

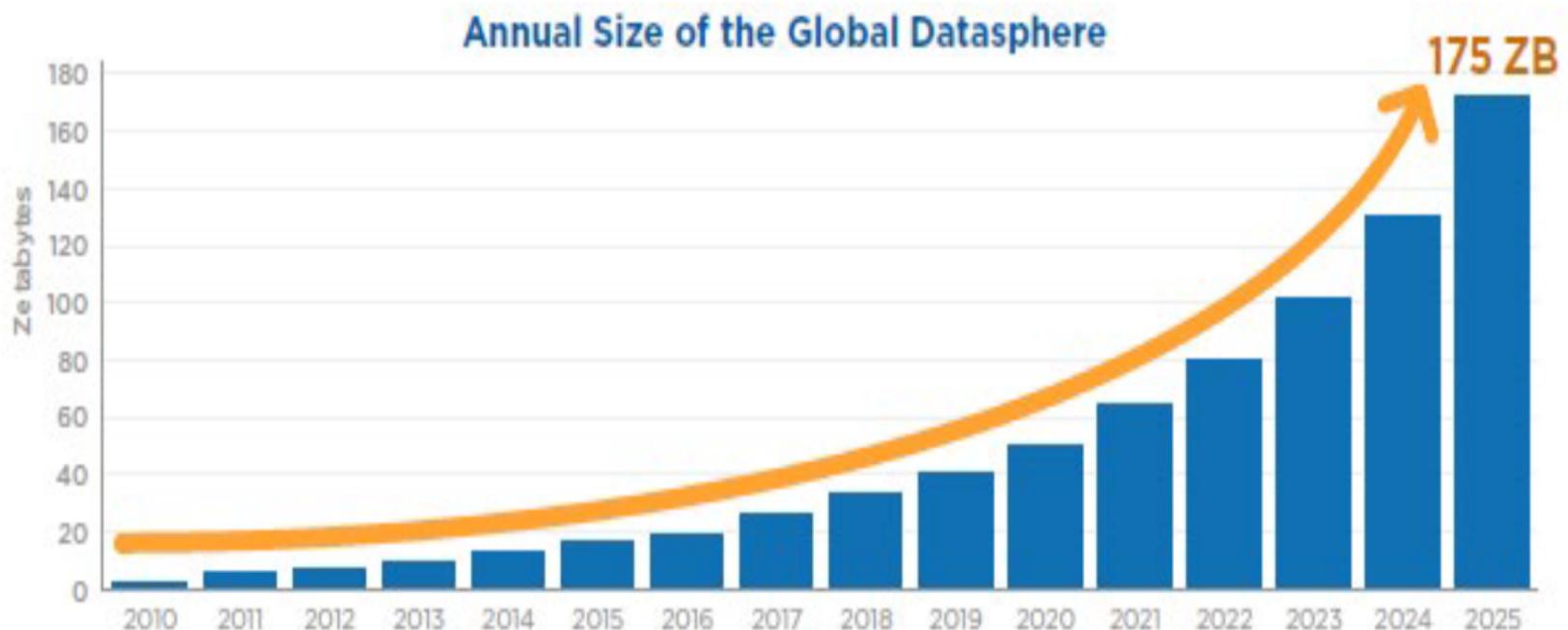


INTERROGATION DE DONNÉES DE TYPE GRAPHE

BDLE (BASES DE DONNÉES LARGE ECHELLE)

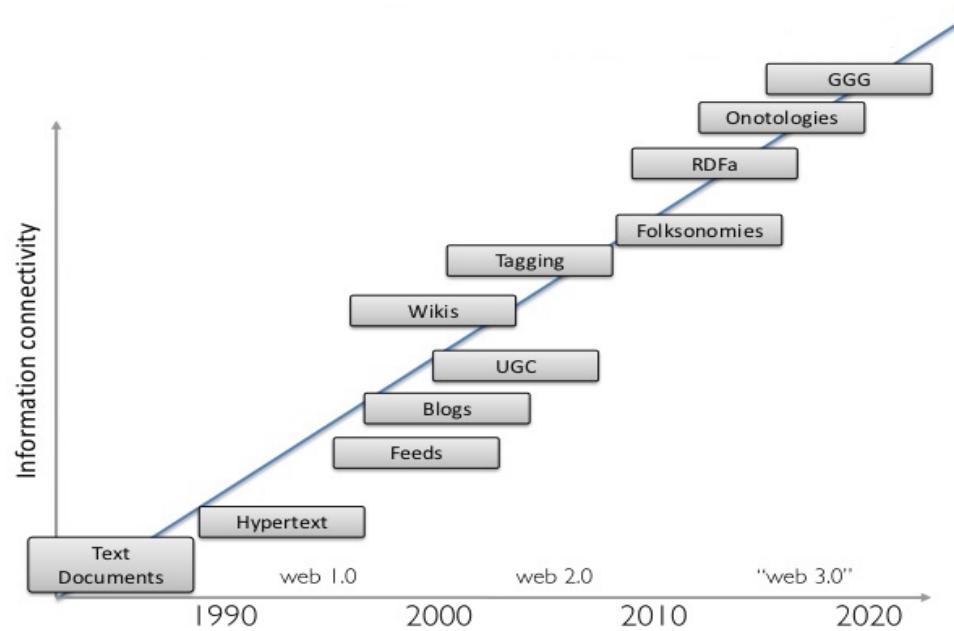
CAMELIA CONSTANTIN -- PRÉNOM.NOM@LIP6.FR

Croissance exponentielle du volume des données



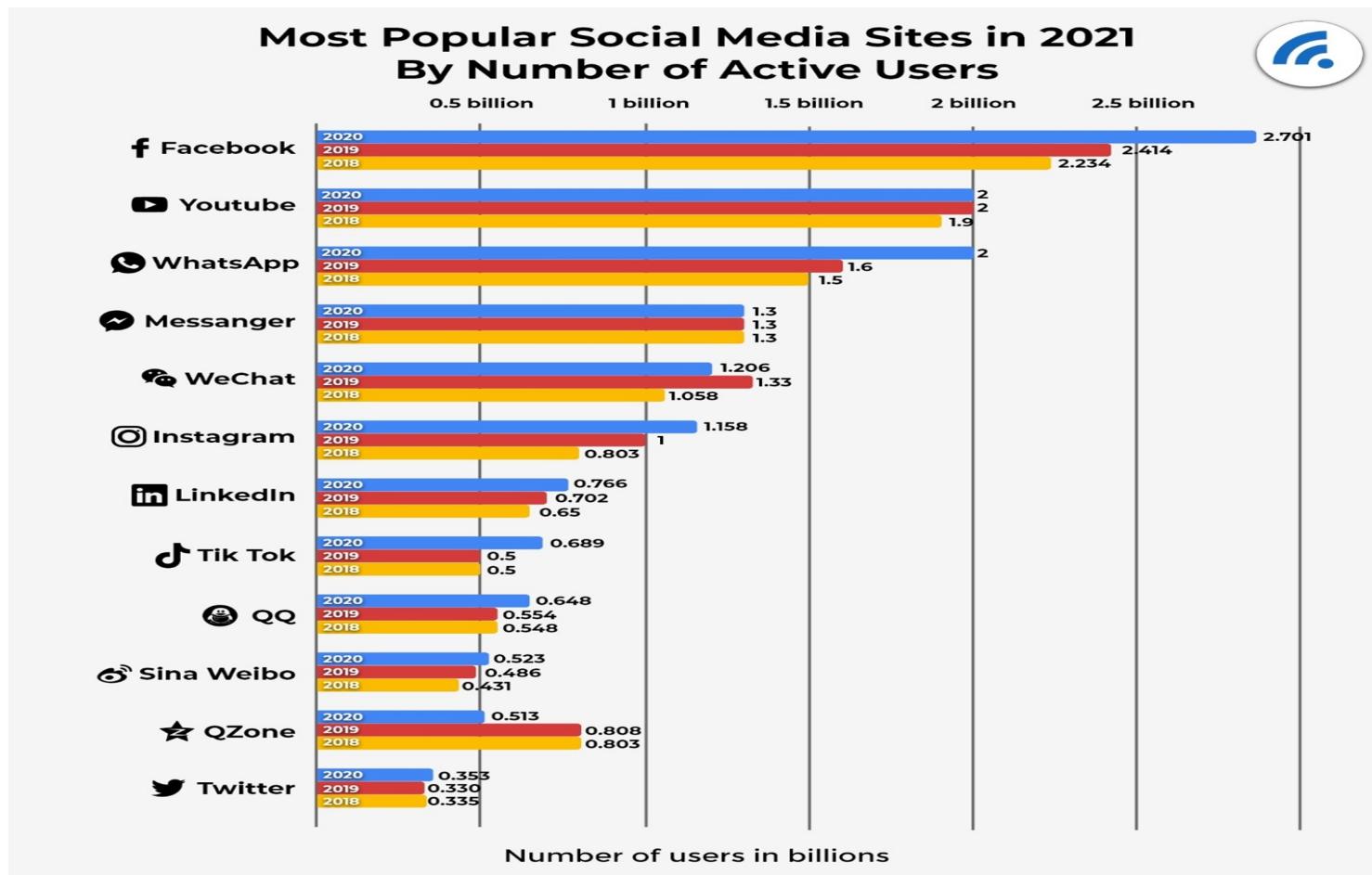
Source: Data Age 2025 report, sponsored by Seagate with data from IDC Global Datasphere, November 2018

CONNECTIVITÉ DES DONNÉES



Volume x Connectivité = **Complexité**

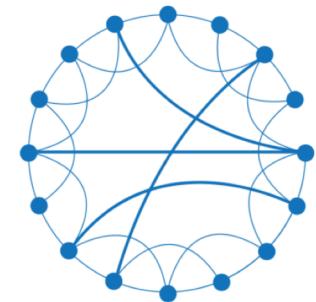
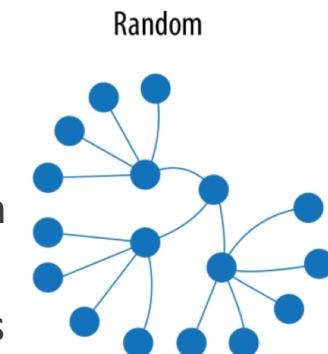
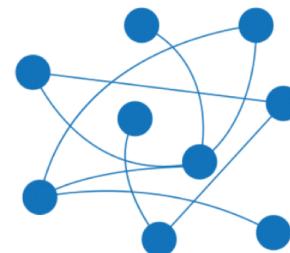
RÉSEAUX SOCIAUX: des graphes gigantesques



Types de réseaux

Trois types représentatifs:

- Aléatoires : tous les nœuds ont la même probabilité d'être connectés à un autre nœud
- Petit monde: chemins très courts entre tous les noeuds (ex: réseaux sociaux)
- Sans échelle: loi de puissance pour la distribution des degrés des nœuds, quelques nœuds très connectés et beaucoup de nœuds mal connectés (ex: WWW)



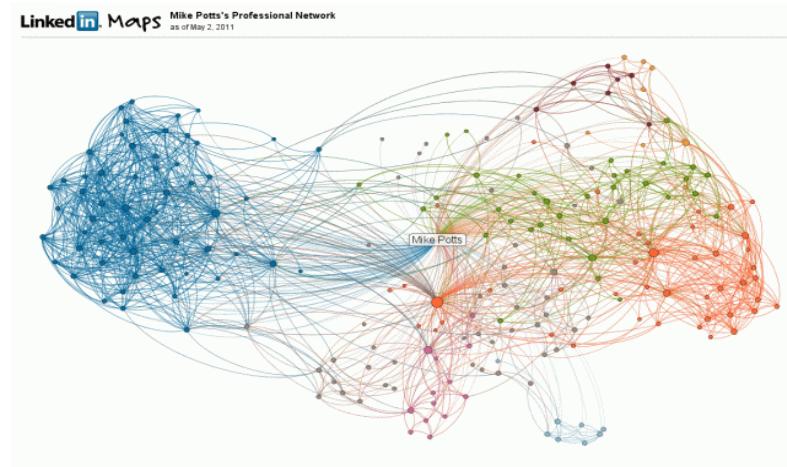
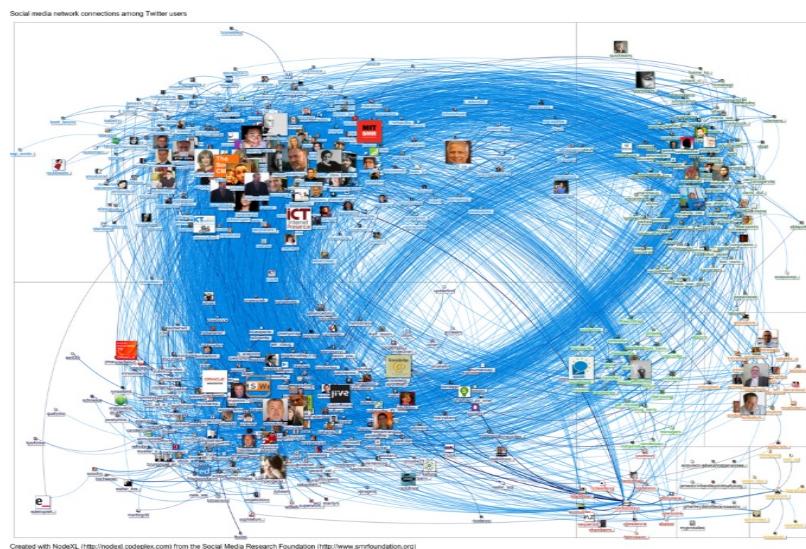
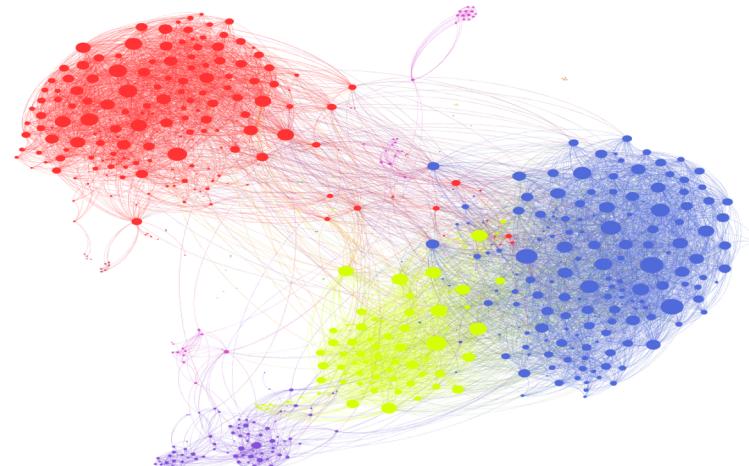
Scale-Free

Random

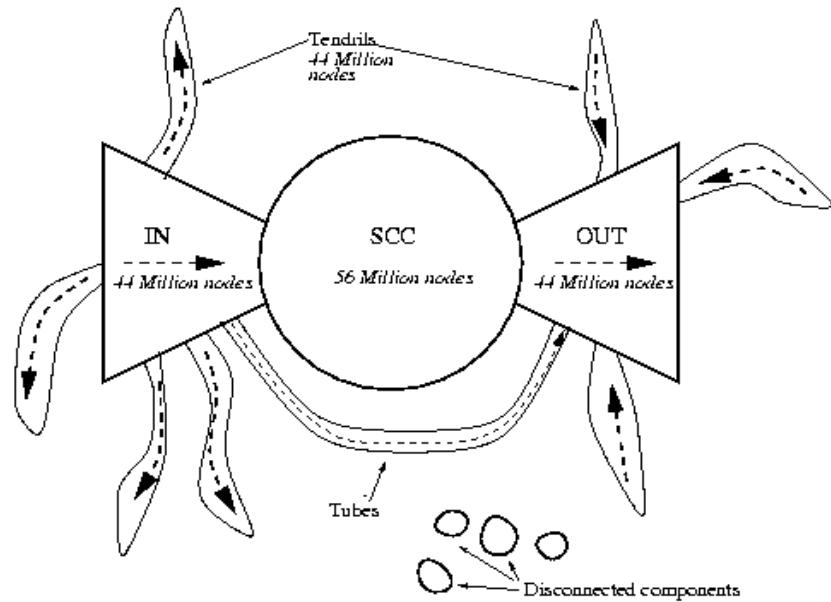
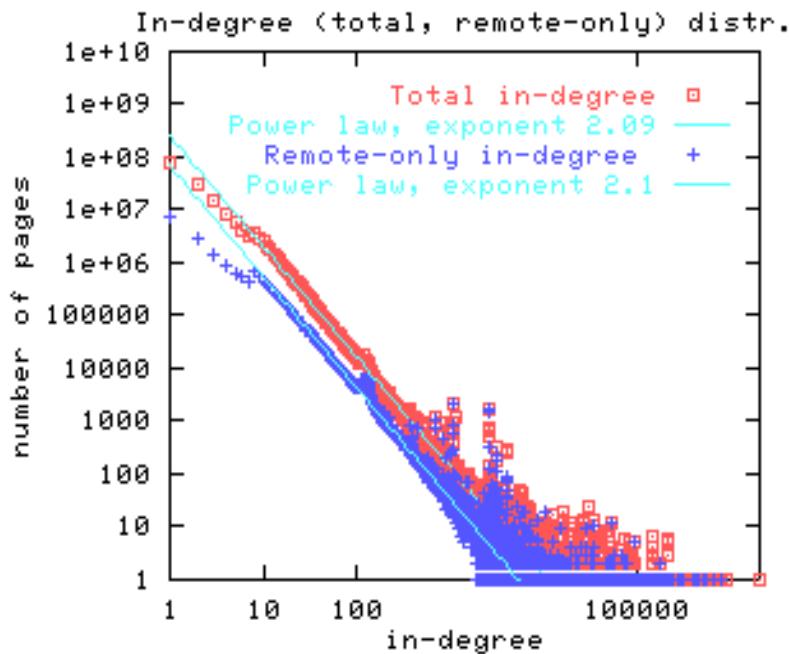
Small-World

Exemple: RÉSEAUX SOCIAUX (2014)

	Utilisateurs (10⁶)	Arcs (10⁹)
Facebook	1 300	400
LinkedIn	330	63
Twitter	650	130



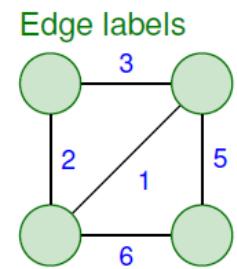
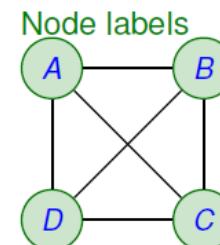
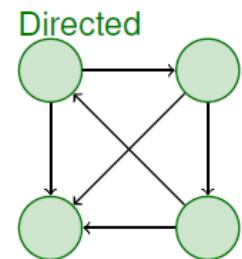
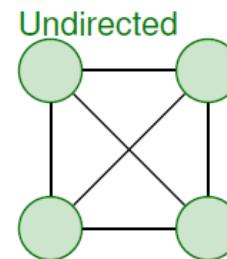
Exemple: Graphe du Web



Pages Web (Broder et al.* , 2000) : A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, J. Wiener. Graph structure in the web. In WWW'00, pages 309-320. Crawl pages en 1999

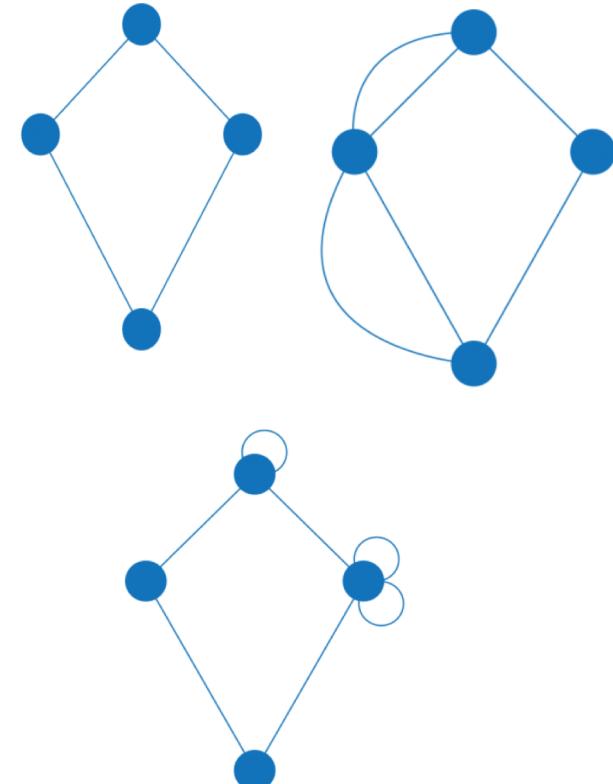
TYPES DE GRAPHE

- *dirigés* : réseau social, citations bibliographiques, web hypertexte, web sémantique, graphe de recommandation, graphe d'évolution...
- *non-dirigés* : réseau routier, réseau des collaboration, graphes de cooccurrences,
- étiquetés:
 - nœuds : nom, âge, contenu
 - arcs : amitiés, coût, durée, ..



TYPES DE GRAPHE

- *simple* : une seule arête entre chaque paire de noeuds
- *multigraphe* : plusieurs arêtes entre chaque paire de noeuds
- *pseudographe*: plusieurs arêtes et des boucles





REPRÉSENTATION DES GRAPHS DE DONNÉES

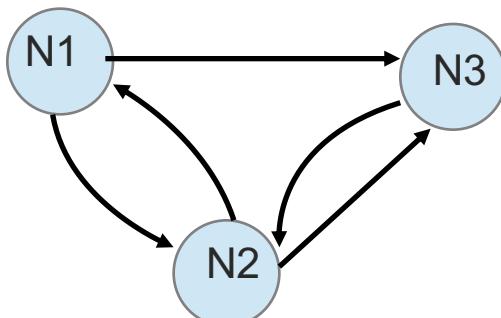


MATRICE D'ADJACENCE

compte	nom	email	...
N1	Jean	...	
N2	Lucie	...	
N3	Marc	...	

noeud	N1	N2	N3
N1	0	1	1
N2	1	0	1
N3	0	1	0

Données



Matrice d'adjacence :

- Carrée (dimension n^2)
- Symétrique pour un graphe non-orienté

MATRICE D'ADJACENCE

- **Avantages :**

- Utilisation de librairies d'algèbre linéaire si l'on dispose d'un stockage et d'un traitement efficaces de telles matrices (frameworks récents)
- Temps constant pour trouver si un lien existe entre deux nœuds (en regardant l'entrée correspondante dans la matrice)

- **Inconvénients :**

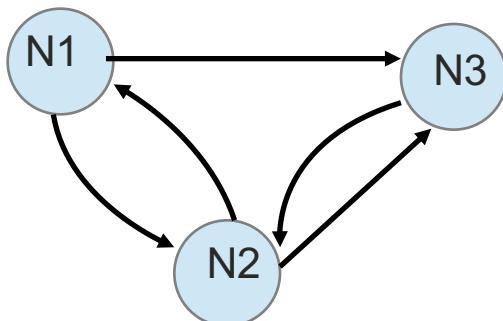
- Matrice de grande taille, creuse (beaucoup de valeurs zéro)
- Recherche linéaire (inefficace) pour trouver les nœuds adjacents à un nœud donné (recherche linéaire sur toute la ligne correspondante au nœud i)

Liste d'arcs

compte	nom	email	...
N1	Jean	...	
N2	Lucie	...	
N3	Marc	...	

Données

follower	followee
N1	N2
N1	N3
N2	N1
N2	N3
...	...



Liste d'arcs

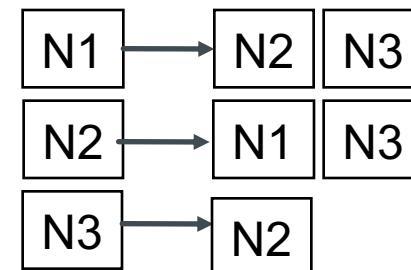
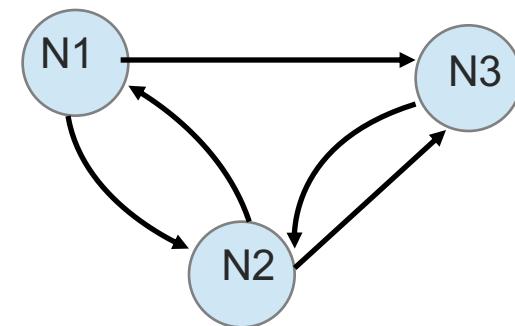
LISTES D'ADJACENCE

compte	nom	email	...
N1	Jean	...	
N2	Lucie	...	
N3	Marc	...	

Données

follower	followee
N1	N2
N1	N3
N2	N1
N2	N3
N3	N2

Liste d'arcs



Liste d'adjacence

COMPARAISON TAILLE

Exemple :LinkedIn (graphe non-dirigé, chaque entrée sur un bit)

- 330 millions de nœuds

- 63 milliards d'arcs

- Matrice d'adjacence

$$330 * 10^6 * 330 * 10^6 \text{ bits} = 108,9 * 10^{15} \text{ bits} \approx$$

$$14 \cdot 10^{15} \text{ octets} = 14 \text{ pétaoctets}$$

- Table relationnelle / liste d'adjacence

- 1 id de nœud = double = 4o

- $63 * 10^9 * 2 * 4 \text{ octets} = 5 * 10^{11} \text{ octets} = 0,5 \text{ téraoctets}$

COMPARAISON OPÉRATIONS

Mises-à-jour

■ Matrice d'adjacence :

- noeuds: insertion / suppression de lignes et colonnes ($\sim |N|$)
- arcs : maj cellule (constant)

■ Liste d'adjacence

- noeuds : creation de liste ($\sim |N|$)
- arcs: maj liste (\sim out-degree)

■ Liste d'arcs

- arcs : insertion et suppression de nuplets (depend des index)

Accès

■ Matrice d'adjacence

- operations logiques (AND, OR)
- operations matricielle : +, -, *, T

■ Liste d'adjacence

- parcours de listes

■ Liste d'arcs (table relationnelle)

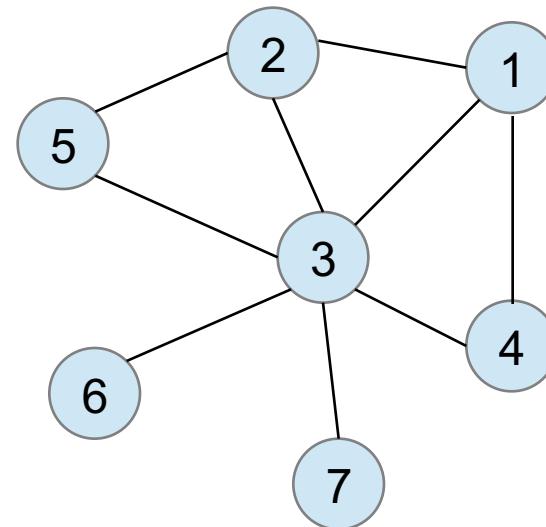
- langages de requêtes
- stockage et indexation

Comparaison liste d'arcs et liste d'adjacence

- Choix de la représentation en fonction de l'application, ex: **Calcul des voisins communs dans Spark**

Liste d'arcs (DataFrame): $N1 < N2$

N1	N2
1	2
1	3
1	4
2	3
2	5
3	4
3	5
3	6
3	7

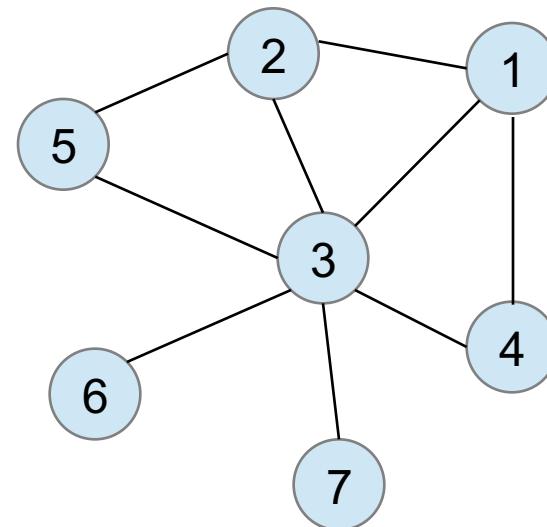


Comparaison des représentations

■ Application: Calcul des voisins communs dans Spark

- Calculer le nombre de voisins que l'utilisateur I a en commun avec tous les autres utilisateurs

Couple d'utilisateurs	Voisins communs
(1,2)	1
(1,3)	2
(1,4)	1
(1,5)	2
(1,6)	1
(1,7)	1



- Nombre de couples d'utilisateurs >> nombre de voisins directs de l'utilisateur -> **Calcul efficace?**

Comparaison des représentations

- Calcul des voisins communs dans Spark

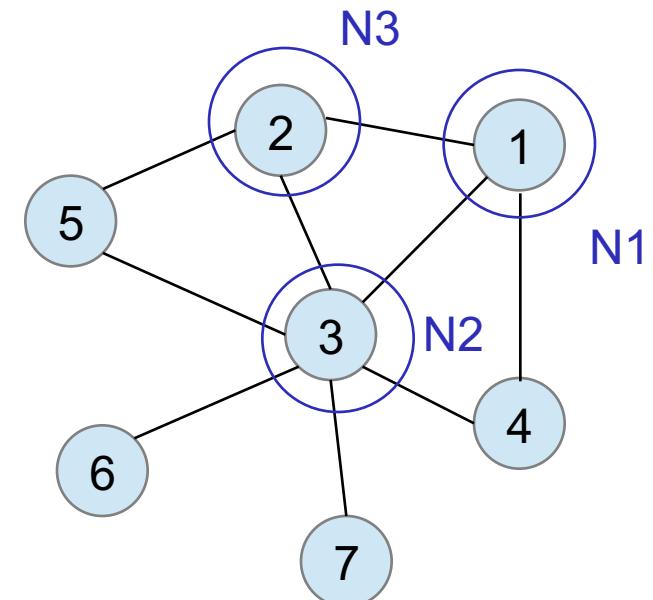
DataFrame (df1)

N1	N2
1	2
1	3
1	4
2	3
2	5
3	4
3	5
3	6
3	7

DataFrame (df2)

N2	N3
1	2
1	3
1	4
2	3
2	5
3	4
3	5
3	6
3	7

jointure



Calcul des voisins communs

```
joinDF = df1.join(df2, df1.N2 == df2.N2)
```

N1	N2
1	2
1	3
1	4
2	3
2	5
3	4
3	5
3	6
3	7

jointure

N2	N3
1	2
1	3
1	4
2	3
2	5
3	4
3	5
3	6
3	7

=

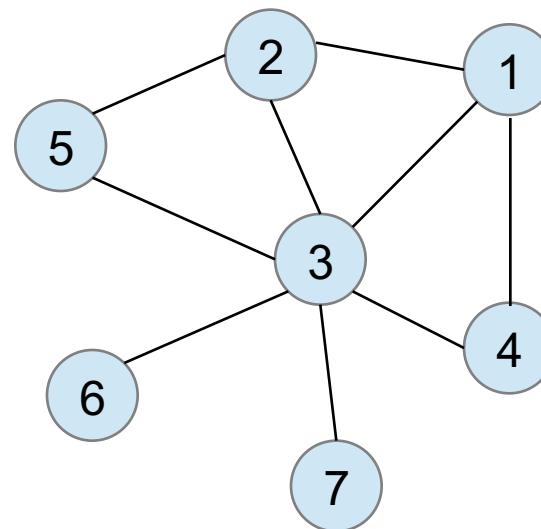
N1	N2	N2	N3
1	2	2	3
1	2	2	5
1	3	3	4
1	3	3	5
1	3	3	6
1	3	3	7
2	3	3	4
2	3	3	5
2	3	3	7

Calcul des voisins communs

joinDF

N1	N2	N2	N3
1	2	2	3
1	2	2	5
1	3	3	4
1	3	3	5
1	3	3	6
1	3	3	7
2	3	3	4
2	3	3	5
2	3	3	7

```
joinDF.drop(col('N2'))  
    .groupBy('N1', 'N3')  
    .count()  
    .filter(col('N1') == 1)
```



Résultat

N1	N3	count
1	3	1 ?
1	4	1
1	5	2
1	6	1
1	7	1

Manquent les voisins
Communs de 1 et de 2!

Calcul des voisins communs

```
joinDF = df1.join(df2, df1.N2 == df2.N2)
```

N1	N2
1	2
1	3
1	4
2	3
2	5
3	4
3	5
3	6
3	7

3 2

jointure

N2	N3
1	2
1	3
1	4
2	3
2	5
3	4
3	5
3	6
3	7

3 2

N1	N2	N2	N3
1	2	2	3
1	2	2	5
1	3	3	4
1	3	3	5
1	3	3	6
1	3	3	7
2	3	3	4
2	3	3	5
2	3	3	7

1 3 3 2

Calcul des voisins communs

N1	N2
1	2
2	1
1	3
3	1
1	4
4	1
2	3
3	2
2	5
5	2
3	4
4	3
3	5
5	3
3	6
6	3
3	7
7	3

jointure

N1	N2
1	2
2	1
1	3
3	1
1	4
4	1
2	3
3	2
2	5
5	2
3	4
4	3
3	5
5	3
3	6
6	3
3	7
7	3

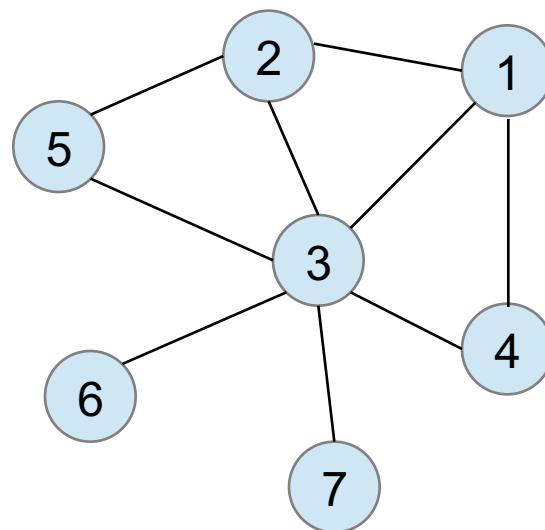
N1	N2	N2	N3
1	2	2	1
1	2	2	3
1	2	2	5
1	3	3	1
1	3	3	2
1	3	3	4
1	3	3	5
1	3	3	6
1	3	3	7
1	4	4	1
1	4	4	3

joinDF .filter(col('N1')==1)

Calcul des voisins communs

N1	N2	N2	N3
1	2	2	1
1	2	2	3
1	2	2	5
1	3	3	1
1	3	3	2
1	3	3	4
1	3	3	5
1	3	3	6
1	3	3	7
1	4	4	1
1	4	4	3

```
joinDF.drop(col('N2'))  
    .groupBy('N1', 'N3')  
    .count()
```



N1	N3	count
1	3	2
1	4	1
1	5	2
1	6	1
1	7	1
1	1	3
1	2	1

Résumé du calcul basé sur les tables d'adjacence

Algorithme de calcul et problèmes:

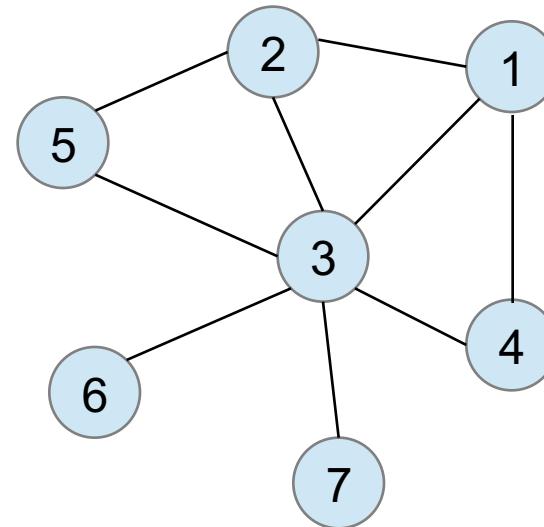
- Modifier le DF: pour chaque arc $x \rightarrow y$ ajouter l'arc $y \rightarrow x$ ← **Problème**: duplication des données
- Auto-jointure du DF construite à l'étape précédente ← **Problème**: shuffle des données
- Compter le nombre d'occurrences de chaque paire d'utilisateurs ← **Problème**: shuffle des données

Comparaison des représentations

- Calcul des voisins communs dans Spark
Comparaison liste d'arcs et liste d'adjacence

Liste d'adjacence (DataFrame): listes triées

N	Voisins
1	[2, 3, 4]
2	[1, 3, 5]
3	[1, 2, 4, 5, 6, 7]
4	[1, 3]
5	[2,3]
6	[3]
7	[3]



Principe calcul: Les utilisateurs appartenant à la même liste d'adjacence ont un utilisateur en commun

Calcul initial basé sur les listes d'adjacence

Algorithme de calcul et problèmes:

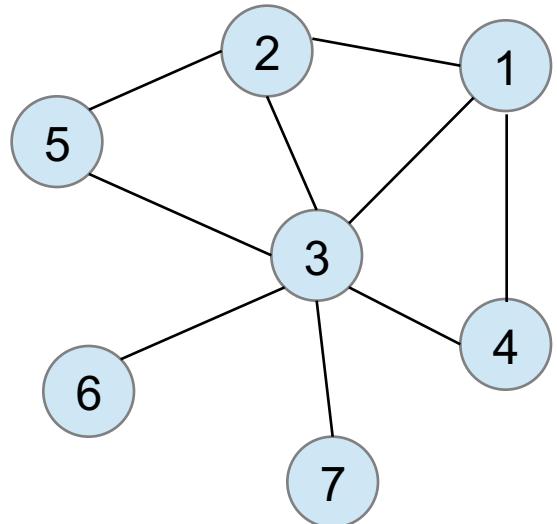
- I. Pour chaque liste d'adjacence construire **tous les couples d'utilisateurs**:

Exemple pour $l: [2,3], [3,2], [2,4], [4,2], [3,4], [4,3] \leftarrow$ **Problème**: duplication des données

2. Pour chaque couple compter le nombre d'occurrences (= nombre de voisins que les nœuds du couple ont en commun) \leftarrow **Problème**: shuffle des données pendant le groupBy

Réduction du volume de données générées à la première étape:

Constat : les couples $[n1, n2]$ et $[n2, n1]$ ont le même nombre de voisins \rightarrow construire uniquement des couples ordonnés par ordre croissant des identifiants



Comparaison des représentations

Liste d'adjacence: listes triées

N	Voisins
1	[2, 3, 4]
2	[1, 3, 5]
3	[1, 2, 4, 5, 6, 7]
4	[1, 3]
5	[2, 3]
6	[3]
7	[3]

Couples contenant
l'utilisateur 1

[N1, N2]
[1, 3]
[1, 5]
[1, 2]
[1, 4]
[1, 5]
[1, 6]
[1, 7]
[1, 3]



Nombre d'utilisateurs en
commun

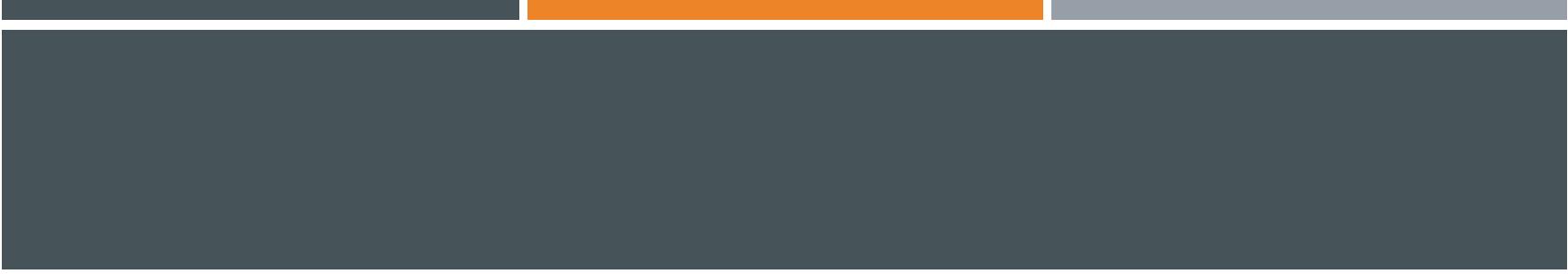
[N1, N2]	count
[1, 3]	2
[1, 5]	2
[1, 2]	1
[1, 4]	1
[1, 6]	1
[1, 7]	1



Nombre d'utilisateurs en commun correct

Pour [1,3] et [1,2] sans dupliquer les
données!

Exercice: quels sont les opérations nécessaires pour obtenir
le résultat final?



ALGORITHMES DE CALCUL SUR LES GRAPHES

EXEMPLES DE REQUÊTES GRAPHE

- Dans un réseau de type transport, alimentation, communication
 - Comment aller de l'adresse a à l'adresse b ?
 - Combien de chemins entre le nœud réseau a et le nœud réseau b ?
 - Le chemin le plus rapide entre l'usine a et le magasin b ?
- Dans un réseau de représentation de connaissance:
 - Une classe (élément) A est elle un sous-classe d'une classe B?
 - Existe-t'il un lien entre l'entité A et l'entité B?
 - Similarité entre l'entité A et l'entité B basé sur le graphe sémantique
- Réseaux de citations
 - Quels auteurs sont le plus cités directement et indirectement ?
 - Quels chercheurs / articles sont important dans un domaine (centralité) ?
- Réseaux sociaux
 - You Might Also Know de Facebook: si on partage beaucoup d'amis, on doit sans doute se connaître
 - Recommandation de lieux dans FourSquare d'après avis des amis: si des amis recommandent un lieu, alors de bonne chance que j'aime aussi
 - Degré de séparation (ex : nombre d'utilisateurs entre deux utilisateurs sur Facebook)

ANALYSE DES GRAPHS

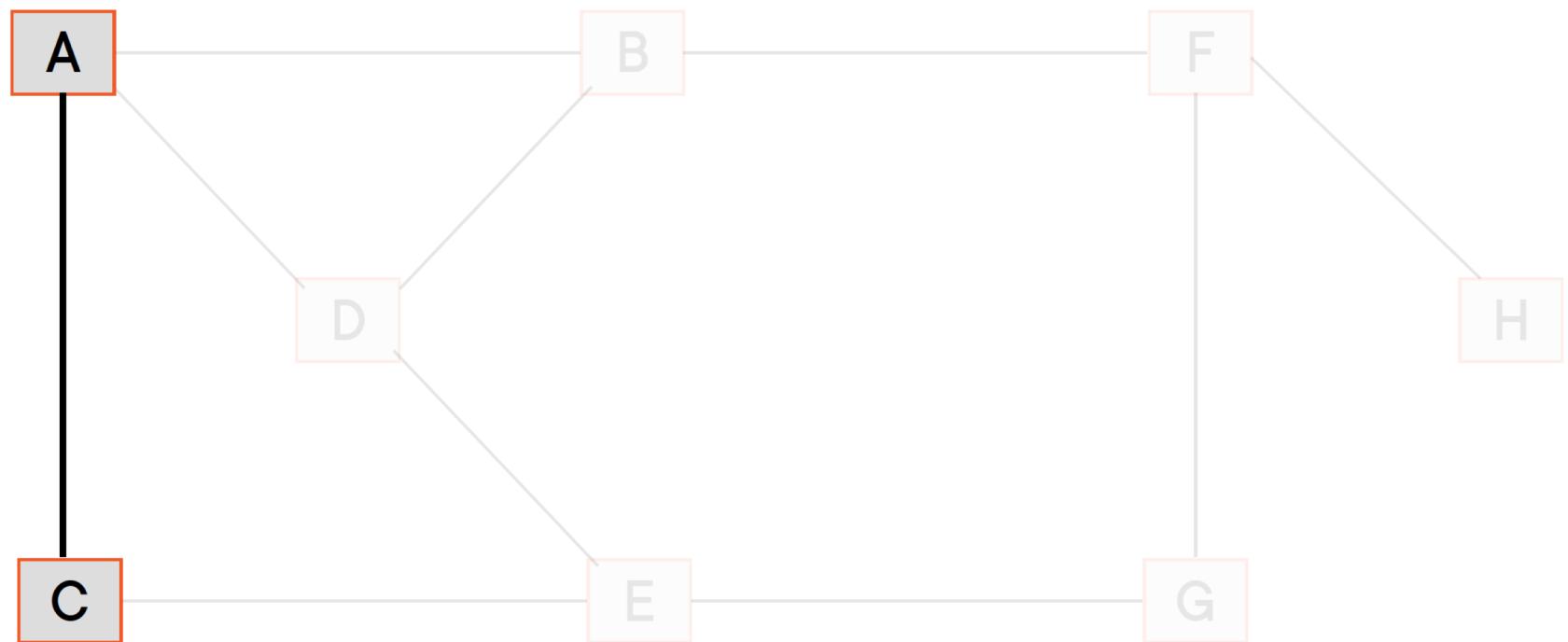
■ Mesures de base :

- Nombre de nœuds : $|V|$
- Nombre d'arcs : $|E|$ (plus important que $|V|$)
- in/out-degree(n) : nombre d'arcs entrants/sortants

■ Types d'algorithmes:

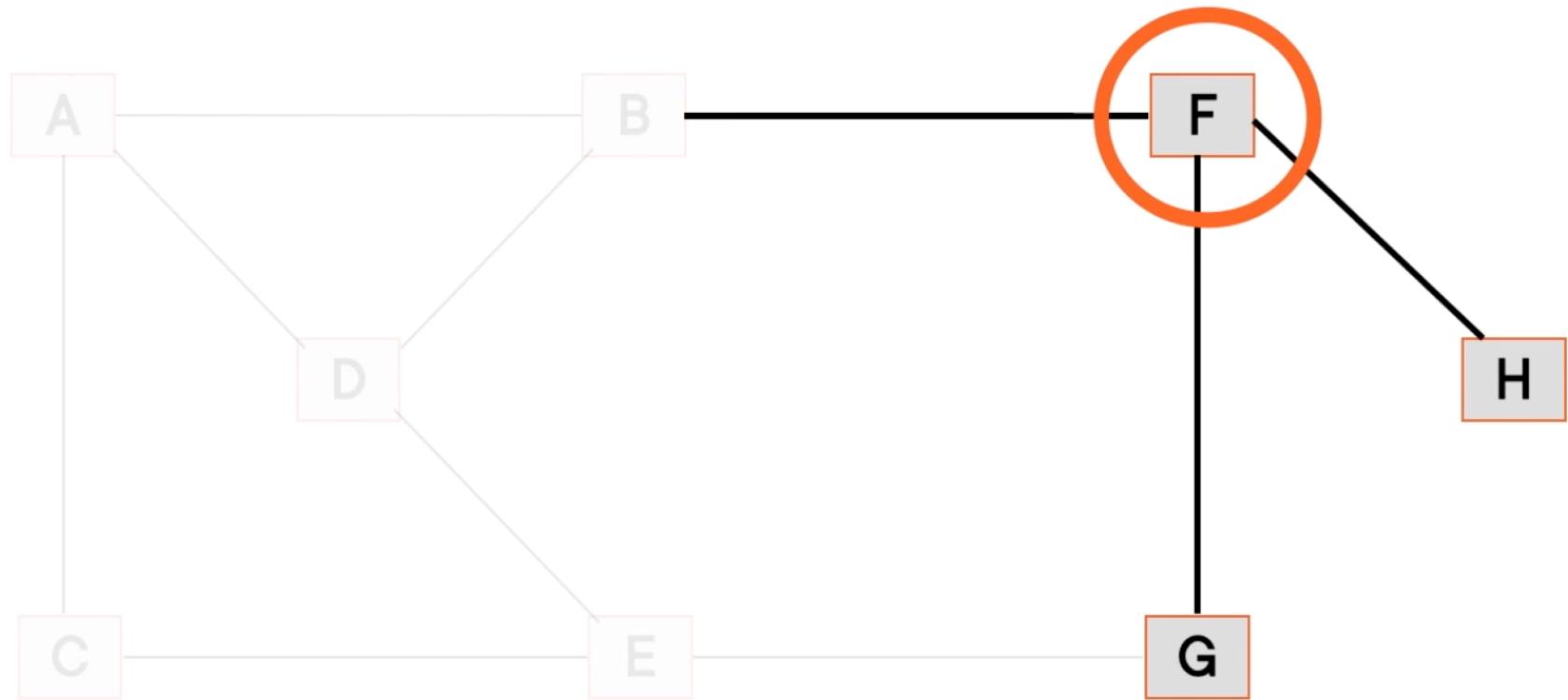
- Recherche de chemins: existence/nombre de chemins, plus court/long chemin, diamètre, chemins correspondant à une expression régulière $(A \ B)^*(C|D)^+$
- Centralité: l'importance des noeuds dans le réseau
- Connexité: découverte de structures, recherche de communautés
- Recherche de motifs de graphes : cycles, "motifs de graphes", arbre couvrant, circuit hamiltonien

Graphe non dirigé: Nœuds adjacents



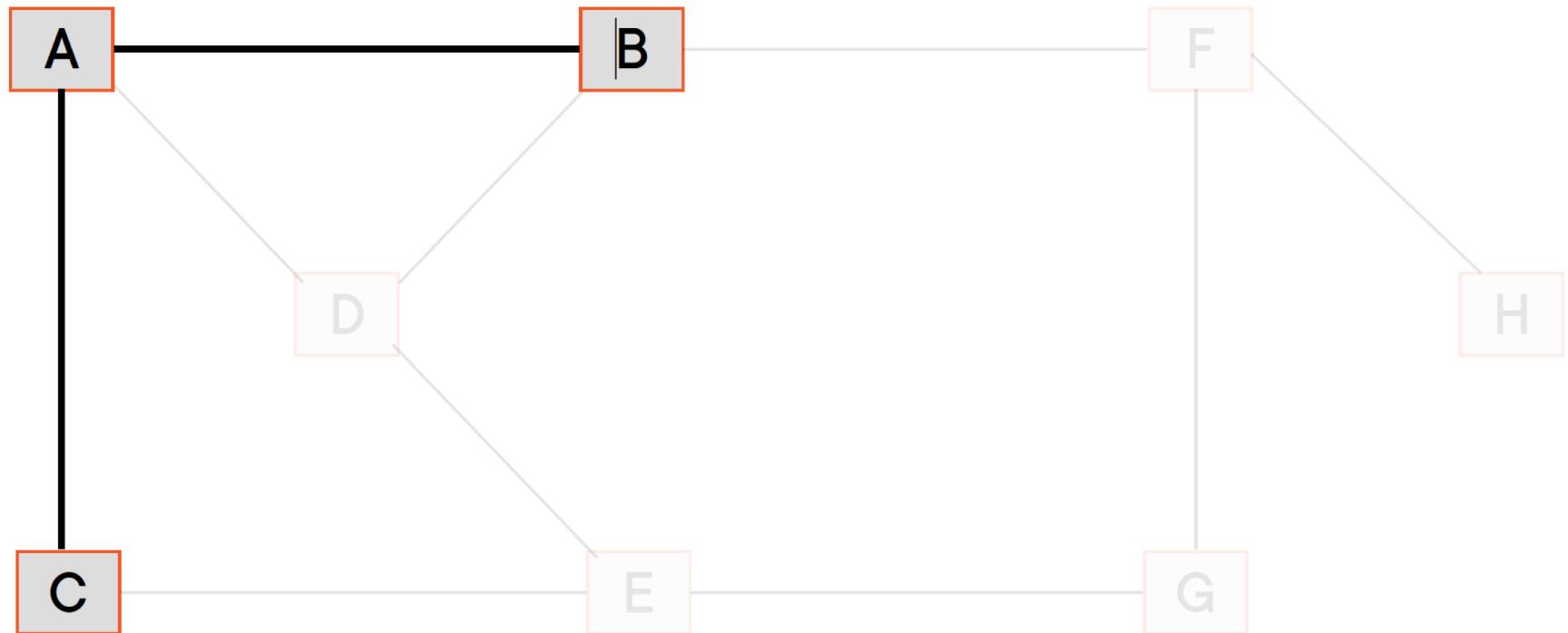
- Il existe une arête qui les relie

Graphe non dirigé: Degré d'un noeud



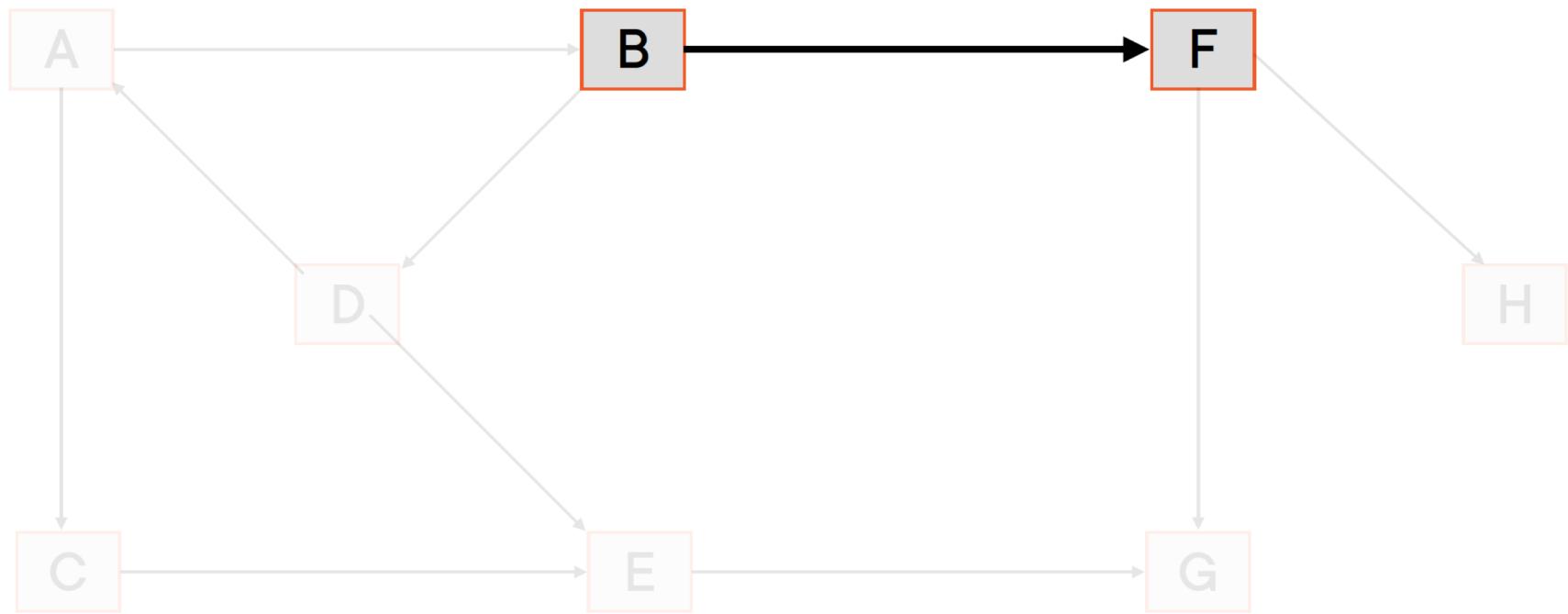
Le sommet F a le degré 3 (nombre d'arêtes incidentes)

Chemin dans un graphe non dirigé



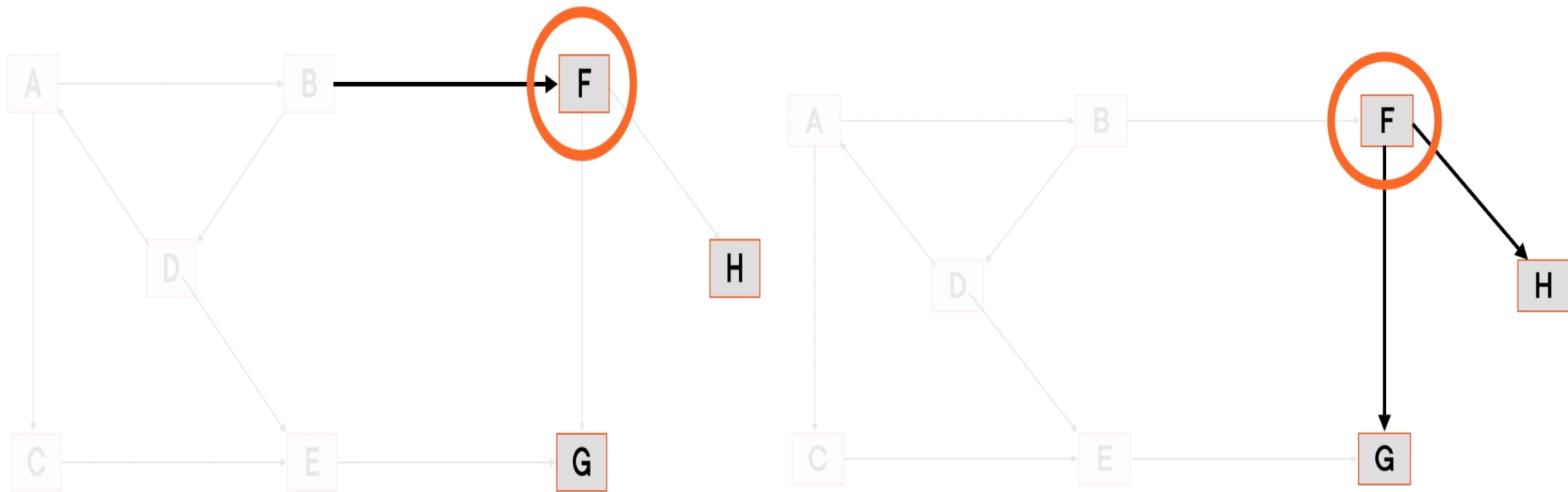
Suite finie d'arêtes consécutives, reliant B à C (dans les deux sens)

Nœuds adjacents dans un graphe dirigé



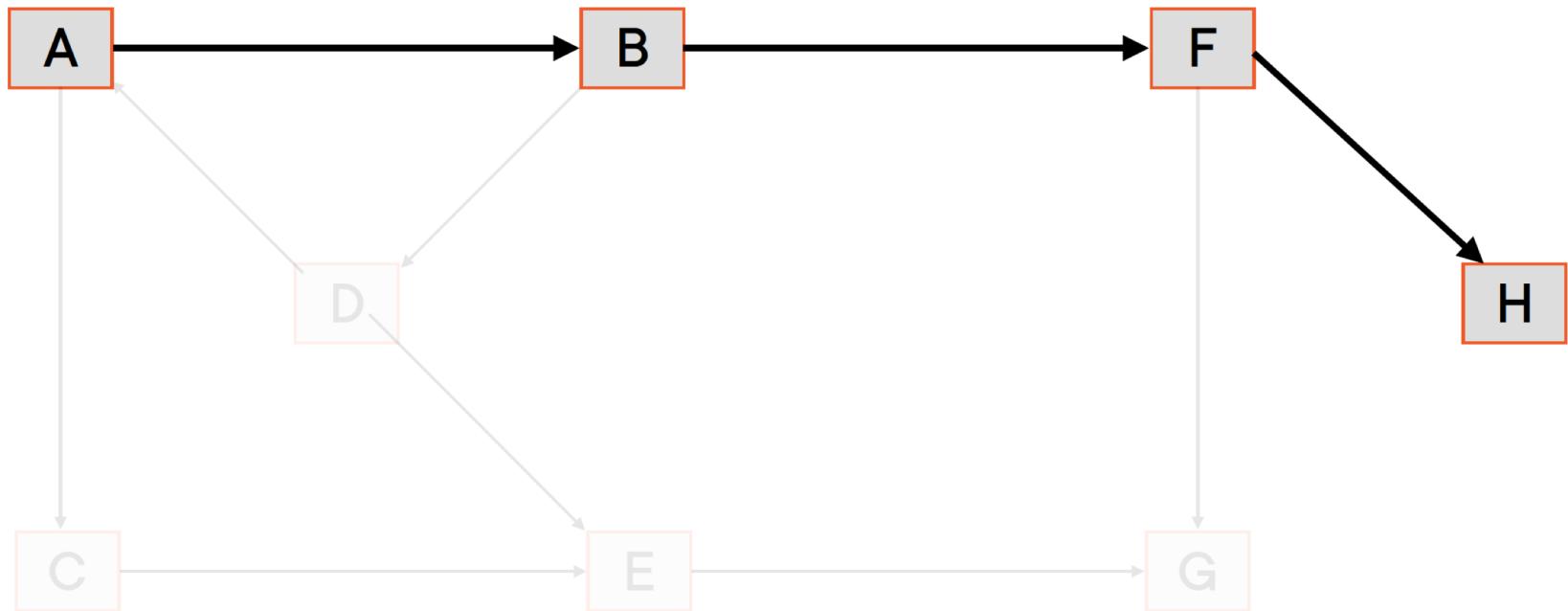
Le noeud B est adjacent au noeud F car il existe un chemin de B à F
(F n'est pas adjacent à B)

Degré entrant/sortant dans un graphe dirigé



Le degré entrant de F est 1, le degré sortant de F est 2

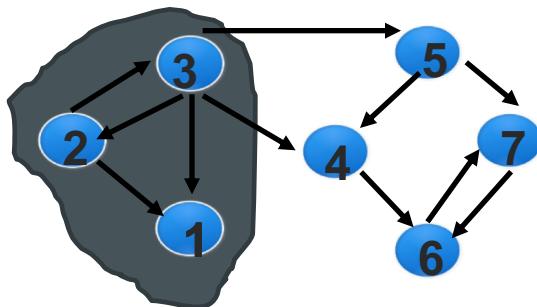
Chemin dans un graphe dirigé



Suite d'arcs consécutifs reliant le noeud A au noeud H (le chemin suit la direction des arcs)

CENTRALITÉ D'UN NOEUD

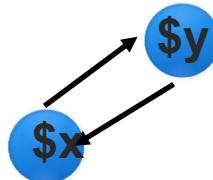
- Utile par exemple pour l'analyse de communautés
- Centralité basée sur le degré : $\text{in-degree}(v)/|E|$
- Centralité de proximité:
 - Distance moyenne des plus courts chemins vers ce nœud (graphe dirigé) →
nœud central = a une faible distance des autres nœuds
- Centralité d'intermédiarité de v :
 - La proportion des plus courts chemins entre deux autres sommets qui passent par v



Exemple : centralité d'intermédiarité de 4 ?
(2,3,4,6), (3,4,6), (5,4,6)
3/nombre total de plus courts chemins
3/6 = 0.5

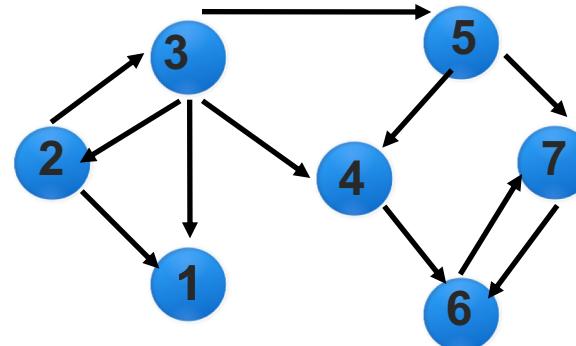
RECHERCHE DE MOTIFS

- Trouver toutes les instances d'un motif (peut contenir des étiquettes de sommets ou d'arrêtes)
- Exemple de motif :



Instanciation :

- $\$x = 2, \$y=3$ ($\$x=3, \$y=2$)
- $\$x = 7, \$y=6$ ($\$x=6, \$y=7$)



EXEMPLE DE RECHERCHE DE MOTIFS : TRIANGLES

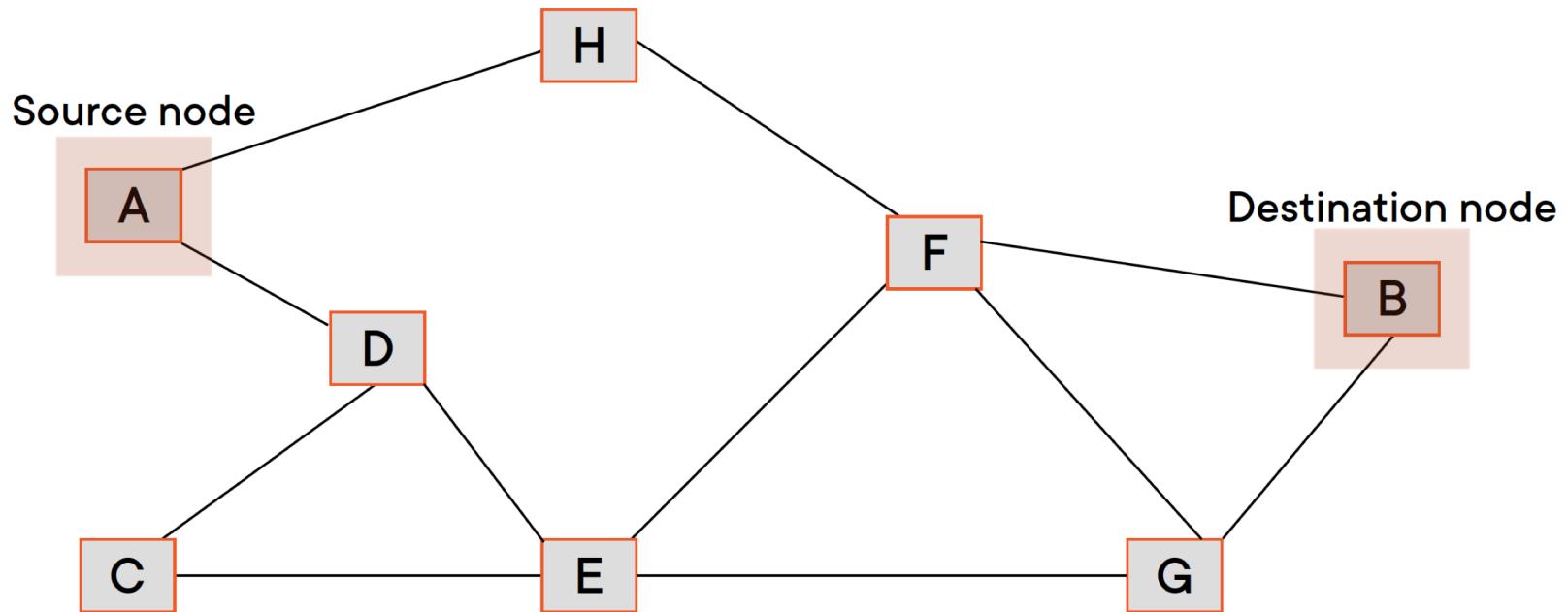
- Trouver les triangles ayant le motif : $\$x \rightarrow \$y \rightarrow \$z \rightarrow \x (nombre total de triangles= autre mesure de connexité)
- Il existe beaucoup d'algorithmes. Un algorithme naïf trouvera le même triangle plusieurs fois :

$\$x \rightarrow \$y \rightarrow \$z \rightarrow \x

$\$y \rightarrow \$z \rightarrow \$x \rightarrow \y

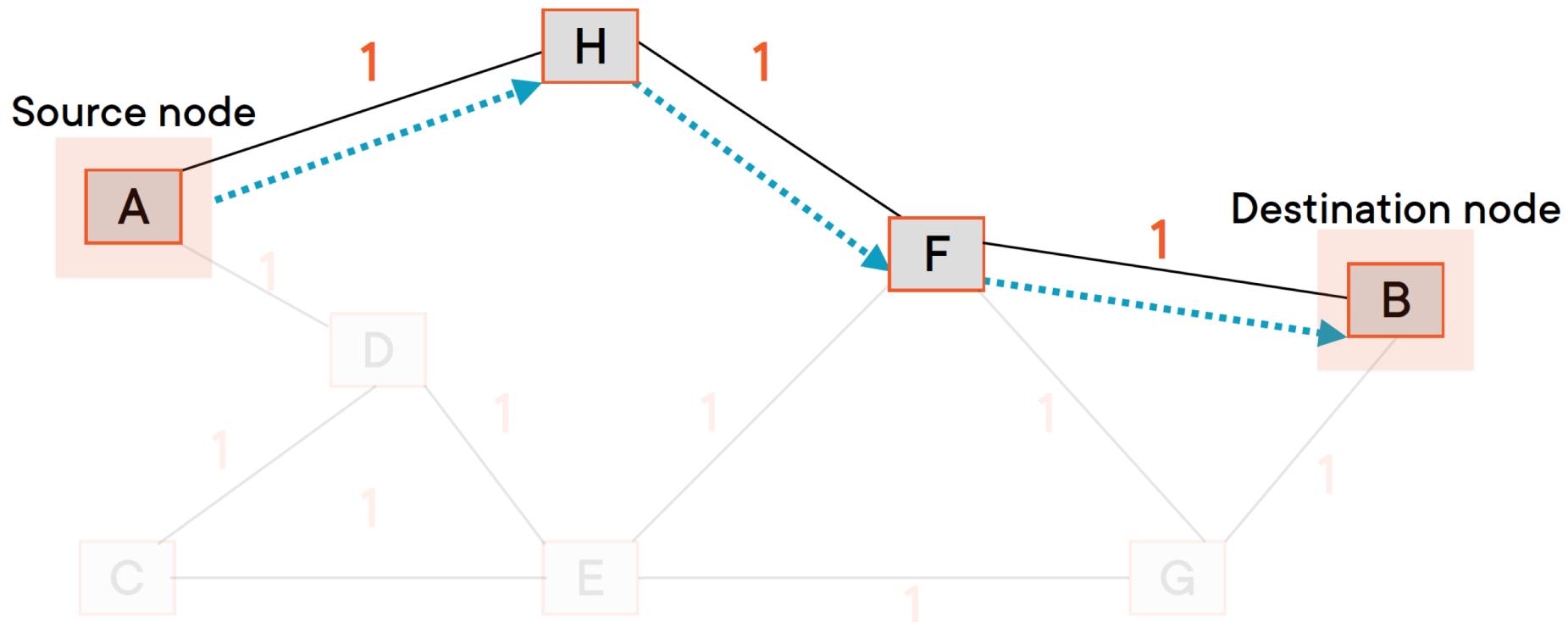
$\$z \rightarrow \$x \rightarrow \$y \rightarrow \z

Plus court chemin



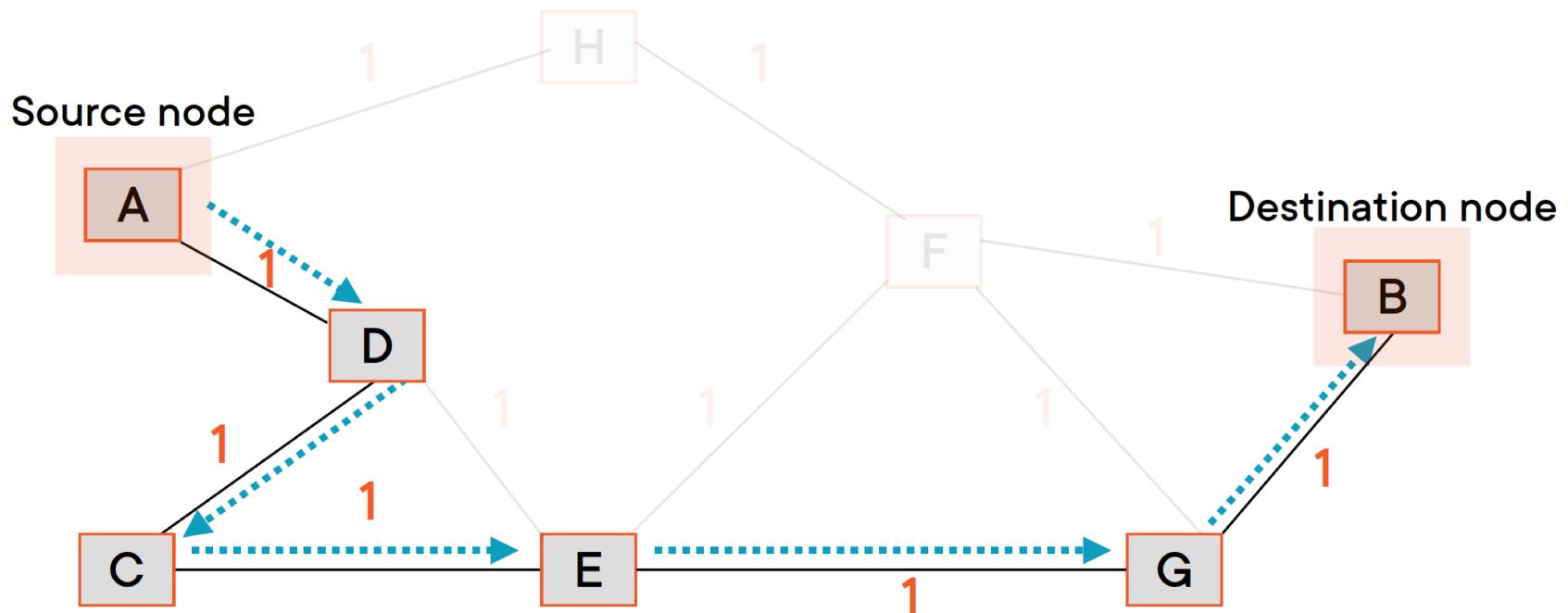
- Trouver le plus court chemin entre un nœud source et un nœud destination → dépend du poids des arêtes

Graphe non pondéré



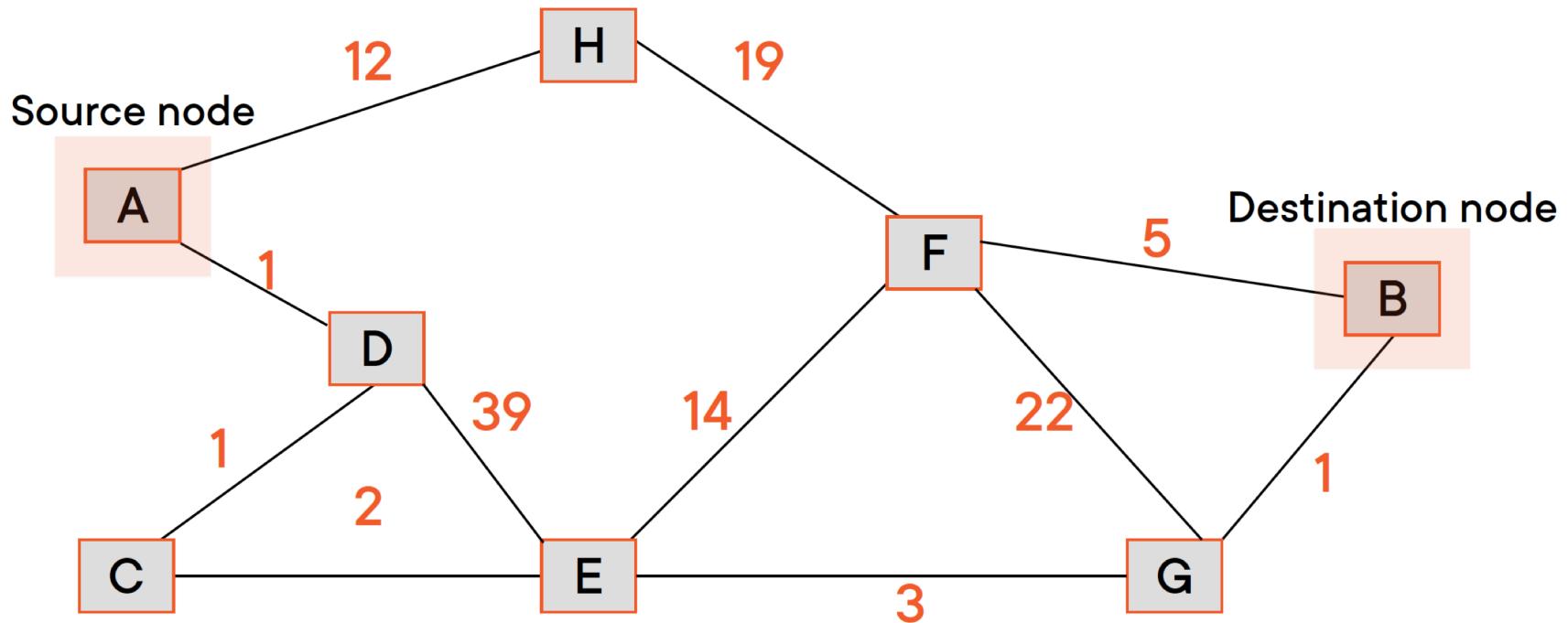
- Toutes les arêtes ont le même poids (=1) → le plus court chemin avec nombre minimum de sommets

Graphe non pondéré



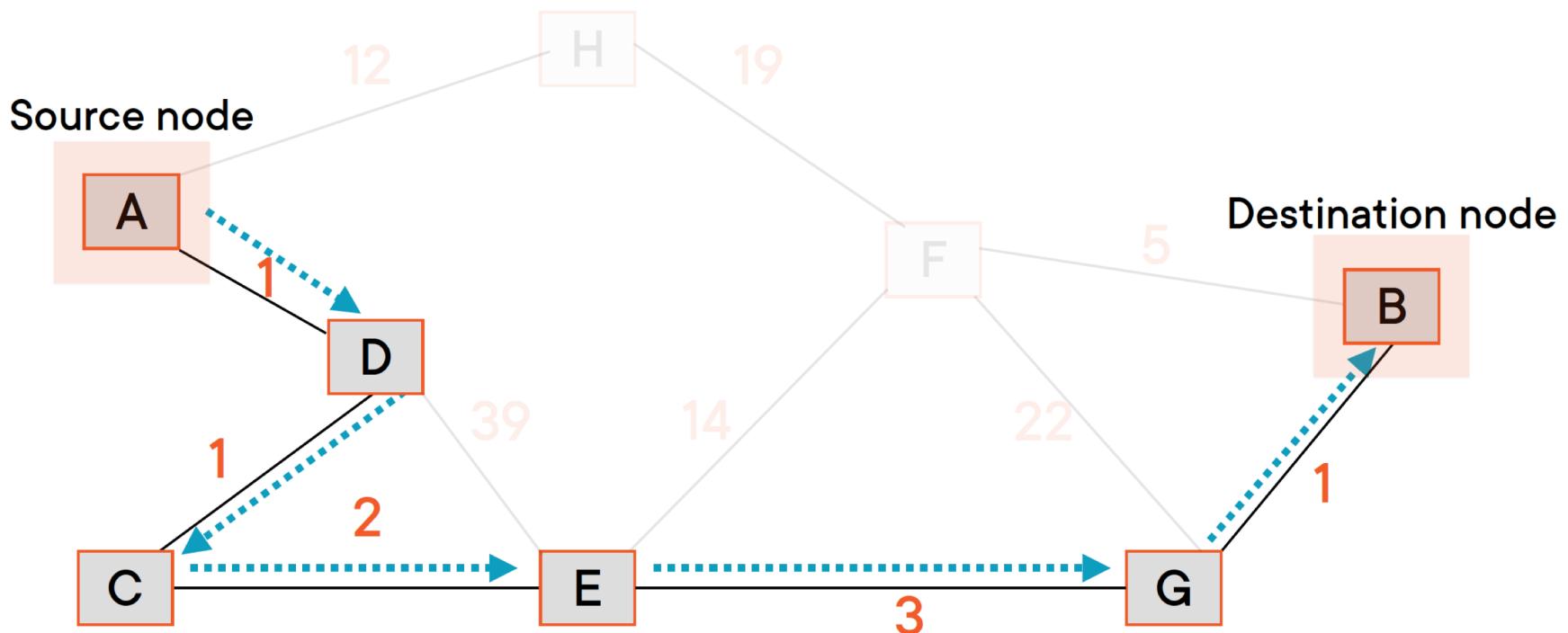
- Il existe d'autres chemins plus longs (5)

Graphe pondéré



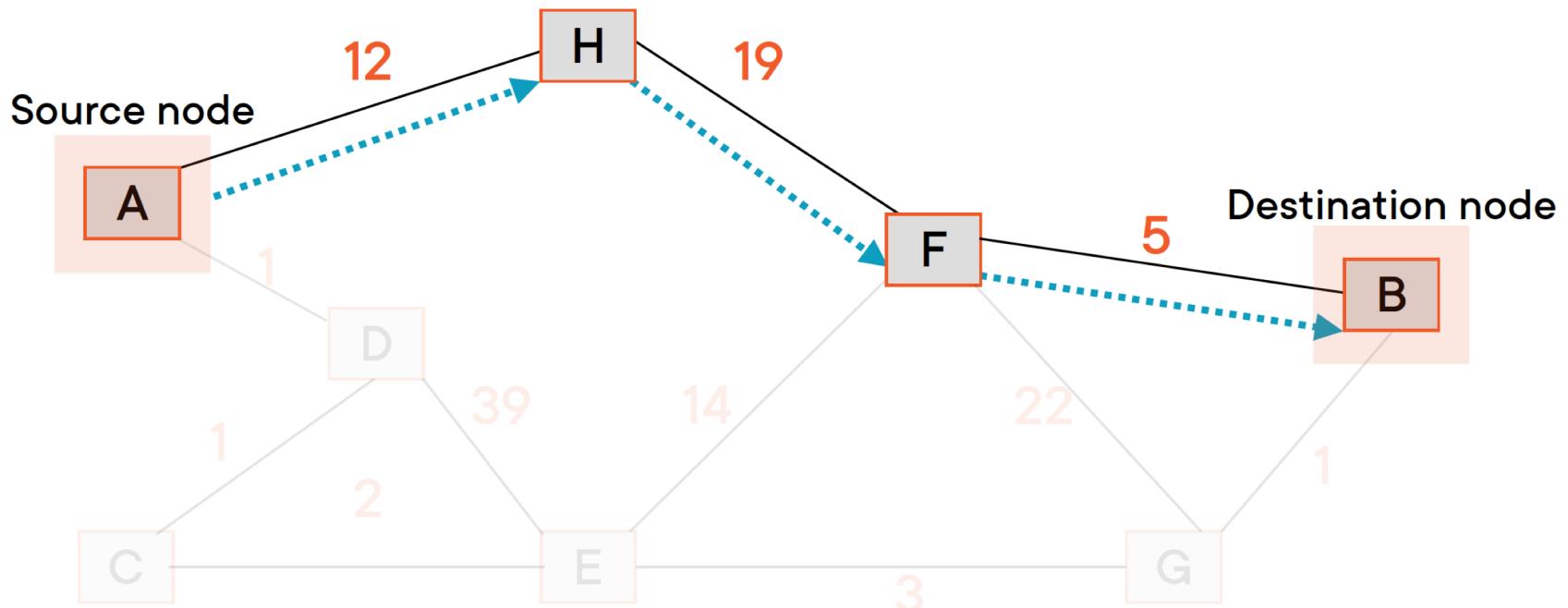
- Plus court chemin → dépend de la somme des poids des arêtes (algorithme de Dijkstra)

Graphe pondéré



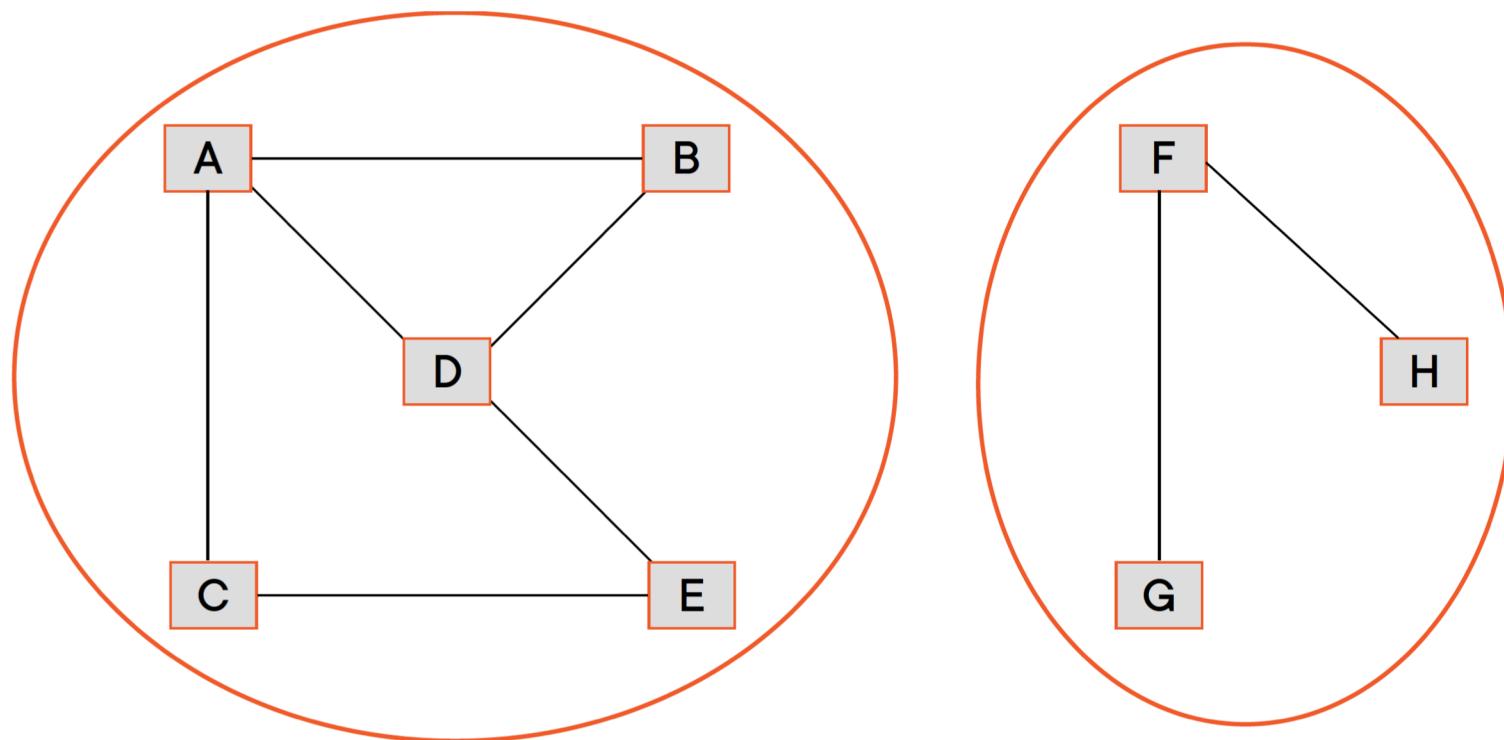
■ Coût du plus court chemin: $1+1+2+3+1 = 8$

Graphe pondéré



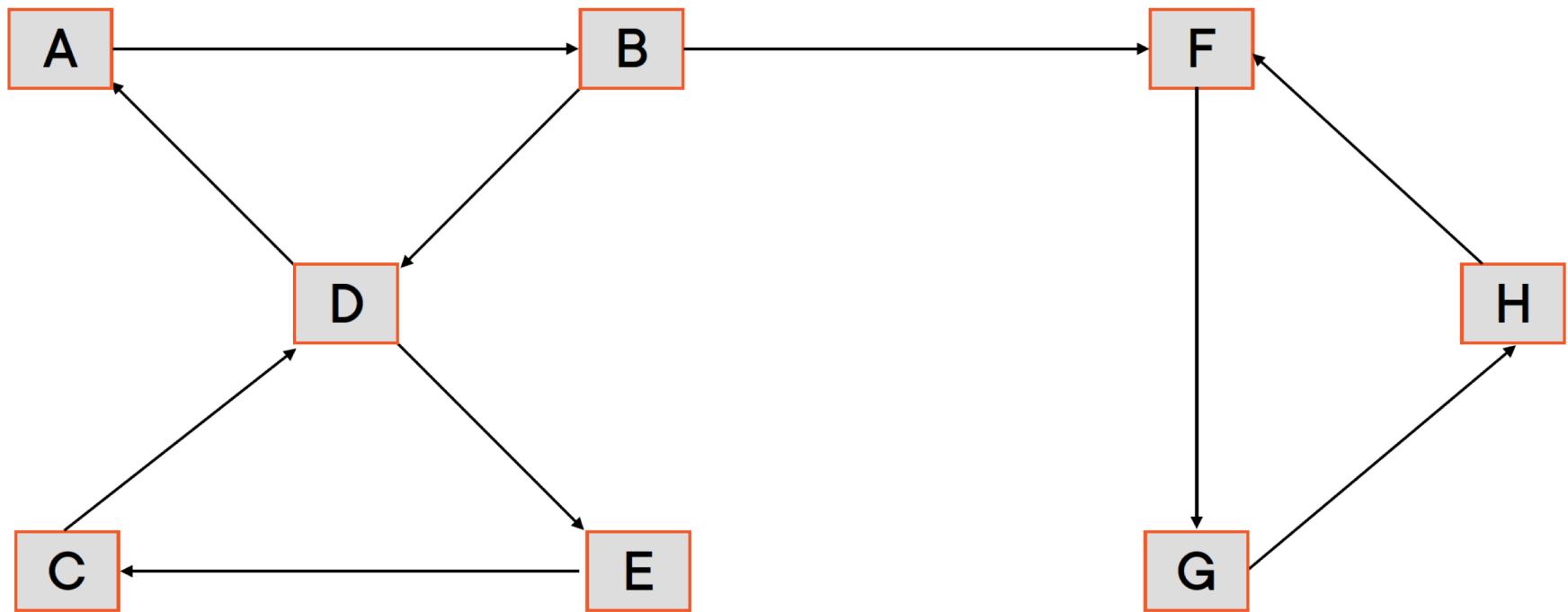
■ Coût du chemin: $12+19+5 = 36$

Composantes connexes: graphe non dirigé



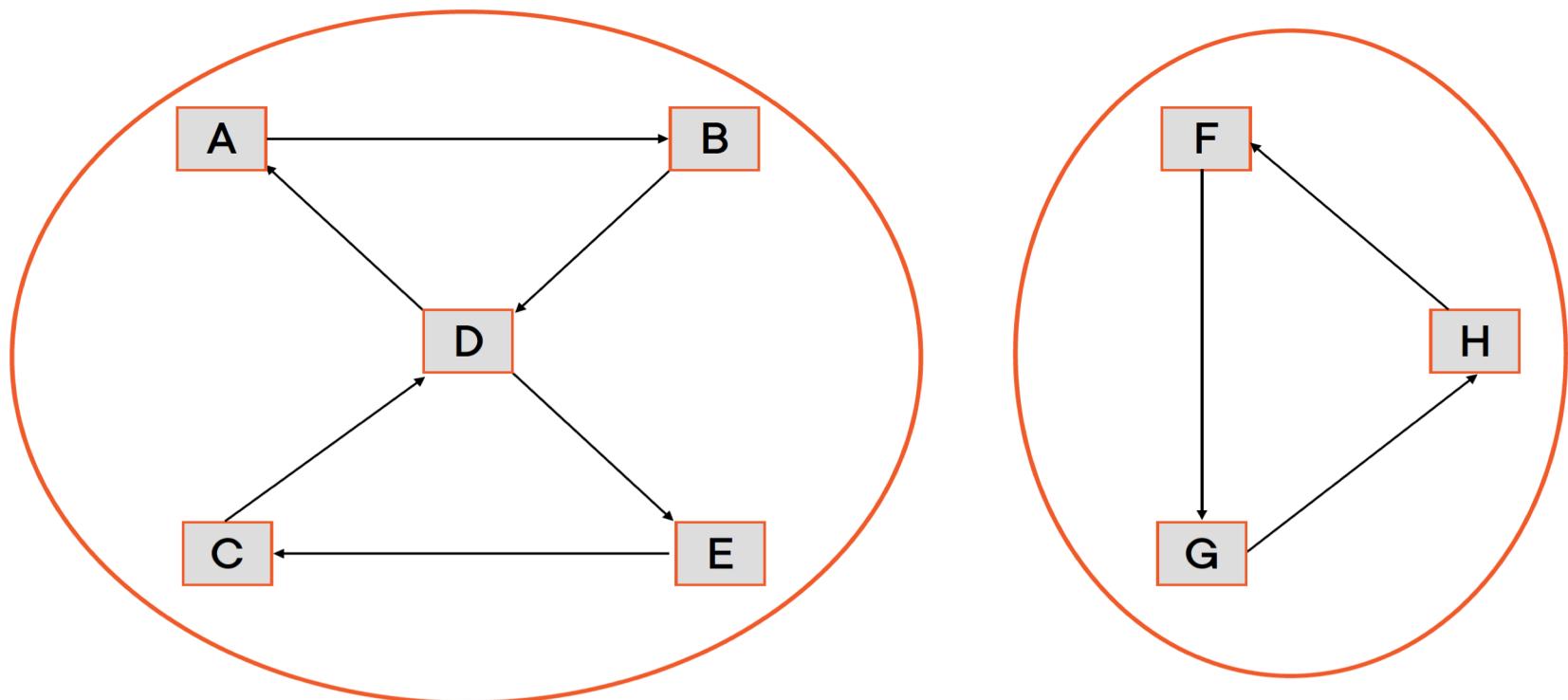
- Ensemble de sommets reliés deux à deux par un chemin (sous-graphe connexe maximal) qui ne sont pas connectés à d'autres nœuds du graphe

Composantes fortement connexes: graphe dirigé



- Il existe un chemin entre chaque couple de sommets (sous-graphe fortement connexe maximal)

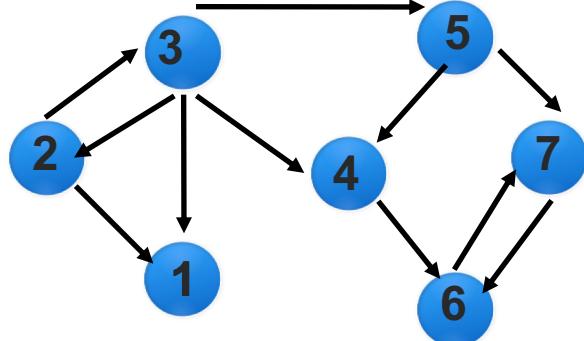
Composantes fortement connexes: graphe dirigé



- Deux composantes fortement connexes

CONNEXITÉ D'UN GRAPHE

- Degré de connexité : nombre minimum de sommets qu'on doit enlever afin de déconnecter le graphe.
- Utile pour décider si un graphe est "intéressant" pour un certain type d'analyse.



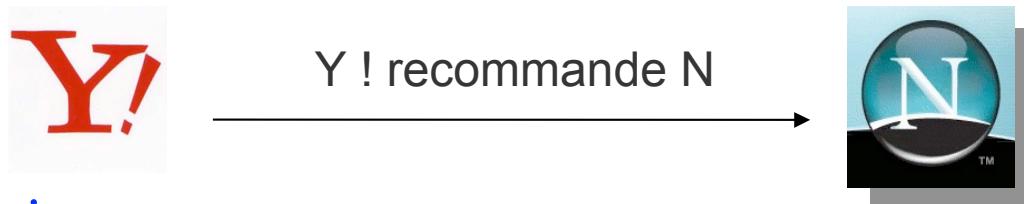
Exemple : degré de connexité du graphe ?
Degré 1 : enlever 3 on a deux composantes déconnectées

PageRank

PageRank: principe

]

Liens hypertexte = recommandations



Principe

- Les pages avec beaucoup de recommandations sont plus importantes
 - Importance aussi de *qui* donne la recommandation
 - être recommandé par Yahoo! est mieux que par X*
 - la recommandation compte moins si Yahoo! recommande beaucoup de pages*
- l'importance d'une page dépend du nombre et de la qualité
(importance de celui qui recommande) de ses liens entrants

[PageRank simplifié]

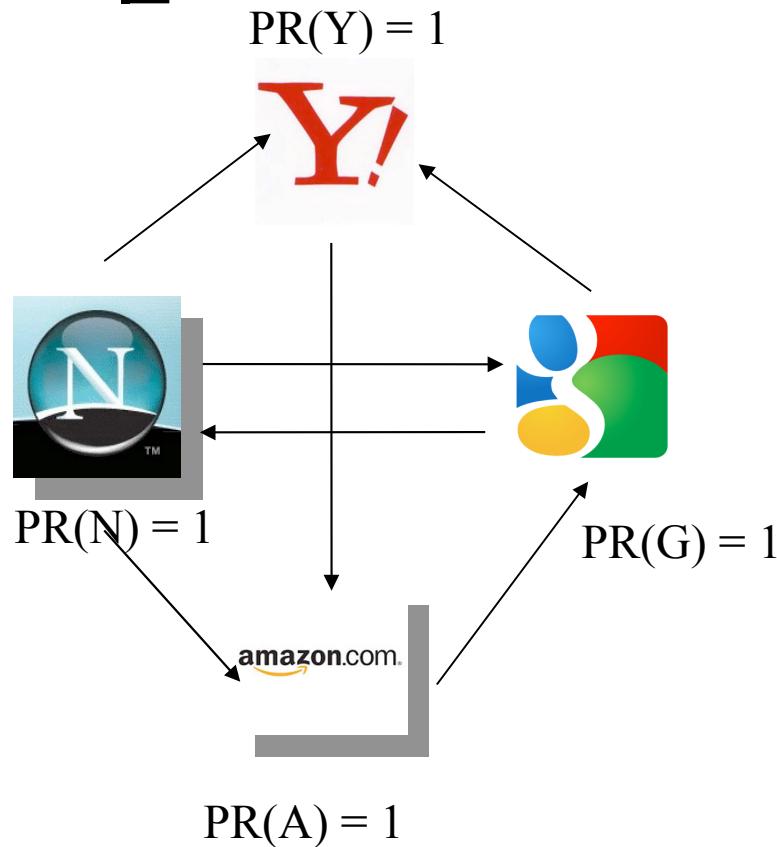
Recommandation donnée par $Y!$ à N :

$$\frac{PR(Y!)}{|out(Y!)|} \quad \text{où} \quad \left\{ \begin{array}{l} PR(Y!) = \text{l'importance de } Y! \\ |out(Y!)| = \text{nombre de liens sortants de } Y! \end{array} \right\}$$

Importance de Netscape est la somme de ses recommandations

$$PR(N) = \sum \frac{PR(P)}{|out(P)|} \quad P = \text{pages qui recommandent } N$$

Exemple



$$PR(A) = PR(N) / 3 + PR(Y)$$

$$PR(Y) = PR(N) / 3 + PR(G)/2$$

$$PR(N) = PR(G)/2$$

$$PR(G) = PR(A) + PR(N)/3$$

Calcul des valeurs PR

Résolution système linéaire

- 4 équations avec 4 inconnues
- pas de solution unique

→ ajouter la contrainte $PR(A) + PR(Y) + PR(N) + PR(G) = 1$ pour assurer l'unicité

Observation:

- système linéaire de grandes dimensions, beaucoup de pages sans liens sortants => les méthodes de calcul directes (ex. méthode de Gauss) sont plus coûteuses que les *méthodes itératives*

Représentation matricielle

On considère n pages, pour chaque page i , on note:

- $out(i)$ est l'ensemble de pages j référencées par i

$M(m_{ij})$ est la matrice d'adjacence associée au graphe du Web

- m_{ij} : fraction de l'importance de j qui est donnée à i ($m_{ij} = 1/|out(j)|$, si j a des liens sortants, $p_{ij} = 0$ dans le cas contraire)
- ligne i = fractions d'importance reçues par i
- colonne j = distribution de l'importance de j (pour les pages j avec des liens sortants, la somme des éléments sur les colonnes est 1)

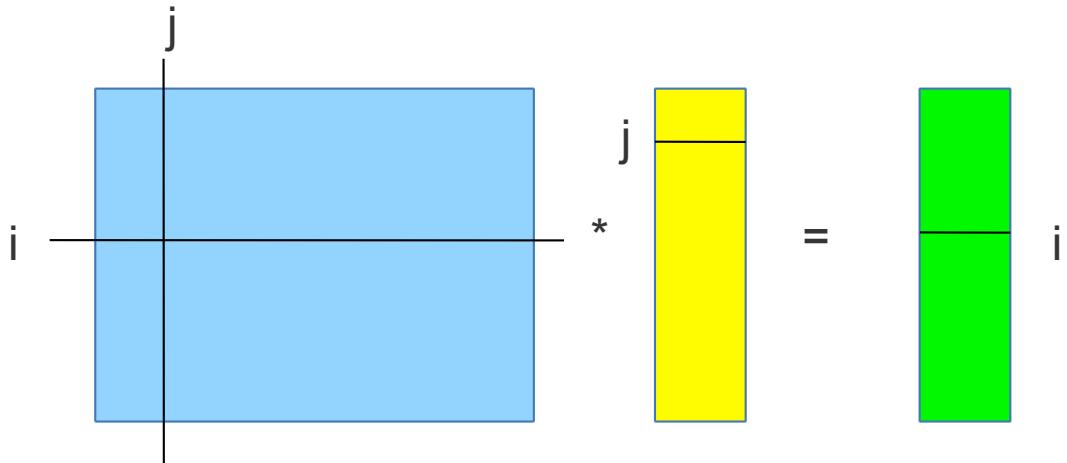
$PR(PR_1, PR_2, \dots, PR_n)$ est le vecteur des inconnues (importance)

PR_i est l'importance de la page i

Exemple

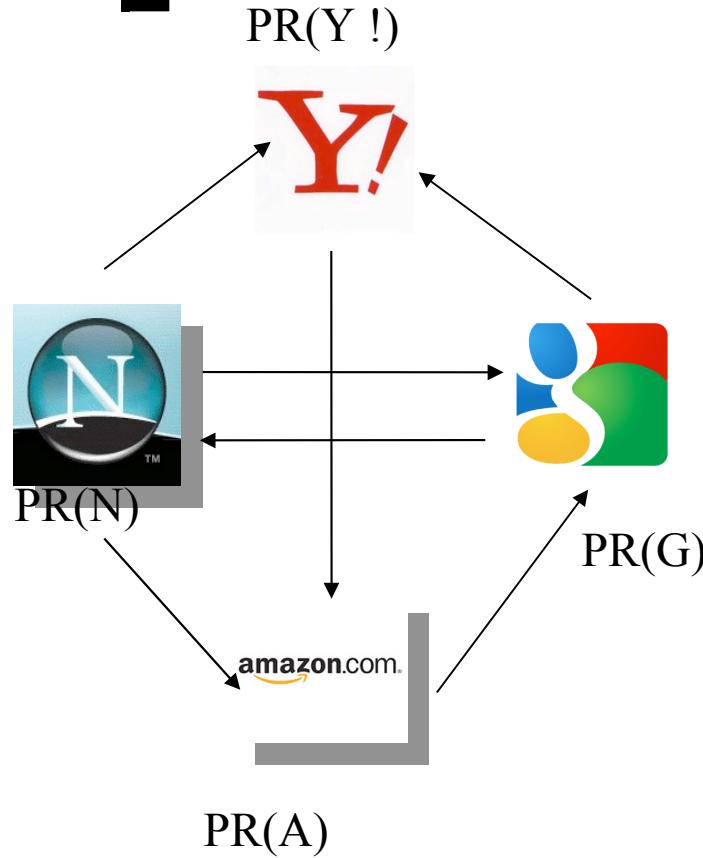
]

Mise à jour de PR_i :
$$PR_i = \sum \frac{PR_j}{|out(j)|}$$



$$PR_i = \sum m_{ij} * PR_j$$

Exemple



$$\begin{aligned}
 PR(A) &= PR(N)/3 + PR(Y) \\
 PR(Y) &= PR(N)/3 + PR(G)/2 \\
 PR(N) &= PR(G)/2 \\
 PR(G) &= PR(A) + PR(N)/3
 \end{aligned}$$

$$\begin{array}{c}
 \begin{array}{c}
 PR(Y) \\ PR(N) \\ PR(A) \\ PR(G)
 \end{array} = \begin{array}{c}
 Y \\ N \\ A \\ G
 \end{array} \begin{array}{c|c|c|c}
 Y & N & A & G \\
 \hline
 1/3 & & & 1/2 \\
 \hline
 & & & 1/2 \\
 \hline
 1 & 1/3 & & \\
 \hline
 & 1/3 & 1 &
 \end{array} * \begin{array}{c}
 PR(Y) \\ PR(N) \\ PR(A) \\ PR(G)
 \end{array} \\
 PR = M *
 \end{array}$$

Algorithme de calcul itératif

- Un graphe avec n noeuds
- Initialisation : $PR^0 = [1, \dots, 1]$
- À chaque itération k, recalculer $PR^{(k)}$

$$PR^{(k)} = M * PR^{(k-1)}$$

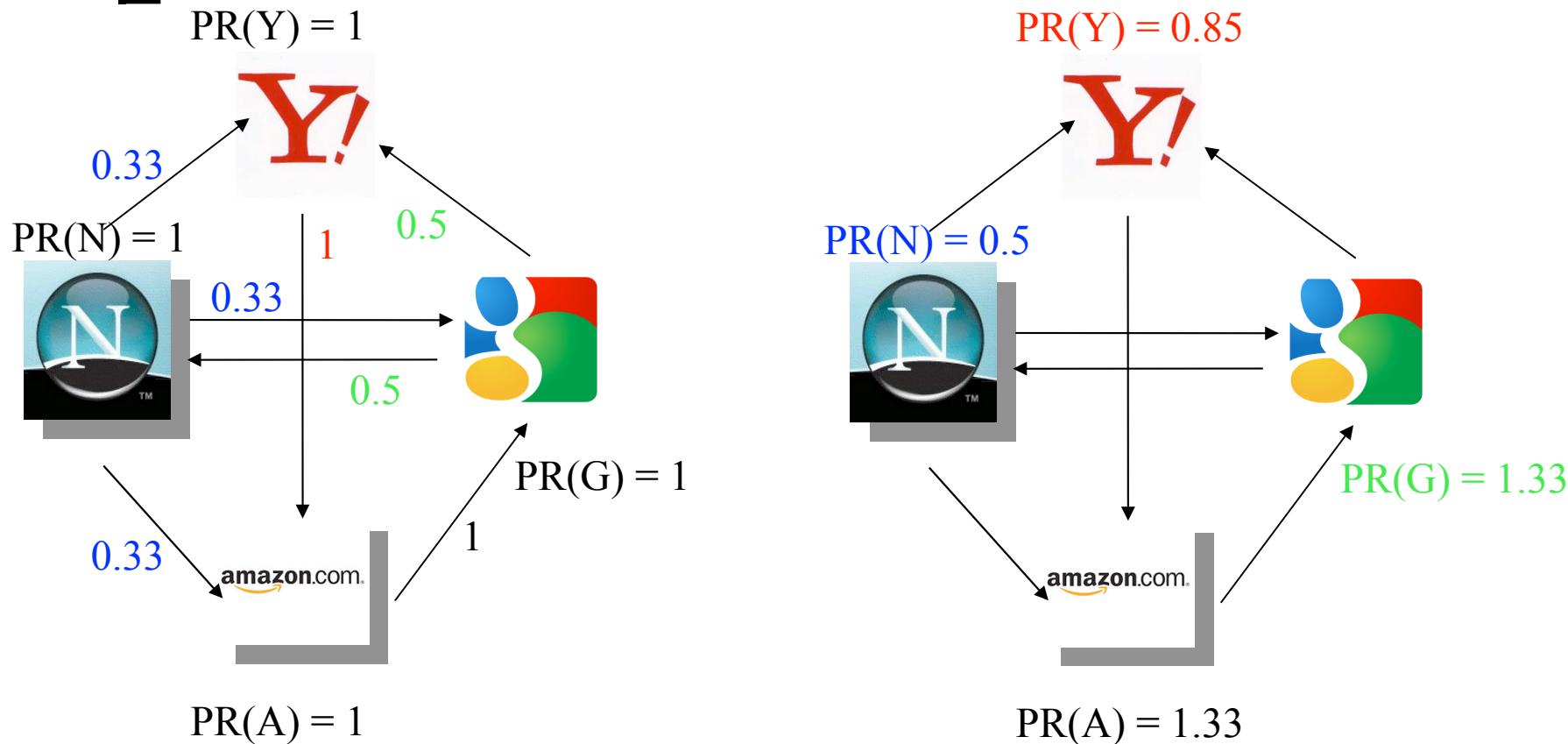
- Arrêt du calcul (convergence) :

$$\frac{\sum |PR_i^k - PR_i^{(k-1)}|}{\|PR^k\|_1} < \varepsilon, \varepsilon \in (0, 1)$$

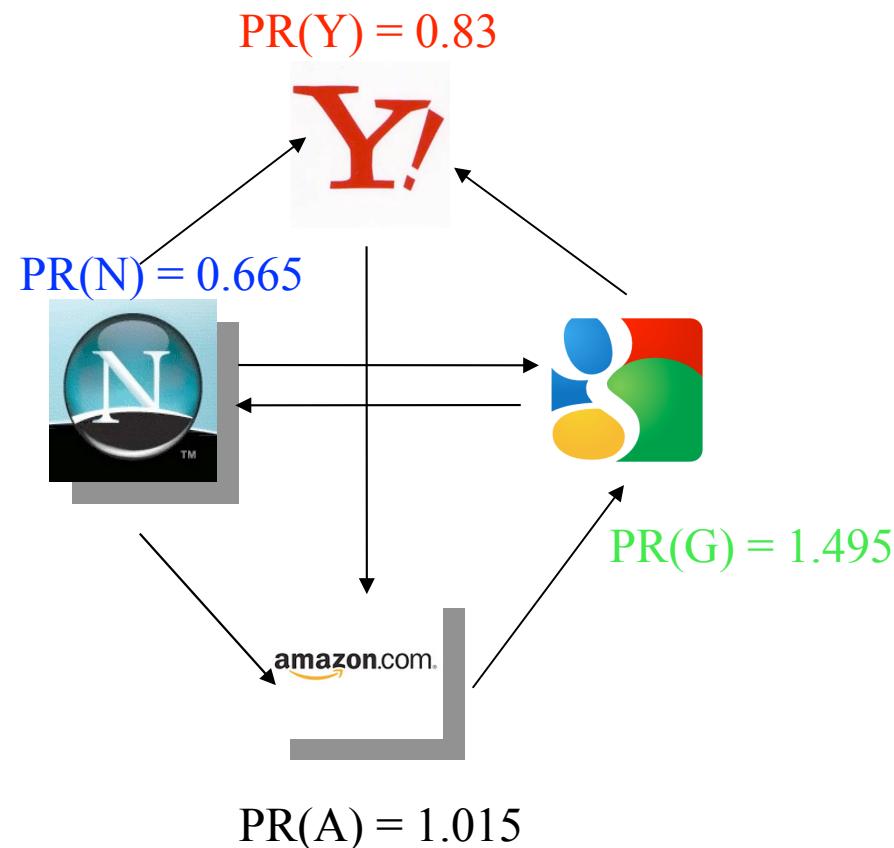
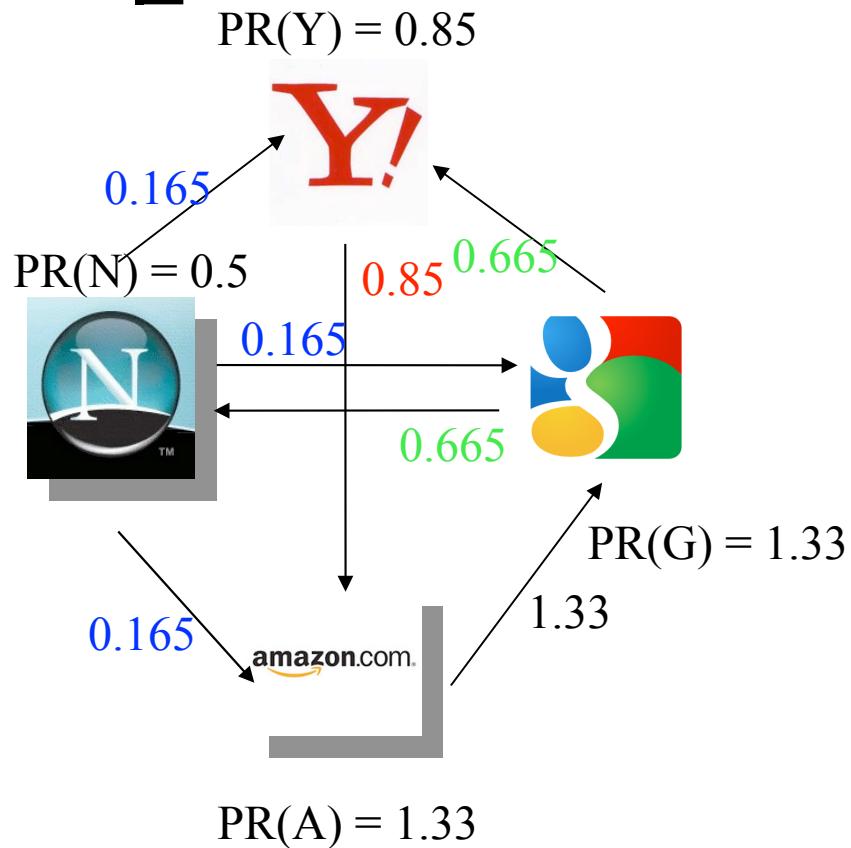
Le vecteur PR obtenu à la convergence satisfait la condition :

$$PR = M * PR$$

[Exemple – Itération 1]



[Exemple – Itération 2]



[Algorithme itératif complet]

Un graphe avec n nœuds

- Initialisation : $PR^0 = [1/N, \dots, 1/N]$
- Vecteur ajouté à chaque itération : $V = [1/N, \dots, 1/N]$
- À chaque itération k, recalculer $PR^{(k)}$

$$PR^{(k)} = d * M * PR^{(k-1)} + (1-d) * V$$

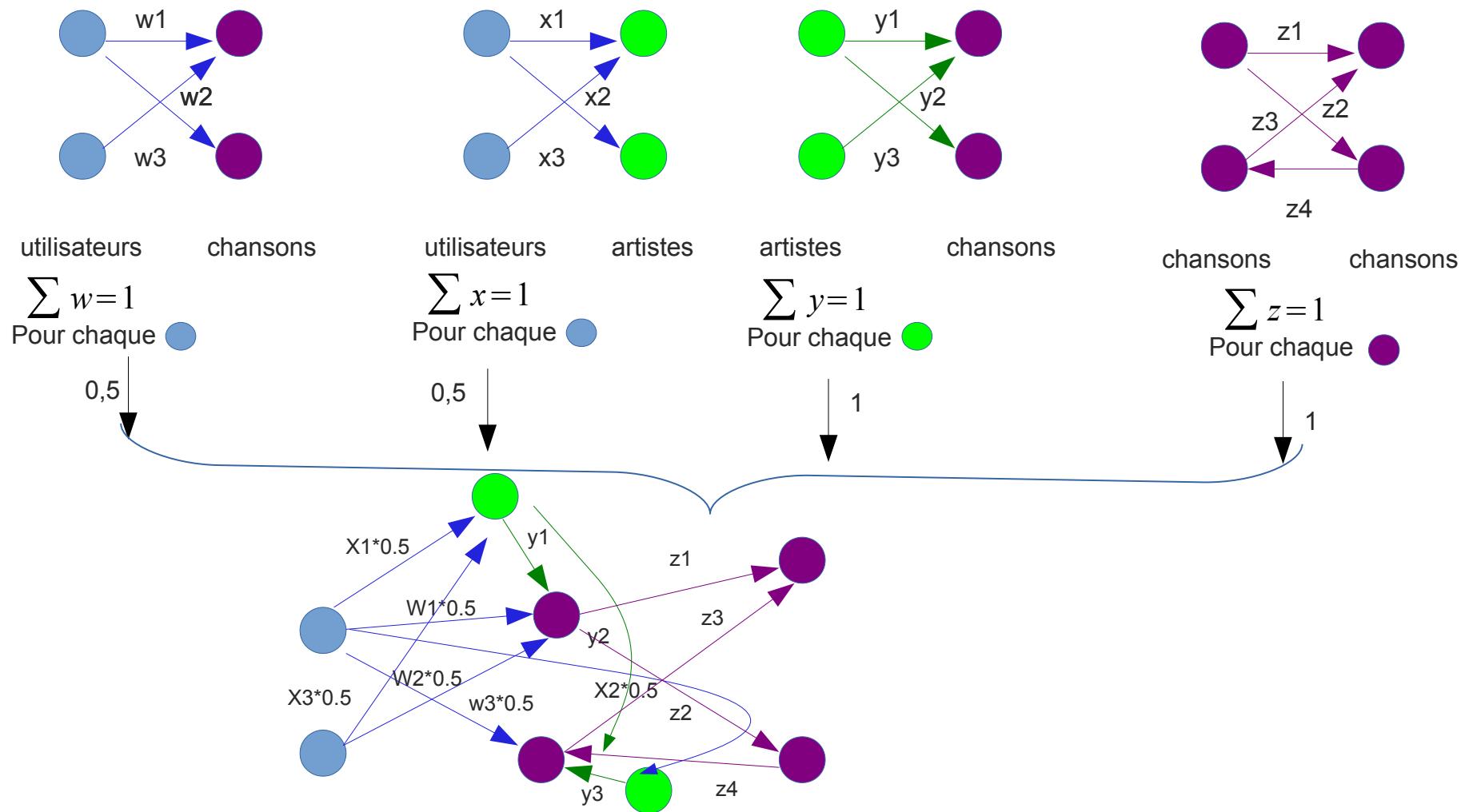
$$(ou équivalent : \forall i: PR_i^k = d * \sum \frac{PR_j}{|out(j)|} + \frac{(1-d)}{N})$$

arrêt lorsque :

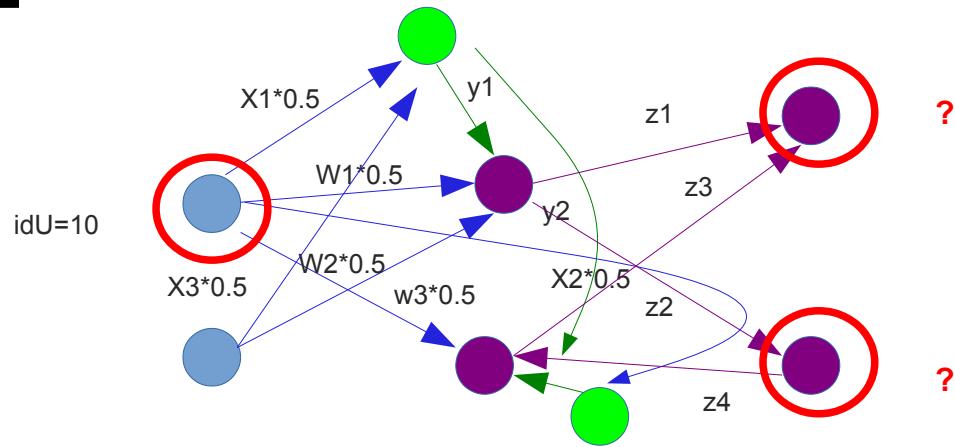
$$\frac{\sum |PR_i^k - PR_i^{(k-1)}|}{\|PR^k\|_1} < \varepsilon, \varepsilon \in [0,1]$$

le facteur de décroissance d (valeur habituelle 0.85) est utilisé pour assurer l'unicité du vecteur PR calculé et la convergence du calcul itératif

TME: Graphe utilisateurs, artistes, chansons



[TME: PPR]



Itération k :

$$si \ i = 10 : PR_i^k = d * \sum PR_j * poids_{j \rightarrow i} + (1 - d)$$

$$si \ i \neq 10 : PR_i^k = d * \sum PR_j * poids_{j \rightarrow i}$$

Initialisation : $PR_{10}^0 = 1, PR_i^0 = 0 \ si \ i \neq 10$

Références

<https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ach04.html>

<http://ampcamp.berkeley.edu/wp-content/uploads/2012/06/matei-zaharia-amp-camp-2012-advanced-spark.pdf>

https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

Mining of Massive Datasets (Chapitre 5) : <http://infolab.stanford.edu/~ullman/mmds/bookL.pdf>

Graph Algorithms: Practical Examples in Apache Spark and Neo4j (<https://neo4j.com/graph-algorithms-book/>)