

# 24

## PageRank Algorithm

# PageRank 算法

用网络图中页面之间链接关系，以衡量其重要性和排名



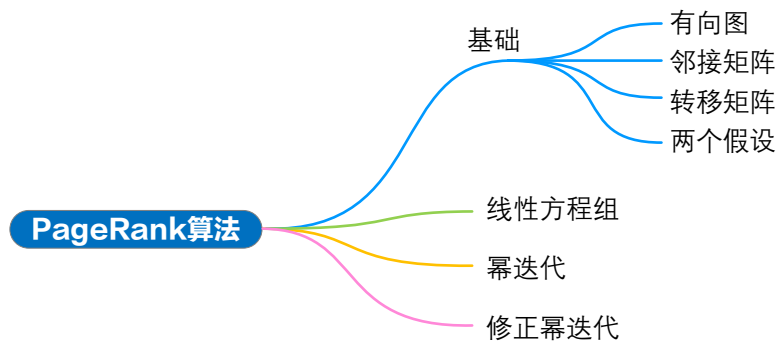
远离那些试图贬低你的雄心壮志的人。小人物总是这样做，但真正伟大的人会让你相信你也可以变得伟大。

***Keep away from those who try to belittle your ambitions. Small people always do that, but the really great make you believe that you too can become great.***

—— 马克·吐温 (Mark Twain) | 美国作家 | 1835 ~ 1910



- ◀ `networkx.adjacency_matrix()` 计算邻接矩阵
- ◀ `networkx.circular_layout()` 节点圆周布局
- ◀ `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.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
- ◀ `numpy.linalg.eig()` 特征值分解
- ◀ `numpy.linalg.norm()` 计算范数
- ◀ `numpy.ones()` 按指定形状生成全 1 矩阵



# 24.1 PageRank 算法

互联网、社交网络都可以看做是图。

PageRank 算法，即网页排名，是由谷歌公司创始人**拉里·佩奇** (Larry Page) 和**谢尔盖·布林** (Sergey Brin) 于 1996 年提出的，是一种用于评估网页重要性的算法。

PageRank 算法最初是为了优化搜索引擎结果而设计的。通过分析网页之间的链接结构，搜索引擎可以更好地确定哪些网页更重要，从而为用户提供更相关和有质量的搜索结果。

简单来说，PageRank 算法通过分析网页之间的链接关系，为每个网页赋予一个权重值，从而确定搜索结果的排名顺序。我们可以把 PageRank 算法看做是在有向图。

以图 1 为例，6 个网页之间存在相互链接关系。网页被越多的其他网页链接，就越重要。一个网页的重要性可以通过其被其他网页链接的数量来衡量。比如，有 5 个网页都有指向网页 *e* 的链接，显然网页 *e* 很重要。

根据这个思路，由于网页数量有限，我们已经可以给这些网页做个排名。但是，全球互联网的网页数量已经以 10 亿计，显然我们需要量化手段来帮助排名。

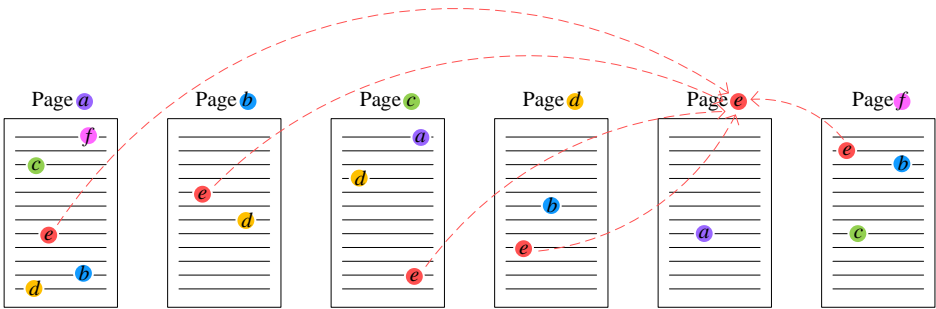


图 1. 6 个网页之间的链接关系

## 有向图

图 1 这种关系显然可以用有向图来表示，具体如图 2 所示。这幅有向图有 6 个节点，节点之间存在 16 条有向边。

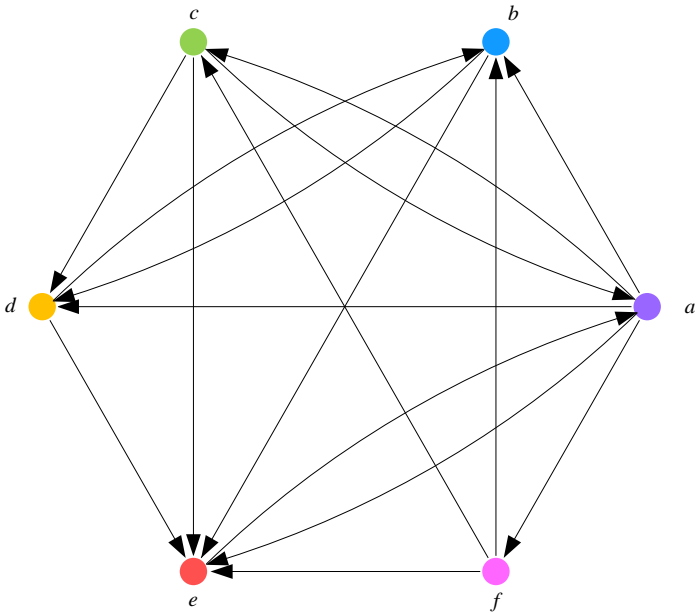


图 2. 6 个网页之间的有向图

邻接矩阵

图 2 这幅有向图的邻接矩阵为

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

(1)

图 3 所示为邻接矩阵热图。

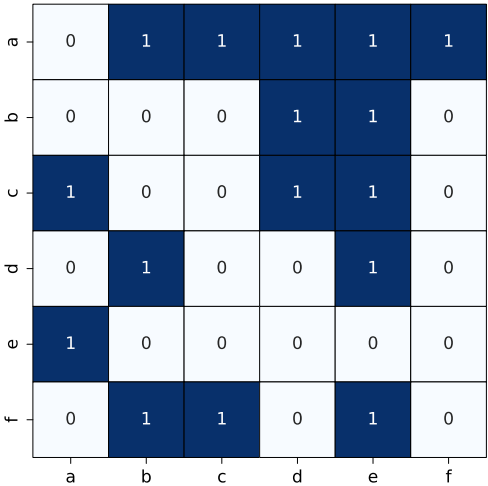


图 3. 有向图对应邻接矩阵的热图

代码 1 创建图 2，并计算有向图对应的邻接矩阵，下面聊聊其中关键语句。

- a 用 `networkx.DiGraph()` 创建有向图对象。
- b 用 `add_nodes_from()` 在有向图对象中增加节点。
- c 用 `add_edges_from()` 在有向图对象中增加几组有向边，节点有先后顺序。
- d 用 `networkx.circular_layout()` 创建节点在平面上的圆形布局。
- e 创建节点颜色的列表，用在 `networkx.draw_networkx()` 的参数 `node_color`。
- f 用于生成有向图的邻接矩阵，并通过调用 `.todense()` 将这个邻接矩阵转换为一个密集矩阵形式。

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import seaborn as sns

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

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

# 添加几组有向边
c directed_G.add_edges_from([( 'a', 'b'), ( 'a', 'c'), ( 'a', 'd'),
                              ( 'a', 'e'), ( 'a', 'f')])
directed_G.add_edges_from([( 'b', 'd'), ( 'b', 'e')])
directed_G.add_edges_from([( 'c', 'a'), ( 'c', 'd'), ( 'c', 'e')])
directed_G.add_edges_from([( 'd', 'b'), ( 'd', 'e')])
directed_G.add_edges_from([( 'e', 'a')])
directed_G.add_edges_from([( 'f', 'b'), ( 'f', 'c'), ( 'f', 'e')])

d pos = nx.circular_layout(directed_G)
e node_color = ['purple', 'blue', 'green', 'orange', 'red', 'pink']

# 可视化
plt.figure(figsize = (6,6))
nx.draw_networkx(directed_G,
                  pos = pos,
                  node_color = node_color,
                  node_size = 180)
plt.savefig('网页之间关系的有向图.svg')

# 邻接矩阵
f A = nx.adjacency_matrix(directed_G).todense()

sns.heatmap(A, cmap = 'Blues',
             annot = True, fmt = '.0f',
             xticklabels = list(directed_G.nodes),
             yticklabels = list(directed_G.nodes),
             linecolor = 'k', square = True,
             linewidths = 0.2)
plt.savefig('邻接矩阵.svg')
```

代码 1. 创建网页关系的有向图 | Bk6\_Ch24\_01.ipynb

转移矩阵

把邻接矩阵转化为转移矩阵  $T$

$$T = \begin{bmatrix} 0 & 0 & 1/3 & 0 & 1 & 0 \\ 1/5 & 0 & 0 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 1/3 \\ 1/5 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 1/2 & 1/3 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

图 4 所示为转移矩阵热图。大家应该还记得本书前文讲过，邻接矩阵  $A$  每行求和便得到有向图节点的出度；矩阵  $A$  每行除以对应出度结果再转置便得到上述转移矩阵。图 5 所示为从邻接矩阵到转移矩阵的计算过程。

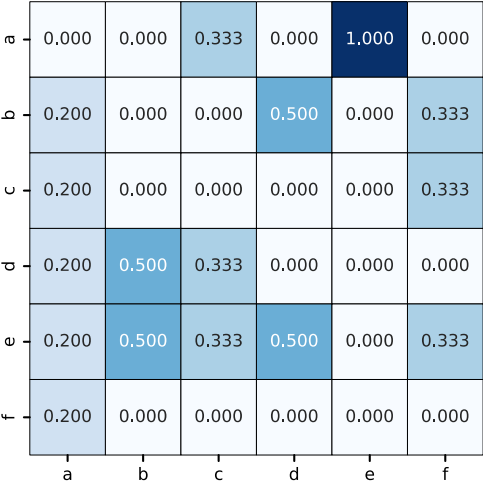


图 4. 有向图对应转移矩阵的热图

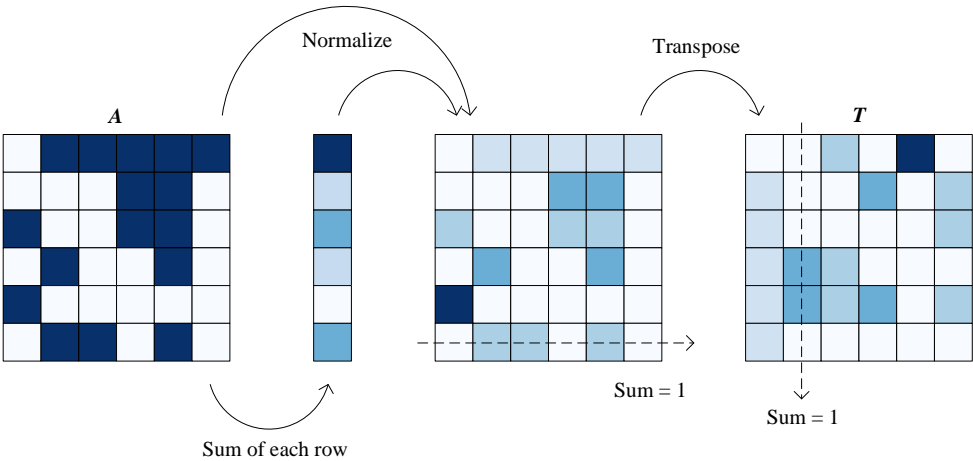


图 5. 从邻接矩阵到转移矩阵的计算过程

矩阵乘法角度理解转移矩阵

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)

下面聊聊在 PageRank 算法的语境下几个不同角度理解转移矩阵  $T$ 。

首先来看转移矩阵  $T$  的第 1 列，这一列有 5 个非零元素，值都是  $1/5$ 。如图 6 所示，网页  $a$  指向其他 5 个网页。节点  $a$  的出度为 5，取倒数得到  $1/5$ ，这样转移矩阵  $T$  的第 1 列元素之和为 1。

换个角度来看，从网页  $a$  出发有  $1/5$  可能性到达其他 5 个节点：

$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 1 & 0 \\ 1/5 & 0 & 0 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 1/3 \\ 1/5 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 1/2 & 1/3 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \\ 1/5 \end{bmatrix} \quad (3)$$

也就是说，如果网页  $a$  的影响力为 1，它将影响力均分为 5 份，每份  $1/5$ ，分别给  $b$ 、 $c$ 、 $d$ 、 $e$ 、 $f$  这 5 个网页。

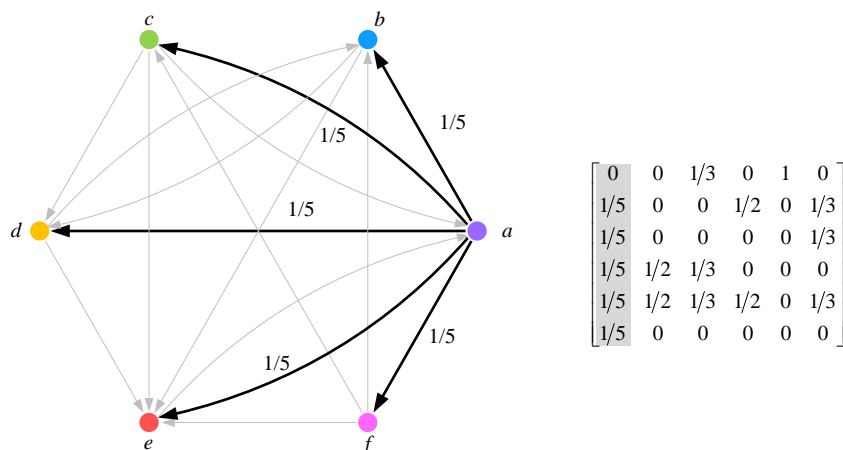


图 6. 网页  $a$  指向其他网页

转移矩阵  $T$  的第 2 列有 2 个非零元素，值都是  $1/2$ ，对应图 7。对于节点  $b$ ，它指向 2 个网页 ( $d$ 、 $e$ )。从出度角度来看，节点  $b$  的出度为 2，取倒数结果为  $1/2$ 。

从网页  $b$  出发，到达其他网页的可能性为：

$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 1 & 0 \\ 1/5 & 0 & 0 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 1/3 \\ 1/5 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 1/2 & 1/3 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1/2 \\ 1/2 \\ 0 \end{bmatrix} \quad (4)$$

如果网页  $b$  的影响力为 1，它将影响力均分为 2 份，每份  $1/2$ ，分别给  $d$ 、 $e$  这 2 个网页。

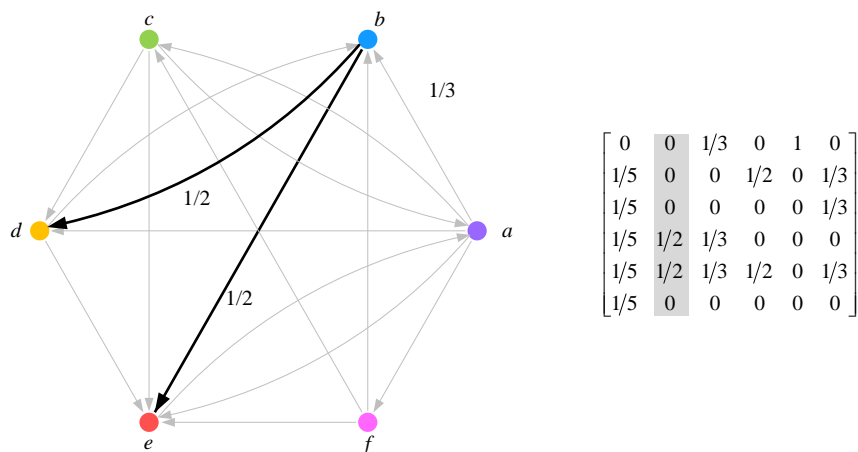
图 7. 网页  $b$  指向其他网页

图 8 对应如下矩阵乘法,

$$\begin{bmatrix}
 0 & 0 & 1/3 & 0 & 1 & 0 \\
 1/5 & 0 & 0 & 1/2 & 0 & 1/3 \\
 1/5 & 0 & 0 & 0 & 0 & 1/3 \\
 1/5 & 1/2 & 1/3 & 0 & 0 & 0 \\
 1/5 & 1/2 & 1/3 & 1/2 & 0 & 1/3 \\
 1/5 & 0 & 0 & 0 & 0 & 0
 \end{bmatrix}
 \begin{bmatrix}
 0 \\
 0 \\
 1 \\
 0 \\
 0 \\
 0
 \end{bmatrix}
 =
 \begin{bmatrix}
 1/3 \\
 0 \\
 0 \\
 1/3 \\
 1/3 \\
 0
 \end{bmatrix}
 \quad (5)$$

如果网页  $c$  的影响力为 1, 它将影响力均分为 3 份, 每份  $1/3$ , 分别给  $a$ 、 $d$ 、 $e$  这 3 个网页。

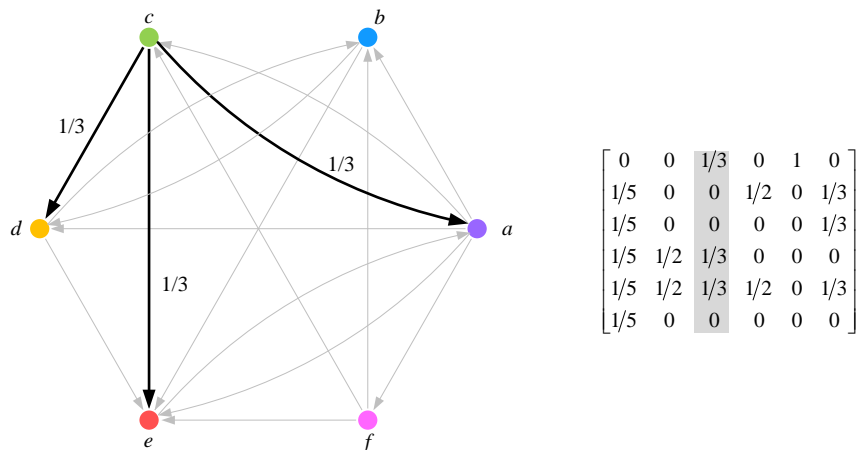
图 8. 网页  $c$  指向其他网页

图 9 对应如下矩阵乘法,



$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 1 & 0 \\ 1/5 & 0 & 0 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 1/3 \\ 1/5 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 1/2 & 1/3 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/2 \\ 0 \\ 0 \\ 1/2 \\ 0 \end{bmatrix} \quad (6)$$

如果网页  $d$  的影响力为 1，它将影响力均分为 2 份，每份  $1/2$ ，分别给  $b$ 、 $e$  这 2 个网页。

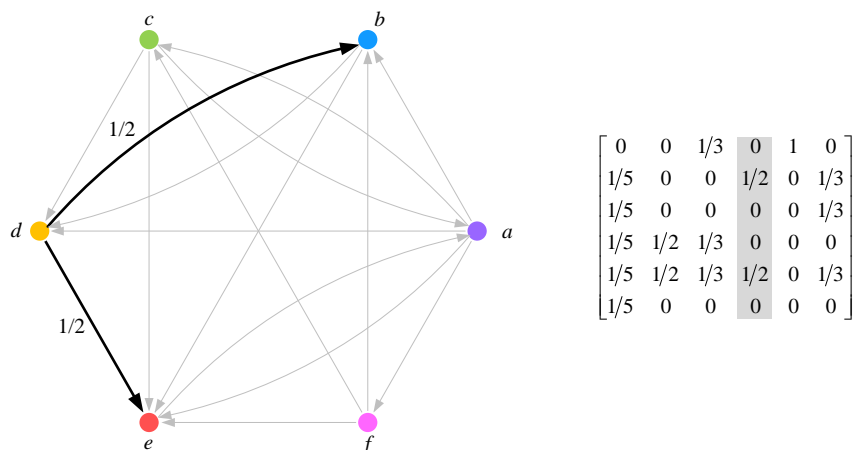


图 9. 网页  $d$  指向其他网页

图 10 对应如下矩阵乘法，

$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 1 & 0 \\ 1/5 & 0 & 0 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 1/3 \\ 1/5 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 1/2 & 1/3 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (7)$$

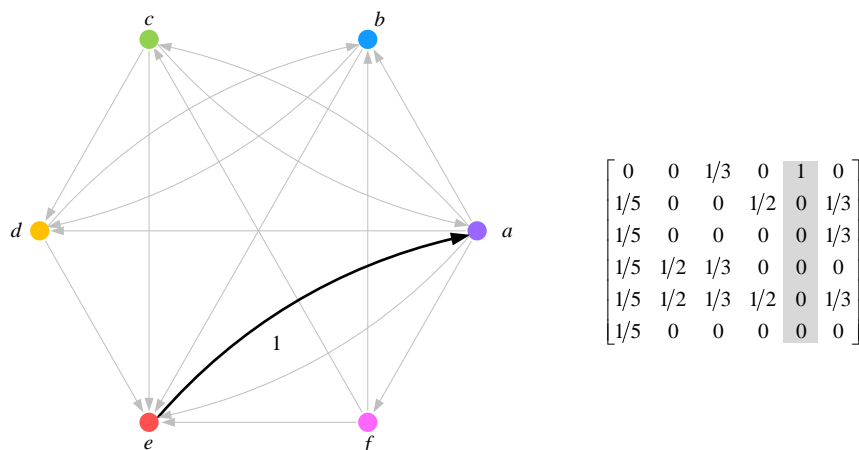


图 10. 网页 e 指向其他网页

图 11 对应如下矩阵乘法,

$$\begin{bmatrix}
 0 & 0 & 1/3 & 0 & 1 & 0 \\
 1/5 & 0 & 0 & 1/2 & 0 & 1/3 \\
 1/5 & 0 & 0 & 0 & 0 & 1/3 \\
 1/5 & 1/2 & 1/3 & 0 & 0 & 0 \\
 1/5 & 1/2 & 1/3 & 1/2 & 0 & 1/3 \\
 1/5 & 0 & 0 & 0 & 0 & 0
 \end{bmatrix}
 \begin{bmatrix}
 0 \\
 1/3 \\
 1/3 \\
 0 \\
 1/3 \\
 1
 \end{bmatrix}
 =
 \begin{bmatrix}
 0 \\
 1/3 \\
 1/3 \\
 0 \\
 1/3 \\
 0
 \end{bmatrix}
 \quad (8)$$

请大家从矩阵乘法、出度、影响力传递角度自行仔细分析图 7 ~ 图 11。

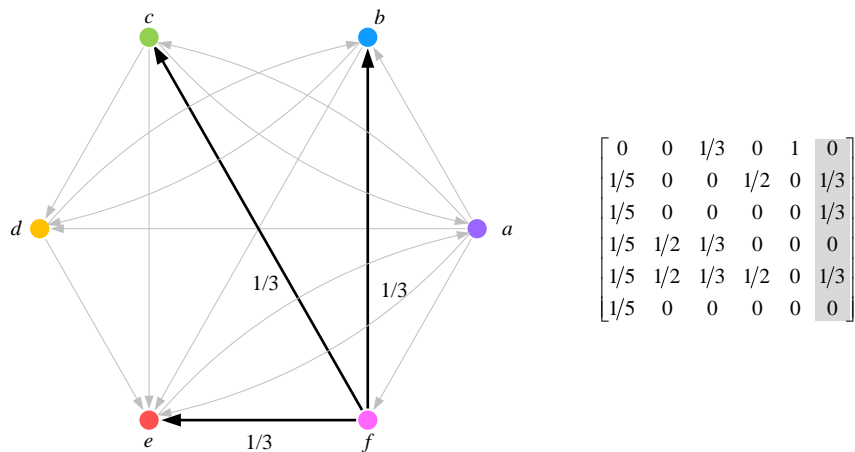


图 11. 网页 f 指向其他网页

代码 2 将邻接矩阵转化成转移矩阵，下面聊聊以下几句代码。

**a** `A.sum(axis=1)` 计算 A 中每一行的元素和。参数 `axis=1` 表示沿着行的方向进行求和。在图的邻接矩阵中，这等价于计算每个节点の出度，即从该节点出发的边的数量。

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)

`[:, np.newaxis]` 将前面步骤得到的一维数组转换成二维数组的形式，具体来说，是增加了一个新的轴，使得原本的一维数组变成了列向量。`np.newaxis` 用于在指定位置增加一个轴，这里是将一维数组变形为列向量，即把原本的形状  $(n,)$  转换为  $(n, 1)$ ，其中  $n$  是节点的数量。这样，`deg_out` 就变成了一个列向量，其每一行的元素代表对应节点的出度。

② 利用广播原则在行方向归一化邻接矩阵，结果的每一行元素之和为 1。这个结果也是一个转移矩阵，只不过转置之后才能得到本书常用的转移矩阵形式，即 ③。

```

a deg_out = A.sum(axis=1)[:, np.newaxis]
  # 节点出度

b T_T = A / deg_out
  # 邻接矩阵的行归一化

c T = T_T.T
  # 转置获得转移矩阵

sns.heatmap(T, cmap = 'Blues',
             annot = True, fmt = '.3f',
             xticklabels = list(directed_G.nodes),
             yticklabels = list(directed_G.nodes),
             linecolor = 'k', square = True,
             linewidths = 0.2)
plt.savefig('转移矩阵.svg')

```

代码 2. 计算转移矩阵 | Bk6\_Ch24\_01.ipynb

## 24.2 线性方程组

直觉告诉我们，一个被高排名网页指向的网页肯定也很重要。这便引出 PageRank 算法的两个基本假设：

- ▶ 数量假设：如果一个页面节点接收到的其他网页指向的入链数量越多，那么这个页面越重要。
- ▶ 质量假设：质量高（影响力大）的页面会通过链接向其他页面传递更多的权重。也就是说，质量高的页面指向某个页面，则该页面越重要。

下面，我们需要做的就是想办法量化排名。

通过前文有关转移矩阵内容的学习，大家应该知道，在一定条件下，如果网页之间的影响力相互传递存在稳态的话，各个节点的稳态概率就是 PageRank 值。

假设，最终计算得到网页  $a$ 、 $b$ 、 $c$ 、 $d$ 、 $e$ 、 $f$  的 PageRank 值分别为  $r_a$ 、 $r_b$ 、 $r_c$ 、 $r_d$ 、 $r_e$ 、 $r_f$ ，我们可以构造如下向量  $\mathbf{r}$ ，

$$\mathbf{r} = \begin{bmatrix} r_a \\ r_b \\ r_c \\ r_d \\ r_e \\ r_f \end{bmatrix} \quad (9)$$

稳态存在的话， $T\mathbf{r} = \mathbf{r}$ ，即

$$T\mathbf{r} = \begin{bmatrix} 0 & 0 & 1/3 & 0 & 1 & 0 \\ 1/5 & 0 & 0 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 1/3 \\ 1/5 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 1/2 & 1/3 & 1/2 & 0 & 1/3 \\ 1/5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} r_a \\ r_b \\ r_c \\ r_d \\ r_e \\ r_f \end{bmatrix} = \begin{bmatrix} r_a \\ r_b \\ r_c \\ r_d \\ r_e \\ r_f \end{bmatrix} = \mathbf{r} \quad (10)$$

将上述矩阵乘法展开得到如下线性方程组

$$\begin{cases} 1/3 r_c + r_e = r_a \\ 1/5 r_a + 1/2 r_d + 1/3 r_f = r_b \\ 1/5 r_a + 1/3 r_f = r_c \\ 1/5 r_a + 1/2 r_b + 1/3 r_c = r_d \\ 1/5 r_a + 1/2 r_b + 1/3 r_c + 1/2 r_d + 1/3 r_f = r_e \\ 1/5 r_a = r_f \end{cases} \quad (11)$$

显然，我们可以求解上述方程组。

但是我们并不急着求解结果，理解方程组每个等式的意义更重要。

### 理解 6 个等式

首先，让我们看 (11) 第 1 个等式

$$1/3 r_c + r_e = r_a \quad (12)$$

由于排名  $r_a$  相当于网页  $a$  的“影响力”，而网页  $a$  的影响力来自于网页  $c$ 、 $e$  传递来的权重，分别为  $1/3 r_c$ 、 $r_e$ ，具体如图 12 所示。特别地，网页  $e$  把自己所有的影响力都传递给了  $a$ 。

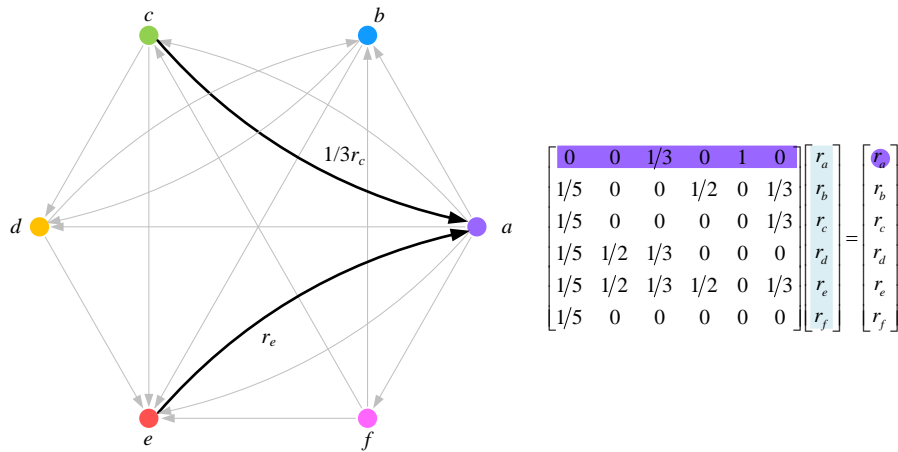


图 12. 网页 a 接受其他网页传递来的权重

下式是 (11) 的第 2 个等式,

$$1/5 r_a + 1/2 r_d + 1/3 r_f = r_b \quad (13)$$

这意味着, 网页 b 的影响力来自 3 个网页 (a、d、f)。

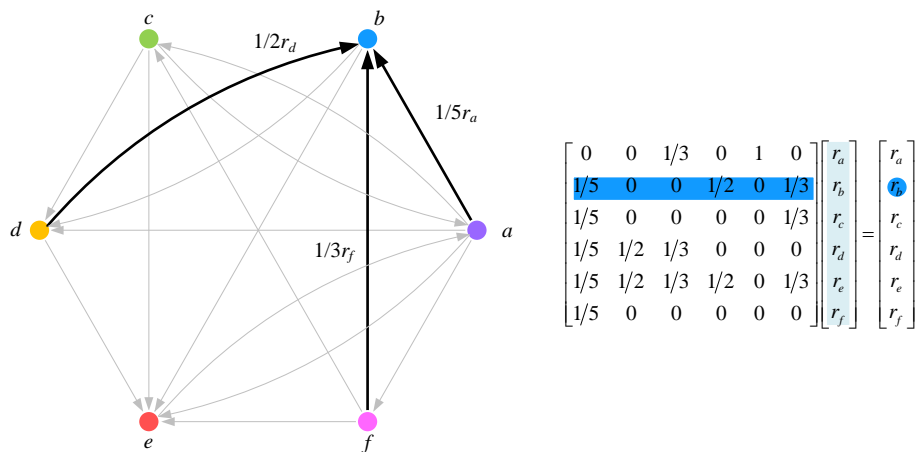


图 13. 网页 b 接受其他网页传递来的权重

如图 14 所示, 网页 c 的影响力来自 a、f:

$$1/5 r_a + 1/3 r_f = r_c \quad (14)$$

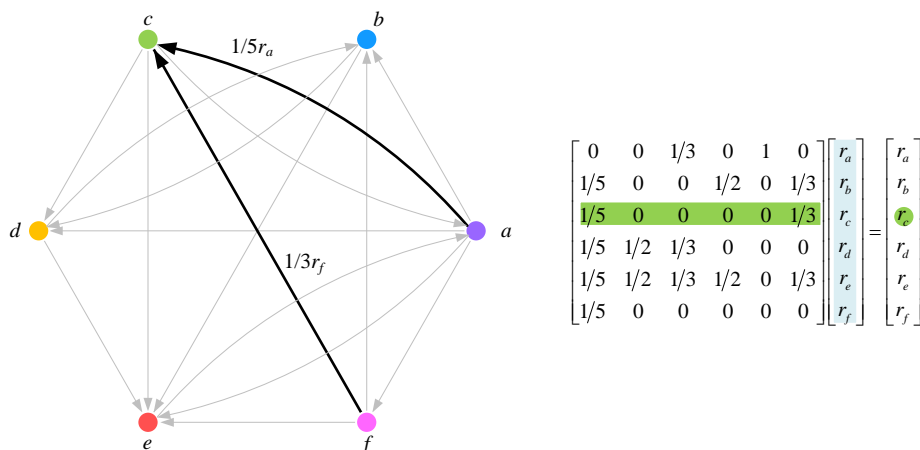


图 14. 网页 c 接受其他网页传递来的权重

如图 15 所示，网页 d 的影响力来自 a、b、c：

$$1/5 r_a + 1/2 r_b + 1/3 r_c = r_d \quad (15)$$

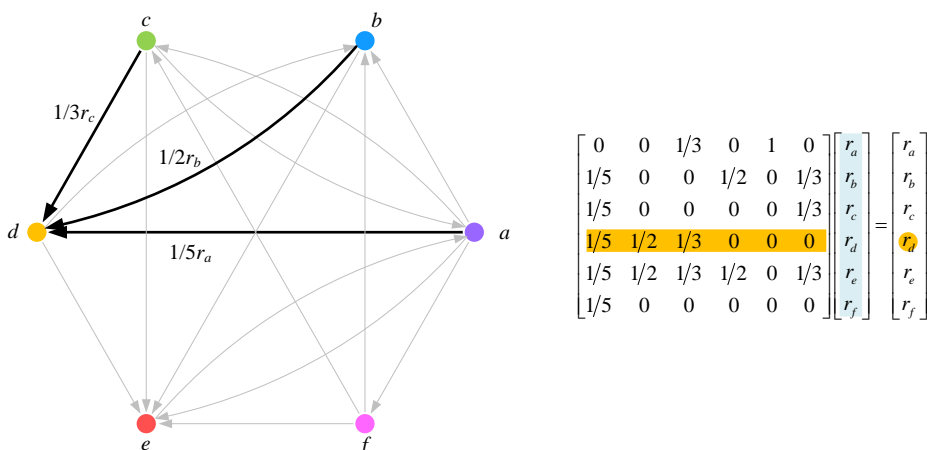


图 15. 网页 d 接受其他网页传递来的权重

如图 16 所示，网页 e 的影响力构成最为复杂：

$$1/5 r_a + 1/2 r_b + 1/3 r_c + 1/2 r_d + 1/3 r_f = r_e \quad (16)$$

指向网页 e 的网页很多，显然网页 e 很重要；但是网页 e 又显得很“吝啬”，爱惜羽毛，仅仅指向网页 a。

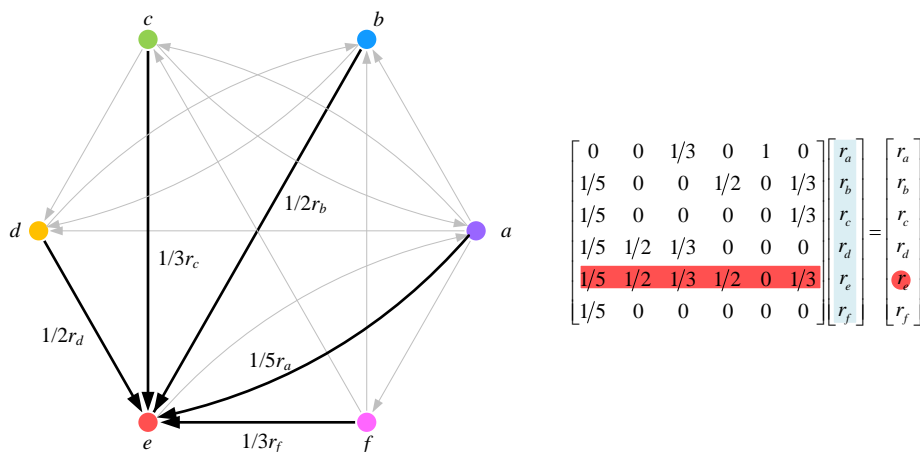
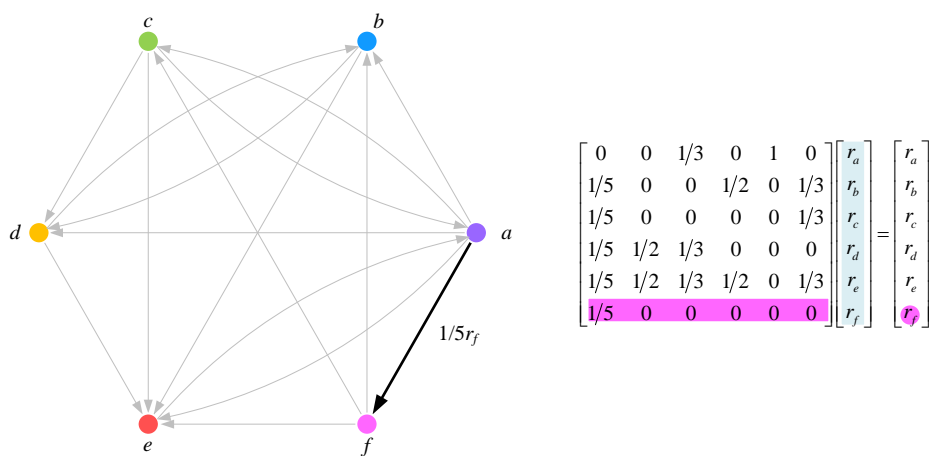
图 16. 网页  $e$  接受其他网页传递来的权重

图 17 对应的等式为

$$1/5 r_a = r_f \quad (17)$$

网页  $f$  的影响力仅仅来自  $a$ ；即便网页  $a$  的 PageRank 值大，也就是说排名很高，网页  $f$  的 PageRank 值也未必高。

图 17. 网页  $f$  接受其他网页传递来的权重

读完本节和上一节，大家可能已经发现我们用的分析方法本质上就是矩阵乘法的不同视角。

## 24.3 幂迭代

PageRank 本质上是基于有向图的随机漫步，数学模型可以抽象为一阶马尔可夫链。如果马尔可夫链满足特定条件，这个随机漫步最终会收敛到一个平稳分布。在这个平稳分布中，每个节点被访问的概率便是对应网页的 PageRank 值。

**幂迭代** (Power Iteration) 是计算 PageRank 的一种常用方法，其基本思想是通过迭代计算，找到 PageRank 向量的稳定分布。

幂迭代的基本思想是通过迭代地调整每个网页的 PageRank 值，使其趋于稳定。在实际计算中，通常会对 PageRank 向量进行归一化，这样可以更好地体现网页的相对重要性。

幂迭代算法的计算过程可以分为以下几个步骤：

- ▶ 初始化：为每个网页分配一个初始的 PageRank 值。通常，所有网页的 PageRank 值之和为 1。
- ▶ 迭代计算：通过多次迭代计算，不断更新每个网页的 PageRank 值。每次迭代都会考虑网页之间的链接关系，根据链接数量和质量来调整 PageRank 值。
- ▶ 收敛检测：在每次迭代后，检查 PageRank 值是否趋于稳定，即是否收敛。如果收敛，算法停止；否则，继续迭代。
- ▶ 计算结果：当算法收敛时，每个网页的 PageRank 值就是其最终的权重，可以根据这些值对网页进行排序。

图 18 所示为利用幂迭代求解本章前文网页排名问题的结果。当然，我们也可以使用特征值分解求解这个问题，本章配套代码给出相关实践。

图 18 这个排名也复合前文的分析。

先看前两名。

首先网页  $e$  的排名肯定不低，因为有 5 个网页指向网页  $e$ 。

而网页  $e$  的唯一引出网页指向了  $a$ ，而网页  $a$  在收到网页  $e$  的所有影响力基础上又叠加了来自网页  $c$  的部分影响力；因此，不难理解网页  $a$  的得分略高于  $e$ 。

再分析后两名。

网页  $a$  影响力的  $1/5$  分给了网页  $f$ ，这是  $f$  接受的唯一影响力，因此网页  $f$  排名垫底。

网页  $c$  也接受了网页  $a$  分给的  $1/5$  影响力，但是也有来自网页  $f$  的  $1/3$  影响力；因此，网页  $c$  排名略高于网页  $f$ 。

中间  $b$ 、 $d$  两个网页排名差距不大；两者都是“3 进 2 出”。

$b$  影响力来自于  $a$ 、 $d$ 、 $f$ 。

$d$  影响力来自于  $a$ 、 $b$ 、 $c$ 。两者都有来自  $a$  的等量影响力，这个因素排除。它们相互接受对方的影响力，这起到的是均衡作用；也就是说，因为这对有向边的存在，排名高的变低些，排名低的拉高些。因此，造成两者排名的因素就落在  $f$ 、 $c$  上了。恰好， $f$ 、 $c$  分别贡献给  $b$ 、 $d$  各自的  $1/3$  影响力。 $f$ 、 $c$  的排名则直接影响了  $b$ 、 $d$  排名。



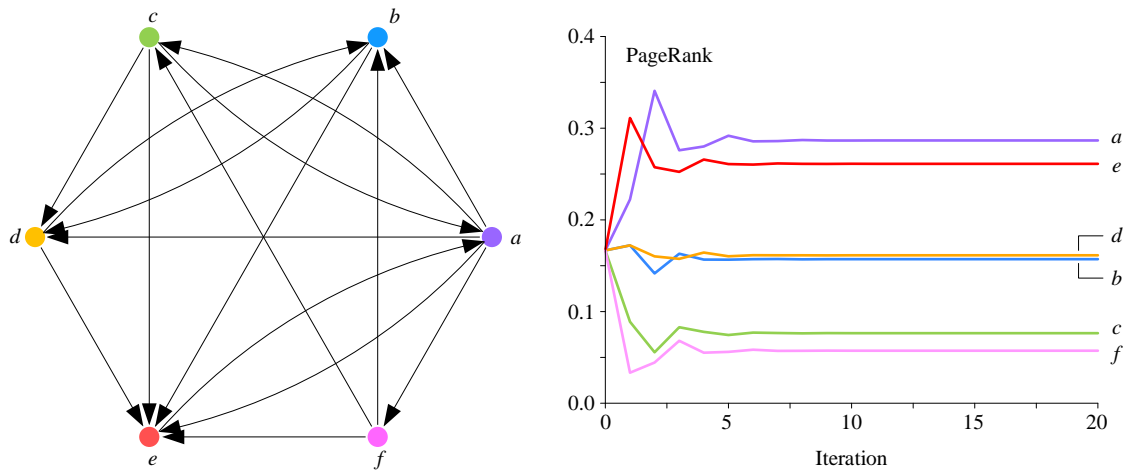


图 18. 幂迭代求解网页排名

```

# 自定义函数幂迭代
def power_iteration(T_, num_iterations: int, L2_norm = False):

    # 初始状态
    r_k = np.ones((len(T_),1))/len(T_)
    r_k_iter = r_k

    for _ in range(num_iterations):

        # 矩阵乘法 T @ r
        r_k1 = T_ @ r_k

        if L2_norm:
            # 计算L2范数
            r_k1_norm = np.linalg.norm(r_k1)

            # L2范数单位化
            r_k = r_k1 / r_k1_norm
        else:
            # 归一化
            r_k = r_k1 / r_k1.sum()

        # 记录迭代过程结果
        r_k_iter = np.column_stack((r_k_iter,r_k))

    return r_k,r_k_iter

# 调用自行函数完成幂迭代
r_k,r_k_iter = power_iteration(T, 20)

# 可视化幂迭代过程

fig, ax = plt.subplots()
for i,node_i in zip(range(len(node_color)),list(directed_G.nodes)):
    ax.plot(r_k_iter[i,:], color = node_color[i], label = node_i)
ax.set_xlim(0,20)
ax.set_ylim(0,0.4)
ax.set_xlabel('Iteration')
ax.set_ylabel('PageRank')
ax.legend(loc = 'upper right')
plt.savefig('幂迭代.svg')

```

代码 3. 幂迭代 | Bk6\_Ch24\_01.ipynb

### 修正幂迭代

PageRank 算法中使用的幂迭代方法可能会失效，主要是在下列情况之一发生时。

- ▶ 陷阱：在图中，陷阱或悬挂节点是指没有出链接的节点。这些节点会导致 PageRank 的流失，因为算法是基于概率分配的，而悬挂节点没有出链接来分配这些概率值。这会导致迭代过程中概率分配的不一致，使得算法难以收敛。将图 2 中有向边 *ea* 删除后，我们得到图 19 这幅有向图。我们发现，*e* 没有出链接，*e* 像是一个陷阱。
- ▶ 排他性循环：如果图形成了一个完全封闭的循环，其中所有的 PageRank 值在循环内的节点之间传递，但没有足够的机制将其分配给循环外的节点，这可能导致幂迭代过程无法达到一个全局一致的 PageRank 值分配。

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。

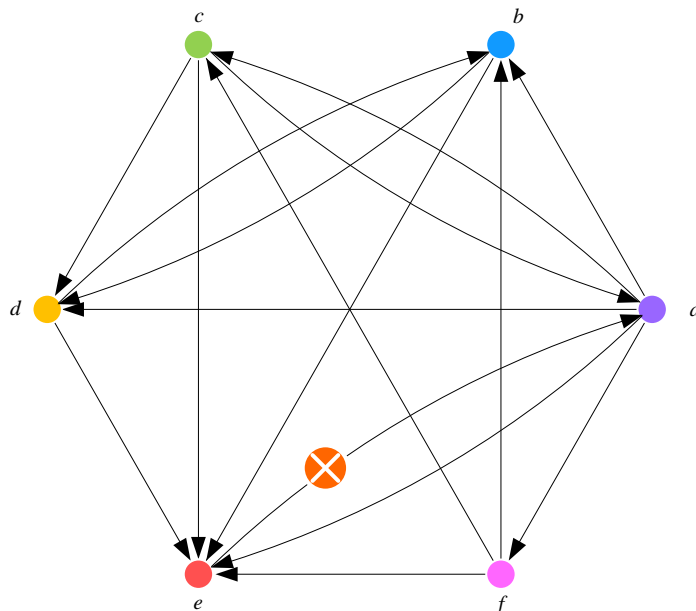
版权归清华大学出版社所有，请勿商用，引用请注明出处。

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)

► 分隔的子图：如果图中存在不相连的子图，即图不是完全连通的，那么 PageRank 算法可能会在各个分隔的子图中独立收敛，但无法在整个图范围内达到一致的 PageRank 值。这是因为分隔的子图之间没有链接，导致 PageRank 值无法在它们之间传递。

图 19. 删除有向边  $ea$ 

a	0	1	1	1	1	1
b	0	0	0	1	1	0
c	1	0	0	1	1	0
d	0	1	0	0	1	0
e	0	0	0	0	0	0
f	0	1	1	0	1	0
	a	b	c	d	e	f

图 20. 有向图对应邻接矩阵的热图，删除有向边  $ea$

a	0.000	0.000	0.333	0.000	0.000	0.000
b	0.200	0.000	0.000	0.500	0.000	0.333
c	0.200	0.000	0.000	0.000	0.000	0.333
d	0.200	0.500	0.333	0.000	0.000	0.000
e	0.200	0.500	0.333	0.500	0.000	0.333
f	0.200	0.000	0.000	0.000	0.000	0.000
	a	b	c	d	e	f

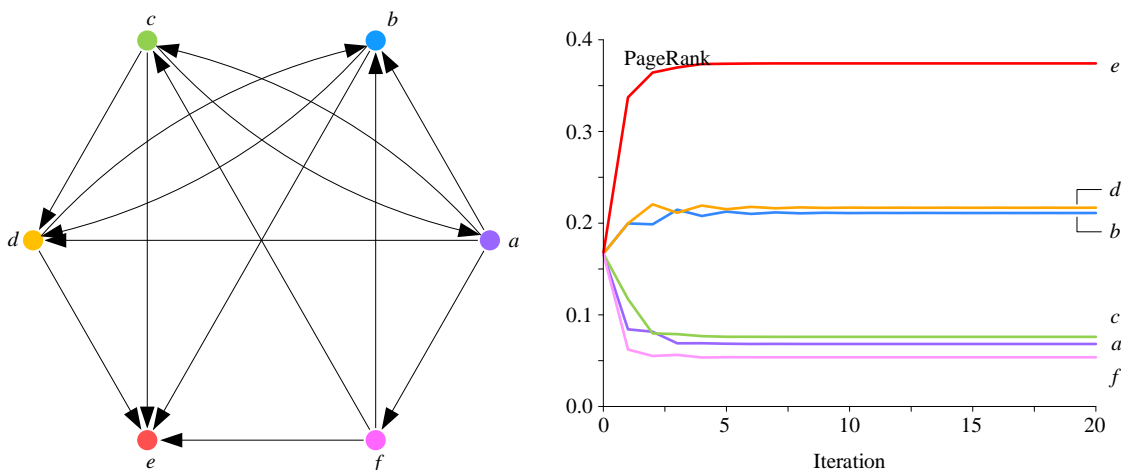
图 21. 有向图对应转移矩阵的热图，删除有向边  $ea$ 

为了解决这些问题，PageRank 算法引入了一个随机跳转的概念，通常通过一个称为**阻尼系数** (damping factor) 的参数实现，通常设置为 0.85。这意味着有 85% 的概率用户会按照链接继续浏览下一个页面，而有 15% 的概率用户会随机跳到任何一个页面。这个修正帮助算法避免了上述情况导致的失效问题，确保了算法能够收敛到一个稳定的分布。

具体迭代公式如下

$$\mathbf{r}_{t+1} = d\mathbf{Tr}_t + \frac{1-d}{n}\mathbf{1} \quad (18)$$

参数  $d$  就是阻尼系数，这样保证没有页面的 PageRank 值会是 0。

图 22. 修正幂迭代求解网页排名，删除有向边  $ea$

```

# 幂迭代, 修正
def power_iteration_adjust(T_, num_iterations: int, d = 0.85,
                           tol=1e-6, L2_norm = False):

    n = len(T_)
    # 初始状态
    r_k = np.ones((len(T_), 1))/n
    r_k_iter = r_k

    # 幂迭代过程
    for _ in range(num_iterations):

        # 核心迭代计算式
        r_k1 = d * T_ @ r_k + (1-d)/n

        # 检测是否收敛
        if np.linalg.norm(r_k - r_k1, 1) < tol:
            break

        if L2_norm:
            # 计算L2范数
            r_k1_norm = np.linalg.norm(r_k1)

            # L2范数单位化
            r_k = r_k1 / r_k1_norm
        else:
            # 归一化
            r_k = r_k1 / r_k1.sum()

        # 记录迭代过程结果
        r_k_iter = np.column_stack((r_k_iter, r_k))

    return r_k, r_k_iter

r_k_adj, r_k_iter_adj = power_iteration_adjust(T_2, 20, 0.85)

# 可视化修正幂迭代过程
fig, ax = plt.subplots()
for i, node_i in zip(range(len(node_color)), list(directed_G.nodes)):
    ax.plot(r_k_iter_adj[i, :], color = node_color[i], label = node_i)
ax.set_xlim(0, 20)
ax.set_ylim(0, 0.4)
ax.set_xlabel('Iteration')
ax.set_ylabel('PageRank')
ax.legend(loc = 'upper right')
plt.savefig('幂迭代, 修正.svg')

```

代码 4. 幂迭代修正 | Bk6\_Ch24\_01.ipynb

PageRank 虽然是为网页排名设计的算法，但是其核心思想现在被广泛应用在各种场景。PageRank 算法的学习过程，我们回顾了有向图、邻接矩阵、转移矩阵、马尔科夫链、线性方程组、幂迭代、特征值分解等数据工具。