PageRank

在本节中，我们研究PageRank，这是一种使用Web图形中的链接结构按重要性对网页进行排名的方法。这是Google发明的网络搜索常用算法。在讨论PageRank之前，让我们先将网络概念化为图，然后尝试使用图论语言研究其结构。

The web as a graph

我们可以通过将网页作为节点并将连接它们的超链接作为有向边来将整个网络表示为图形。对此我们还需要做一些简化的假设：

- 仅考虑静态网页
- 忽略网络上的隐藏网页（即无法访问的或者防火墙后的页面）
- 所有链接都是可导航的。不考虑交易链接（例如：喜欢，购买，关注等）

以这种方式概念化网络类似于某些流行的搜索引擎的做法。例如，Google使用爬虫为网页编制索引，这些爬虫通过按广度优先顺序访问链接来浏览网络。可以通过这种方式抽象化为图的其他示例包括：科学研究论文之间的引文图，百科全书中的参考文献。

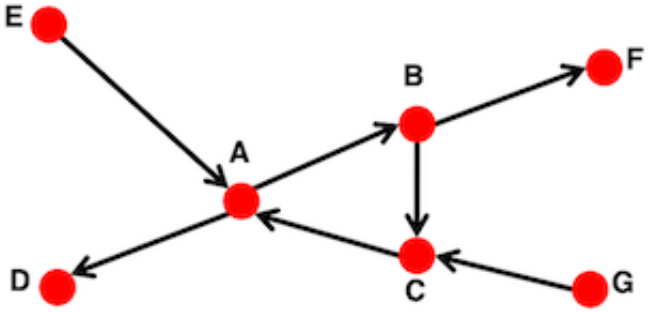
What does the graph of the web look like

在2000年，AltaVista的创始人进行了一项实验来探索Web的外观。他们问了一个问题：给定一个节点 v：什么节点是 v 可以到达的？什么节点是可以到达 v 的？

这使得节点可以分为两类：

$$\text{In}(v) = \{w | w \text{ can reach } v\}$$
$$\text{Out }(v) = \{w | v \text{ can reach } w\}$$

这些集合可以通过运行简单的BFS来确定。例如，在下图中：



$$\text{In}(A) = A, B, C, E, G$$
$$\text{Out}(A) = A, B, C, D, F$$

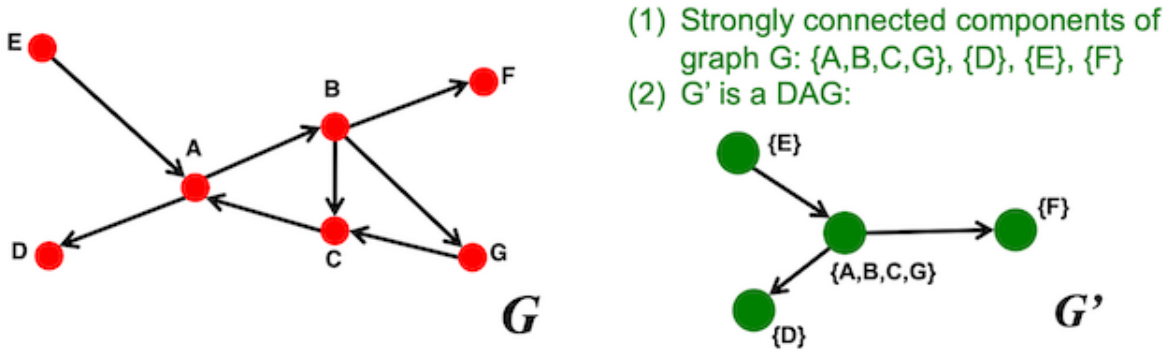
More on directed graphs

有向图有两种类型：

- **紧密连接的图**：图中任何节点都可以到达任何其他节点。
- **有向无环图 (DAG)**：图中没有环，如果图中节点 u 能到达 v，但是 v 不能到达 u 。

任何有向图都可以表示为这两种类型的组合。即：

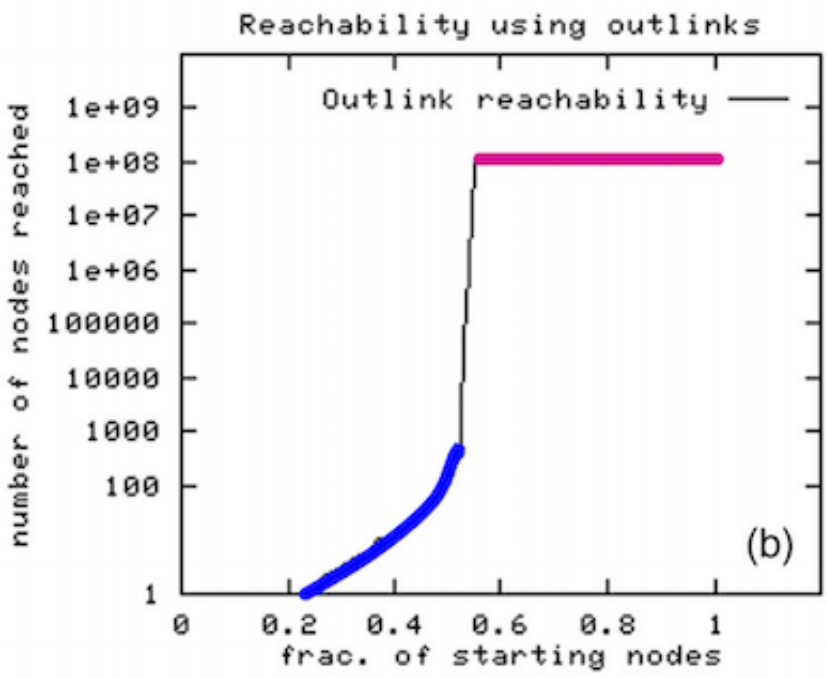
1. **识别有向图中的强连接组件 (strongly connected components, SCCs)**：SCC是图 G 中的一组紧密连接的节点集合 S ，并且在 G 中没有比 S 更大的集合。SCCs可以通过以下方式来识别：对所有的给定节点的入图节点运行BFS，并在来自同一节点的所有出图节点运行另一个BFS，然后计算这两个集合的交集。
2. **将SCC合并到超节点中，创建新图 G'**: 如果G中相应SCC之间存在边，则在SCC和节点之间创建边形成 G'，G' 现在是DAG。



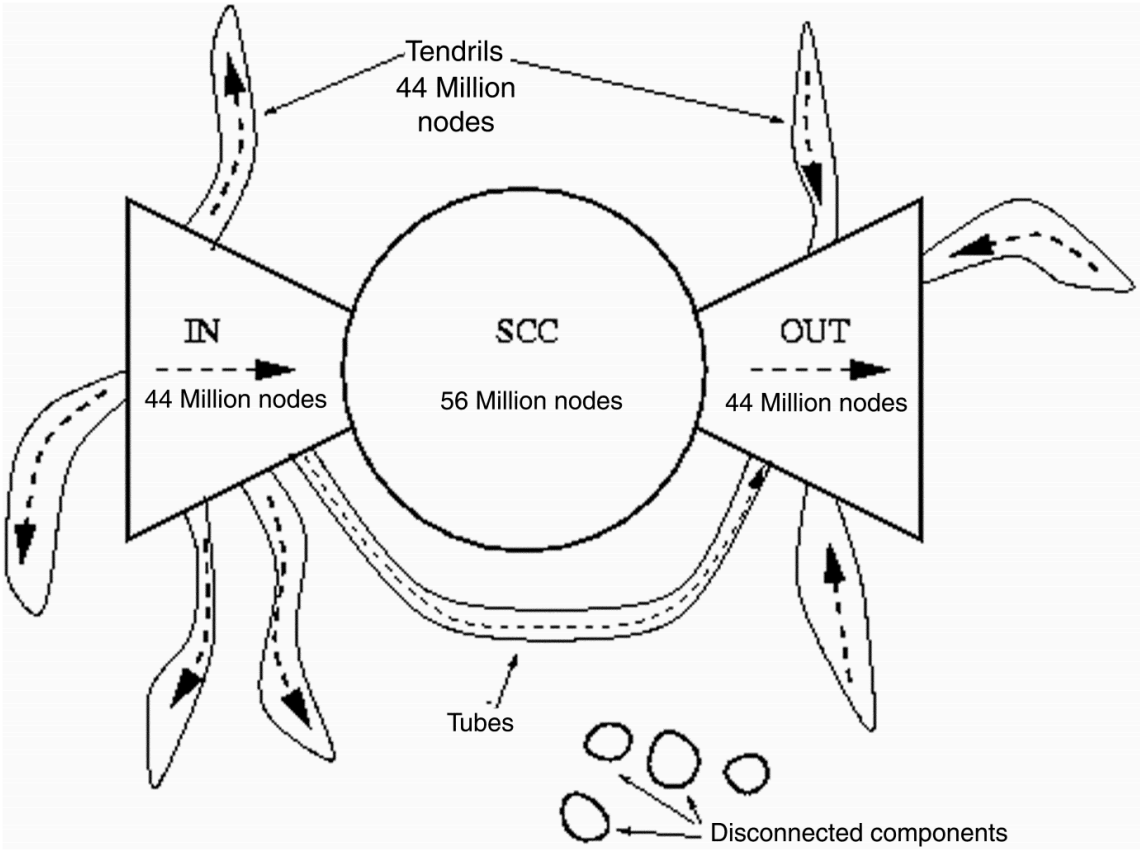
Bowtie structure of the web graph

Broder et al. (1999)对网络进行了一次大的快照，并试图了解网络图中的SCC如何组合在一起作为DAG。

他们的发现显示在下图中。在这里，起始节点按从该节点开始进行BFS访问的节点数量进行排序



- 左侧蓝色的节点表示在停止之前到达很少数量的节点
- 右侧洋红色的节点表示可以在停止之前到达大量的节点
- 使用此方法，我们可以确定如下所示的蝴蝶结形网络图的IN和OUT组件中的节点数。



- SCC对应于图中最大的强连接组件。
- IN组件对应于具有SCC的出站链接但没有来自SCC的入站链接的节点。OUT组件对应于具有来自SCC的入站链接但没有SCC的出站链接的节点。
- 同时具有入站链接和出站链接的节点都属于SCC。

- 卷须 (Tendrils) 对应于从IN伸出的边缘，或对应于进入OUT的边缘。Tubes是从IN到OUT的绕过SCC的连接。
- 断开连接的组件根本未连接到SCC

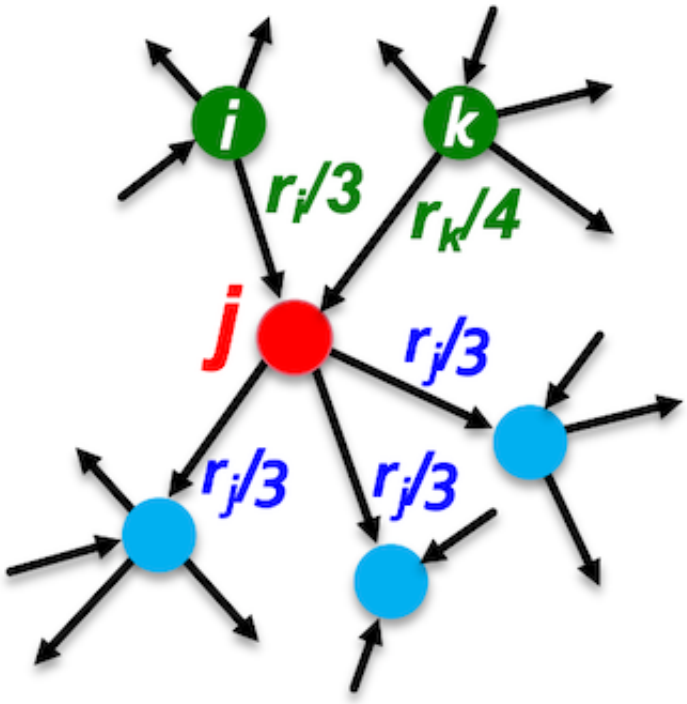
PageRank-Ranking nodes on the graph

并非网络上的所有页面都相同。当我们进行查询时，我们不想找到所有包含查询关键词的页面。因此，如果我们想学习如何对页面进行排序。我们如何根据网络图的链接结构确定页面的重要性?

这里的核心思想是链接是投票

进入页面的链接被视为该页面的投票，这有助于确定该页面的重要性。从重要页面进站的链接更为重要。这变成了递归关系：页面A的重要性取决于页面B，页面B的重要性取决于页面C，依此类推。

每个链接的投票与其源页面的重要性成正比。在下图中，节点 j 具有从节点 i 和节点 k 的进入链接，重要性分别为 $\frac{r_i}{3}$ 和 $\frac{r_k}{4}$ 这是因为节点 i 有三个出站链接,节点 k 有四个出站链接 。同样，节点 j 的重要性在其他三个出站节点之间平均分配。



总而言之，如果其他重要页面指向该页面，则该页面很重要。给定一组页面，我们可以定义节点 j 的页面重要性 r_j :

$$\mathbf{r}_j = \sum_{i \to j} \frac{\mathbf{r}_i}{d_i}$$

其中 i 是 j 的每个进站边节点，d_i是节点 i 的出度。

Matrix formulation

我们可以使用N个变量将这些关系公式化为N个PageRank方程。 我们可以使用高斯消元法求解，但是对于大型图(例如网络图)，这将花费很长时间。

此外，我们可以将图表示为列随机的邻接矩阵M，这意味着所有列的总和必须为1。因此对于节点 j 的所有条目，第 j 列的和必须为1。

$$\text{If } j \to i, \text{ then } \mathbf{M}_{ij} = \frac{1}{d_j}$$

r_i 是页面 i 的重要性：$\sum_i \mathbf{r}_i = 1$

这将满足每个节点的重要性必须总和为1的要求。我们现在将PageRank向量重写为以下矩阵方程式:

$$\mathbf{r} = \mathbf{M} \cdot \mathbf{r}$$

在这种情况下，PageRank向量将是随机Web矩阵 $\mathbf{M}$ 的特征向量，对应的特征值为1。

Random walk formulation

PageRank与随机游走非常像。想象一下网络图和浏览该图的随机浏览者。对于任何时刻 t，浏览者可以在任何页面 i。然后，浏览者将在时刻 t+1 随机地统一选择任何出站链接，并且这个过程无限地继续。设 $p(t)$ 为向量，其中第 i 个分量是冲浪者在时间 t 处在第 i 页的概率。

$$p(t+1) = \mathbf{M} \cdot p(t)$$

当 t 接近无穷大时，随机游动将达到稳态:

$$p(t+1) = \mathbf{M} \cdot p(t) = p(t)$$

当我们求解方程 $\mathbf{r} = \mathbf{M} \cdot \mathbf{r}; \mathbf{r}$ 时，实际上只是当t接近无穷大时，该冲浪者在时间t处的概率分布。 它在图形上模拟了这种随机游走过程的平稳分布。

Power law iteration(幂迭代)

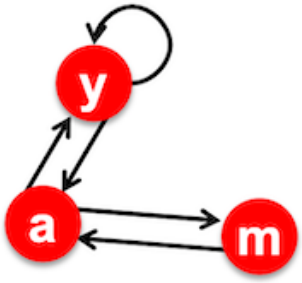
从任何向量 u 开始，极限 $\mathbf{M}(\mathbf{M}(\dots\mathbf{M}(\mathbf{Mu})))$ 是浏览者的长期分布。换句话说，当我们取向量 u 并乘以 $\mathbf{M}$ 足够长的时间时，$\mathbf{r}$ 是极限。 这意味着我们可以使用幂次迭代有效地求解 $\mathbf{r}$ 。

极限分布=M 的主特征向量 = PageRank

$$
\begin{aligned}
&\text{Initialize: } \mathbf{r}^{(0)} = \left[\frac{1}{N}, \ldots, \frac{1}{N}\right]^T \\
&\text{Iterate: } \mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)} \\
&\text{Stop when: } \left|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}\right| < \epsilon
\end{aligned}
$$

我们不断迭代，直到 *epsilon* 收敛为止。实际上趋向于在50-100次迭代中收敛。

Example



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$$
\begin{aligned}
r_y &= r_y/2 + r_a/2 \\
r_a &= r_y/2 + r_m \\
r_m &= r_a/2
\end{aligned}
$$

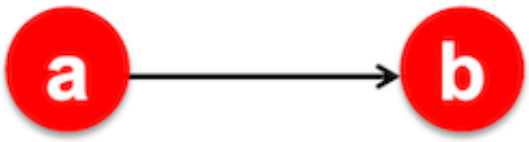
$$
\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 \\ 1/3 & 3/6 & 1/3 & 11/24 & \ldots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}
$$

Iteration 0, 1, 2, …

r 向量是页面排名或页面重要性。所以页面 y 的重要性是 $\frac{6}{15}$，页面 a 的重要性是 $\frac{6}{15}$，所以页面 m 的重要性是 $\frac{3}{15}$

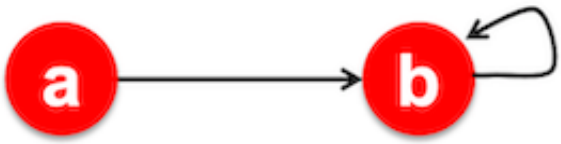
PageRank: Problems

1. Dead ends(死胡同) ：这些页面具有入站链接，但没有出站链接。作为一个随机的浏览者，这就像来到悬崖上，无处可去。这将意味着邻接矩阵不再是列随机的，并且会泄漏重要性。



2. Spider traps(爬虫陷阱) : 这些页面只有自我边作为外出边，导致浏览者被困住。

最终，Spider traps 将吸收所有重要性。给定一个带有自环的图 b，对于接下来的迭代随机冲浪者最终将跳转到 b 并陷入困境 。幂次迭代将收敛于具有所有重要性的 b，而使 a 失去重要性。



我们该如何解决？使用随机传送或随机跳跃

每当随机步行者迈出一步时，冲浪者就会有两个选择。它可以掷硬币，并且概率 β 继续跟随链接，或者概率以 $(1-\beta)$ 传送到另一个网页。可以跳到任何具有相等概率的节点。通常 β 设置在0.8到0.9左右。

如果碰到 Spider trap：以有限的步数传送出去。如果出现 dead end：以概率1传送出去。这将使矩阵列随机。

综上所述，PageRank方程 (由Brin-Page，98提出) 可以写为：

$$r_j = \sum_{i \to j} \frac{r_i}{d_i} + (1-\boldsymbol{\beta})\frac{1}{N}$$

现在，我们可以定义Google Matrix A并和以前一样应用幂次迭代来求解 r

$$A = \beta \mathbf{M} + (1-\beta)[\frac{1}{N}]_{N \times N}$$
$$\mathbf{r} = \mathbf{A} \cdot \mathbf{r}$$

注意：这个公式假定 M 没有 dead ends。我们可以对矩阵 M 进行预处理，以去除所有dead ends，也可以从 dead ends 中以概率1.0显式地进入随机传送通道。

Computing PageRank: Sparse matrix formulation

计算page rank的关键步骤是矩阵向量乘法

$$\mathbf{r}_{new} = \mathbf{A} \cdot \mathbf{r}_{old}$$

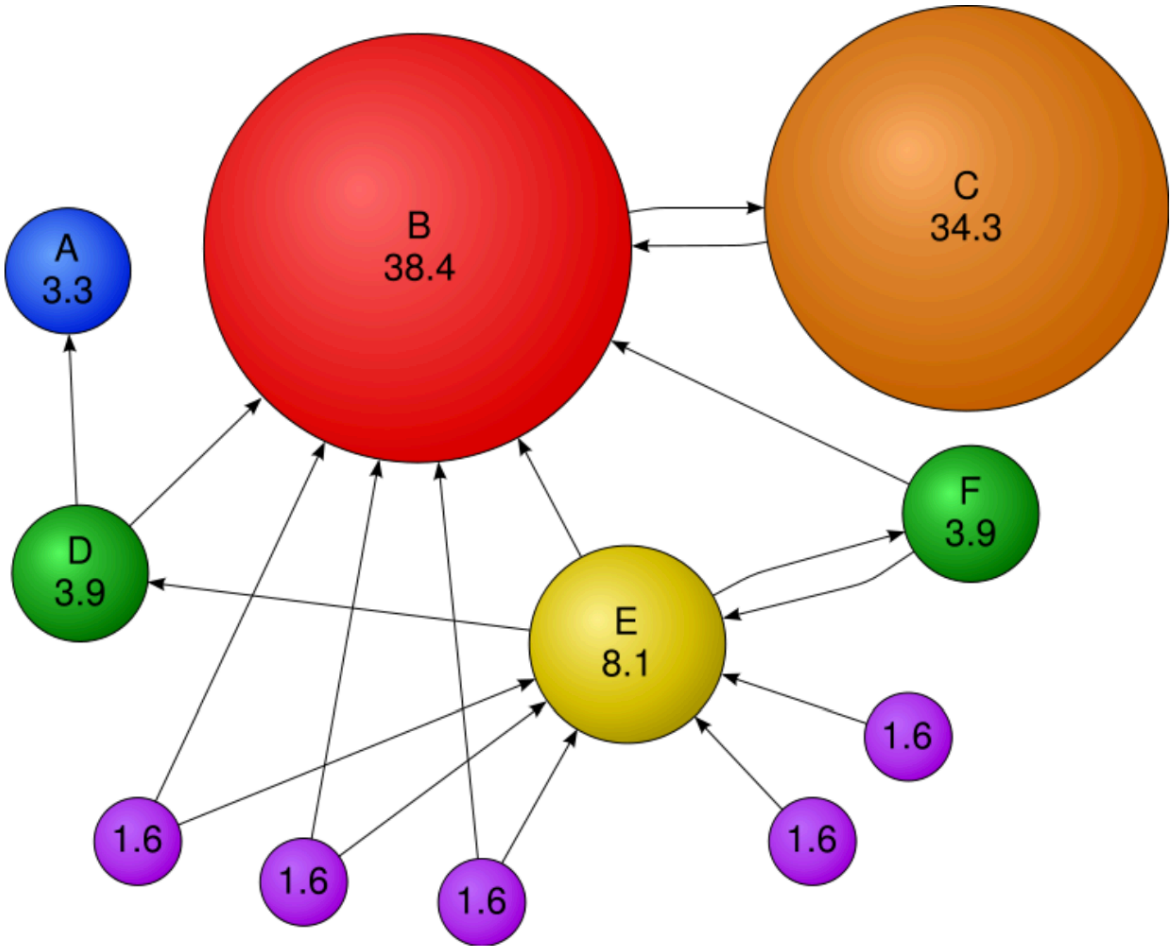
我们希望能够对此进行尽可能多的迭代。如果 $\mathbf{A}, \mathbf{r}_{old}, \mathbf{r}_{new}$ 足够小，可以装载到内存中，这倒是没有问题。但是如果有 N=10 亿个页面，每个页面4字节，那么仅仅是存储 $\mathbf{r}_{old}, \mathbf{r}_{new}$ 就将会需要8GB内存。矩阵 $\mathbf{A}$ 有 $\mathbf{N}^2 = 10^{18}$ 个条目，这将需要近1000万GB的内存。

我们可以重新设置计算公式，如下所示：

$$\mathbf{r} = \beta \mathbf{M} \cdot \mathbf{r} + \frac{[1-\beta]}{\mathbf{N}}$$

因为 M 是一个稀疏矩阵，这将会更容易计算，将它与标量相乘仍然是稀疏的，然后将其与向量相乘就不会占用大量计算资源。在此之后，我们只需添加一个常数，即随机游走直接跳到的 $\mathbf{r}_{new}$ 概率。那么总共需要的内存总量将会从 $O(N^2)$ 降到 $O(N)$ 。在每次迭代时，某些页面排名可能会泄漏出去，通过对 M 重新进行规范化，我们可以重新插入泄漏的页面排名。

这是将PageRank应用于图的工作示例：



在上图中，每个节点内是其页面排名得分或重要性得分。分数总和为100，节点的大小与其分数成正比。节点B重要性最高，因为很多节点都指向它。 这些节点虽然没有入站链接，但仍然很重要，因为随机游走跳跃可以跳转到它们。 节点C仅具有一个入站链接，但是由于它来自B，因此它也变得非常重要。 但是，C的重要性仍然不如B，因为B还有很多其他入站链接。

PageRank Algorithm

输入：

- 有向图 G (可以具有 Spider traps 和 dead ends)

- 参数 β

输出：

- PageRank 向量 $\mathbf{r}^{new}$

设置：$r_j^{old} = \frac{1}{N}$：

重复以下过程直到收敛：

$$\sum_j \left| r_j^{new} - r_j^{old} \right| < \epsilon$$

$$\forall j : r_j'^{new} = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}$$

$$r_j'^{new} = 0 \text{ if in-degree of j is 0}$$

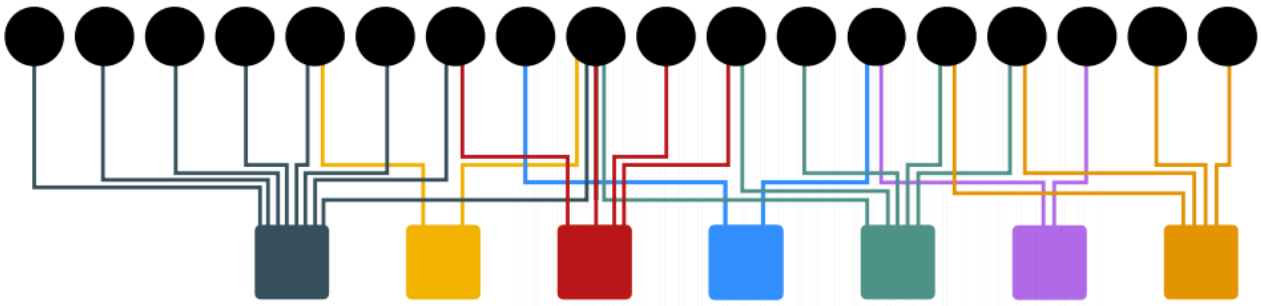
现在，重新插入泄漏的PageRank：

$$\forall j : r_j^{new} = r_j'^{new} + \frac{1-s}{N} \text{ where } S = \sum_j r_j'^{new}$$

$$r^{old} = r^{new}$$

Personalized PageRank and random walk with restarts

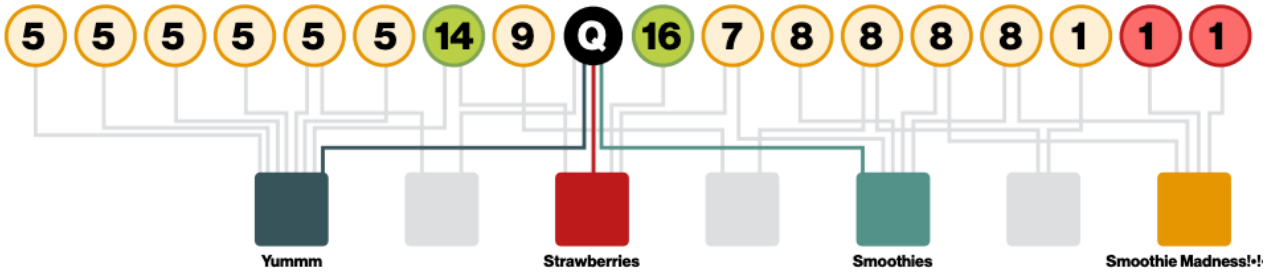
想象一下，我们有一个二部图，它由一侧的用户(下图中的圆圈)和另一侧的项目(正方形)组成。 我们想问两个项目之间的相关性或两个用户之间的相关性。假设用户过去已经购买过，那么根据他们与其他用户的共同点，我们可以向他们推荐什么？



我们可以使用不同的指标来量化，例如最短路径或公共邻居的数量，但是这些功能不是很通用。相反，我们可以使用PageRank的修改版本，该版本不会按重要性对所有页面进行排名，而是根据与给定集合的接近程度对其进行排名。 该集称为传送集S，此方法称为个性化PageRank。

一种实现方法是采用传送集并使用幂次迭代来计算Pagerank向量。但是，由于这种情况下在 S 中只有一个节点，因此执行简单的随机游走会更快。因此，随机游走者从节点Q开始，然后每当它进行传送时，它就会返回到Q。这将通过识别访问次数而获得与Q最相似的所有节点。 因此，我们获得了一个非常简单的推荐系统，该系统在

实践中效果很好，我们可以将其称为带重启的随机游走。



重新启动的随机游走能够解决

- 多个连接
- 多路径
- 直接和间接连接
- 节点度

PageRank Summary：

- Normal pagerank：传送矢量是统一的
- Personalized PageRank：传送到主题特定的页面集。浏览者登陆节点可能有不同概率
- Random walk with restarts：特定主题的pagerank，其中传送始终是从同一节点出发的。在这种情况下不需要幂次迭代，我们可以使用随机游走，它非常快速且容易