4. Graph as Matrix : PageRank, Random Walks and Embeddings

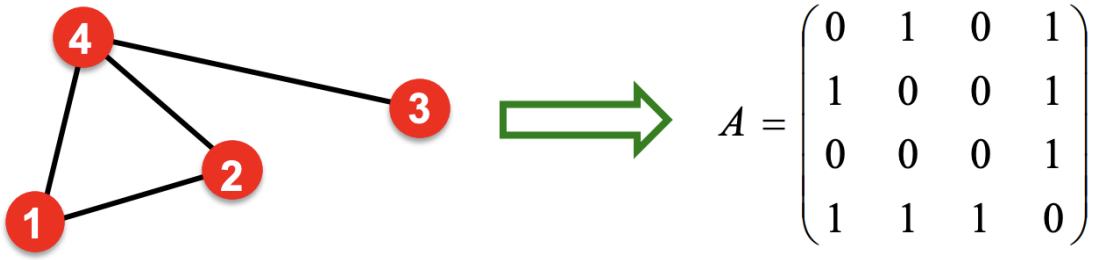
https://www.youtube.com/watch?v=TU0ankRcHmo&t=5s

Contents

Graph as Matrix

In this lecture, we investigate graph analysis and learning from a matrix perspective.

- Treating a graph as a matrix allows us to:
  - Determine node importance via **random walk**(PageRank)
  - Obtain node embeddings via **matrix factorization(MF)**
  - view other **node embeddings** (e.g.Node2Vec) as MF
- **Random walk, matrix factorization and node embeddings are closely related!**



1. PageRank(aka the Google Algorithm)

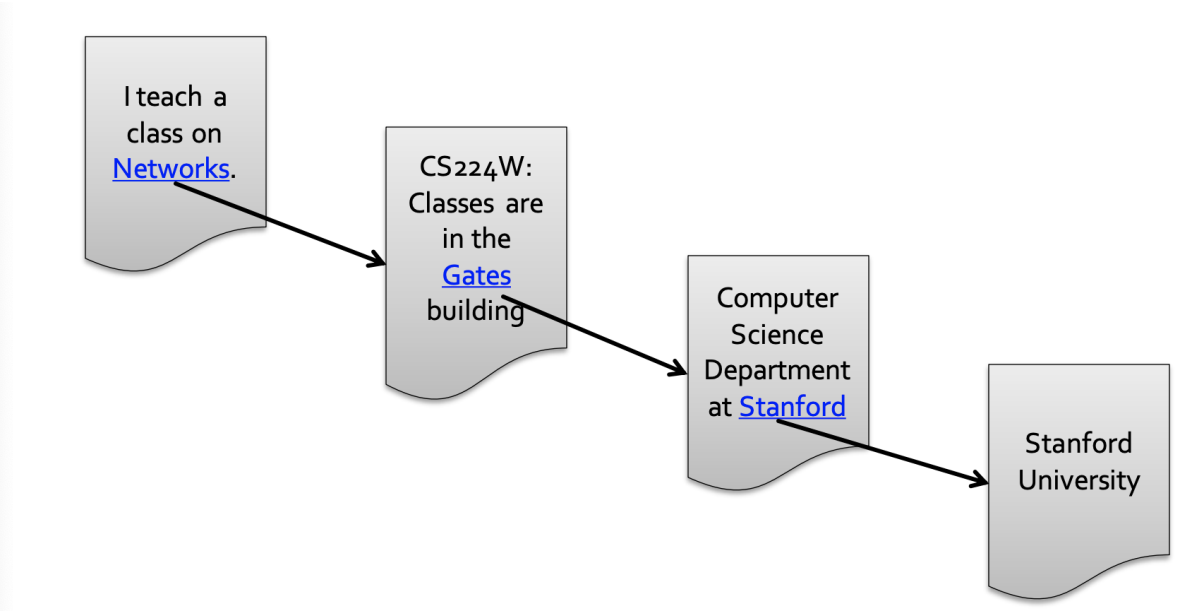
Example : The Web as a Graph

Q : What does the Web “look like” at a global level?

- Web as a graph:
  - Nodes = web pages
  - Edges = hyperlinks

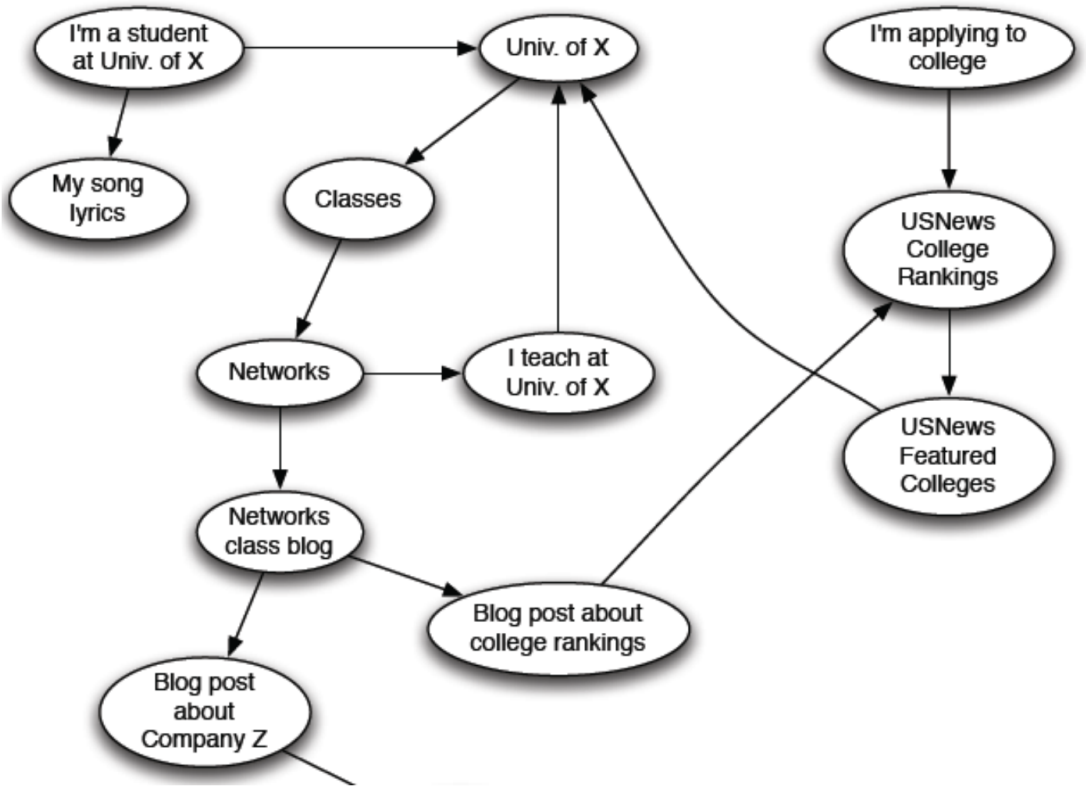
  - Side issue : What is a node?
    - Dynamic pages created on the fly
    - “dark matter” - inaccessible database generated pages

The Web as a Graph

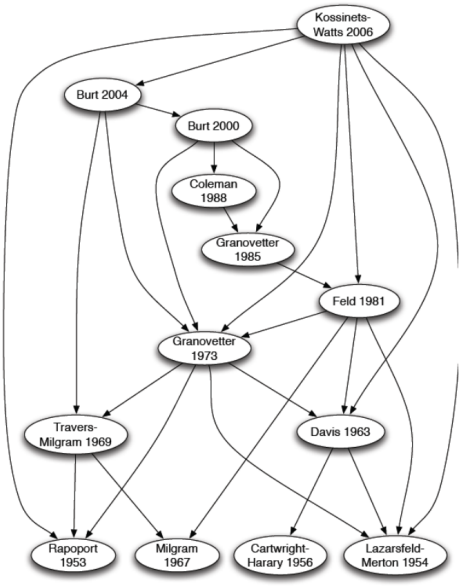


- In early days of the Web links were **navigational**
- Today many links are **transactional(used not to navigate from page to page, but to post, comment, like, buy,...)**

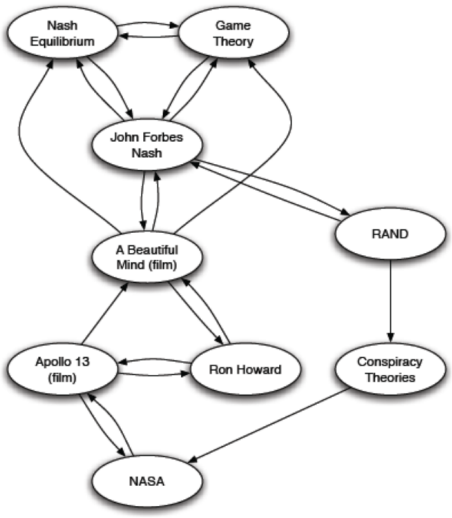
The Web as a Directed Graph



Other Information Networks



Citations



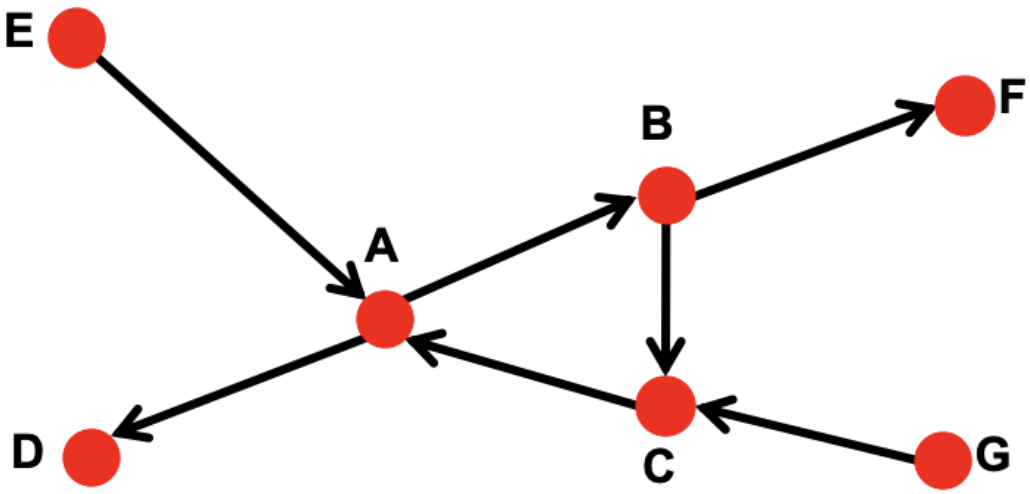
References in an Encyclopedia

What does the Web Look like?

- **How is the Web linked?**
- **What is the “map” of the Web?**

Web as a directed graph [Broder et al. 2000]:

- Given node ***v***, what nodes can ***v*** reach?
- What other nodes can reach ***v***?



Ranking Nodes on the Graph

- All web pages are not equally “important”
- There is large diversity in the web-graph node connectivity.
- **So, let’s rank the pages using the web graph link structure!**

Link Analysis Algorithms

- We will cover the following **Link Analysis approaches** to compute the **importance** of **nodes** in a graph:
  - PageRank
  - Personalized PageRank (PPR)
  - Random Walk with Restarts

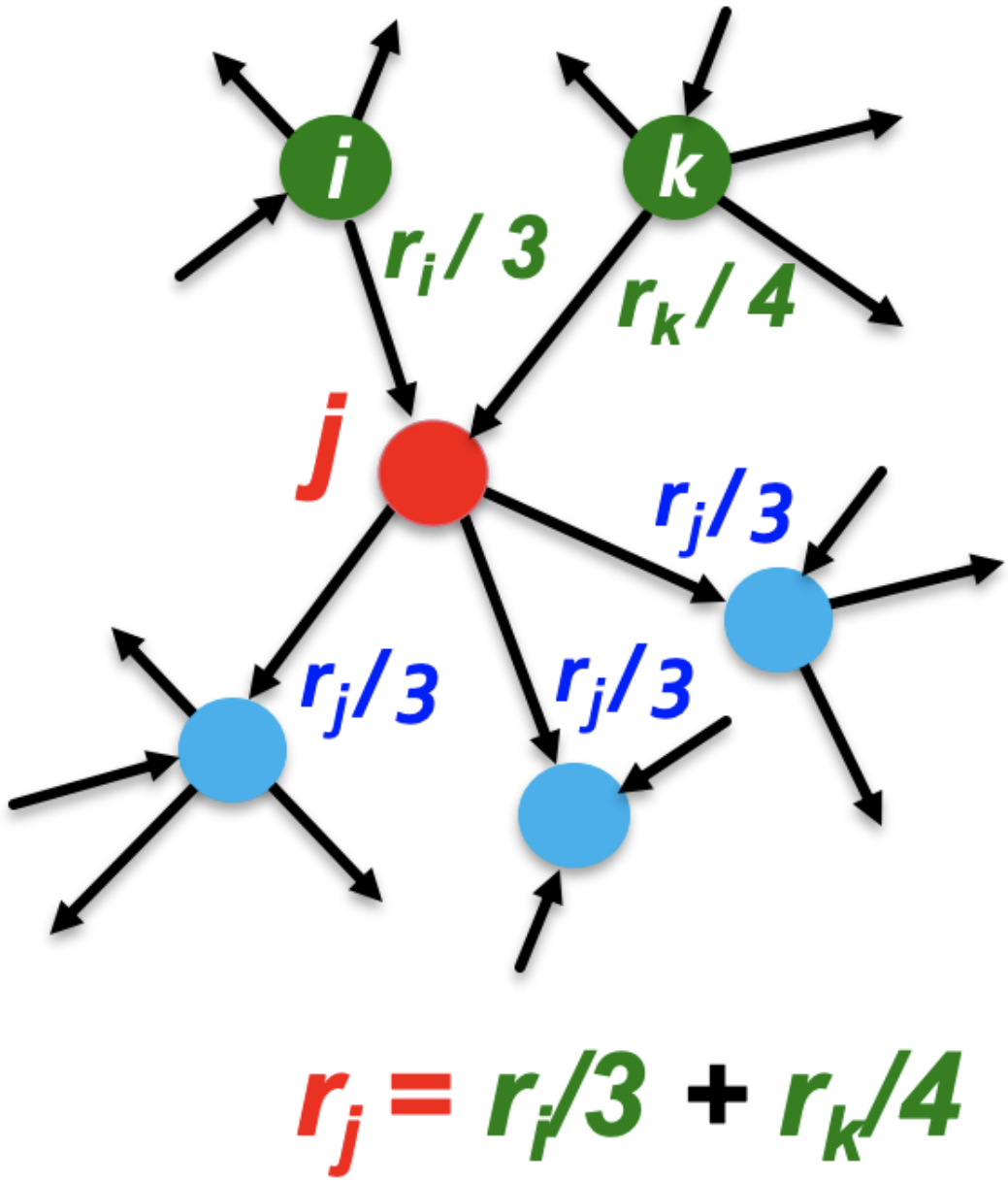
Links as Votes

- **Idea : Links as votes**
  - **Page is more important if it has more links**
    - In-coming links? Out-going links?
- **Think of in-links as votes:**
  - www.standford.edu has 23,400 in-links
  - thispersondoesnotexist.com has 1 in-link
- **Are all in-links equal?**
  - Links from important pages count more
  - Recursive question!

PageRank : The “Flow” model

- **A “Vote” from an important page is worth more:**
  - Each link’s vote is proportional to the **importance** of its source page
  - If page ***i*** with importance ***ri*** has ***di*** out-links, each link gets ***ri/di*** votes

- Page *j*'s own importance *rj* is the sum of the votes on its in-links



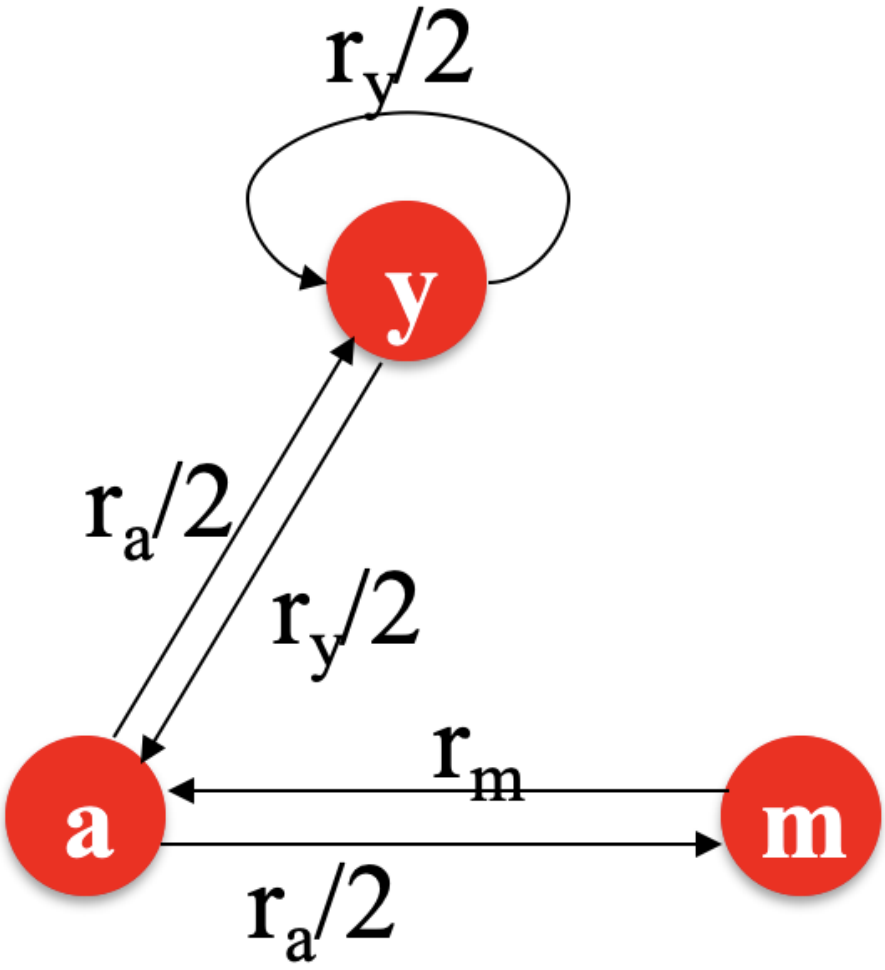
- A **page is important if it is pointed to by other important pages**

- **Define "rank" r_j for node j**

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

d_i … out-degree of node i

The web in 1839



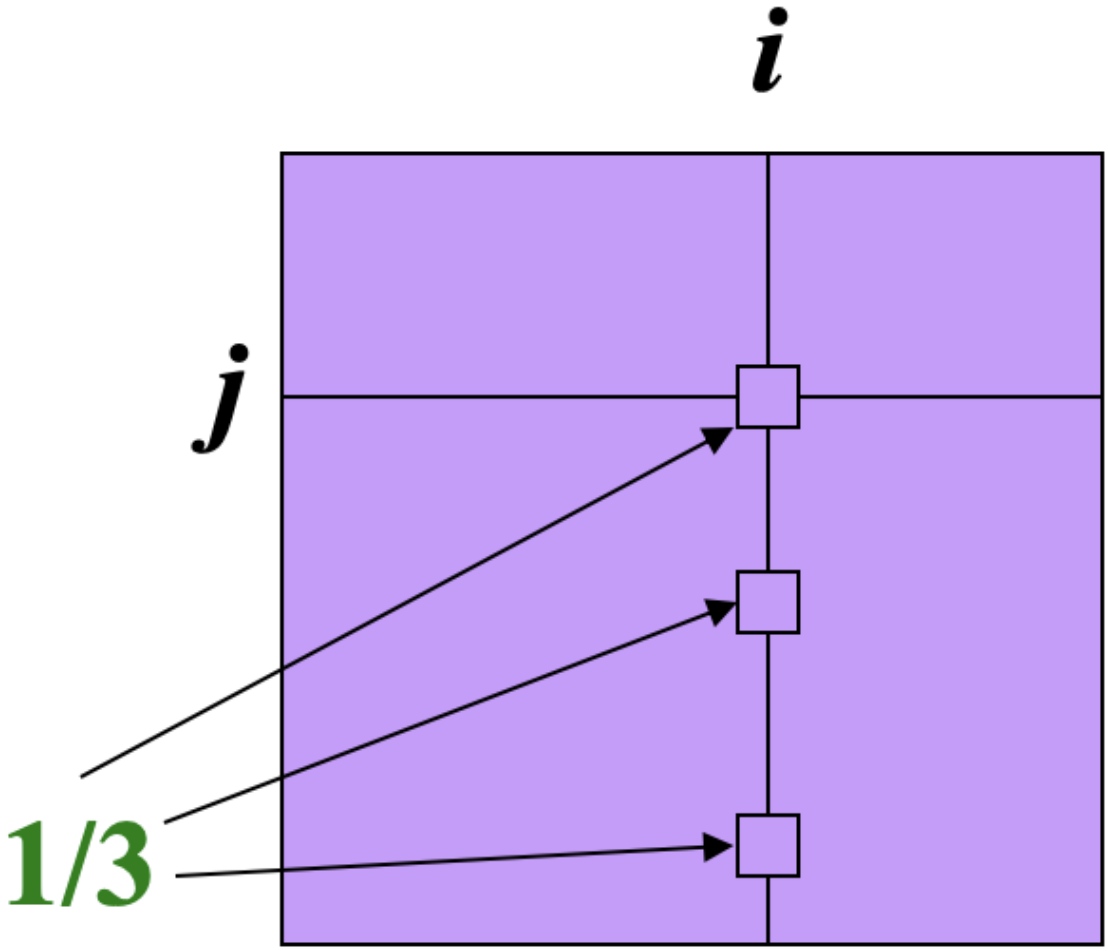
"Flow" equations:

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

You might wonder : Let's just use Gaussian elimination to solve this system of linear equations. Bad idea!

PageRank: Matrix Formulation

- **Stochastic adjacency matrix *M***
  - d_i is the outdegree of node i
  - If $i \rightarrow j$, then $M_{ji} = \frac{1}{d_i}$
    - M is a **column stochastic matrix**
      - **Columns** sum to **1**

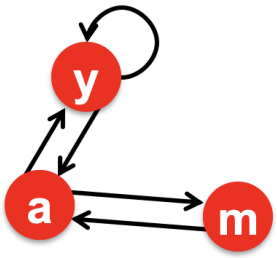


- Rank vector r

- **Rank vector r:** An entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$

- The flow equations can be written > r = M * r

Example : Flow Equations & M



	r_y	r_a	r_m
r_y	½	½	0
r_a	½	0	1
r_m	0	½	0

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$
$$r \qquad M \qquad r$$

- **In-link가 많을수록 중요도가 높으며** in-link가 같을 경우 중요한 페이지로부터 들어온 link가 많은지로 중요도를 판단하여 **recursive**한 문제가 된다.
- In-link는 vote로 볼 수 있으며 page *i*의 중요도가 *ri*이고 out-links가 *di*개면 각 링크들은 ***rildi*의 votes**를 가지게 된다.
- Page *j*의 ***rj*는 모든 votes의 합**이 된다.

Connection to Random Walk

- Imagine a random web surfer:
  - At any time ***t***, surfer is on some page *i*
  - At time ***t + 1***, the surfer follows an out-link from ***i*** uniformly at random
  - Ends up on some page ***j*** linked from ***i***
  - Process repeats indefinitely

- **Let:**
  - **P(t)** ... vector whose ***i*** th coordinate is the prob. that the surfer is at page *i* at time *t*

- **p(t)는 균일한 확률로 random walk하는 surfer가 t시점에 page i에 있을 확률이 i번째 원소에 담긴 벡터이다.**
- So, ***P(t)*** is a probability distribution over pages

The Stationary Distribution

- **Where is the surfer at time t+1?**
  - Follow a link uniformly at random

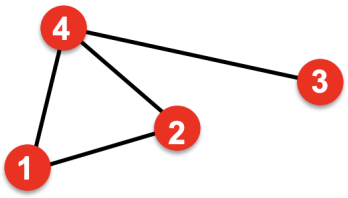
    P(t+1) = M * P(t)
- Suppose the random walk reaches a **state**

  P(t+1) = M * P(t) = P(t)

  then p(t) is **stationary distribution** of a random walk
- **Our original rank vector *r*** statisfies **r = M * r**
  - **So, r is a stationary distribution for the random walk**

Recall Eigenvector of A Matrix

- Recall from lecture 2 (eigenvector centrality), let $\boldsymbol{A \in \{0,1\}^{n \times n}}$ be an adj. matrix of undir. graph:



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

- Eigenvector of adjacency matrix: vectors satisfying $\boldsymbol{\lambda c = Ac}$
- $\boldsymbol{c}$: eigenvector; $\boldsymbol{\lambda}$: eigenvalue

- Note :
  - This is the definition of eigenvector centrality (for undirected graphs)
  - PageRank is defined for directed graphs

Eigenvector Formulation

- **The flow equation:**

$$1 \cdot \boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

$\boldsymbol{r}$ $\quad$ $\boldsymbol{M}$ $\quad$ $\boldsymbol{r}$

- So the **rank vector** $\boldsymbol{r}$ is an **eigenvector** of the stochastic adj. matrix $\boldsymbol{M}$ (with eigenvalue 1)
  - Starting from any vector $\boldsymbol{u}$, the limit $\boldsymbol{M}(\boldsymbol{M}(\ldots \boldsymbol{M}(\boldsymbol{M}\,\boldsymbol{u})))$ is the **long-term distribution** of the surfers.
    - **PageRank** = Limiting distribution = **principal eigenvector** of M
    - Note: If $\boldsymbol{r}$ is the limit of the product $\boldsymbol{MM} \ldots \boldsymbol{Mu}$, then $\boldsymbol{r}$ satisfies the **flow equation** $1 \cdot \boldsymbol{r} = \boldsymbol{Mr}$
    - So $\boldsymbol{r}$ is the **principal eigenvector** of $\boldsymbol{M}$ with eigenvalue 1
- **We can now efficiently solve for *r*!**
  - The method is called **Power iteration**

PageRank : Summary

- PageRank :
  - Measures importance of nodes in a graph using the link structure of the web
  - Models a random web surfer using the **stochastic adjacency Matrix M**
  - PageRank solves ***r = Mr*** where ***r*** can be viewed as both the **principle eigenvector** of ***M*** and as **the stationary distribution of a random walk** over the graph

A stationary distribution is such a distribution π that if the distribution over states at step k is π, then also the distribution over states at step $k+1$ is π. That is,

$$\pi = \pi P.$$

A limiting distribution is such a distribution π that no matter what the initial distribution is, the distribution over states converges to π as the number of steps goes to infinity:

$$\lim_{k\to\infty} \pi^{(0)} P^k = \pi,$$

---

2. PageRank : How to solve?

Given a graph with *n* nodes, we use an iterative procedure:

- Assign each node an initial page rank
- Repeat until convergence

$$\left(\sum_i \left| r_i^{t+1} - r_i^t \right| < \epsilon\right)$$

calculate the page rank of each node

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

d_i …. out-degree of node i

Power Iteration Method

- **Given a web graph with N nodes, where the nodes are pages and edges are hyperlinks**
- **Power iteration** : a simple iterative scheme
  - Initialize: $\boldsymbol{r}^{(0)} = [1/N, \ldots, 1/N]^T$
  - Iterate: $\boldsymbol{r}^{(t+1)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(t)}$
  - Stop when $|\boldsymbol{r}^{(t+1)} - \boldsymbol{r}^{(t)}|_1 < \varepsilon$

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

d_i …. out-degree of node i

$|\boldsymbol{x}|_1 = \sum_1^N |\boldsymbol{x}_i|$ is the L1 norm
Can use any other vector norm, e.g., Euclidean

About 50 iterations is sufficient to estimate the limiting solution.

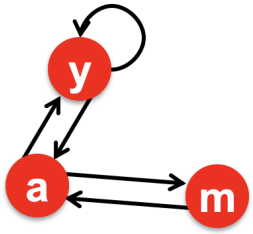
Power iteration - Wikipedia

In mathematics, power iteration (also known as the power method) is an eigenvalue algorithm: given a diagonalizable matrix , the algorithm will produce a number , which is the greatest (in absolute value) eigenvalue of , and a nonzero vector , which is a corresponding eigenvector of , that is, .

𝕎 https://en.wikipedia.org/wiki/Power_iteration

PageRank : How to solve?

- **Power Iteration:**
  - Set $r_j \leftarrow 1/N$
  - **1:** $r'_j \leftarrow \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** If $|r - r'| > \varepsilon$:
    - $r \leftarrow r'$
  - **3:** go to **1**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

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

Iteration 0, 1, 2, …

PageRank : Three Questions

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i} \quad \text{or equivalently} \quad r = Mr$$

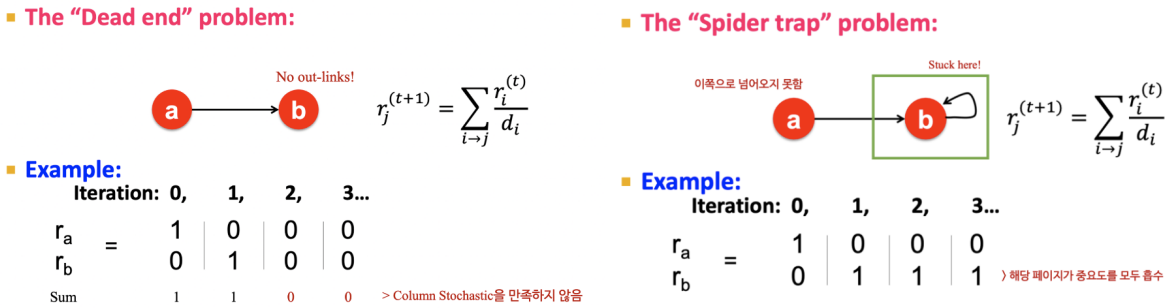
- **Does this converge?**
- **Does it converge to what we want?**
- **Are results reasonable?**

PageRank : Problems

Two problems:

1. Some pages are **dead ends** (have no out-links)fggh ng
   a. Such pages cause importance to “leak out”
2. **Spider traps** (all out-links are within the group)
   a. Eventually spider traps absorb all importance

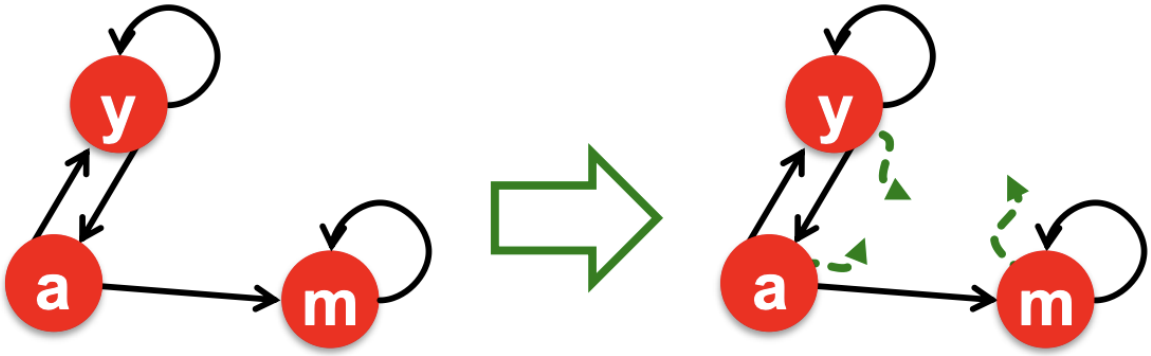
Does this converge? Does it converge to what we want?



- **Dead ends:** out-link가 없는 경우 *M*이 column stochastic하지 않아(column의 합이 1이 아닌) **중요도에 leakage**가 발생한다.
- **Spider traps:** 모든 out-links가 그룹 안에 속할 때 한 번 들어가면 갇히는 현상이 발생해 해당 page가 **중요도를 흡수**한다.

Solution to Spider Traps

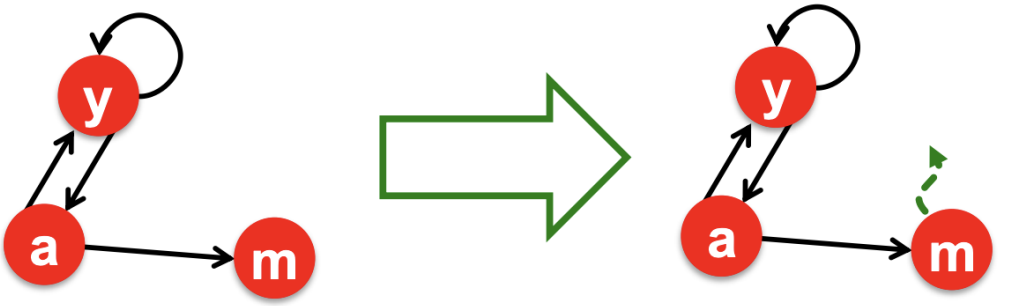
- Solution for spider traps : **At each time step, the random surfer has two options**
  - With prob. **β, follow a link at random**
  - With prob. 1-**β, jump to a random page**
  - Common values for **β are in the range 0.8 to 0.9**
- **Surfer will teleport out of spider trap within a few time steps**



random teleport를 활용하여 **β의 확률로 기존의 link를 타고 이동하거나 (1-β)의 확률로 연결되지 않았던 페이지로 이동한다.**

Solution to Dead Ends

- Teleports : Follow random teleport links with total probability **1.0** from dead-ends
  - Adjust matrix accordingly



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

확률을 배분하여 column stochastic을 만족시킨다.

Why Teleports Solve the Problem?

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- **Spider-traps** are not a problem, but with traps PageRank scores are **not** what we want
  - **Solution** : Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** are a problem
  - The matrix is **not column stochastic** so our initial assumpotins are not met
  - **Solution** : Make matrix column stochastic by always teleporting when there is nowhere else to go

Solution : Random Teleports

- **Google's solution that does it all:**
  At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1\text{-}\beta$, jump to some random page
- **PageRank equation** [Brin-Page, 98]

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

d_i … out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends or explicitly follow random teleport links with

probability 1.0 from dead-ends

The Google Matrix

- **PageRank equation** [Brin-Page, '98]

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

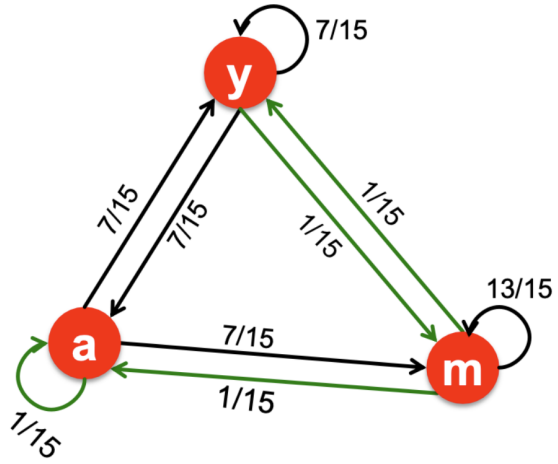
- **The Google Matrix *G*:**

$[1/N]_{N\times N}$…N by N matrix where all entries are 1/N

$$G = \beta\, M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$

- **We have a recursive problem: $r = G \cdot r$**
  And the Power method still works!
- **What is β?**
  - In practice $\beta = 0.8, 0.9$ (make 5 steps on avg., jump)

Random Teleport (β = 0.8)



$$M \qquad [1/N]_{NxN}$$

$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$\begin{matrix} y \\ a \\ m \end{matrix} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

$$G$$

$$\begin{matrix} y \\ a \\ m \end{matrix} = \begin{matrix} 1/3 & 0.33 & 0.24 & 0.26 & \\ 1/3 & 0.20 & 0.20 & 0.18 & \dots \\ 1/3 & 0.46 & 0.52 & 0.56 & \end{matrix} \begin{matrix} \\ \\ \end{matrix} \begin{matrix} 7/33 \\ 5/33 \\ 21/33 \end{matrix}$$

PageRank Example

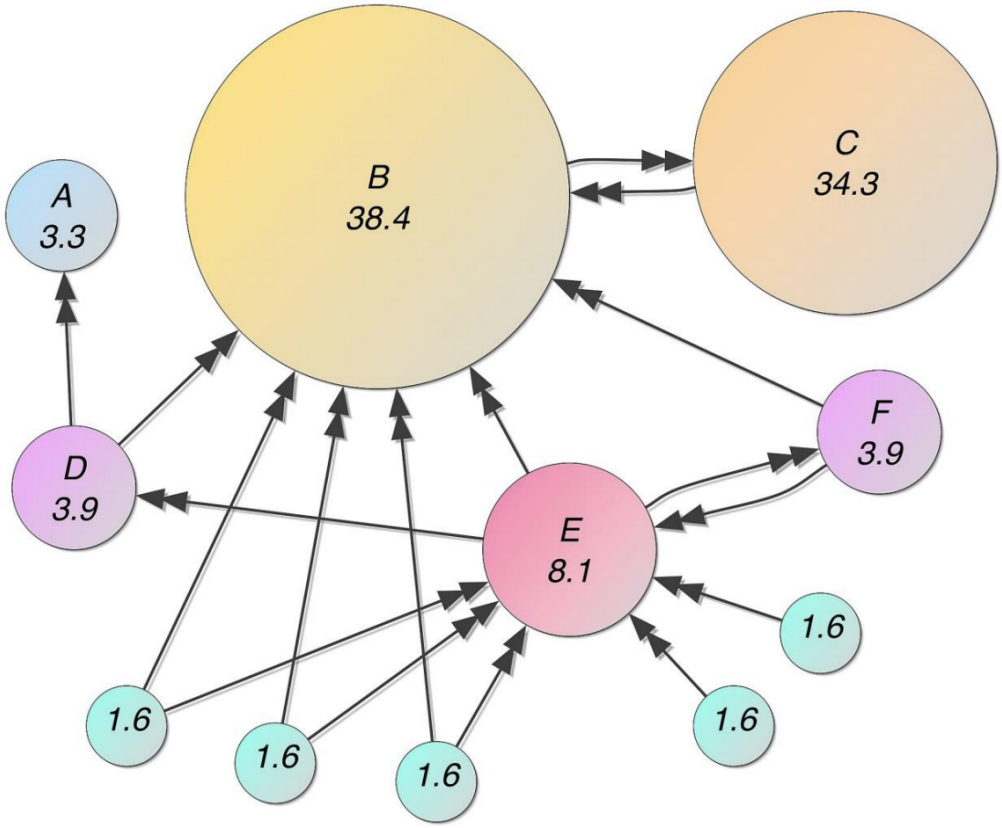


Image credit: Wikipedia

Solving PageRank : Summary

- PageRank solves for ***r* = *Gr*** and can be efficiently compluted by **power iteration of the stochastic adjacency matrix (G)**
- Adding random uniform teleportation solves issues of dead-ends and spider-traps

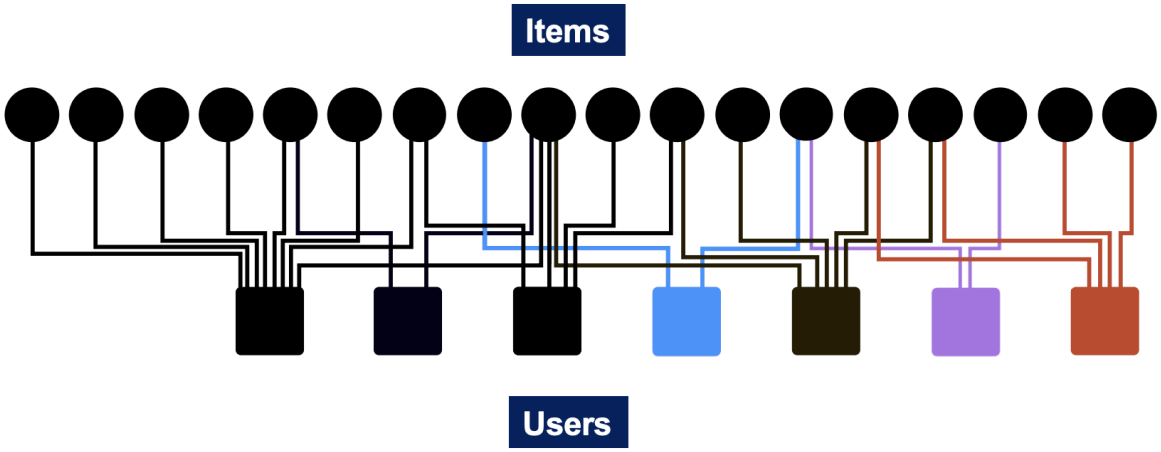
---

3. Random Walk with Restarts and Personalized PageRank

Example : Recommendation

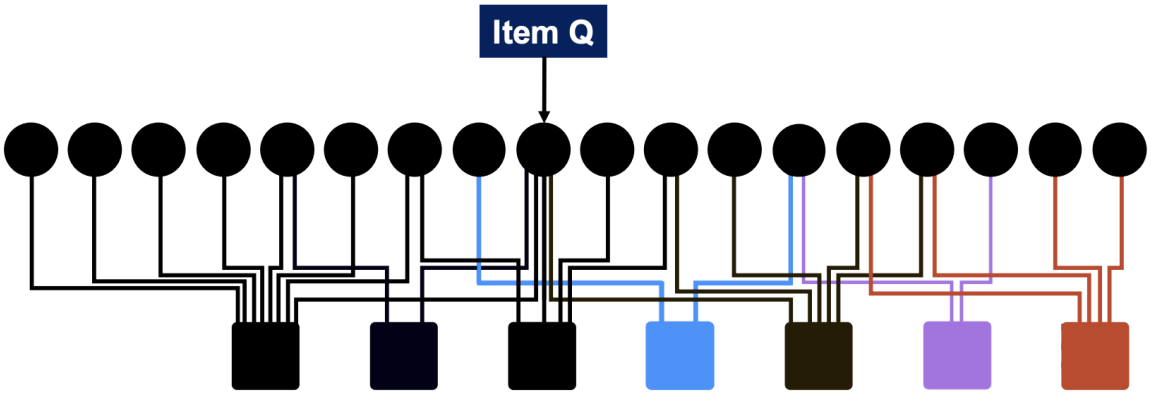
- Given:

  A bipartite graph representing user and item interactions(e.g. purchase)



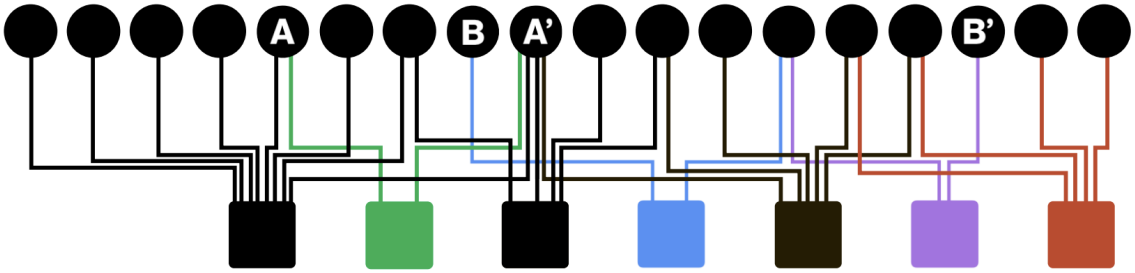
Bipartite User-Item Graph

- **Goal : Proximity on graphs**
  - **What items should we recommend to a user who interacts with item Q?**
  - **Intuition** : if items Q and P are interacted by similar users, recommend P when user interacts with Q



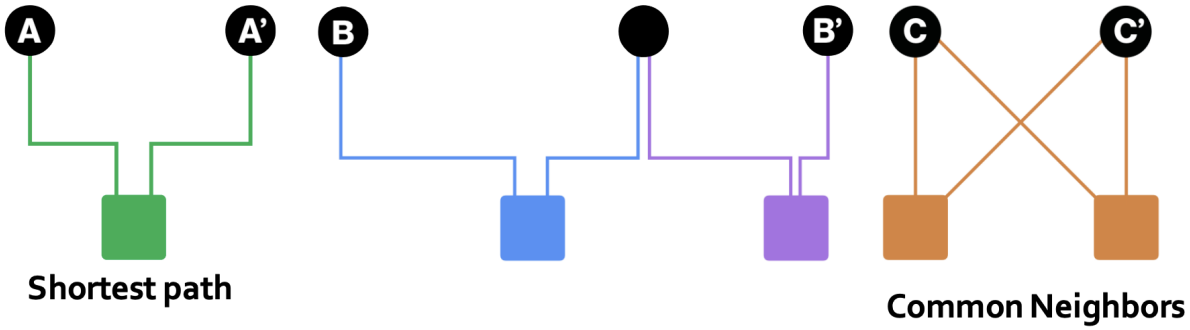
Bipartite User-to-Item Graph

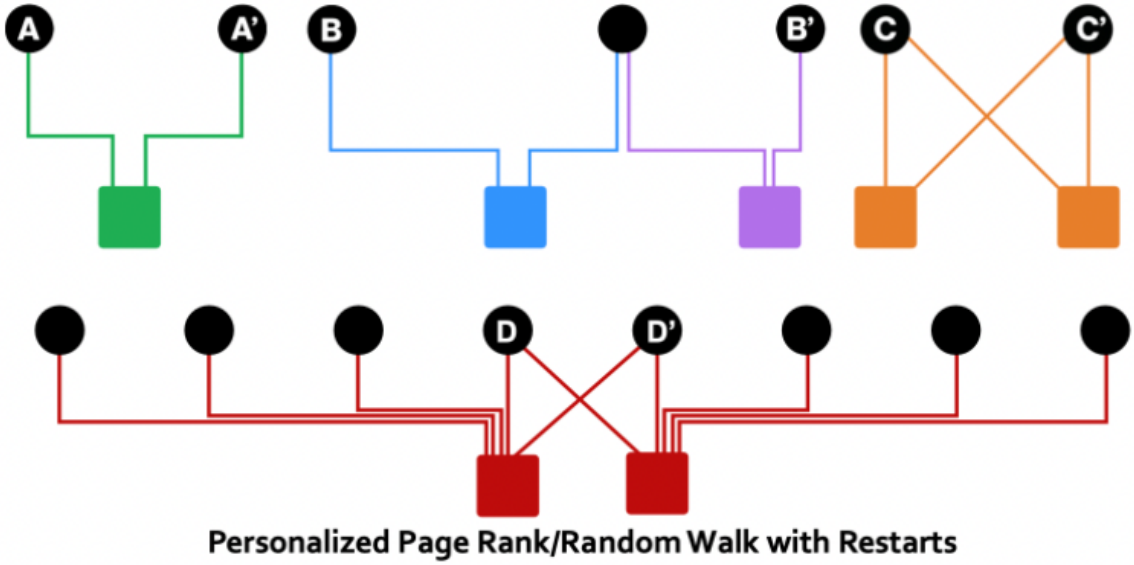
- **Which is more related A, A' or B, B'?**



Node proximity Measurements

- Which is more related A, A', B, B' or C,C'?





- 위 그래프는 사용자(사각형)와 아이템(원)으로 이루어진 **biparatite graph**이다.

- (C,C'),(A,A'),(B,B') 순으로 연관성이 큼을 알 수 있고 (D,D')는 사용자가 너무 많은 아이템을 구매하여 우연성이 있기 때문에 (C,C')만큼 연관성이 크다고 보기는 어렵다.
- 이를 표현하기 위해 중요도를 기반으로 rank를 설정하고 전체 노드 간에 랜덤하게 텔레포트를 하는 PageRank와 다른 **Personalized PageRank**와 **Random Walks with Restarts**가 있다.

Proximity on Graphs

- PageRank:
  - Ranks nodes by “importance”
  - Teleports with uniform probability to any node in the network

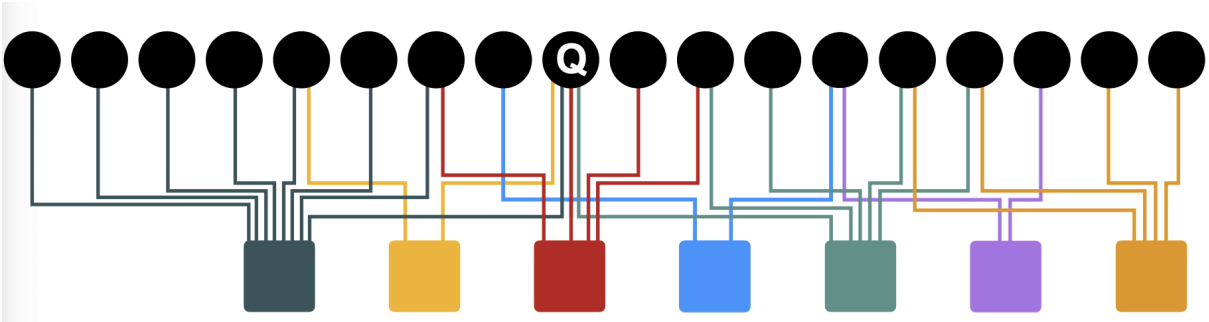
- Personalized PageRank:
  - Ranks proximity of nodes to the teleport node ***S***

- Proximity on graphs:
  - Q : What is most related item to **Item Q**?
  - Random Walks with Restarts
    - Teleport back to the starting node: **S = {Q}**

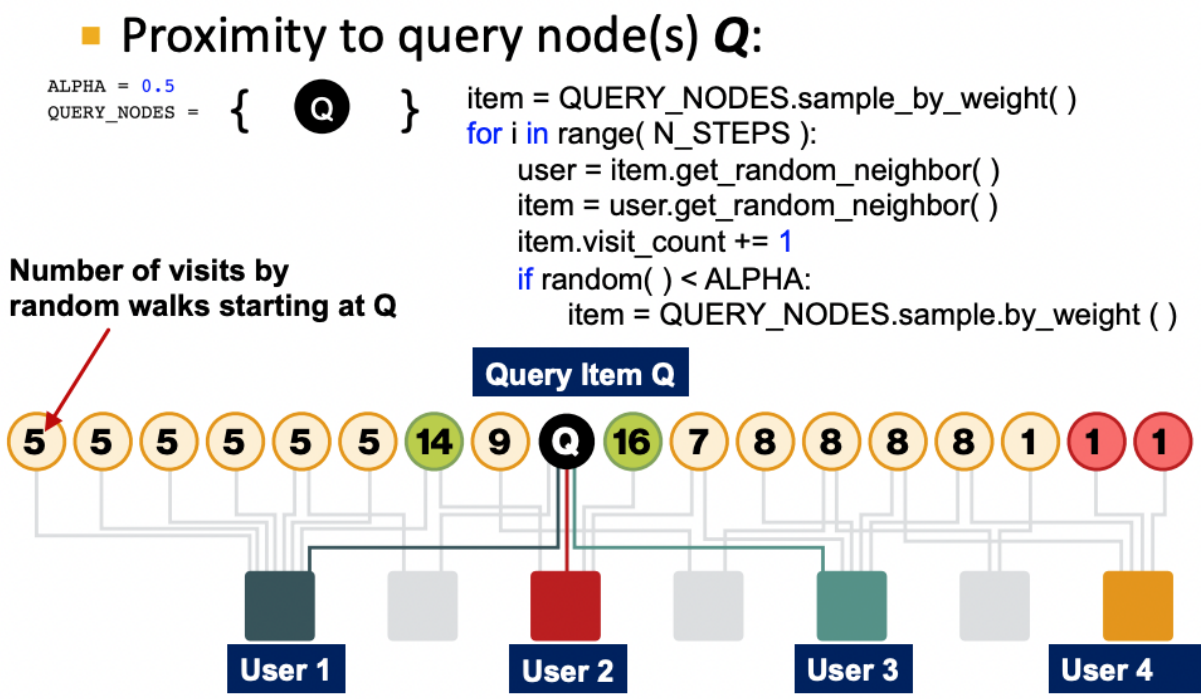
Idea : Random Walks

- **Idea**
  - Every node has some importance
  - Importance gets evenly split among all edges and pushed to the neighbors:
- **Given a set of QUERY_NODES, we simulate a random walk:**
  - Make a step to a random neighbor and record the visit (visit count)
  - With probability ALPHA, restart the walk at one of the QUERY_NODES
  - The nodes with the highest visit count have highest proximity to the QUERY_NODES

- Idea :
  - Every node has some importance
  - Importance gets evenly split among all edges and pushed to the neighbors
- Given a set of **QUERY NODES Q**, simulate a random walk:



Random Walk Algorithm



- 랜덤워크를 **재시작할 노드에만 확률값 1**을 부여한다.[*T*1*j*,*T*2*j*,...,*TNj*]= [0,0,0,0,0,1,0,0,0]
- **ALPHA**는 랜덤워크를 **재시작할 확률**이다.
- 시작 노드와 특정 노드 간에 엣지가 많거나, 경로가 많거나, 직접 연결되어 있거나, degree가 낮은 사용자로 연결되어 있다면 랜덤워크에서 많이 방문하게 되고 유사도가 높아진다.

Benefits

- Why is this a good solution?
- Because the **“Similarity”** considers:
  - Multiple connections
  - Multiple paths
  - Direct and indirect connections
  - Degree of the node

Summary : Page Rank Variants

- **PageRank:**
  - Teleports to any node
  - Nodes can have the same probability of the surfer landing:
    $$\boldsymbol{S} = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]$$
- **Topic-Specific PageRank** aka **Personalized PageRank:**
  - Teleports to a specific set of nodes
  - Nodes can have different probabilities of the surfer landing there:
    $$\boldsymbol{S} = [0.1, 0, 0, 0.2, 0, 0, 0.5, 0, 0, 0.2]$$
- **Random Walk with Restarts:**
  - Topic-Specific PageRank where teleport is always to the same node:
    $$\boldsymbol{S} = [0, 0, 0, 0, \mathbf{1}, 0, 0, 0, 0, 0, 0]$$

Sumary

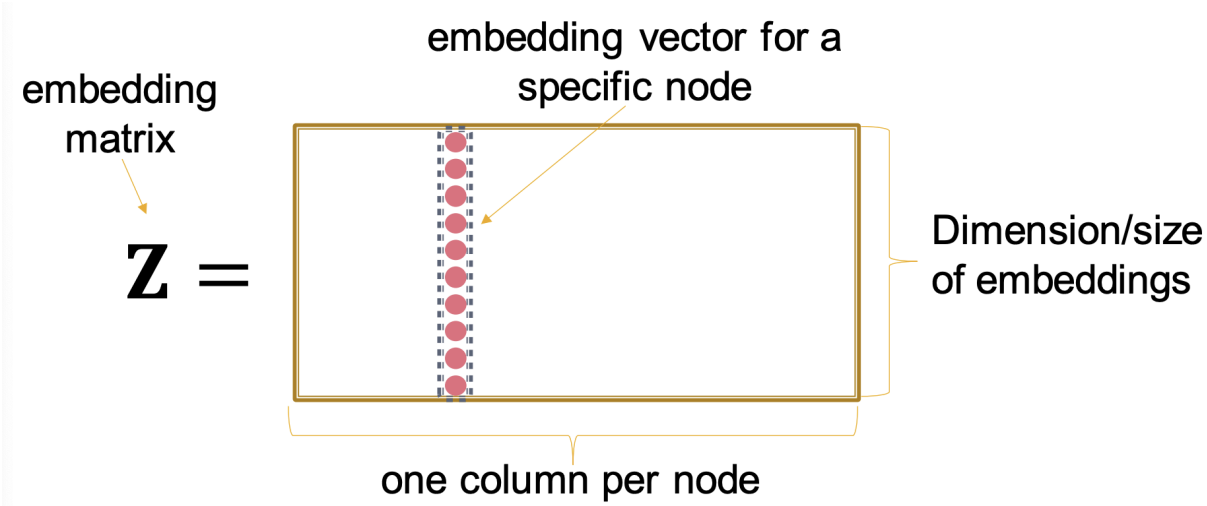
- A graph is naturally represented as a matrix
- we defined a random walk process over the graph
  - Random surfer moving across the links and with random teleportation
  - Stochastic adjacency matrix M
- PageRank = Limiting distribution of the surfer location represented node importance
  - Corresponds to the leading eigenvector of transformed adjacency matrix M.

---

4. Matrix Factorization and Node Embeddings

Embeddings & Matrix Factorization

- Recall : encoder as an embedding lookup



Objective: maximize $\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$ for node pairs (u, v) that are **similar**

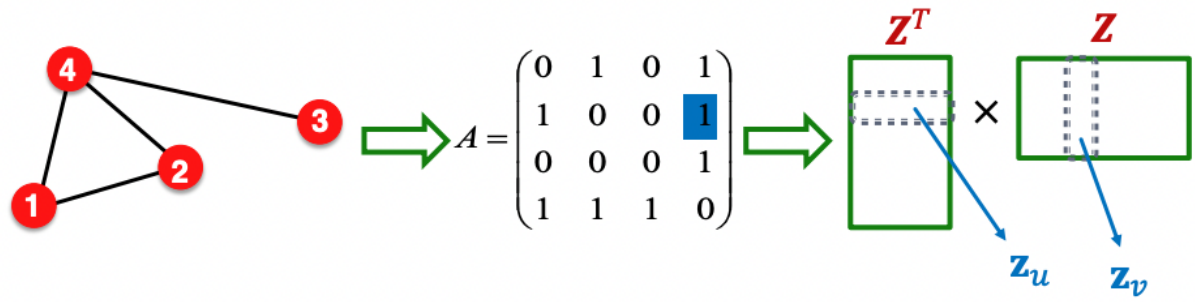
Connection to Matrix Factorization

- simplest node similarity : Nodes u, v are similar if they are connected by an edge
- This means :

$$\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u = A_{u,v}$$

which is the (u, v) entry of the graph adjacency matrix A

- Therefore, ZTZ = A

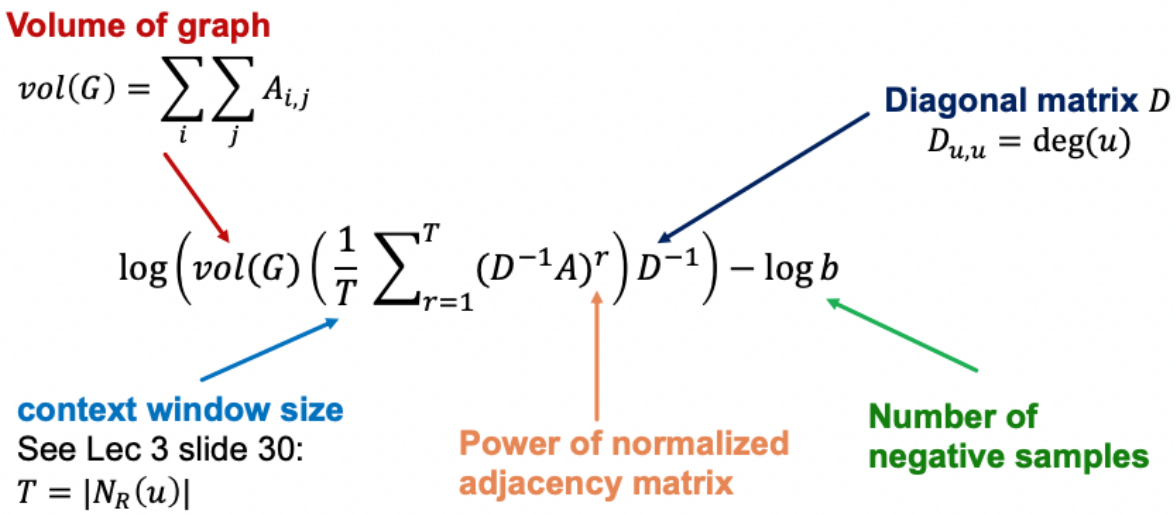


Matrix Factorization

- The embedding dimension d (number of rows in Z) is much smaller than number of nodes n.
- Exact factorization $A = Z^T Z$ is generally not possible
- However, we can learn Z approximately
- **Objective**: $\min_Z \| A - Z^T Z \|_2$
  - We optimize Z such that it minimizes the L2 norm (Frobenius norm) of $A - Z^T Z$
  - Note in Lecture 3 we used softmax instead of L2. But the goal to approximate A with $Z^T Z$ is the same.
- Conclusion: **Inner product decoder with node similarity defined by edge connectivity is equivalent to matrix factorization of A.**

Random Walk-based Similarity

- **DeepWalk** and **node2vec** have a more complex **node similarity** definition based on random walks
- **DeepWalk** is equivalent to matrix factorization of the following complex matrix expression:

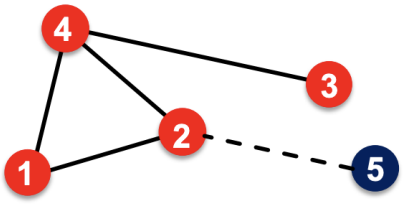


- Node2vec can also be formulated as a matrix factorization(a little bit a more complex matrix)
- Refer to the paper for more details

Limitations (1)

Limitations of node embeddings via matrix factorization and random walks

- Cannot obtain embeddings for nodes not in the training set



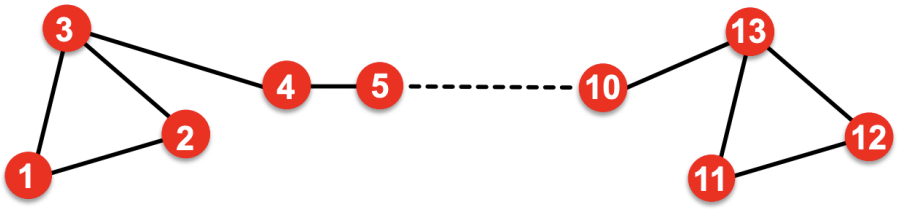
Training set

A newly added node 5 at test time (e.g., new user in a social network)

Cannot compute its embedding with DeepWalk / node2vec. Need to recompute all node embeddings.

Limitation (2)

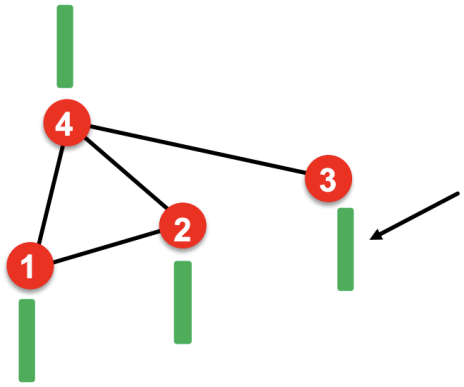
- Cannot capture **structural similarity**:



- Node 1 and 11 are **structurally similar** – part of one triangle, degree 2, …
- However, they have very **different** embeddings.
  - It's unlikely that a random walk will reach node 11 from node 1.
- **DeepWalk and node2vec do not capture structural similarity.**

Limitations (3)

- Cannot utilize node, edge and graph features



Feature vector
(e.g. protein properties in a protein-protein interaction graph)

DeepWalk / node2vec embeddings do not incorporate such node features

Solution to these limitations: Deep Representation Learning and Graph Neural Networks
(To be covered in depth next week)

---

5. Summary

- PageRank
  - Measures importance of nodes in graph
  - Can be efficiently computed by **power iteration of adjacency matrix**
- Personalized PageRank(PPR)
  - Measures importance of nodes with respect to a particular node or set of nodes
  - Can be efficiently computed by **random walk**
- **Node embeddings** based on random walks can be expressed as **matrix factorization**
- **Viewing graphs as matrices plays a key role in all above algorithms!**

References

https://velog.io/@kimkj38/CS224W-Lecture4-Graph-as-Matrix

http://web.stanford.edu/class/cs224w/