谷歌如何从网络的大海里捞到针

David Austin/ 文　沈栋 / 译

想象一个含有 250 亿份文件，却没有集中管理机构和馆员的图书馆，而且任何人都可以在任何时间添加新的文件而不需要通知其他人。一方面你可以确定，这庞大的文件堆中有一份文件含有对你至关重要的信息，而另一方面，你又像我们中的大多数人那样没有耐心，想要在几秒钟之内就找到这条信息。你有什么办法呢？

摆在你面前的这个难题看起来似乎无法解决。而这个文件堆跟万维网（World Wide Web）其实相差无几，后者就是一个超大的、高度混乱的以各种形式存放的文件堆。当然，从万维网中找信息我们有办法解决，因为我们对搜索引擎非常熟悉（或许你就是通过搜索找到这篇文章的）。本文将介绍谷歌的网页排序算法（PageRank Algorithm），以及它如何从 250 亿份网页中捞到与你的搜索条件匹配的结果。它的匹配效果如此之好，以至于“谷歌”（google）今天已经成为一个被广泛使用的动词了。

包括谷歌在内，多数搜索引擎都是不断地运行计算机程序群，来检索网络上的网页、搜索每份文件中的词语并且将相关信息以高效的形式进行存储。每当用户检索一个短语，例如“搜索引擎”，搜索引擎就将找出所有含有被检索短语的网页。（或许，类似“搜索”与“引擎”之间的距离这样的额外信息都被会考虑在内。）但问题是，谷歌现在需要检索 250 亿个页面，而这些页面上大约 95% 的文本仅由大约一万个单词组成。也就是说，对于大多数搜索而言，将会有超级多的网页含有搜索短语中的单词。我们所需要的其实是这样一种办法，它能够将这些符合搜索条件的网页按照重要程度进行排序，这样才能够将最重要的页面排在最上面。

确定网页重要性的一个方法是使用人为排序。例如，你或许见过这样一些网页，他们包含了大量的链接，后者连接到某个特定兴趣领域的其他资源。假定维护这个网页的人是可靠的，那么他推荐的网页在很大程度上就可能有用。当然，这种做法也有其局限性，比如这个列表可能很快就过期了，也可能维护这个列表的人会无意或因某种未知的偏见而遗漏掉一些重要的网页。

谷歌的网页排序算法则不借助人为的内容评估来确定网页的重要性。事实上，谷歌发现，它的服务的价值很大程度上是它能够提供给用户无偏见的搜索结果。谷歌声称，“我们软件的核心就是网页排序（PageRank）。”正如我们将要看到的，技巧就是让网页自身按照重要性进行排序。

如何辨别谁重要

如果你曾建立过一个网页，你应该会列入一些你感兴趣的链接，它们很容易使你点击到其它含有重要、可靠信息的网页。这样就相当于你肯定了你所链接页面的重要性。谷歌的网页排序算法每月在所有网页中进行一次受欢迎程度的评估，以确定哪些网页最重要。网页排序算法的提出者，谢尔盖 • 布林 (Sergey Brin) 和拉里 • 佩奇 (Lawrence Page) 的基本想法是：一个网页的重要性是由链接到它的其他网页的数量及其重要性来决定。

我们对任意一个网页 P，以 $I(P)$ 来表述其重要性，并称之为网页的网页排序。在很多网站，你可以找到一个近似的网页排序值。（例如，美国数学会的首页目前的网页排序值为 8，最高分是 10。你可以试试找到一个网页排序值为 10 的网页吗？）这个网页排序值仅是一个近似值，因为谷歌拒绝提供真实的网页排序值，以阻止那些试图干扰排序的行为。

网页排序是这样确定的。假定网页 P_j 有 l_j 个链接。如果这些链接中的一个链接到网页 P_j，那么网页 P_j 将会将其重要性的 $1/l_j$ 赋给 P_j。网页 P_j 的重要性就是所有指向这个网页的其他网页所贡献的重要性的加和。换言之，如果我们记链接到网页 P_j 的网页集合为 B_i，那么

$$I(P_i)=\sum_{P_j\in B_i}\frac{I(P_j)}{l_j}.$$

这或许让你想起“先有鸡还是先有蛋”的问题：为了确定一个网页的重要性，我们首先得知道所有指向它的其他网页的重要性。然而，我们可将这个问题改写为一个更数学化的问题。

首先建立一个矩阵，称为超链矩阵（hyperlink matrix），$\mathbf{H}=[H_{ij}]$，其中第 i 行第 j 列的元素为

$$H_{ij}\begin{cases}\dfrac{1}{l_j} & \text{如果 } P_j\in B_j\\ 0 & \text{上述条件不成立}\end{cases}$$

注意到 $\mathbf{H}$ 有一些特殊的性质。首先，它所有的元都是

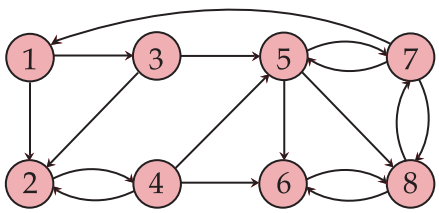
非负的。其次，除非对应这一列的网页没有任何链接，它的每一列的和为 1。所有元均非负且列和为 1 的矩阵称为随机矩阵，随机矩阵将在下述内容中起到重要作用。

我们还需要定义向量 $I=[I(p_i)]$，它的元素为所有网页的网页排序——重要性的排序值。前面定义的网页排序可以表述为

$$I = \mathbf{H}I$$

换言之，向量 I 是矩阵 **H** 对应特征值 1 的特征向量。我们也称之为矩阵 **H** 的平稳向量（stationary vector）。

让我们来看一个例子。下图所示为一个网页集合（8 个），箭头表示链接。

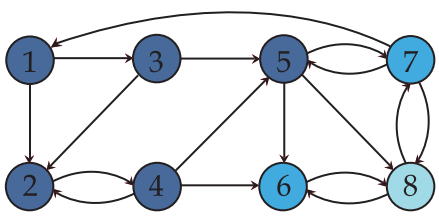


其相应的矩阵为 0

$$\mathbf{H}=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 1/2 & 0 & 1/2 & 1/3 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 1 & 1/3 & 0 \end{bmatrix},$$

其中平稳向量为 $I=\begin{bmatrix} 0.0600 \\ 0.0675 \\ 0.0300 \\ 0.0675 \\ 0.0975 \\ 0.2025 \\ 0.1800 \\ 0.2950 \end{bmatrix}$,

这说明网页 8 的受欢迎程度最高。下图是阴影化的图，其中网页排序值越高的网页阴影越浅。



计算平稳向量

有很多方法可以找到一个方阵的特征向量。然而，我们面对的是一个特殊的挑战，因为矩阵 **H** 是一个这样的方阵，它的每一列都对应谷歌检索到的一个网页。也就是说，大约有 n=250 亿行和列。不过其中大多数的元都是 0；事实上，研究表明每个网页平均约有 10 个链接，换言之，平均而言，每一列中除了 10 个元外全是 0。我们将选择被称为幂法（power method）的方法来找到矩阵 **H** 的平稳向量 I。

幂法如何实现呢？首先选择 I 的备选向量 I^0，进而按下式产生向量序列 I^k

$$I^{k+1}=\mathbf{H}I^k, \quad k=0, 1, 2, \ldots$$

这个方法是建立在如下的一般原理上：

一般原理：序列 I^k 将收敛到平稳向量 I。

我们首先用个例子验证上面的结论。

I^0	I^1	I^2	I^3	I^4	…	I^{60}	I^{61}
1	0	0	0	0.0287	…	0.06	0.06
0	0.5	0.25	0.1667	0.0833	…	0.0675	0.0675
0	0.5	0	0	0	…	0.03	0.03
0	0	0.5	0.25	0.1667	…	0.0675	0.0675
0	0	0.25	0.1667	0.1111	…	0.0975	0.0975
0	0	0	0.25	0.1806	…	0.2025	0.2025
0	0	0	0.0833	0.0972	…	0.18	0.18
0	0	0	0.0833	0.3333	…	0.295	0.295

一个自然的问题是，这些数字有什么含义。当然，关于一个网页的重要性，可能没有绝对的度量，而仅有比较两个网页的重要性的比例度量，如“网页 A 的重要性是网页 B 的两倍”。基于这一原因，我们可以用一个固定量去同乘以所有的重要性排序值，这并不会影响我们能获得的信息。这样，我们总是假定所有受欢迎程度值（popularity）的和为 1，原因稍后解释。

三个重要的问题

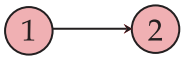
自然而然产生的三个问题是：

* 序列 I^k 总是收敛吗？（即运算多次后，I^k 和 I^{k+1} 几乎是一样的）
* 收敛后的平稳向量是否和初始向量 I^0 的选取没有关系？
* 重要性排序值是否包含了我们想要的信息？

对目前的方法而言，上述三个的答案都是否定的！下

面，我们将看看如何改进我们的方法，使得改进后的算法满足上述三个要求。

先看个非常简单的例子。考虑如下包含两个网页的小网络，其中一个链接到另一个：



其矩阵为 $H=\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$

下例展示了算法的运行过程：

I^0	I^1	I^2	$I^3=I$
1	0	0	0
0	1	0	0

在这个例子中，两个网页的重要性排序值均为 0，这样我们无法获知两个网页之间的相对重要性信息。问题在于网页 P_2 没有任何链接。因此，在每个迭代步骤中，它从网页 P_1 获取了一些重要性，但却没有赋给其他任何网页。这样将耗尽网络中的所有重要性。没有任何链接的网页称为悬挂点（dangling nodes），显然在我们要研究的实际网络中存在很多这样的点。稍后我们将看到如何处理这样的点，在此之前我们先考虑一种新的理解矩阵 **H** 和平稳向量 I 的思路。

H 的概率化解释

想象我们随机地在网上跳转网页；也就是说，当我们访问一个网页时，一秒钟后我们随机地选择当前网页的一个链接到达另一个网页。例如，我们正访问含有 l_j 个链接的网页 P_j，其中一个链接引导我们访问了网页 P_i，那么下一步转到网页 P_i 的概率就是 $1/l_j$。

由于跳转网页是随机的，我们用 T_j 表示停留在网页 P_j 上的时间。那么我们从网页 P_j 转到网页 P_i 的时间为 T_j/l_j。如果我们转到了网页 P_i，那么我们必然是从一个指向它的网页而来。这意味着

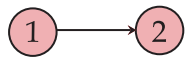
$$T_i=\sum_{p_j\in B_i}\frac{T_j}{l_j}$$

其中求和是对所有链接到 P_i 的网页 P_j 进行的。注意到这个方程与定义网页排序值的方程相同，因此 $I(P_i)=T_i$。那么一个网页的网页排序值可以解释为随机跳转时花在这个网页上的时间。如果你曾经上网浏览过某个你不熟悉的话题的相关信息时，你会有这种感觉：按照链接跳转网页，过一会你会发现，相较于其他网页，你会更频繁地回到某一部分网页。正如谚语所说“条条大路通罗马，”这部分网页显然是更重要的网页。

基于这个解释，很自然地可以要求网页排序向量 I 的所有元之和为 1。

当然，这种表述中还存在一个问题：如果我们随机地跳转网页，在某种程度上，我们肯定会被困在某个悬挂点上，这个网页没有给出任何链接。为了能够继续进行，我们需要随机地选取下一个网页；也就是说，我们假定悬挂点可以链接到其他任何一个网页。这个效果相当于将超链矩阵 **H** 做如下修正：将其中所有元都为 0 的列替换为所有元均为 1/n 的列，前者就对应于网页中的悬挂点。这样修正后悬挂点就不存在了。我们称修正后的新矩阵为 **S**。

我们之前的例子，现在就变成了



对应矩阵 $S=\begin{bmatrix} 0 & \frac{1}{2} \\ 1 & \frac{1}{2} \end{bmatrix}$ 及特征向量 $I=\begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix}$

换言之，网页 P_2 的重要性是网页 P_1 的两倍，符合你的直观认知了。

矩阵 **S** 有一个很好的性质，即其所有元均非负且每列的和均为 1。换言之，**S** 为随机矩阵。随机矩阵具有一些很有用的性质。例如，随机矩阵总是存在平稳向量。

为了稍后的应用，我们要注意到 **S** 是由 **H** 通过一个简单的修正得到。定义矩阵 **A** 如下：对应于悬挂点的列的每个元均为 1/n，其余各元均为 0。则 **S** = **H**+**A**。

幂法如何实现？

一般而言，幂法是寻找矩阵对应于绝对值最大的特征值的特征向量。就我们而言，我们要寻找矩阵 **S** 对应于特征值 1 的特征向量。首先要说到的是最好的情形。在这种情形下，其他特征值的绝对值都小于 1；也就是说，矩阵 **S** 的其它特征值都满足 $|\lambda|<1$。

我们假定矩阵 **S** 的特征值为 λ_j 且

$$1=\lambda_1>|\lambda_2|\geq|\lambda_3|\geq\cdots\geq|\lambda_n|,$$

对矩阵 **S**，假设对应于特征值 λ_j 的特征向量存在一个基向量 v_j。这一假设在一般情况下并不一定要成立，但如果成立可以帮助我们更容易地理解幂法如何实现。将初始向量 I^0 写成如下形式

$$I^0=c_1v_1+c_2v_2+\cdots+c_nv_n.$$

那么

$$I^1=SI^0=c_1v_1+c_2\lambda_2v_2+\cdots+c_n\lambda_nv_n,$$

$$I^2=SI^1=c_1v_1+c_2\lambda_2^2v_2+\cdots+c_n\lambda_n^2v_n,$$

$$\cdots\quad\cdots$$

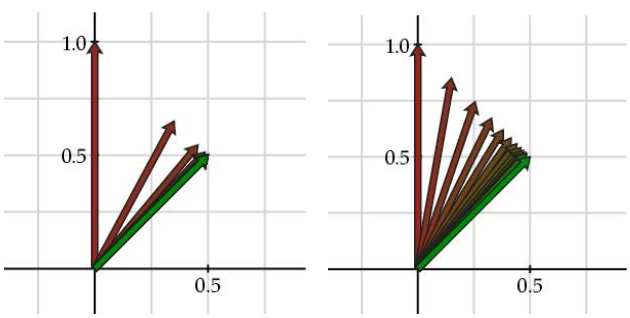
$$I^k=SI^{k-1}=c_1v_1+c_2\lambda_2^kv_2+\cdots+c_n\lambda_n^kv_n.$$

当 $j\geq 2$ 时，因为所有特征值的绝对值小于 1，因此这是 $\lambda_j^k \to 0$。从而 $I^k \to I=c_1v_1$，后者是对应于特征值 1 的一个特征向量。

需要指出的是，$I^k \to I$ 的速度由 $|\lambda_2|$ 确定。当 $|\lambda_2|$ 比较接近于 0 时，那么 $\lambda_2^k \to 0$ 会相当快。例如，考虑下述矩阵

$$\mathbf{S}=\begin{bmatrix} 0.65 & 0.35 \\ 0.35 & 0.65 \end{bmatrix}$$

这个矩阵的特征值为 $\lambda_1=1$ 及 $\lambda_2=0.3$。下图左可以看出用红色标记的向量 I^k 收敛到用绿色标记的平稳向量 I。



再考虑矩阵

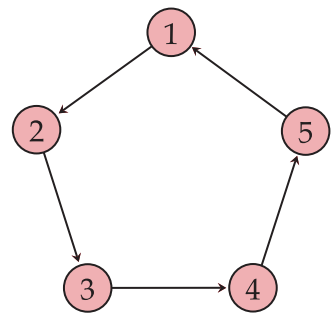
$$\mathbf{S}=\begin{bmatrix} 0.85 & 0.15 \\ 0.15 & 0.85 \end{bmatrix}$$

其特征值为 $\lambda_1=1$ 及 $\lambda_2=0.7$。从上图右可以看出，本例中向量 I^k 收敛到平稳向量 I 的速度要慢很多，因为它的第二个特征值较大。

不顺之时

在上述讨论中，我们假定矩阵 **S** 需要满足条件 $\lambda_1=1$ 和 $|\lambda_2|<1$。然而，我们可能会发现，这一点并不总成立。

假定网络关系如下：



在这种情形下，矩阵 **S** 为

$$\mathbf{S}=\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

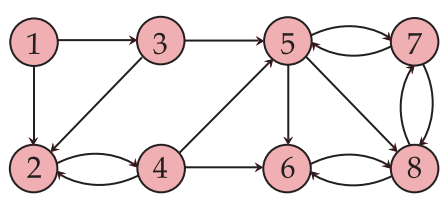
那么我们可以得到

I^0	I^1	I^2	I^3	I^4	I^5
1	0	0	0	0	1
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0

在这种情况下，向量序列 I^k 不再收敛。这是为什么？注意到矩阵 **S** 的第二个特征值满足 $|\lambda_2|=1$，因此前述幂法的前提不再成立。

为了保证 $|\lambda_2|<1$，我们需要矩阵 **S** 为本原（primitive）矩阵。这意味着，对某个 m，$\mathbf{S}^m$ 的所有元均为正。换言之，若给定两个网页，那么从第一个网页经过 m 个链接后可以到达第二个网页。显然，上述最后的这个例子并不满足这个条件。稍后，我们将看到如何修正矩阵 **S** 以获得一个本原随机矩阵，从而满足 $|\lambda_2|<1$。

下面说明我们的方法行不通的另一个例子。考虑如下图所示的网络

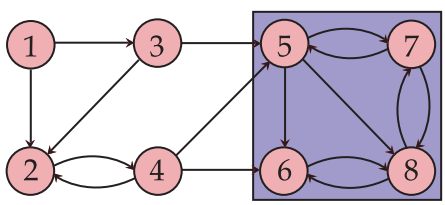


在此例中，矩阵 **S** 为

$$S=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 1/2 & 0 & 1/2 & 1/3 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/3 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 1 & 1/2 & 0 \end{bmatrix},$$

其中平稳向量为 $I=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0.12 \\ 0.24 \\ 0.24 \\ 0.4 \end{bmatrix}$

注意到前四个网页的网页排序值均为 0。这使我们感觉不太对：每个页面都有其它网页链接到它，显然总有人喜欢这些网页！一般来说，我们希望所有网页的重要性排序值均为正。这个例子的问题在于，它包含了一个小网络，即下图中蓝色方框部分。



在这个方框中，有链接进入到蓝色方框，但没有链接转到外部。正如前述中关于悬挂点的例子一样，这些网页构成了一个“重要性水槽”，其他四个网页的重要性都被“排”到这个“水槽”中。这种情形发生在矩阵 **S** 为可约（reducible）时；也即，可以写成如下的块形式

$$\mathbf{S}=\begin{bmatrix} * & 0 \\ * & * \end{bmatrix}$$

实际上，我们可以证明：如果矩阵 **S** 不可约，则一定存在一个所有元均为正的平稳向量。

对一个网络，如果任意给定两个网页，一定存在一条由链接构成的路使得我们可以从第一个网页转到第二个网页，那么称这个网络是强连通的（strongly connected）。显然，上面最后的这个例子不是强连通的。而强连通的网络对应的矩阵 **S** 是不可约的。

简言之，矩阵 **S** 是随机矩阵，即意味着它有一个平稳向量。然而，我们同时还需要 **S** 满足 (a) 本原，从而 $|\lambda_2|<1$；(b) 不可约，从而平稳向量的所有元均为正。

最后一个修正

为得到一个本原且不可约的矩阵，我们将修正随机跳转网页的方式。就目前来看，我们的随机跳转模式由矩阵确定：或者是从当前网页上的链接中选择一个，或者是对没有任何链接的网页，随机地选取其他网页中的任意一个。为了做出修正，首先选择一个介于 0 到 1 之间的参数 α。然后假定随机跳转的方式略作变动。具体是，遵循矩阵的方式跳转的概率为 α，而随机地选择下一个页面的概率是 $1-\alpha$。

若记所有元均为 1 的 $n\times n$ 矩阵为 **J**，那么我们就可以得到谷歌矩阵（Google matrix）：

$$\mathrm{G}=\alpha\mathrm{S}+(1-\alpha)\frac{1}{n}\mathrm{J}$$

注意到 **G** 为随机矩阵，因为它是随机矩阵的组合。进而，矩阵 **G** 的所有元均为正，因此 **G** 为本原且不可约。从而，**G** 存在唯一的平稳向量 I，后者可以通过幂法获得。

参数 α 的作用是一个重要因素。若 $\alpha=1$，则 $\mathbf{G}=\mathbf{S}$。这意味着我们面对的是原始的网络超链结构。然而，若 $\alpha=0$，则 $\mathbf{G}=1/n\mathbf{J}$。也即我们面对的是一个任意两个网页之间都有连接的网络，它已经丧失了原始的网络超链结构。显然，我们将会把 α 的值取得接近于 1，从而保证网络的超链结构在计算中的权重更大。

然而，还有另外一个问题。请记住，幂法的收敛速度是由第二个特征值的幅值 $|\lambda_2|$ 决定的。而对谷歌矩阵，已经证明了第二个特征值的幅值为 $|\lambda_2|=\alpha$。这意味着当 α 接近于 1 时，幂法的收敛速度将会很慢。作为这个矛盾的折中方案，网页排序算法的提出者谢尔盖 • 布林和拉里 • 佩奇选择 α=0.85。

计算排序向量 I

到目前为止，我们所讨论的看起来是一个很棒的理论，然而要知道，我们需要将这个方法应用到一个维数 n 约为 250 亿的 $n\times n$ 矩阵！事实上，幂法特别适用于这种情形。

回想随机矩阵 **S** 可以写成下述形式

$$\mathrm{S}=\mathrm{H}+\mathrm{A}.$$

从而谷歌矩阵有如下形式

$$G = \alpha H + \alpha A \frac{1-\alpha}{n} J$$

其中 **J** 是元素全为 1 的矩阵，从而

$$GI^k = \alpha HI^k + \alpha AI^k \frac{1-\alpha}{n} JI^k.$$

现在注意到，矩阵 **H** 的绝大部分元都是 0；平均而言，一列中只有 10 个元是非零数。从而，求 $\mathbf{H}I^k$ 的每个元时，只需要知道 10 个项即可。而且，和矩阵 **J** 一样，矩阵 **A** 的行元素都是相同的。从而，求 $\mathbf{A}I^k$ 与 $\mathbf{J}I^k$ 相当于添加悬挂点或者所有网页的当前重要性排序值。而这只需要一次即可完成。

当 α 取值接近于 0.85，布林和佩奇指出，需要 50 到 100 次迭代来获得对向量 I 的一个足够好的近似。计算到这个最优值需要几天才能完成。

当然，网络是不断变化的。首先，网页的内容，尤其是新闻内容，变动频繁。其次，网络的隐含超链结构在网页或链接被加入或被删除时也要相应变动。有传闻说，谷歌大约 1 个月就要重新计算一次网页排序向量 I。由于在此期间可以看到网页排序值会有一个明显的波动，一些人便将其称为谷歌舞会（Google Dance）。

总结

布林和佩奇在 1998 年创建了谷歌，正值网络的增长步伐已经超过当时搜索引擎的能力范围。在那个时代，大多数的搜索引擎都是由那些没兴趣发布其产品运作细节的企业研发的。在发展谷歌的过程中，布林和佩奇希望“推动学术领域更多的发展和认识”。换言之，他们首先希望，将搜索引擎引入一个更开放的、更学术化的环境，来改进搜索引擎的设计。其次，他们感到其搜索引擎产生的统计数据能够为学术研究提供很多的有趣信息。看来，联邦政府最近试图获得谷歌的一些统计数据，也是同样的想法。

还有一些其他使用网络的超链结构来进行网页排序的算法。值得一提的例子是 HITS 算法，由乔恩 • 克莱因伯格（Jon Kleinberg）提出，它是 Teoma 搜索引擎的基础。事实上，一个有意思的事情是比较一下不同搜索引擎获得的搜索结果，这也可以帮助我们理解为什么有人会抱怨谷歌寡头（Googleopoly）。

原文链接：
http://www.ams.org/samplings/feature-column/fcarc-pagerank

参考文献

1. Michael Berry, Murray Browne, Understanding Search Engines: Mathematical Modeling and Text Retrieval. Second Edition, SIAM, Philadelphia. 2005.
2. Sergey Brin, Lawrence Page, The antaomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems, 33: 107-17, 1998. Also available online at http://infolab.stanford.edu/pub/papers/google.pdf
3. Kurt Bryan, Tanya Leise, The $25,000,000,000 eigenvector. The linear algebra behind Google. SIAM Review, 48 (3), 569-81. 2006. Also avaiable at http://www.rose-hulman.edu/~bryan/google.html
4. Google Corporate Information: Technology.
5. Haveliwala, Kamvar, The second eigenvalue of the Google matrix.
6. Amy Langville, Carl Meyer, Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press, 2006. This is an informative, accessible book, written in an engaging style. Besides providing the relevant mathematical background and details of PageRank and its implementation (as well as Kleinberg's HITS algorithm), this book contains many interesting "Asides" that give trivia illuminating the context of search engine design.