

15

Path and More

从路径说起

通道、迹、路径、回路、环，各种路径问题



只有那些没有任何实际目的而追求它的人才能获得科学发现和科学知识。

Scientific discovery and scientific knowledge have been achieved only by those who have gone in pursuit of it without any practical purpose whatsoever in view.

—— 马克斯·普朗克 (Max Planck) | 德国物理学家，量子力学的创始人 | 1858 ~ 1947



- ◀ `networkx.algorithms.approximation.christofides()` 使用 Christofides 算法为加权图找到一个近似最短哈密顿回路的
- ◀ `networkx.all_simple_edge_paths()` 查找图中两个节点之间所有简单路径，以边的形式返回
- ◀ `networkx.all_simple_paths()` 查找图中两个节点之间所有简单路径
- ◀ `networkx.complete_graph()` 生成一个完全图，图中每对不同的节点之间都有一条边
- ◀ `networkx.DiGraph()` 创建一个有向图
- ◀ `networkx.find_cycle()` 在图中查找一个环
- ◀ `networkx.get_node_attributes()` 获取图中所有节点的指定属性
- ◀ `networkx.has_path()` 检查图中是否存在从一个节点到另一个节点的路径
- ◀ `networkx.shortest_path()` 计算图中两个节点之间的最短路径
- ◀ `networkx.shortest_path_length()` 计算图中两个节点之间最短路径的长度
- ◀ `networkx.simple_cycles()` 查找有向图中所有简单环
- ◀ `networkx.utils.pairwise()` 生成一个节点对的迭代器，用于遍历图中相邻的节点对

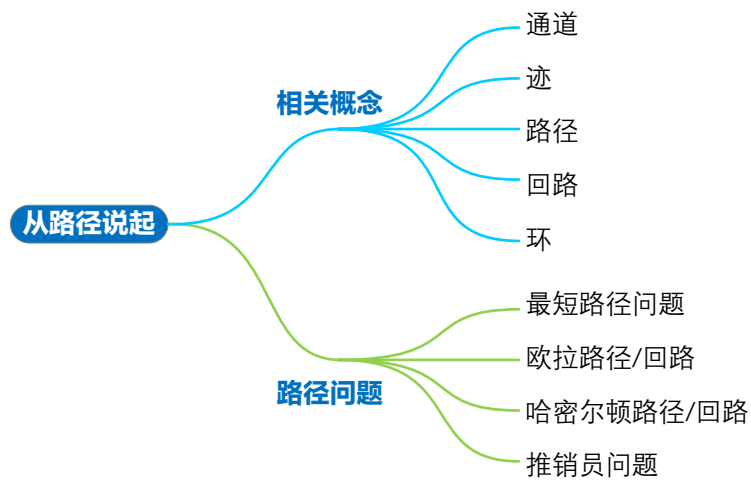
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

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



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

15.1 通道、迹、路径、回路、环

本章要涉及的话题都和路径有关；为了方便介绍路径，这一节先把通道、迹、路径、回路、环这几个相似概念放在一起对比来讲。

- ▶ **通道** (walk): 一个节点和边的交替序列，可以包含重复边或经过相同的节点。
- ▶ **迹** (trail): 无重复边的通道，但是可以允许重复节点。
- ▶ **路径** (path): 无重复节点、无重复边的通道；也就是说，一个路径中，每个节点和边最多只能经过一次。
- ▶ **回路** (circuit): 闭合的迹，即起点和终点形成闭环。回路中的节点可以重复，但是边不能重复。
- ▶ **环** (cycle): 没有重复节点的回路；环都是回路，但是回路不都是环。

大家可能已经发现，它们之间的区别主要涉及到是否允许重复访问节点和边，以及首尾是否闭合。

通道

在图论中，**通道** (walk) 是指沿图的边依次经过一系列节点的序列。

通道的特点如下。

- ▶ 可能包含重复节点：即同一个节点可以在通道中多次出现。
- ▶ 可能包含重复边：即同一个节边可以在通道中多次出现。

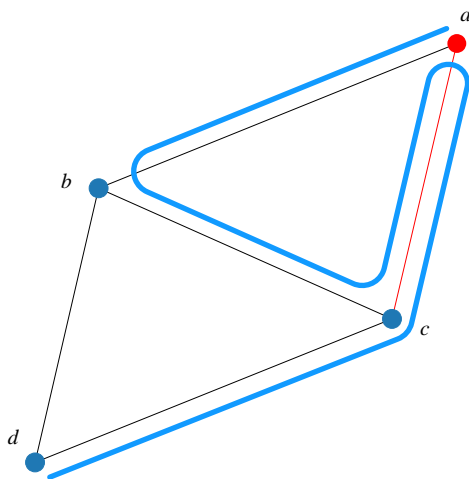


图 1. 无向图中的一条通道

例如，在图 1 所示的一个简单的无向图中， $a \rightarrow b \rightarrow c \rightarrow a \rightarrow c \rightarrow d$ 就是一条通道。在这条通道中，节点 c 重复，无向边 ac (ca) 重复。

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

通道的定义适用于有向图和无向图。在有向图中，通道考虑了边的方向；而在无向图中，通道只关注边的存在而不考虑方向。

迹：不含重复边的通道

在图论中，**迹** (trail) 描述了节点之间的连接，但边的序列中不允许出现重复的边，即每个边只能经过一次。迹是通道的一个特殊情况，限制了边的重复性。

迹的特点如下。

- ▶ 不含重复边：迹是一条不含重复边的通道。
- ▶ 可以包含重复节点：节点可以在迹中重复出现。

例如，在图2所示的一个简单的无向图中， $a \rightarrow b \rightarrow d \rightarrow a \rightarrow c$ 就是一条迹。在这条迹中，节点 a 重复，但是不存在任何重复无向边。

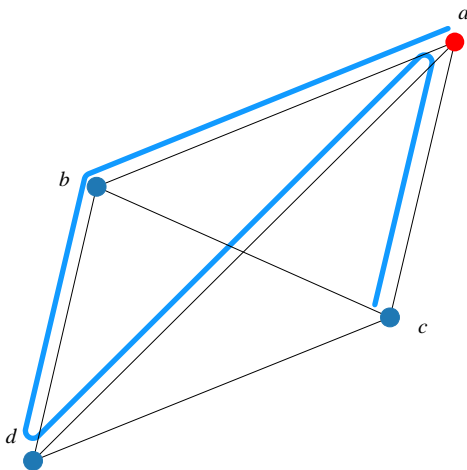


图 2. 无向图中的一条迹

迹的定义适用于有向图和无向图。在有向图中，迹考虑了边的方向，而在无向图中，迹只关注边的存在而不考虑方向。

路径：不含重复节点和不含重复边的通道

路径 (path) 是通道的一种特殊情况，它描述了图中节点之间的一条连接，并确保每个节点和每条边只经过一次。路径是图中两个节点之间的最简单的连接。

路径的特点如下。

- ▶ 不含重复节点：路径是一条不含重复节点的通道。
- ▶ 不含重复边：路径是一条不含重复边的通道。

例如，在图3所示的一个简单的无向图中， $a \rightarrow b \rightarrow c \rightarrow d$ 就是一条路径。这条路径连接了 a 、 d 节点，且不存在任何重复节点，也不存在任何重复边。

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

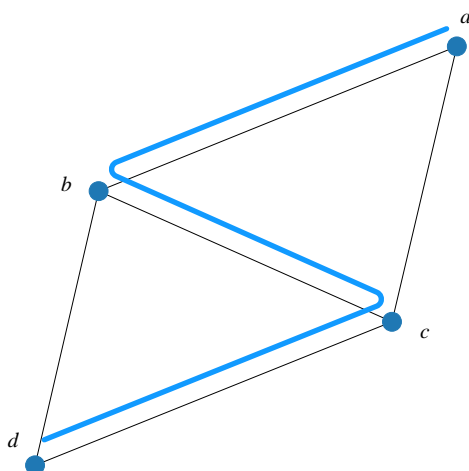


图 3. 无向图中的一条路径

路径的定义适用于有向图和无向图。在有向图中，路径考虑了边的方向，而在无向图中，路径只关注边的存在而不考虑方向。

不考虑边的权重的话，两个节点之间的**路径长度** (path length) 是指路径上的边数。考虑权重的话，两个节点之间的路径长度表示从一个节点到另一个节点沿路径经过的边的权重之和。这适用于图中的边具有权重的情况，例如，道路网络中的距离、通信网络中的传输成本等。

下面，让我们看一个例子。如图 4 所示，我们要在这个 5 个节点完全图的 0、3 节点之间找到所有路径。

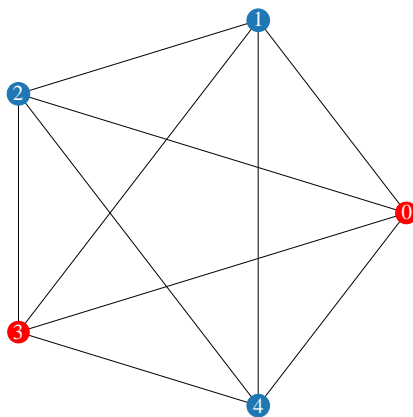


图 4. 5 个节点的完全图，无向边

图 5 给出了答案，节点 0、3 之间一共存在 16 条路径。

图 5 (a) 这条路径的长度为 3。

显然，图 5 (k) 这条路径最短，我们管这种路径叫做**最短路径** (shortest path)。最短路径是指在图中连接两个节点的路径中，具有最小长度 (无权图) 或最小权重 (有权图) 的路径。在图论中，寻找最短路径是一个重要的问题，下一章将介绍这个问题。

请大家注意，图 5 (b)、(d)、(g)、(i)、(l)、(n) 这几幅子图给出的路径。我们发现它们都有个相同特征——路径经过图中所有节点。一个经过图中所有节点的路径被称为**哈密顿路径** (Hamiltonian path)。哈密顿路径是一种特殊的路径，它是穿过图中每个节点且仅经过一次的路径，但不一定经过每条边。本章后文还会介绍这种路径。

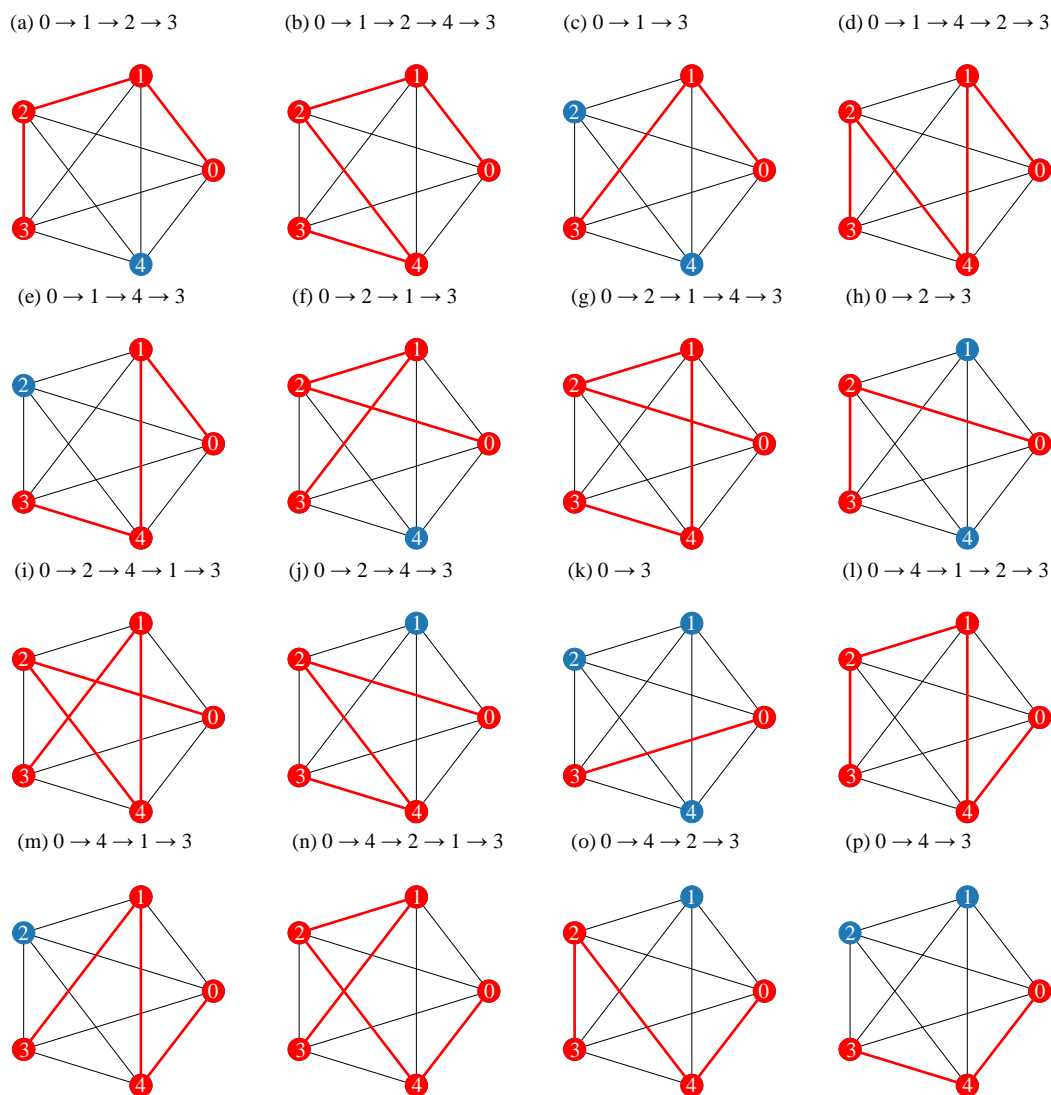


图 5. 节点 0、3 之间的路径

总的来说，路径是图论中描述两个节点之间连接的一种清晰、简单的通道。

- a** 用 `networkx.complete_graph(5)` 创建 5 节点完全图。
- b** 用 `networkx.circular_layout()` 生成节点的圆形布局。

c 用 `networkx.all_simple_paths()` 找到图中所有从源节点到终点的简单路径。该函数返回一个生成器对象，该生成器会产生所有从源节点到终点的简单路径，每个路径都表示为节点列表。参数 `source` 是源节点，参数 `target` 是终点；但是对于无向图，源节点和终点的顺序并不重要。

d 用 `networkx.all_simple_edge_paths()` 在给定的图中找到所有从源节点到终点的简单边路径。函数返回的是一个生成器，它会产生每一条路径。每条路径都表示为边的序列，其中每个边是一个表示起点和终点的元组。可以遍历这个生成器来访问所有找到的路径。

e 用 `networkx.draw_networkx()` 绘制无向图。

f 用 `networkx.draw_networkx_nodes()` 绘制图的节点。参数 `ax` 指定了图形绘制在哪个轴上。参数 `odelist` 是一个节点列表，指定了哪些节点将被绘制。

参数 `node_size` 控制节点的大小。可以是单个数值，对所有节点应用相同的大小；也可以是一个节点大小的列表或字典，为每个节点指定不同的大小。

参数 `node_color` 指定节点的颜色。它可以是一个单独的颜色代码或颜色名称，应用于所有节点；也可以是一个颜色序列，为每个节点指定不同的颜色。

g 用 `networkx.draw_networkx_edges()` 绘制图的边。参数 `edgelist` 是一个边的列表，指定了哪些边将被绘制。参数 `edge_color` 指定边的颜色。它可以是一个单独的颜色代码或颜色名称，应用于所有边；也可以是一个颜色序列，为每条边指定不同的颜色。此外，还可以通过边的属性（如边权重）动态计算颜色。

h 用 `networkx.shortest_path()` 找到图中指定两个节点之间的最短路径。这个函数可以应用于无向图和有向图，并且能够处理有权重和无权重的边。参数 `source` 指定起点；参数 `target` 指定终点。

```

import networkx as nx
import matplotlib.pyplot as plt

a G = nx.complete_graph(5)
# 完全图

# 可视化图
plt.figure(figsize = (6,6))
b pos = nx.circular_layout(G)
nx.draw_networkx(G,
                  pos = pos,
                  with_labels = True,
                  node_size = 188)
plt.savefig('完全图.svg')

# 节点0、3之间所有路径

c all_paths_nodes = nx.all_simple_paths(G, source=0, target=3)
# 节点0、3之间所有路径上的节点

d all_paths_edges = nx.all_simple_edge_paths(G, source=0, target=3)
# 节点0、3之间所有路径上的边

# 可视化
fig, axes = plt.subplots(4, 4, figsize = (8,8))
axes = axes.flatten()

for nodes_i, edges_i, ax_i in zip(all_paths_nodes, all_paths_edges, axes):
e     nx.draw_networkx(G, pos = pos,
                        ax = ax_i, with_labels = True,
                        node_size = 88)

f     nx.draw_networkx_nodes(G, pos = pos,
                             ax = ax_i, nodelist = nodes_i,
                             node_size = 88, node_color = 'r')


g     nx.draw_networkx_edges(G, pos = pos,
                             ax = ax_i, edgelist = edges_i, edge_color = 'r')

    ax_i.set_title(' → '.join(str(node) for node in nodes_i))
    ax_i.axis('off')

plt.savefig('节点0、3之间所有路径.svg')

# 最短路径
h print(nx.shortest_path(G, source=0, target=3))

```

代码 1. 5 个节点完全图中节点 0、3 之间的路径 |  Bk6_Ch15_01.ipynb

下面再看一个有向图中路径的例子。在图 4 的基础上，我们随机地给每条无向边赋予方向，得到如图 6 所示的有向图。下面，让我们在这幅有向图中先找到节点 0 为始点、3 为终点的路径；然后，再找节点 3 为始点、0 为终点的路径。

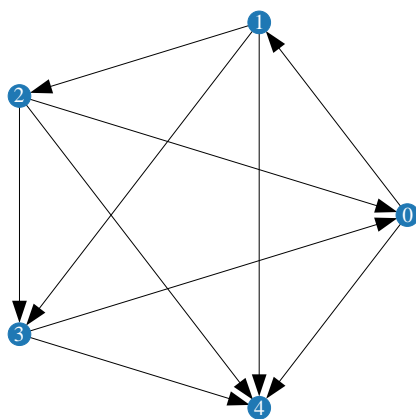


图 6. 5 个节点有向图

图 7 所示为节点 0 为始点、节点 3 为终点的路径，存在两条。

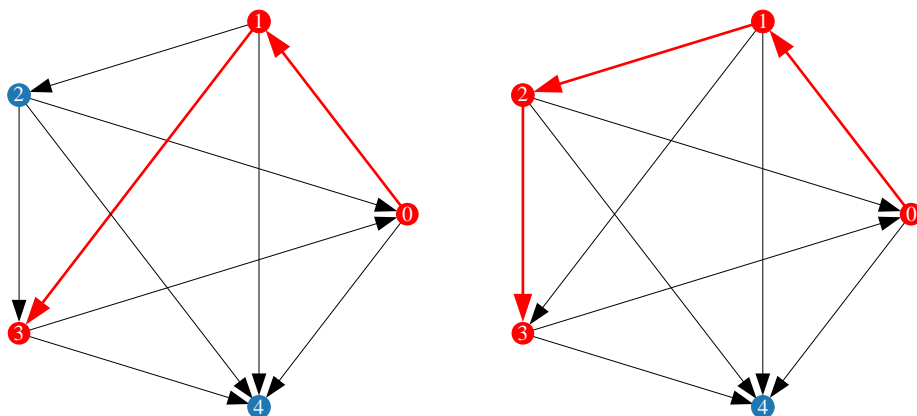


图 7. 节点 0 为始点、节点 3 为终点的路径，有向图

图 8 所示为节点 3 为始点、节点 0 为终点的路径，仅有一条。

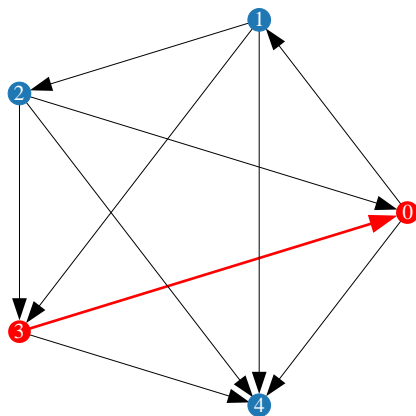


图 8. 节点 3 为始点、节点 0 为终点的路径，有向图

代码 2 完成图 6、图 7、图 8 相关计算，下面聊聊其中关键语句。

- a** 先用 `networkx.complete_graph(5)` 绘制 5 个节点的完全图，这幅图为无向图。
- b** 用 `networkx.DiGraph()` 创建全新空的有向图。
- c** 从一个无向图 `G_undirected` 创建一个有向图 `G_directed`。这段代码用 `for` 循环遍历无向图中的所有边，然后用 `random.choice()` 以 50% 的概率决定边的方向，在新的有向图中用 `add_edge()` 方法添加对应的有向边。
 函数 `random.choice([True, False])` 将等概率地返回 `True` 或 `False`，来决定边的方向。
 如果随机选择的结果是 `True`，则用 `G_directed.add_edge(u, v)` 在有向图 `G_directed` 中添加一条从 `u` 到 `v` 的有向边，即节点 `u` 指向节点 `v`。
 如果随机选择的结果是 `False`，则用 `G_directed.add_edge(v, u)` 在有向图 `G_directed` 中添加一条从 `v` 到 `u` 的有向边，即节点 `v` 指向节点 `u`。
- d** 用 `networkx.all_simple_edge_paths()` 在有向图 `G_directed` 中找到从节点 0 (`source=0`) 到节点 3 (`target=3`) 的所有路径。函数返回一个迭代器，该迭代器生成从 `source` 节点到 `target` 节点的所有简单路径。每条路径都表示为节点列表，顺序表示路径上的移动。可以用 `list()` 将迭代器转换为列表，查看每条路径中具体节点。
- e** 用 `networkx.all_simple_edge_paths(G)` 在有向图 `G_directed` 中查找从始点 (`source`) 0 到 (`target`) 3 的所有路径。这个函数返回的是边的序列，而不是节点的序列。每条路径都表示为一系列边，其中每条边由其端点的元组（起点，终点）表示。

```

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

# 创建一个包含5个节点的无向完全图
a G_undirected = nx.complete_graph(5)

# 创建一个新的有向图
b G_directed = nx.DiGraph()

# 为每对节点随机选择方向
random.seed(8)
c for u, v in G_undirected.edges():
    if random.choice([True, False]):
        G_directed.add_edge(u, v)
    else:
        G_directed.add_edge(v, u)

# 节点0为始点、3为终点之间所有路径

d all_paths_nodes = nx.all_simple_paths(G_directed, source=0, target=3)
# 节点0为始点、3为终点所有路径上的节点

e all_paths_edges = nx.all_simple_edge_paths(G_directed, source=0, target=3)
# 节点0为始点、3为终点所有路径上的边

# 节点3为始点、0为终点之间所有路径

all_paths_nodes = nx.all_simple_paths(G_directed, source=3, target=0)
# 节点3为始点、0为终点所有路径上的节点

all_paths_edges = nx.all_simple_edge_paths(G_directed, source=3, target=0)
# 节点3为始点、0为终点所有路径上的边

```

代码 2. 有向图中节点 0、3 之间的路径 | Bk6_Ch15_02.ipynb

回路：首尾闭合，可以重复节点，不允许重复边

在图论中，**回路** (circuit) 是一种首尾闭合的迹。

回路的特点如下。

- ▶ 闭合：起点和终点相同，首尾闭合。
- ▶ 可以包含重复节点：回路(途中，不包括首尾节点)可以包含重复的节点。
- ▶ 不含重复边：回路不允许包含重复的边。

回路的定义适用于有向图和无向图。在有向图中，回路沿着有向边形成环路；在无向图中，回路沿着无向边形成环路。

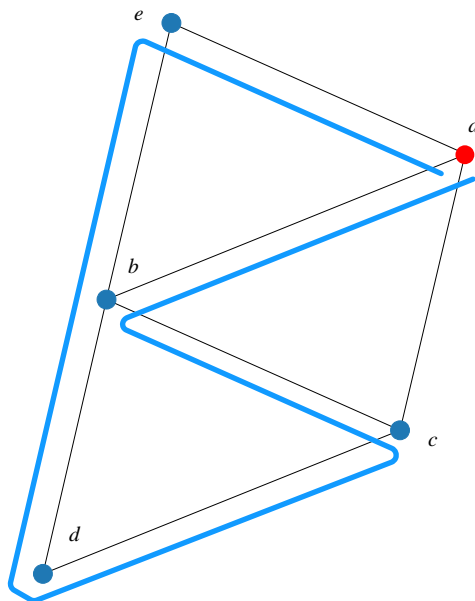


图 9. 无向图中的一条回路

环

在图论中，环是一种特殊回路；起点和终点之外的所有节点都不重复，当然环中边都不重复。这种环也叫**简单环** (simple cycle)。

环的特点如下。

- ▶ 闭合：起点和终点相同，首尾闭合。这意味着可以从环上的任一节点出发，经过一系列边，最终回到同一节点。
- ▶ 途中不包含重复节点：每个节点只能在环中出现一次，除了起点和终点。
- ▶ 不含重复边：每条边只能在环中出现一次。

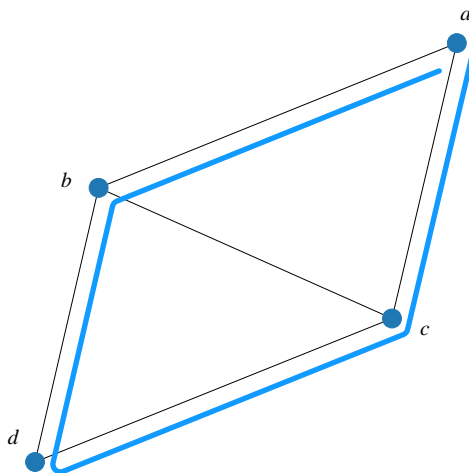


图 10. 无向图中的一个环

图 11 所示为图 6 有向图中找到的 3 个环。

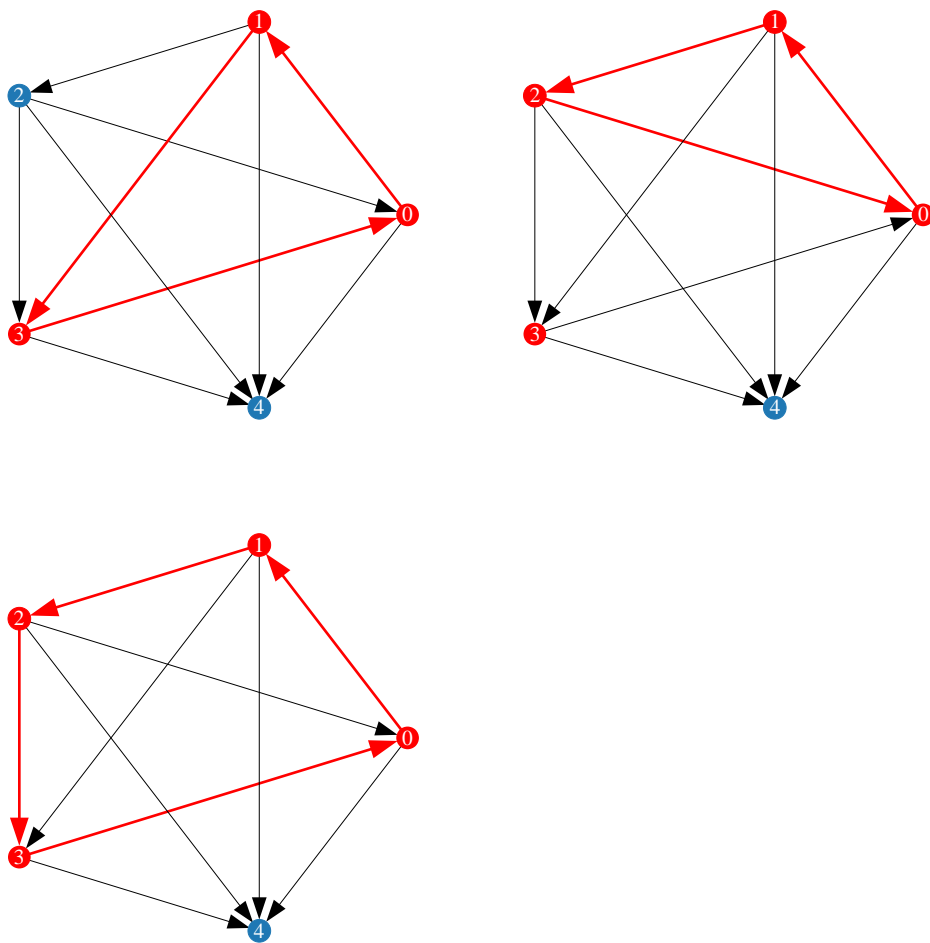


图 11. 有向图中的 3 个环

代码 3 找出图 6 有向图中所有环，并绘制图 11。下面聊聊其中关键语句。

a 用 `networkx.find_cycle()` 找到有向图中一个（并不是所有）环。参数 `orientation` 指定搜索环时边的方向性考虑方式，其中 `orientation="original"` 意味着函数在查找环时会考虑边的原始方向。函数返回一个环的列表，其中每个元素是表示边的元组，包括边的起点和终点。如果图中没有环，函数会抛出一个 `NetworkXNoCycle` 异常。

b 先用 `networkx.simple_cycles(G_directed)` 找到有向图 `G_directed` 中所有的简单环。然后，再用 `list()` 将生成器转换成嵌套列表。嵌套列表中每个元素是一个环中的顺序节点构成的列表。

c 自定义函数将节点列表转化为一个环中边的列表。

d 遍历节点列表，除了最后一个节点，为每对相邻的节点生成一个元组（起点，终点）。这样得到的边列表 `list_edges` 包含了从第一个节点到最后一个节点的所有边，但没有形成一个闭环。

e 创建了一个从列表中最后一个节点到第一个节点的边，这样做的目的是为了在节点的连接上形成一个闭环。

大家对 **f** 这句应该很熟悉了，它可视化有向图，并用红色着重强调环。

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

完整代码请查看 Bk6_Ch15_03.ipynb。

```

a cycle = nx.find_cycle(G_directed, orientation="original")
# 在有向图找到一个环，并不是所有环

b list_cycles = list(nx.simple_cycles(G_directed))
# 找到有向图中所有环

# 自定义函数将节点序列转化为边序列（闭环）

c def nodes_2_edges(node_list):
    # 使用列表生成式创建边的列表
    d list_edges = [(node_list[i], node_list[i+1])
                    for i in range(len(node_list)-1)]

    # 加上一个额外的边从最后一个节点回到第一个节点，形成闭环
    e closing_edge = [(node_list[-1], node_list[0])]
    list_edges = list_edges + closing_edge
    return list_edges

# 可视化有向图中3个环

fig, axes = plt.subplots(1, 3, figsize = (9,3))
axes = axes.flatten()

f for nodes_i, ax_i in zip(list_cycles, axes):
    edges_i = nodes_2_edges(nodes_i)

    nx.draw_networkx(G_directed, ax = ax_i,
                     pos = pos, with_labels = True, node_size = 88)

    nx.draw_networkx_nodes(G_directed, ax = ax_i,
                           nodelist = nodes_i, pos = pos,
                           node_size = 88, node_color = 'r')

    nx.draw_networkx_edges(G_directed, pos = pos,
                           ax = ax_i, edgelist = edges_i,
                           edge_color = 'r')

    ax_i.set_title(' → '.join(str(node) for node in nodes_i))
    ax_i.axis('off')

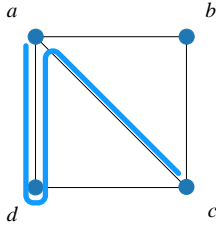
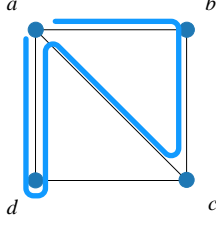
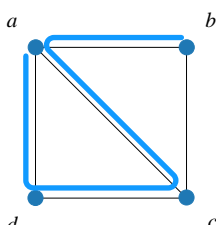
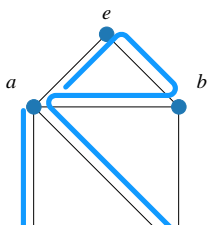
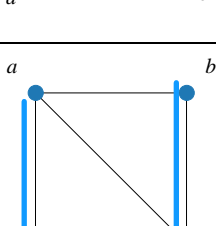
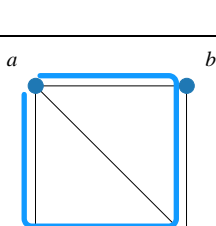
plt.savefig('有向图中所有cycles.svg')

```

代码 3. 查找有向图中的环 | Bk6_Ch15_03.ipynb

表 1 总结比较通道、迹、路径、回路、环之间的异同。请大家注意，如果通道的起点和终点相同，则称其**闭合** (close)；否则，称其**开放** (open)。

表 1. 比较通道、迹、路径、回路、环

类型	途中重复节点?	途中重复边?	首尾闭合?	示例
Walk (open)	Yes	Yes	No	 $a \rightarrow d \rightarrow a \rightarrow c$
Walk (closed) Loop	Yes	Yes	Yes	 $a \rightarrow d \rightarrow a \rightarrow c \rightarrow b \rightarrow a$
Trail	Yes	No	No	 $a \rightarrow d \rightarrow c \rightarrow a \rightarrow b$
Circuit	Yes	No	Yes	 $a \rightarrow d \rightarrow c \rightarrow a \rightarrow b \rightarrow e \rightarrow a$
Path	No	No	No	 $a \rightarrow d \rightarrow c \rightarrow b$
Cycle	No	No	Yes	 $a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$

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

15.2 常见路径问题

常见路径问题总结如下。

- ▶ **最短路径问题** (shortest path problem): 在一个有向图或无向图中，特别是考虑权重条件下，找到两个指定节点的最短距离。
- ▶ **欧拉路径** (Eulerian path): 不重复地经过所有边，不要求最终回到起点。
- ▶ **欧拉回路** (Eulerian cycle), 不重复地经过所有边，并最终回到起点。欧拉回路也是所谓的七桥问题。
- ▶ **中国邮递员问题** (Chinese Postman Problem) 在一个有向图或无向图中，找到一条回路 (最终回到起点)，使得每条边都至少被经过一次，并最小化路径总长度 (考虑权重)。
- ▶ **哈密尔顿路径** (Hamiltonian path, Hamilton path): 不重复地经过所有节点，不要求最终回到起点。
- ▶ **哈密尔顿回路** (Hamiltonian cycle, Hamilton cycle): 不重复地经过 (途中) 所有节点，最终回到起点。
- ▶ **推销员问题** (Traveling Salesman Problem): 不重复地经过所有节点，最终回到起点，并最小化路径总长度 (考虑权重)。推销员问题是特殊的哈密尔顿回路。

表 1 比较以上几种路径问题，本章后续介绍几个能够用 NetworkX 直接求解的路径问题。

表 2. 比较几种常见路径问题

问题	是否回到起点	节点要求	边要求	优化要求
最短路径问题	否	否	否	距离最短
欧拉路径	否	否	不重复地经过所有边	无
欧拉回路	是	否	不重复地经过所有边	无
中国邮递员问题	是	否	每条边至少经过一次	最小化路径长度
哈密尔顿路径	否	不重复地经过所有节点	每条边最多经过一次	无
哈密尔顿回路	是	不重复地经过所有节点	每条边最多经过一次	无
推销员问题	是	不重复地经过所有节点	每条边最多经过一次	最小化路径长度

15.3 最短路径问题

最短路径问题 (shortest path problem) 是图论中的一个经典问题，其目标是在图中找到两个节点之间的最短路径，即路径上各边权重之和最小的路径。**所有两节点最短路径问题** (All-Pairs Shortest Path problem) 则更进一步找到任意两个节点的最短距离。

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

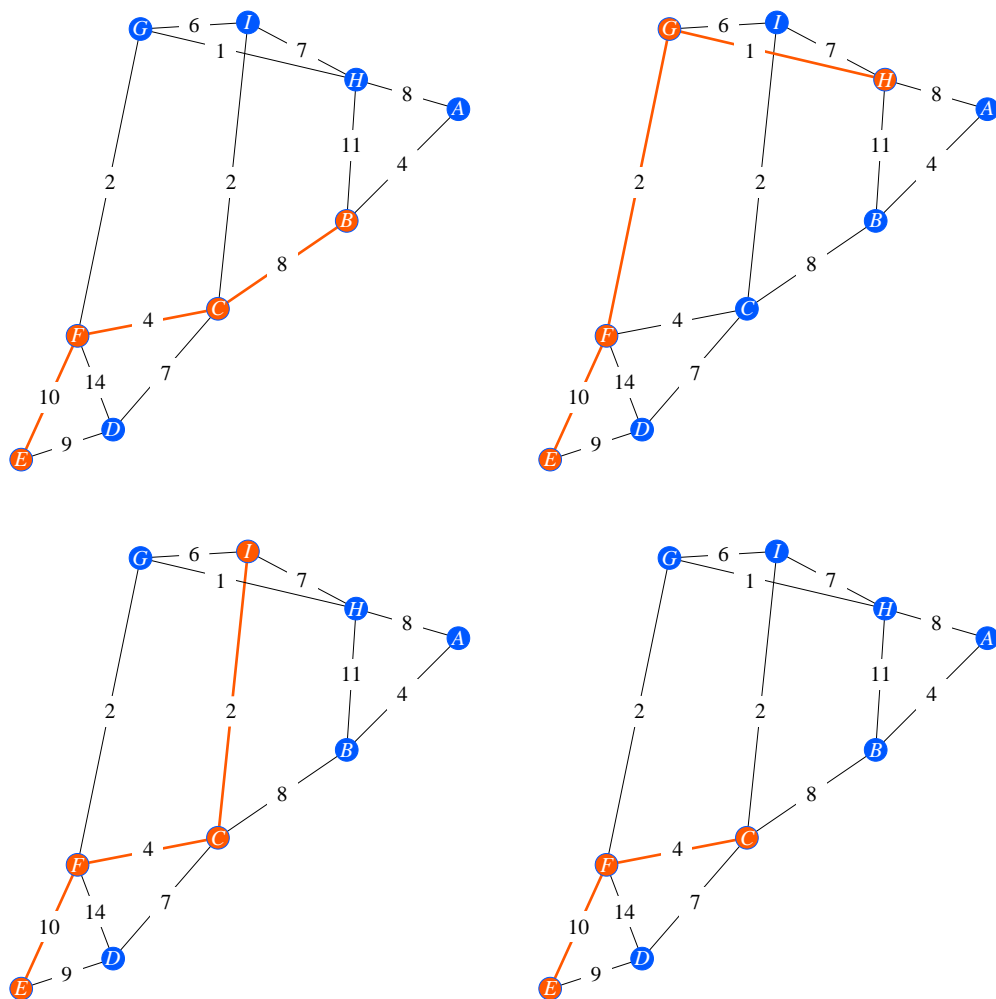


图 15. 节点 E 为终点的四条最短路径，考虑边权重

图 16 所示为一矩阵形式展示无向图中任意两个节点之间的最短路径距离。这个矩阵有自己的名字——图距离矩阵。本书后文将介绍这个概念。

A	0	4	12	19	21	11	9	8	14
B	4	0	8	15	22	12	12	11	10
C	12	8	0	7	14	4	6	7	2
D	19	15	7	0	9	11	13	14	9
E	21	22	14	9	0	10	12	13	16
F	11	12	4	11	10	0	2	3	6
G	9	12	6	13	12	2	0	1	6
H	8	11	7	14	13	3	1	0	7
I	14	10	2	9	16	6	6	7	0
	A	B	C	D	E	F	G	H	I

图 16. 无向图成对最短路径长度，矩阵形式

有向图中的最短路径问题

下面，让我们看一下图 17 这幅有向图中的最短路径问题。以节点 A 为起点，还是以节点 E 为终点，我们无法找到一条路径。原因很简单，节点 E 的两条边均是离开 E。想要解决这个问题，需要调转 EF 或者 ED。

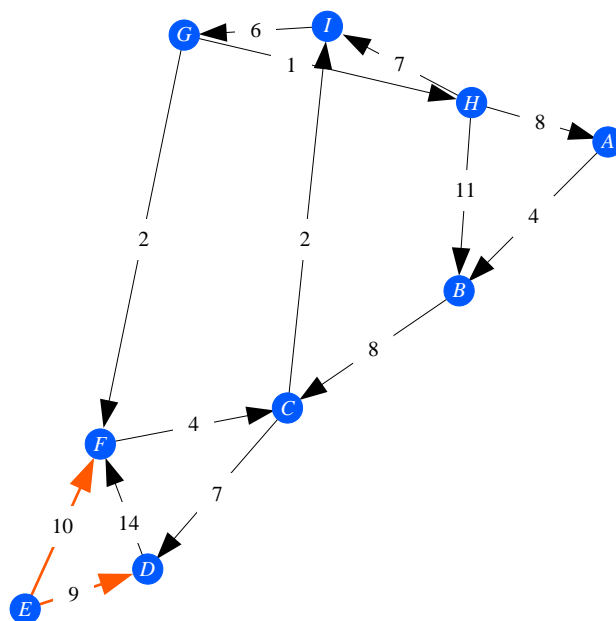


图 17. 有向图，考虑边权重

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

但是，如果以节点 E 为起点，以节点 A 为终点，我们倒是可以找到一条组最短路径，如图 18 所示。这条路径的长度为 31。

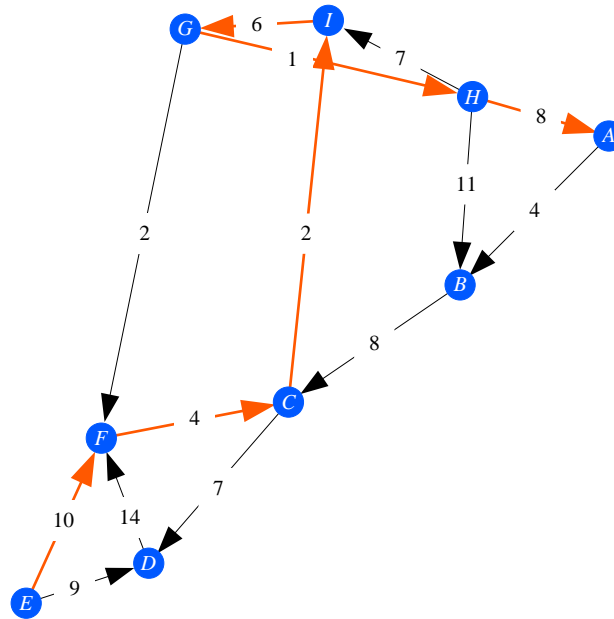


图 18. 有向图, 节点 E (起点) 到节点 A (终点) 的最短路径, 考虑边权重

A	0	4	12	19		22	20	21	14
B	25	0	8	15		18	16	17	10
C	17	20	0	7		10	8	9	2
D	35	38	18	0		14	26	27	20
E	31	34	14	9	0	10	22	23	16
F	21	24	4	11		0	12	13	6
G	9	12	6	13		2	0	1	8
H	8	11	19	26		15	13	0	7
I	15	18	12	19		8	6	7	0
	A	B	C	D	E	F	G	H	I

图 19. 有向图成对最短路径长度, 矩阵形式

15.4 欧拉路径

欧拉路径 (Eulerian path) 是指在图中，通过图中的每一条边恰好一次且仅一次，并且路径的起点和终点不同的路径。

欧拉回路 (Eulerian cycle) 是图中的一个闭合路径，该路径通过图中的每一条边恰好一次且仅一次，并且路径的起点和终点是同一个节点。如果存在这样的回路，该图称为**欧拉图** (Eulerian graph, Euler graph)。五个柏拉图图中只有六面体图是欧拉回路，如图 20 所示。

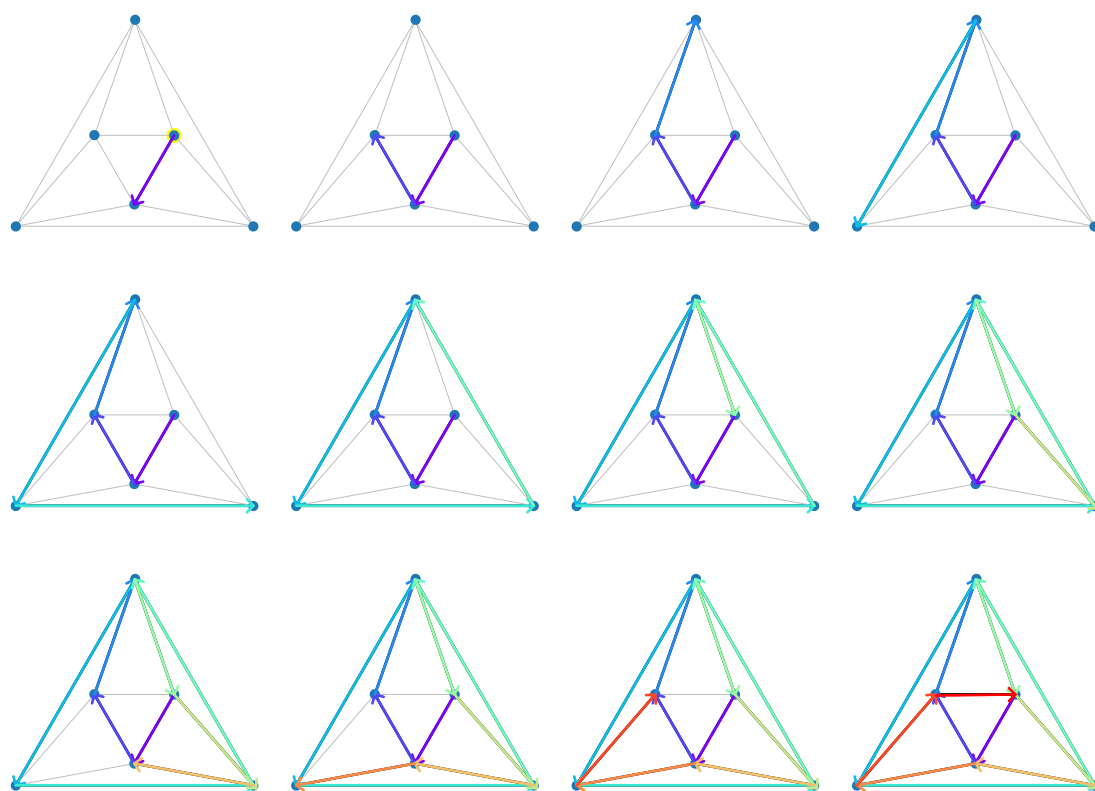


图 20. 六面体图，欧拉回路

15.5 哈密尔顿路径

哈密尔顿路径 (Hamiltonian path, Hamilton path) 是图论中的一个概念，指的是在一个图中，经过每个节点恰好一次且不重复的路径。具体来说，哈密尔顿路径是图中的一个节点序列，使得对于序列中的相邻两个节点，图中存在一条边直接连接它们，且路径中的每个节点都不重复。

一个图中可能有多个哈密尔顿路径，也可能没有哈密尔顿路径。如果存在哈密尔顿路径，那么这个图被称为哈密尔顿图。

与哈密尔顿路径相关的另一个概念是**哈密尔顿回路** (Hamiltonian cycle)，准确来说是哈密尔顿环 (节点不重复)。哈密尔顿回路是一种哈密尔顿路径，其起点和终点相同。如果一个图中存在哈密尔顿回路，那么这个图被称为哈密尔顿回路图。

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

如图 21 所示，五个柏拉图图都是哈密顿回路。

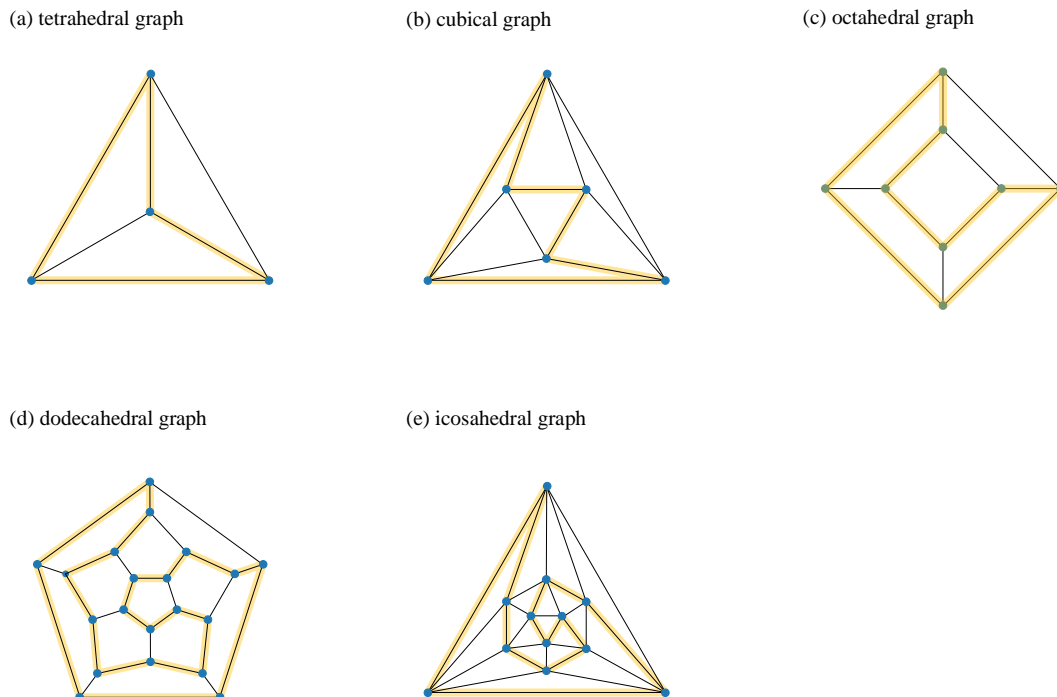


图 21. 五个柏拉图图，都是哈密顿回路

15.6 推销员问题

推销员问题 (Traveling Salesman Problem, TSP) 和哈密顿路径问题有密切关系；实际上，TSP 可以看作是哈密顿路径问题的一个特例。

推销员问题的描述是这样的，假设有一个推销员要拜访一组城市，并且他想找到一条最短的路径，使得他每个城市都拜访一次，最后回到起始城市。这个问题的目标是找到这条最短路径，使得总旅行距离最小。

将这个问题抽象成图论中的问题，每个城市可以看作图中的一个节点，城市之间的路径长度可以看作图中的边权重。推销员问题实际上就是在图中寻找一个哈密顿回路，即一个经过每个城市一次且最终回到起始城市的路径。

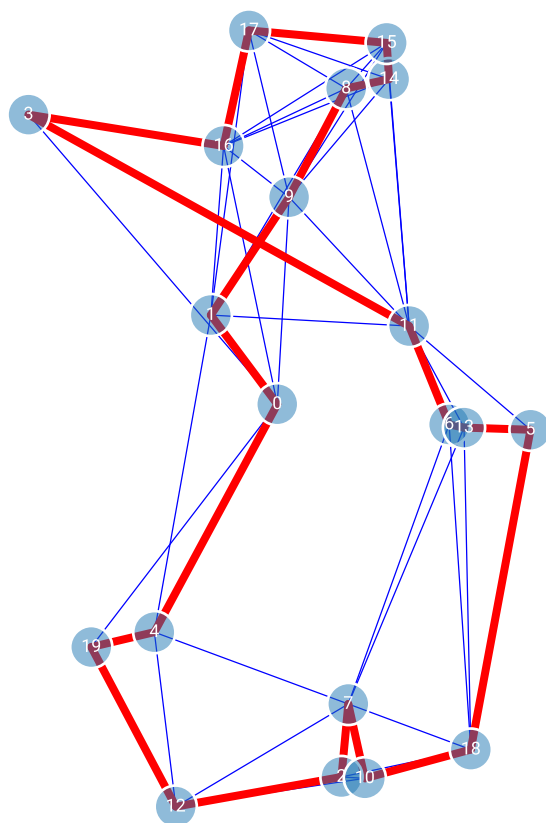
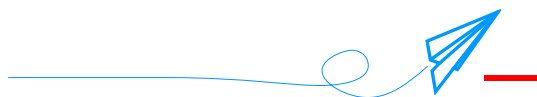


图 22. 销售员问题



一个路径中，每个节点和边最多只能经过一次。最短路径问题寻找两点间最短路径（一般要考虑权重），欧拉路径问题要求经过每条边恰好一次，哈密尔顿路径问题要求经过每个节点恰好一次，推销员问题则是在哈密尔顿回路基础上寻找最短的闭合回路，解决实际中的最优路径选择问题。