计算机应用编程实验

网络VIP节点分析

熊永平@计算机学院

ypxiong@bupt.edu.cn

教三楼132

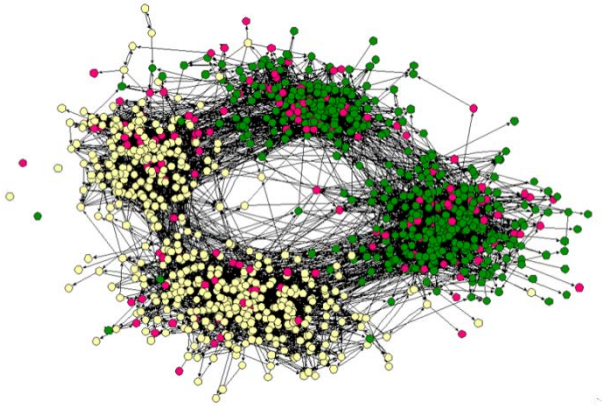
2020.11

实验任务

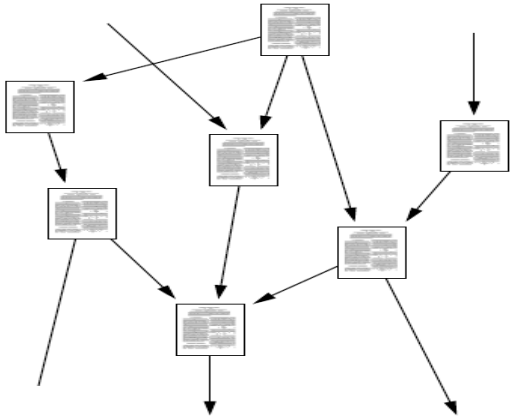
- **目标**
  - 设计一个图节点分析程序
    - 十几万个网页构建页面链接网络，计算前20个VIP页面
- **编程技能**
  - C语言编程
  - 稀疏矩阵与图结构
  - 随机过程
  - 特征值与特征向量
  - 图算法

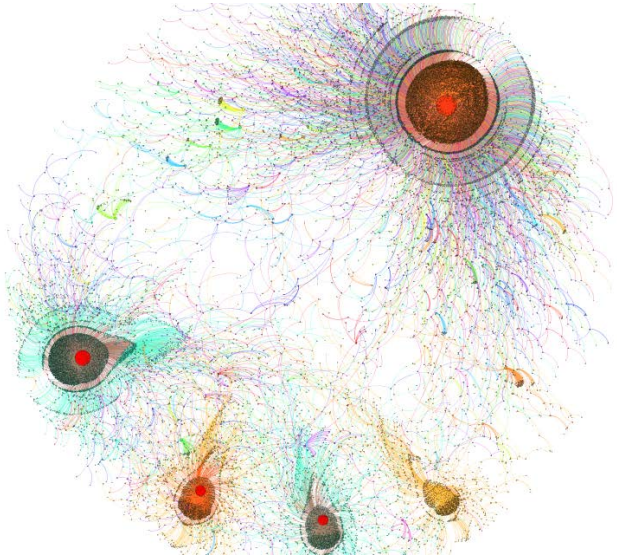
实验背景：复杂网络

朋友关系网

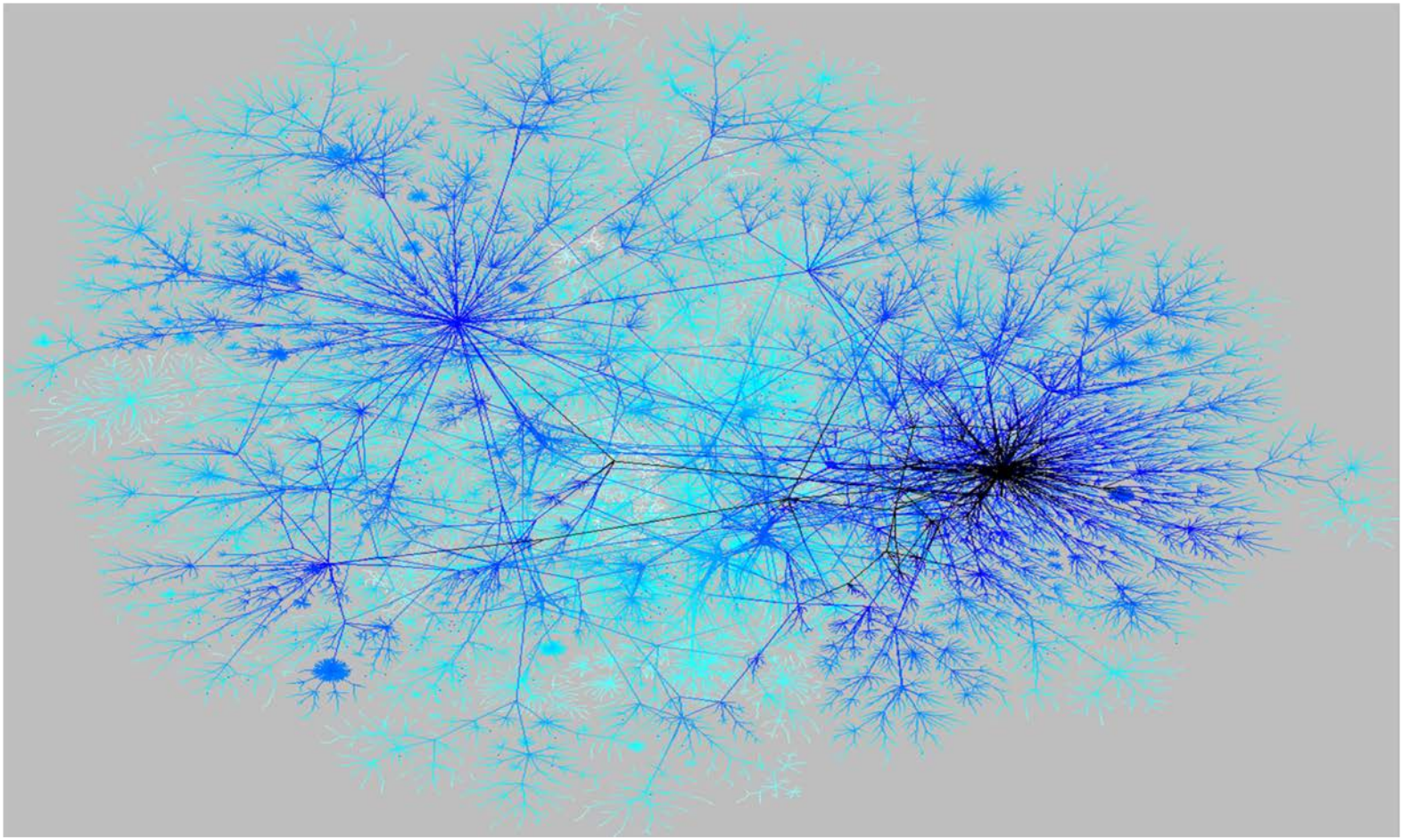


科学引文网

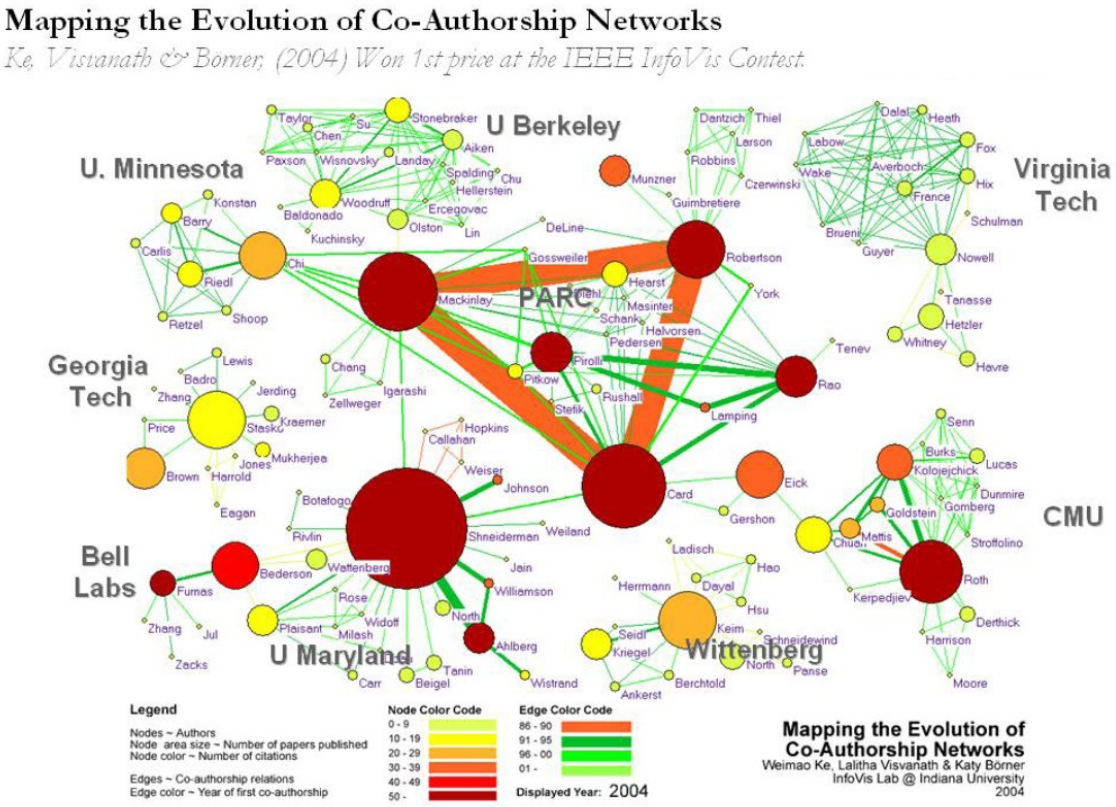




www网络



网络节点重要性度量



中心性测量

- Degree Centrality
- Eigenvector Centrality
- Katz Centrality
- Closeness Centrality
- Betweenness Centrality
  Transitivity
- PageRank

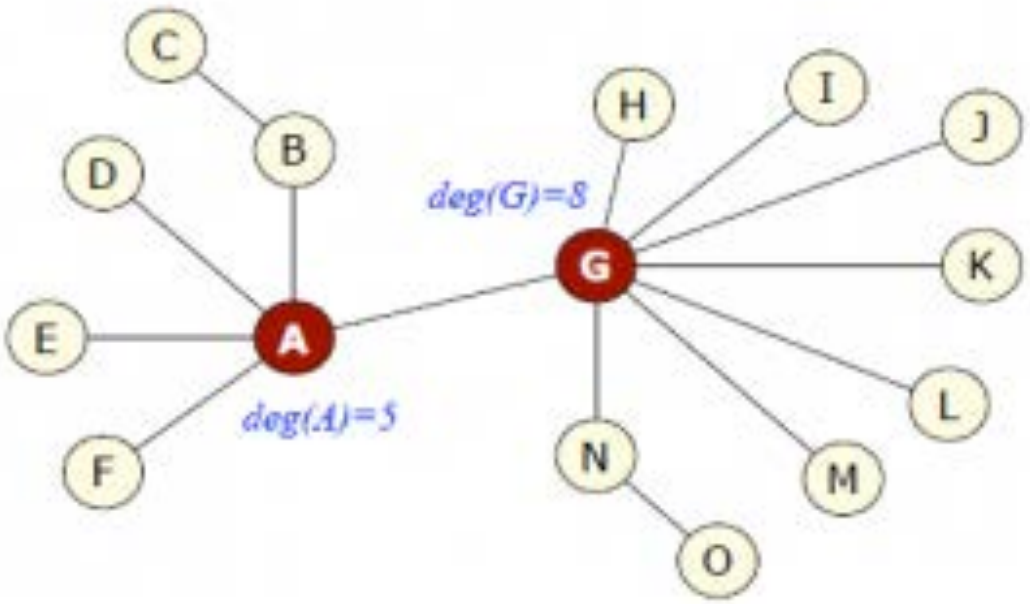
发现名人—节点度(degree centrality)

节点度是指和该节点相关联的边的条数。

特别地，对于有向图，

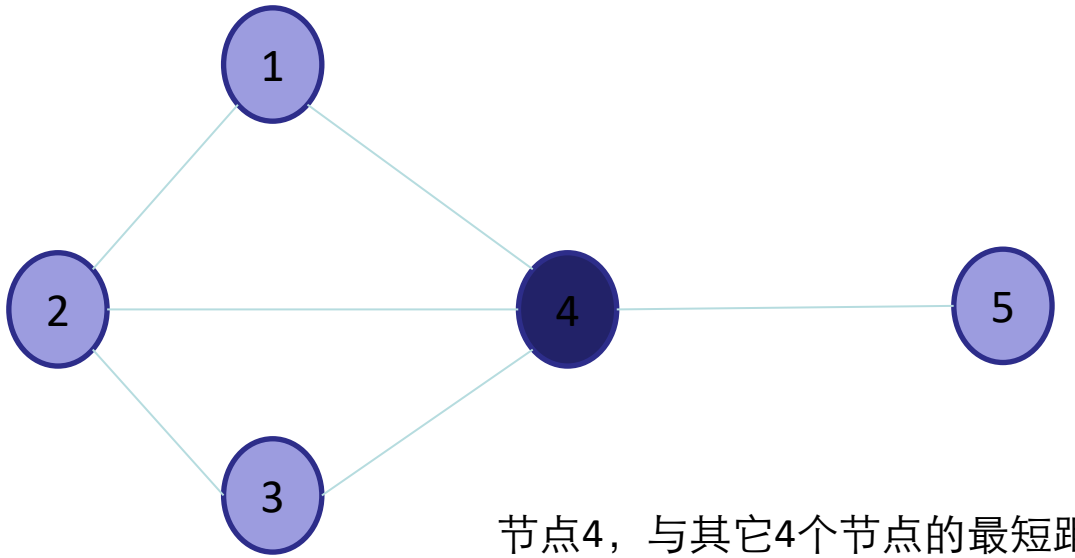
节点的入度 是指进入该节点的边的条数；

节点的出度是指从该节点出发的边的条数。



八卦传播者—接近中心性(closeness centrality)

如果一个点与网络中所有其它点的距离都很短，则该点是整体中心点。
在图中，这样的点与许多其它点都“接近”

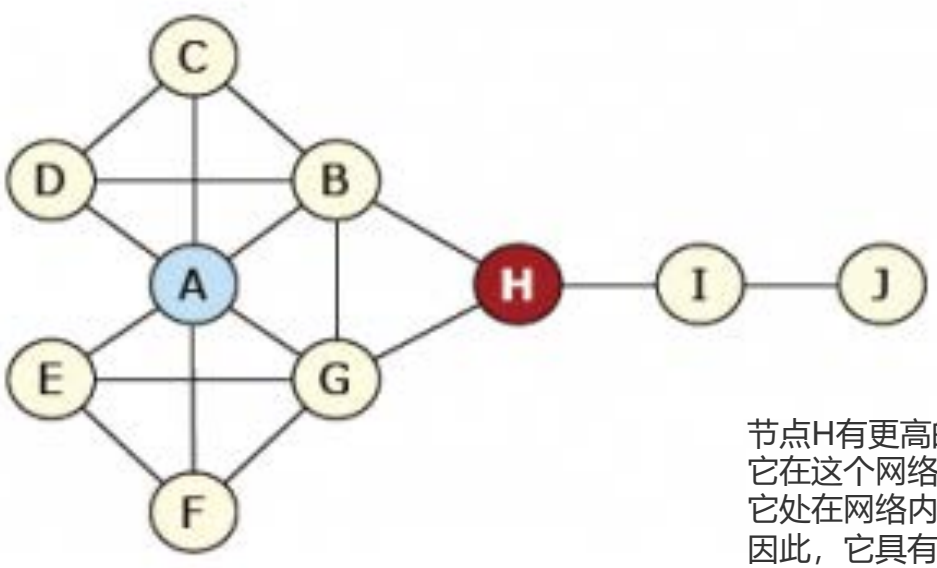


节点4，与其它4个节点的最短距离为1，和为4，节点4的接近中心度为1/4

节点2，与3个节点的最短距离为1，与节点5的最短距离为2，和5，节点2的　接近中心度为1/5

社群桥梁— 中介中心性(betweenness centrality)

中介中心性指出现在许多其他节点间最短路径上的节点有较高的中介中心性分数。



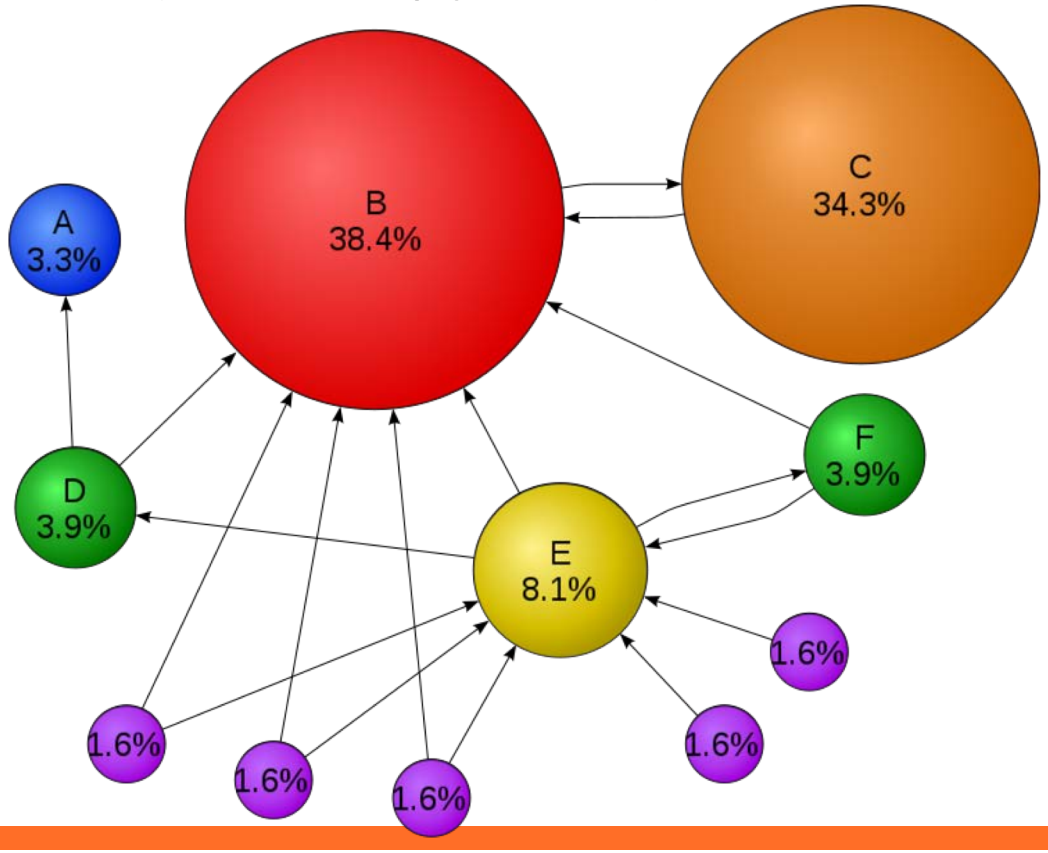
节点H有更高的中介中心性，
它在这个网络中扮演经纪人的角色，
它处在网络内许多节点交往的路径上，
因此，它具有控制其他人交往的能力。

Betweeness Centrality

传播影响力 —— PageRank

一个节点的“得票数”由所有链向它的节点的重要性来决定，到一个节点的边相当于对该节点投一票。一个节点的PageRank是由所有链向它的节点的重要性经过递归算法得到的。一个有较多链入的节点会有较高的等级，相反如果一个节点没有任何链入边，那么它没有等级。

在网络整体结构的意义上寻找整个网络中最核心的成员



幕后高手— 特征向量中心性(eigenvector centrality)

特征向量中心性，与page rank值类似，一个节点与其临近节点的中心性得分的总和成正比。
与重要的节点连接的节点更重要。
有少量有影响的联系人的节点其中心性可能超过拥有大量平庸的联系人的节点。

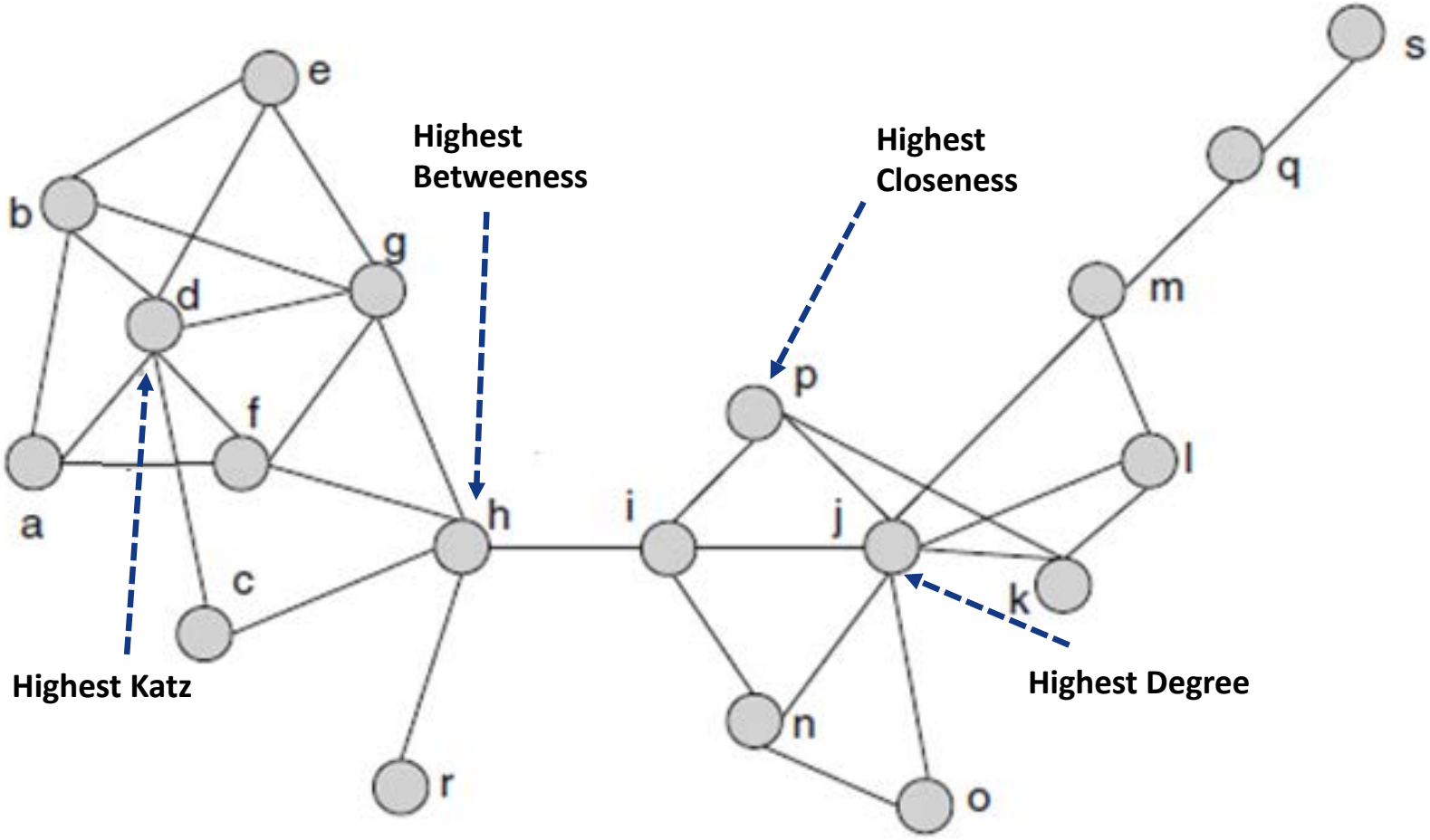
特征向量中心性的计算：

1、计算图的成对临近矩阵的特征分解

2、选择有最大特征值的特征向量

3、第i个节点的中心性等于特征向量中的第i元素

不同中心性度量对比

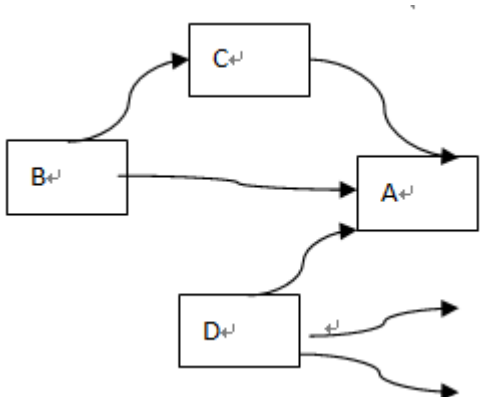

Highest
Betweeness
Highest
Closeness
Highest Katz
Highest Degree
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s

VIP度量1：PageRank算法

PageRank身世

- **提出**
  - Google的创始人之一Larry Page于1998年提出了PageRank，并应用在Google搜索引擎的检索结果排序上，该技术也是Google早期的核心技术之一
  - 有向图上的特征向量中心性
- **核心思想**
  - 一个节点的“得票数”由所有链向它的节点的重要性来决定，到一个节点的边相当于对该节点投一票。
  - 一个节点的PageRank是由所有链向它的节点的重要性经过递归算法得到的。
  - 一个有较多链入的节点会有较高的等级，相反如果一个节点没有任何链入边，那么它没有等级。

PageRank示例



- 假设一个由只有4个页面组成的集合：A，B，C和D。如果所有页面都链向A，那么A的PR（PageRank）值将是B，C及D的和。

$$PR(A) = PR(B) + PR(C) + PR(D)$$

- 继续假设B也有链接到C，并且D也有链接到包括A的3个页面。一个页面不能投票2次。所以B给每个页面半票。以同样的逻辑，D投出的票只有三分之一算到了A的PageRank上。

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1} + \frac{PR(D)}{3}$$

- 换句话说，根据链出总数平分一个页面的PR值。

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}$$

PR形式化表示

- 定义邻接矩阵为G，若网页j到网页i有超链接，则$g_{ij}=1$；反之，$g_{ij}=0$
- 设共有 m 个网页，分别编号为 1、2、3、...、m，它们的级别（重要性）分别记为 r_1、r_2、r_3、...、r_m，G表示由这些网页组成的有向图的邻接矩阵。根据有向图理论：

$$r(u)=\sum_{v\in B_u}\frac{r(v)}{n_v} \quad \Rightarrow \quad r_i=\sum_{j=1}^{m}\frac{g_{ij}}{n_j}r_j$$

$\frac{g_{ij}}{n_j}$ 中第 j 列的列和

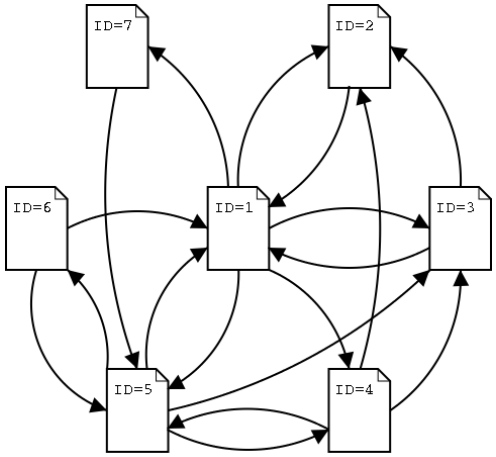
矩阵形式

$$r=G_m\cdot r$$

其中 $\begin{cases} r=(r_1,r_2,\ldots,r_m)^T \\ G_m=\{g_{ij}/n_j\} \end{cases}$

➢ 可知 r 是 G_m 的对应于特征值为 1 的特征向量

7个网页图的PR



$$G = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Gm = \begin{pmatrix} 0 & 1 & 1/2 & 0 & 1/4 & 1/2 & 0 \\ 1/5 & 0 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

求矩阵Gm特征值1对应的特征向量

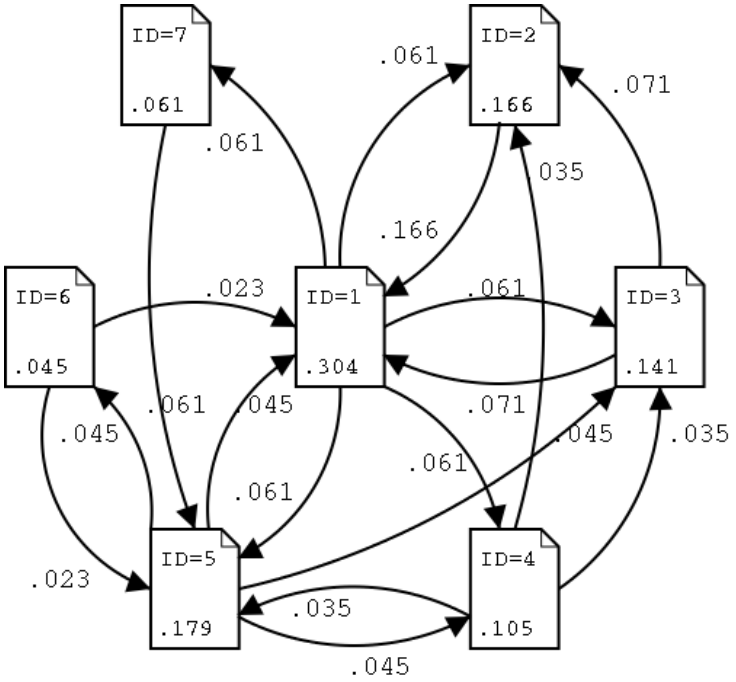
0.699456533837389
0.382860418521518
0.323958815672054
0.242969111754040
0.412311219946251
0.103077804986563
0.139891306767478

归一化

0.303514376996805
0.166134185303514
0.140575079872204
0.105431309904153
0.178913738019169
0.044728434504792 3
0.060702875399361 0

PageRank结果

名次	PageRank	文件ID	发出链接ID	被链接ID
1	0.304	1	2, 3, 4, 5, 7	2, 3, 5, 6
2	0.179	5	1, 3, 4, 6	1, 4, 6, 7
3	0.166	2	1	1, 3, 4
4	0.141	3	1, 2	1, 4, 5
5	0.105	4	2, 3, 5	1, 5
6	0.061	7	5	1
7	0.045	6	1, 5	5



- 分析
  - ID=1 的页面的PageRank 是0.304，占据全体的三分之一。
  - 特别需要说明的是，起到相当大效果的是从排在第3位的 ID=2 页面中得到了所有的PageRank (0.166) 数。ID=2页面有从3个地方过来的链入链接，而只有面向 ID=1页面的一个链接，因此(面向ID=1页面的)链接就得到ID=2的所有的PageRank数。
  - ID=1页面是链出链接和链入链接最多的页面，它是最受欢迎的页面。
  - 即使有同样链入链接的数目，链接源页面评价的高低也影响 PageRank 的高低。

PageRank是求解Gm的特征值为1的特征向量

Over?

1、矩阵 G_m 一定有特征值1吗？即上面的方程是否有解？

如果 $G = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$，则 $r_1 = r_2$，此时就无法进行求解

从代数角度

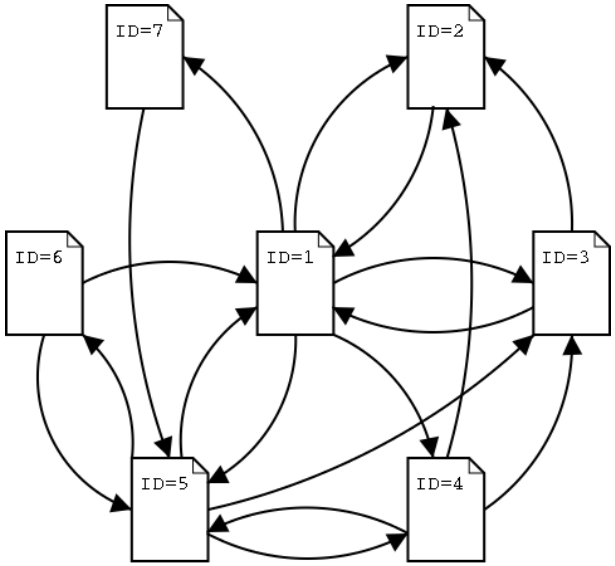
- **Perron-Frobenius定理**
  - 矩阵A不可约，即满足
    - 强连通
    - 非周期
  - 结论

$$\boldsymbol{x = A x}\ ,\ \boldsymbol{x}\ \text{满足：}\ \sum_{i=1}^{n} x_i = 1$$

    - 该方程组解存在且唯一
    - x是A的最大特征值1所对应的特征向量

如何构造A使之不可约?

网页浏览过程马尔科夫建模

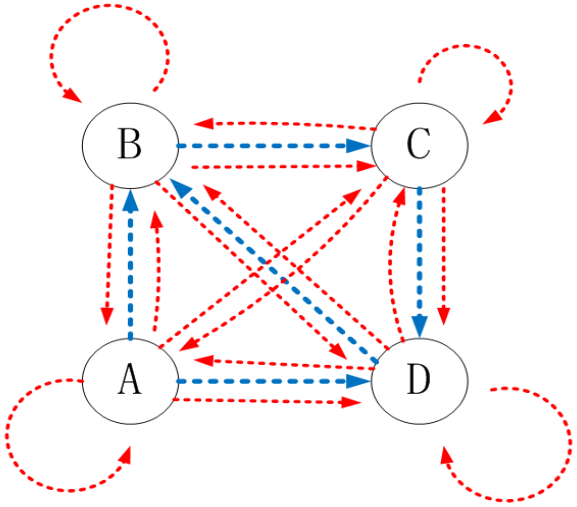


$$Gm = \begin{matrix} 0 & 1 & 1/2 & 0 & 1/4 & 1/2 & 0 \\ 1/5 & 0 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

- 马尔可夫浏览模型
  - 设想有一个永不休止浏览网页的人，每次随机选择一个指向链接继续访问，这个过程与过去浏览的页面无关，而仅依赖于当前页面。
  - 稳态情况下，每个网页v会有一个被访问的概率p(v)，等价于网页的重要程度rank，依赖于上一个时刻到达“链向”v的网页的概率，以及那些网页中超链的个数。

随机浏览修正

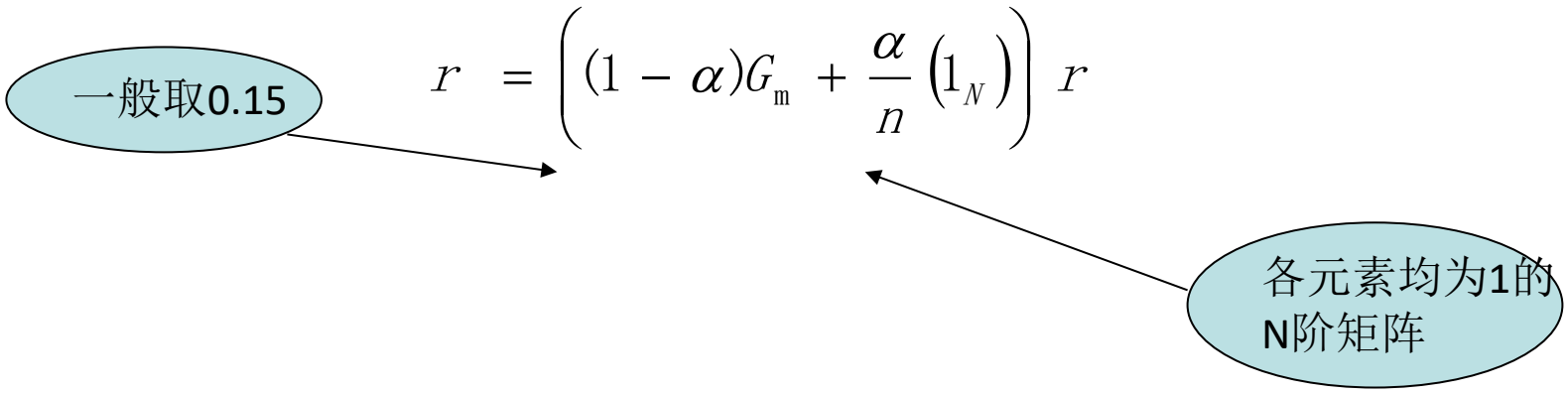
- 不可约
  - 当浏览者所浏览的网页矩阵存在不可达或周期连通分量时，该浏览者将无法继续浏览其它网页。
- 修正
  - 每次访问网页时，可以随机选择一个其它的网页重新开始浏览
    - ① 这种随机模型更加接近于用户的浏览行为
    - ② 上解决了rank leak和rank sink的问题



设定任意两个顶点之间都有直接通路，
在每个顶点处以概率d按原来蓝色方向转移，以概率1-d按红色方向转移。

修正模型

- 让浏览者每次以一定的概率（1-α）沿着链接走，以概率（α）重新随机选择一个新的起始节点
- α选在0.1和0.2之间，被称为阻尼系数
- 矩阵M=(1-α)Gm+ α/N(1_N)满足不可约特性，存在平稳分布r



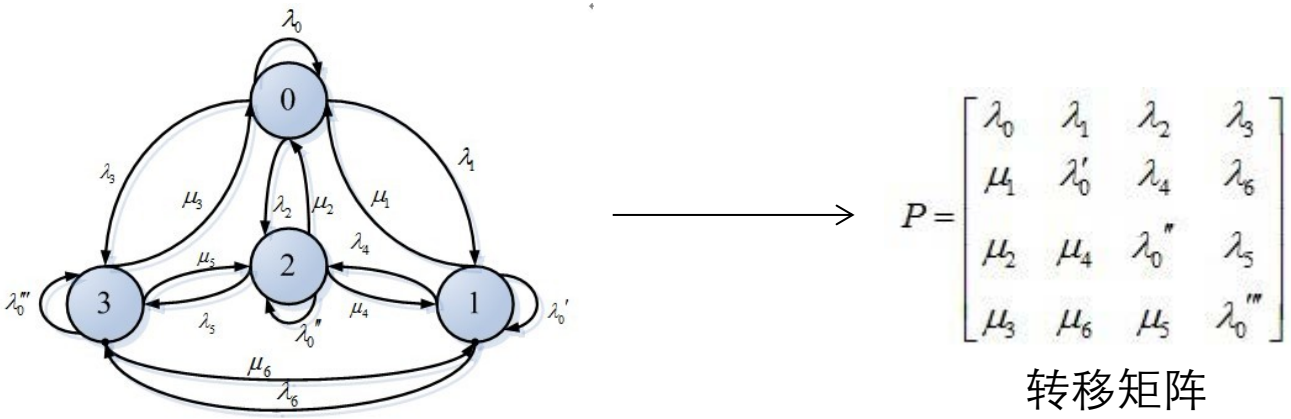
Over?

矩阵 G_m 特征向量怎么求？

- 特征向量计算规模是O(n^3)
- 特征向量的求解，就是求解方程 $A\alpha = \alpha$是 N 元一次方程组，一般不能得到分析解，所以只能解其数值。

马尔科夫过程的稳态

- **转移概率矩阵**
  - 将来只由现在决定，和过去无关
  - 图上每个点有一个值，会被不断更新。每个点通过一些 边连接到其它一些点上，对于每个点，这些边的值都是正的，和为1。在图上每次更新一个点的值，就是对和它相连接的点的值加权平均。

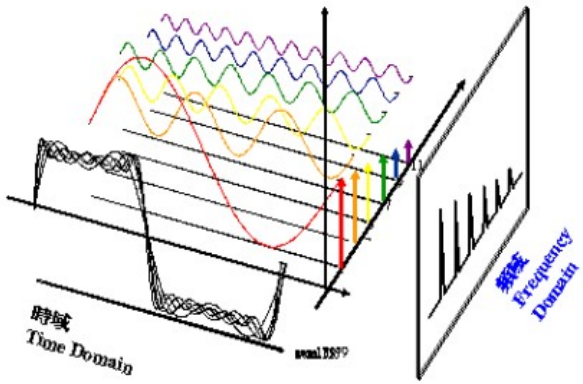


转移矩阵

  - 如果图是联通并且非周期 （数学上叫各态历经性， ergodicity)，那么这个过程最后会**收敛到一个唯一稳定的状态（平衡状态)**。

矩阵谱

- **矩阵谱**
  - 特征值和特征向量
  - 谱代表了一种分量结构，它使用“分而治之”策略来研究矩阵
  - 矩阵变换作用就是把一个向量变成另外一个向量：y = A x
  - 对于某些向量v，A v = cv，相当于向量拉长了c倍，这种和矩阵A密切配合的向量v1, v2, ... 叫做特征向量，这个倍数c1, c2,...叫特征值。
  - 当出现一个新的向量x 的时候，可以把x 分解为这些向量的组合，x = a1 v1 + a2 v2 +...，那么A对x的作用就可以分解：A x = A (a1 v1 + a2 v2 + ...) = a1 c1 v1 + a2 c2 v2...
  - 矩阵的谱是用于分解一个向量。



圖二 時域與頻域的差異

回到马尔科夫过程

- **计算方法**
  - 设A是马尔可夫过程的转移概率矩阵，数学严格证明，A最大的特征值就是1，对应于平衡状态v1，其它的特征状态v2, v3, ...,对应于特征值1 > c2 > c3 > ... > -1
  - 给定任意一个初始状态v（各节点的值），迭代计算v(t+1) = A v(t)。
  - 把v分解成 v = v1 +c2 v2 + c3 v3 + ...。
  - 更新进行了t 步之后，状态变成v(t) = v1 + c2^t v2 + c3^t v3 + ...，除了代表平衡状态的分量保持不变外，其它分量随着t增长而指数衰减
  - 最后得到系统的稳态v = Av，稳定状态就是A的特征值1对应的特征向量
  - **收敛速度取决于第二大特征值**c2，c2的大小越接近于1，收敛越慢，越接近于0，收敛越快。

Power Iteration计算

- 解决方法-Power Iteration幂迭代

当矩阵 A 的阶很大，无法直接计算其特征值和特征向量时，需要使用该方法

1) 输入矩阵 A 和迭代初始向量 v，以及精度 $\varepsilon > 0$(如0.0001，向量各元素对应差值绝对值），令 $k = 0$;

2) 计算：$v_{k+1} = Av_k$;

3) 如果 $|v_{k+1} - v_k| < \varepsilon > 0$ ，则计算 PageRank 值并停止。否则转第二步。

$$x = A^k v / sum(A^k v)$$

$A^k v$ 即 v_k

PageRank算法只有两步：构造矩阵A和迭代求解

幂迭代计算

- **幂迭代算法**
  - 确定合适的初始向量v，例如v= indegree(node)/|E|
  - 每次迭代是一次矩阵向量乘法复杂度>$O(n^2)$，但A是稀疏矩阵，所以整个迭代速度非常快
  - 3亿个页面的Web Graph
    -> 50 iterations to convergence (Brin and Page, 1998)
- **幂迭代算法的马尔可夫链模型**
  - page importance ⇔ steady-state Markov probabilities ⇔ eigenvector
  - Larry Page和Sergey Brin 的贡献，一方面加入随机游走解决了矩阵收敛问题，另一方面由于互联网网页数量巨大，生成的二维矩阵巨大，两人利用稀疏矩阵计算简化了计算量。

VIP度量2：介数betweeness centrality

介数定义

- 定义
  - 将每对顶点条件下的节点v介数计算定义为：$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$
  - 其中 $\sigma_{st}(v)$ 表示经过节点v的s到t的最短路径条数
  - σ_{st} 表示节点s到节点t的所有最短路径条数
  - 网络节点v介数定义为：

$$C_B(v) = \sum_{s=v\neq t\in V} \delta_{st}(v)$$

  - 即图中所有顶点对之间的经过v的最短路径条数占所有顶点对之间最短路径条数的比例
  - 需要求出所有顶点对之间的**所有最短路径**

算法回顾：Dijkstra单源最短路径算法

Dijkstra提出了一种按路径长度递增的次序求从源点到各终点的最短路径的算法。

假设从源点v_0到各终点$v_1,v_2,\ldots,v_k$之间存在最短路径，其路径长度分别为$l_1,l_2,\ldots,l_k$，并且$l_p(1<=p<=k)$为其中的最小值，即从源点v_0到终点v_p的最短路径是从源点到其他终点的最短路径中长度最短的一条路径，这条路径上必定只有一条弧，否则它就不可能是所有最短路径中的最短者。

第二条次短(从源点v_0到终点v_q)的最短路径只能产生在以下两种情况中：

- 从源点到该点存在弧$<v_0,v_q>$
- 从已求得的最短路径的顶点v_p到该点有弧$<v_p,v_q>$存在，且$<v_0,v_p>$和$<v_p,v_q>$上的权值之和小于$<v_0,v_q>$上的权值。

依此类推，一般情况下，假设已求得最短路径的顶点有$\{v_{p1},v_{p2},\ldots v_{pk}\}$，则下一条最短路径的产生只可能是已求出的这些最短路径的延伸，也就是从源点间接到达终点，或者是从源点直接通过一条弧到达终点。

算法流程

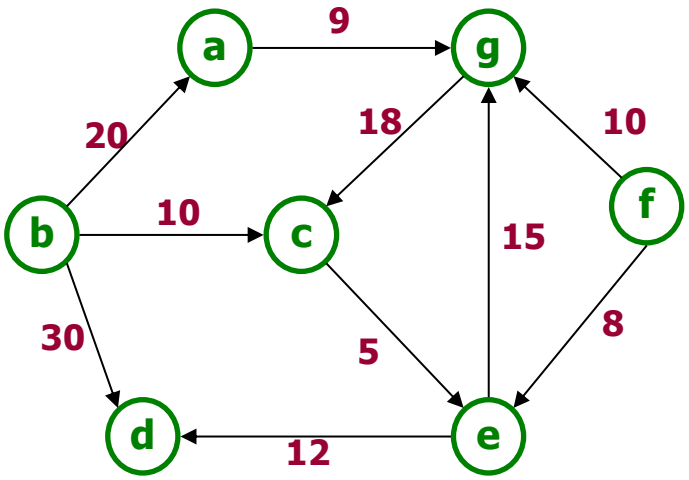
根据以上思路，Dijkstra算法描述如下：

1. 假设AS[n][n]为有向网的邻接矩阵，S为已找到最短路径的终点集合，其初始状态只含一个顶点，即源点。另设一维数组Dist[n]，其中每个分量表示当前所找到的从源点出发(经过集合S中的顶点)到各个终点的最短路径长度，则Dist的初值为： Dist[k]=AS[i,k]
2. 选择u，使得 Dist[u]=min{Dist[w]|w ∈ V-S}
3. 修改Dist数组中所有尚未找到最短路径的终点的对应分量值，

   如果 Dist[u]+AS[u,w]<Dist[w]

   则 Dist[w]=Dist[u]+AS[u,w]
4. 重复上述2和3的操作n-1次，即可求得从源点到所有终点的最短路径。

此外，为了记下长度为Dist[w]的最短路径，还需要附设路径矩阵Path[n][n]。



$$\begin{bmatrix} 0 & \infty & \infty & \infty & \infty & \infty & 9 \\ 20 & 0 & 10 & 30 & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & 5 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & 12 & 0 & \infty & 15 \\ \infty & \infty & \infty & \infty & 8 & 0 & 10 \\ \infty & \infty & 18 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Path[7][7]

b	a					
b	c					
b	c	e	d			
b	c	e				
b	a	g				

修改**Dist**和**Path**的值，
可见从**b**到**f**没有路径

Dist[7]

a	b	c	d	e	f	g
20	0	10	27	15	max	29

S: {b,c,e,a,d,g}

V-S: {f}

至此，从源点**b**出发到各终点的最短路径求解完毕，最短路径序列如矩阵**Path**所示

Floyd算法：所有顶点对最短路径

–求最短路径步骤

- 初始时设置一个n阶方阵，令其对角线元素为0，若存在弧<Vi,Vj>，则对应元素为权值；否则为∞
- 逐步试着在原直接路径中增加中间顶点，若加入中间点后路径变短，则修改之；否则，维持原值
- 所有顶点试探完毕，算法结束

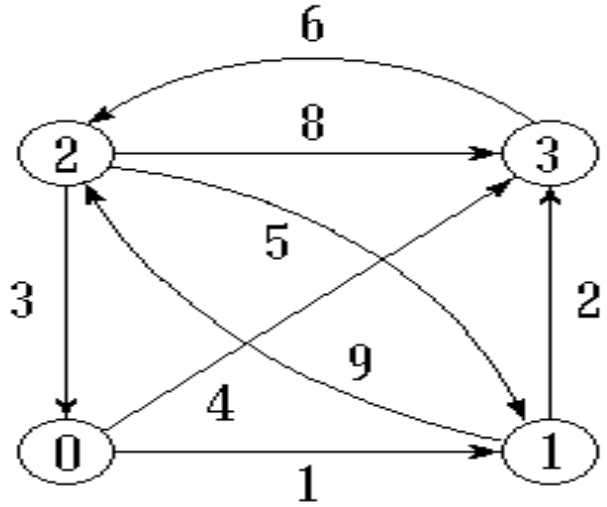
算法实现（O(n^3)复杂度）

- 图用邻接矩阵存储
- edge[ ][ ]存放最短路径长度
- path[i][j]是从Vi到Vj的最短路径上Vj前一顶点序号

```
void floyd ( ){
    for ( int i = 0; i < n; i++ )  //矩阵dist与path初始化
        for ( int j = 0; j < n; j++ ) {  //置A(-1)
            dist[i][j] = Edge[i][j];
            path[i][j] = -1; }    // 初始不经过任何顶点
    for ( int k = 0; k < n; k++ ) //产生dist(k)及path(k)
        for ( i = 0; i < n; i++ )
            for ( j = 0; j < n; j++ )
                if (dist[i][k] + dist[k][j] < dist[i][j] ) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                    path[i][j] = k; }//缩短路径, 绕过 k 到 j
} // floyd
```

	$A^{(-1)}$				$A^{(0)}$				$A^{(1)}$				$A^{(2)}$				$A^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	1	∞	4	0	1	∞	4	0	1	10	3	0	1	10	3	0	1	9	3
1	∞	0	9	2	∞	0	9	2	∞	0	9	2	12	0	9	2	11	0	8	2
2	3	5	0	8	3	4	0	7	3	4	0	6	3	4	0	6	3	4	0	6
3	∞	∞	6	0	∞	∞	6	0	∞	∞	6	0	9	10	6	0	9	10	6	0

	$Path^{(-1)}$				$Path^{(0)}$				$Path^{(1)}$				$Path^{(2)}$				$Path^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	3	1
1	0	0	1	1	0	0	1	1	0	0	1	1	2	0	1	1	2	0	3	1
2	2	2	0	2	2	0	0	0	2	0	0	1	2	0	0	1	2	0	0	1
3	0	0	3	0	0	0	3	0	0	0	3	0	2	0	3	0	2	0	3	0



$$\begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 1 & \infty & 4 \\ \infty & 0 & 9 & 2 \\ 3 & 5 & 0 & 8 \\ \infty & \infty & 6 & 0 \end{bmatrix} \end{array} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix}$$

	$A^{(-1)}$				$A^{(0)}$				$A^{(1)}$				$A^{(2)}$				$A^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	1	∞	4	0	1	∞	4	0	1	10	3	0	1	10	3	0	1	9	3
1	∞	0	9	2	∞	0	9	2	∞	0	9	2	12	0	9	2	11	0	8	2
2	3	5	0	8	3	4	0	7	3	4	0	6	3	4	0	6	3	4	0	6
3	∞	∞	6	0	∞	∞	6	0	∞	∞	6	0	9	10	6	0	9	10	6	0

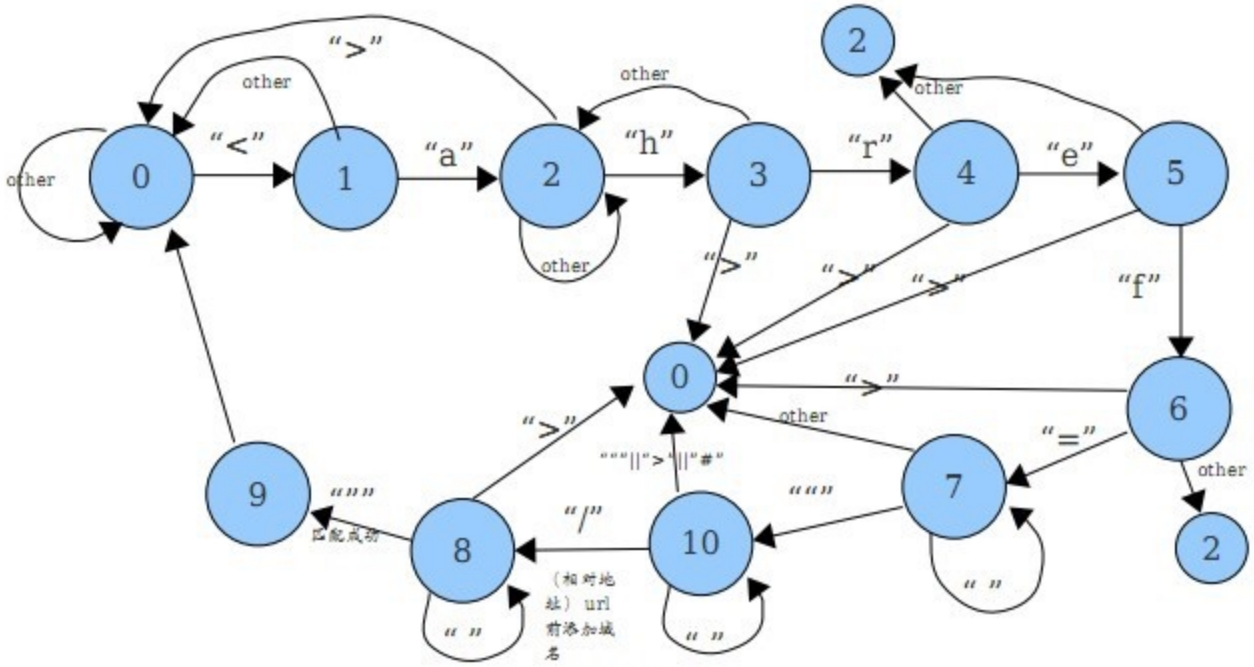
	$Path^{(-1)}$				$Path^{(0)}$				$Path^{(1)}$				$Path^{(2)}$				$Path^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	3	1
1	0	0	1	1	0	0	1	1	0	0	1	1	2	0	1	1	2	0	3	1
2	2	2	0	2	2	0	0	0	2	0	0	1	2	0	0	1	2	0	0	1
3	0	0	3	0	0	0	3	0	0	0	3	0	2	0	3	0	2	0	3	0

以 $Path^{(3)}$为例，对最短路径的读法加以说明。从$A^{(3)}$知，顶点1到0的最短路径长度为$a[1][0] = 11$,
其最短路径看 $path[1][0] = 2$，$path[1][2] = 3$，$path\ [1][3] = 1$, 表示顶点0 ← 顶点2 ← 顶点3 ← 顶点1;从顶点1到顶点0最短路径为<1, 3>,<3, 2>,<2, 0>。

实验工程实现问题

基于DFA的url提取

- **linux C regex库只提供贪婪匹配，不支持非贪婪匹配**
  - 贪婪匹配 href="m.html",xxxxxxxxsrc="n.jpg"
  - 非贪婪匹配 href="m.html",xxxxxxxxsrc="n.jpg"
- **功能**
  - 通过有限状态自动机进行字符串匹配。



图邻接存储和计算？

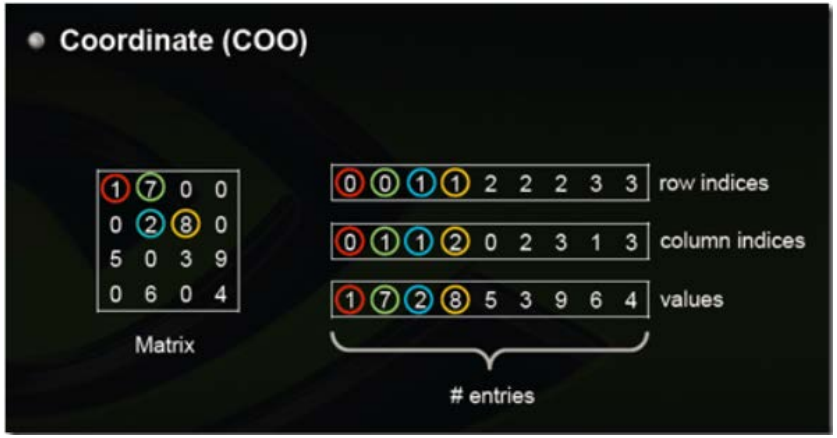
- 假设 N 是 10000，通常数值计算程序内部行列和矢量是用双精度记录的，N 次正方行列 A 的存储空间为 sizeof(double)* N * N =8 *104 * 104 = 800MB
- 本实验N是160000个，方阵存储空间超过200GB

稀疏矩阵

- **概念**
  - 设在矩阵A中，有s个非零元素。令 e=s/(m*n)，称e为矩阵的稀疏因子。
  - 通常认为e≦0.05时称之为稀疏矩阵。
- **存储方式**
  - 非零元素的分布一般没有规律，存储非零元素的同时，同时记下它所在的行和列的位置（i,j)。
  - 两种存储方式：
    - 三元组(i,j,a_{ij})唯一确定了矩阵A的一个非零元。因此，稀疏矩阵可由表示非零元的三元组及其行列数唯一确定。
    - 十字链表方法，矩阵的每一个非零元素用一个结点表示， 该结点除了（row，col，value）以外，还要有以下两个链域： right： 用于链接同一行中的下一个非零元素；down： 用于链接同一列中的下一个非零元素。

稀疏矩阵存储：COO

- 稀疏矩阵存储结构
  - Coordinate(COO)
  - 特点：格式简单，更加灵活，易于操作，常用于从文件中进行稀疏矩阵的读写，如matrix market即采用COO格式，但计算效率一般




```
typedef struct
{
    int row, col; /*该非零元素的行下标和列下标*/
    float e;  /*该非零元素的值*/
}Triple;

typedef struct
{
    Triple data [MAXSIZE+1] ; /* 非零元素的三元组表，data [0] 未用*/
    int mu,nu,tu; /*矩阵的行数、列数、非零元素个数*/
}TriSparMatrix;
```

COO三元组存储示例

data[p]	i	j	e
data[1]	1	2	12
data[2]	1	3	9
data[3]	3	1	-3
data[4]	3	6	14
data[5]	4	3	24
data[6]	5	2	18
data[7]	6	1	15
data[8]	6	4	-7

mu=6 nu=7 tu=8

原矩阵:

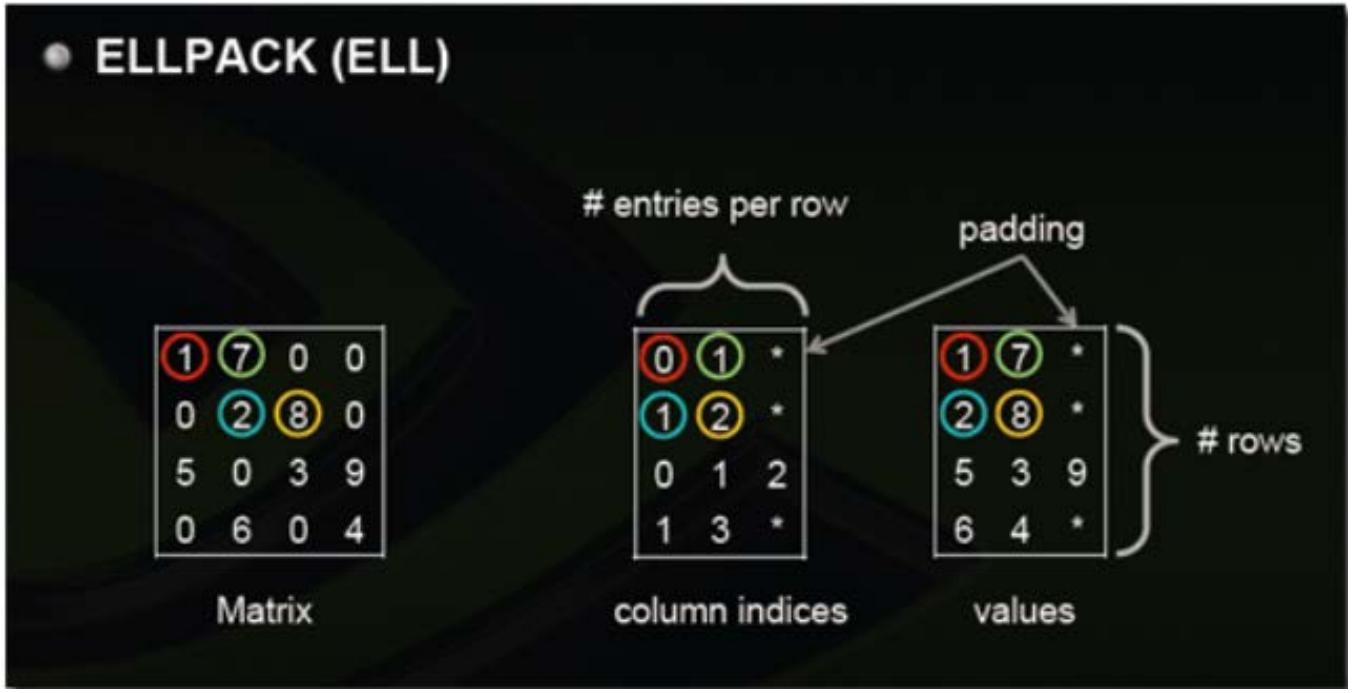
$$M=\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}$$

注意:

为了保存矩阵的行数、列数和非零元素个数，还需增设三个量：mu nu tu

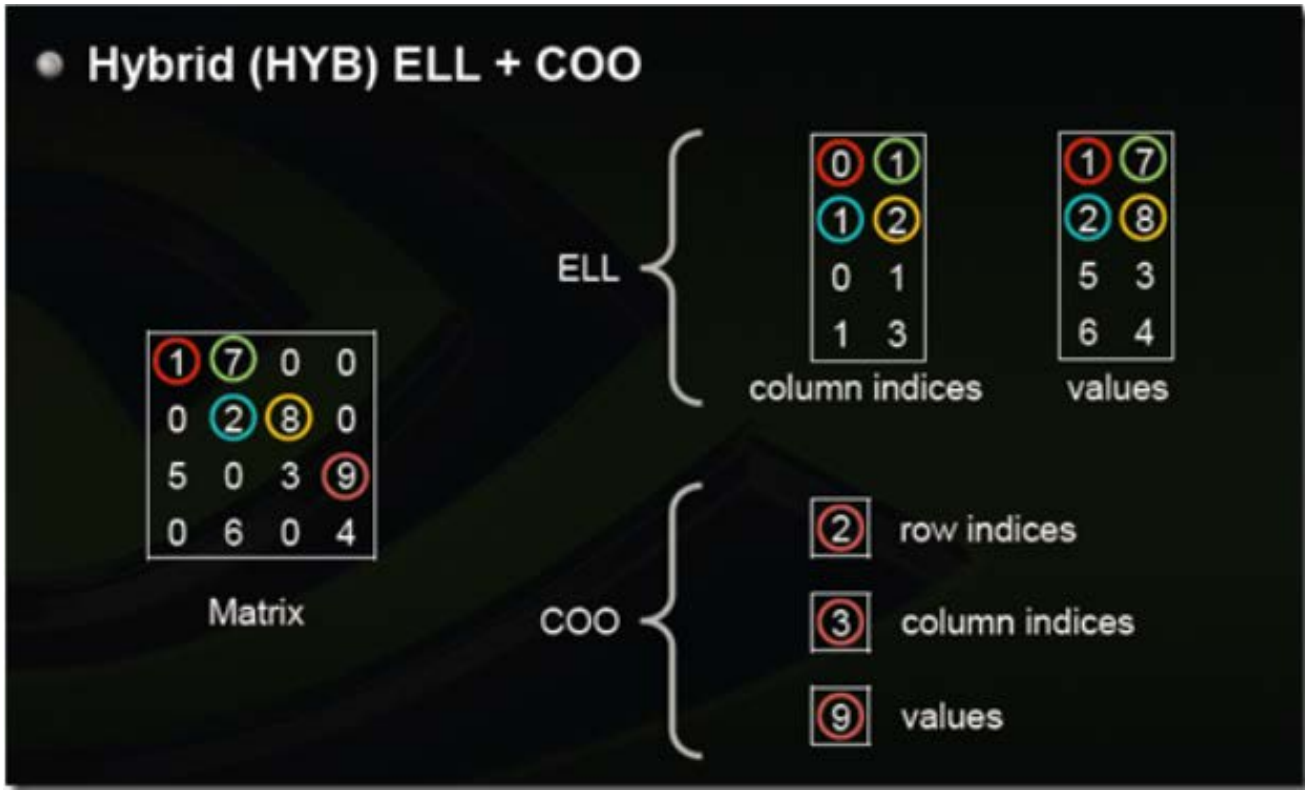
稀疏矩阵存储：ELL

- 稀疏矩阵存储结构
  - ELLPACK (ELL)
  - 特点：在进行稀疏矩阵-矢量乘积时效率最高，是应用迭代法解稀疏线性系统最快的格式；但如果某一行很多元素，那么后面两个矩阵就会很胖，其他行结尾*很多，浪费存储空间



稀疏矩阵存储：HYB

- 稀疏矩阵存储结构
  - Hybrid (HYB) ELL + COO
  - 特点：ELL的优点是快速，而COO优点是灵活，二者结合后的HYB格式是一种不错的稀疏矩阵表示格式



程序要求

- **实现3个工具，程序运行参数：**
  - buildgraph webdir graph.bin
    - 遍历webdir下所有网页，构建稀疏矩阵结构的图，并持久化存储到文件graph.bin
  - vip_pagerank graph.bin result.txt
    - 读取graph.bin并在result.txt输出pagerank最高的前20文件
    - 第1行pagerank迭代次数，从2-21行每行一个文件名 空格 pagerank值
  - vip_between graph.bin result.txt(O(n^3))
    - 读取graph.bin并在result.txt输出介数最高的前20文件
    - 每行一个文件名 空格 介数值
  - **vip_brandes_between graph.bin result.txt（加分项）**
    - ”A faster algorithm for betweenness centrality”，复杂度O(mn+nlogn)，在网络稀疏的时候，复杂度接近O(n^2).
    - 输出同上

报告要求

- **实验报告**
  - 主要数据结构和流程
  - 实验过程
  - 遇到的问题
  - 结果指标
    - 内存占用
    - 计算时间（相同计算机上运行）
  - 结论和总结


THE END