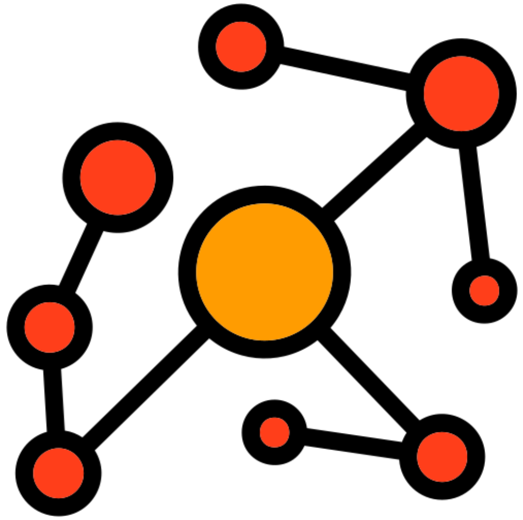
UNIVERSITÀ DEL SALENTO

Link Analysis



Graphs: a gentle introduction

Un **grafo** è una struttura dati definita come

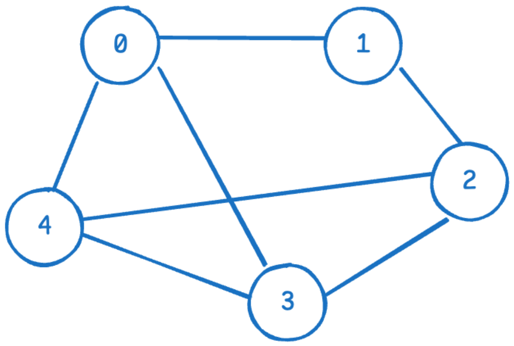
$$G = (V, E),$$

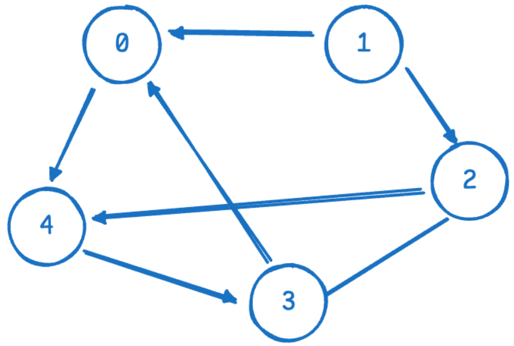
dove:

- V: insieme dei **nodi** o **vertici**, con $|V| = n$
- E: insieme degli **archi**, con $|E| = m$, dove un **arco** è una coppia $(v1, v2)$ di nodi in V
  - Se si può andare solo da u (**sorgente**) a v (**destinazione**), (u,v), il grafo è detto **diretto** o **orientato**
  - Se si può andare in entrambe le direzioni {v1,v2}, il grafo è detto **indiretto** o **non orientato**

La **dimensione** del grafo G è data da $n + m$.

G = (
[v0,v1, v2, v3, v4],
[(v0,v1), (v0,v3),(v0,v4),(v1,v2),(v2,v3),(v2,v4), (v3,v4)]
)

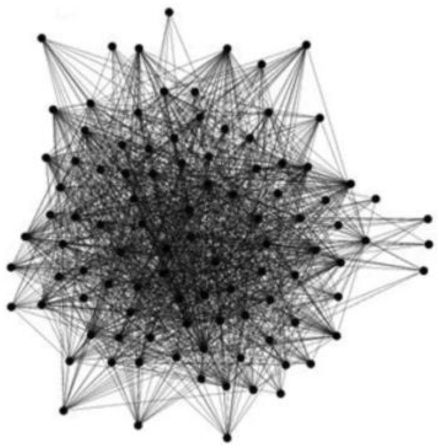


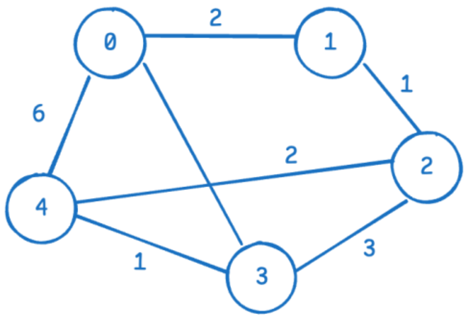


Graphs: a gentle introduction

Classificazione dei grafi

- **Sparso:** pochi archi → $m = O(n)$
- **Denso:** molti archi → $m = O(n^2)$
- **Pesato:** ad ogni arco è associato un valore numerico detto **peso**

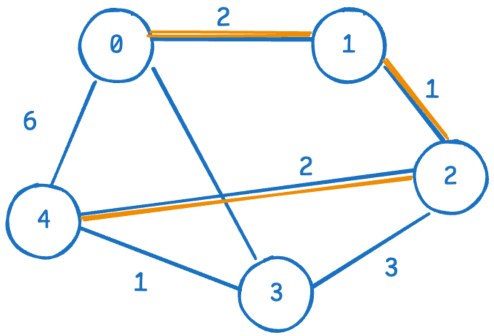




Graphs: Cammini, cicli e grado dei nodi

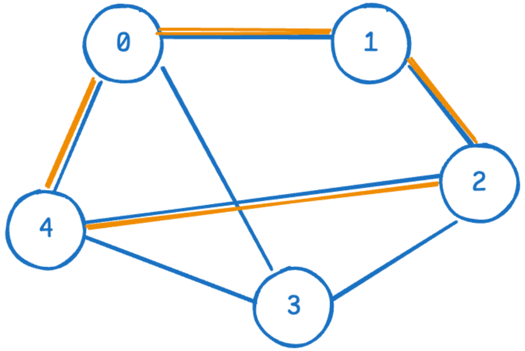
Cammino

- È una sequenza di nodi collegati da archi
- **Lunghezza** è numero di archi (hop) in un cammino
- Se il grafo è pesato, il **peso** del cammino è la somma dei pesi degli archi percorsi



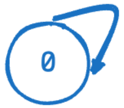
Ciclo

- È un cammino in cui il primo e l'ultimo nodo coincidono
- Può anche consistere in un solo nodo (n)-[e]-(n)
- Un grafo senza cicli è detto **aciclico**



Grado di un nodo

- È il numero di archi che lo coinvolgono
- Nei grafi **diretti** si distinguono:
    - Grado entrante (**in-degree**): archi che arrivano al nodo
    - Grado uscente (**out-degree**): archi che partono dal nodo
- Nei grafi **non diretti**, è il numero totale di archi incidenti

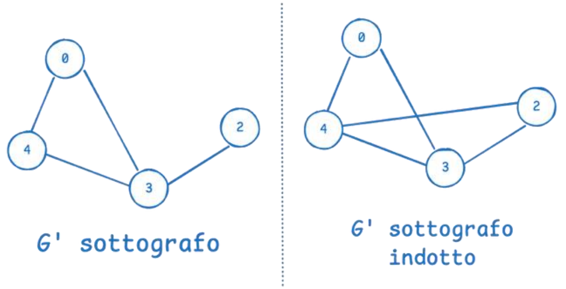


Graphs: Connettività nei grafi

Dati $G = (V, E)$ e $G' = (V', E')$, G' si dice **sottografo** di G se vale

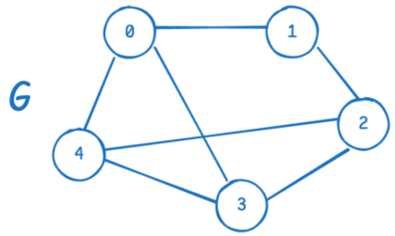
$$V' \subseteq V \text{ e } E' \subseteq E, con \, \{\mathrm{u}, \mathrm{v}\} \in E' \; \Rightarrow u, v \in V'$$

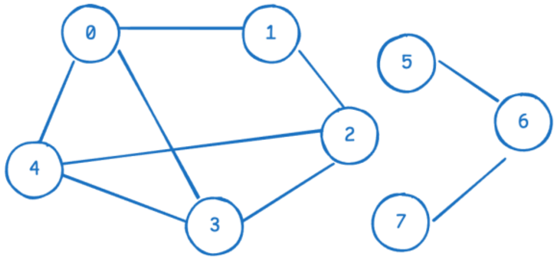
Si dice **sottografo indotto** da V' se $\{\mathrm{u}, \mathrm{v}\} \in E' \; \Leftrightarrow u, v \in V'$



Connettività nei grafi non diretti

- Due nodi u e v sono **connessi** se esiste un cammino tra di loro
- Un grafo è **connesso** se ogni coppia di nodi è connessa
- Una **componente connessa** è un sottografo connesso e massimale. Quindi un grafo è connesso se esiste una sola componente connessa

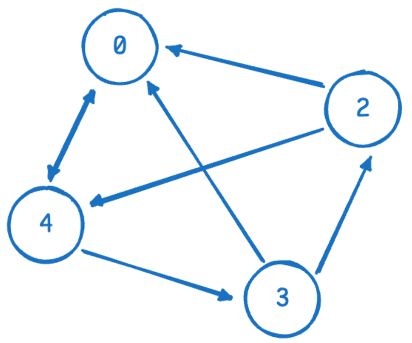




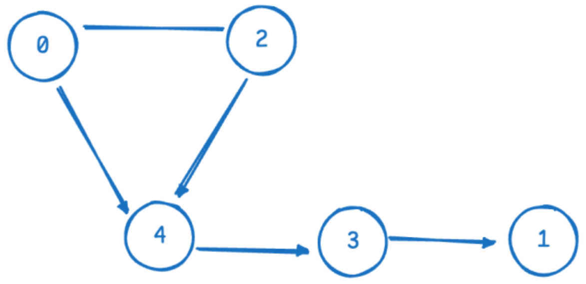
Graphs: Connettività nei grafi

Connettività nei grafi diretti

- **Debolmente connesso**: se sostituendo ogni arco diretto con uno non diretto il grafo è connesso
- **Fortemente connesso**: se esistono cammini da u a v e da v a u. (In un grafo non diretto, due vertici connessi sono anche fortemente connessi)
- Le **componenti fortemente connesse** sono sottografi massimali in cui tutti i nodi sono fortemente connessi tra loro



Strong connected



Weak connected

https://medium.com/@yingpeng0221/unlocking-the-world-of-connections-an-introduction-to-graph-theory-b8cf4da5e090

Graphs: classi particolari

Classi di grafi particolari

- $\mathcal{C}_n$: **cicli semplici** di n nodi
- $\mathcal{K}_n$: **clique** di n nodi, cioè grafo completo di dimensione n (ci sono tutti gli archi possibili)
- $\mathcal{K}_{i,j}$: **grafo bipartito** completo con i nodi i nella partizione di sinistra e quelli j di destra. Cioè:

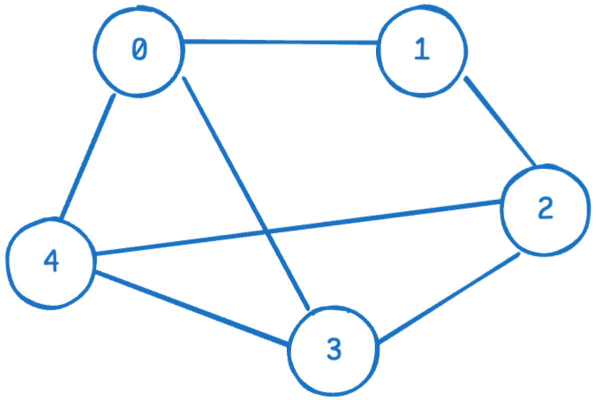
  $G = (Vsx \cup Vd_x, E)$ è un grafo tale che
  - V $= V_{sx} \cup Vdx$, con $V_{sx} \cap Vd_x = \emptyset$ e
  - tale che $\forall \{u, v\} \in E\ u \in V_{sx}$ e v $\in V_{dx}$

Graphs: Come memorizziamo un grafo?

Matrice di adiacenza gli archi vengono memorizzati in una matrice le cui righe e le colonne rappresentano i vertici del grafo.

Per un grafo $G = (V, E)$, con $|V| = n$, si costruisce una matrice quadrata M di dimensione nxn, dove:

$$M[i][j] = \begin{cases} 1 \; se \; (i, j) \; \in \; E \\ 0 \text{ altrimenti} \end{cases}$$



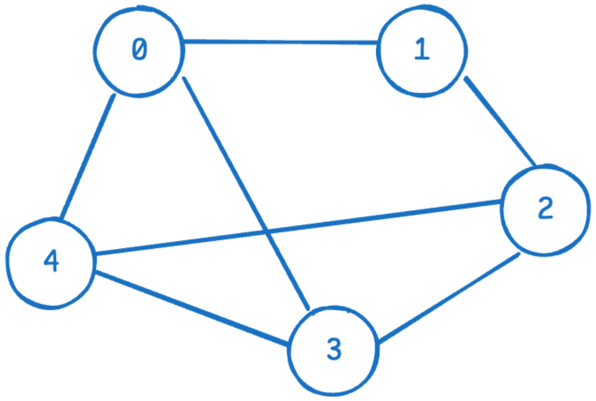
	0	1	2	3	4
0					
1					
2					
3					
4					

Graphs: Come memorizziamo un grafo?

Matrice di adiacenza gli archi vengono memorizzati in una matrice le cui righe e le colonne rappresentano i vertici del grafo.

Per un grafo $G = (V, E)$, con $|V| = n$, si costruisce una matrice quadrata M di dimensione nxn, dove:

$$M[i][j] = \begin{cases} 1 \ se \ (i, j) \ \in \ E \\ \ 0 \text{ altrimenti} \end{cases}$$



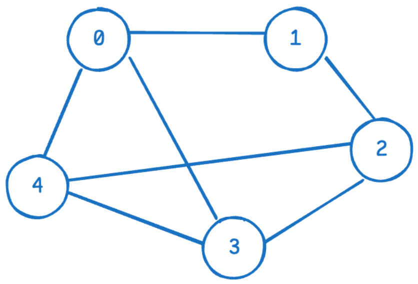
	0	1	2	3	4
0	0	1	0	1	1
1	1	0	1	0	0
2	0	1	0	1	1
3	1	0	1	0	1
4	1	0	1	1	0

Graphs: Come memorizziamo un grafo?

Matrice di adiacenza gli archi vengono memorizzati in una matrice le cui righe e le colonne rappresentano i vertici del grafo.

Per un grafo $G = (V, E)$, con $|V| = n$, si costruisce una matrice quadrata M di dimensione nxn, dove:

$$M[i][j] = \begin{cases} 1 \ se \ (i,j) \ \in \ E \\ 0 \text{ altrimenti} \end{cases}$$



	0	1	2	3	4
0	0	1	0	1	1
1	1	0	1	0	0
2	0	1	0	1	1
3	1	0	1	0	1
4	1	0	1	1	0

Caratteristiche

- Grafi non diretti: la matrice è simmetrica $M[i][j] = M[j][i]$
- Grafi pesati: al posto di 1 si memorizza il peso dell'arco
- Grafi senza loop: la diagonale $M[i][i]$ contiene tutti 0

Pro & Con

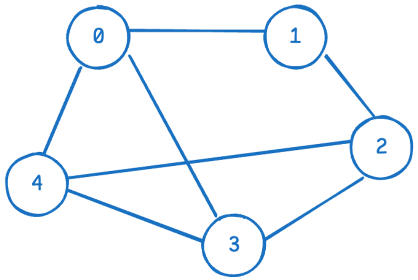
- Accesso rapido per verificare l'esistenza di un arco (i, j): $O(1)$
- Uso inefficiente della memoria per grafi sparsi: $O(n^2)$

Graphs: Come memorizziamo un grafo?

Lista di adiacenza gli archi vengono raggruppati in base al loro vertice di origine.

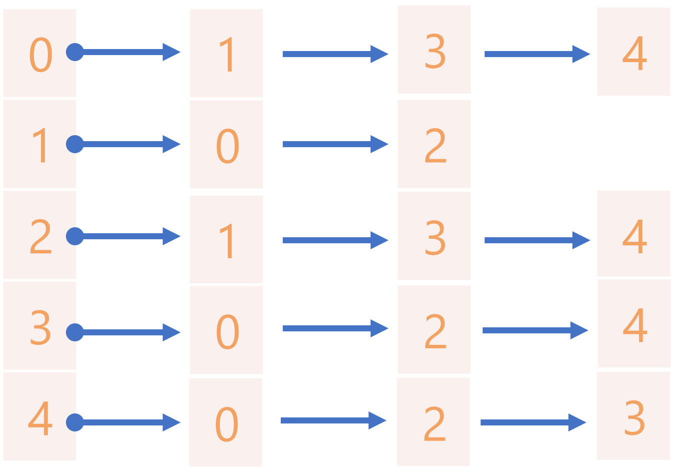
Per un grafo $G = (V, E)$, si associa a ogni vertice v una lista di adiacenza contenente tutti i vertici raggiungibili da v (cioè i vertici per cui esiste un arco in uscita da v):

$$\text{adj}[v] = \{u \in V \mid (v, u) \in E\}$$



Pro & Con

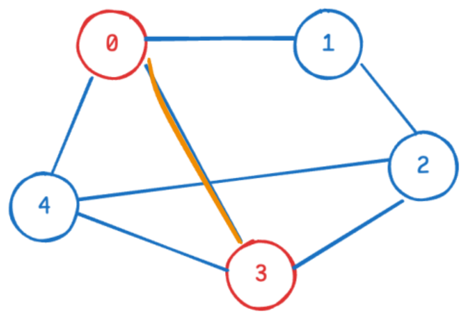
- Efficiente per grafi sparsi: spazio usato proporzionale a $O(m + n)$
- Rappresentazione più compatta della matrice di adiacenza
- No accesso diretto alla presenza di un arco (u,v)! Tempo richiesto $O(\text{degree(u)})$



Graphs: Cammini, cicli e grado dei nodi

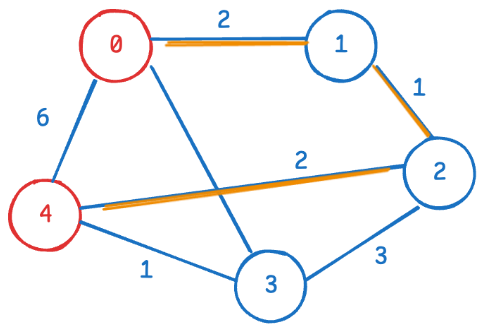
Cammino
- È una sequenza di nodi collegati da archi
- **Lunghezza** è numero di archi (hop) in un cammino
- Se il grafo è pesato, il **peso** del cammino è la somma dei pesi degli archi percorsi

Ciclo
- È un cammino in cui il primo e l'ultimo nodo coincidono
- Può anche consistere in un solo nodo (n)-[e]-(n)
- Un grafo senza cicli è detto **aciclico**



Un cammino è detto **semplice** se nessun nodo è ripetuto
Un ciclo è detto **semplice** se gli unici nodi ripetuti sono il primo e l'ultimo

Il **cammino minimo** tra u e v è il cammino che parte da u, termina in v avente lunghezza minima.



Se G è pesato, il **cammino di peso minimo** è il cammino che parte da u, termina in v avente peso minimo.

Si dice **distanza** tra u e v la lunghezza del cammino minimo da u a v

Page Rank

Cos'è PageRank?

PageRank è un algoritmo sviluppato da **Larry Page e Sergey Brin** (fondatori di Google).

Serve a **misurare l'importanza dei nodi** in un grafo, in particolare per **classificare pagine web**.

L’idea base: **un nodo è importante se è linkato da altri nodi importanti**.

Intuizione di Base

Ogni nodo (pagina) distribuisce il proprio "valore" ai *nodi a cui punta*.

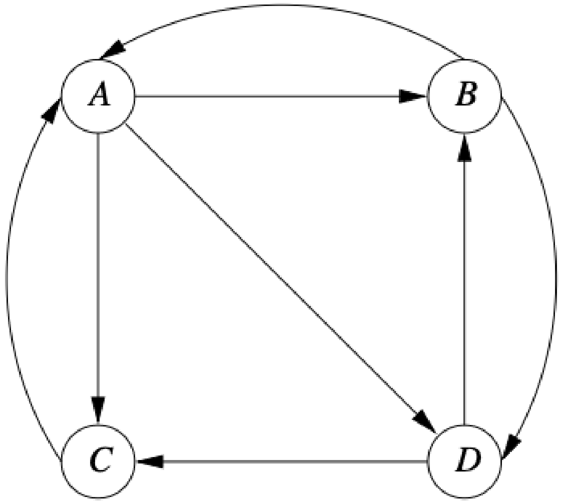
Più link in entrata riceve una pagina, più è considerata importante.

Il valore si stabilizza iterativamente.

Page Rank

Suppose a random surfer starts at node A. There are links to B, C, and D, so this surfer will next be at each of those pages with probability 1/3, and has zero probability of being at A.

A random surfer at B has, at the next step, probability 1/2 of being at A, 1/2 of being at D, and 0 of being at B or C.



$$M = \begin{array}{cc} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \end{array}$$

Page Rank

The **probability distribution** for the **location** of a **random surfer** can be described by a **vector** whose jth component is the probability that the surfer is at page j.

This **probability** is the (idealized) **PageRank** function

How it works:

- A random surfer starts at any of the n pages
- M is the transition matrix
- initial vector v_0 will have 1/n for each component
- after one step, the distribution of the surfer will be Mv_0
- after two steps it will be $M\,(M\,v_0)$

In general, multiplying the initial vector v_0 by M a total of i times will give us the distribution of the surfer after i steps.

Page Rank

Necessary Condition:

The graph is **strongly connected** (it is possible to get from any node to any other node)

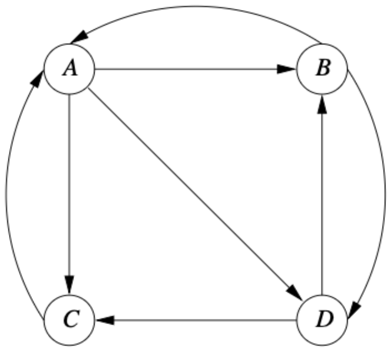
There are **no dead ends** **(**nodes that have no edges out)

The **limit** is reached when multiplying the distribution by M another time does **not change** the **distribution**

The intuition behind PageRank is that the more likely a surfer is to be at a page, the more important the page is.

Page Rank: example

Let’s work on a practical example



$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \end{array}$$

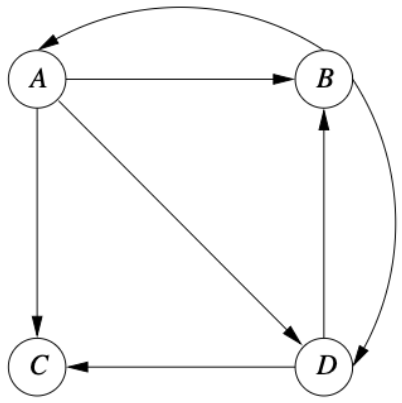
$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix}, \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix}, \ldots, \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

Page Rank: Dead Ends

A Node with no link out is called a **dead end.**

Dead ends make the transition matrix of the Web is no longer stochastic

If we compute $M^i v$ for increasing powers of a **substochastic** matrix M , then some or all of the components of the vector go to 0.

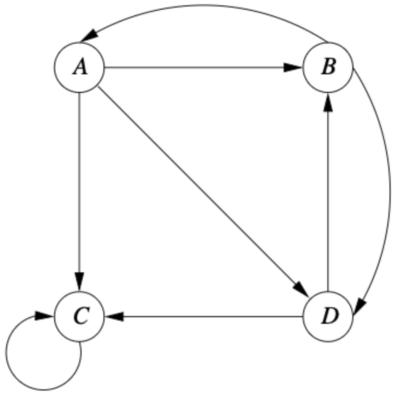


$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 3/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 5/48 \\ 7/48 \\ 7/48 \\ 7/48 \end{bmatrix}, \begin{bmatrix} 21/288 \\ 31/288 \\ 31/288 \\ 31/288 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Page Rank: Spider Traps

A **spider trap** is a set of **nodes** with no dead ends but **no arcs out**.



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

If we perform the usual iteration to compute the PageRank of the nodes, we will get the condition in which the node associated with the spider trap will **get all the page rank**

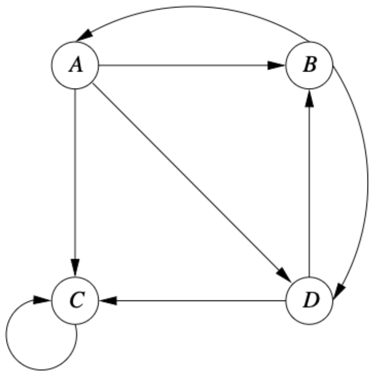
$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 3/24 \\ 5/24 \\ 11/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 5/48 \\ 7/48 \\ 29/48 \\ 7/48 \end{bmatrix}, \begin{bmatrix} 21/288 \\ 31/288 \\ 205/288 \\ 31/288 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Page Rank: Avoiding Dead End and Spider Trap

To avoid the problem, we can give to the random surfer a small probability of **teleporting** to a random node, rather than following an out-link from their current page.

$$\boldsymbol{v}' = \beta M \boldsymbol{v} + (1-\beta)\boldsymbol{e}/n$$

- β is a chosen constant,
- **e** is a vector of all 1's
- The term $\beta M \boldsymbol{v}$ represents the case where, with probability β, the random surfer decides to follow an out-link
- The term $(1-\beta)\boldsymbol{e}/n$ is a vector each of whose components has value $(1-\beta)/n$ and represents the teleporting, with probability 1−β, of the random surfer at a random page.



Ex: With $\beta = 0.8 = 1/5$

$$\mathbf{v}' = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}$$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/60 \\ 13/60 \\ 25/60 \\ 13/60 \end{bmatrix}, \begin{bmatrix} 41/300 \\ 53/300 \\ 153/300 \\ 53/300 \end{bmatrix}, \begin{bmatrix} 543/4500 \\ 707/4500 \\ 2543/4500 \\ 707/4500 \end{bmatrix}, \ldots, \begin{bmatrix} 15/148 \\ 19/148 \\ 95/148 \\ 19/148 \end{bmatrix}$$

Page Rank: Simple code example

NetworkX is a powerful Python library for:

- Creating, manipulating, and analyzing **graphs and networks**
- Supporting different types of graphs: directed, undirected, weighted, multi-graphs
- Performing complex **network algorithms** like shortest paths, clustering, centrality, and **PageRank**

```python
import networkx as nx

G = nx.DiGraph()
G.add_edges_from([("A", "B"), ("A", "C"), ("B", "C"), ("C", "A")])
pagerank = nx.pagerank(G, alpha=0.85)

print(pagerank)
```

Line 1: imports NetworkX with the alias nx, which is standard in the community.

Line 2: creates a **directed graph** (edges have direction: A → B)

Line 3: Adds **edges** between nodes. Nodes "A", "B", and "C" are added automatically

Line4: Computes **PageRank** scores using the NetworkX implementation

- alpha=0.85 is the **damping factor** (standard value, simulating random surfing)
- Returns a dictionary of nodes and their PageRank scores

Hubs and authorities

This **Hubs**-and-**Authorities** algorithm, sometimes called HITS (hyperlink-induced topic search)

HITS views important pages as having two flavors of importance.

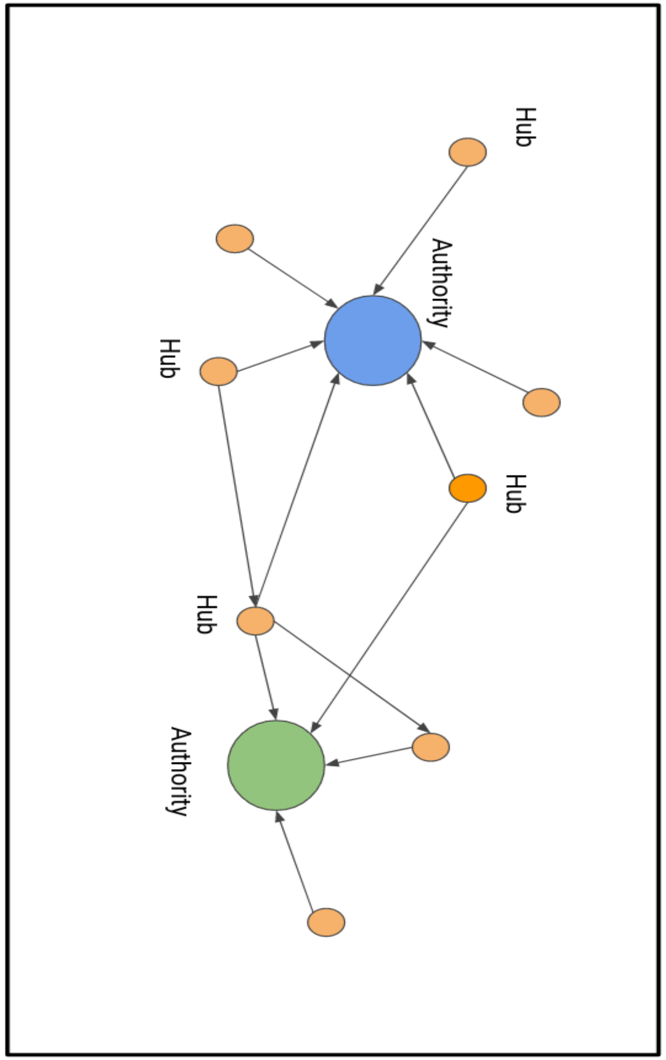
Authorities: pages that are valuable because they provide information about a topic.

Hubs: pages are valuable because they tell us where to go to find out about that topic.

HITS uses a mutually recursive definition of two concepts: "a page is a good hub if it links to good authorities, and a page is a good authority If it is linked to by good hubs."

Assuming that pages are enumerated, we represent these scores by vectors **h** and **a**.

- The ith component of h gives the hubbiness of the ith page
- The ith component of a gives the authority of the



Hubs and authorities

The importance of a node is divided among the successors of that node.

The way to describe the computation of hubbiness and authority is to:

- *add the authority of successors to estimate hubbiness*
- *add the hubbiness of predecessors to estimate authority*

Usually, the values of the vectors **h** and **a** are scaled so that the largest component is 1.

The iterative computation of **h** and **a** formally, we use the link matrix **L**

Link Matrix L is the n×n matrix:

- L_{ij}= 1 if there is a link from page i to page j,
- L_{ij} = 0 if not.

Hubs and authorities

Hubbiness of a page is proportional to the sum of the authority of its successors

expressed by the equation $\boldsymbol{h} = \lambda \boldsymbol{L}\boldsymbol{a}$

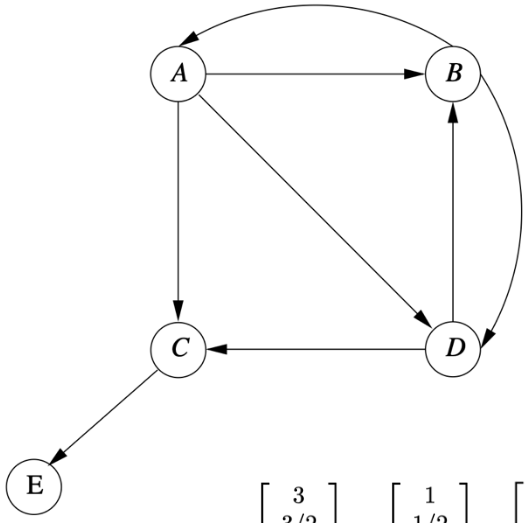
Authority of a page is proportional to the sum of the hubbinesses of its predecessors

expressed by the equation $\boldsymbol{a} = \mu \boldsymbol{L}^T \boldsymbol{h}$

where μ and λ are scale factors

1. Compute $\boldsymbol{a} = \boldsymbol{L}^T \boldsymbol{h}$ and then scale so the largest component is 1.

2. Next, compute $\boldsymbol{h} = \boldsymbol{L}\boldsymbol{a}$ a and scale again.



$$\underset{\mathbf{h}}{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}} \quad \underset{L^{\mathrm{T}}\mathbf{h}}{\begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \end{bmatrix}} \quad \underset{\mathbf{a}}{\begin{bmatrix} 1/2 \\ 1 \\ 1 \\ 1 \\ 1/2 \end{bmatrix}}$$

$$\underset{L\mathbf{a}}{\begin{bmatrix} 3 \\ 3/2 \\ 1/2 \\ 2 \\ 0 \end{bmatrix}} \quad \underset{\mathbf{h}}{\begin{bmatrix} 1 \\ 1/2 \\ 1/6 \\ 2/3 \\ 0 \end{bmatrix}} \quad \underset{L^{\mathrm{T}}\mathbf{h}}{\begin{bmatrix} 1/2 \\ 5/3 \\ 5/3 \\ 3/2 \\ 1/6 \end{bmatrix}} \quad \underset{\mathbf{a}}{\begin{bmatrix} 3/10 \\ 1 \\ 1 \\ 9/10 \\ 1/10 \end{bmatrix}} \quad \underset{L\mathbf{a}}{\begin{bmatrix} 29/10 \\ 6/5 \\ 1/10 \\ 2 \\ 0 \end{bmatrix}} \quad \underset{\mathbf{h}}{\begin{bmatrix} 1 \\ 12/29 \\ 1/29 \\ 20/29 \\ 0 \end{bmatrix}}$$

$$\boxed{\mathbf{h} = \begin{bmatrix} 1 \\ 0.3583 \\ 0 \\ 0.7165 \\ 0 \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} 0.2087 \\ 1 \\ 1 \\ 0.7913 \\ 0 \end{bmatrix}}$$