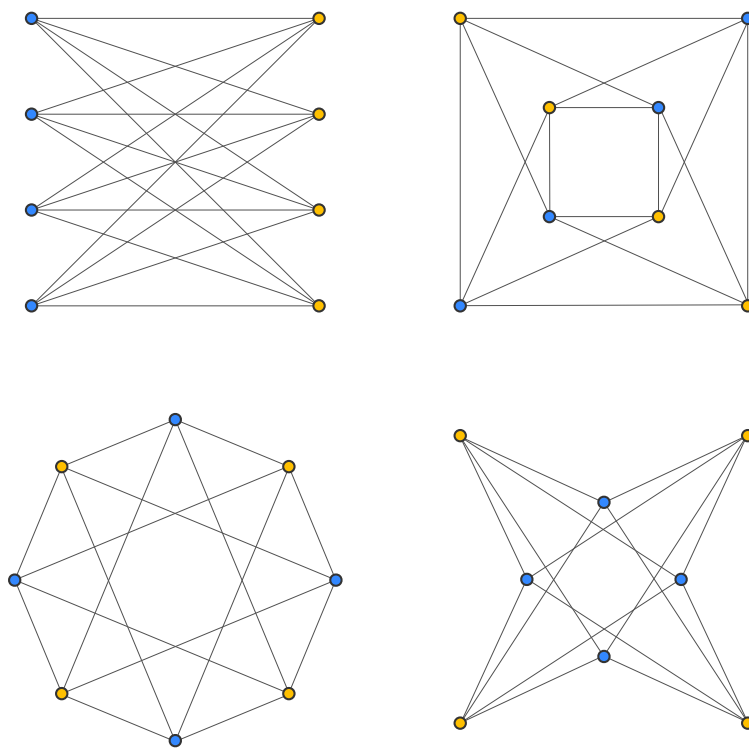


图 8. 图的同构，二分图

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

图9所示的一组图也同构。这组图也有自己的名字——**立方体图** (cubical graph)。顾名思义，立方体图和**立方体** (cube) 肯定有关。立方体，也称正六面体，是五种**柏拉图立体** (Platonic solid) 中的一种。类似地，立方体图也是**柏拉图图** (Platonic graph) 的一种。观察图9，我们还可以发现立方体图也是一种二分图，但不是完全二分图。

本书后续将专门介绍柏拉图图。



《数学要素》专门介绍过柏拉图立体，请大家回顾。

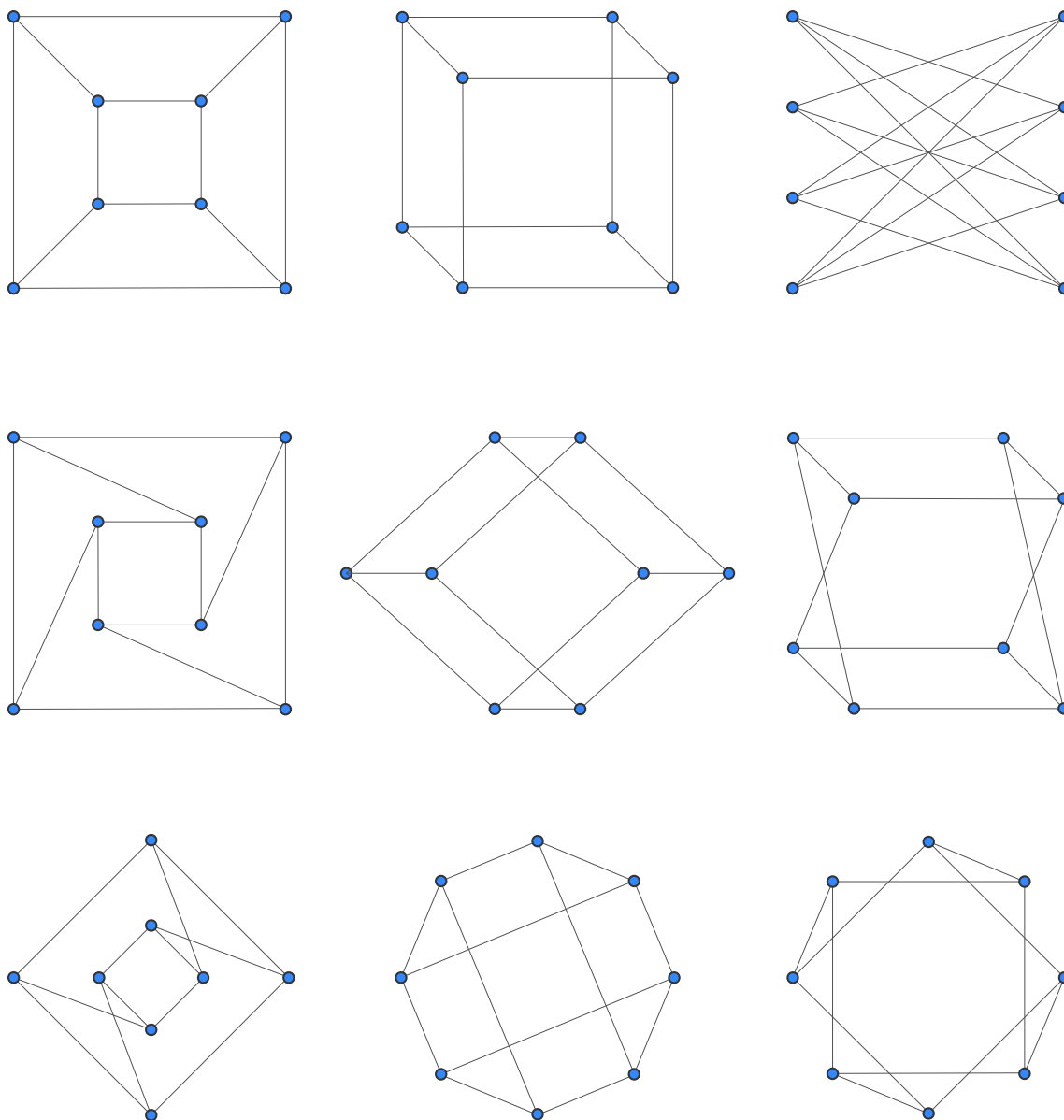


图9. 图的同构，立方体图

代码 4 判断图10两幅图是否同构，下面聊聊其中关键词句。

- a** 用 `networkx.cubical_graph()` 生成立方体图。
- b** 用 `add_edges_from()` 方法添加 12 条无向边。
- c** 用 `networkx.is_isomorphic()` 判断两幅图是否同构。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

d 用 `networkx.vf2pp_isomorphism()` 生成两幅同构图节点对应关系，结果为字典。

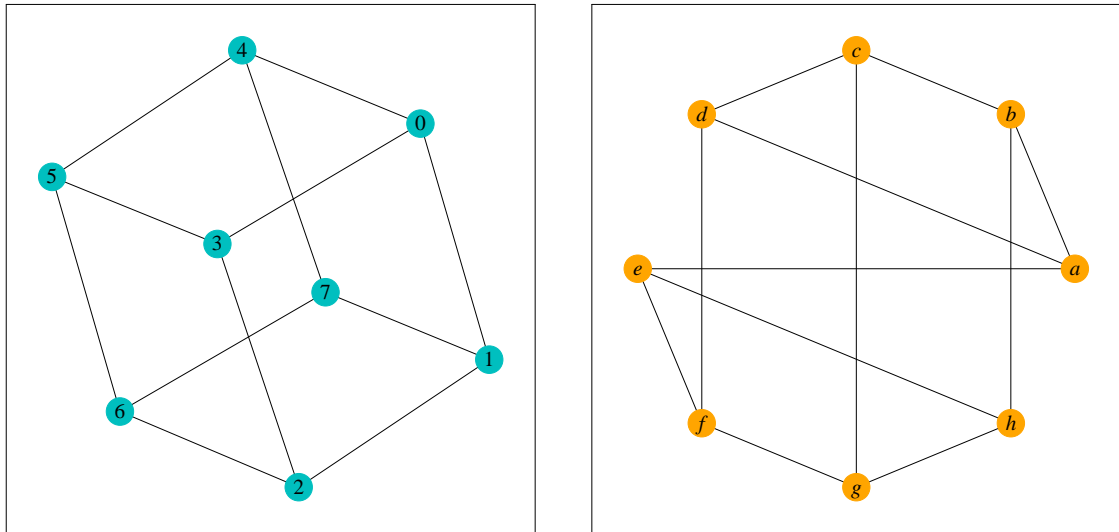


图 10. 判断两幅图是否同构

```

import networkx as nx
import matplotlib.pyplot as plt

# 第一幅图
a G = nx.cubical_graph()
# 立方体图

plt.figure(figsize = (6,6))

nx.draw_networkx(G,
                  pos = nx.spring_layout(G, seed = 8),
                  with_labels=True,
                  node_color="c")
plt.savefig('图G.svg')

# 第二幅图
b H = nx.Graph()
H.add_edges_from([('a', 'b'), ('b', 'c'),
                  ('c', 'd'), ('e', 'f'),
                  ('f', 'g'), ('g', 'h'),
                  ('b', 'h'), ('c', 'g'),
                  ('d', 'a'), ('e', 'h'),
                  ('e', 'a'), ('d', 'f')])


plt.figure(figsize = (6,6))

nx.draw_networkx(H,
                  pos = nx.circular_layout(H),
                  with_labels=True,
                  node_color="orange")
plt.savefig('图H.svg')

c nx.is_isomorphic(G,H)
# 判断是否同构

d nx.vf2pp_isomorphism(G,H, node_label="label")
# 节点对应关系

```

代码 4. 判断两幅图是否同构 |  Bk6\_Ch14\_02.ipynb

## 11.4 多图：同一对节点存在不止一条边

观察图 11，我们会发现一个有趣的现象——连接两个节点的边可能不止一条！我们管这种图叫**多图** (multigraph)。

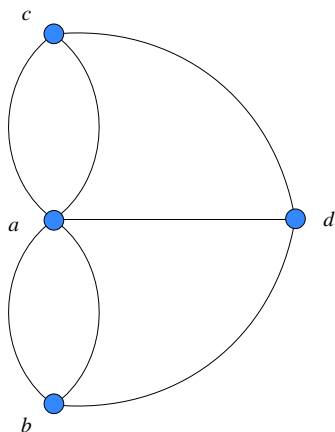


图 11. 七桥问题对应的无向图

在图论中，一个多图是一种图的扩展形式，允许在同一对节点之间存在多条边。**简单图** (simple graph) 中，任意两个节点之间只能有一条边。换句话说，一个简单图是不含自环和重边的图。

多图的定义允许图中存在**平行边** (parallel edge)，也叫**重边**，即连接相同两个节点的多条边。

如图 12 所示，在多图中，两个节点之间可以有多条边，每条边可能具有不同的权重或其他属性。对于本书前文介绍的七桥问题，显然平行边代表不同位置的桥。

多图的概念对于某些应用很有用，例如网络建模、流量分析等。在多图中，我们可以更灵活地表示节点之间复杂的关系，以及在同一对节点之间可能存在不同类型或性质的连接。

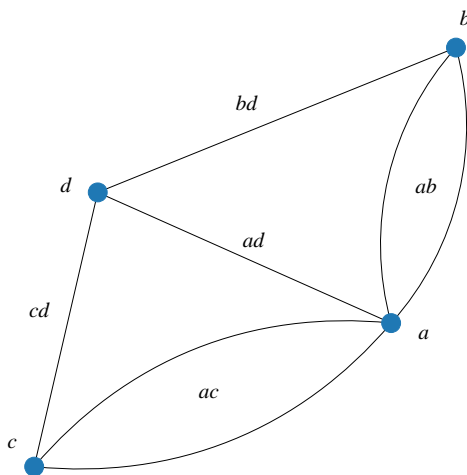


图 12. 多图

很明显节点  $a$ 、 $b$  之间的边数为 2，节点  $a$ 、 $c$  之间的边数也是 2。

代码 5 定义并可视化多图。请大家自行分析这段代码。

值得注意的是目前 `networkx.draw_networkx()` 函数还不能很好呈现多图中的平行边。图 12 中带有弧度的平行边是 NetworkX 出图后再处理的结果。在 StackOverflow 中可以找到几种解决方案，但都不是特别理想；希望 NetworkX 推出新版本时，能够解决这一问题。

```

import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

a Multi_G = nx.MultiGraph()
# 多图对象实例

b Multi_G.add_nodes_from(['a', 'b', 'c', 'd'])
# 添加多个顶点

c Multi_G.add_edges_from([('a', 'b'), # 平行边
                          ('a', 'b'),
                          ('a', 'c'), # 平行边
                          ('a', 'c'),
                          ('a', 'd'),
                          ('b', 'd'),
                          ('c', 'd')])

# 添加多条边

# 可视化
plt.figure(figsize = (6,6))
d nx.draw_networkx(Multi_G, with_labels=True)

e adjacency_matrix = nx.to_numpy_matrix(Multi_G)
# 获得邻接矩阵

```

代码 5. 定义并可视化多图 |  Bk6\_Ch14\_03.ipynb

## 11.5 子图：图的一部分

一个图的**子图** (subgraph) 是指原始图的一部分，它由图中的节点和边的子集组成。子图可以包含图中的部分节点和部分边，但这些节点和边的组合必须遵循原始图中存在的连接关系。

子图可以是原始图的任意子集。给定一个图  $G = (V, E)$ ，其中  $V$  是节点集合， $E$  是边集合。如果  $H = (V', E')$  是  $G$  的子图，则  $V'$  是  $V$  的子集， $E'$  是  $E$  的子集。

简单来说，子图是通过选择原始图中的一些节点和边而形成的一个图，保持了这些选定的节点之间的连接关系。这个过程并不创造新的节点，也不产生新的边。

图 1 中的图节点集合为  $V(G) = \{a, b, c, d\}$ ，如果选取  $V$  的子集  $\{a, b, c\}$  作为一个  $G$  子图的节点，我们便得到图 13 右侧子图。

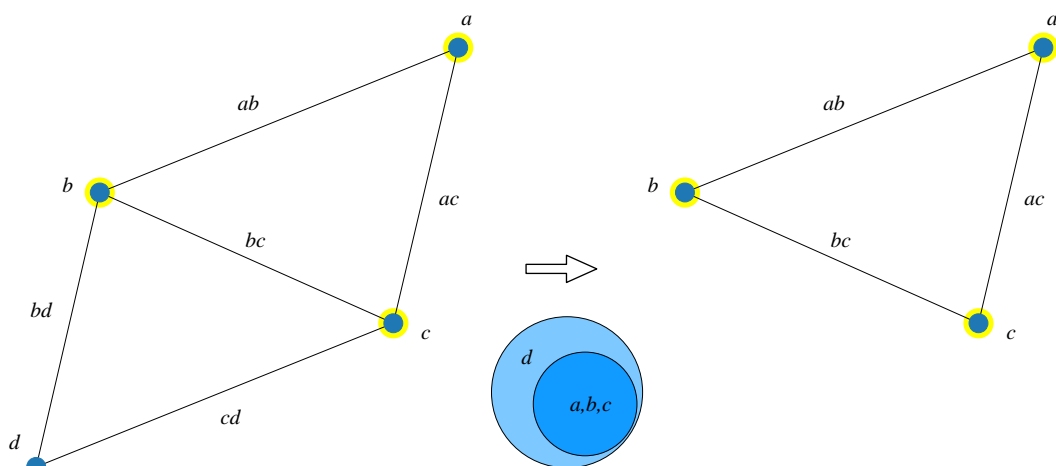


图 13. 基于节点集合子集的子图

此外，我们还可以利用图的边集合子集构造子图。图 1 中的图节点集合为  $E(G) = \{ab, ac, bc, bd, cd\}$ ，如果选取  $E$  的子集  $\{ab, bc, cd\}$  作为一个  $G$  子图的节点，我们便得到图 14 右侧子图。

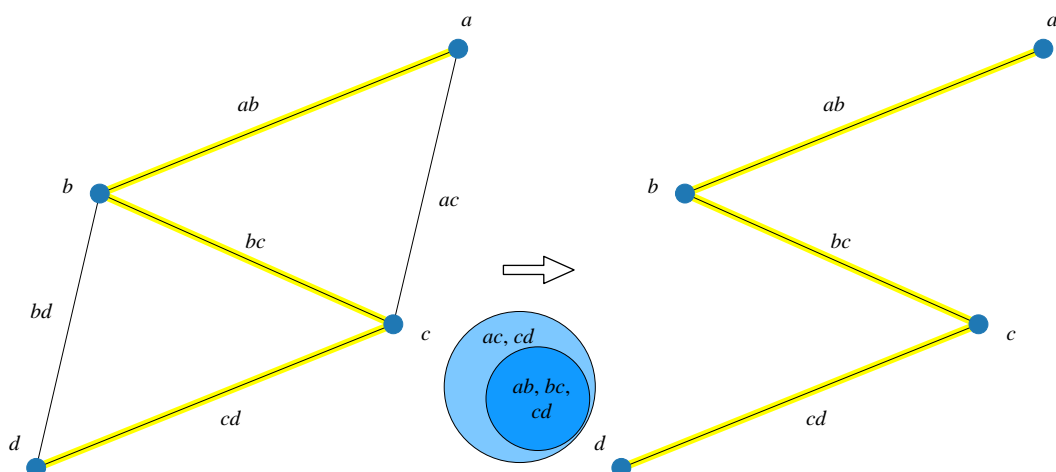


图 14. 基于边集合子集的子图

代码 6 展示如何用 NetworkX 创建子图。请大家注意以下几句。

- d** 用 `subgraph()` 方法基于节点集合子集创建子图。
- e** 计算原始图节点集合和子图节点集合之差。
- f** 用 `edge_subgraph()` 方法基于边集合子集创建子图。
- g** 计算原始图边集合和子图边集合之差。



```

import matplotlib.pyplot as plt
import networkx as nx

a G = nx.Graph()
# 创建无向图的实例

b G.add_nodes_from(['a', 'b', 'c', 'd'])
# 添加多个顶点

c G.add_edges_from([( 'a', 'b' ),
                     ( 'b', 'c' ),
                     ( 'b', 'd' ),
                     ( 'c', 'd' ),
                     ( 'c', 'a' )])
# 增加一组边

d Sub_G_nodes = G.subgraph(['a', 'b', 'c'])
# 基于节点子集的子图

e set(G.nodes) - set(Sub_G_nodes.nodes)
# 计算节点集合之差
# 结果为 {'d'}

f Sub_G_edges = G.edge_subgraph([( 'a', 'b' ),
                                   ( 'b', 'c' ),
                                   ( 'c', 'd' )])
# 基于边子集的子图

g set(G.edges) - set(Sub_G_edges.edges)
# 计算边集合之差
# 结果为 {( 'a', 'c' ), ( 'b', 'd' )}

```



代码 6. 创建子图 | Bk6\_Ch14\_04.ipynb

## 11.6 有权图：边自带权重

**有权无向图** (weighted undirected graph) 是一种图论中的数据结构，基于无向图，但在每条边上附加了一个权重或值。这个权重表示了连接两个节点之间的某种度量，例如距离、成本、时间等。相对于有权无向图，不考虑边权重的图也叫无权无向图。

每条边上的权重可以是实数或整数，用来表示相应边的重要性或其他度量。

在有权无向图中，通常通过在图的边上添有权值来模拟现实世界中的关系或约束。这种图结构在许多应用中都很实用，如网络规划、交通规划、社交网络分析等。在算法和问题解决中，有权无向图的引入使得我们能够更准确地建模和分析实际情况中的关系。

举个例子，图 15 可视化 1886 年至 1985 年的所有 685 场世界国际象棋锦标赛比赛参赛者、赛事、成绩。边宽度代表对弈的数量，点的大小代表获胜棋局数量。

图 15 这个例子来自 NetworkX 官方示例，大家可以自己学习。

[https://networkx.org/documentation/stable/auto\\_examples/drawing/plot\\_chess\\_masters.html](https://networkx.org/documentation/stable/auto_examples/drawing/plot_chess_masters.html)

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

此外，图 15 也告诉我们用 NetworkX 绘制图时，节点、边可以调整设计来展示更多有价值的信息。

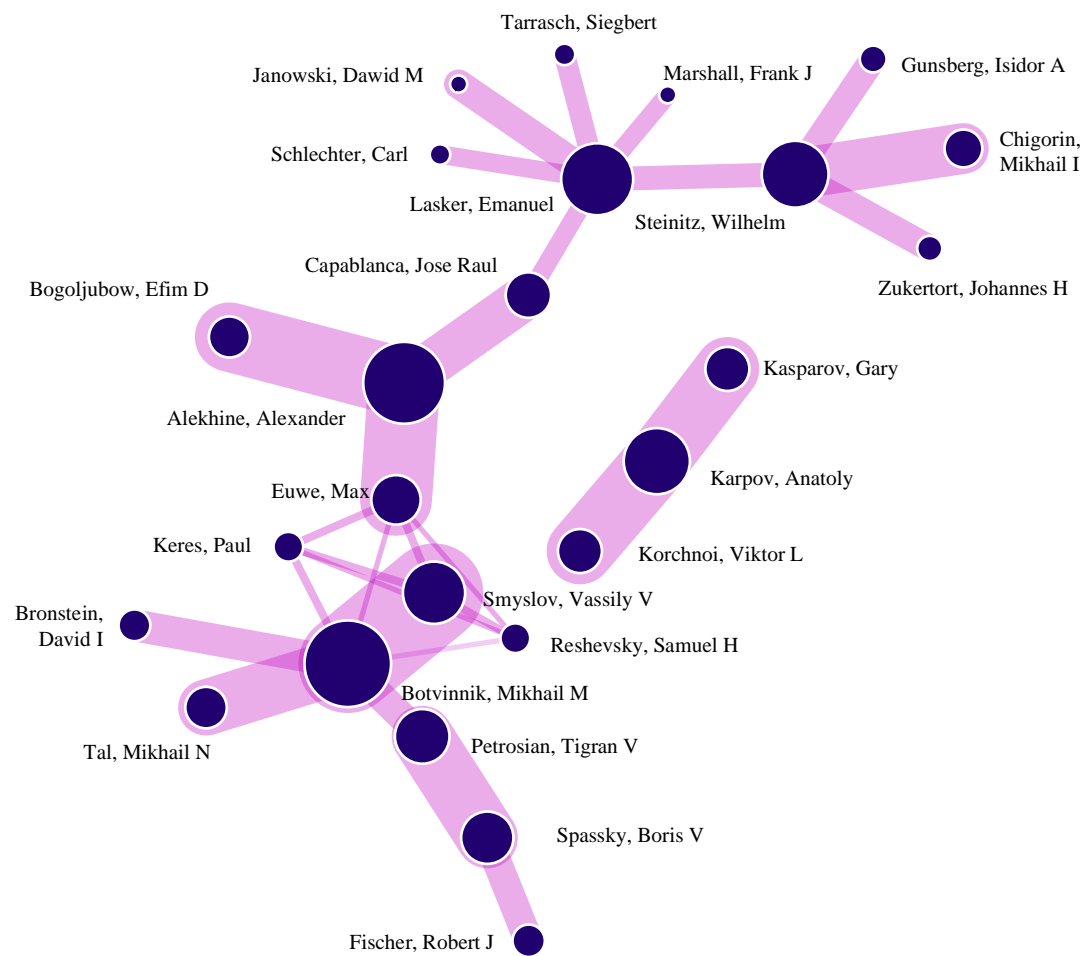


图 15. 可视化 1886 年至 1985 年的所有 685 场世界国际象棋锦标赛比赛参赛者、赛事、成绩；图片来自《可视之美》

大家应该对图 16 这幅图很熟悉了，上一章介绍过这个图。图 16 不同的是，图的每条边都有自己权重值。

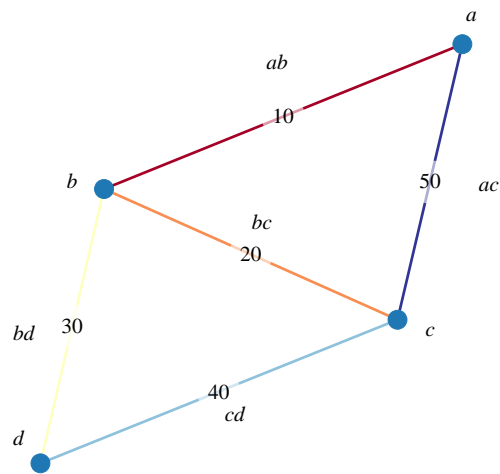


图 16. 有权无向图

下面聊聊代码 7 如何用 NetworkX 绘制有权无向图。

```
import matplotlib.pyplot as plt
import networkx as nx

a weighted_G = nx.Graph()
# 创建无向图的实例

b weighted_G.add_nodes_from(['a', 'b', 'c', 'd'])
# 添加多个顶点

c weighted_G.add_edges_from([( 'a', 'b', { 'weight':10}),
                              ( 'b', 'c', { 'weight':20}),
                              ( 'b', 'd', { 'weight':30}),
                              ( 'c', 'd', { 'weight':40}),
                              ( 'c', 'a', { 'weight':50})])

# 增加一组边，并赋予权重

d weighted_G['a']
# 取出节点a的邻居

e weighted_G['a']['b']
# 取出ab边的权重，结果为字典

f weighted_G['a']['b']['weight']
# 取出ab边的权重，结果为数值

g edge_weights = [weighted_G[i][j]['weight'] for i, j in weighted_G.edges]
# 所有边的权重

h edge_labels = nx.get_edge_attributes(weighted_G, "weight")
# 所有边的标签

plt.figure(figsize = (6,6))
i pos = nx.spring_layout(weighted_G)
j nx.draw_networkx(weighted_G,
                   pos = pos,
                   with_labels = True,
                   node_size = 180,
                   edge_color=edge_weights,
                   edge_cmap = plt.cm.RdYlBu,
                   edge_vmin = 10, edge_vmax = 50)

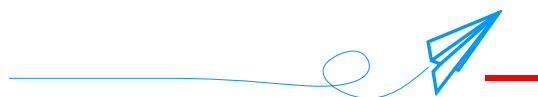
k nx.draw_networkx_edge_labels(weighted_G,
                              pos = pos,
                              edge_labels=edge_labels,
                              font_color='k')

plt.savefig('加权无向图.svg')
```

代码 7. 绘制有权无向图 |  Bk6\_Ch14\_05.ipynb

- a 用 `networkx.Graph()` 创建无向图实例。
- b 用 `add_nodes_from()` 方法添加四个节点，节点的标签分别为 'a', 'b', 'c', 'd'。

- c 用 `add_edges_from()` 方法向图中添加多条边，每条边用一个包含起点、终点和权重的元组表示。这里的权重是使用字典形式的边属性进行设置。
- d 取出节点 'a' 的邻居，返回一个邻居节点的字典。
- e 取出节点 'a' 到 'b' 的边的属性，返回一个字典，包含边的所有属性。
- f 取出节点 'a' 到 'b' 的边的权重值。
- g 创建一个列表，包含图中所有边的权重。通过遍历图中所有的边，将每条边的权重添加到列表中。
- h 用 `networkx.get_edge_attributes()` 获取图中所有边的权重作为字典。
- i 用 `networkx.spring_layout()` 计算节点的布局位置，返回一个包含节点位置信息的字典。
- j 用 `networkx.draw_networkx()` 绘制图，其中包括节点、边和边的权重。`edge_color` 参数用于指定边的颜色，根据权重值映射到 `plt.cm.RdYlBu`。`edge_vmin` 和 `edge_vmax` 指定边颜色映射的范围。
- k 用 `networkx.draw_networkx_edge_labels()` 在图上添加边的标签，这里是边的权重值。



图论是数学的一个分支，研究的是图的性质和结构以及与图相关的问题。图由节点和边组成，节点表示对象，边表示对象之间的关系。图论被广泛应用于计算机科学、网络分析、社交网络、电路设计等领域。在机器学习中，图论可以处理复杂的关系型数据，提取有用的信息，并为模型提供更深层的理解。

本章主要介绍无向图，请大家注意这些概念——阶、大小、度、邻居、端点、孤立点、自环、同构、多图、子图、有权图。下一章将专门介绍有向图，请大家对比本章阅读。