

作者	生姜 DrGinger
脚本	生姜 DrGinger
视频	崔崔 CuiCui
开源学习资源	https://github.com/Visualize-ML
平台	https://www.youtube.com/@DrGinger_Jiang https://space.bilibili.com/3546865719052873 https://space.bilibili.com/513194466

15

Graph Theory and Network

图论和网络

无向图、有向图、邻接矩阵、拉普拉斯矩阵

图论 (graph theory) 是研究由节点 (nodes) 和边 (edges) 组成的图结构的数学领域。机器学习、人工智能，甚至日常生活中，图、网络无处不在。而矩阵、矩阵分解是描述和分析图、网络的绝佳工具。

本章相当于图论和网络的入门教程，本书前文讲过的各种线性代数工具随处可见！

15.1 无向图



本节你将掌握的核心技能：

- ▶ 无向图中的边没有方向，连接是对称的。
- ▶ 无向图的邻接矩阵：对称矩阵。
- ▶ 邻接矩阵的幂表示节点之间经过多个中介节点的路径数量。
- ▶ 邻接矩阵行列求和计算节点度。
- ▶ 有权无向图：边含有权重。

本节介绍无向图，下一节讲解有向图，建议平行阅读两节。

无向图

无向图 (undirected graph) 是指边没有方向的图，边只表示两个节点之间的连接关系。也就是说，如果两个节点之间存在一条边，则意味着它们彼此相连，连接是对称的。

举个例子，如图 1 所示，公园一共有 4 个景点，每个景点可以看作是一个**节点**。图 1 这幅图一共有四个节点： a 、 b 、 c 、 d 。

景点之间若存在一条徒步小径，对应的**节点**就画一条线段连接，这条线段就是**边**。比如，节点 a 、 b 之间存在一条徒步小径，对应节点 ab 。显然，边 ab 没有方向，因为我们可以从 a 走到 b ；反过来，也可以从 b 走到 a 。这也就是为什么图 1 叫做无向图的原因。

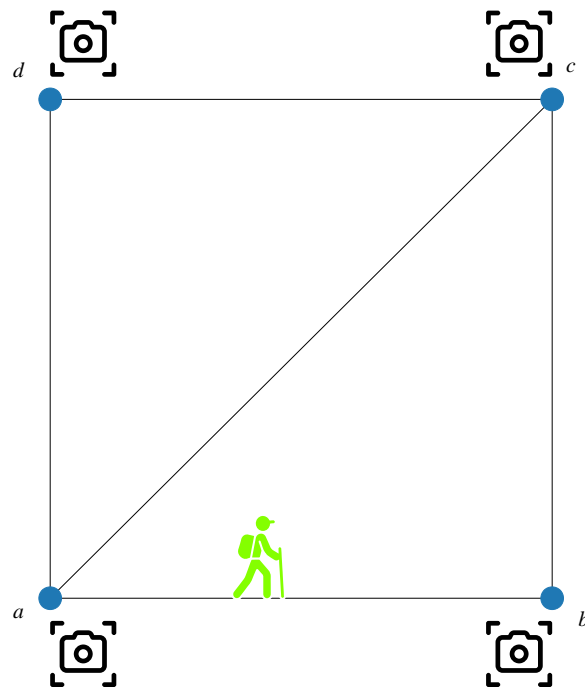


图 1. 公园景点之间构成的无向图

NetworkX 绘制无向图

代码 1 用 NetworkX 绘制图 1 这副无向图。下面聊聊其中关键词句。

a 导入一个专门用于处理图结构的库，叫 NetworkX。它支持各种图（比如无向图、有向图、带权图等）的创建、操作和分析，这里我们给它取个简写叫 `nx`，后面所有用到 NetworkX 的函数都通过 `nx.xxx()` 来调用。

b 创建一个空的无向图对象。所谓无向图，就是图中每条边没有方向，比如“ a 和 b 相连”表示两者之间可以互相到达，没有先后之分。这个图被命名为 `undirected_G`，你可以理解成是一个容器，我们后面会往这个容器里添加节点和边。

c 用 `add_nodes_from(['a','b','c','d'])` 添加四个节点，分别是 ' a '、' b '、' c ' 和 ' d '。这些节点是图中最基本的元素，就像社交网络中的用户，地图中的地点，网络上的网页。函数 `add_nodes_from()` 可以同时添加多个节点，用列表传进去就可以。

d 用 `add_edges_from([('a','b'), ('a','c'), ('a','d'), ('b','c'), ('c','d')])` 给无向图添加边。每条边连接两个节点，比如 `('a','b')` 表示 `a` 和 `b` 之间有一条边。这个函数也能批量添加多条边，方式是传入一个边的列表。注意，因为我们之前创建的是无向图，所以 `('a','b')` 和 `('b','a')` 是一样的。

e 为图中的每个节点指定一个在二维平面中的位置。它创建了一个字典，键是节点的名字，值是它们的位置坐标。比如 `'a'` 的位置是 `[0, 0]`，表示它放在图的左下角。这一步的目的是让图在绘图时看起来更清晰、整齐，位置由我们手动控制。


f 用 `nx.draw_networkx()` 绘制之前代码定义的无向图。

参数 `undirected_G` 是我们要绘制的图对象。

`pos = pos` 告诉它每个节点应该放在哪里（就是前面定义的坐标）。

`node_size = 188` 控制每个节点圆圈的大小，数值越大节点越大。

默认情况下，函数会自动画出节点、边，并在节点旁边加上名字。

代码 1. 用 Networkx 绘制无向图 |  LA_15_01_01.ipynb

```
## 初始化
import matplotlib.pyplot as plt
a import networkx as nx

## 无向图
b undirected_G = nx.Graph()

## 添加节点
c undirected_G.add_nodes_from(['a', 'b', 'c', 'd'])

## 添加边
d undirected_G.add_edges_from([('a', 'b'),
                                ('a', 'c'),
                                ('a', 'd'),
                                ('b', 'c'),
                                ('c', 'd')])

## 定义节点坐标
e pos = {'a': [0, 0],
         'b': [1, 0],
         'c': [1, 1],
         'd': [0, 1]}

## 可视化
f plt.figure(figsize = (6,6))
    nx.draw_networkx(undirected_G, pos = pos, node_size = 188)
```

邻接矩阵

图 1 这幅无向图可以通过邻接矩阵 (adjacency matrix) 来表示。

简单来说，无向图的邻接矩阵是用来表示图中节点之间连接关系的一种矩阵形式。对于一个有 n 个节点的无向图，它的邻接矩阵是一个 $n \times n$ 的对称矩阵，记作 A 。

比如，图 1 无向图的邻接矩阵

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad (1)$$

其中第 i 行、第 j 列的元素 a_{ij} 表示节点 i 和节点 j 之间是否存在一条边。

如图 2 (a) 所示，无向图节点 a 、 b 之间存在一条无向边，因此矩阵 A 的第 1 行第 2 列元素为 1；由于无向图边没有方向，所以矩阵 A 的第 2 行第 1 列元素也是 1。

如图 2 (f) 所示，无向图节点 c 、 d 之间不存在无向边，因此矩阵 A 的第 3 行第 4 列元素为 0；矩阵 A 的第 4 行第 3 列元素也是 0。

大家可能已经注意到，(1) 中矩阵 A 的主对角线元素均为 0，这是因为图 1 每个节点不存在节点到自身的边。这种边如果存在的话叫做自环 (self-loop)。

简单来说，自环是指一条连接某个节点自身的边。打个比方，如果景点 a 有自环，说明有一条从 a 出发又回到 a 的徒步小径。

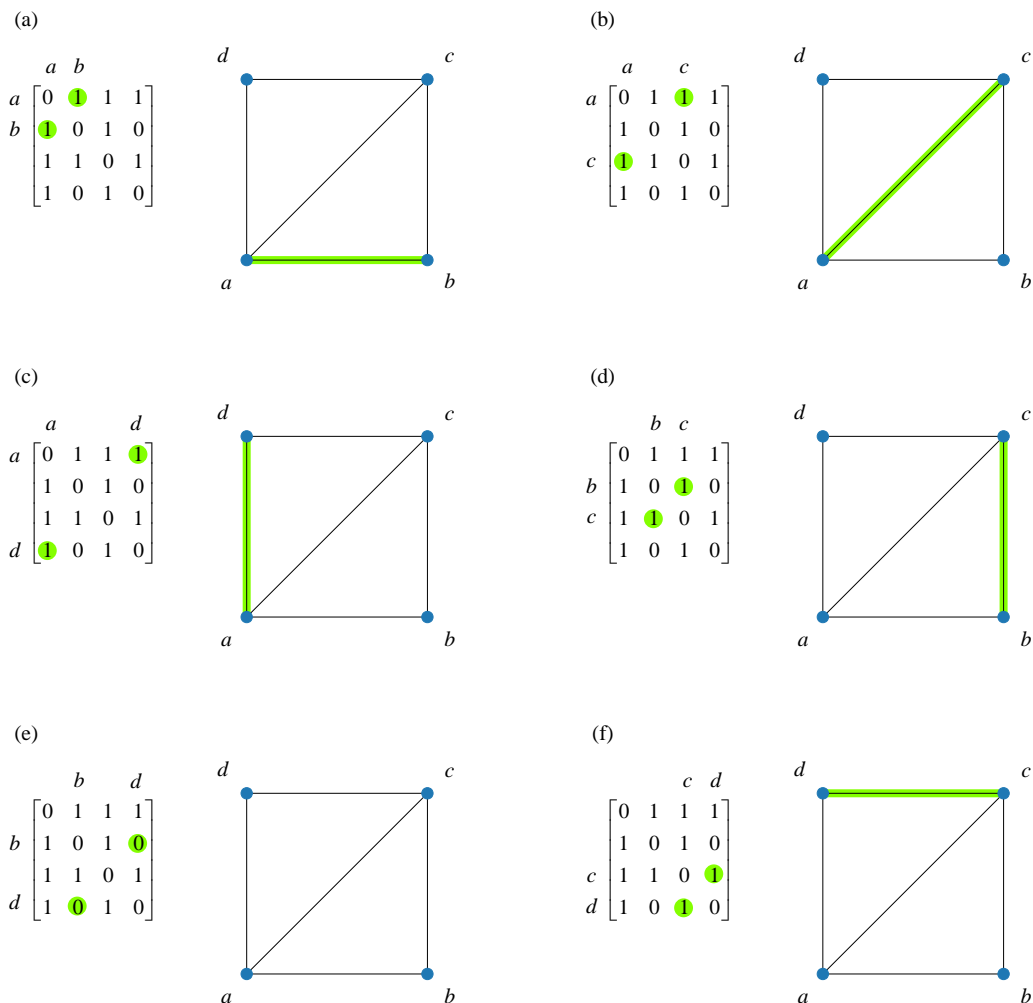


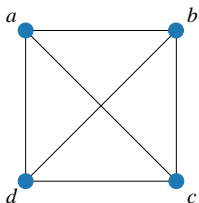
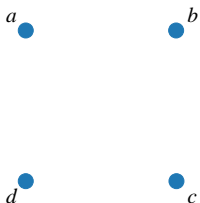
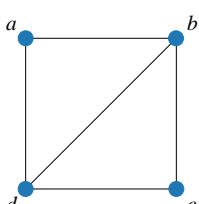
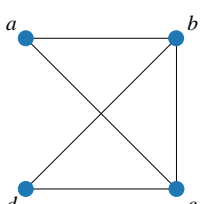
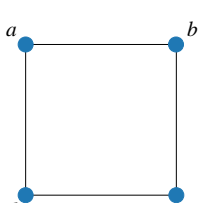
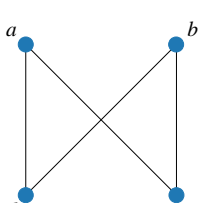
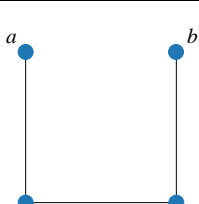
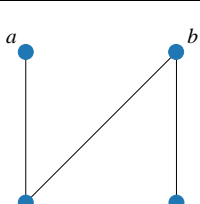
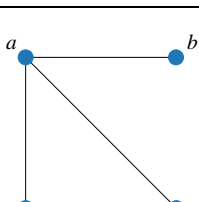
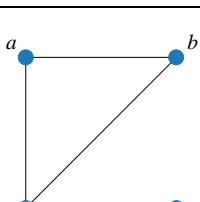
图 2. 无向图邻接矩阵的每个元素

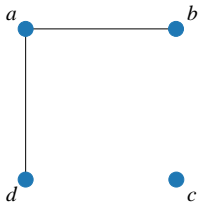
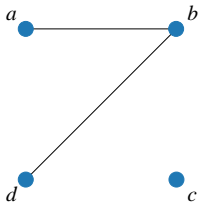


LA_15_01_01.ipynb 还用 `networkx.adjacency_matrix()` 计算无向图的邻接矩阵。

? 请大家逐一分析表 1 中每个无向图和邻接矩阵。

表 1. 4 个节点构造的几种无向图及邻接矩阵，表格示例来自《数据有道》

无向图	邻接矩阵	无向图	邻接矩阵
	$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
	$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$
	$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$
	$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$
	$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$

	$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$
---	--	--	--

从邻接矩阵到无向图

代码 2 展示用邻接矩阵（二维数组）构造无向图。下面聊聊关键语句。


a 用 `numpy.array()` 定义了一个 4×4 的邻接矩阵，表示一个图的结构。因为图是无向的，邻接矩阵是对称矩阵。

b 调用 NetworkX 的 `Graph` 函数，用邻接矩阵创建了一个无向图。这个函数会自动根据矩阵中为 1 的位置加边。`nodetype=int` 表示节点的编号是整数，比如 0、1、2、3。图创建后存在变量 `undirected_G` 中。

c 构造了一个字典，表示将原本的节点编号 (0, 1, 2, 3) 重新命名成字母 (a, b, c, d)。这样可以图更易读，便于在图中标注节点。

d 用 NetworkX 的 `relabel_nodes` 函数，把图中的节点编号按上面的 `mapping` 字典改名了。改完之后，图里的节点就不再是数字，而是字母，比如节点 0 变成了节点 'a'。

剩余的代码，本节前文已经讲过，请大家逐句注释学习。

代码 2. 从邻接矩阵到无向图 |  LA_15_01_02.ipynb

```

## 初始化
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

## 定义邻接矩阵
a adjacency_matrix = np.array([[0, 1, 1, 1],
                                [1, 0, 1, 0],
                                [1, 1, 0, 1],
                                [1, 0, 1, 0]])

## 创建无向图
# 用邻接矩阵创建无向图
b undirected_G = nx.Graph(adjacency_matrix, nodetype=int)

# 构造节点映射
c mapping = {0: 'a', 1: 'b', 2: 'c', 3: 'd'}

# 重命名节点
d undirected_G = nx.relabel_nodes(undirected_G, mapping)

## 定义节点坐标
pos = {'a': [0, 0],
       'b': [1, 0],
       'c': [1, 1],
       'd': [0, 1]}

## 可视化
plt.figure(figsize = (6,6))
nx.draw_networkx(undirected_G, pos = pos,
                  node_size = 188, with_labels=True)

```

再看邻接矩阵

如图 3 所示，邻接矩阵 A 的第 1 行 (除了主对角线元素之外) 数值为 1 的元素代表 a 可以直达的节点。也就是说，不途径任何节点，节点 a 可以直接走到 b 、 c 、 d 。

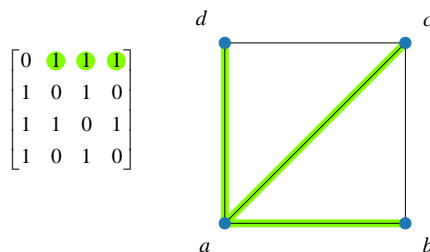


图 3. a 直达三个节点，邻接矩阵 A 的第 1 行

如图 4 所示，邻接矩阵 A 的第 1 列给出的也是相同的信息。

? 请大家用相同的思路分析邻接矩阵 A 剩余的行、列。

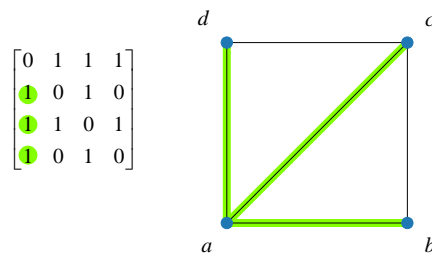


图 4. a 直达三个节点，邻接矩阵 A 的第 1 列

节点 a 途径一个节点到达 c 的路径数量可以通过以下矩阵乘法得到

$$A @ A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} @ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 2 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 1 & 3 & 1 \\ 1 & 2 & 1 & 2 \end{bmatrix} \quad (2)$$

如图 5 所示， $A @ A$ 结果第 1 行、第 3 列 (或第 3 行、第 1 列) 元素值就是节点 a 途径一个节点到达 c 的路径数量。

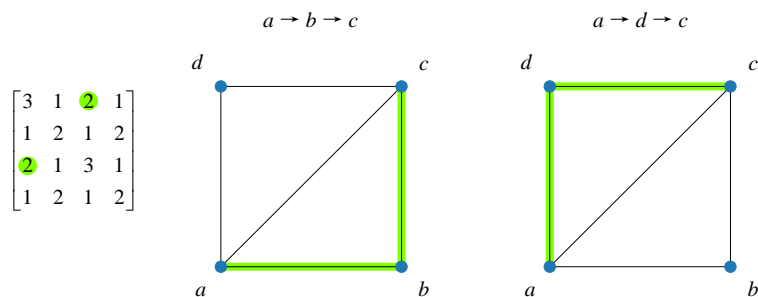


图 5. 节点 a 途径一个节点到达 c 的路径

而 $A @ A$ 主对角线元素则代表从某个节点出发、又回到同一节点路径。比如，如图 6 所示，节点 a 途径一个节点再回到 a 的路径有 3 条。

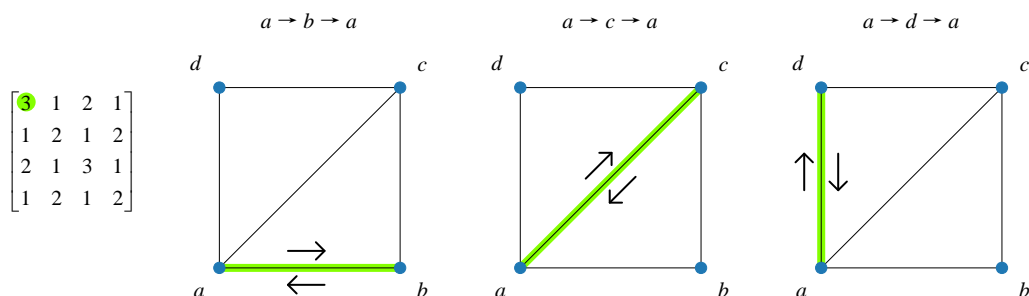


图 6. 节点 a 途径一个节点再回到 a 的路径

请大家分析 (2) 中 $A @ A$ 结果中剩余元素的含义。

要计算从 a 到 c 途径不超过一个节点的路径数量，我们可以利用如下运算

$$A + A @ A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 1 & 2 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 1 & 3 & 1 \\ 1 & 2 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 3 & 2 \\ 2 & 2 & 2 & 2 \\ 3 & 2 & 3 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix} \quad (3)$$

请大家自行分析如上结果。

度、度矩阵

无向图中，度 (degree) 指的是一个节点与图中其他节点连接的条数。度数反映节点的活跃程度。

具体来说，对于一个无向图，某个节点的“度”就是有多少条边与它相连。每连接一条边，节点的度数就加一。比如说，图 1 中节点 a 和 b 、 c 、 d 都有边相连，那 a 的度就是 3。

度矩阵 (degree matrix) 是一个用来记录所有节点度数的方阵。

度矩阵是一个对角方阵；也就是说，除了主对角线，其它位置上的元素全是 0。主对角线上的每个数值，对应的是每个节点的度数。

比如图 1 这幅无向图的度矩阵为

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad (4)$$

如图 7 所示，这个度矩阵告诉我们四个节点 a 、 b 、 c 、 d 的度数分别是 3、2、3、2。

一个社交网络中，节点度数高的人类似于“交际花”；节点度数低的人表示他们远离社交，喜欢独处。

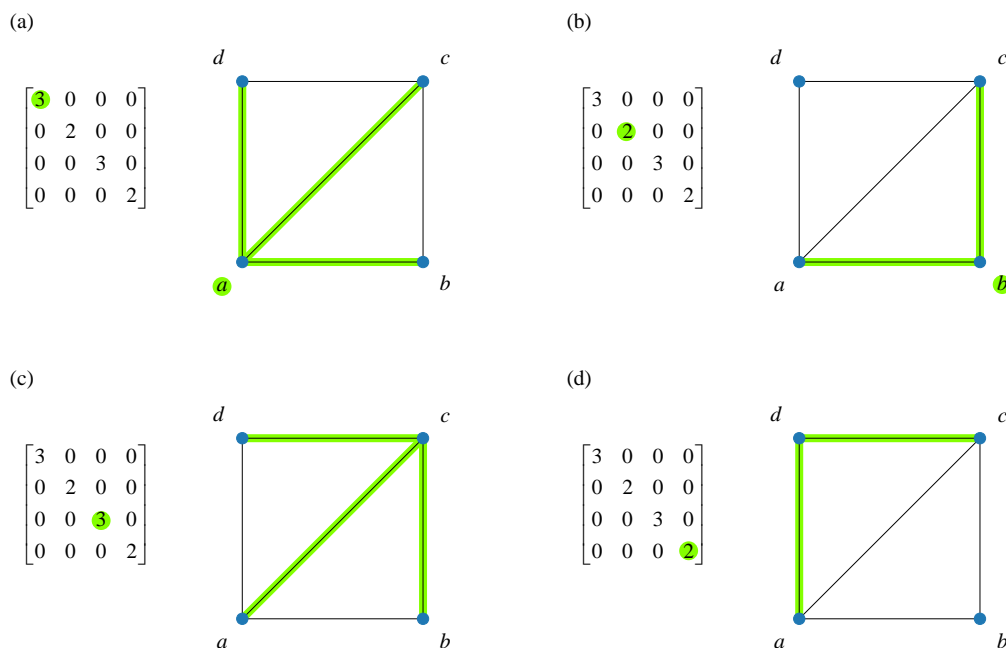


图 7. 无向图的度矩阵

无向图邻接矩阵 A 沿列求和的结果就是每个节点的度。用如下矩阵乘法运算

$$I^T A = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 3 & 2 \end{bmatrix} \quad (5)$$

得到的结果再用 `diag()` 函数构造对角方阵。

由于无向图的邻接矩阵 A 为对称矩阵，沿行求和的结果也是每个节点的度，即

$$A I = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 3 \\ 2 \end{bmatrix} \quad (6)$$

度和度矩阵是研究图结构、图计算、图算法的重要基础，常常和邻接矩阵一起使用，比如在计算图的普拉斯矩阵时。



LA_15_01_03.ipynb 用几种方法计算度矩阵，请大家自行学习。

有权无向图

有权无向图 (weighted undirected graph) 是指每条边都有数值权重；与之相对的是本节前文的无向图，本节前文的无向图也叫无权无向图 (unweighted undirected graph)。

无向图边的权重可以用来表示很多特征。以本节前文图 1 为例，边的权重可以是节点距离值、步行时间等等。

举个例子，如图 8 所示，我们把节点之间平均步行时间标注在无向图上，我们便得到有权有向图。

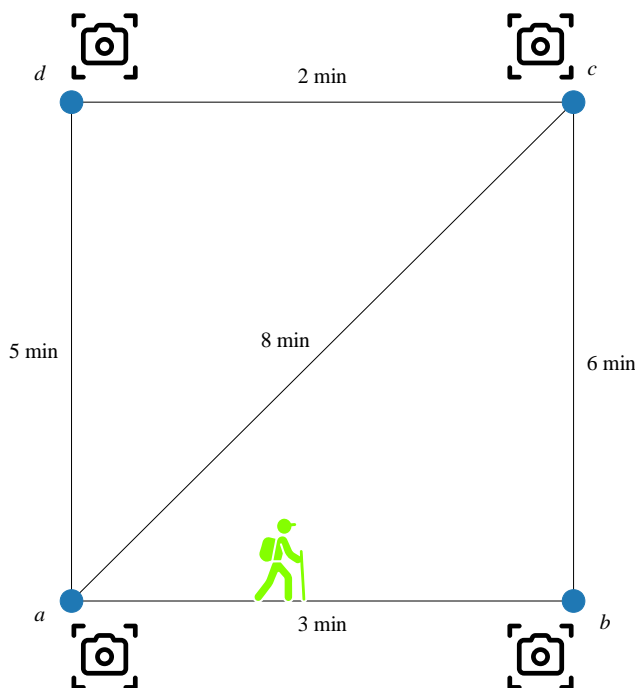


图 8. 公园景点之间构成的无向图，权重为节点之间步行的平均时间（分钟）



LA_15_01_04.ipynb 展示如何构造并可视化有权无向图。



请大家用 DeepSeek/ChatGPT 等工具完成本节如下习题。

- Q1.** 请用 NetworkX 逐一构建并绘制表 1 中所有无向图。
- Q2.** 请计算表 1 中所有无向图节点的度。
- Q3.** 请计算表 1 中所有无向图的邻接矩阵。
- Q4.** 请计算表 1 中所有无向图的度矩阵。
- Q5.** 请自学完全图、空图这两个概念，并指出表 1 中的完全图、空图分别是哪个。