

12

Directed Graphs

有向图

由一组节点和连接节点的有向边组成结构



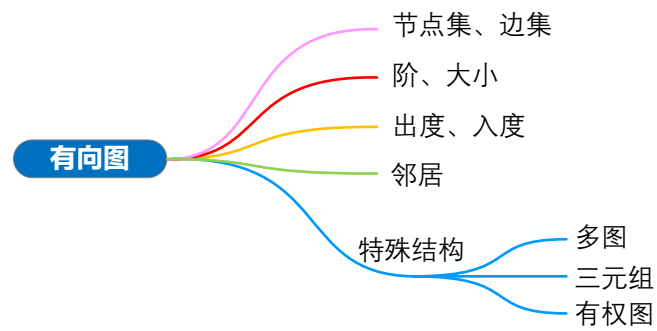
没有人是一座孤岛，汪洋中自己独踞一隅；每个人都像一块小小陆地，连接成整片大陆。

No man is an island entire of itself; every man is a piece of the continent, a part of the main.

—— 约翰·邓恩 (John Donne) | 英国诗人 | 1572 ~ 1631



- 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



12.1 有向图：边有方向

将图 1 中的有向图记做 G_D 。有向图两个重要集合：(1) 节点集 $V(G_D)$ ；(2) 有向边集 $A(G_D)$ 。因此， G_D 也常常被写成 $G_D = (V, A)$ 。

上一章提过，无向图中边集记做 $E(G)$ ， E 代表 edge；而有向图中有向边集记做 $A(G_D)$ ，它是**有向边** (directed edge) 的集合；其中， A 代表**弧** (arc，复数 arcs)，也叫 arrows，本书叫它有向边。有向边是节点的有序对。下标 D 代表 directed。

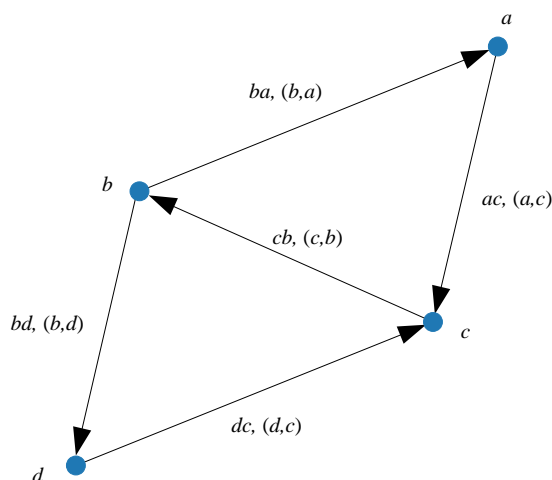


图 1. 4 个节点，5 条有向边的有向图

大家是否立刻想到“鸡兔互变”也可以抽象成一幅有向图，具体如图 2 所示。只不过图 1 的有向图无权，叫**无权有向图** (unweighted directed graph)；图 2 这幅有向图有权，叫**有权有向图** (weighted directed graph)。

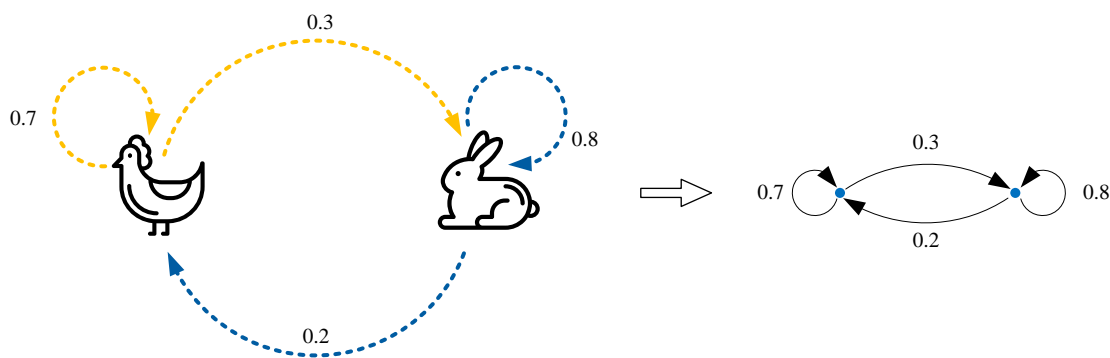


图 2. “鸡兔互变”对应的有向图

同样为了和无向图对照学习，本章采用和本书前文无向图一样的结构，不同的是每条赋予了方向。

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

以图 1 的有向图 G_D 为例， G_D 的节点集 $V(G_D)$ 为：

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

有向图 G_D 节点集和无向图并无差别。

G_D 的有向边集 $A(G_D)$ 为：

$$A(G_D) = \{ba, cb, bd, dc, ac\} = \{(b, a), (c, b), (b, d), (d, c), (a, c)\} \quad (2)$$

由于图 1 中图 G_D 是有向图，因此节点 b 到节点 a 的边 ba ，不同于节点 a 到节点 b 边 ab 。

有向边 ab 中 a 叫**头** (head)， b 叫**尾** (tail)。

图 1 是用 NetworkX 绘制，下面聊聊代码 1。

a 用 `networkx.DiGraph()` 创建一个空的有向图对象实例。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_nodes_from()` 方法增加 4 个节点。当然，我们也可以用 `add_node()` 方法增加单一节点，这和无向图一致。

c 用 `add_edges_from()` 方法向图中添加一组有向边。注意，`('a', 'b')` 不同于 `('b', 'a')`。

d 用 `networkx.draw_networkx()` 绘制有向图，传入图 `directed_G`、节点的位置信息 `pos`、箭头大小 `arrowsize`、节点的大小 `node_size`。

图 3 提供了几个供大家练习的有向图。

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

a directed_G = nx.DiGraph()
# 创建有向图的实例

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

c directed_G.add_edges_from([(('b', 'a'),
                               ('c', 'b'),
                               ('b', 'd'),
                               ('d', 'c'),
                               ('a', 'c'))])

# 增加一组有向边

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

pos = nx.spring_layout(directed_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

d plt.figure(figsize = (6,6))
nx.draw_networkx(directed_G, pos = pos,
                 arrowsize = 28,
                 node_size = 180)
plt.savefig('G_D_4顶点_5边.svg')
```

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


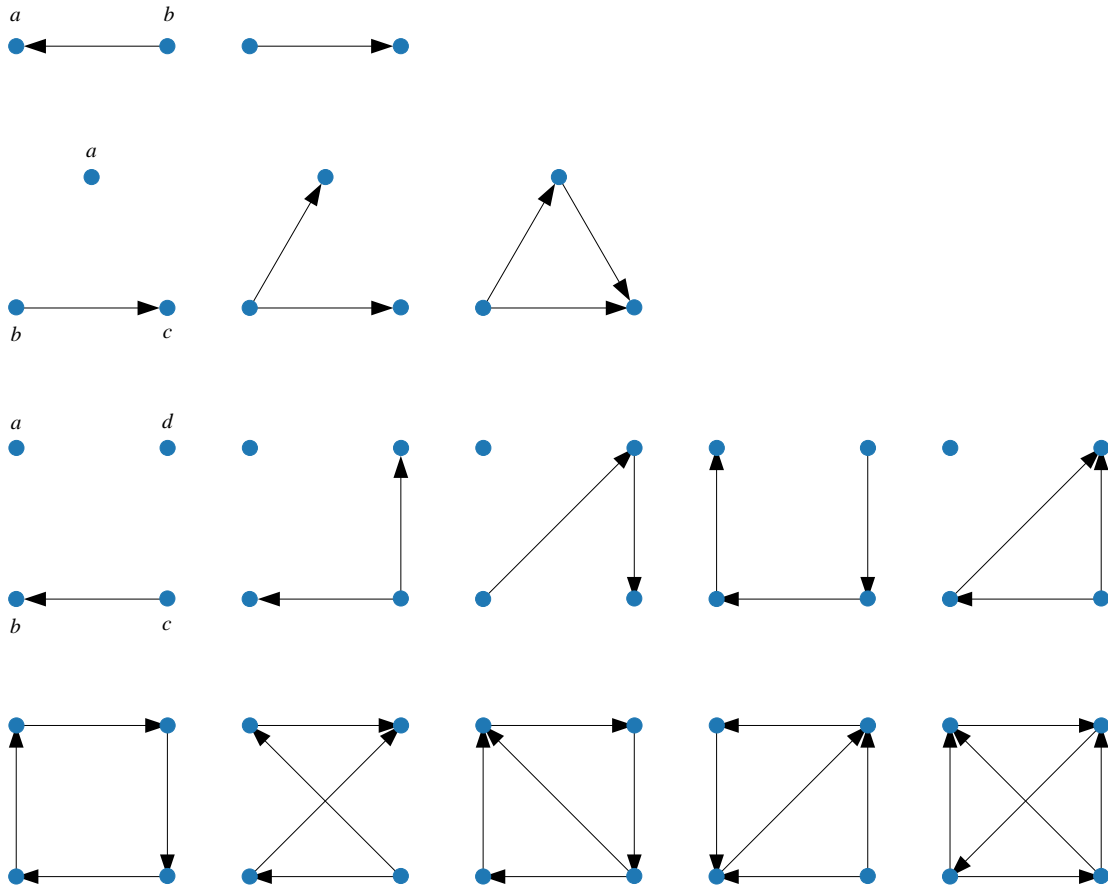
代码 1. 用 NetworkX 绘制无向图 |  Bk6_Ch12_01.ipynb

图 3. 供大家练习的几个有向图

阶、大小

和无向图一样，有向图 G_D 的节点数量叫做**阶** (order)，常用 n 表示。图 1 所示的有向图 G_D 的阶为 4 ($n = 4$)，也就是说 G_D 为 4 阶图。

和无向图一样，图 G_D 的边的数量叫做图的**大小** (size)，常用 m 表示。图 1 所示的图 G_D 的大小为 5 ($m = 5$)。

接着前文代码，代码 2 计算有向图的阶、大小等度量。请大家格外注意 **a** 和 **b**。对于有向图实例，用 `has_edge()` 判断边是否存在时，需要注意方向。

```

directed_G.order()
# 图的阶

directed_G.number_of_nodes()
# 图的节点数

directed_G.nodes
# 列出图的节点

directed_G.size()
# 图的大小

directed_G.edges
# 列出图的边


directed_G.number_of_edges()
# 图的边数

a directed_G.has_edge('a', 'b')
# 判断是否存在ab有向边

b directed_G.has_edge('b', 'a')
# 判断是否存在ba有向边

```



代码 2. 用 NetworkX 计算有向图的阶、大小 |  Bk6_Ch12_01.ipynb

12.2 出度、入度

和无向图一样，有向图任意一个节点的**度** (degree) 是与它相连的边的数量。

但是，有向图中由于边有方向，我们更关心**入度** (indegree)、**出度** (outdegree) 这两个概念。

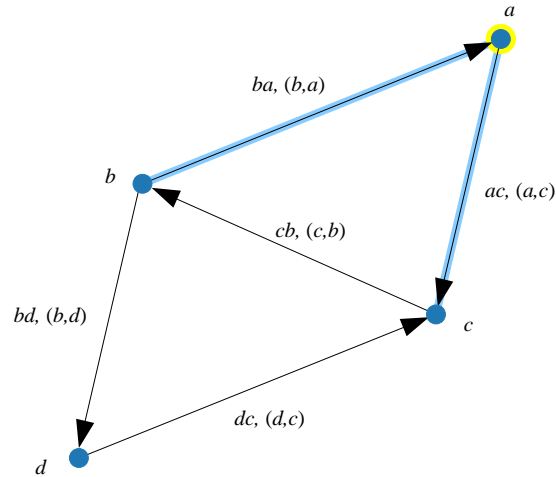
在有向图中，节点的**入度**是指指向该节点的边的数量，即从其他节点指向该节点的有向边的数量。而**出度**是指从该节点出发的边的数量，即从该节点指向其他节点的有向边的数量。这两个概念用于描述有向图中节点的连接性质，**入度**和**出度**的总和等于节点的**度数**。

比如，如图 1 所示，图 G_D 中节点 a 的度为 2，记做 $\deg_{G_D}(a) = 2$ 。

而图 G_D 中节点 a 的入度为 1，即有 1 条有向边“进入”节点 a ，记做 $\deg_{G_D}^+(a) = 1$ 。图 G_D 中节点 a 的出度为 1，即有 1 条有向边“离开”节点 a ，记做 $\deg_{G_D}^-(a) = 1$ 。显然，节点 a 的入度和出度之和为其度数

$$\deg_{G_D}(a) = \deg_{G_D}^+(a) + \deg_{G_D}^-(a) \quad (3)$$

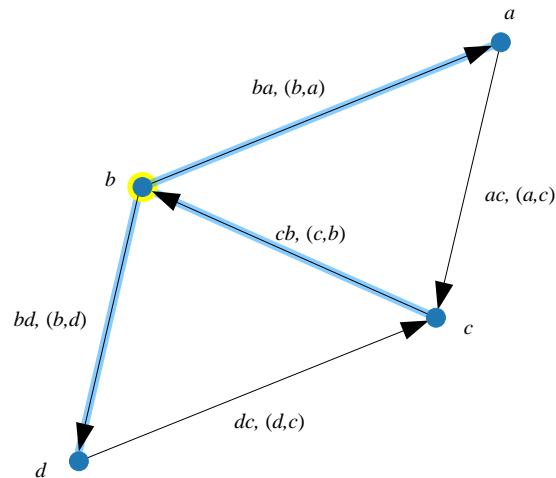
整个有向图来看，入度之和等于出度之和。

图 4. 节点 a 的度为 2，入度为 1，出度为 1

再看个例子，图 G_D 中节点 b 的度为 3，记做 $\deg_G(a) = 3$ 。

而图 G_D 中节点 b 的入度为 1，即有 1 条有向边“进入”节点 b ，记做 $\deg_{G_D}^+(b) = 1$ 。图 G_D 中节点 b 的出度为 2，即有 2 条有向边“离开”节点 b ，记做 $\deg_{G_D}^-(b) = 2$ 。

同样，入度和出度之和为度数，即 $\deg_{G_D}(b) = \deg_{G_D}^+(b) + \deg_{G_D}^-(b)$ 。

图 5. 节点 b 的度为 3，入度为 1，出度为 2

```

a directed_G.degree()
# 图的度

dict(directed_G.degree())

b directed_G.in_degree()
# 有向图的入度

c directed_G.out_degree()
# 有向图的出度

d directed_G.degree('a')
# 节点a的度

e directed_G.in_degree('a')
# 节点a的入度

f directed_G.out_degree('a')
# 节点a的出度

```

代码 3. 用 NetworkX 计算有向图的度、入度、出度 | Bk6_Ch12_01.ipynb

12.3 邻居：上家、下家

无向图中，给定特定节点的**邻居** (neighbors) 指的是与该节点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。

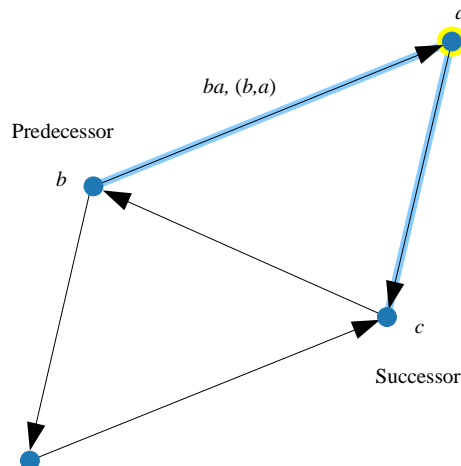
但是，在有向图中，邻居的定义则多了一层考虑——边的方向。

由于，对于任意节点的度分为入度、出度。据此，我们把邻居也分为——**入度邻居** (incoming neighbor, indegree neighbor)、**出度邻居** (outgoing neighbor, outdegree neighbor)。

入度邻居，可以理解为**上家** (predecessor)。对于节点 a 而言，入度邻居是所有指向节点 a 的节点，即节点 b 。

出度邻居，可以理解为**下家** (successor)。节点 a 的出度邻居是所节点 a 指向的节点，即节点 c 。

请大家自行分析节点 b 的邻居有哪些？入度邻居、出度邻居分别是谁？



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

图 6. 节点 a 的入度邻居、出度邻居

接着前文代码，代码 4 计算有向图节点 a 的邻居、入度邻居、出度邻居。

- a 用 `networkx.all_neighbors()` 获取有向图中节点 a 所有的邻居，包括入度、出度。
- b 对有向图 `directed_G` 节点 a 用 `neighbors()` 方法只能获取其出度邻居。
- c 对有向图 `directed_G` 节点 a 也可以用 `successors()` 方法获取其出度邻居。
- d 对有向图 `directed_G` 节点 a 用 `predecessors()` 方法获取其入度邻居。

```

a list(nx.all_neighbors(directed_G, 'a'))
  # 节点a所有邻居

b list(directed_G.neighbors('a'))
  # 节点a的（出度）邻居

c list(directed_G.successors('a'))
  # 节点a的出度邻居

d list(directed_G.predecessors('a'))
  # 节点a的入度邻居

```

代码 4. 用 NetworkX 计算有向图的邻居、入度邻居、出度邻居 | Bk6_Ch12_01.ipynb

12.4 有向多图：平行边

本书前文介绍过无向图的多图 (multigraph)，即允许在同一对节点之间存在平行边 (parallel edge)，也叫重边。

图 7 所示为用 NetworkX 绘制的有向多图。节点 a 和 b 之间有两条有向边，节点 a 和 c 之间也有两条有向边。

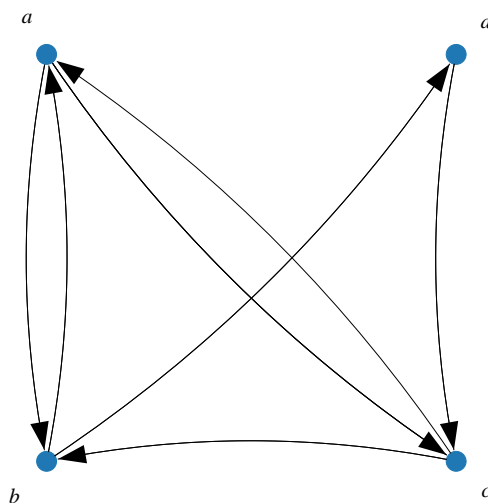


图 7. 有向多图

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

代码 5 绘制图 7。请大家格外注意 **d** 和 **e** 两句。**d** 人为设定每个节点在平面上的坐标位置。**e** 在使用 `networkx.draw_networkx()` 绘图时，通过 `pos` 参数输入每个节点坐标，利用 `connectionstyle` 将有向边设为圆弧，并指定弧度。

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

a directed_G = nx.MultiDiGraph()
# 创建有向图的实例


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

c directed_G.add_edges_from([('b', 'a'),
                              ('a', 'b'),
                              ('c', 'b'),
                              ('b', 'd'),
                              ('d', 'c'),
                              ('a', 'c'),
                              ('c', 'a')])

# 增加一组有向边

# 人为设定节点位置
d nodePosDict = {'b': [0, 0],
                  'c': [1, 0],
                  'd': [1, 1],
                  'a': [0, 1]}

e plt.figure(figsize = (6,6))
nx.draw_networkx(directed_G,
                  pos = nodePosDict,
                  arrowsize = 28,
                  connectionstyle='arc3, rad = 0.1',
                  node_size = 180)
plt.savefig('G_D_4顶点_7边.svg')
```

代码 5. 用 NetworkX 绘制有向多图 |  Bk6_Ch12_02.ipynb

12.5 三元组：三个节点的 16 种关系

图 8 所示为 16 种可能的**三元组** (triad)。三元组类型是指在社交网络或其他网络中，根据节点之间的连接关系，将节点组合成不同类型的三元组。三元组由三个节点组成，它们之间存在特定的连接模式。

图 8 中，每个三元组都有自己的标号。其中，标号的前三位数字分别表示相互、非对称、空值二元组（即双向、单向、非连接边）的数量；字母表示方向，分别是**向上** (U, up)、**向下** (D, down)、**循环** (C, cyclical) 或**传递** (T, transitive)。

三元组常用在社交网络分析中。对三元组感兴趣的读者可以参考：

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

http://www.stats.ox.ac.uk/~snijders/Trans_Triads_ha.pdf

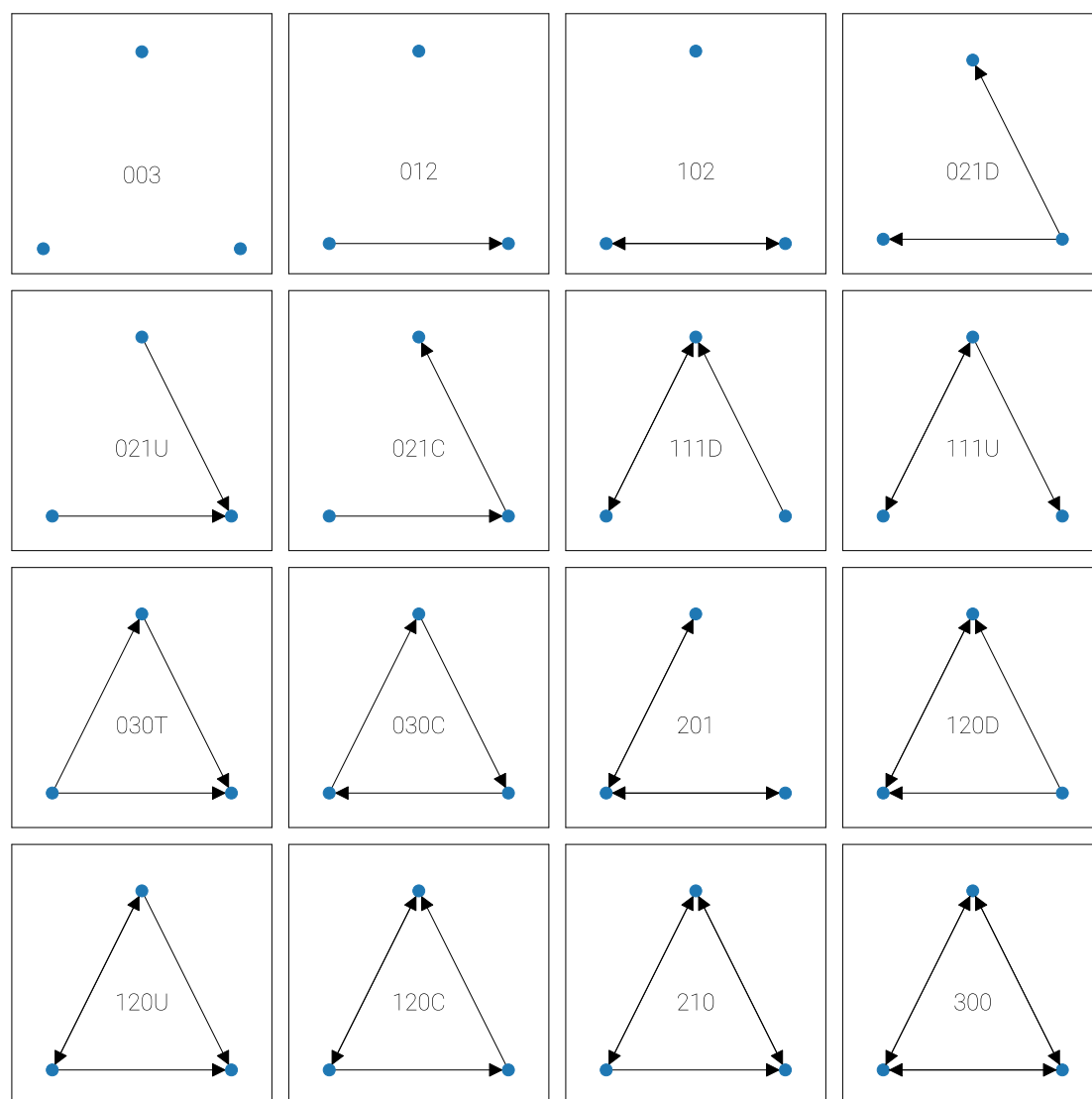


图 8. 16 种三元组类型

代码 6 绘制图 8，下面聊聊其中关键语句。

- a 创建列表，其中为 16 种三元组代号的字符串。
- b 用 `networkx.combinations()` 根据三元组代号创建图。
- c 用 `networkx.draw_networkx()` 绘制图，节点位置用 `nx.planar_layout()` 生成。
- d 用 `text()` 方法在子图轴上添加三元组代号。字号为 15 pt，字体为 Roboto Light，水平居中对齐。

```

import networkx as nx
import matplotlib.pyplot as plt

# 16种三元组的名称
a list_triads = ('003', '012', '102', '021D',
                '021U', '021C', '111D', '111U',
                '030T', '030C', '201', '120D',
                '120U', '120C', '210', '300')

# 可视化

fig, axes = plt.subplots(4, 4, figsize=(10, 10))

b for triad_i, ax in zip(list_triads, axes.flatten()):
    G = nx.triad_graph(triad_i)
    # 根据代号创建三元组
    # 绘制三元组
    c nx.draw_networkx(
        G,
        ax=ax,
        with_labels=False,
        node_size=58,
        arrowsize=20,
        width=0.25,
        pos=nx.planar_layout(G))

    ax.set_xlim(val * 1.2 for val in ax.get_xlim())
    ax.set_ylim(val * 1.2 for val in ax.get_ylim())

    # 增加三元组名称
    d ax.text(0, 0, triad_i,
            fontsize=15,
            font = 'Roboto',
            fontweight="light",
            horizontalalignment="center")

fig.tight_layout()
plt.savefig('16种三元组.svg')
plt.show()

```

代码 6. 绘制 16 种三元组 | Bk6_Ch12_03.ipynb

图 9 显然不是三元组，因为节点数为 4。但是这幅有向图却包含了 4 个三元组，具体如图 10 所示。

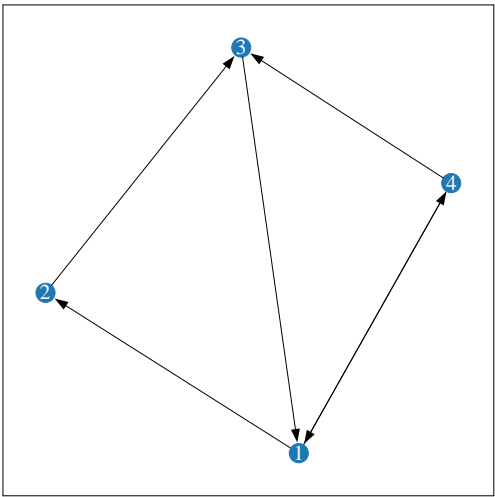


图 9. 图中包含若干三元组

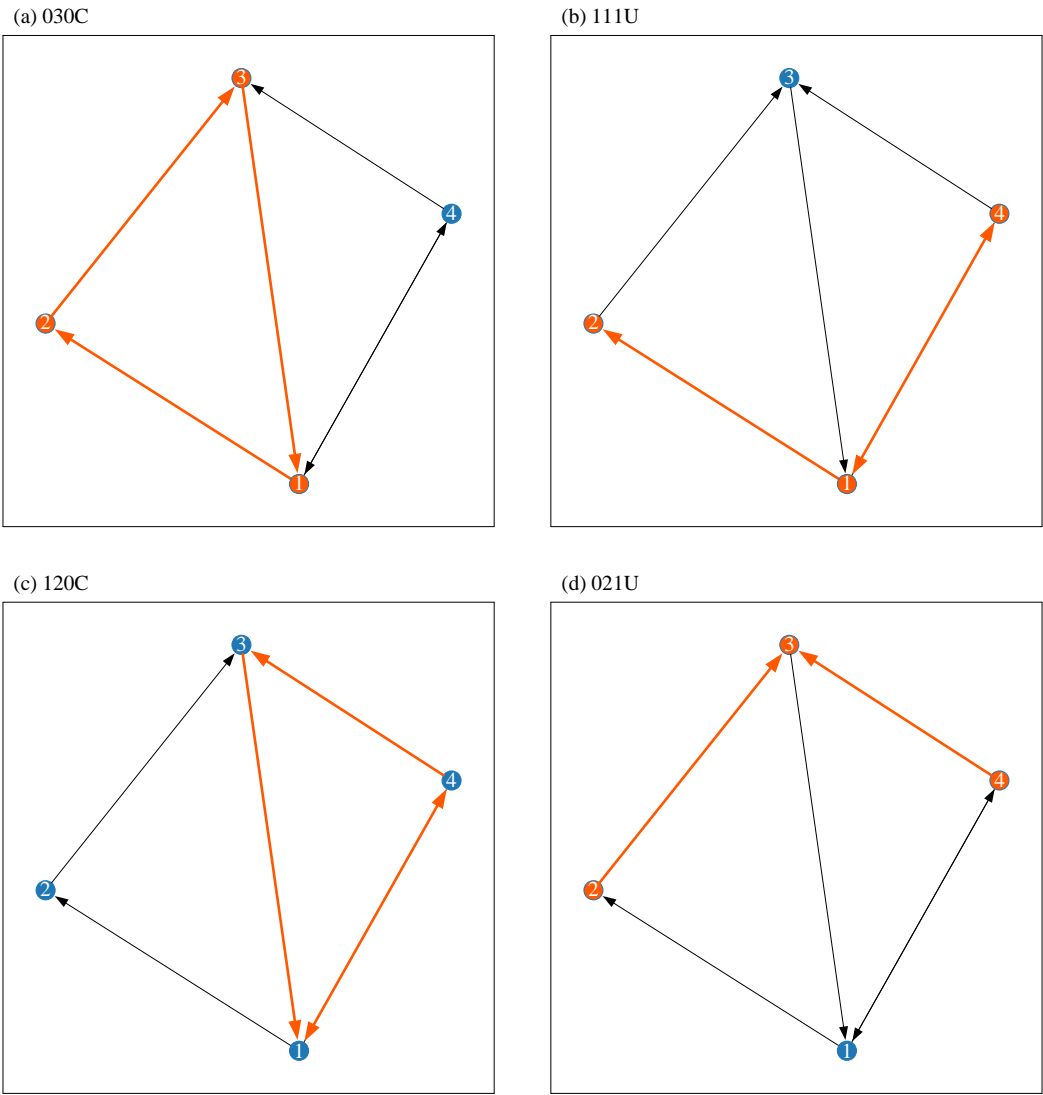


图 10. 分别可视化有向图中三元组子图

代码 7 绘制图 9 和图 10，下面聊聊其中关键语句。

- a 用 `networkx.DiGraph()` 创建有向图。
- b 用 `networkx.is_triad()` 判断有向图是否为三元组。
- c 的 `for` 循环中用 `networkx.all_triads()` 找到有向图中所有三元组子图。
- d 用 `networkx.draw_networkx_nodes()` 绘制三元组子图的节点。
- e 用 `networkx.draw_networkx_edges()` 绘制三元组子图的有向边。下一章会专门介绍 d 和 e 用到的可视化函数。

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

# 创建有向图
G = nx.DiGraph([(1, 2), (2, 3),
                (3, 1), (4, 3),
                (4, 1), (1, 4)])

pos = nx.spring_layout(G, seed = 68)

# 可视化
plt.figure(figsize=(8, 8))
nx.draw_networkx(G,
                 pos = pos,
                 with_labels = True)
plt.savefig('有向图.svg')

# 判断G是否为三元组triad
nx.is_triad(G)

# 寻找并可视化G中三元组子图

fig, axes = plt.subplots(2, 2,
                        figsize = (8,8))

axes = axes.flatten()

for triad_i, ax_i in zip(nx.all_triads(G), axes):

    nx.draw_networkx(G,
                    pos = pos,
                    ax = ax_i,
                    width=0.25,
                    with_labels = False)

    # 绘制三元组子图
    nx.draw_networkx_nodes(G, nodelist = triad_i.nodes,
                          node_color = 'r',
                          ax = ax_i,
                          pos = pos)

    nx.draw_networkx_edges(G, edgelist = triad_i.edges,
                          edge_color = 'r',
                          width=1,
                          ax = ax_i,
                          pos = pos)

    ax_i.set_title(nx.triad_type(triad_i))
plt.savefig('有向图中4个三元组子图.svg')
```


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

代码 7. 寻找图中三元组子图 |  Bk6_Ch12_04.ipynb

12.6 NetworkX 创建图

NetworkX 可以通过不同数据类型创建图，本节简单介绍常见几种数据类型。

列表

图 11 所示为通过列表数据创建的无向图、有向图。

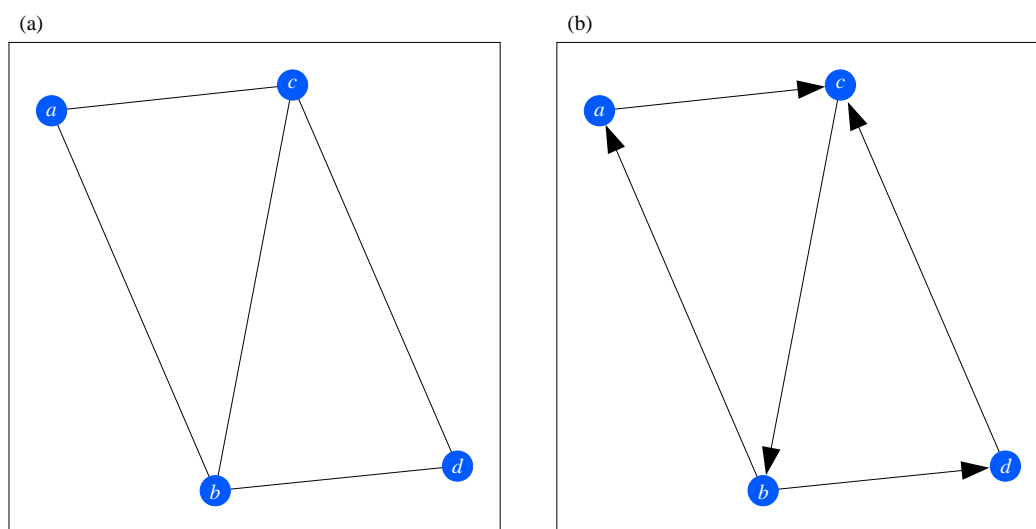


图 11. 通过列表创建的无向图、有向图

代码 8 绘制图 11，下面聊聊这段代码关键语句。

- a** 创建列表，列表元素为元组；元组中第一个字符表示始点，第二个字符表示终点。对于无向图，调转始点、终点无所谓。
- b** 用 `networkx.from_edgelist()` 从列表数据创建无向图；注意，需要通过 `create_using=nx.Graph()` 指定图的类型为无向图。其他图的类型可以是 `nx.DiGraph()`、`nx.MultiGraph()`、`nx.MultiDiGraph()` 等。
- c** 利用 `networkx.spring_layout()` 布局节点位置。
- d** 利用 `networkx.draw_networkx()` 绘制无向图。参数 `pos` 控制节点位置，参数 `node_color` 指定节点颜色，`with_labels = True` 展示节点标签，`node_size` 指定节点大小。
- e** 用 `networkx.from_edgelist()` 从列表数据创建有向图，`create_using=nx.DiGraph()` 指定图的类型为有向图。
- f** 同样利用 `networkx.draw_networkx()` 绘制有向图。

```

import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# 创建list
a edgelist = [('b', 'a'),
              ('b', 'd'),
              ('d', 'c'),
              ('c', 'b'),
              ('a', 'c')]

# 创建无向图
b G = nx.from_edgelist(edgelist,
                      create_using=nx.Graph())

# 可视化
c plt.figure(figsize = (6,6))
d pos = nx.spring_layout(G, seed = 88)
nx.draw_networkx(G,
                 pos = pos,
                 node_color = '#0058FF',
                 with_labels = True,
                 node_size = 188)

# 创建有向图
e Di_G = nx.from_edgelist(edgelist,
                        create_using=nx.DiGraph())

# 可视化
f plt.figure(figsize = (6,6))
nx.draw_networkx(Di_G,
                 pos = pos,
                 node_color = '#0058FF',
                 with_labels = True,
                 node_size = 188)

```

代码 8. 利用列表数据创建图 | Bk6_Ch12_05.ipynb

数据帧

图 12 所示为通过数据帧创建的无向图、有向图。这两幅图用表 1 所示的数据帧。这个数据帧有 4 列，第 1、2 列分别代表边的起点、终点（当然，对于无向图，这两列数据都视作节点，并无差别）。第 3 列 edge_key 是边的名称，第 4 列 weight 是边的权重。

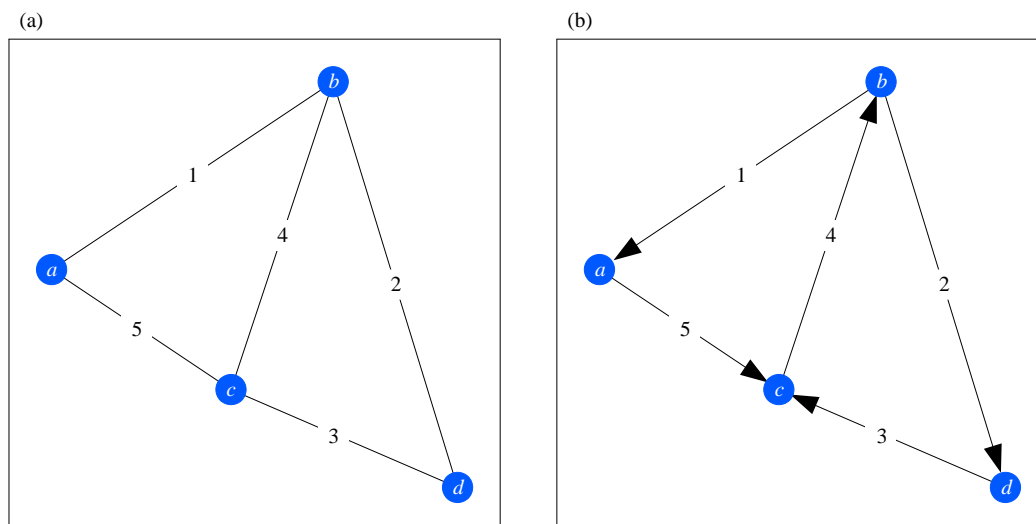


图 12. 通过数据帧创建的无向图、有向图

表 1. 用来创建无向图、有向图的数据帧

	source	target	edge_key	weight
0	<i>b</i>	<i>a</i>	<i>ba</i>	1
1	<i>b</i>	<i>d</i>	<i>bd</i>	2
2	<i>d</i>	<i>c</i>	<i>dc</i>	3
3	<i>c</i>	<i>b</i>	<i>cb</i>	4
4	<i>a</i>	<i>c</i>	<i>ac</i>	5

代码 9 绘制图 12，下面聊聊其中关键语句。

a 用 `pandas.DataFrame()` 创建数据帧，4 列、5 行。

b 用 `networkx.from_pandas_edgelist()` 从数据帧创建无向图。

参数 `source = "source"` 指定始点对应的列，参数 `target = "target"` 指定终点对应的列；对于无向图，始点、终点顺序不重要。

参数 `edge_key="edge_key"` 指定边的标签对应的列，参数 `edge_attr=["weight"]` 指定边的权重。

参数 `create_using=nx.Graph()` 确定创建的是无向图。

c 用 `networkx.get_edge_attributes()` 获取无向图 `G` 的边权重作为边标签。

d 用 `networkx.draw_networkx()` 绘制无向图。大家可以尝试使用 `networkx.draw()` 绘制图；此外，本书后续会介绍其他绘制无向图方法。

e 用 `networkx.draw_networkx_edge_labels()` 在图上增加边标签。

f 用 `networkx.from_pandas_edgelist()` 从数据帧创建有向图。

参数 `create_using=nx.DiGraph()` 确定创建的是有向图。

```

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

# 创建数据帧
a edges_df = pd.DataFrame({
    'source': ['b', 'b', 'd', 'c', 'a'],
    'target': ['a', 'd', 'c', 'b', 'c'],
    'edge_key': ['ba', 'bd', 'dc', 'cb', 'ac'],
    'weight': [1, 2, 3, 4, 5]})

# 创建无向图
b G = nx.from_pandas_edgelist(
    edges_df,
    source = "source",
    target = "target",
    edge_key="edge_key",
    edge_attr=["weight"],
    create_using=nx.Graph())

# 边权重
c G_edge_labels = nx.get_edge_attributes(G, "weight")

# 可视化
d plt.figure(figsize = (6,6))
pos = nx.spring_layout(G, seed = 28)
nx.draw_networkx(G,
    pos = pos,
    node_color = '#0058FF',
    with_labels = True,
    node_size = 188)
e nx.draw_networkx_edge_labels(G, pos,
    G_edge_labels)

plt.savefig('无向图.svg')

# 创建有向图
f Di_G = nx.from_pandas_edgelist(
    edges_df,
    source = "source",
    target = "target",
    edge_key="edge_key",
    edge_attr=["weight"],
    create_using=nx.DiGraph())

# 边权重
Di_G_edge_labels = nx.get_edge_attributes(Di_G, "weight")

# 可视化
plt.figure(figsize = (6,6))
nx.draw_networkx(Di_G,
    pos = pos,
    node_color = '#0058FF',
    with_labels = True,
    node_size = 188)
nx.draw_networkx_edge_labels(Di_G, pos,
    Di_G_edge_labels)

plt.savefig('有向图.svg')

```

代码 9. 利用数据帧数据创建图 | Bk6_Ch12_06.ipynb

图 12 也同时告诉我们有向图也可以是有权图，即**有权有向图** (weighted directed graph)。图 13 比较四种图——**无权无向图** (unweighted undirected graph)、**有权无向图** (weighted undirected graph)、**无权有向图** (unweighted directed graph)、**有权有向图** (weighted directed graph)。

无权无向图、无权有向图合称为**无权图** (unweighted graph)；有权无向图、有权有向图合称为**有权图** (weighted graph)。

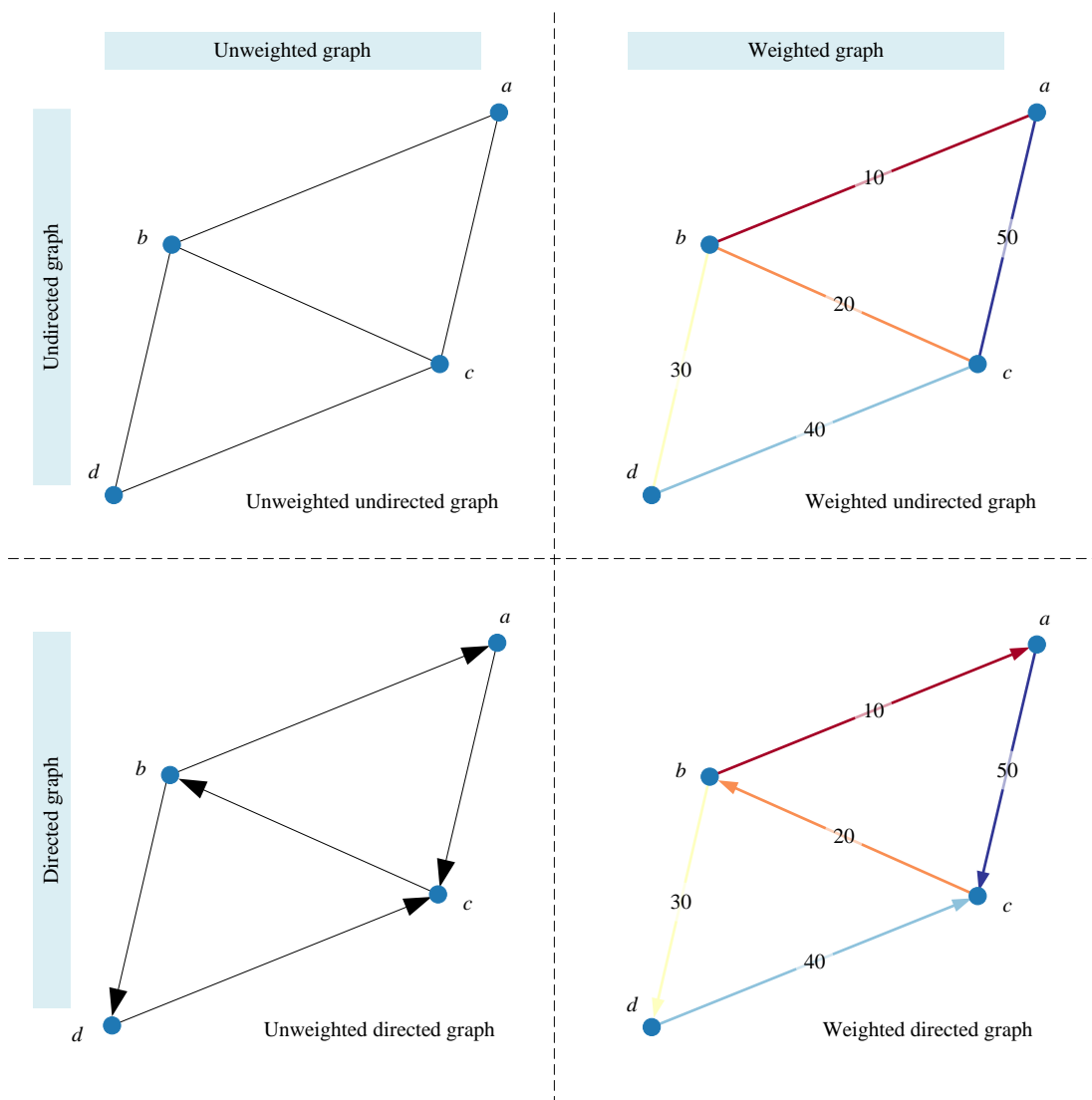


图 13. 比较四种图

NumPy 数组

NetworkX 也允许通过 NumPy 数组创建无向图、有向图；只不过相比前面介绍的两种创建图的方法，NumPy 数组的结构显得“别有洞天”！

如下数据是用来构造图 11 (a) 无向图的矩阵

$$\begin{array}{c}
 \begin{array}{ccccc}
 & a & b & c & d \\
 a & \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} \\
 b & \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} \\
 c & \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix} \\
 d & \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}
 \end{array}
 \end{array} \quad (4)$$

矩阵为 4 行、4 列。矩阵的每行代表 4 个节点， a 、 b 、 c 、 d ；矩阵的每列也代表 4 个节点， a 、 b 、 c 、 d 。矩阵中 1 代表存在一条边，0 代表不存在边。比如，矩阵中第 1 行、第 2 列元素为 1，代表节点 a 、 b 之间存在一条边。

显然，(4) 为对称矩阵；这是因为，无向图一条边的两个端点可以调换顺序。

如下数据是用来构造图 11 (b) 有向图的矩阵

$$\begin{array}{c}
 \begin{array}{ccccc}
 & a & b & c & d \\
 a & \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \\
 b & \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \\
 c & \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \\
 d & \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}
 \end{array}
 \end{array} \quad (5)$$

图 14 的无向图有 5 条边，(5) 矩阵有 5 个 1。图 14 展示了 (5) 矩阵和有向图之间关系。请大家自行指出有向边 cb 对应哪个元素。

很明显 (5) 这个矩阵不对称。

看到这里大家是否想到本书前文说过的一句话——图就是矩阵，矩阵就是图！

而 (4) 和 (5) 所示的矩阵有自己的名字——**邻接矩阵** (adjacency matrix)，这是本书后文要介绍的重要内容之一。

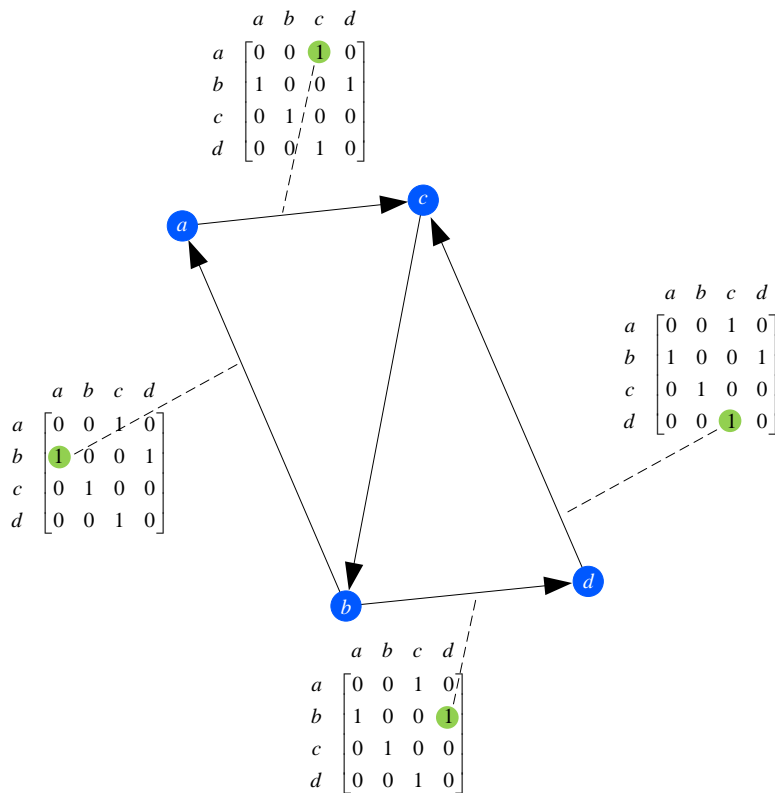


图 14. 通过邻接矩阵创建的有向图

代码 10 利用 NumPy 数组（邻接矩阵）创建图 11 中无向图、有向图，下面聊聊其中关键语句。

- a 用 `numpy.array()` 创建 NumPy 数组，代表无向图的邻接矩阵。
- b 利用 `networkx.from_numpy_array()` 根据邻接矩阵创建无向图，需要通过参数 `create_using=nx.Graph` 指定创建无向图。
- c 创建字典用于节点标签映射。无向图默认节点标签为 0 开始的非负整数。
- d 用 `networkx.relabel_nodes()` 修改无向图节点标签。
- e 用 `numpy.array()` 创建 NumPy 数组，代表有向图的邻接矩阵。
- f 利用 `networkx.from_numpy_array()` 根据邻接矩阵创建有向图，需要通过参数 `create_using=nx.DiGraph` 指定创建有向图。
- g 也用 `networkx.relabel_nodes()` 修改有向图节点标签。

```
import pandas as pd
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

a matrix_G = np.array([[0, 1, 1, 0],
                      [1, 0, 1, 1],
                      [1, 1, 0, 1],
                      [0, 1, 1, 0]])
# 定义无向图邻接矩阵


# 用邻接矩阵创建无向图
b G = nx.from_numpy_array(matrix_G,
                        create_using=nx.Graph)

# 修改节点标签
c mapping = {0: "a", 1: "b", 2: "c", 3: "d"}
d G = nx.relabel_nodes(G, mapping)

e matrix_Di_G = np.array([[0, 0, 1, 0],
                        [1, 0, 0, 1],
                        [0, 1, 0, 0],
                        [0, 0, 1, 0]])
# 定义有向图邻接矩阵

# 用邻接矩阵创建有向图
f Di_G = nx.from_numpy_array(matrix_Di_G,
                          create_using=nx.DiGraph)

# 修改节点标签
g Di_G = nx.relabel_nodes(Di_G, mapping)
```

代码 10. 利用 NumPy 数组（邻接矩阵）创建图 |  Bk6_Ch12_07.ipynb

大家可能会好奇既然我们可以用所谓的邻接矩阵来表达图 11 的无权无向图、有向图，能不能也用类似的矩阵形式表达图 12 中有权无向图、有向图？答案是肯定的！

代码 11 便可以创建图 12 中有权无向图、有向图，请大家自行分析这段代码。

```

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

a matrix_G = np.array([[0, 1, 5, 0],
                        [1, 0, 4, 2],
                        [5, 4, 0, 3],
                        [0, 2, 3, 0]])
# 定义无向图邻接矩阵

# 用邻接矩阵创建无向图
b G = nx.from_numpy_array(matrix_G,
                           create_using=nx.Graph)


# 修改节点标签
c mapping = {0: "a", 1: "b", 2: "c", 3: "d"}
d G = nx.relabel_nodes(G, mapping)

e matrix_Di_G = np.array([[0, 0, 5, 0],
                           [1, 0, 0, 2],
                           [0, 4, 0, 0],
                           [0, 0, 3, 0]])
# 定义有向图邻接矩阵

# 用邻接矩阵创建有向图
f Di_G = nx.from_numpy_array(matrix_Di_G,
                              create_using=nx.DiGraph)

# 修改节点标签
g Di_G = nx.relabel_nodes(Di_G, mapping)

```

代码 11. 利用 NumPy 数组（邻接矩阵）创建有权图 |  Bk6_Ch12_08.ipynb

在图论中，图可以分为无向图和有向图两种基本类型。无向图中的边没有方向，即连接两个节点的边不区分起点和终点。有向图中的边有方向，即连接两个节点的边有明确的起点和终点。

下一章将专门讲解用 NetworkX 可视化图。