

# DATA MINING

# LINK ANALYSIS RANKING

---

**PageRank -- Random walks**

**HITS**

**Absorbing Random Walks and Label Propagation**



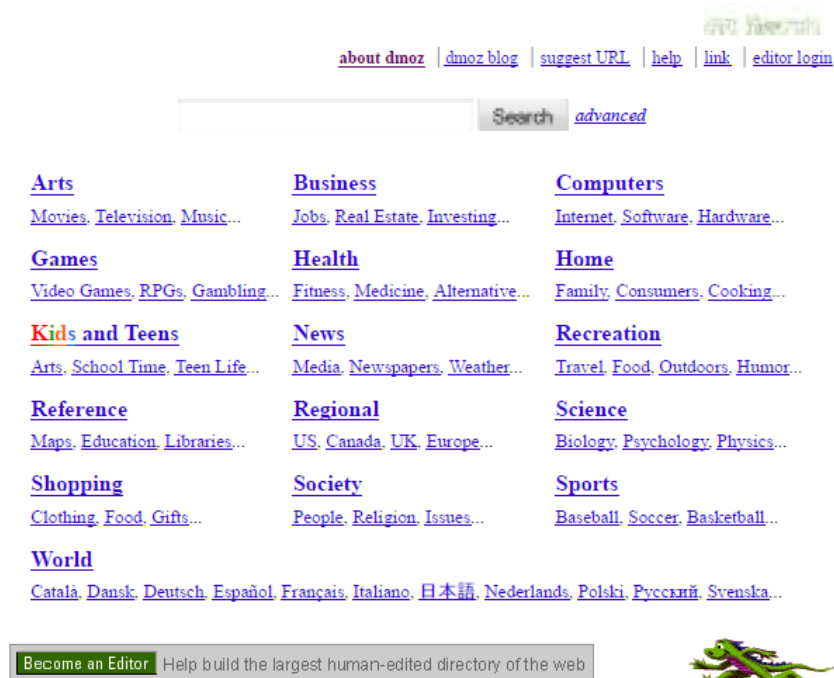
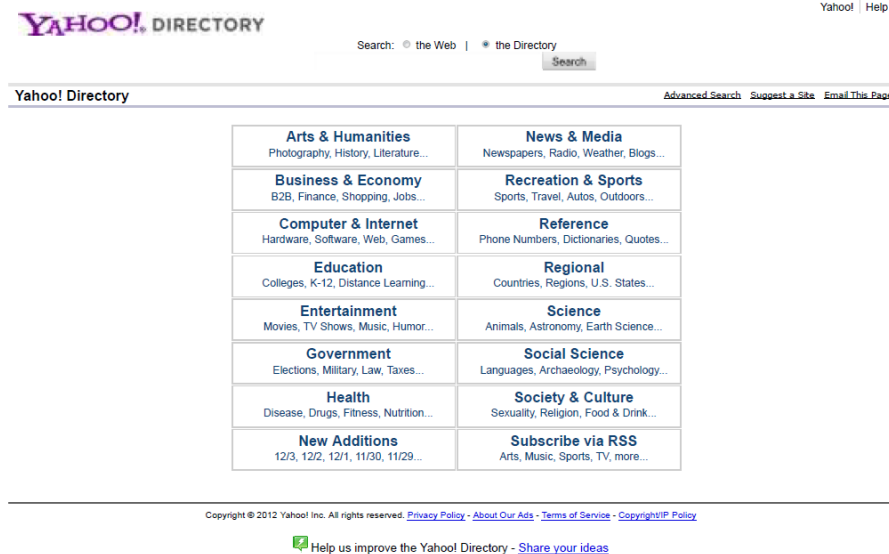
# Network Science

- A number of complex systems can be modeled as **networks** (graphs).
  - The **Web**
  - (Online) Social Networks
  - Biological systems
  - Communication networks (internet, email)
  - The Economy
- We cannot truly understand such **complex systems** unless we understand the **underlying network**.
  - Everything is **connected**, studying individual entities gives only a partial view of a system
- Data mining for networks is a very popular area
  - Applications to the **Web** is one of the success stories for network data mining.



# How to organize the web

- **First try:** Manually curated Web Directories



Copyright © 2012 Netscape

5,114,642 sites - 96,895 editors - over 1,014,858 categories



# How to organize the web

- **Second try:** Web Search

- **Information Retrieval** investigates:

- Find relevant docs in a small and trusted set e.g., Newspaper articles, Patents, etc. (“needle-in-a-haystack”)
    - Limitation of keywords (synonyms, polysemy, etc)

- **But:** Web is huge, full of untrusted documents, random things, web spam, etc.

- Everyone can create a web page of high production value
    - Rich diversity of people issuing queries
    - Dynamic and constantly-changing nature of web content



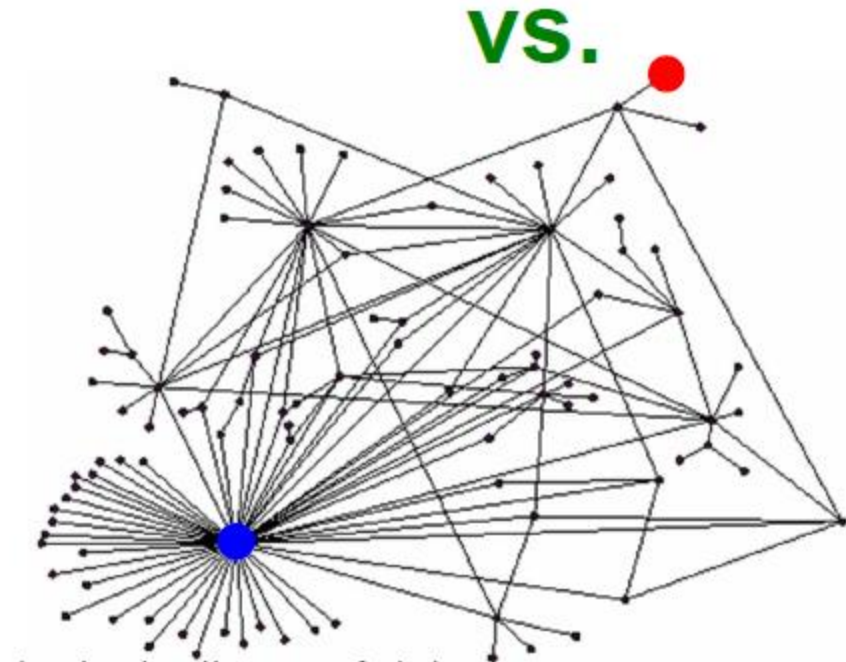
# How to organize the web

- **Third try** (the **Google** era): using the web graph
  - Sift from **relevance** to **authoritativeness**
  - It is not only important that a page is relevant, but that it is also important on the web
- For example, what kind of results would we like to get for the query “greek newspapers”?



# Link Analysis Ranking

- Use the **graph structure** in order to determine the **relative importance** of the nodes
  - Applications: Ranking on graphs (Web, Twitter, FB, etc)
- **Intuition**: An edge from node **p** to node **q** denotes **endorsement**
  - Node **p** **endorses/recommends/confirm**s the **authority/centrality/importance** of node **q**
  - Use the graph of recommendations to assign an **authority value** to every node

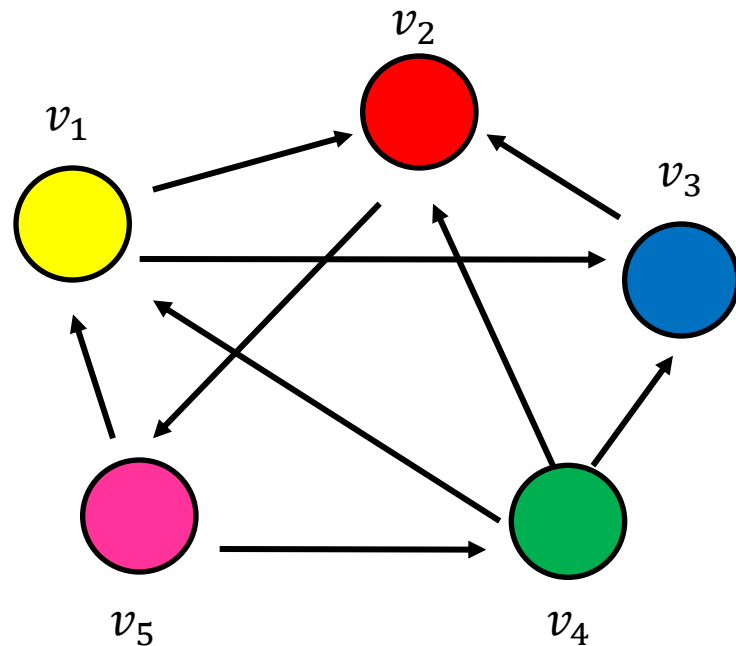


What is the simplest way to measure importance of a page on the web?



# Rank by Popularity

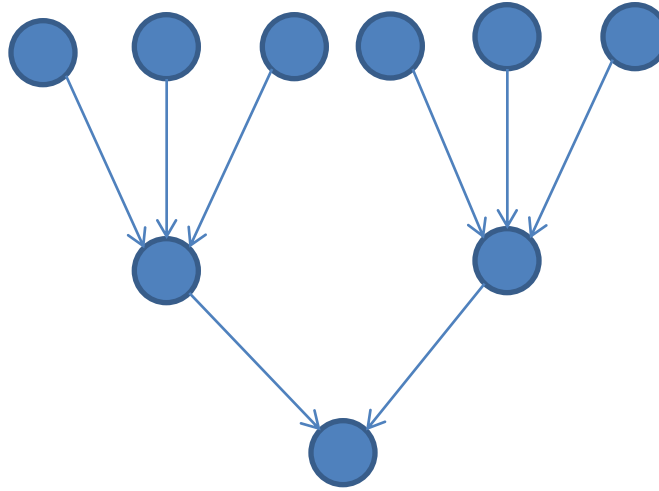
- Rank pages according to the number of incoming edges (**in-degree**, **degree centrality**)



- 1. Red Page**
- 2. Yellow Page**
- 3. Blue Page**
- 4. Purple Page**
- 5. Green Page**



# Popularity



- It is not important only how many link to you, but how important are the people that link to you.
- **Good** authorities are pointed by **good** authorities
  - Recursive definition of importance



# PAGERANK

---



# PageRank

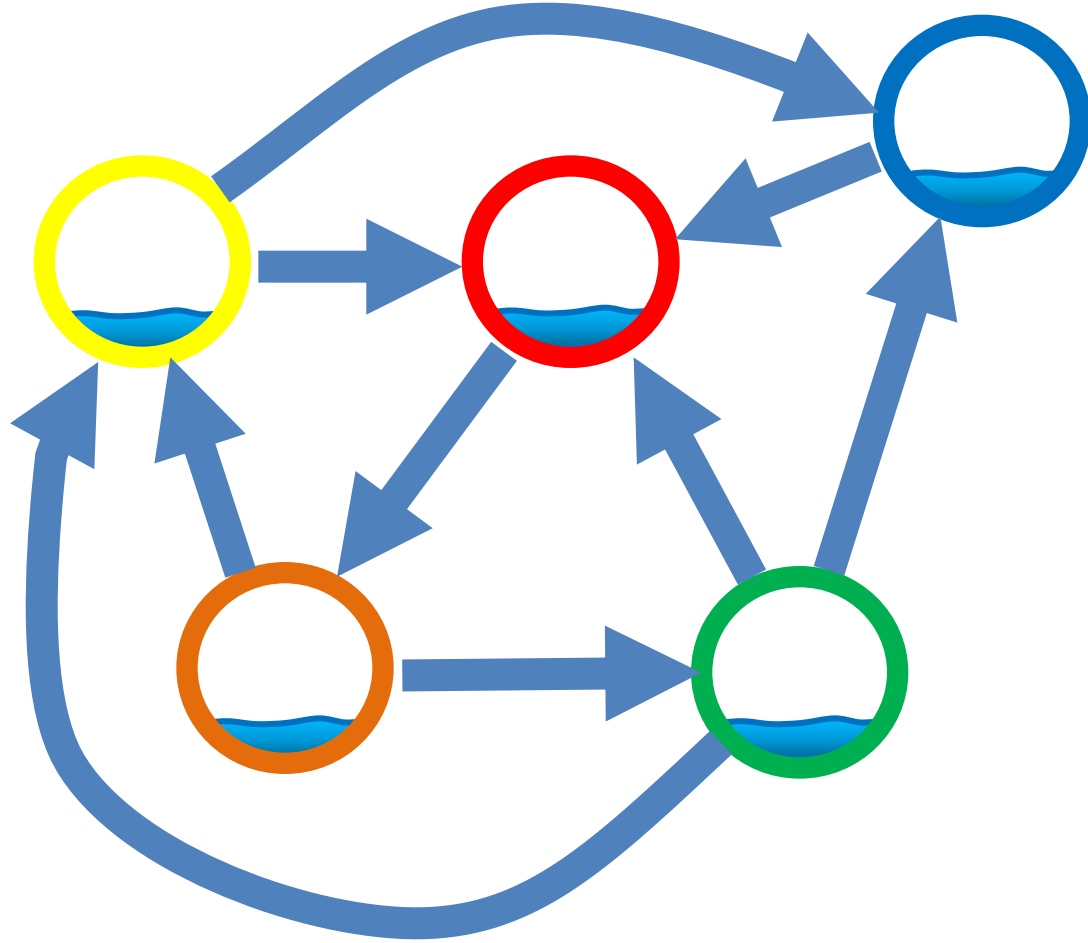
- **Good** authorities should be pointed by **good** authorities
  - The value of a node is the value of the nodes that point to it.
- How do we implement that?
  - Assume that we have **a unit of authority** to distribute to all nodes.
    - Initially each node gets  $\frac{1}{n}$  amount of authority
  - Each node **distributes** the authority value they have **to their neighbors**
  - The authority value of each node is the sum of the **authority fractions** it collects from its neighbors.



# The PageRank algorithm

Think of the nodes in the graph as **containers** of capacity of 1 liter.

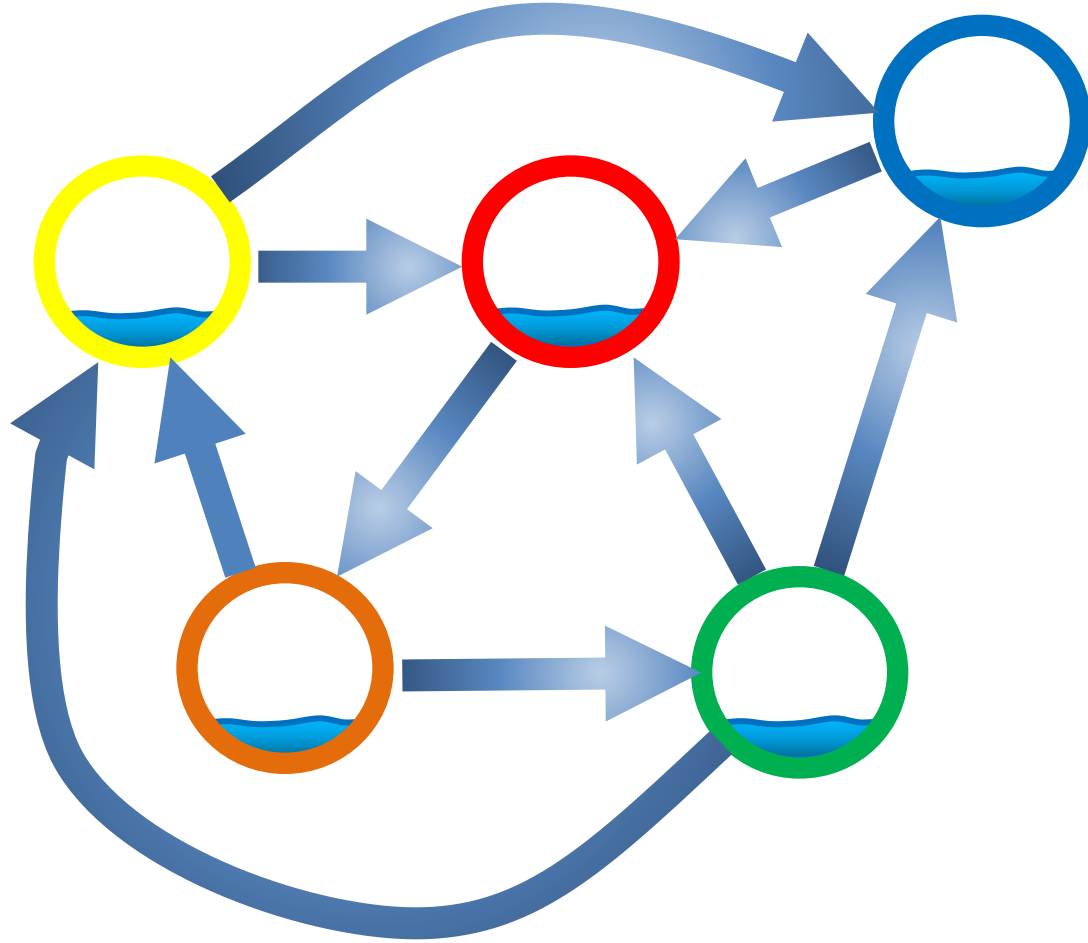
We distribute a liter of liquid equally to all containers





# The PageRank algorithm

The edges act like pipes that **transfer** liquid between nodes.

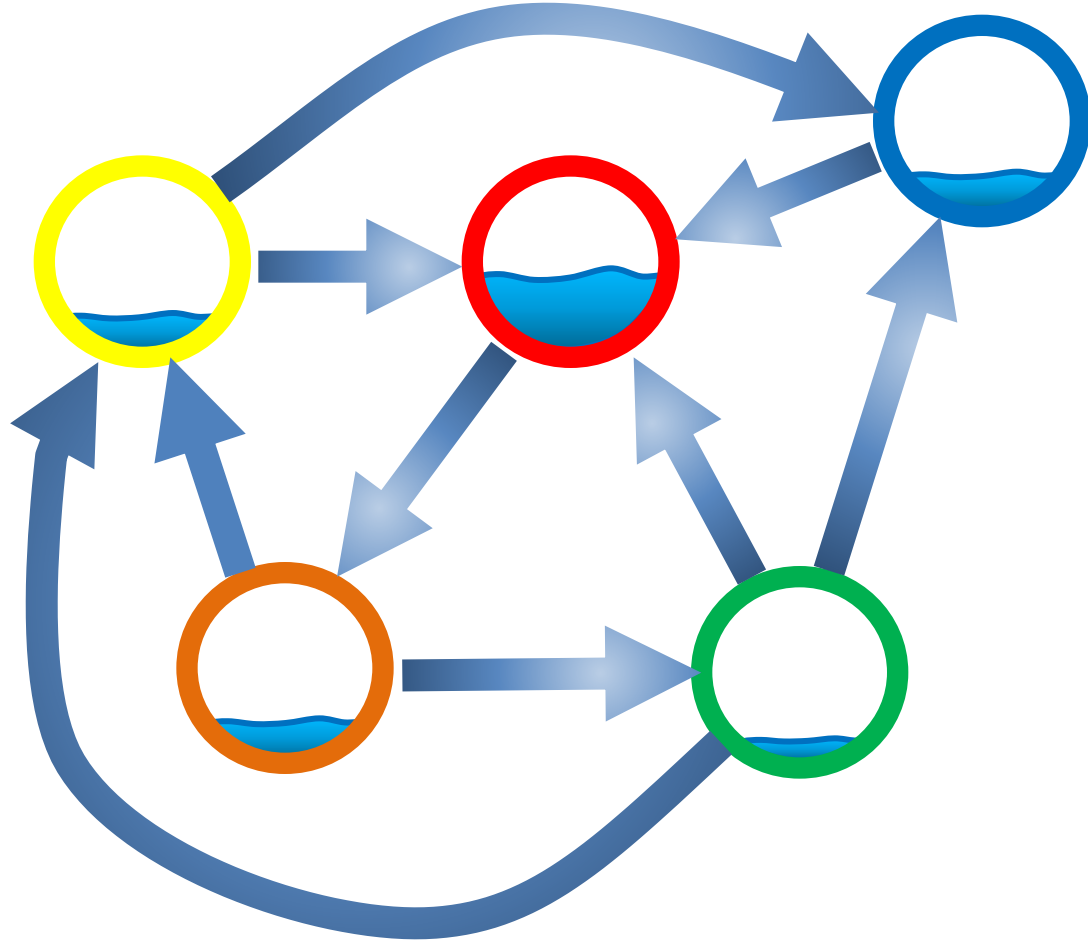




# The PageRank algorithm

The edges act like pipes that **transfer** liquid between nodes.

The contents of each node are **distributed** to its neighbors.

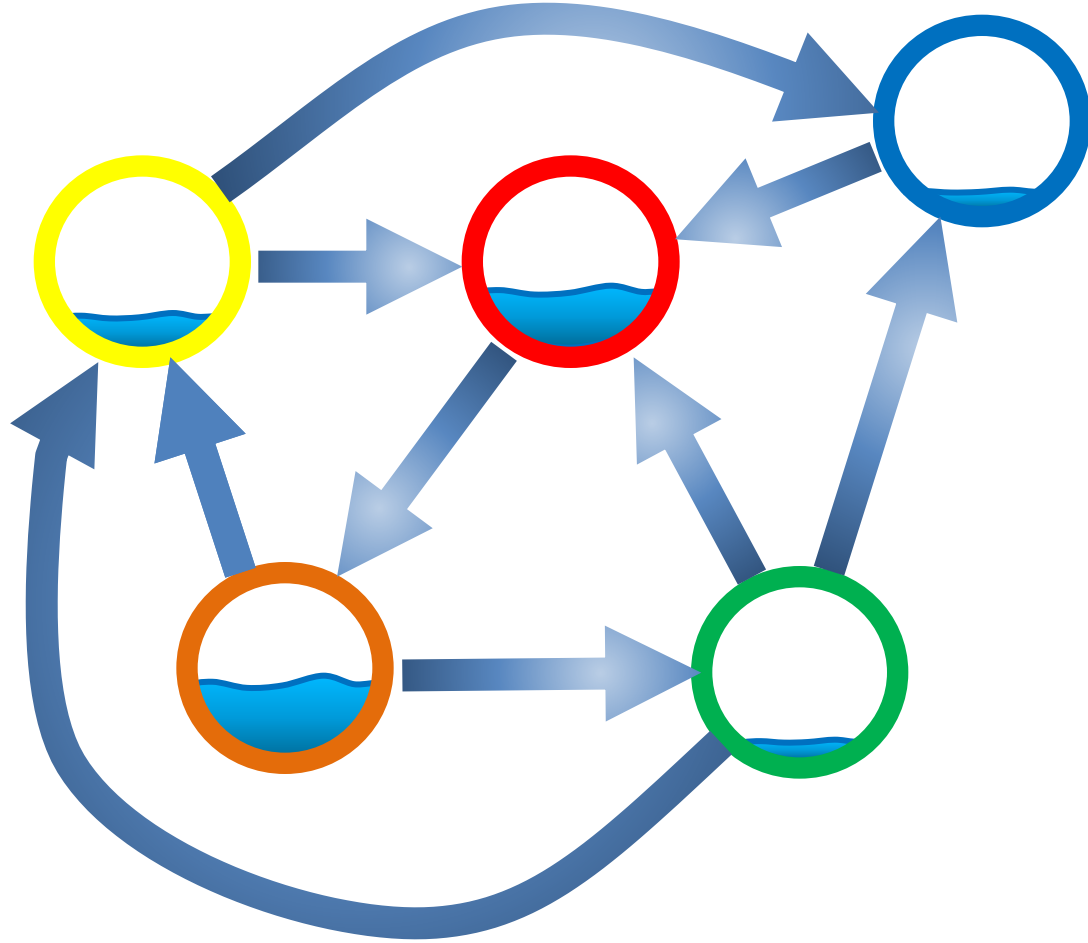




# The PageRank algorithm

The edges act like pipes that **transfer** liquid between nodes.

The contents of each node are **distributed** to its neighbors.

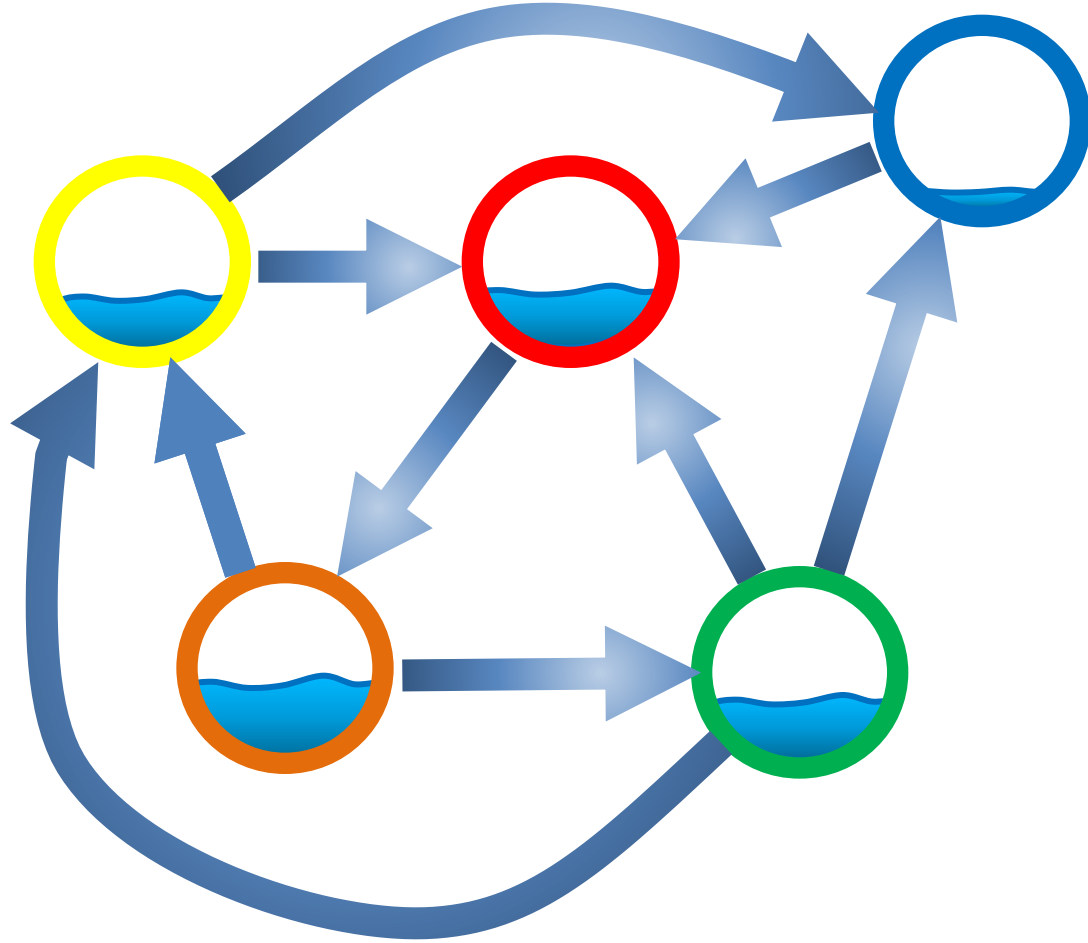




# The PageRank algorithm

The edges act like pipes that **transfer** liquid between nodes.

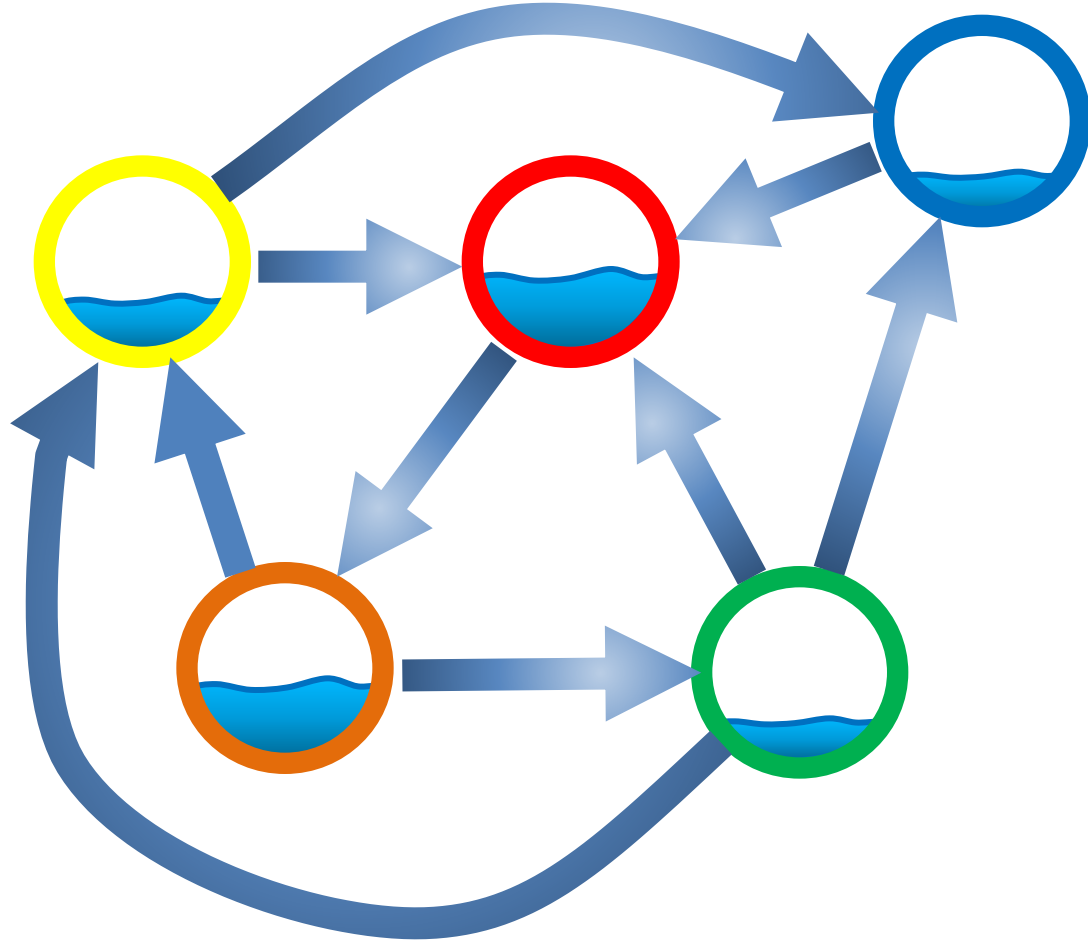
The contents of each node are **distributed** to its neighbors.





# The PageRank algorithm

The system will reach an **equilibrium** state where the amount of liquid in each node remains constant.

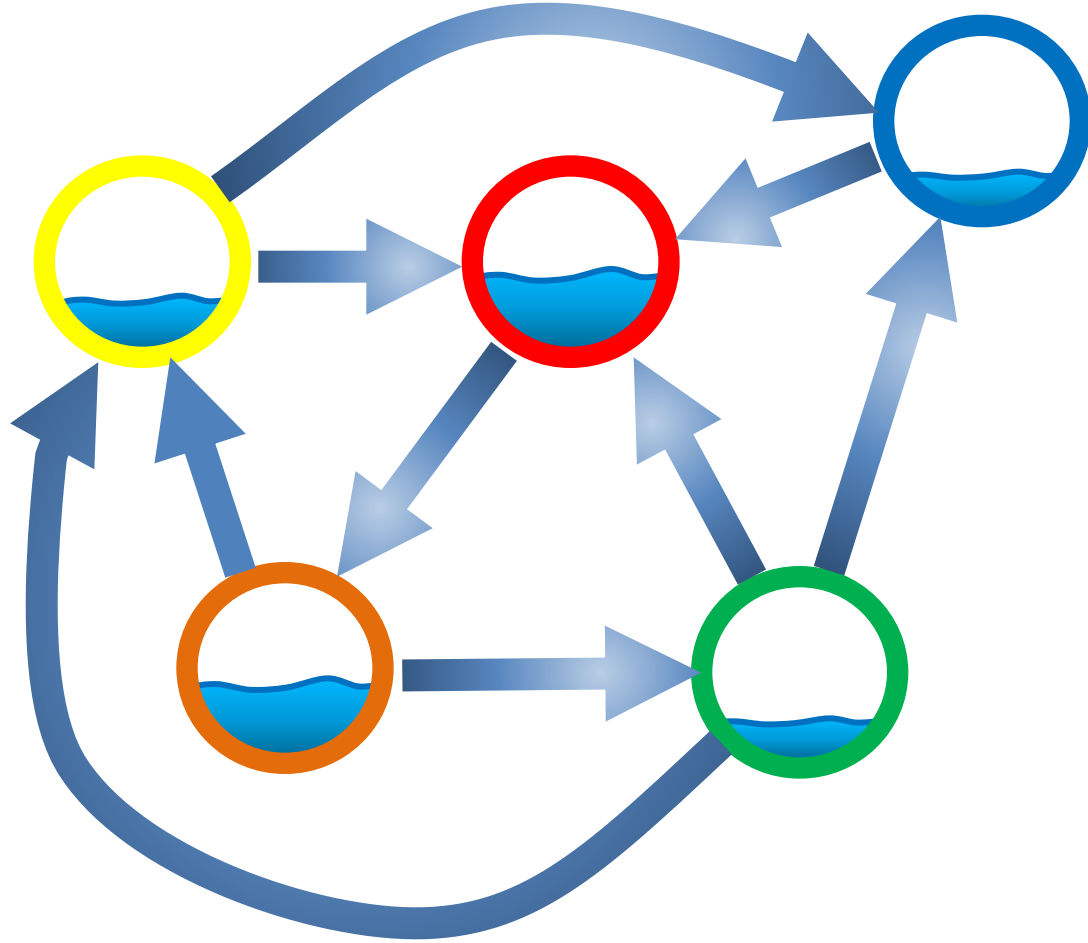




# The PageRank algorithm

The amount of liquid in each node determines the **importance** of the node.

**Large quantity** means large **incoming flow** from nodes with **large quantity** of liquid.





# PageRank

- **Good** authorities should be pointed by **good** authorities
  - The value of a node is the value of the nodes that point to it.
- How do we implement that?
  - Assume that we have **a unit of authority** to distribute to all nodes.
    - Initially each node gets  $\frac{1}{n}$  amount of authority
  - Each node **distributes** the authority value they have **to their neighbors**
  - The authority value of each node is the sum of the **authority fractions** it collects from its neighbors.

$$w_v = \sum_{u \rightarrow v} \frac{1}{d_{out}(u)} w_u$$

$w_v$ : the **PageRank value** of node  $v$

Recursive definition



# Example

$$w_v = \sum_{u \rightarrow v} \frac{1}{d_{out}(u)} w_u$$

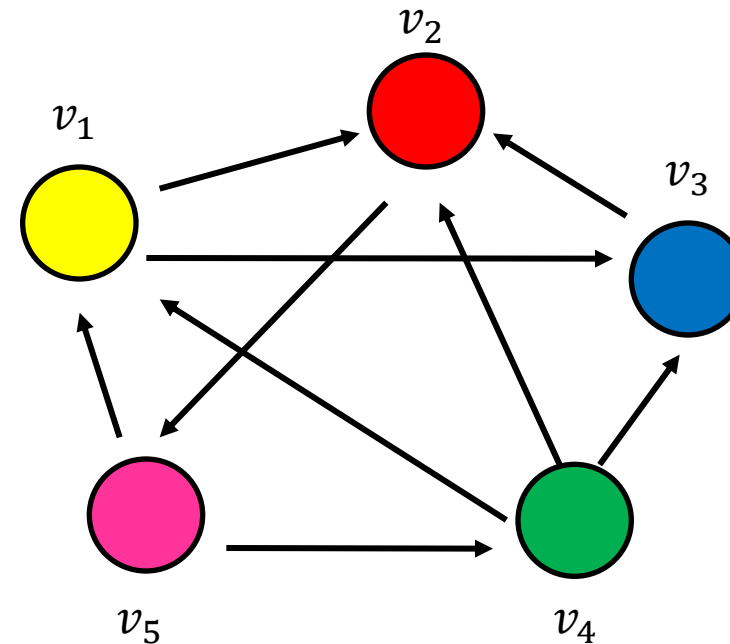
$$w_1 = 1/3 w_4 + 1/2 w_5$$

$$w_2 = 1/2 w_1 + w_3 + 1/3 w_4$$

$$w_3 = 1/2 w_1 + 1/3 w_4$$

$$w_4 = 1/2 w_5$$

$$w_5 = w_2$$





# Computing PageRank weights

- A simple way to compute the weights is by iteratively updating the weights using the equations
- PageRank Algorithm

Initialize all PageRank weights to  $\frac{1}{n}$

Repeat:

$$w_v = \sum_{u \rightarrow v} \frac{1}{d_{out}(u)} w_u$$

Until the weights do not change

- This process converges



# Example

$$w_1 = 1/3 w_4 + 1/2 w_5$$

$$w_2 = 1/2 w_1 + w_3 + 1/3 w_4$$

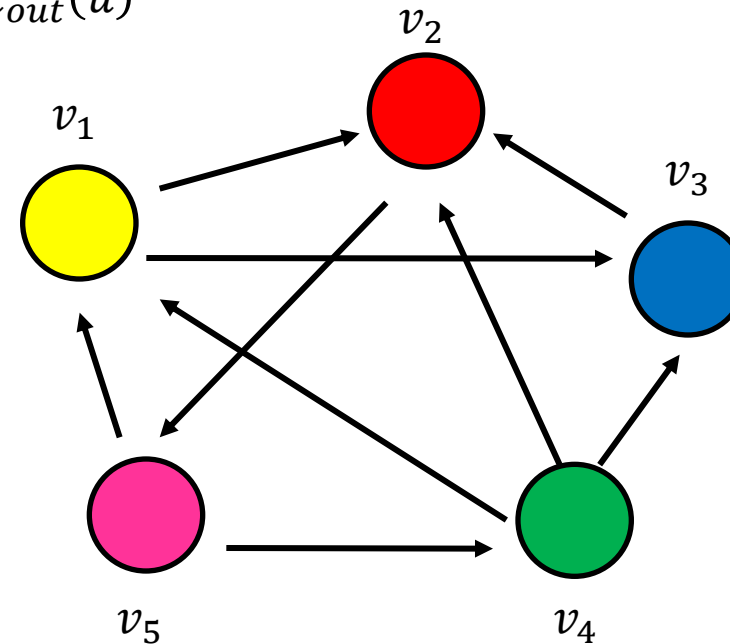
$$w_3 = 1/2 w_1 + 1/3 w_4$$

$$w_4 = 1/2 w_5$$

$$w_5 = w_2$$

	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$
t=0	0.2	0.2	0.2	0.2	0.2
t=1	0.16	0.36	0.16	0.1	0.2
t=2	0.13	0.28	0.11	0.1	0.36
t=3	0.22	0.22	0.1	0.18	0.28
t=4	0.2	0.27	0.17	0.14	0.22

$$w_v = \sum_{u \rightarrow v} \frac{1}{d_{out}(u)} w_u$$



Think of the weight as a **fluid**: there is constant amount of it in the graph, but it moves around until it stabilizes



# Example

$$w_1 = 1/3 w_4 + 1/2 w_5$$

$$w_2 = 1/2 w_1 + w_3 + 1/3 w_4$$

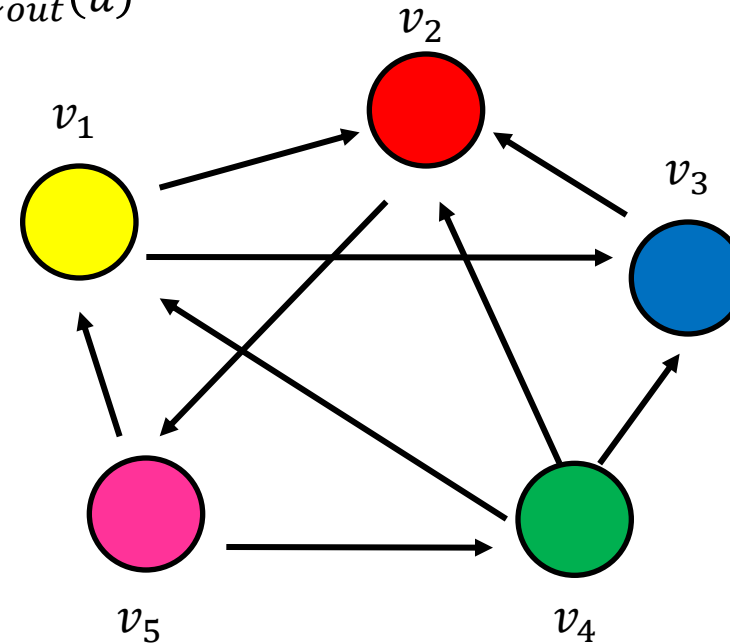
$$w_3 = 1/2 w_1 + 1/3 w_4$$

$$w_4 = 1/2 w_5$$

$$w_5 = w_2$$

	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$
t=25	0.18	0.27	0.13	0.13	0.27

$$w_v = \sum_{u \rightarrow v} \frac{1}{d_{out}(u)} w_u$$



Think of the weight as a **fluid**: there is constant amount of it in the graph, but it moves around until it stabilizes



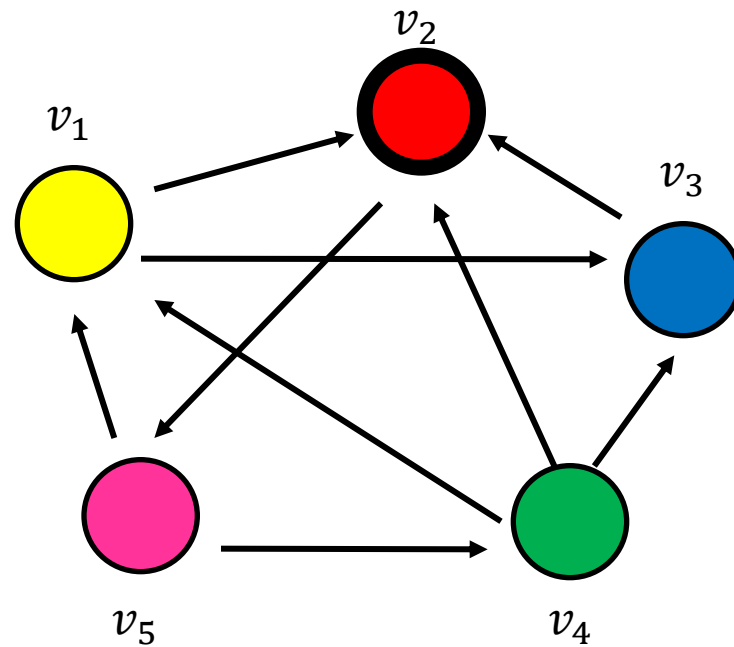
# Random Walks on Graphs

- The algorithm defines a **random walk** on the graph
- Random walk:
  - **Start** from a node chosen **uniformly at random** with probability  $\frac{1}{n}$ .
    - **Pick** one of the **outgoing edges** **uniformly at random**
    - **Move** to the destination of the edge
    - Repeat.
- The PageRank of node  $v$  is the probability that the random walk is at node  $v$  after a very large number of steps.
- The **Random Surfer** model
  - Users wander on the web, following links.



# Example

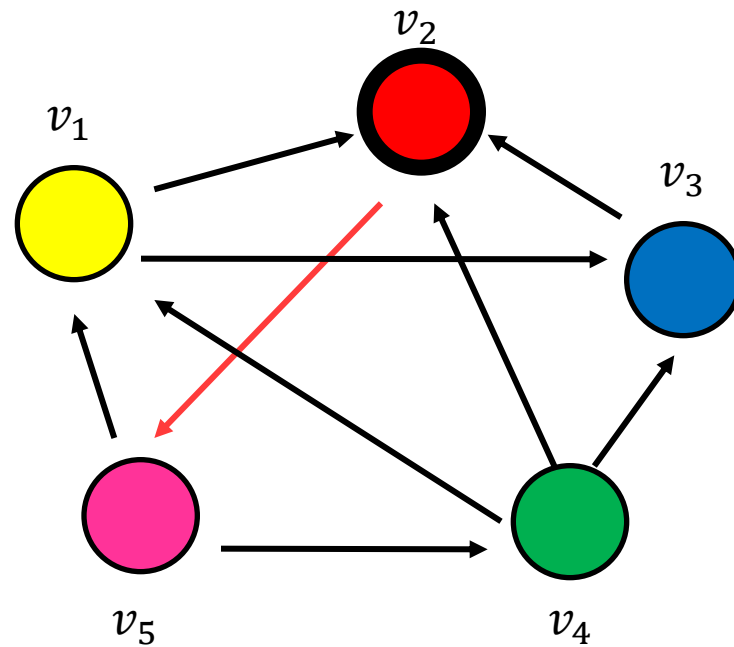
- Step 0





# Example

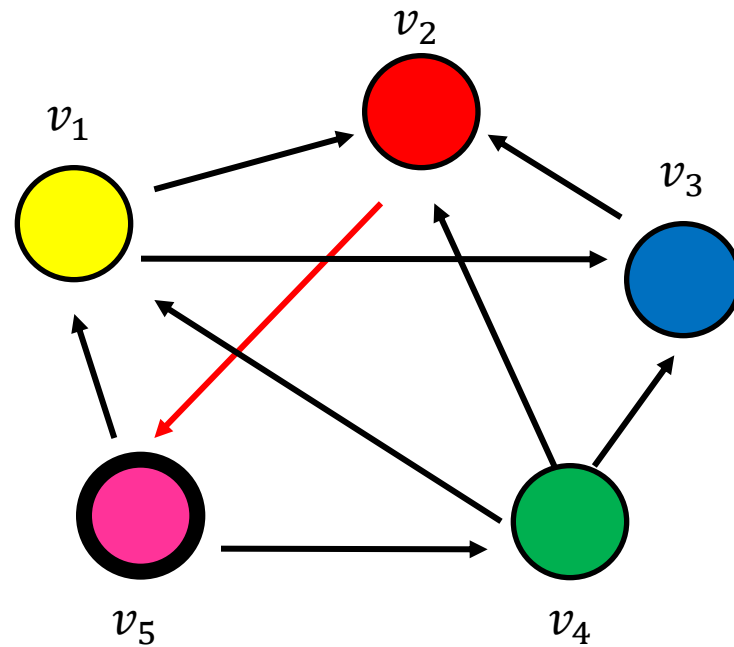
- Step 0





# Example

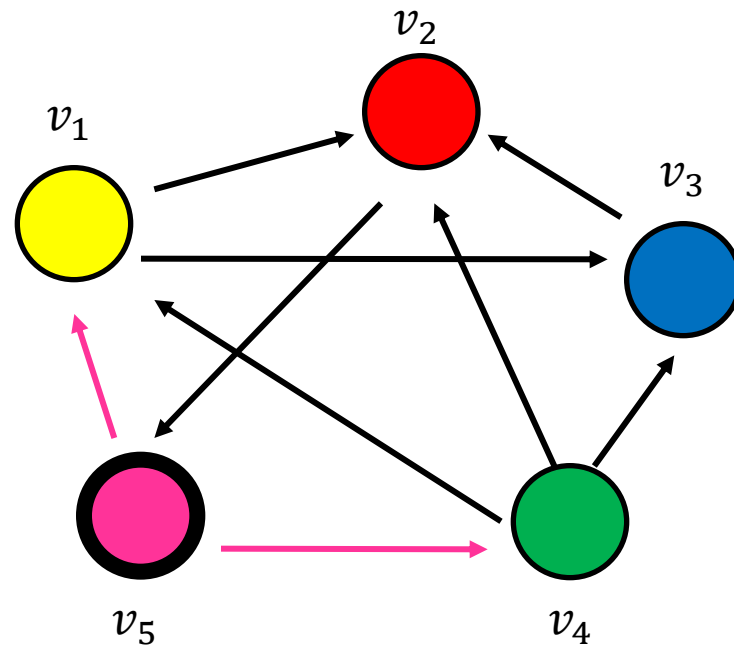
- Step 1





# Example

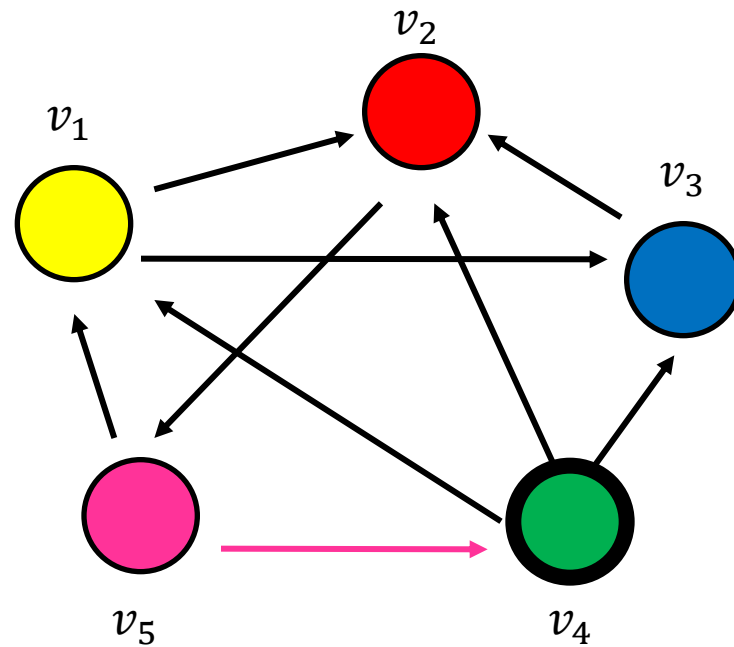
- Step 1





# Example

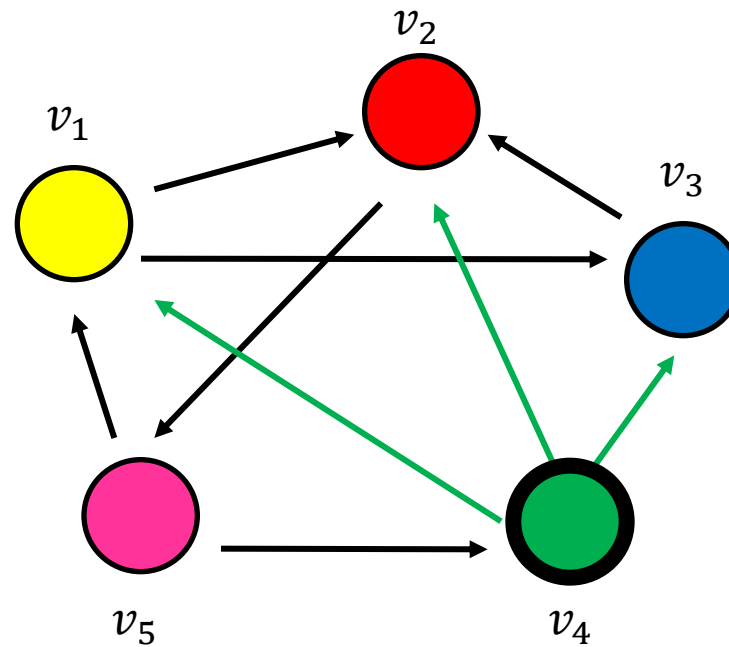
- Step 2





# Example

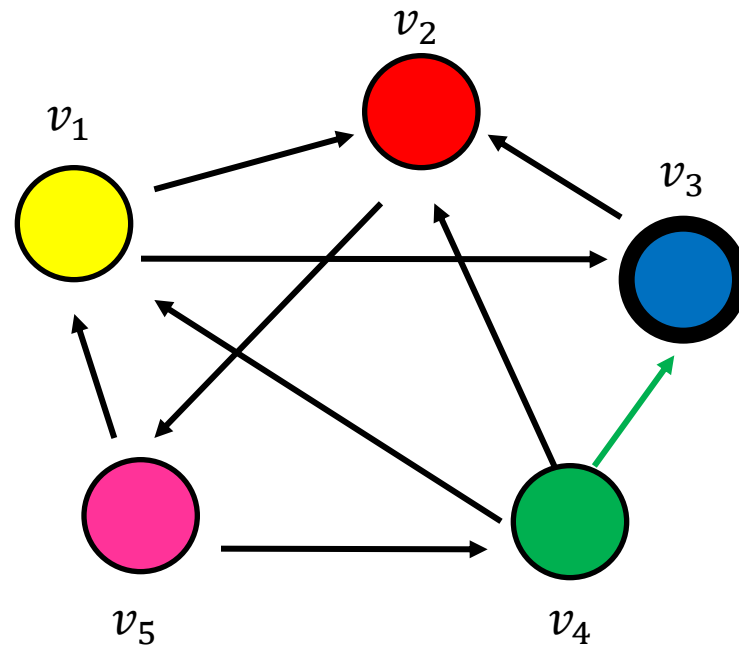
- Step 2





# Example

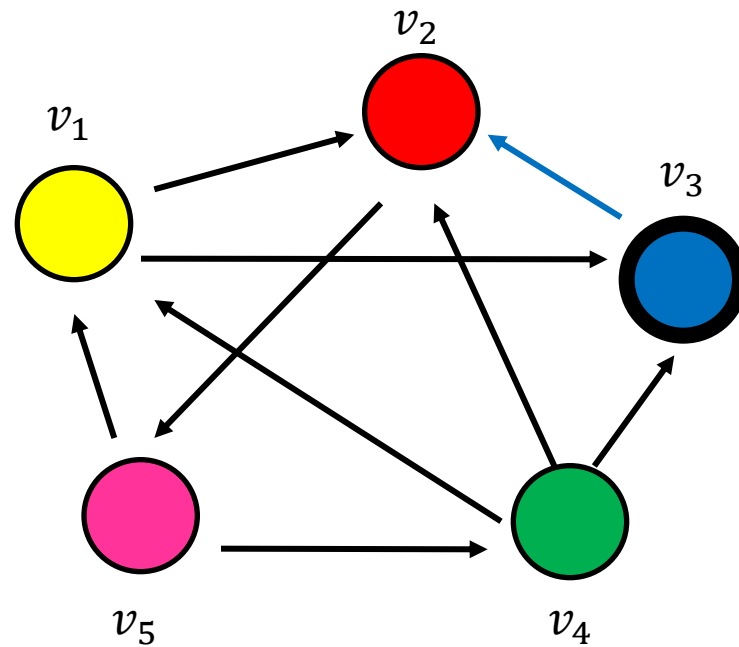
- Step 3





# Example

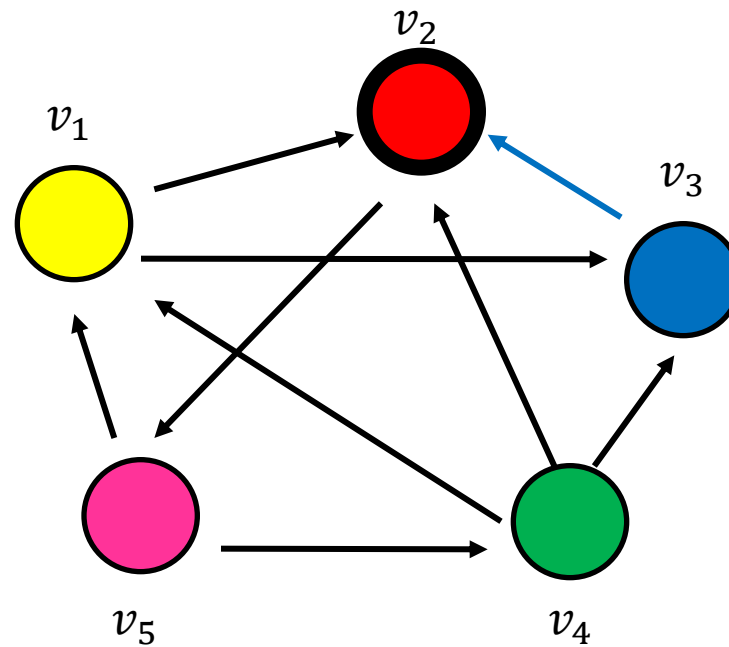
- Step 3





# Example

- Step 4...





# Random walk

- Question: what is the probability  $p_i^t$  of being at node  $i$  after  $t$  steps?

$$p_1^0 = \frac{1}{5}$$

$$p_2^0 = \frac{1}{5}$$

$$p_3^0 = \frac{1}{5}$$

$$p_4^0 = \frac{1}{5}$$

$$p_5^0 = \frac{1}{5}$$

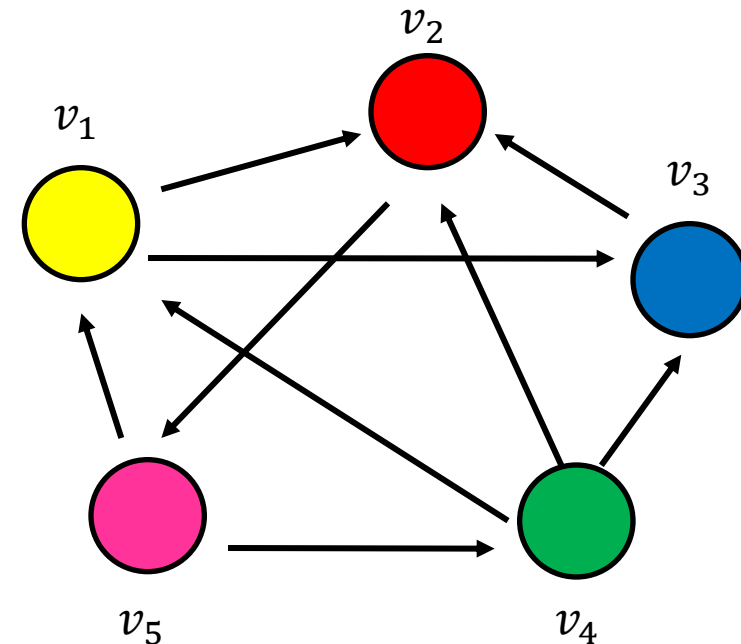
$$p_1^t = \frac{1}{3}p_4^{t-1} + \frac{1}{2}p_5^{t-1}$$

$$p_2^t = \frac{1}{2}p_1^{t-1} + p_3^{t-1} + \frac{1}{3}p_4^{t-1}$$

$$p_3^t = \frac{1}{2}p_1^{t-1} + \frac{1}{3}p_4^{t-1}$$

$$p_4^t = \frac{1}{2}p_5^{t-1}$$

$$p_5^t = p_2^{t-1}$$



The equations are the same as those for the PageRank computation



# Markov chains

- A Markov chain describes a **discrete time stochastic process** over a set of states

$$S = \{s_1, s_2, \dots, s_n\}$$

according to a transition probability matrix  $P = \{P_{ij}\}$

- $P_{ij}$  = probability of moving to state  $j$  when at state  $i$
- Matrix  $P$  has the property that the entries of all **rows sum to 1**

$$\sum_j P[i, j] = 1$$

A matrix with this property is called **stochastic**

- **State probability distribution**: The vector  $p^t = (p_1^t, p_2^t, \dots, p_n^t)$  that stores the probability of being at state  $s_i$  after  $t$  steps
- **Memorylessness property**: The **next state** of the chain **depends only at the current state** and not on the past of the process (**first order MC**)
  - **Higher order** MCs are also possible
- **Markov Chain Theory**: After infinite steps the **state probability vector converges** to a **unique** distribution if the chain is **irreducible** (possible to get from any state to any other state) and **aperiodic**
  - These are the PageRank values



# Random walks

- Random walks on graphs correspond to Markov Chains
  - The set of states  $S$  is the set of nodes of the graph  $G$
  - The **transition probability matrix** is the probability that we follow an edge from one node to another

$$P[i, j] = \frac{1}{d_{out}(i)}$$

- We can compute the vector  $p^t$  at step  $t$  using a vector-matrix multiplication

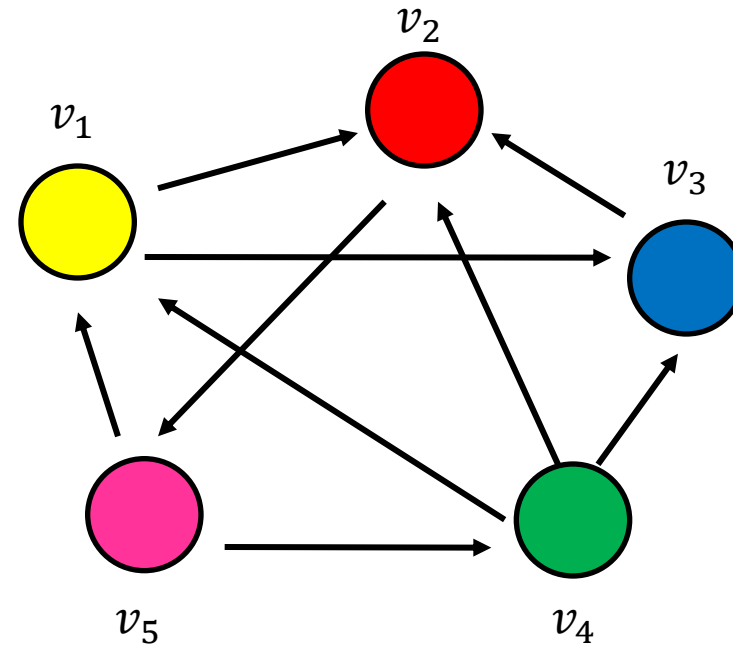
$$p^{t+1} = p^t P$$



# An example

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

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





# An example

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

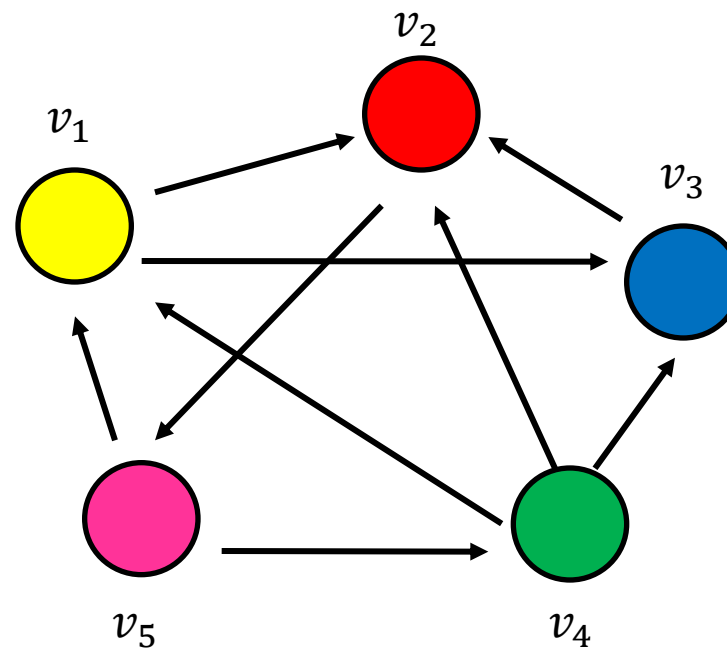
$$p_1^t = \frac{1}{3} p_4^{t-1} + \frac{1}{2} p_5^{t-1}$$

$$p_2^t = \frac{1}{2} p_1^{t-1} + p_3^{t-1} + \frac{1}{3} p_4^{t-1}$$

$$p_3^t = \frac{1}{2} p_1^{t-1} + \frac{1}{3} p_4^{t-1}$$

$$p_4^t = \frac{1}{2} p_5^{t-1}$$

$$p_5^t = p_2^{t-1}$$





# Stationary distribution

- The **stationary distribution** of a random walk with transition matrix  $P$ , is a probability distribution  $\pi$ , such that  $\pi = \pi P$
- The stationary distribution is an **eigenvector** of matrix  $P$ 
  - the **principal left eigenvector** of  $P$  – stochastic matrices have maximum eigenvalue 1
- **Markov Chain Theory**: The random walk converges to a **unique stationary distribution independent of the initial vector** if the graph is **strongly connected**, and **not bipartite**.
  - These are the PageRank values



# Computing the stationary distribution

- The Power Method

Initialize  $p^0$  to some distribution

Repeat

$$p^t = p^{t-1}P$$

Until convergence

- After many iterations  $p^t \rightarrow \pi$  regardless of the initial vector  $p^0$
- Power method because it computes  $p^t = p^0 P^t$
- Rate of convergence
  - determined by the second eigenvalue  $\lambda_2$



# The stationary distribution

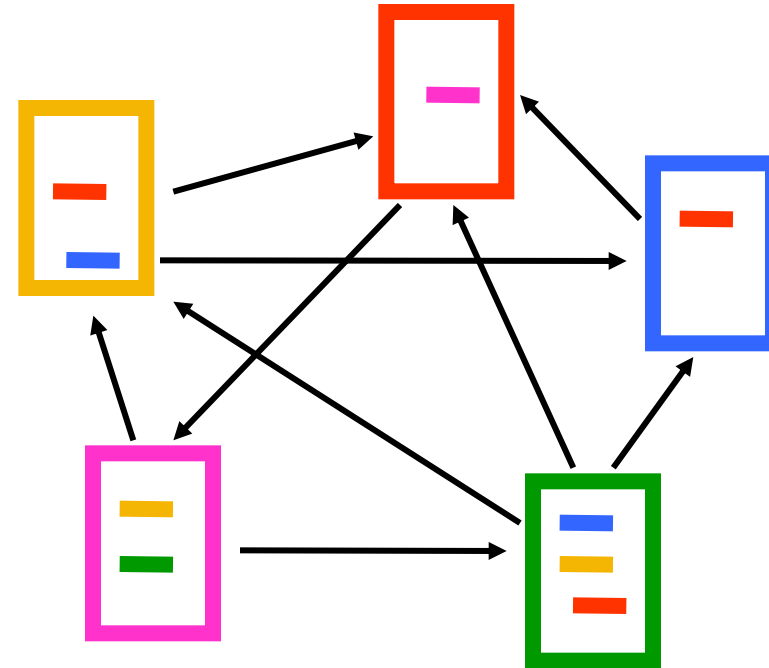
- What is the meaning of the stationary distribution  $\pi$  of a random walk?
- $\pi(i)$ : the fraction of times that we visited state  $i$  as  $t \rightarrow \infty$
- $\pi(i)$ : the probability of being at node  $i$  after very large (infinite) number of steps
- $\pi$  is the left eigenvector of transition matrix  $P$
- $\pi = p_0 P^\infty$ , where  $P$  is the transition matrix,  $p_0$  the original vector
  - $P(i, j)$ : probability of going from  $i$  to  $j$  in one step
  - $P^2(i, j)$ : probability of going from  $i$  to  $j$  in two steps (probability of all paths of length 2)
  - $P^\infty(i, j) = \pi(j)$ : probability of going from  $i$  to  $j$  in infinite steps – starting point does not matter.



# The PageRank random walk

- Vanilla random walk
  - make the adjacency matrix stochastic and run a random walk

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

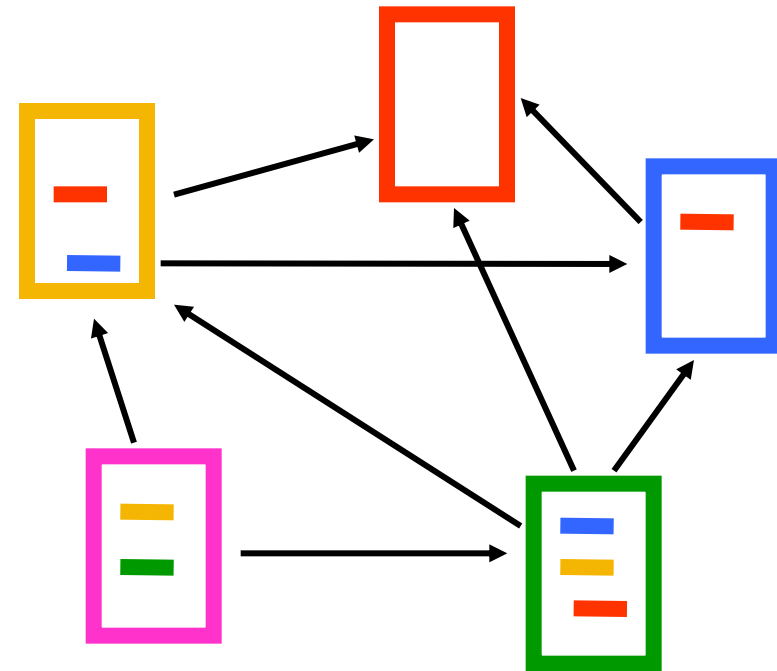




# The PageRank random walk

- What about **sink** nodes?
  - what happens when the random walk moves to a node without any outgoing links?

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





# The PageRank random walk

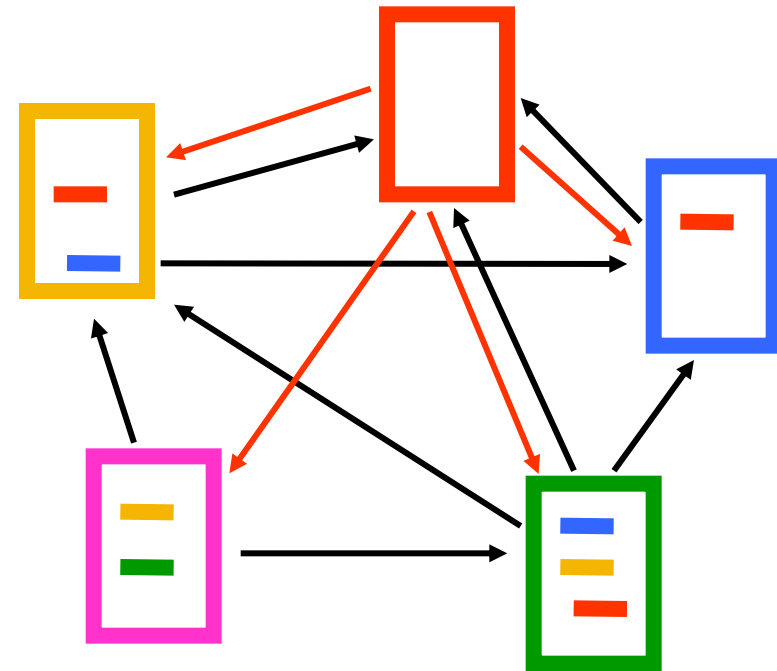
- Replace these row vectors with a vector  $\mathbf{v}$ 
  - typically, the uniform vector

$$P' = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$

Outer  
product

$$P' = P + d\mathbf{u}^T \quad d = \begin{cases} 1 & \text{if } i \text{ is sink} \\ 0 & \text{otherwise} \end{cases}$$

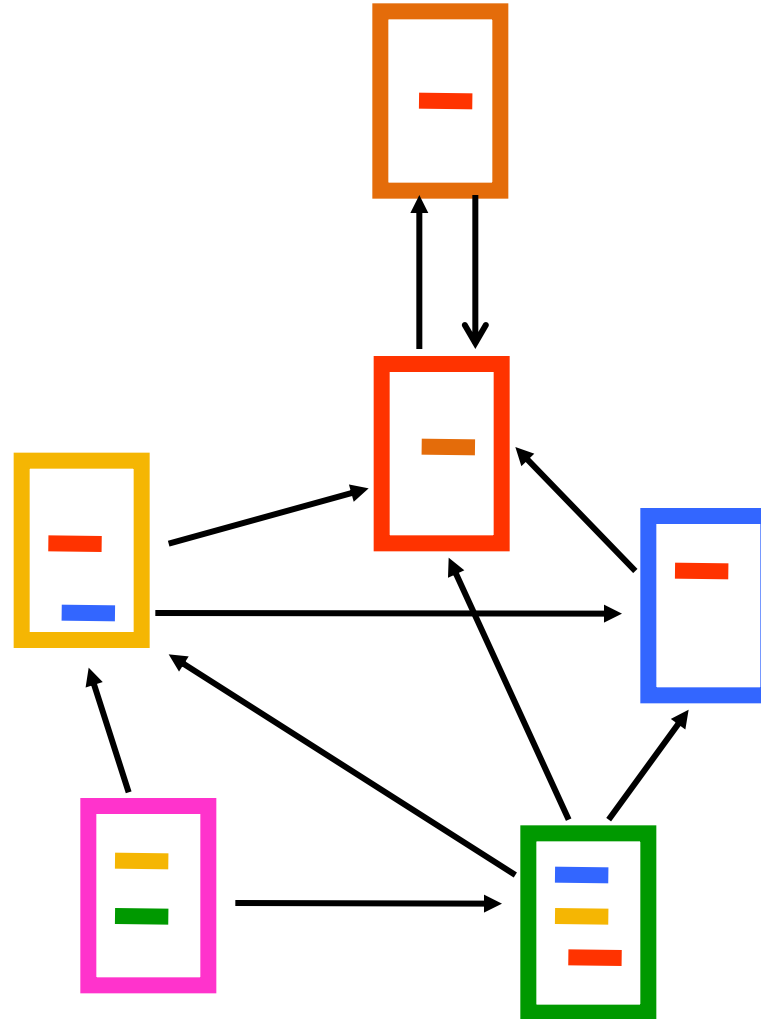
$\mathbf{u}$ : The jump vector





# The PageRank random walk

- What about loops?
  - Spider traps





# The PageRank random walk

- Add a **random jump** to vector  $v$  with prob  $\alpha$ 
  - Typically, to a uniform vector
  - Guarantees irreducibility, convergence
- You can think of the random jump as a **restart** of the random walk
  - Restart the walk after  $1/\alpha$  steps in expectation

$$P'' = (1 - \alpha) \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/2 \end{bmatrix} + \alpha \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix}$$

$P'' = (1 - \alpha)P' + \alpha \mathbf{1}u^T$ , where  $\mathbf{1}$  is the vector of all 1s

$\alpha$ : jump probability

Random walk with restarts

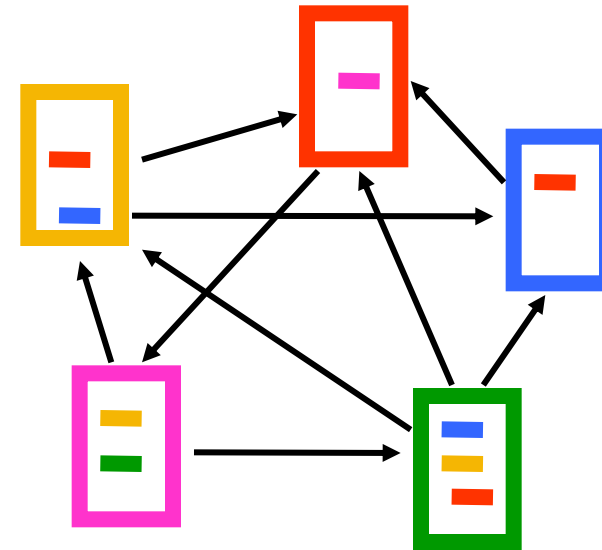


# PageRank algorithm [BP98]

- Rank according to the stationary distribution

$$w_v = (1 - \alpha) \sum_{u \rightarrow v} \frac{1}{d_{out}(u)} w_u + \alpha \frac{1}{n}$$

- $\alpha = 0.15$  in most cases
- The Random Surfer model
  - Start with a random page
  - With probability  $1 - \alpha$  follow one of the links in the page
  - With probability  $\alpha$  restart from a random page



- Red Page**
- Purple Page**
- Yellow Page**
- Blue Page**
- Green Page**



# Stationary distribution with random jump

- If  $u$  is the jump vector

$$p^0 = u$$

$$p^1 = (1 - \alpha)p^0P + \alpha u = (1 - \alpha)uP + \alpha u$$

$$p^2 = (1 - \alpha)p^1P + \alpha u = (1 - \alpha)^2uP^2 + (1 - \alpha)\alpha uP + \alpha u$$

$$p^2 = (1 - \alpha)p^2P + \alpha u = (1 - \alpha)^3uP^3 + (1 - \alpha)^2\alpha uP^2 + (1 - \alpha)\alpha uP + \alpha u$$

$\vdots$

$$p^\infty = \alpha u + (1 - \alpha)\alpha uP + (1 - \alpha)^2\alpha uP^2 + \dots = \alpha(I - (1 - \alpha)P)^{-1}u$$

- Explanation: When you start a random walk:
  - With probability  $\alpha$  you will **restart** immediately
  - With probability  $(1 - \alpha)\alpha$  you will do **one step** and then **restart**
  - With probability  $(1 - \alpha)^2\alpha$  you will do **two steps** and then **restart**
  - Etc...
- Conclusion: you are not likely to walk very far
  - On average the random walk restarts **every  $1/\alpha$  steps**

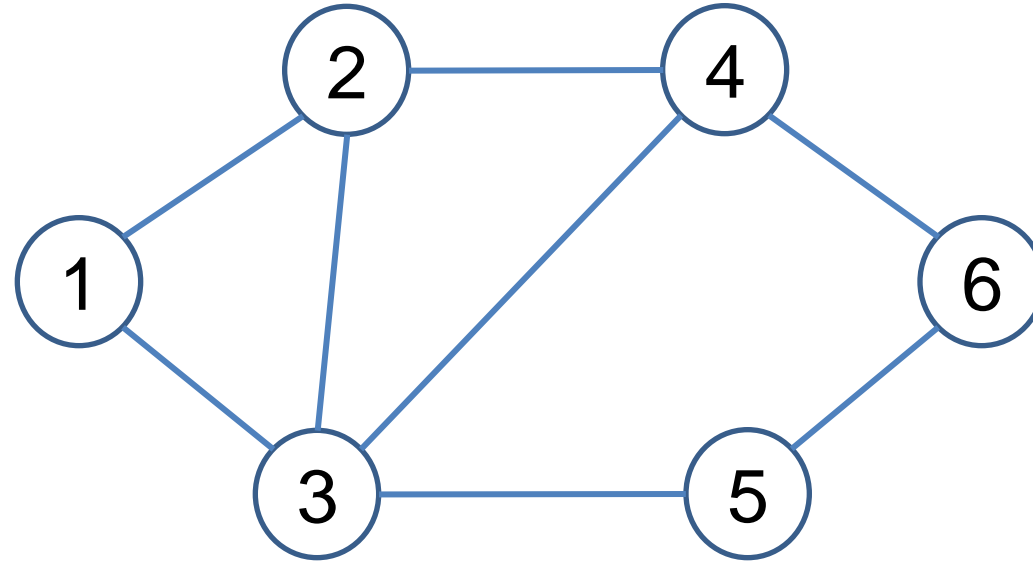


# Random walks with restarts

- With the random jump the **shorter paths** are more important, since the weight decreases **exponentially**
  - This changes the stationary distribution. When starting from some node  $x$ , nodes close to  $x$  have higher probability
- Restart vector:
  - If  $u$  is **not uniform**, we can **bias** the random walk towards the nodes that are **close** to  $u$
  - **Personalized** Pagerank:
    - Always restart to some node  $x$ , e.g., the home page of a user
  - **Topic-Specific** Pagerank
    - Restart to nodes about a specific topic, e.g., Greek pages, University home pages
      - Anti-spam
- **Random Walks with restarts** is a general technique for measuring closeness on graphs.



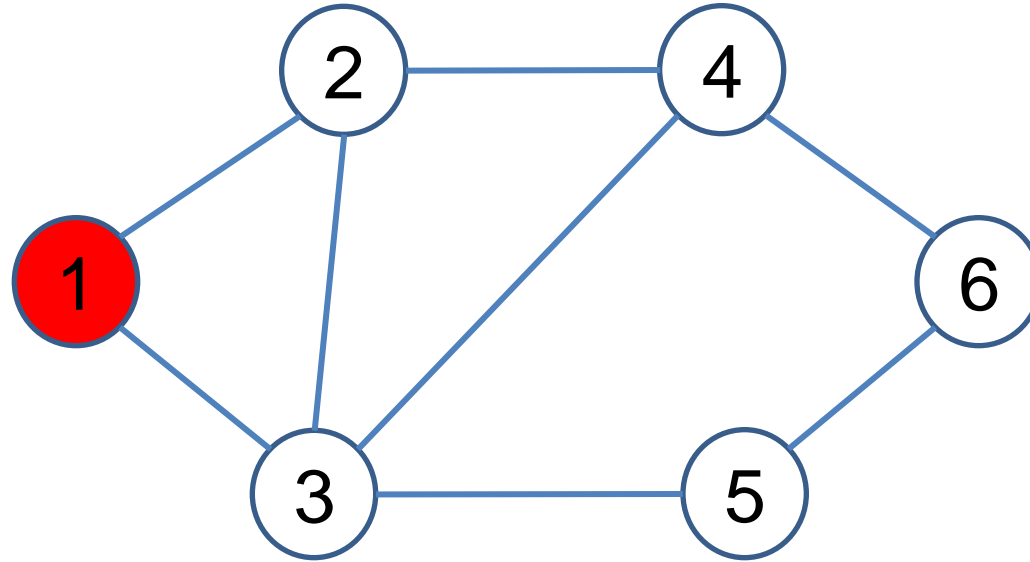
# Personalized Pagerank Example



- Global Pagerank vector (jump vector  $[1/5, 1/5, 1/5, 1/5, 1/5]$ )  
[0.13, 0.18, 0.24, 0.18, 0.13, 0.13]



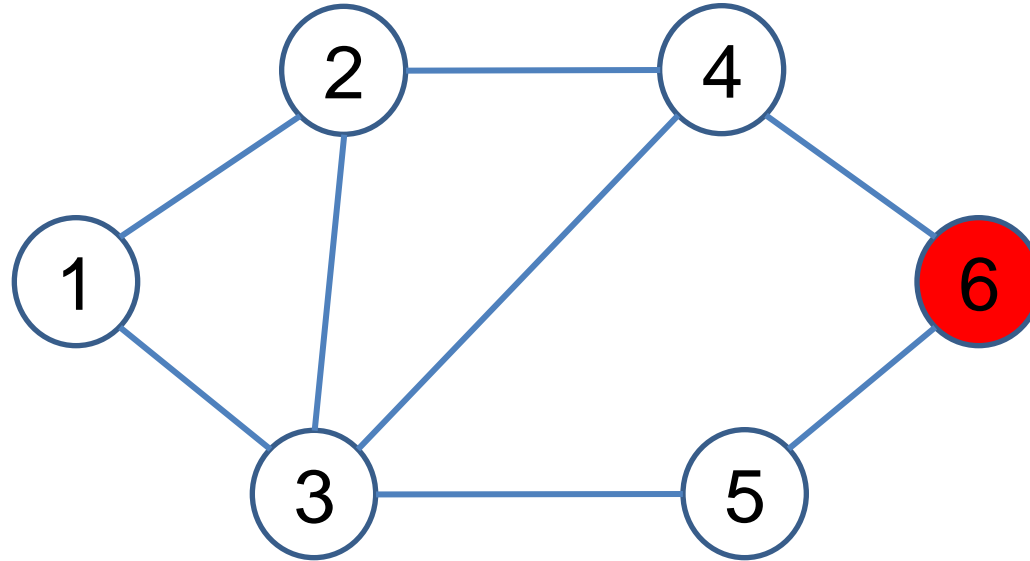
# Personalized Pagerank Example



- Global Pagerank vector (jump vector  $[1/5, 1/5, 1/5, 1/5, 1/5]$ ) :  
[0.13, 0.18, 0.24, 0.18, 0.13, 0.13]
- Personalized Pagerank for node 1 (jump vector  $[1, 0, 0, 0, 0]$ ):  
[0.26, 0.20, 0.24, 0.14, 0.08, 0.07]



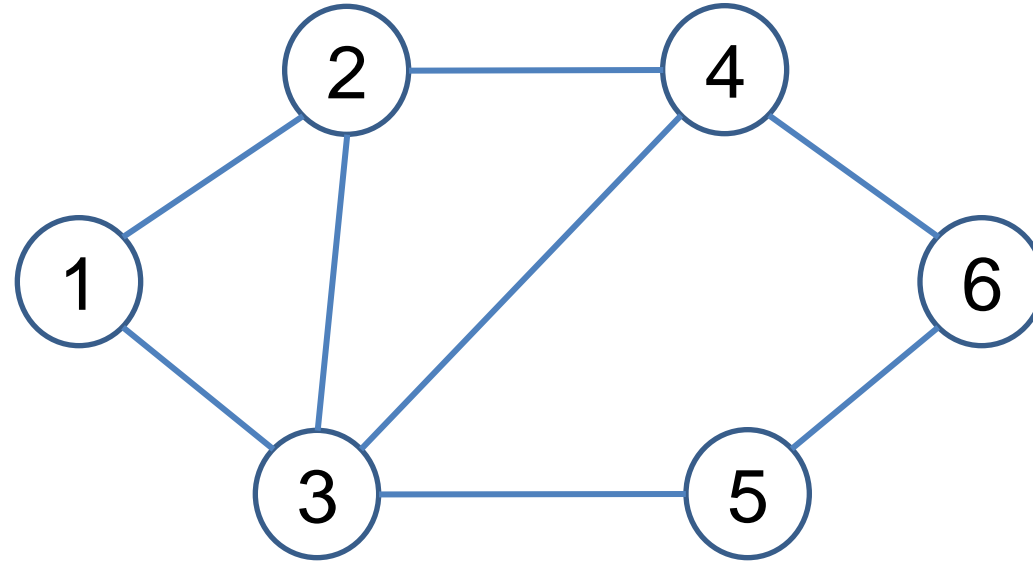
# Personalized Pagerank Example



- Global Pagerank vector (jump vector  $[1/5, 1/5, 1/5, 1/5, 1/5]$ ) :  
[0.13, 0.18, 0.24, 0.18, 0.13, 0.13]
- Personalized Pagerank from node 1 (jump vector  $[1, 0, 0, 0, 0]$ ):  
[0.26, 0.20, 0.24, 0.14, 0.08, 0.07]
- Personalized Pagerank for node 6 (jump vector  $[0, 0, 0, 0, 1]$ ):  
[0.07, 0.13, 0.19, 0.19, 0.15, 0.27]



# Personalized Pagerank Example



With  $\alpha = 0.5$

- Global Pagerank vector (jump vector  $[1/5, 1/5, 1/5, 1/5, 1/5]$ ) :  
[0.14, 0.17, 0.21, 0.18, 0.15, 0.15]
- Personalized Pagerank from node 1 (jump vector  $[1, 0, 0, 0, 0]$ ):  
[0.55, 0.17, 0.18, 0.05, 0.03, 0.02]
- Personalized Pagerank for node 6 (jump vector  $[0, 0, 0, 0, 1]$ ):  
[0.02, 0.04, 0.07, 0.16, 0.15, 0.56]



# Random walks on undirected graphs

- For **undirected** graphs, the stationary distribution is **proportional to the degrees** of the nodes
  - Thus in this case a random walk is the **same as degree popularity**
- This is **no longer true** if we do **random jumps**
  - Now the short paths play a greater role, and the previous distribution does not hold.



# Pagerank implementation

- Store the graph in adjacency list, or list of edges
- Keep current pagerank values and new pagerank values
- Go through edges and update the values of the destination nodes.
- Repeat until the difference ( $L_1$  or  $L_\infty$  difference) is below some small value  $\epsilon$ .



# A (Matlab-friendly) PageRank algorithm

- Performing vanilla power method is now too expensive – the matrix is not sparse

```
q0 = u
t = 1
repeat
  qt = (P'')T qt-1
  δ = ||qt - qt-1||
  t = t + 1
until δ < ε
```

Efficient computation of  $y = (P'')^T x$

$$\begin{aligned} y &= (1 - \alpha) P^T x \\ \beta &= \|x\|_1 - \|y\|_1 \\ y &= y + \beta u \end{aligned}$$

$P$  = normalized adjacency matrix

$P' = P + d u^T$ , where  $d_i$  is 1 if  $i$  is sink and 0 o.w.

$P'' = (1 - \alpha) P' + \alpha \mathbf{1} u^T$ , where  $\mathbf{1}$  is the vector of all 1s



# Pagerank history

- Huge advantage for Google in the early days
  - It gave a way to get an idea for the **value of a page**, which was useful in many different ways
    - Put an **order to the web**.
  - After a while it became clear that the anchor text was probably more important for ranking
  - Also, **link spam** became a new (dark) art
- Flood of research
  - Numerical analysis got rejuvenated
  - Huge number of variations
  - **Efficiency** became a great issue.
  - Huge number of applications in different fields
    - Random walk is often referred to as PageRank.



# THE HITS ALGORITHM

---



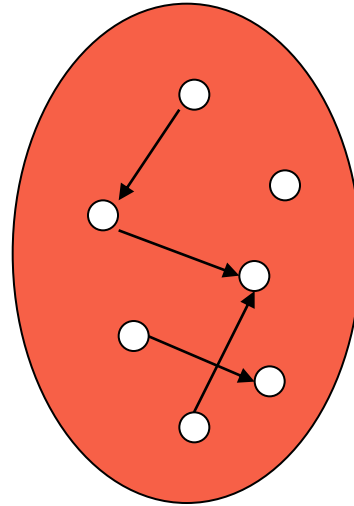
# The HITS algorithm

- Another algorithm proposed around the same time as Pagerank for using the hyperlinks to rank pages
  - Kleinberg: then an intern at IBM Almaden
  - IBM never made anything out of it



# Query dependent input

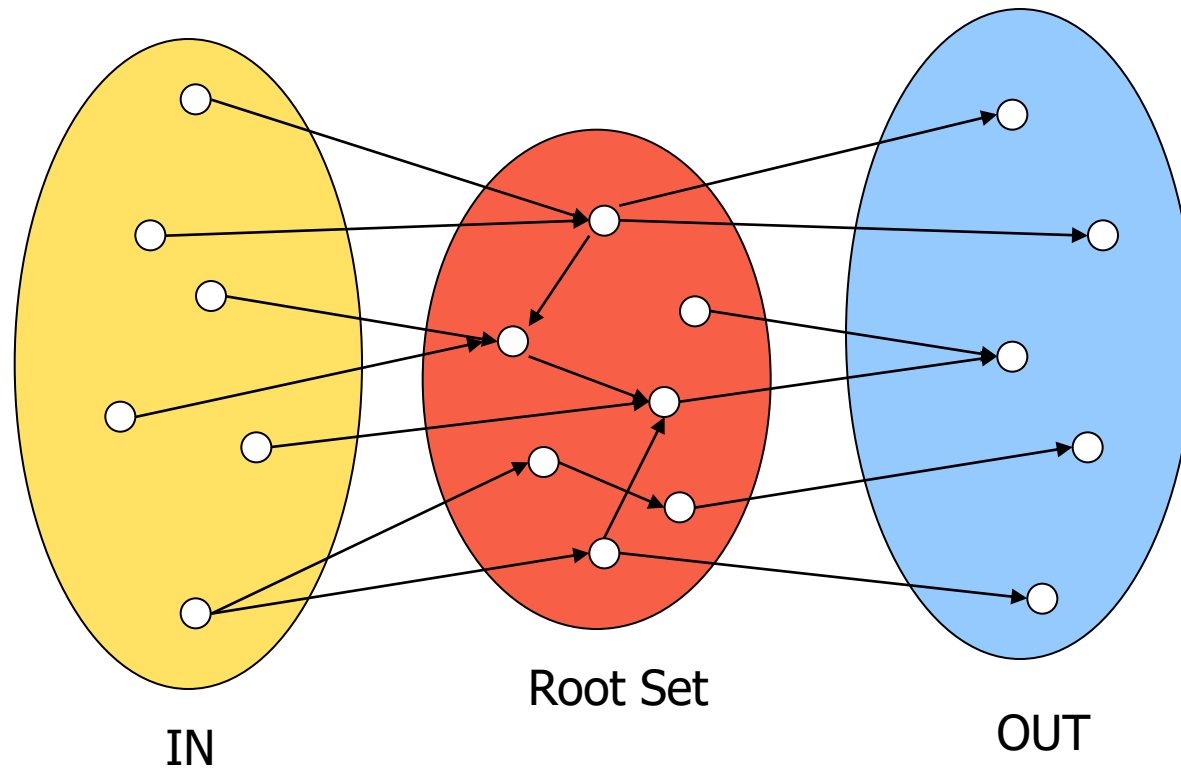
Root set obtained from a text-only search engine



Root Set

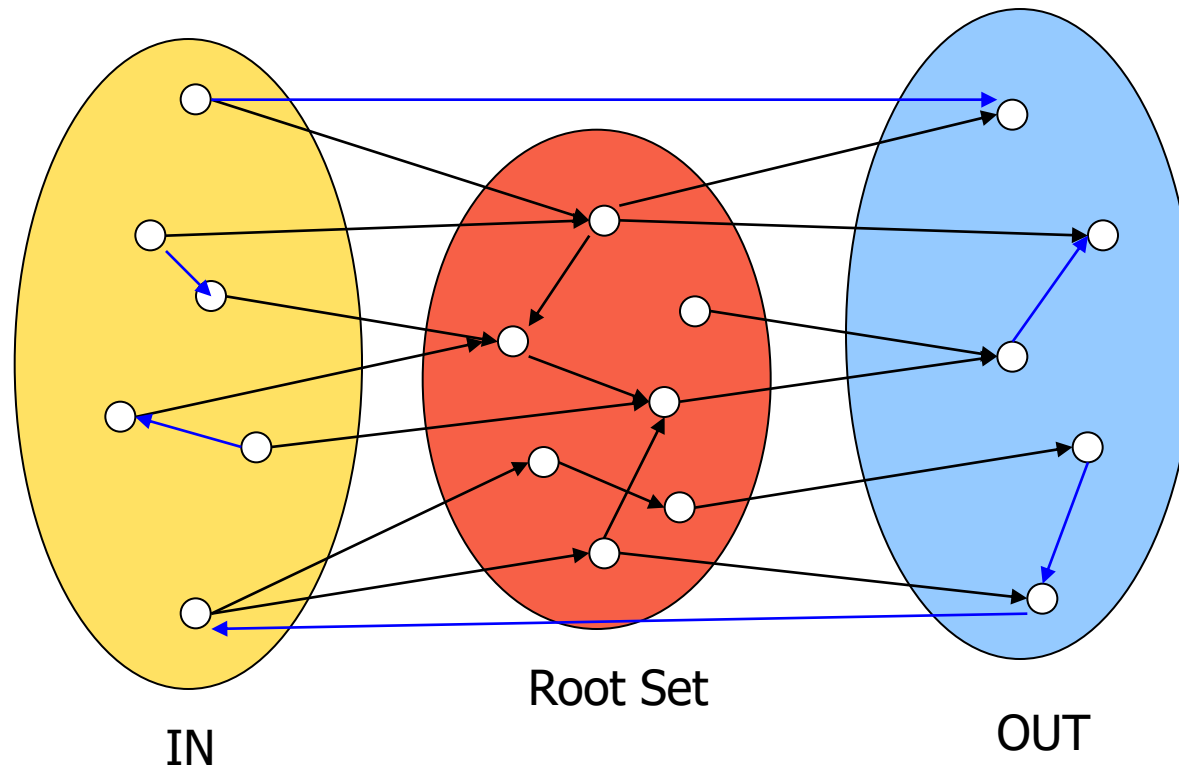


# Query dependent input



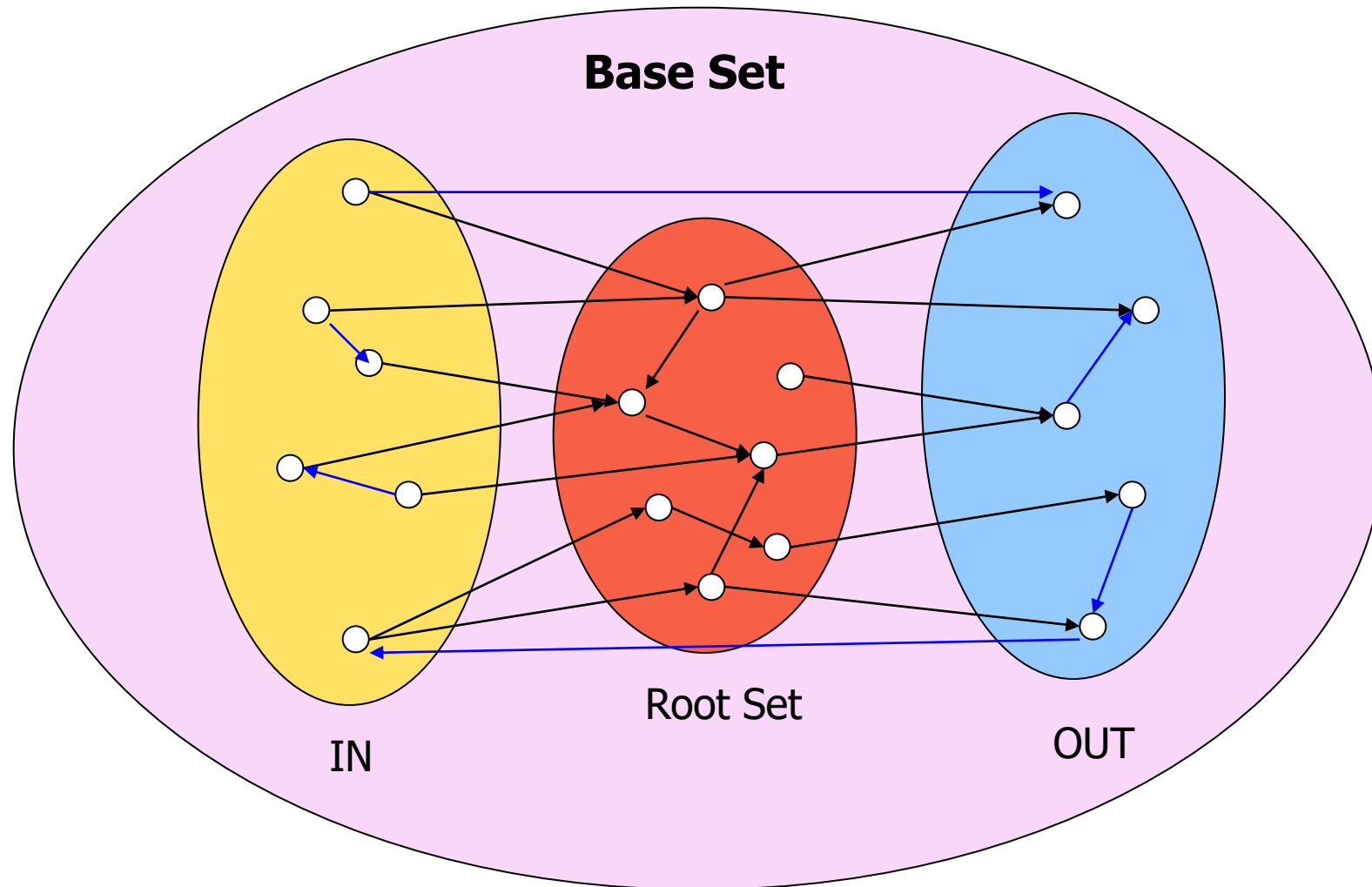


# Query dependent input





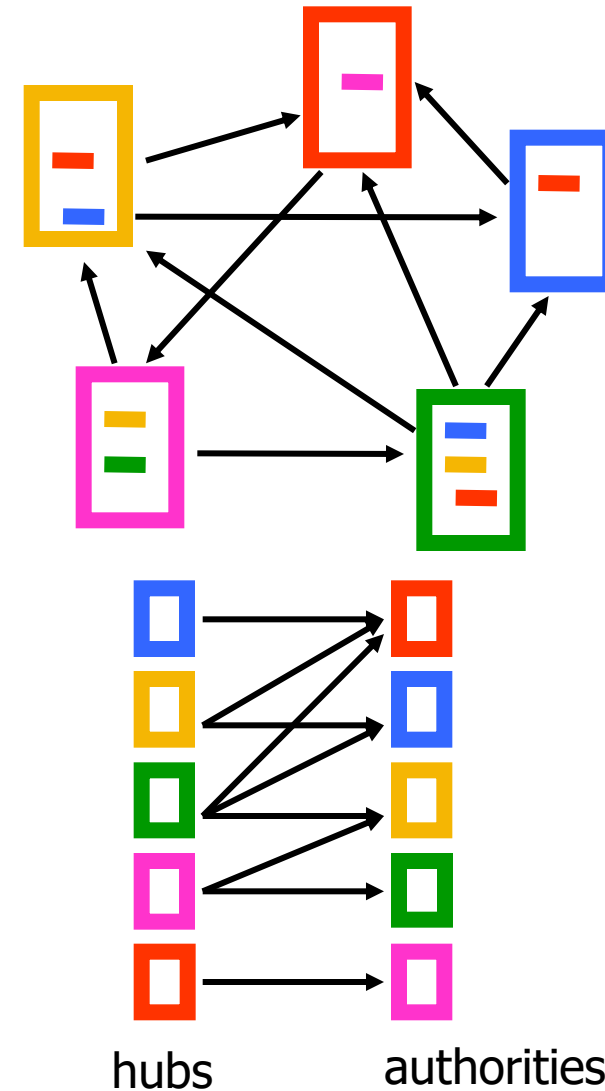
# Query dependent input





# Hubs and Authorities [K98]

- Authority is not necessarily transferred directly between authorities
- Pages have double identity
  - **hub** identity
  - **authority** identity
- **Good** hubs point to **good** authorities
- **Good** authorities are pointed by **good** hubs





# Hubs and Authorities

- Two kind of weights:
  - Hub weight
  - Authority weight
- The hub weight is the sum of the authority weights of the authorities pointed to by the hub
- The authority weight is the sum of the hub weights that point to this authority.



# HITS Algorithm

- Initialize all weights to 1.
- Repeat until convergence
  - *O* operation : hubs collect the weight of the authorities

$$h_i = \sum_{j:i \rightarrow j} a_j$$

- *I* operation: authorities collect the weight of the hubs

$$a_i = \sum_{j:j \rightarrow i} h_j$$

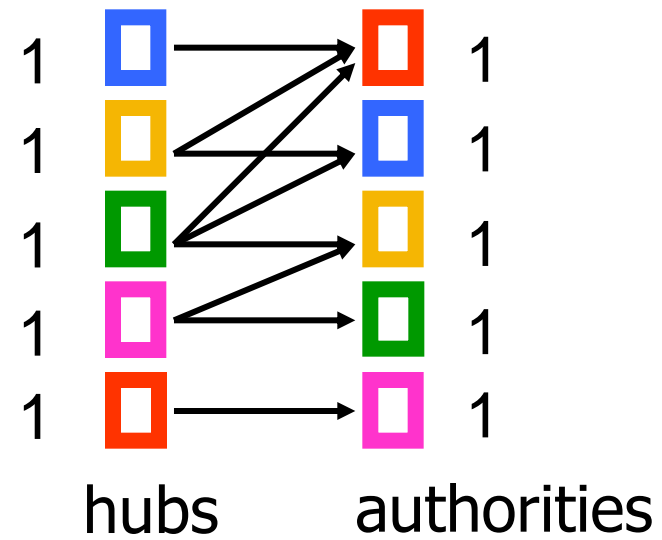
- Normalize weights under some norm

The order of updates does not matter after many iterations.  
We can also use the previous hub vector for the authority update



# Example

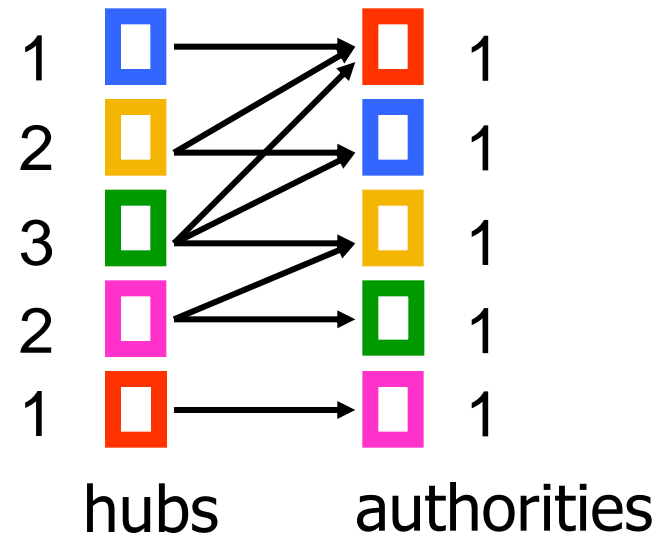
Initialize





# Example

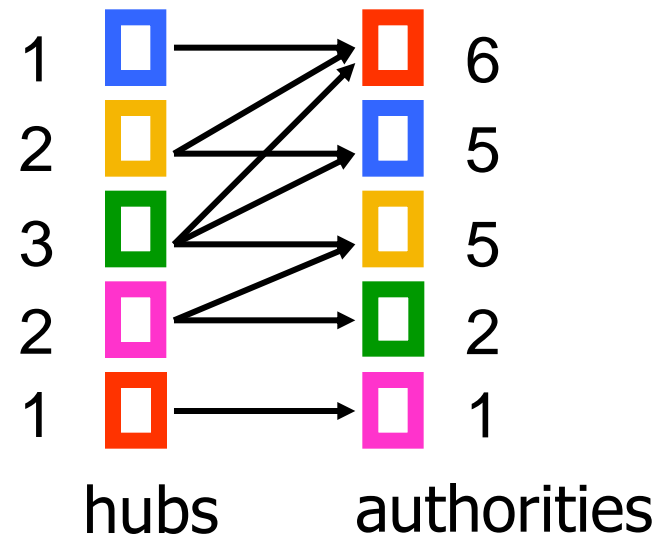
Step 1: O operation





# Example

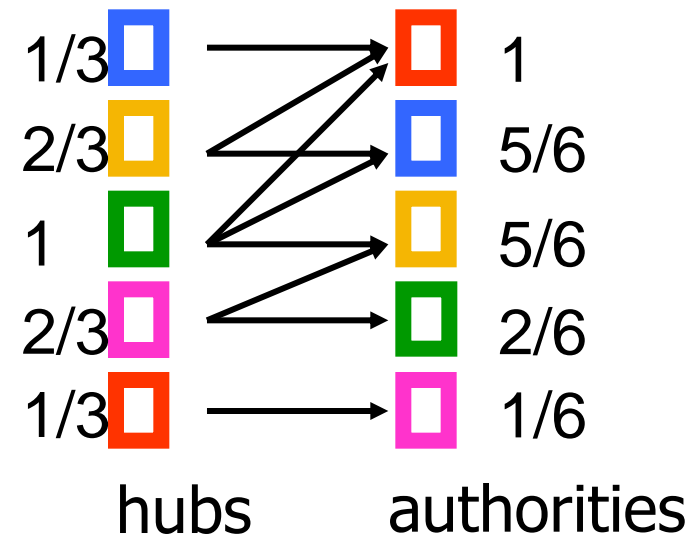
Step 1: I operation





# Example

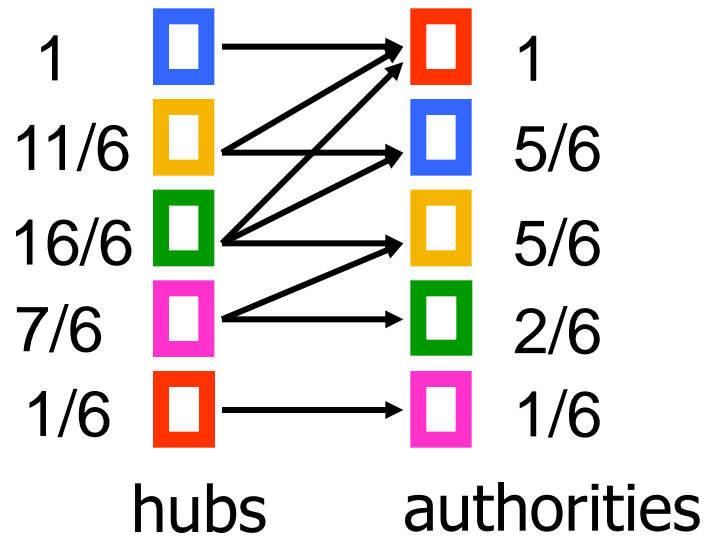
Step 1: Normalization (Max norm)





# Example

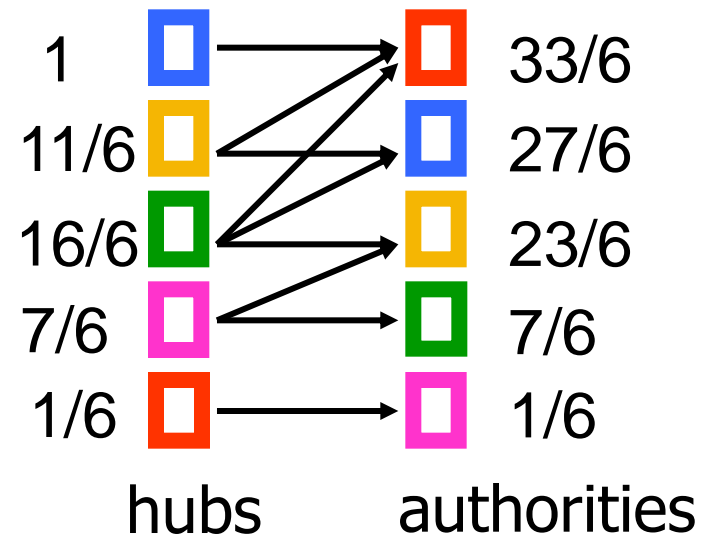
Step 2: O step





# Example

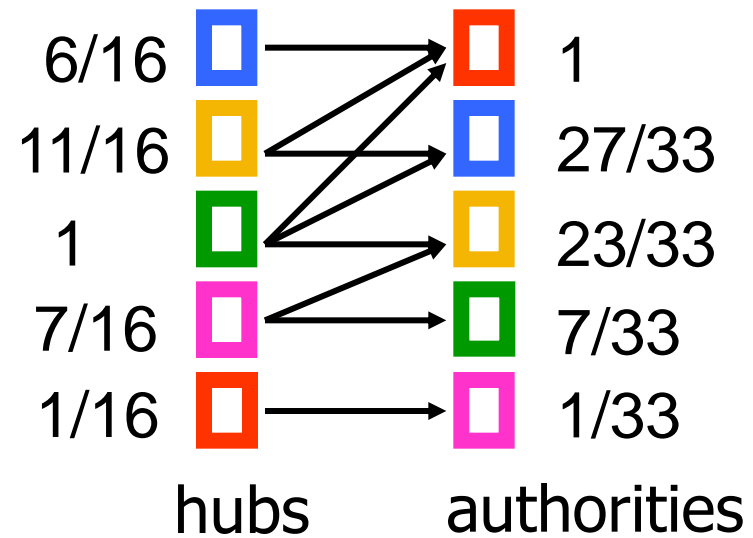
Step 2: 1 step





# Example

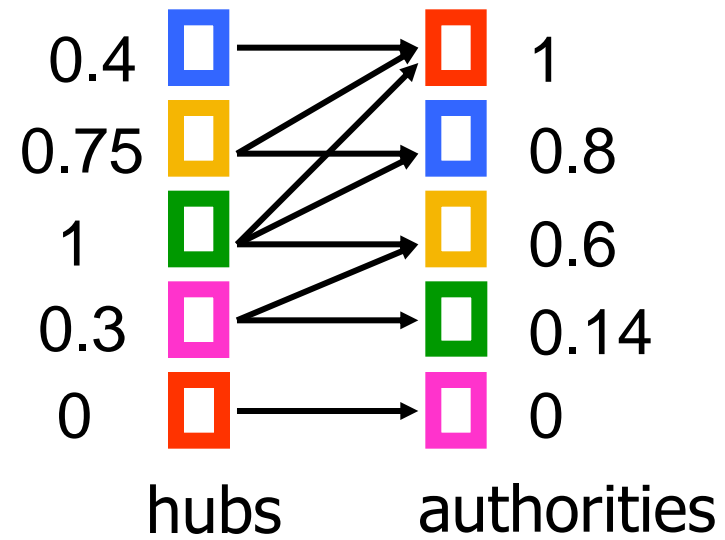
## Step 2: Normalization





# Example

Convergence





# HITS and eigenvectors

- The HITS algorithm is a **power-method** eigenvector computation
- In vector terms
  - $a^t = A^T h^{t-1}$  and  $h^t = A a^{t-1}$
  - $a^t = A^T A a^{t-1}$  and  $h^t = A A^T h^{t-1}$
  - Repeated iterations will converge to the eigenvectors
- The **authority** weight vector  $a$  is the **eigenvector** of  $A^T A$
- The **hub** weight vector  $h$  is the **eigenvector** of  $A A^T$
- The vectors  $a$  and  $h$  are called the **singular vectors** of the matrix  $A$



# Singular Value Decomposition

$$A = U \Sigma V^T = \begin{matrix} & \begin{bmatrix} \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_r \end{bmatrix} \end{matrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_r \end{bmatrix}$$

$[n \times r] \quad [r \times r] \quad [r \times n]$

- $r$  : rank of matrix  $A$
- $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$  : singular values (square roots of eig-vals  $AA^T, A^TA$ )
- $\vec{u}_1, \vec{u}_2, \dots, \vec{u}_r$  : left singular vectors (eig-vectors of  $AA^T$ )
- $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_r$  : right singular vectors (eig-vectors of  $A^TA$ )

- $$A = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \dots + \sigma_r \vec{u}_r \vec{v}_r^T$$



# Why does the Power Method work?

- If a matrix  $R$  is **real and symmetric**, it has real eigenvalues and eigenvectors:  $(\lambda_1, w_1), (\lambda_2, w_2), \dots, (\lambda_r, w_r)$ 
  - $r$  is the rank of the matrix
  - $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_r|$
- For any matrix  $R$ , the eigenvectors  $w_1, w_2, \dots, w_r$  of  $R$  define **a basis of the vector space**
  - For any vector  $x$ ,  $Rx = \alpha_1 w_1 + \alpha_2 w_2 + \dots + \alpha_r w_r$
- After  $t$  multiplications we have:
$$R^t x = \lambda_1^{t-1} \alpha_1 w_1 + \lambda_2^{t-1} \alpha_2 w_2 + \dots + \lambda_r^{t-1} \alpha_r w_r$$
- Normalizing leaves only the term  $w_1$ .



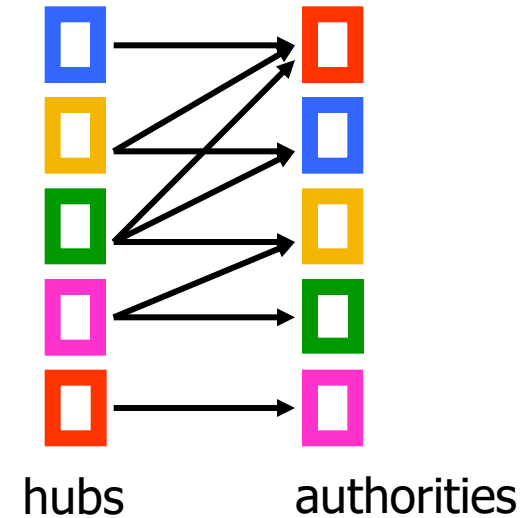
# OTHER ALGORITHMS

---



# The SALSA algorithm

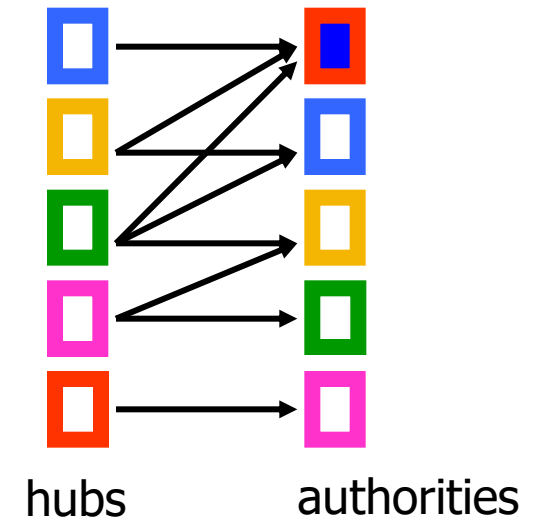
- Perform a random walk on the bipartite graph of hubs and authorities alternating between the two
- What does this random walk converges to?





# The SALSA algorithm

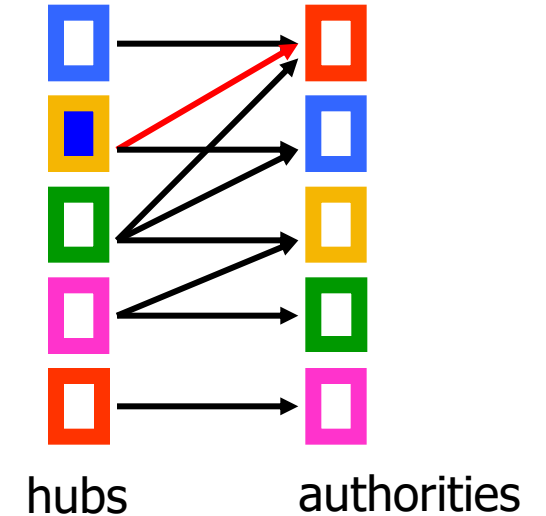
- Start from an authority chosen uniformly at random
  - e.g. the red authority





# The SALSA algorithm

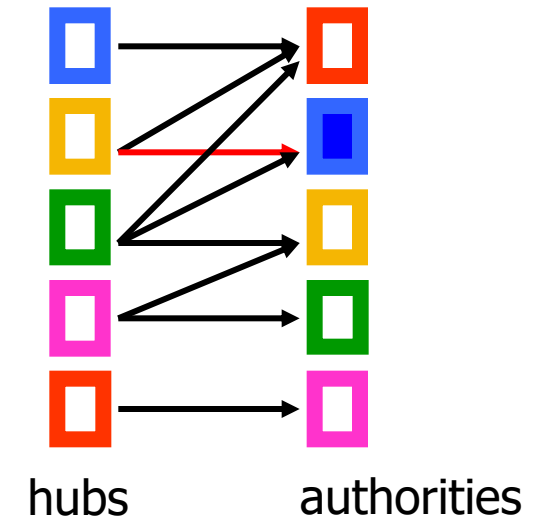
- Start from an authority chosen uniformly at random
  - e.g. the red authority
- Choose one of the in-coming links uniformly at random and move to a hub
  - e.g. move to the yellow authority with probability  $1/3$





# The SALSA algorithm

- Start from an authority chosen uniformly at random
  - e.g. the red authority
- Choose one of the in-coming links uniformly at random and move to a hub
  - e.g. move to the yellow authority with probability  $1/3$
- Choose one of the out-going links uniformly at random and move to an authority
  - e.g. move to the blue authority with probability  $1/2$





# The SALSA algorithm

- Formally we have probabilities:
  - $a_i$ : probability of being at authority  $i$
  - $h_j$ : probability of being at hub  $j$
- The probability of being at authority  $i$  is computed as:

$$a_i = \sum_{j \in N_{in}(i)} \frac{1}{d_{out}(j)} h_j$$

- The probability of being at hub  $j$  is computed as

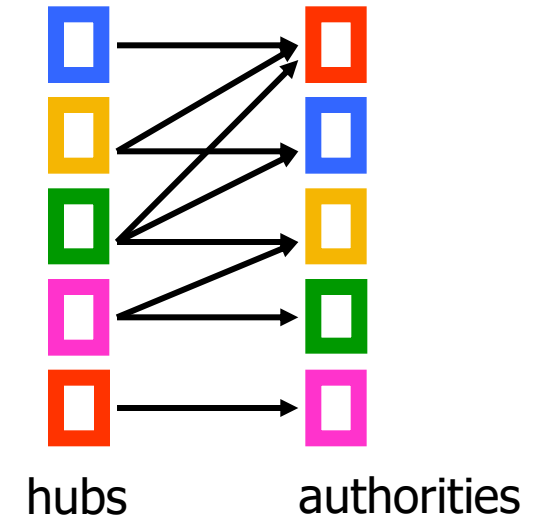
$$h_j = \sum_{i \in N_{out}(j)} \frac{1}{d_{in}(i)} a_i$$

- Repeated computation converges



# The SALSA algorithm

- In matrix terms
  - $A_c$  = the matrix  $A$  where columns are normalized to sum to 1
  - $A_r$  = the matrix  $A$  where rows are normalized to sum to 1
- The hub computation
  - $h = A_c a$
- The authority computation
  - $a = A_r^T h = A_r^T A_c a$
- In MC terms the transition matrix
  - $P = A_r A_c^T$



$$h_2 = 1/3 a_1 + 1/2 a_2$$

$$a_1 = h_1 + 1/2 h_2 + 1/3 h_3$$



# Social network analysis

- Evaluate the **centrality** of individuals in social networks
  - **degree centrality**
    - the (weighted) degree of a node
  - **distance centrality**
    - the average (weighted) distance of a node to the rest in the graph

$$D_c(v) = \frac{1}{\sum_{u \neq v} d(v, u)}$$

- **betweenness centrality**
  - the average number of (weighted) shortest paths that use node  $v$

$$B_c(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$



# Counting paths – Katz 53

- The importance of a node is measured by the weighted sum of paths that lead to this node
- $A^m[i,j]$  = number of paths of length  $m$  from  $i$  to  $j$
- Compute

$$P = bA + b^2A^2 + \dots + b^mA^m + \dots = (I - bA)^{-1} - I$$

- converges when  $b < \lambda_1(A)$
- Rank nodes according to the column sums of the matrix  $P$



# Bibliometrics

- Impact factor (E. Garfield 72)
  - counts the number of citations received for papers of the journal in the previous two years
- Pinsky-Narin 76
  - perform a random walk on the set of journals
  - $P_{ij}$  = the fraction of citations from journal i that are directed to journal j



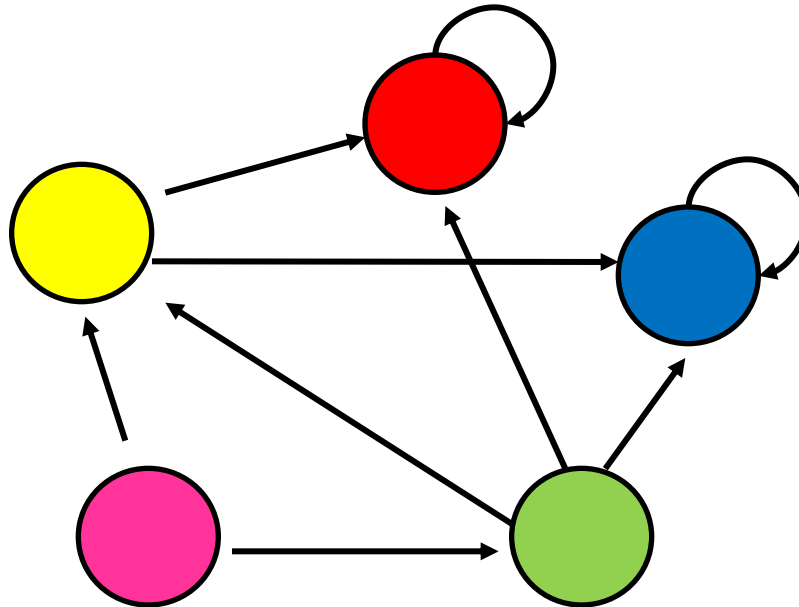
# ABSORBING RANDOM WALKS

---



# Random walk with absorbing nodes

- **Absorbing nodes**: nodes from which the random walk cannot escape.

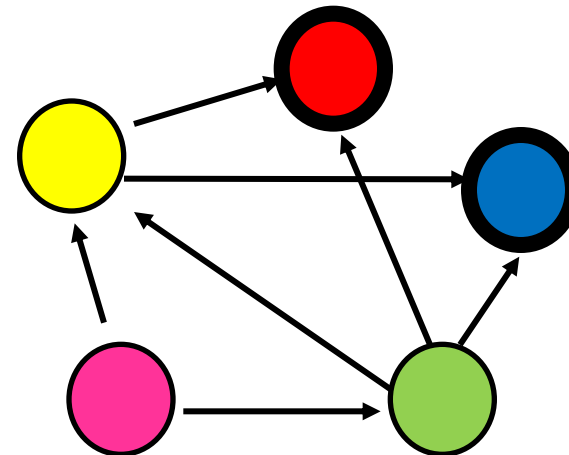


- Two absorbing nodes: the red and the blue.



# Absorption probability

- In a graph with more than one **absorbing nodes** a random walk that starts from a **non-absorbing (transient)** node **t** will be absorbed in one of them with some probability
  - For node **t** we can compute the **probabilities of absorption**





# Absorption probability

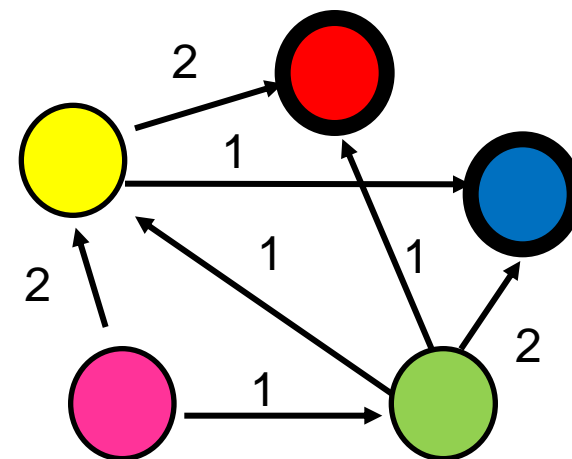
- Computing the probability of being absorbed:
  - The **absorbing nodes** have probability 1 of being absorbed in themselves and zero of being absorbed in another node.
  - For the **non-absorbing nodes**, take the (weighted) average of the absorption probabilities of your neighbors
    - if one of the neighbors is the absorbing node, it has probability 1
  - Repeat until convergence (= very small change in probs)

$$P(\text{Red}|\text{Pink}) = \frac{2}{3}P(\text{Red}|\text{Yellow}) + \frac{1}{3}P(\text{Red}|\text{Green})$$

$$P(\text{Red}|\text{Green}) = \frac{1}{4}P(\text{Red}|\text{Yellow}) + \frac{1}{4}$$

$$P(\text{Red}|\text{Yellow}) = \frac{2}{3}$$

$$P(\text{Red}|\text{Red}) = 1, P(\text{Red}|\text{Blue}) = 0$$





# Absorption probability

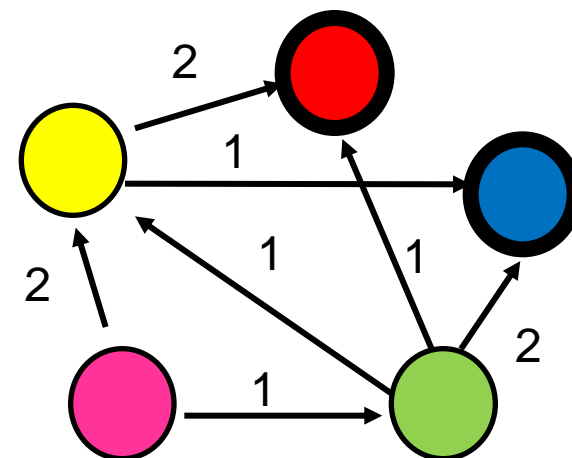
- Computing the probability of being absorbed:
  - The **absorbing nodes** have probability 1 of being absorbed in themselves and zero of being absorbed in another node.
  - For the **non-absorbing nodes**, take the (weighted) average of the absorption probabilities of your neighbors
    - if one of the neighbors is the absorbing node, it has probability 1
  - Repeat until convergence (= very small change in probs)

$$P(\text{Blue}|\text{Pink}) = \frac{2}{3}P(\text{Blue}|\text{Yellow}) + \frac{1}{3}P(\text{Blue}|\text{Green})$$

$$P(\text{Blue}|\text{Green}) = \frac{1}{4}P(\text{Blue}|\text{Yellow}) + \frac{1}{2}$$

$$P(\text{Blue}|\text{Yellow}) = \frac{1}{3}$$

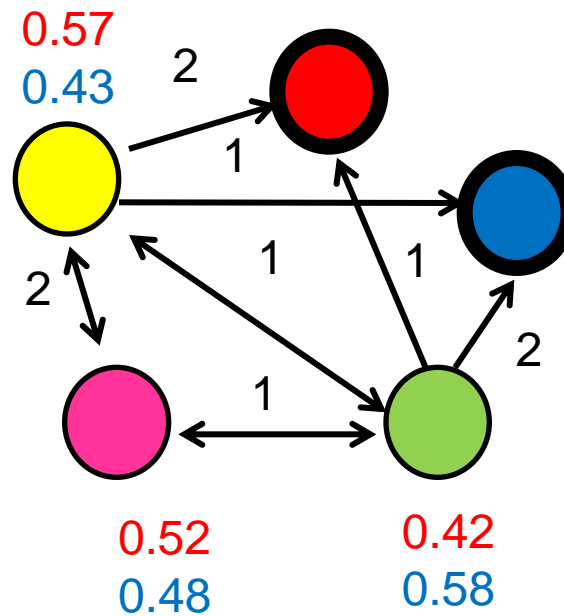
$$P(\text{Blue}|\text{Blue}) = 1, P(\text{Blue}|\text{Red}) = 0$$





# Absorption probabilities

- The absorption probabilities for red and blue





# Absorption probabilities

- The absorption probability has several practical uses.
- Given a graph (**directed** or **undirected**) we can choose to **make** some nodes **absorbing**.
  - Simply **direct** all edges incident on the chosen nodes towards them and create a self-loop.
- The absorbing random walk provides a measure of **proximity** of transient nodes to the chosen nodes.
  - Useful for **understanding** proximity in graphs
  - Useful for **propagation** in the graph
    - E.g, on a social network some nodes are **malicious**, while some are **certified**, to which class is a transient node closer?



# Linear Algebra

- Our matrix looks like this

$$P = \begin{bmatrix} P_{TT} & P_{TA} \\ 0 & I \end{bmatrix}$$

- $P_{TT}$ : transition probabilities **between transient nodes**
- $P_{TA}$ : transition probabilities **from transient to absorbing nodes**
- Computing the absorption probabilities corresponds to iteratively multiplying matrix  $P$  with itself



# Linear algebra

- The fundamental matrix

$$F = P_{TT} + P_{TT}^2 + \dots = \sum_{i=1}^{\infty} P_{TT}^i = (1 - P_{TT})^{-1}$$

- $F[i, j]$  = The probability of being in a **transient** state  $t_j$  when starting from state  $t_i$  after infinite steps
- The transient-to-absorbing matrix  $Q$ 
$$Q = FP_{TA}$$
  - $Q[i, k]$  = The **probability** of being **absorbed** in absorbing state  $a_k$  when starting from transient state  $t_i$

$$P^{\infty} = \begin{bmatrix} 0 & Q \\ 0 & I \end{bmatrix}$$



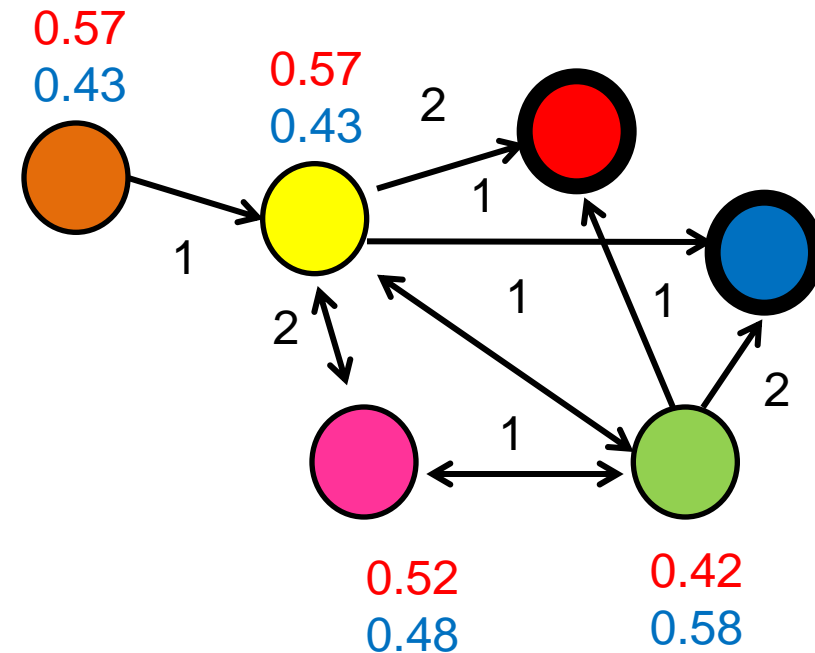
# Penalizing long paths

- The orange node has the same probability of reaching red and blue as the yellow one

$$P(\text{Red}|\text{Orange}) = P(\text{Red}|\text{Yellow})$$

$$P(\text{Blue}|\text{Orange}) = P(\text{Blue}|\text{Yellow})$$

- Intuitively though it is further away





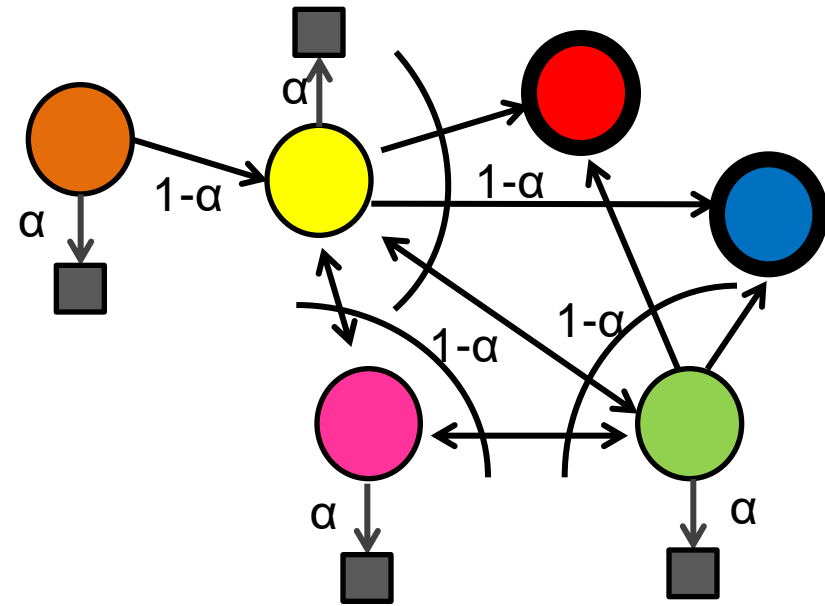
# Penalizing long paths

- Add an **universal absorbing node** to which each node gets absorbed with probability  $\alpha$ .

With probability  $\alpha$  the random walk **dies**

With probability  $(1-\alpha)$  the random walk continues as before

The longer the path from a node to an absorbing node the more likely the random walk dies along the way, the lower the absorption probability



e.g.

$$P(\text{Red}|\text{Green}) = (1 - \alpha) \left( \frac{1}{5} P(\text{Red}|\text{Yellow}) + \frac{1}{5} P(\text{Red}|\text{Pink}) + \frac{1}{5} \right)$$



# Random walk with restarts

- Adding a jump with probability  $\alpha$  to a universal absorbing node seems similar to Pagerank
- Random walk with restart:
  - Start a random walk from node  $u$
  - At every step with probability  $\alpha$ , jump back to  $u$
  - The probability of being at node  $v$  after large number of steps defines again a similarity between nodes  $u, v$
- The Random Walk With Restarts (RWS) and Absorbing Random Walk (ARW) are similar but not the same
  - RWS computes the probability of paths from the starting node  $u$  to a node  $v$ , while ARW the probability of paths from a node  $v$ , to the absorbing node  $u$ .
  - RWS defines a distribution over all nodes, while ARW defines a probability for each node
  - An absorbing node blocks the random walk, while restarts simply bias towards starting nodes
    - Makes a difference when having multiple (and possibly competing) absorbing nodes



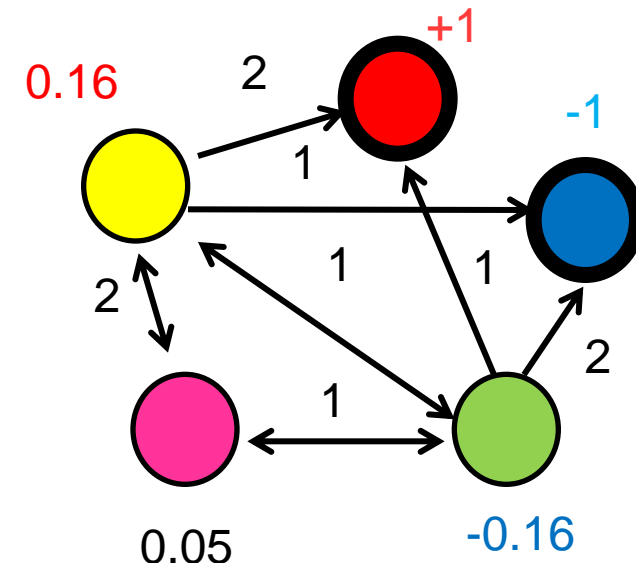
# Propagating values

- Assume that **Red** has a positive value and **Blue** a negative value
  - Positive/Negative **class**, Positive/Negative **opinion**
- We can compute a value for all the other nodes by repeatedly **averaging** the values of the neighbors
  - The value of node **u** is the **expected** value at the point of absorption for a random walk that starts from **u**

$$V(\text{Pink}) = \frac{2}{3}V(\text{Yellow}) + \frac{1}{3}V(\text{Green})$$

$$V(\text{Green}) = \frac{1}{5}V(\text{Yellow}) + \frac{1}{5}V(\text{Pink}) + \frac{1}{5} - \frac{2}{5}$$

$$V(\text{Yellow}) = \frac{1}{6}V(\text{Green}) + \frac{1}{3}V(\text{Pink}) + \frac{1}{3} - \frac{1}{6}$$





# Linear algebra

- Computation of values is essentially multiplication of the matrix  $Q$  with the vector of values of the absorbing nodes

$$v = Qs$$

- $s$ : vector of values of the absorbing nodes
- $v$ : vector of values of the transient nodes



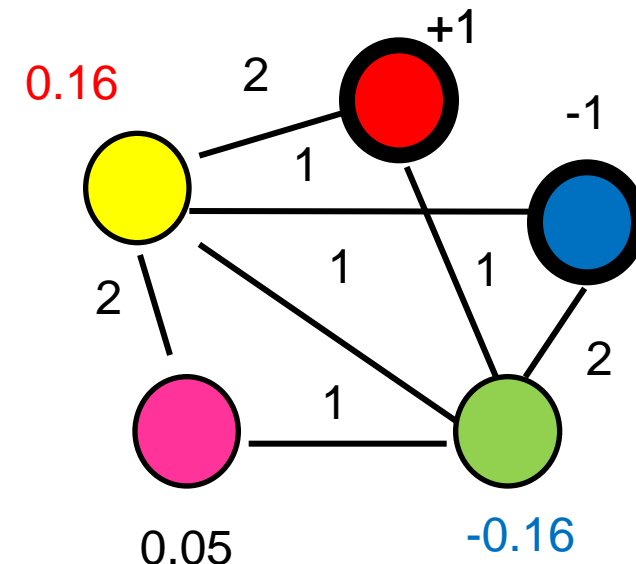
# Electrical networks and random walks

- Our graph corresponds to an **electrical network**
- There is a positive **voltage** of **+1** at the Red node, and a negative voltage **-1** at the Blue node
- There are **resistances** on the edges **inversely proportional** to the weights (or **conductance proportional** to the weights)
- The computed values are the **voltages** at the nodes

$$V(\text{Pink}) = \frac{2}{3}V(\text{Yellow}) + \frac{1}{3}V(\text{Green})$$

$$V(\text{Green}) = \frac{1}{5}V(\text{Yellow}) + \frac{1}{5}V(\text{Pink}) + \frac{1}{5} - \frac{2}{5}$$

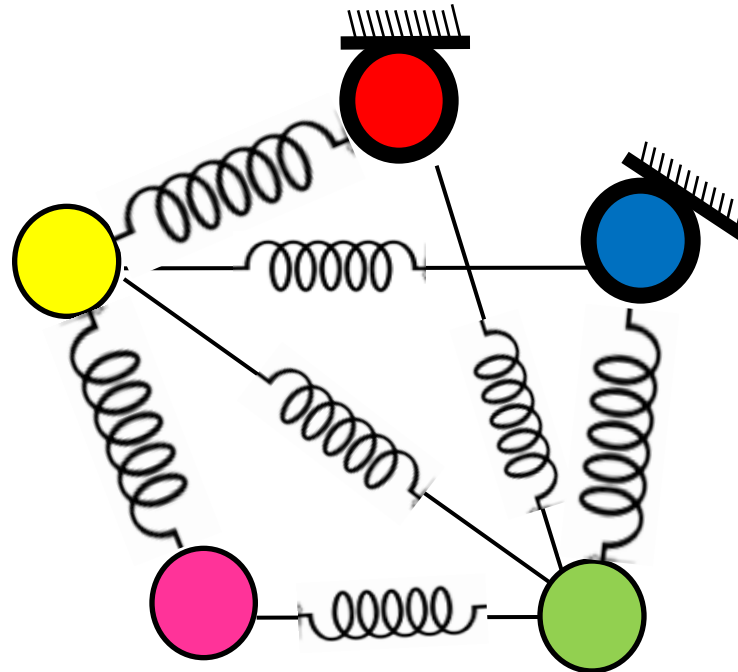
$$V(\text{Yellow}) = \frac{1}{6}V(\text{Green}) + \frac{1}{3}V(\text{Pink}) + \frac{1}{3} - \frac{1}{6}$$





# Springs and random walks

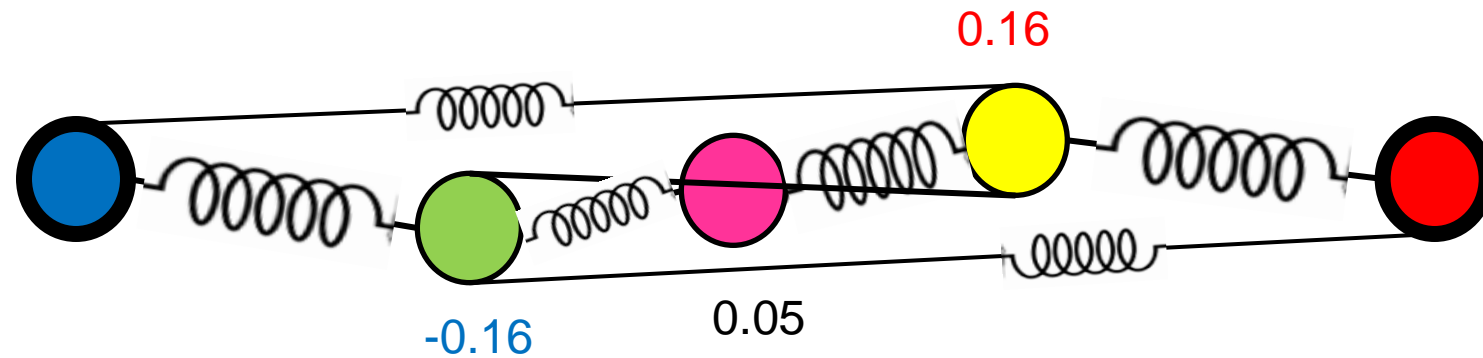
- Our graph corresponds to an **spring system**
- The Red node is pinned at position +1, while the Blue node is pinned at position -1 on a line.
- There are **springs** on the edges with hardness **proportional** to the weights
- The computed values are the **positions** of the nodes on the line





# Springs and random walks

- Our graph corresponds to an **spring system**
- The Red node is pinned at position +1, while the Blue node is pinned at position -1 on a line.
- There are **springs** on the edges with hardness **proportional** to the weights
- The computed values are the **positions** of the nodes on the line





# Opinion formation model (Friedkin and Johnsen)

- Opinions are **values** in  $[-1, +1]$
- Every user  $i$  has an **intrinsic opinion**  $s_i \in [-1, +1]$  and an **expressed opinion**  $z_i \in [-1, +1]$
- The public opinion  $z_i$  of each user in the network is **iteratively** updated, each time taking the **average** of the **expressed opinions** of its neighbors and the **intrinsic opinion** of herself

$$z_i^t = \frac{w_{ii}s_i + \sum_{j \in N(i)} w_{ij}z_j^{t-1}}{w_{ii} + \sum_{j \in N(i)} w_{ij}}$$



# Opinion formation as a game

- Assume that network users are **rational** (selfish) agents
- Each user has a **personal cost** for expressing an opinion

$$c(z_i) = w_{ii}(z_i - s_i)^2 + \sum_{j \in N(i)} w_{ij}(z_i - z_j)^2$$

**Inconsistency cost:** The cost for **deviating** from one's intrinsic opinion

**Conflict cost:** The cost for **disagreeing** with the opinions in one's social network

- Each user is selfishly trying to minimize her personal cost.

D. Bindel, J. Kleinberg, S. Oren. *How Bad is Forming Your Own Opinion?*  
Proc. 52nd IEEE Symposium on Foundations of Computer Science, 2011.



# Opinion formation as a game

- The opinion  $z_i$  that **minimizes** the personal cost of user  $i$

$$z_i = \frac{w_{ij}s_i + \sum_{j \in N(i)} w_{ij}z_j}{w_{ij} + \sum_{j \in N(i)} w_{ij}}$$

- In linear algebra terms (assume 0/1 weights):

$$(L + I)\mathbf{z} = \mathbf{s} \Rightarrow \mathbf{z} = (L + I)^{-1}\mathbf{s}$$

where  $L$  is the **Laplacian** of the graph.

- The **Laplacian** is the negated adjacency matrix with the degree on the diagonal

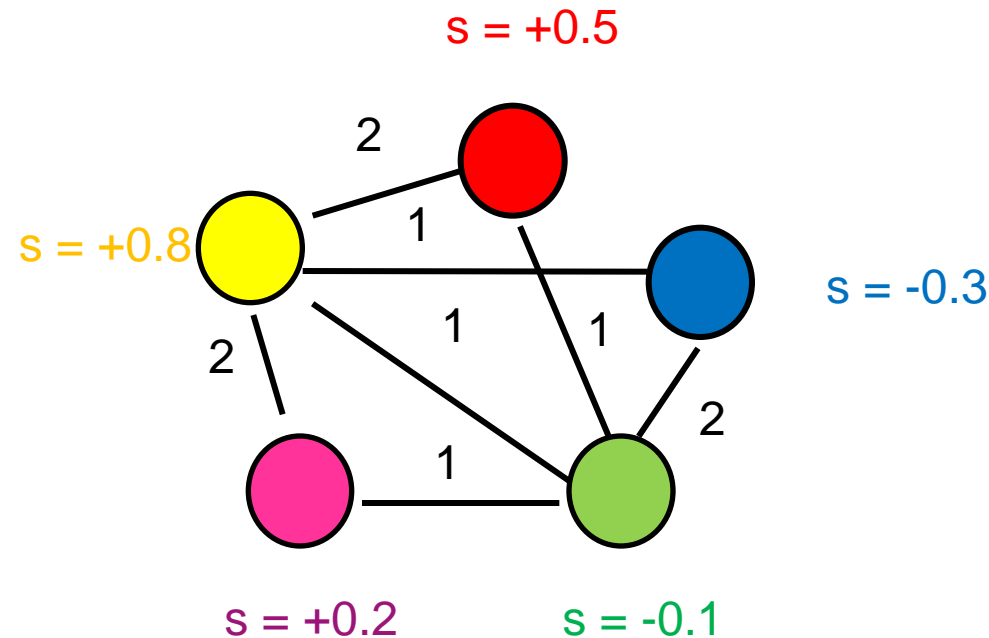
$$L = D - A$$

- $D$ : diagonal matrix with the degrees on the diagonal



# Example

- Social network with **internal opinions**



- There is a connection of this model with absorbing random walks and value propagation – can you see it?



# Opinion formation and absorbing random walks

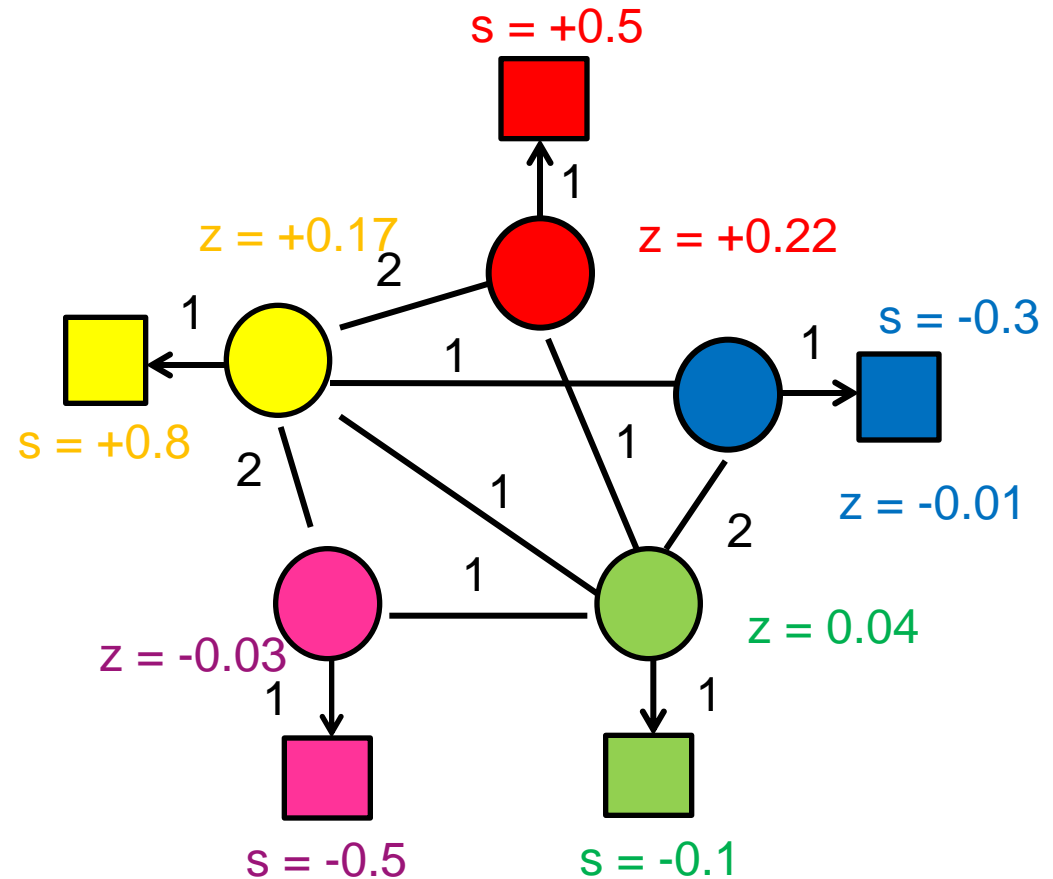
One absorbing node per user with value the **intrinsic opinion** of the user

One transient node per user that links to her absorbing node and the transient nodes of her neighbors

The **expressed opinion** for each node is computed using the value propagation we described

- Repeated averaging

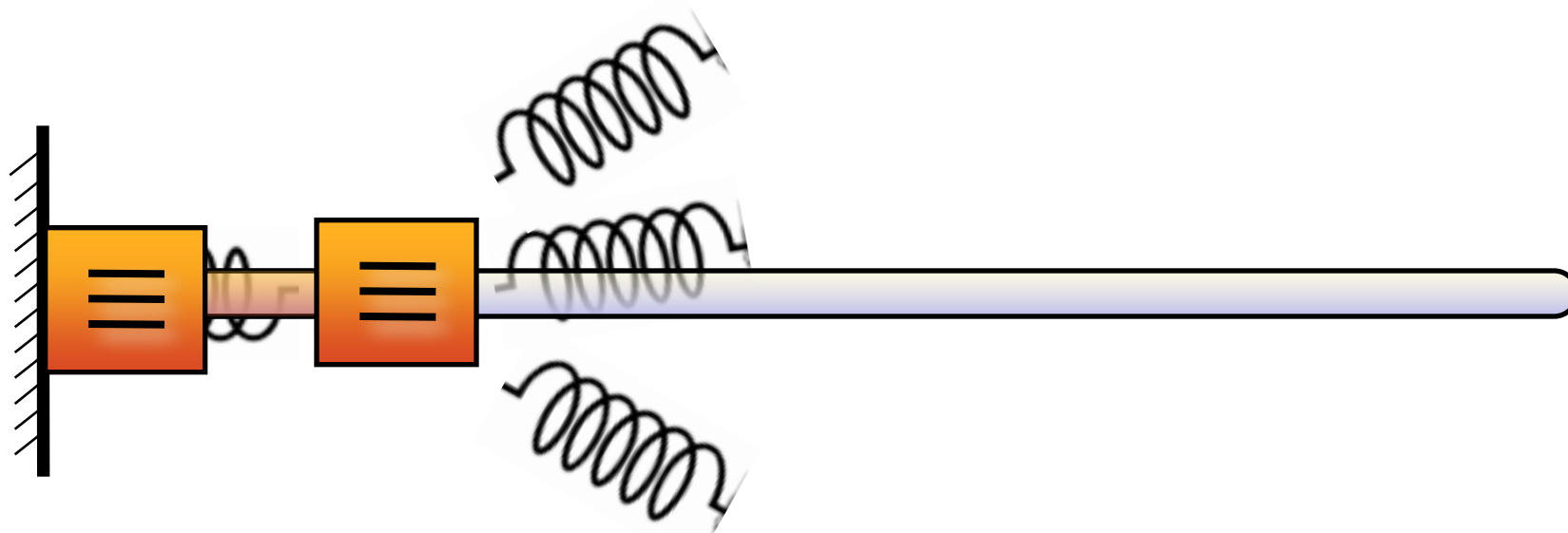
It is equal to the **expected intrinsic opinion** at the place of absorption



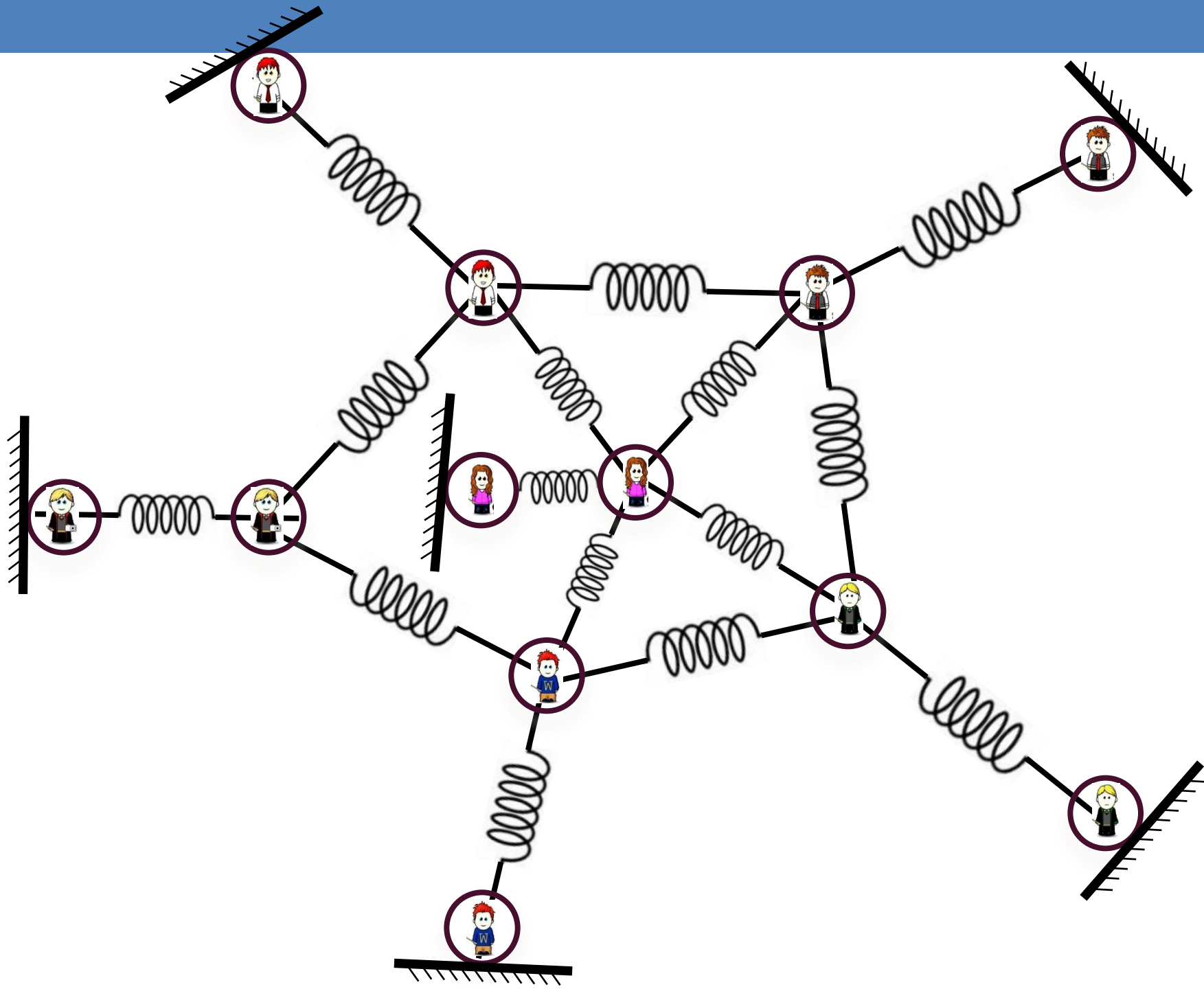


# Opinion of a user

- For an individual user  $u$ 
  - $u$ 's absorbing node is a stationary point
  - $u$ 's transient node is connected to the absorbing node with a spring.
  - The neighbors of  $u$  pull with their own springs.









# Hitting time

- A related quantity: **Hitting time**  $H(u,v)$ 
  - The **expected number of steps** for a random walk starting from node  $u$  to end up in  $v$  **for the first time**
    - Make node  $v$  absorbing and compute the expected number of steps to reach  $v$
    - Assumes that the graph is strongly connected, and there are no other absorbing nodes.
- **Commute time**  $H(u,v) + H(v,u)$ : often used as a **distance metric**
  - Proportional to the **total resistance** between nodes  $u$ , and  $v$



# Transductive learning

- If we have a graph of relationships and some **labels** on some nodes we can **propagate** them to the remaining nodes
  - Make the labeled nodes to be absorbing and compute the probability for the rest of the graph
  - E.g., a social network where some people are tagged as spammers
  - E.g., the movie-actor graph where some movies are tagged as action or comedy.
- This is a form of **semi-supervised learning**
  - We make use of the unlabeled data, and the relationships
- It is also called **transductive learning** because it does not produce a model, but just labels the unlabeled data that is at hand.
  - Contrast to **inductive learning** that learns a model and can label any new example



# Implementation details

- Implementation is in many ways similar to the PageRank implementation
  - For an edge  $(u, v)$  instead of updating the value of  $v$  we update the value of  $u$ .
    - The value of a node is the average of its neighbors
  - We need to check for the case that a node  $u$  is absorbing, in which case the value of the node is not updated.
  - Repeat the updates until the change in values is very small.