Ranking: HITS and PageRank

Chapter 4.5

And additional information

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2015 ; distribution outside this group of students is NOT allowed.*

1.1 Ranking based on similarity first: $s(q, d_j)$. The relevance of a query to a document (e.g. a web-page) is established by first computing the function $s(q, d_j)$ that determines the similarity between query q and document d_j, i.e. between two "documents". This similarity measure is based solely on information retrieval-based knowledge (e.g. the searchable contents of the document). Search engines (or web-searching methods in general) do not rely on this measure exclusively to rank a web-page (e.g. document d_j).

1.2 Ranking based on link structure of d_j. Search engines also use additional information such as the link structure of d_j to generate $R(q, d_j)$. The link structure relates to information such as:

- **(a)** what documents/pages are pointed by d_j and are these pages important (e.g. authoritative),
- **(b)** what documents/pages point to d_j and are these pages important (e.g. hubs of information such as a directory page).

1.3 Rank is similarithy based and link structured based. All such link information can be made to contribute a separate component to the rank of d_j or a web-page in general. Thus the **rank** of d_j relative to query q denoted by $R(q, d_j)$ is a function that will represent the rank of a document/web-page. One component of $R(q, d_j)$ is $s(q, d_j)$.

1.4 First ranking model: Vector Spread Activation Model. One of the first attempts to use linkage information was the **vector spread activation model** by Yuwono. The ranking score $R(q, d_j)$ is the sum of the similarity $s(q, d_j)$ of d_j plus a portion of the similarity of each document that points to d_j. This is under the assumption that if relevant documents point to d_j, then d_j should also be relevant. Let $l(i, j)$ be 1 or 0 based on whether document d_i points to d_j or not. Then

$$R(q, d_j) = s(q, d_j) + \beta \sum_i R(q, d_i) l(i, j)$$

where β is a user defined parameter (e.g. $\beta = 0.2$).

1.5 References.

1. M. Cutler, Y. Shih, and W. Meng: Using the Structures of HTML Documents to Improve Retrieval. Usenix Symposium on Internet Technologies and Systems (USITS'97) . pp.241-251, Monterey, California, December 1997.

2. M. Cutler, H. Deng, S. Manicaan, and W. Meng: A New Study on Using HTML Structures to Improve Retrieval. Eleventh IEEE Conference on Tools with Artificial Intelligence (ICTAI'99) , Chicago, November 1999.

HITS: Authority and Hub page rank (Kleinberg 1998)

2.1 Another model: Kleinberg's HITS model. Kleinberg (1998) introduced a model that assigns two ranks to each web-page/document. One rank signifies how important the document is as a **hub of information**, and the other how important it is as an **authoritative source of information**. The resulting ranking algorithm is sometimes referred to as **HITS** i.e. a **Hypertext Induced Topic Search**. The ideas behind the model can be summarized as follows.

2.2 HITS modeling: Hubs and Authorities.

(1) A page is an **authoritative** page if it is referenced by many **hub** pages that are **relevant** to the query,

(2) a page is a **hub** page for a query if it points to many authoritative pages for that query, and

(3) good **authoritative** and **hub** pages reinforce one another.

2.3 Finding pages for a query q. How can one find such authoritative and hub pages for a query q?

2.3.1 Step 1: Submit query q to a similarity-based engine and record the top n i.e. the **root set** $RS(q)$ pages based on $s(q, d_j)$ results.

2.3.2 Step 2: Expand set $RS(q)$ into the **base set** $BS(q)$ to include pages pointed by $RS(q)$ pages. That is,
$\mathbf{BS_1(q)} = \{\textbf{all pages pointed by an } RS(q) \textbf{ page}\}$.

2.3.3 Step 3: Also include into $BS(q)$ pages pointing to $RS(q)$ pages (if possible, but restrict the number for each $RS(q)$ page to no more that a number say B) that is $\mathbf{BS_2(q)} = \{\textbf{B pages pointing to an } RS(q) \textbf{ page}\}$, for some fixed B.

2.3.4 Step 4: Thus $\mathbf{BS(q)} = \mathbf{RS(q)} \cup \mathbf{BS_1(q)} \cup \mathbf{BS_2(q)}$.

2.3.5 Authority and Hub computation on $BS(q)$ only: d_j and $a(d_j)$ and $h(d_j)$. For the subgraph induced by $BS(q)$, let E be the set of links (i.e. edges) induced by it. A link from document d_i to d_j will be denoted by the pair (i, j). For each document d_j, we shall compute the **authority score** $a(d_j)$ of d_j, and the **2.3.5.2 hub score** $h(d_j)$ of d_j by performing an A (for authority) update **followed** by an H (for hub) update **in that order**.

HITS algorithm: Authority (A) and Hub (H) update steps

2.4 Authority and Hub vectors for all d_j. All the $a(d_j)$ and $h(d_j)$ values are collected into two vectors a^t, h^t, i.e. containing n rows and one column! The ^{t} indicates the transpose, i.e. a vector of n rows a, h is shown as row vector a^t, h^t of one row and n columns.

$$h^t = (h(d_1), \ldots, h(d_n)) \qquad a^t = (a(d_1), \ldots, a(d_n))$$

Those two vectors a, h will be updated synchronously (most often) for a number of iterations. The i-th iteration will be denoted by a superscript $^{(i)}$ and values of those vectors will be denoted accordingly by $a^{(i)}$ and $h^{(i)}$ respectively.

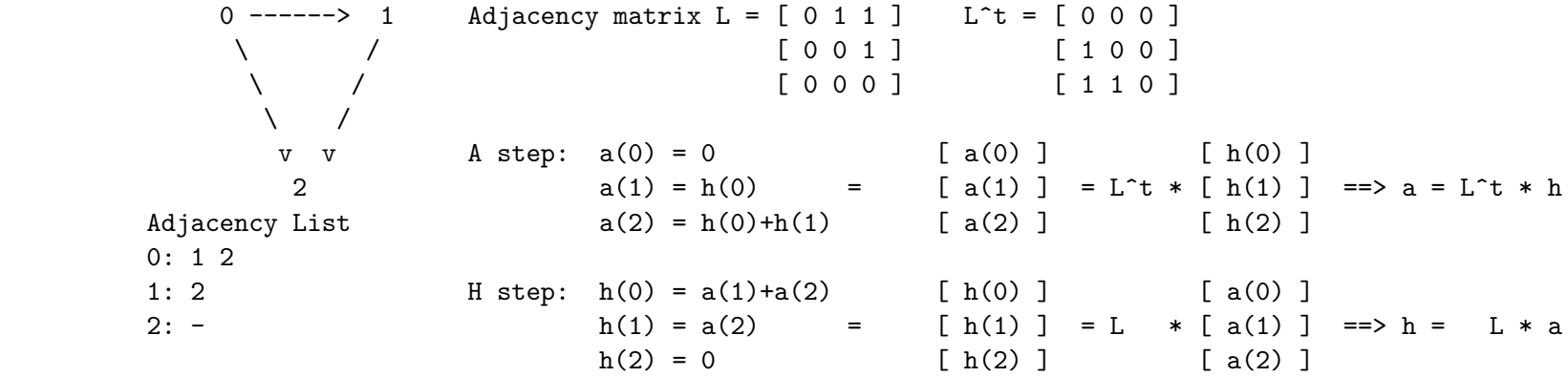
2.5.1 A: Authority update step. The first update involves the authority scores. The previously available hub scores are being used for this update. If an edge points from i to j i.e. $(i, j) \in E$ then the authority of d_j is to be updated using the score of $h(d_i)$. Therefore, for every link $(i, j) \in E$ that points to document d_j

$$a(d_j) = \sum_{(i,j)\in E} h(d_i). \tag{1}$$

2.5.2 H: Hub update step. For the hub update step the just computed authority values are to be used. For every link $(j, k) \in E$ of documents d_k pointed by d_j

$$h(d_j) = \sum_{(j,k)\in E} a(d_k). \tag{2}$$

2.6 Example 0. For the three vertex graph shown below



2.7 An A step precedes an H step (An H step follows an A step). The two steps of Eq. 1 and Eq. 2 are interwined. One first uses the h values of a previous computation (iteration) to get the a values in the A step. The newly computed a values are then used to generate a new set of h values in the H step.

2.8 Adjacency Matrix relationship of the $a^{(i)}$ and $h^{(i)}$ vectors. Let L be the **adjacency matrix** over the set of links E of $BS(q)$. With this we mean that $L(i,j) = 1$ if $(i,j) \in E$. Otherwise, $L(i,j) = 0$ for $(i,j) \notin E$. Let L^t be the transpose matrix of L i.e. one that results if the rows of L become columns in L^t and the columns respectively rows. Let $h^{(i)}$, $a^{(i)}$ be the hub, authority vectors after i iterations of the A,H steps, for the n documents of the corpus.

$$h^{(i)\prime} = (h^{(i)}(d_1), h^{(i)}(d_2), \ldots, h^{(i)}(d_n)), \qquad a^{(i)\prime} = (a^{(i)}(d_1), a^{(i)}(d_2), \ldots, a^{(i)}(d_n)),$$

Then by rewriting Eq. 1 (see also Example 0 of the previous page) we have

$$a^{(i)} = L^t h^{(i-1)}$$

and by rewriting Eq. 2 we obtain

$$h^{(i)} = L a^{(i)}.$$

2.8.1 Note the superscripts above! Note the superscripts on the right side of the equation mark in both equations. In the former case it refers to iteration $i-1$ and in the latter case to iteration i. Note also that the order of execution matters. First the authority values are updated for a given iteration i using the $i-1$ iteration hub values, and then the hub values get updated using the iteration i authority values just computed.

2.9 Example 0 continued. For the previous example, let the initial values for the a, h vectors be all ones.

```
        v  v         A step:  a(0) = 0              [ a(0) ]          [ h(0) ]
         2                    a(1) = h(0)      =    [ a(1) ]  = L^t * [ h(1) ]  ==> a = L^t * h
Adjacency List                a(2) = h(0)+h(1)     [ a(2) ]          [ h(2) ]
1: 2                 H step:  h(0) = a(1)+a(2)     [ h(0) ]          [ a(0) ]
2: -                          h(1) = a(2)      =    [ h(1) ]  = L   * [ a(1) ]  ==> h =   L * a
                              h(2) = 0             [ h(2) ]          [ a(2) ]
              h :  1    1    1     a :  1    1   1
Step 1     A:                           0    1   2
           H:         3    2    0
     scale  :      0.83 0.55 0.0         0.0 0.44 0.89
```

2.10 Power method. These recursive formulae of (2.8) are interwined. If we unfold them and unwind them, then we get the following. This is also known as the **power method**.

$$a^{(1)} = L^t h^{(0)} \qquad h^{(1)} = La^{(1)}$$
$$a^{(2)} = L^t h^{(1)} \Rightarrow a^{(2)} = L^t \left(La^{(1)}\right) = \left(L^t L\right) a^{(1)} \qquad h^{(2)} = La^{(2)} \Rightarrow h^{(2)} = L\left(L^t h^{(1)}\right) = \left(LL^t\right) h^{(1)}$$

Thus after the first iteration, the values of a, h can then be multiplied by a power of the product LL^t or $L^t L$ to generate the a, h values in any iteration as shown on the right hand side below.

$$\begin{aligned} a^{(i)} = L^t h^{(i-1)} &\implies a^{(i)} = L^t L a^{(i-1)} \Longrightarrow \ldots \Longrightarrow a^{(i)} = (L^t L)^{i-1} a^{(1)} \\ h^{(i)} = La^{(i)} &\implies h^{(i)} = LL^t h^{(i-1)} \Longrightarrow \ldots \Longrightarrow h^{(i)} = (LL^t)^{i-1} h^{(1)} \end{aligned}$$

2.11 Instability and Scaling: Avoiding growing values. One step is missing in this computation. The values $a^{(i)}, h^{(i)}$ can grow large as they are dependent on $(L^t L)^i$ and $(LL^t)^i$. Thus a scaling needs to be performed immediately after a round of A and H updates gets completed to make sure than those values are no more than 1. This scaling is shown below. The scaling step can appear as step 2.5.3 immediately after the 2.5.2 H step.

$$a^{(i)}(d_j) = \frac{a^{(i)}(d_j)}{\sqrt{\sum_k (a^{(i)}(d_k))^2}}, \quad h^{(i)}(d_j) = \frac{h^{(i)}(d_j)}{\sqrt{\sum_k (h^{(i)}(d_k))^2}},$$

For the computation to work properly and this iterative process to converge, and Algorithm `HubAuthority-Rank` (shown two pages after this page) to be complete, we need to explain two missing parts from this exposition.

Issue 1. What are the initialization vectors $a^{(0)}$ and $h^{(0)}$? (The answer is in line 4 of Algorithm `HubAuthority-Rank`.)

Issue 2. Why does the algorithm converge? How fast is convergence? What are the preconditions for convergence?

2.12 Initial values to address Issue 1. The initial values of $a^{(0)}$ and $h^{(0)}$ affect convergence. In order to have convergence, $a^{(0)}$ and $h^{(0)}$ must be chosen in such a way that they are not orthogonal to the solution $a^{(i)}$ and $h^{(i)}$ of the i-th iteration. Note also that only $h^{(0)}$ needs to be initialized, not $a^{(0)}$ since the former is only used in the first A step to generate $a^{(1)}$. In the subsequent H step, the $h^{(1)}$ will use the $a^{(1)}$ values just computed.

2.12.1 One choice of initial values: all-one values. A choice $a^{(0)} = h^{(0)} = e$, where e is the unit vector (i.e. all its components are ones) suffices.

2.12.2 One choice of initial values: all-$1/n$ values. Another alternative is to set $a^{(0)} = h^{(0)} = e/n$ i.e. scale the initial values.

2.12.3 A choice of initial values: all-$1/\sqrt{n}$ values. Another alternative is to set $a^{(0)} = h^{(0)} = e/\sqrt{n}$ i.e. scale the initial values according to the scaling method of the A and H steps.

2.12.4 Full Answer to Issue 1. Initial values are all ones for the hub and authority vector elements as indicated in step 4 of the Algorithm `HubAuthority-Rank`. A "more appropriate" alternative is $e/\sqrt{n}$ though.

2.13 Convergence to address Issue 2. Both matrices LL^t and L^tL are well-defined and by matrix analysis (e.g. Perron-Frobenius theorem) vectors $a^{(i)}$ and $h^{(i)}$ will then converge to the dominant eigenvector of one or the other matrix.

2.13.1 Rate of convergence. The rate of convergence however depends on the difference (i.e. **eigengap**) between the largest λ_1 and second largest λ_2 eigenvalues of the two matrices. HITS usually converges (but no guarantee is offered) to a solution after 10-20 iterations.

2.14 A mild critique of the mathematical properties of HITS. One problem with HITS is the existence of multiple connected components in the graph represented by L. Then the algorithm might ignore all but one of these components when it converges. We are brief at this point on the mathematical properties of HITS; a more thorough discussion that also relates to HITS will deal with Google's PageRank algorithm.

HITS: Pseudocode

Algorithm `HubAuthority-Rank` depicts not only the A,H update steps but also the correspondin scaling.

```
  Algorithm HubAuthority-Rank
1.  Obtain RS(q) for query q.
2.  Expand RS(q) into BS(q) as described.
3.  Use BS(q) to obtain a graph G=(V,E) of links with adjacency matrix L
4.  For each dj initialize hub and authority values to 1 i.e.   a(dj) = h(dj ) = 1;
5.  for every document dj component of a,h  repeat {
                                              (i)            (i-1)
6.    Run the authority update step A to get a    (dj)  from h    (...)
                                              (i)             (i)
7.    Run the hub       update step H to get h    (dj)  from a    (...)
             (i)      (i)
8.    Scale a   (dj) h  (dj) as explained.
    } until convergence is reached
9.  Return documents  ordered by authority scores (high first)
```

Figure 1: Kleinberg's HITS algorithm computation

In the scaling step one can remove the superscripts that indicate iteration and have no other significance in the computation to obtain a simpler

$$a(d_j) = \frac{a(d_j)}{\sqrt{\sum_i a^2(d_i)}}, \quad h(d_j) = \frac{h(d_j)}{\sqrt{\sum_i h^2(d_i)}},$$

(Note that the denominator is computed partially and only once by the end of step 7, then inverted so that multiplication rather than the more expensive division is performed.)

HITS: Example 0 completed

For the example of page 4, we show the computations involved for 7 iterations using (a) initial all-one vectors, (b) initial all-$1/n$ vectors, and (c) initial all-$1/\sqrt{n}$ vectors.

```
Kleinberg's HITS (Hub and Authority) Algorithm
 0 :  0  1  1
 1 :  0  0  1   (Adjacency list)
 2 :  0  0  0
(a) vectors are initialized to = 1.000000
Base   :  0 :A/H[ 0]=1.00000/1.00000 A/H[ 1]=1.00000/1.00000 A/H[ 2]=1.00000/1.00000
Iterat :  1 :A/H[ 0]=0.00000/0.83205 A/H[ 1]=0.44721/0.55470 A/H[ 2]=0.89443/0.00000
Iterat :  2 :A/H[ 0]=0.00000/0.84800 A/H[ 1]=0.51450/0.53000 A/H[ 2]=0.85749/0.00000
Iterat :  3 :A/H[ 0]=0.00000/0.85027 A/H[ 1]=0.52410/0.52635 A/H[ 2]=0.85166/0.00000
Iterat :  4 :A/H[ 0]=0.00000/0.85059 A/H[ 1]=0.52549/0.52582 A/H[ 2]=0.85080/0.00000
Iterat :  5 :A/H[ 0]=0.00000/0.85064 A/H[ 1]=0.52570/0.52574 A/H[ 2]=0.85067/0.00000
Iterat :  6 :A/H[ 0]=0.00000/0.85065 A/H[ 1]=0.52573/0.52573 A/H[ 2]=0.85065/0.00000
Iterat :  7 :A/H[ 0]=0.00000/0.85065 A/H[ 1]=0.52573/0.52573 A/H[ 2]=0.85065/0.00000

(b) vectors are initialized to = 0.333333
Base   :  0 :A/H[ 0]=0.33333/0.33333 A/H[ 1]=0.33333/0.33333 A/H[ 2]=0.33333/0.33333
Iterat :  1 :A/H[ 0]=0.00000/0.83205 A/H[ 1]=0.44721/0.55470 A/H[ 2]=0.89443/0.00000
Iterat :  2 :A/H[ 0]=0.00000/0.84800 A/H[ 1]=0.51450/0.53000 A/H[ 2]=0.85749/0.00000
Iterat :  3 :A/H[ 0]=0.00000/0.85027 A/H[ 1]=0.52410/0.52635 A/H[ 2]=0.85166/0.00000
Iterat :  4 :A/H[ 0]=0.00000/0.85059 A/H[ 1]=0.52549/0.52582 A/H[ 2]=0.85080/0.00000
Iterat :  5 :A/H[ 0]=0.00000/0.85064 A/H[ 1]=0.52570/0.52574 A/H[ 2]=0.85067/0.00000
Iterat :  6 :A/H[ 0]=0.00000/0.85065 A/H[ 1]=0.52573/0.52573 A/H[ 2]=0.85065/0.00000
Iterat :  7 :A/H[ 0]=0.00000/0.85065 A/H[ 1]=0.52573/0.52573 A/H[ 2]=0.85065/0.00000

(c) vectors are initialized to = 0.577350
Base   :  0 :A/H[ 0]=0.57735/0.57735 A/H[ 1]=0.57735/0.57735 A/H[ 2]=0.57735/0.57735
Iterat :  1 :A/H[ 0]=0.00000/0.83205 A/H[ 1]=0.44721/0.55470 A/H[ 2]=0.89443/0.00000
Iterat :  2 :A/H[ 0]=0.00000/0.84800 A/H[ 1]=0.51450/0.53000 A/H[ 2]=0.85749/0.00000
Iterat :  3 :A/H[ 0]=0.00000/0.85027 A/H[ 1]=0.52410/0.52635 A/H[ 2]=0.85166/0.00000
Iterat :  4 :A/H[ 0]=0.00000/0.85059 A/H[ 1]=0.52549/0.52582 A/H[ 2]=0.85080/0.00000
Iterat :  5 :A/H[ 0]=0.00000/0.85064 A/H[ 1]=0.52570/0.52574 A/H[ 2]=0.85067/0.00000
Iterat :  6 :A/H[ 0]=0.00000/0.85065 A/H[ 1]=0.52573/0.52573 A/H[ 2]=0.85065/0.00000
Iterat :  7 :A/H[ 0]=0.00000/0.85065 A/H[ 1]=0.52573/0.52573 A/H[ 2]=0.85065/0.00000
```

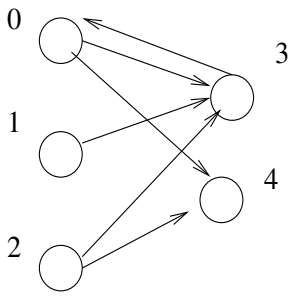



Figure 2: HITS algorithm: An example graph

Example 1. For the graph in Figure 2 we keep things simple as before.

```
    |0  1  2  3  4    Iteration  0       a=(1   1   1   1   1)     h =(  1   1   1   1   1)
   ---------------
A= 0|0  0  0  1  1    Iteration  1  A:   a=(1   0   0   3   2) <---[use h of iteration 0]
L= 1|0  0  0  1  0                  H:   [use a of iteration 1]--->h =(  5   3   5   1   0)
   2|0  0  0  1  1              Scale:   a=(.26 0   0  .80 .53)    h =(.64 .38 .64 .12   0)
   3|1  0  0  0  0    [Note sqrt(14)=3.74 and sqrt(60)=7.74]
   4|0  0  0  0  0    after few more iterations
                                       a=(0   0   0  .78 .61)    h =(.65  .36 .65  0   0)
```

As a result of those computations the ranking of the 5 pages based on authority values is $3, 4, 0, 1, 2$ since page 3 has authority rank 0.78.

HITS: Example 1 continued

```
(a) vectors are initialized to = 1.000000
Base   :  0 :A/H[ 0]=1.00000/1.00000 A/H[ 1]=1.00000/1.00000 A/H[ 2]=1.00000/1.00000 A/H[ 3]=1.00000/1.00000 A/H[ 4]=1.00000/1.00000
Iterat :  1 :A/H[ 0]=0.26726/0.64550 A/H[ 1]=0.00000/0.38730 A/H[ 2]=0.00000/0.64550 A/H[ 3]=0.80178/0.12910 A/H[ 4]=0.53452/0.00000
Iterat :  2 :A/H[ 0]=0.06086/0.65634 A/H[ 1]=0.00000/0.37097 A/H[ 2]=0.00000/0.65634 A/H[ 3]=0.79115/0.02854 A/H[ 4]=0.60858/0.00000
Iterat :  3 :A/H[ 0]=0.01337/0.65712 A/H[ 1]=0.00000/0.36924 A/H[ 2]=0.00000/0.65712 A/H[ 3]=0.78856/0.00626 A/H[ 4]=0.61481/0.00000
Iterat :  4 :A/H[ 0]=0.00293/0.65719 A/H[ 1]=0.00000/0.36907 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78824/0.00137 A/H[ 4]=0.61536/0.00000
Iterat :  5 :A/H[ 0]=0.00064/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00030 A/H[ 4]=0.61541/0.00000
Iterat :  6 :A/H[ 0]=0.00014/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00007 A/H[ 4]=0.61541/0.00000
Iterat :  7 :A/H[ 0]=0.00003/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00001 A/H[ 4]=0.61541/0.00000
Iterat :  8 :A/H[ 0]=0.00001/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00000 A/H[ 4]=0.61541/0.00000
Iterat :  9 :A/H[ 0]=0.00000/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00000 A/H[ 4]=0.61541/0.00000
Iterat : 10 :A/H[ 0]=0.00000/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00000 A/H[ 4]=0.61541/0.00000

(b) vectors are initialized to = 0.447214
Base   :  0 :A/H[ 0]=0.44721/0.44721 A/H[ 1]=0.44721/0.44721 A/H[ 2]=0.44721/0.44721 A/H[ 3]=0.44721/0.44721 A/H[ 4]=0.44721/0.44721
Iterat :  1 :A/H[ 0]=0.26726/0.64550 A/H[ 1]=0.00000/0.38730 A/H[ 2]=0.00000/0.64550 A/H[ 3]=0.80178/0.12910 A/H[ 4]=0.53452/0.00000
Iterat :  2 :A/H[ 0]=0.06086/0.65634 A/H[ 1]=0.00000/0.37097 A/H[ 2]=0.00000/0.65634 A/H[ 3]=0.79115/0.02854 A/H[ 4]=0.60858/0.00000
Iterat :  3 :A/H[ 0]=0.01337/0.65712 A/H[ 1]=0.00000/0.36924 A/H[ 2]=0.00000/0.65712 A/H[ 3]=0.78856/0.00626 A/H[ 4]=0.61481/0.00000
Iterat :  4 :A/H[ 0]=0.00293/0.65719 A/H[ 1]=0.00000/0.36907 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78824/0.00137 A/H[ 4]=0.61536/0.00000
Iterat :  5 :A/H[ 0]=0.00064/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00030 A/H[ 4]=0.61541/0.00000
Iterat :  6 :A/H[ 0]=0.00014/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00007 A/H[ 4]=0.61541/0.00000
Iterat :  7 :A/H[ 0]=0.00003/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00001 A/H[ 4]=0.61541/0.00000
Iterat :  8 :A/H[ 0]=0.00001/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00000 A/H[ 4]=0.61541/0.00000
Iterat :  9 :A/H[ 0]=0.00000/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00000 A/H[ 4]=0.61541/0.00000
Iterat : 10 :A/H[ 0]=0.00000/0.65719 A/H[ 1]=0.00000/0.36905 A/H[ 2]=0.00000/0.65719 A/H[ 3]=0.78821/0.00000 A/H[ 4]=0.61541/0.00000
```


2.15.1 Terminology: Transverse link information. Links between pages with different domain names.

2.15.2 Intrinsic link information. Links within the same domain.

2.16.1 Question 1. What if someone creates hundreds of pages each one pointing to all the other ones with each one of those pages including lots of "hot" keywords so that they are included in the initial $BS(q)$ set?

2.16.2 Question 2. What if some one does the same over a set of web-sites rather than web-pages of the same site?

Answer to Questions 1 and 2. One can use different weights for the two types of transverse and intrinsic links or ignore the second type (e.g. weight becomes 0).

2.16.3 Question 3. Say x web-pages (or web-sites if we got wised up by the prior page discussion) point to page/URL X and y pages point to URL Y where $x \gg y$. What happens with the hubs and authority values of X and Y respectively? Or one page X points to x pages and one page Y points to y pages and nothing else. What happens to the pointed pages?

Answer to Question 3. We show what happens with the HITS algorithm after a single iteration of it for this case. For page X its authority value becomes x and for page Y it becomes y after the A step. Since $x \gg y$ X's authority is larger than Y's. When the hub step H is performed, the hub values of the pages pointing to X get upped to x and those of Y get (less) upped to y. After scaling, whatever the scaled values will be, we still expect $x' \gg y'$ and of course $d' \gg f'$.

```
                      X  Y  x-pages   y-pages          X  Y  x-pages  y-pages
Iteration 0 :      a=( 1  1  1 ... 1   1 ... 1)    h=( 1  1  1 ... 1  1 ... 1 )

Iteration 1 :  A:  a=( x  y  0     0   0     0)
               H:                                  h=( 0  0  x     x  y     y )
          Scale :  a=( x' y' 0     0   0     0)    h=( 0  0  d'    d' f'    f')
```

The important observation is that $a(X)/a(Y) = x/y \gg 1$. After another iteration, $a(X)/a(Y) = (x/y)^2 \gg 1$ and so on. Thus after i iterations we expect $a(X)/a(Y) = (x/y)^i \gg 1$ and thus $a(X)$ would be much larger than $a(Y)$. Thus the more popular X will dominate the less popular Y completely. (The same would apply to hub values as well of the x and y pages.) So if $x/y = 10$ after 3-4 iterations the X-related values will grow to become 1000 to 10000 larger than the Y-related values. X will completely dominate Y. In the example next page $x = 3$ and $y = 2$, and the 4 vertices $3, 4, 5, 6$ dominate the triplet $0, 1, 2$ in hub and authority values.

How to fool HITS: Example 2 (two variations)

```
Kleinberg Hub and Authority Algorithm
 0 :  0  0  1  0  0  0  0    2 is pointed by 0 and 1       but 6 is pointed by 3,4,5
 1 :  0  0  1  0  0  0  0    After few iterations  6 has high authority value
 2 :  0  0  0  0  0  0  0    which reinforced the hub value of 3,4,5
 3 :  0  0  0  0  0  0  1    0,1,2 have low hub/authority values
 4 :  0  0  0  0  0  0  1
 5 :  0  0  0  0  0  0  1
 6 :  0  0  0  0  0  0  0
Base:0 :A/H[0]=1.00000/1.00000 A/H[1]=1.00000/1.00000 A/H[2]=1.00000/1.00000 A/H[3]=1.00000/1.00000 A/H[4]=1.00000/1.00000 A/H[5]=1.00000/1.00000 A/H[6]=1.00000/1.00000
Iter:1 :A/H[0]=0.00000/0.33806 A/H[1]=0.00000/0.33806 A/H[2]=0.55470/0.00000 A/H[3]=0.00000/0.50709 A/H[4]=0.00000/0.50709 A/H[5]=0.00000/0.50709 A/H[6]=0.83205/0.00000
Iter:2 :A/H[0]=0.00000/0.24121 A/H[1]=0.00000/0.24121 A/H[2]=0.40614/0.00000 A/H[3]=0.00000/0.54272 A/H[4]=0.00000/0.54272 A/H[5]=0.00000/0.54272 A/H[6]=0.91381/0.00000
Iter:3 :A/H[0]=0.00000/0.16627 A/H[1]=0.00000/0.16627 A/H[2]=0.28409/0.00000 A/H[3]=0.00000/0.56116 A/H[4]=0.00000/0.56116 A/H[5]=0.00000/0.56116 A/H[6]=0.95880/0.00000
Iter:4 :A/H[0]=0.00000/0.11259 A/H[1]=0.00000/0.11259 A/H[2]=0.19379/0.00000 A/H[3]=0.00000/0.56998 A/H[4]=0.00000/0.56998 A/H[5]=0.00000/0.56998 A/H[6]=0.98104/0.00000
Iter:5 :A/H[0]=0.00000/0.07559 A/H[1]=0.00000/0.07559 A/H[2]=0.13056/0.00000 A/H[3]=0.00000/0.57404 A/H[4]=0.00000/0.57404 A/H[5]=0.00000/0.57404 A/H[6]=0.99144/0.00000
Iter:6 :A/H[0]=0.00000/0.05056 A/H[1]=0.00000/0.05056 A/H[2]=0.08746/0.00000 A/H[3]=0.00000/0.57587 A/H[4]=0.00000/0.57587 A/H[5]=0.00000/0.57587 A/H[6]=0.99617/0.00000
Iter:7 :A/H[0]=0.00000/0.03375 A/H[1]=0.00000/0.03375 A/H[2]=0.05843/0.00000 A/H[3]=0.00000/0.57669 A/H[4]=0.00000/0.57669 A/H[5]=0.00000/0.57669 A/H[6]=0.99829/0.00000
Iter:8 :A/H[0]=0.00000/0.02252 A/H[1]=0.00000/0.02252 A/H[2]=0.03899/0.00000 A/H[3]=0.00000/0.57706 A/H[4]=0.00000/0.57706 A/H[5]=0.00000/0.57706 A/H[6]=0.99924/0.00000
Iter:9 :A/H[0]=0.00000/0.01501 A/H[1]=0.00000/0.01501 A/H[2]=0.02600/0.00000 A/H[3]=0.00000/0.57722 A/H[4]=0.00000/0.57722 A/H[5]=0.00000/0.57722 A/H[6]=0.99966/0.00000
Iter:10:A/H[0]=0.00000/0.01001 A/H[1]=0.00000/0.01001 A/H[2]=0.01734/0.00000 A/H[3]=0.00000/0.57729 A/H[4]=0.00000/0.57729 A/H[5]=0.00000/0.57729 A/H[6]=0.99985/0.00000
 0 :  0  1  1  0  0  0  0  In this variation 0 points to 1 and 2, and 3 points to 4,5,6
 1 :  0  0  0  0  0  0  0  After few iterations the hub value of 3 and authority values of 4,5,6 exceed
 2 :  0  0  0  0  0  0  0  by 30-40 times the values of 0,1,2. Note that 1.5**10 ~ 57
 3 :  0  0  0  0  1  1  1
 4 :  0  0  0  0  0  0  0
 5 :  0  0  0  0  0  0  0
 6 :  0  0  0  0  0  0  0
Base:0 :A/H[0]=1.00000/1.00000 A/H[1]=1.00000/1.00000 A/H[2]=1.00000/1.00000 A/H[3]=1.00000/1.00000 A/H[4]=1.00000/1.00000 A/H[5]=1.00000/1.00000 A/H[6]=1.00000/1.00000
Iter:1 :A/H[0]=0.00000/0.55470 A/H[1]=0.44721/0.00000 A/H[2]=0.44721/0.00000 A/H[3]=0.00000/0.83205 A/H[4]=0.44721/0.00000 A/H[5]=0.44721/0.00000 A/H[6]=0.44721/0.00000
Iter:2 :A/H[0]=0.00000/0.40614 A/H[1]=0.33806/0.00000 A/H[2]=0.33806/0.00000 A/H[3]=0.00000/0.91381 A/H[4]=0.50709/0.00000 A/H[5]=0.50709/0.00000 A/H[6]=0.50709/0.00000
Iter:3 :A/H[0]=0.00000/0.28409 A/H[1]=0.24121/0.00000 A/H[2]=0.24121/0.00000 A/H[3]=0.00000/0.95880 A/H[4]=0.54272/0.00000 A/H[5]=0.54272/0.00000 A/H[6]=0.54272/0.00000
Iter:4 :A/H[0]=0.00000/0.19379 A/H[1]=0.16627/0.00000 A/H[2]=0.16627/0.00000 A/H[3]=0.00000/0.98104 A/H[4]=0.56116/0.00000 A/H[5]=0.56116/0.00000 A/H[6]=0.56116/0.00000
Iter:5 :A/H[0]=0.00000/0.13056 A/H[1]=0.11259/0.00000 A/H[2]=0.11259/0.00000 A/H[3]=0.00000/0.99144 A/H[4]=0.56998/0.00000 A/H[5]=0.56998/0.00000 A/H[6]=0.56998/0.00000
Iter:6 :A/H[0]=0.00000/0.08746 A/H[1]=0.07559/0.00000 A/H[2]=0.07559/0.00000 A/H[3]=0.00000/0.99617 A/H[4]=0.57404/0.00000 A/H[5]=0.57404/0.00000 A/H[6]=0.57404/0.00000
Iter:7 :A/H[0]=0.00000/0.05843 A/H[1]=0.05056/0.00000 A/H[2]=0.05056/0.00000 A/H[3]=0.00000/0.99829 A/H[4]=0.57587/0.00000 A/H[5]=0.57587/0.00000 A/H[6]=0.57587/0.00000
Iter:8 :A/H[0]=0.00000/0.03899 A/H[1]=0.03375/0.00000 A/H[2]=0.03375/0.00000 A/H[3]=0.00000/0.99924 A/H[4]=0.57669/0.00000 A/H[5]=0.57669/0.00000 A/H[6]=0.57669/0.00000
Iter:9 :A/H[0]=0.00000/0.02600 A/H[1]=0.02252/0.00000 A/H[2]=0.02252/0.00000 A/H[3]=0.00000/0.99966 A/H[4]=0.57706/0.00000 A/H[5]=0.57706/0.00000 A/H[6]=0.57706/0.00000
Iter:10:A/H[0]=0.00000/0.01734 A/H[1]=0.01501/0.00000 A/H[2]=0.01501/0.00000 A/H[3]=0.00000/0.99985 A/H[4]=0.57722/0.00000 A/H[5]=0.57722/0.00000 A/H[6]=0.57722/0.00000
```


3.0.1 Limitations of the HITS algorithm. The HITS algorithm has certain limitations. To start with we need access to a similarity search engine to obtain RS plus the additional steps to formulate BS. Subsequent ranking depends on the BS that depends on the given query q. For this reason we write $BS(q)$ to indicate that BS depends on the query q. Moreover, HITS uses two vectors a, h. If the web-graph related to $BS(q)$ has isolated components the HITS algorithm may focus on one such component and miss the other ones (midredirection). It is not clear whether the a vector should be used for rank, or the h vector or it makes sense to use a combination of the two and what that combination should be!

3.0.2 Nice features of HITS. The iterative (aka power method) form of the HITS algorithm makes it easy (though time consuming) to establish hub and authority values and makes matters of convergence and existence of a, h values contingent of explicit conditions of the matrix of document (web-page) connections.

3.1 Google's PageRank algorithm. The algorithm used to ranking Google query results is now presented. The original algorithm, named `PageRank` was introduced in 1998. How much it is still being used in Google is open to discussion.

3.1.1 Overview. We will be introducing the mathematics behind `PageRank` through some linear and matrix algebra considerations related to adjacency-matrix/adjacency-list properties of graphs. A more thorough and mathematical oriented discussion starts with page 20. (One can skip directly this latter discussion that starts on page 20.)

3.2.1 Eigenvectors and Eigenvalues. For an $n \times n$ matrix A, vector x is called the (right) eigenvector of eigenvalue λ of A if and only if $Ax = \lambda x$.

3.2.2 Spectral theorem. The spectral theorem for matrices states that if there are n eigenvectors of A that form a basis (i.e. they are linearly independent) of dimension n, then A can be written as $A = Q\Lambda Q^{-1}$, where Q is a matrix whose i-th column is the i-th eigenvector and Λ is a diagonal matrix whose i-th diagonal entry is λ_i (i.e. the eigenvalue corresponding to the i-th eigenvector).

3.2.3 Determinant of A and its eigenvalues. From matrix theory we also have that the determinant of A is the products of its eigenvalues, i.e. $\mid A \mid = \lambda_1 \ldots \lambda_n$.

3.2.4 Convention. From now on we will assume that the eigenvalues are relabeled and numbered so that $\lambda_1 > \lambda_2 > \ldots$ i.e. we list the eigenvalues from the largest to the smallest one. This might work for positive and real eigenvalues; such a listing can also be extended for negative and/or complex eigenvalues if we use $\mid \lambda_1 \mid$ where $\mid . \mid$ represents the measure of a complex number or the absolute value of a real number.

3.2.5 Unit vector e. Vector e is the unit vector i.e. it is an all-one vector i.e. $e^t = (1, 1, \ldots, 1)$.

3.3 Some graph theory: Adjacency Matrix representation of a graph. Let $G = (V, E)$ be a graph on a set of vertices V and a set of edges E. The vertices of the graph may correspond to the web and its edges to the links of web-pages. Let n be the number of vertices of V and m be the number of edges in E. It is $m \leq n^2$.

3.3.1 Adjacency matrix L. L is the adjacency matrix of a graph whose (i, j)-th entry is 1 if $(i, j) \in E$ and 0 otherwise.

3.3.2 Transpose of adjacency matrix L. L^t is the transpose of L i.e. its (i, j)-th entry is 1 if $(j, i) \in E$ and 0 otherwise.

3.4 Weighed adjacency matrix C. The weighed adjacency matrix C can be obtained from L if its (i, j)-th entry is normalized so that $c_{ij} = 1/d(i)$ for $(i, j) \in E$ and 0 otherwise. Here, $d(i)$ is the degree (out-degree for directed graphs) for vertex i. Thus if the i-th row of L is $(0 \quad 1 \quad 1)$, then the corresponding row of C would become $(0 \quad 1/2 \quad 1/2)$.

3.4.1 An eigenvector of C. Matrix C obtained from the adjacency matrix of a graph has a nice property: $Ce = 1e$, i.e. for matrix C the unit vector is an eigenvector for eigenvalue 1. This is because the rows of C add up to 1.

3.4.2 Sinks and matrix C. For the graph of the web, matrix C might fail to hold this property (sum of rows equal to 1) if the graph contains a sink, i.e. a vertex with no outgoing edges. In that case all elements of a row are equal to 0! We can "fix" this, if sink rows are "adjusted to have $1/n$ everywhere, i.e. a row becomes e^t/n, where n is the number of vertices of the graph. This "corrected" matrix has the specified in 3.4.1 property.

3.4.3 Transpose C^t of C. Transposing matrix C to get C^t we obtain a matrix that has an (i, j)-th entry that is $1/d(j)$ for $(j, i) \in E$.

3.5 Idea behind Google's view of the Web: PageRank vs HITS. View the web as a directed graph $G = (V, E)$. The HITS algorithm views as a directed graph not the web but only the base set $BS(q)$; to obtain $BS(q)$ one has to submit q to a similarity-based search engine, obtain $RS(q)$ and then enhance it into $BS(q)$. Google's view of the web is a direct view skipping all those intermediate steps.

3.5.1 Forward (outgoing) links and backlinks (incoming) establish a rank independent of q. Each page p has outgoing/forward links and incoming/backlinks. Each backlink is a citation of the named page p; each outgoing link cites the page pointed by p. Google's original ranking algorithm named **PageRank** is a measure of global web-page importance not of an importance based on the search query results of a similarity search-engine.

3.5.2 1998 PageRank and Google Toolbar PageRank. With respect to that measure Google's rank is (naturally) very high, MIT's is a bit less, and NJIT scores a little less than both. Note that this `PageRank` algorithm that we will be discussing in the remainder of this Subject reflects Google's state in 1998. Google is currently using a slightly different "PageRank" measure and makes it available to Google users through what is known as Google Toolbar (Internet Explorer); around 2012 it was a number between 0 and 10 (Google itself used to be a 10, MIT had a 9 rank, and NJIT an 8).

3.6 Graph of the Web and Browsing: Random Surfing. The idea behind PageRank is simple. Consider the web to be a graph and do the following random walk on it by following the protocol described by the following two rules.

Rule 1. For $d \times 100\%$ (say 85%) of the time, a user follows (outgoing) links (uniformly) at random when the user browses a web-page. (For example, if a web-page has 5 links the user picks one of them at random.) Then the user follows such a selected outgoing link and browses it.

Rule 2. For $(1-d) \times 100\%$ of the time, a web-page might become a sink (no linked pages) or the user just gets bored browsing the current web-page. So instead of using Rule 1 the user decides to pick a random page (not pointed by the current page) to visit next. (The choice of a random page is one among the n web-pages of the whole Web.)

3.7 Parameter $1-d$ decides how often the user gets "bored". Parameter d is a number between 0 and 1. For example one might choose $d = 0.85$.

3.8 Fundamental Question. What is the probability that a given page A will be visited? This determines the "importance" i.e. the "rank" of the page!

3.9 Idea of ranking web-pages. If a page p is linked by **many** pages then p is important. If a page p is linked by **few but important pages** then p should also be important. The importance or not of a page is divided evenly and also inherited to the pages pointed by it. Suppose we have pages $T_1, \ldots, T_m$ pointing to page A. The PageRank $PR(A)$ of A is then defined by the pagerank of the pages pointing to A.

$$PR(A) = \frac{(1-d)}{n} + \frac{1}{d} \cdot \left(\frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_m)}{C(T_m)} \right), \tag{3}$$

where $PR(T_i)$ is the page rank of T_i and $C(T_i)$ is the number of outgoing links of T_i, and n is the total number of pages in the corpus (web). (Sometimes instead of $(1-d)/n$ one can use $(1-d)$ this cause some matrix or algebra-related problems though.)

3.9.1 Technical issues to resolve: How to avoid sinks. Each page has its own PageRank (eg. $PR(A), PR(T_i)$). Each page is recording its number of outgoing links i.e. $C(T_i)$. The RageRank of a page T_i is evenly distributed among the pages pointed by T_i and one of them is A. Each such page gets a rank contribution of $PR(T_i)/C(T_i)$. Even if a page has no citations (i.e. incoming links) it will still have a non-zero rank of $(1-d)/n$ (though for n a billion or so it would be negligibly small). To be fair to every web page, all pages' ranks get pumped up by $(1-d)/n$. The choice of d is important in controlling the rate of convergence in a PageRank computation. (We have $(1-d)/n$ instead of $1-d$ to avoid instability issues by keeping PageRanks smaller than one, and also because we do not want to be too generous.)

3.9.2 PageRank $PR(A)$ for A. The PageRank $PR(A)$ of web-page A thus

$$PR(A) = \frac{(1-d)}{n} + \frac{1}{d} \cdot \left(\frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_m)}{C(T_m)} \right) = \frac{1-d}{n} + d \cdot \sum_{i=1}^{m} \frac{PR(T_i)}{C(T_i)} \tag{4}$$

3.9.3 Note: Rank as "probability". The normalized rank of Eq. (4) could be interpreted to signify probabilities. Thus one might interpret $PR(A)$ as the probability of visiting web-page A during the random walk obtained through the application of the two rules, **Rule 1** and **Rule 2**.

3.10.1 In order to compute $PR(A)$ we need to know $PR(T_i)$. To evaluate the rank $PR(A)$ of A one needs to know the ranks $PR(T_i)$ of all pages T_i pointing to A.

3.10.2. In order to obtain $PR(T_i)$ we need to know T_i. And this requires that we know in advance all the pages that point to A which is a little bit more elaborate than finding the pages pointed by A. The latter is easy : scan document A and locate all anchor fields to determine pages pointed by A. But finding T_i has not yet been resolved.

3.10.3. Even if we know T_i to find $PR(T_i)$ we need to find the pages pointing to T_i! Not only that but in order to find these ranks $PR(T_i)$ of T_i's one needs to know the ranks of the pages pointing to the T_i's.

3.10.4. Chicken and egg problem? What if we have a cycle or a very long back-chain?.

3.10.5. Matrix theory to the rescue. Matrix theory and the power method alluded in the HITS algorithm (see pages 5 and 6) can be used to show that using a simple iterative algorithm (that was also employed in a different form in HITS), PR(A) can be computed in a few iterations and the rank vector (of all n web-pages) computed corresponds to the principal eigenvector of the normalized (and weighed) link matrix of the web. The term `normalized/weighed` means that all sinks have been eliminated and the graph is used by employing the C matrix rather than the original L form and thus rank is expressed through Eq. 4.

3.11. Rank vector. If we have pages $A, B, C, \ldots$ we formulate the PageRank vector PR which is then the vector $PR = (PR(A), PR(B), PR(C), \ldots,)^t$ of PageRanks of the n pages of the Web.

3.12. Rank vector initialization. In any rank computation we need to determine the initial values of $PR(.)$. Are they zero? One (similarly to HITS) ? Or something else? Avoid zeroes! Use $1/n$ or $1/\sqrt{n}$ if you want small numbers (aka probabilities). Or may 1 everywhere.

3.13. Asynchronous vs synchronous approach. Kleinberg's algorithm requires the use of two vectors: the "old" and the "new" set of values. "old" refers to those values of the previous iteration and "new" to the ones of the current iteration. We define this, in the context of PageRank, as the **synchronous** approach. One additional reason to use the asynchronous approach and the two set of vectors is to determine when the error rate is small and thus whether the desirable convergence has been achieved. Such convergence can be confirmed when the difference between "old" and "new" values of every rank is less than a given error bound. For example if the pagerank of a page at iterations $i-1$, i and $i+1$ are 1.00032 and 1.00021 and 1.00020 respectively, the first two are within 10^{-3} and thus stopping at iteration i allows for a 10^{-3} error bound; to get a 10^{-4} bound one has to stop at $i+1$. In an **asynchronous** approach we use one vector. We update a rank by using whatever values are available. Thus the result is possibly dependent on the order of rank evaluations.

3.14 Example 3. A points B and B points to A ranks. Two pages A and B each one pointing to the other. $C(A) = C(B) = 1$. Consider starting first with $PR(A) = PR(B) = 1/n$ and then $PR(A) = PR(B) = 1$.

3.14.1 Case 3.a: Example 3 with $1/n$ initial values. Convergence is quite fast.

```
  [IN ][OUT]
 0[  1][  1] :  0  1
 1[  1][  1] :  1  0
(a) initialization to 1/n = 1/2
Base   :  0 :P[ 0]=0.50000 P[ 1]=0.50000
Iter   :  1 :P[ 0]=0.50000 P[ 1]=0.50000
Iter   :  2 :P[ 0]=0.50000 P[ 1]=0.50000
Iter   :  3 :P[ 0]=0.50000 P[ 1]=0.50000
(b) initialization to 1/n ; asynchronous update
avalue= -1.000000 but all vectors are initialized to = 0.500000
Base   :  0 :P[ 0]=0.50000 P[ 1]=0.50000
Iter   :  1 :P[ 0]=0.50000 P[ 1]=0.50000
Iter   :  2 :P[ 0]=0.50000 P[ 1]=0.50000
Iter   :  3 :P[ 0]=0.50000 P[ 1]=0.50000
```


3.14.2 Case 3.b: Example 3 with 1 **initial values.** Convergence is achieved in 70 plus or 40 plus iterations. The wave-like synchronous approach results in a slower convergence. Details follow. (Synchronous first, asynchronous next.)

```
  [IN ][OUT]
 0[  1][  1] :  0  1
 1[  1][  1] :  1  0
(c) initialization to 1.0
Base   :  0 :P[ 0]=1.00000 P[ 1]=1.00000
Iter   :  1 :P[ 0]=0.92500 P[ 1]=0.92500
Iter   :  2 :P[ 0]=0.86125 P[ 1]=0.86125
Iter   :  3 :P[ 0]=0.80706 P[ 1]=0.80706
Iter   :  4 :P[ 0]=0.76100 P[ 1]=0.76100
  ...
Iter   : 40 :P[ 0]=0.50075 P[ 1]=0.50075
Iter   : 41 :P[ 0]=0.50064 P[ 1]=0.50064
  ...
Iter   : 60 :P[ 0]=0.50003 P[ 1]=0.50003
Iter   : 61 :P[ 0]=0.50002 P[ 1]=0.50002
  ...
Iter   : 70 :P[ 0]=0.50001 P[ 1]=0.50001
Iter   : 71 :P[ 0]=0.50000 P[ 1]=0.50000
Iter   : 72 :P[ 0]=0.50000 P[ 1]=0.50000
(d) initialization to 1.0 ; asynchronous update
Base   :  0 :P[ 0]=1.00000 P[ 1]=1.00000
Iter   :  1 :P[ 0]=0.92500 P[ 1]=0.86125
Iter   :  2 :P[ 0]=0.80706 P[ 1]=0.76100
Iter   :  3 :P[ 0]=0.72185 P[ 1]=0.68857
  ...
Iter   : 35 :P[ 0]=0.50001 P[ 1]=0.50001
Iter   : 36 :P[ 0]=0.50000 P[ 1]=0.50000
Iter   : 37 :P[ 0]=0.50000 P[ 1]=0.50000
```


3.15 Example 4. We use the synchronous approach first and then the asynchronous one. For both cases initial values are $1/n$.

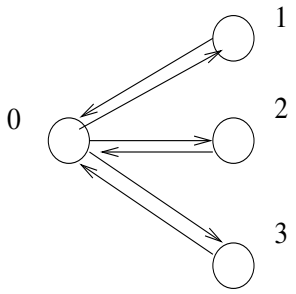


Figure 3: Simple Example 4

```
(a) Synchronous: All vectors are initialized to = 0.250000
Base   :  0 :P[ 0]=0.25000 P[ 1]=0.25000 P[ 2]=0.25000 P[ 3]=0.25000   [IN ][OUT]
Iter   :  1 :P[ 0]=0.67500 P[ 1]=0.10833 P[ 2]=0.10833 P[ 3]=0.10833  0[  3][  3] :  0  1  1  1
Iter   :  2 :P[ 0]=0.31375 P[ 1]=0.22875 P[ 2]=0.22875 P[ 3]=0.22875  1[  1][  1] :  1  0  0  0
Iter   :  3 :P[ 0]=0.62081 P[ 1]=0.12640 P[ 2]=0.12640 P[ 3]=0.12640  2[  1][  1] :  1  0  0  0
  ...                                                                 3[  1][  1] :  1  0  0  0
Iter   : 66 :P[ 0]=0.47972 P[ 1]=0.17343 P[ 2]=0.17343 P[ 3]=0.17343
Iter   : 67 :P[ 0]=0.47973 P[ 1]=0.17342 P[ 2]=0.17342 P[ 3]=0.17342
Iter   : 68 :P[ 0]=0.47973 P[ 1]=0.17342 P[ 2]=0.17342 P[ 3]=0.17342

(b) Asynchronous: All vectors are initialized to = 0.250000
Base   :  0 :P[ 0]=0.25000 P[ 1]=0.25000 P[ 2]=0.25000 P[ 3]=0.25000
Iterat :  1 :P[ 0]=0.67500 P[ 1]=0.22875 P[ 2]=0.22875 P[ 3]=0.22875
Iterat :  2 :P[ 0]=0.62081 P[ 1]=0.21340 P[ 2]=0.21340 P[ 3]=0.21340
Iterat :  3 :P[ 0]=0.58166 P[ 1]=0.20230 P[ 2]=0.20230 P[ 3]=0.20230
  ....
Iterat : 28 :P[ 0]=0.47976 P[ 1]=0.17343 P[ 2]=0.17343 P[ 3]=0.17343
Iterat : 29 :P[ 0]=0.47975 P[ 1]=0.17343 P[ 2]=0.17343 P[ 3]=0.17343
Iterat : 30 :P[ 0]=0.47975 P[ 1]=0.17343 P[ 2]=0.17343 P[ 3]=0.17343
```


Example 5: Effects of collusion

3.16 Example 5. Say we have a collusion graph (cycle) i.e. one page points to another page on a cycle. If the calculations tell us something is that $1/n$ is preferable to 1 as initial values. And the asynchronous approach somehow works better.

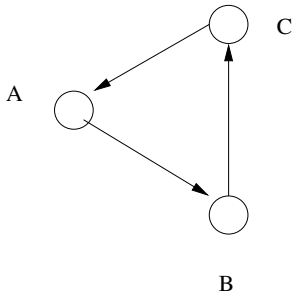


Figure 4: Collusion graph

```
(a) Synchronous (all 1/n)                              [IN ][OUT]
Base    :  0 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333  0[  1][  1] :  0  1  0
Iter    :  1 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333  1[  1][  1] :  0  0  1
(b) Asynchronous (all 1/n)                             2[  1][  1] :  1  0  0
Base    :  0 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
Iterat  :  1 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
(c) Synchronous (all 1)
Base    :  0 :P[ 0]=1.00000 P[ 1]=1.00000 P[ 2]=1.00000
Iter    :  1 :P[ 0]=0.90000 P[ 1]=0.90000 P[ 2]=0.90000
Iter    :  2 :P[ 0]=0.81500 P[ 1]=0.81500 P[ 2]=0.81500
  ....
Iter    : 79 :P[ 0]=0.33334 P[ 1]=0.33334 P[ 2]=0.33334
Iter    : 80 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
Iter    : 81 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
(d) Asynchronous (all 1)
Base    :  0 :P[ 0]=1.00000 P[ 1]=1.00000 P[ 2]=1.00000
Iterat  :  1 :P[ 0]=0.90000 P[ 1]=0.81500 P[ 2]=0.74275
Iterat  :  2 :P[ 0]=0.68134 P[ 1]=0.62914 P[ 2]=0.58477
Iterat  :  3 :P[ 0]=0.54705 P[ 1]=0.51499 P[ 2]=0.48774
  ....
Iterat : 27 :P[ 0]=0.33334 P[ 1]=0.33333 P[ 2]=0.33333
Iterat : 28 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
Iterat : 29 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
```


3.17 The PageRank algorithm (final version). We present below algorithm `PageRank` that uses Eq. (4) to compute the rank of web-pages. The initialization vector is e^t/n i.e. a vector whose all entries are $1/n$. This might work slightly better than e^t (an all-one vector) and is always preferable to an all-zero vector. For the algorithm to be efficient a proper representation of the Web-graph must be available. A compact form of it is through an adjacency list representation of the link structure i.e. for every vertex we maintain its out-degree and a list of end-points for outgoing edges as shown below

```
Source-vertex Out-degree  Endpoints-of-outgoing edges
       u        m        k1 k2 k3 ...  km
```

```
   PageRank(G,V,E)  //The code performs a SYNCHRONOUS Rank update
1. for all vertices u in V     /* Initialization Step */
2.    Src[u] = 1/n;
3. small = something-small;
4. while (convergence-distance > small) {
5.   for all v in V
6.     D[v]=0;
7.   for(i=0;i<|V|;i++) {
8.     Read-Adjacency-List(u,m,k1,k2,...,km);
9.     for(j=1;j<=m;j++)
10.      D[kj] = D[kj] + Src[u]/m
11.  }
12.  for all v in V
13.    D[v] = d * D[v] + (1-d)/n
14.  convergence-distance = ||Src-D|| /* Euclidean distance */
15.  Src=D;
16. }
```

Figure 5: PageRank

3.18.1 Practical Issue 1. What if we run out of space ? Split Vectors `Src` and `D` into B blocks.

3.18.2 Practical Issue 2. Split adjacency list (endpoint-list) into B blocks with the i-th block containing only edges with endpoints the set of vertices of the i-th block of `Src/D`. Run Algorithm above for i-th block, for all $i = 1, \ldots, B$.

3.18.3 Practical Issue 3. Parallelization of this scheme is possible if we assign each block or groups of blocks of size B/p to the p processors of a system. Details omitted.

3.19 Example 0 revisited. We first present the two cases (synchronous and asynchronous) for the graph of Example 0 and then for the graph in which the sink has been eradicated even if the PageRank model does so with the additive $(1-d)/n$ term in the PageRank formula of Eq. (4).

```
(a) Synchronous all 1/3
Base   :  0 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
Iter   :  1 :P[ 0]=0.05000 P[ 1]=0.19167 P[ 2]=0.47500
Iter   :  2 :P[ 0]=0.05000 P[ 1]=0.07125 P[ 2]=0.23417
Iter   :  3 :P[ 0]=0.05000 P[ 1]=0.07125 P[ 2]=0.13181
Iter   :  4 :P[ 0]=0.05000 P[ 1]=0.07125 P[ 2]=0.13181
(b) Asynchronous all 1/3
Base   :  0 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
Iterat :  1 :P[ 0]=0.05000 P[ 1]=0.07125 P[ 2]=0.13181
Iterat :  2 :P[ 0]=0.05000 P[ 1]=0.07125 P[ 2]=0.13181
(c) Synchronous all 1/3
Base   :  0 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
Iter   :  1 :P[ 0]=0.14444 P[ 1]=0.28611 P[ 2]=0.56944
Iter   :  2 :P[ 0]=0.21134 P[ 1]=0.27273 P[ 2]=0.51593
 ...
Iter   : 10 :P[ 0]=0.19758 P[ 1]=0.28155 P[ 2]=0.52086
Iter   : 11 :P[ 0]=0.19758 P[ 1]=0.28155 P[ 2]=0.52087
Iter   : 12 :P[ 0]=0.19758 P[ 1]=0.28155 P[ 2]=0.52087
(d) Asynchronous all 1/3
Base   :  0 :P[ 0]=0.33333 P[ 1]=0.33333 P[ 2]=0.33333
Iterat :  1 :P[ 0]=0.14444 P[ 1]=0.20583 P[ 2]=0.36748
Iterat :  2 :P[ 0]=0.15412 P[ 1]=0.21962 P[ 2]=0.38779
 ...
Iterat : 17 :P[ 0]=0.16832 P[ 1]=0.23986 P[ 2]=0.41762
Iterat : 18 :P[ 0]=0.16833 P[ 1]=0.23987 P[ 2]=0.41763
Iterat : 19 :P[ 0]=0.16833 P[ 1]=0.23987 P[ 2]=0.41763
```

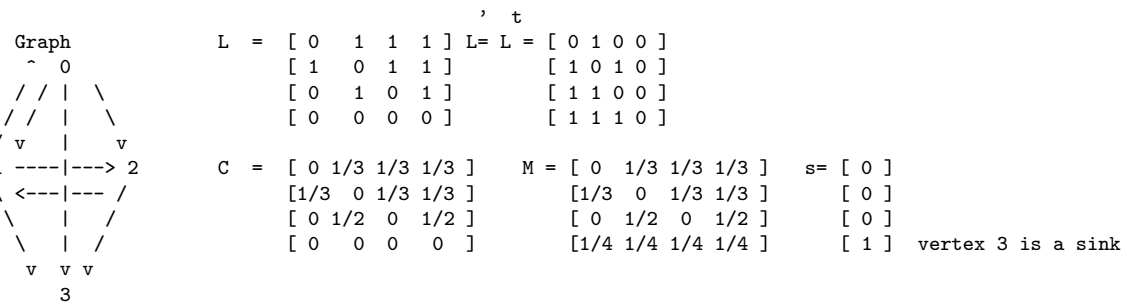



Figure 6: Sinks and their effects

4.1 Example of Figure 6. We use the example of Figure 6 as guidance to the rest of this discussion. We simplify notation and use r_i to denote the rank of page i. Then r is the (column) vector of ranks i.e. an $n \times 1$ matrix. Therefore r' is a row-vector of $1 \times n$ ranks, the transpose of r. L is the adjacency matrix and $L' = L^t$ is the transpose of the adjacency matrix. Moreover C is derived from L if row i is divided by the number of ones of row i (for non sink rows). Thus rows 0 and 1 are divided by 3, row 2 by 2 and row 3 (last row) is a sink. We can then derive from C matrix M (modified C) that for a sink it has $1/n$ on every element of a row. Then $M = C + s(1/n)e'$. In this expression e is an $n \times 1$ vector of ones, and e' its transpose, a $1 \times n$ row vector. Moreover s is an $n \times 1$ vector with entries zeros except for the sinks that have an 1. Thus for our example only the last entry of s is a one. Then the PageRanks of the 4 vertices are computed by Equation 4 as follows.

$$\begin{aligned}
r_0 &= \frac{1-d}{4} + d\left(\frac{1}{3}r_1\right) \\
r_1 &= \frac{1-d}{4} + d\left(\frac{1}{3}r_0 + \frac{1}{2}r_2\right) \\
r_2 &= \frac{1-d}{4} + d\left(\frac{1}{3}r_0 + \frac{1}{3}r_1\right) \\
r_3 &= \frac{1-d}{4} + d\left(\frac{1}{3}r_0 + \frac{1}{3}r_1 + \frac{1}{2}r_2\right)
\end{aligned}$$

4.2 An (eigenvalue) property of matrix M. Matrix $M = C + s(1/n)e'$ has a interesting property: $M \cdot e = 1 \cdot e$, where e is the all-one vector. This is because all rows of M add up to 1. Thus $(e, 1)$ is an eigvenvector and eigenvalue pair of M: we might call it an eigenpair. Moreover 1 is the largest eigenvalue of M.

4.3 Creating the Google matrix G from M or C. Then let

$$G \;=\; dM + \frac{1-d}{n} e\, e' = d(C + s(1/n)e') + \frac{1-d}{n} e\, e' = d \cdot C + (ds + (1-d)e)\frac{1}{n} e'$$

Then for the example in hand,

$$M = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1/3 & 0 & 1/3 & 1/3 \\ 0 & 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix}$$

4.4 The largest eigenvalues of M and G, eigenpair $(e, 1)$, and the second largest eigenvalue. Note that both in M and in G the elements of any row add up to 1. Thus for G the $(e, 1)$ is still an eigenpair and thus 1 is the largest eigenvalue of both. All eigenvalues (other than the largest one which is one) are smaller than 1 (or in general their magnitude is smaller than 1) because the sum of the rows is 1. The difference between G and M is that in G the second largest eigenvalue $\lambda_2(G)$ is guaranteed to be less than d. Thus $\lambda_2(G) \leq d$. Thus the $\mid \lambda_1(G) - \lambda_2(G) \mid = \mid 1 - \lambda_2(G) \mid \leq 0.15$. The distance 0.15 determines how fast the algorithm on the next page converges!

4.4.1 A matrix G and its transpose $G' = G^t$. The two matrices G and G' have the same eigenvalues. This is a fact from linear algebra! So do M and M'. (It is one thing though to have the same eigenvalues and another thing to also have the same eigenvectors...)

4.5 Finding the rank vector r. For a matrix such as G the rank vector r is a probability vector: its i-th element r_i denotes the probability of visiting page i. (Note that the subscript i is not in parentheses!) Moreover,

$$r_1 + r_2 + \ldots + r_n = \sum r_i = 1.$$

By linear algebra/probability theory(Markov chains) r is the solution of $r' = r'G$. That is, r is the dominant "left" eigenvector of G i.e. the "left" eigenvector that corresponds to the largest eigenvalue of it. In $1 \cdot r' = r'G$, the r' is on the "left" of G, that's why it is a "left eigenvector"; the left hand side implies an eigenavalue of 1, as noted. Since this is the largest eigenvalue, it is the dominant or principal eigenvalue and its eigenvector is the dominant or principal eigenvector as well.) By 4.4 whether we talk about G or G' the largest eigenvalue of either is equal to 1. If we transpose both sides of $r' = r'G$ we get that $r = G'r$. That is the solution r is the "right eigenvector" of G' as it gets multiplied with it from the right. Likewise its corresponding eigenvalue is 1. We drop the right from "right eigenvector" to just say eigenvector. Given that G and G' have the same eigenvalues (Item 4.4) this does not affect their eigenvalues. In conclusion, to find the dominant "left" eigenvector of G we find the eigenvector corresponding to the dominant eigenvalue (1) of G'.

4.5.1 How do we find r? Power Method, i.e. iterative convergence. How can we find r? We can use iterative methods such as the **power method** where

$$r'_{(i)} = r'_{(i-1)}G$$

The subscript indicates the iteration. That is why it is in parentheses. The solution at convergence will have an i-th element whose value would be r_i or an approximation to r_i. We move from iteration $i-1$ to i by performing a multiplication with G as indicated above. The process stops when at some iteration k we have convergence i.e. the entries of the r vector remain the same to a given accuracy, i.e. $\mid r_{(k)} - r \mid <$ `error` or given that we do not know r, when $\mid r_{(k)} - r_{(k-1)} \mid <$ `error`. The Google authors, Brin and Page, claimed that $k = 50$ works for most cases.

4.5.2 Initial value. For the iterative process $r'_{(i)} = r'_{(i-1)}G$ what is $r_{(0)}$ i.e. the initial or boundary value? Any vector $r_{(0)}$ is OK as long as it is orthogonal to the solution r! For example the unit vector e or e/n could work. A requirement for convergence implies the need for scaling! This would require the use of e/n over the unscaled e. A $e/\sqrt{n}$ will also work! Never try however an initial vector of zeroes!

4.6 Why does it work? Linear algebra and $(e, 1)$ eigenpair of G.

Eigenvalue 1 is the largest eigenvalue of non-negative matrix M (or G or M' or G'). Every other eigenvalue is smaller. Therefore $\lambda_1 = 1 > \lambda_2 > \ldots$. Let the corresponding eigenvectors be $x_1, \ldots, x_n$. From matrix theory we have that $G'x = \lambda x$ for eigenpair (x, λ). Let us call in the remainder $A = G'$ to avoid problems with transposes. Thus $Ax = \lambda x$ given that A is an alias for G'. Therefore $A^2x = A(Ax) = A\lambda x = \lambda(Ax) = \lambda^2 x$ and in general $A^k x = \lambda^k x$. Let the starting vector $r_{(0)}$ be a linear combination of the eigenvectors i.e. $r_{(0)} = x_1 + a_1x_2 + \ldots + a_nx_n$. Then using the fact that $A^k x_i = \lambda_i^k x_i$ we obtain the following.

$$\begin{aligned} r_{(0)} &= x_1 + a_1x_2 + \ldots + a_nx_n \\ r_{(1)} = Ar_{(0)} &= Ax_1 + Aa_2x_2 + \ldots + Aa_nx_n \\ &= 1x_1 + \ldots + a_n\lambda_n x_n \\ r_{(k)} = Ar_{(k-1)} &= 1x_1 + a_2\lambda_2^k x_2 + \ldots + a_n\lambda_n^k x_n \end{aligned} \tag{5}$$

Therefore

$$r_{(k)} = x_1\left(1 + a_2(\lambda_2/1)^k x_2 + \ldots + a_n(\lambda_n/1)^k x_n\right) \tag{6}$$

From the latter we obtain that $r_{(k)}$ converges to x_1 as long as $\lambda_2^k, \ldots, \lambda_n^k \to 0$. Given that $\lambda_1 = 1 > \lambda_2 > \ldots > \lambda_n$, this is indeed the case. Moreover, the rate of convergence depends on λ_2^k.

4.6.1 Rate of convergence. The reate of converge is λ_2^k after k iterations. Given that $\lambda_2(G) = \lambda_2(G') \leq d = 0.85$, this implies indeed that the rate of convergence is 0.85^k. The "Given" part of the claim of the previous paragraph has not been proven yet! Its proof appears on the following page(s) under section 4.7. (Note that $0.85^{50} \approx 0.00029$ and $0.85^{60} \approx 0.000058$.)

Properties of the G matrix

4.7 Theorem. If matrix M has eigenvalues $\langle 1 \geq \lambda_2 \geq \ldots \geq \lambda_n \rangle$, then matrix G and also G' has eigenvalues $\langle 1 \geq d\lambda_2 \geq \ldots \geq d\lambda_n \rangle$. The two matrices G, M are defined as follows: $G = dM + \frac{1-d}{n} e\, e^t$.
(Note we use in the remainder for clarity t instead of $'$ for the transpose operation.)

Proof. Consider $G = ((1-d)/n)ee^t + dM$. For M, an eigenpair is $(e, 1)$ i.e. $Me = e$.

Consider matrix Q formed by the eigenvectors of M listed so that e appears first, i.e. $Q = (e \quad X)$ then $Q^{-1} = \begin{pmatrix} z^t \\ Z^t \end{pmatrix}$.

Since $Q^{-1}Q = I$ we obtain that $Q^{-1}Q = \begin{pmatrix} z^te & z^tX \\ Z^te & Z^tX \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & I \end{pmatrix}$. From the last equality we conclude that

$$z^tX = 0, \quad z^te = 1 \quad , \quad Z^te = 0, \quad Z^tX = I \tag{7}$$

Consider now the product $Q^{-1}MQ$. We have

$$Q^{-1}MQ \;=\; \begin{pmatrix} z^t \\ Z^t \end{pmatrix} \cdot \begin{pmatrix} Me & MX \end{pmatrix} = \begin{pmatrix} z^tMe & z^tMX \\ Z^tMe & Z^tMX \end{pmatrix} = \begin{pmatrix} 1 & z^tMX \\ 0 & Z^tMX \end{pmatrix} \tag{8}$$

The last equality was derived because $Me = e$ implies $z^tMe = z^te = 1$ and the fact that $z^te = 1$ by Eq. (7). In addition $Z^tMe = 0$ by $Z^te = 0$ again from Eq. (7).

$$\begin{aligned} Q^{-1}GQ = Q^{-1}(dM + ((1-d)/n)ee^t)Q &= dQ^{-1}MQ + (1-d)/nQ^{-1}ee^tQ & (9) \\ &= \begin{pmatrix} d & dz^tMX \\ 0 & dZ^tMX \end{pmatrix} + \frac{(1-d)}{N} \cdot \begin{pmatrix} z^te \\ Z^te \end{pmatrix} \cdot \begin{pmatrix} e^te & e^tX \end{pmatrix}. & (10) \end{aligned}$$

The last two matrices are the expanded form of $Q^{-1}e$ and e^tQ respectively. Therefore we get after observing again from Eq. (7) that $z^te = 1$, $Z^te = 0$.

$$\begin{aligned} Q^{-1}GQ &= \begin{pmatrix} d & dz^tMX \\ 0 & dZ^tMX \end{pmatrix} + \frac{(1-d)}{N} \cdot \begin{pmatrix} z^te(e^te) & z^te(e^tX) \\ Z^te(e^te) & Z^te(e^tX) \end{pmatrix} \\ &= \begin{pmatrix} d & dz^tMX \\ 0 & dZ^tMX \end{pmatrix} + \frac{(1-d)}{N} \cdot \begin{pmatrix} (e^te) & (e^tX) \\ 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} d + ((1-d)/N) * N & dz^tMX + ((1-d)/N)e^tX \\ 0 & dZ^tMX \end{pmatrix} \\ &= \begin{pmatrix} 1 & dz^tMX + ((1-d)/N)e^tX \\ 0 & dZ^tMX \end{pmatrix} \end{aligned} \tag{11}$$

Since $Z^tX = I$ from Eq. (7), and from the rightmost side of Eq. (8) we have Z^tMX has eigenvalues the vector $(\lambda_2, \ldots, \lambda_n)$, and thus we can conclude that dZ^tMX will have as eigenvalues the d multiples of them, i.e. $(d\lambda_2, \ldots, d\lambda_n)$.

Exercise 1 *You are given the graph below. Apply each one of the ranking algorithms till convergence and find the ranks/authority values/ hub values for each one of the nodes $A, \ldots, E$. Use the following algorithms. Use appropriate initial conditions.*

- *Kleinberg's hub and authority algorithm.*
- *PageRank with $d = 0.85$. Initial condition $1/N$ synchronous.*
- *PageRank with $d = 0.85$. Initial condition $1/N$ asynchronous.*

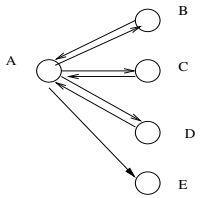


Figure 7: Document Processing