

# [ Algorithmes de calcul sur les graphes ]

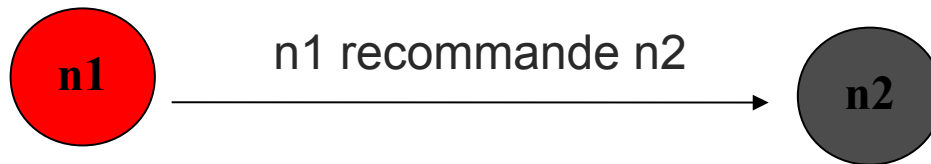
[

]

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]

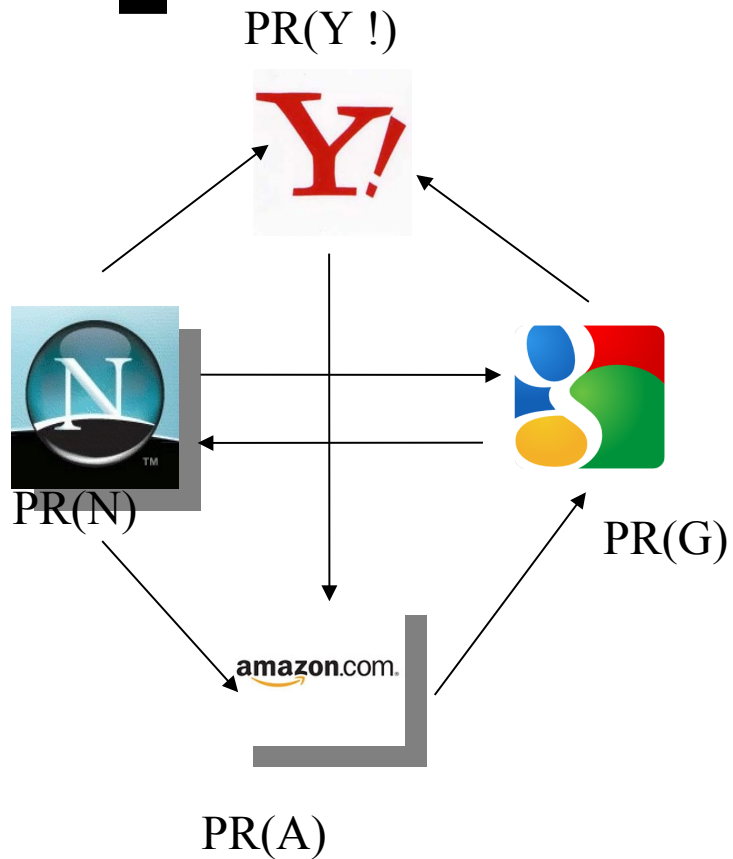
Recommandation donnée par **n1** à **n2** :

$$PR(n1) * \frac{1}{|out(n1)|} \quad \text{où} \quad \left\{ \begin{array}{l} PR(n1) = \text{l'importance de } n1 \\ |out(n1)| = \text{nombre de liens sortants de } n1 \end{array} \right\}$$

Importance de **n2** est la somme de ses recommandations

$$PR(n2) = \sum_i PR(ni) * \frac{1}{|out(ni)|} \quad ni = \text{pages qui recommandent } n2$$

# [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  $i$

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

- $w_{ij}$ : fraction de l'importance de  $j$  qui est donnée à  $i$  ( $w_{ij} = 1/|out(j)|$ , si  $j$  a des liens sortants,  $w_{ij} = 0$  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)

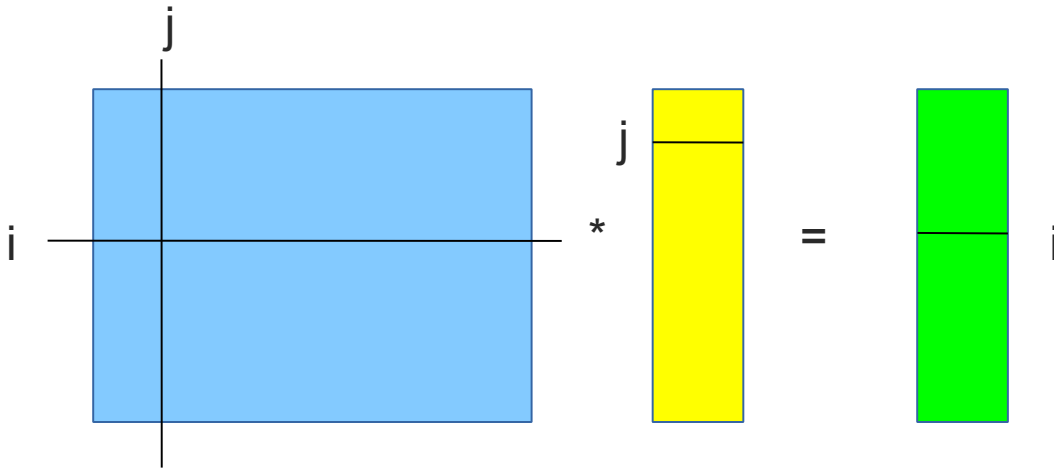
$PR_i$  est l'importance de la page  $i$

# [Exemple]

Mise à jour de  $PR_i$  :

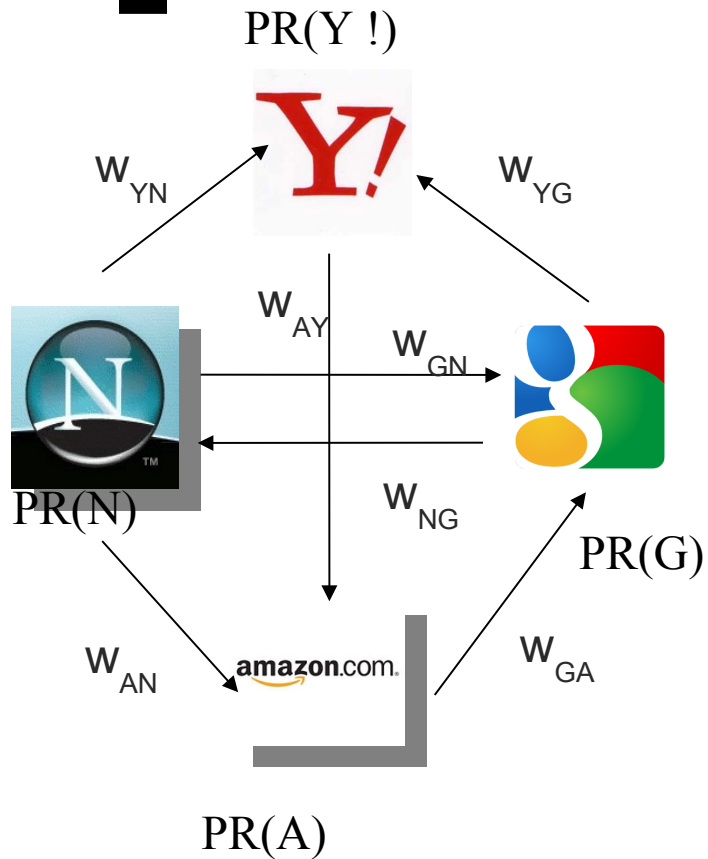
$$PR_i = \sum_j w_{ij} * PR_j$$

$$PR_i = \sum_j \frac{1}{|out(j)|} * PR_j$$





# [Exemple]



$$PR(A) = PR(N) * w_{AN} + PR(Y) * w_{AY}$$

$$PR(Y) = PR(N) * w_{YN} + PR(G) * w_{YG}$$

$$PR(N) = PR(G) * w_{NG}$$

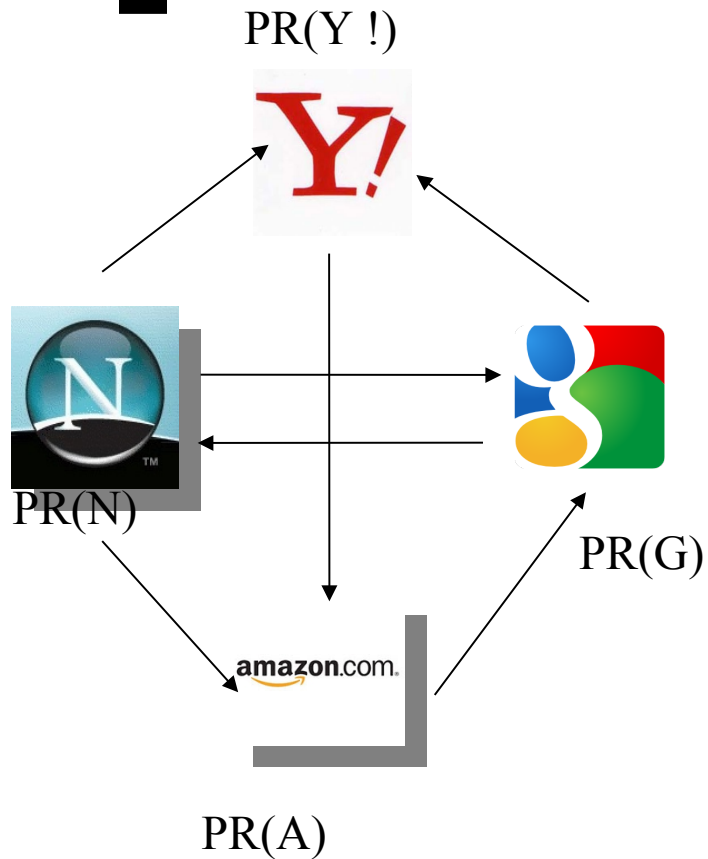
$$PR(G) = PR(A) * w_{GA} + PR(N) * w_{GN}$$

$$PR = M * PR$$

	Y	N	A	G
Y		$w_{YN}$		$w_{YG}$
N				$w_{NG}$
A	$w_{AY}$	$w_{AN}$		
G		$w_{GN}$	$w_{GA}$	



# [Exemple



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

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

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

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

$W_{YN} = 1/3$

		N	A	G
PR(Y)	Y	1/3		1/2
PR(N)	N			1/2
PR(A)	A	1	1/3	
PR(G)	G	1/3	1	

PR = M \* PR

# [ Algorithme de calcul itératif ]

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

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

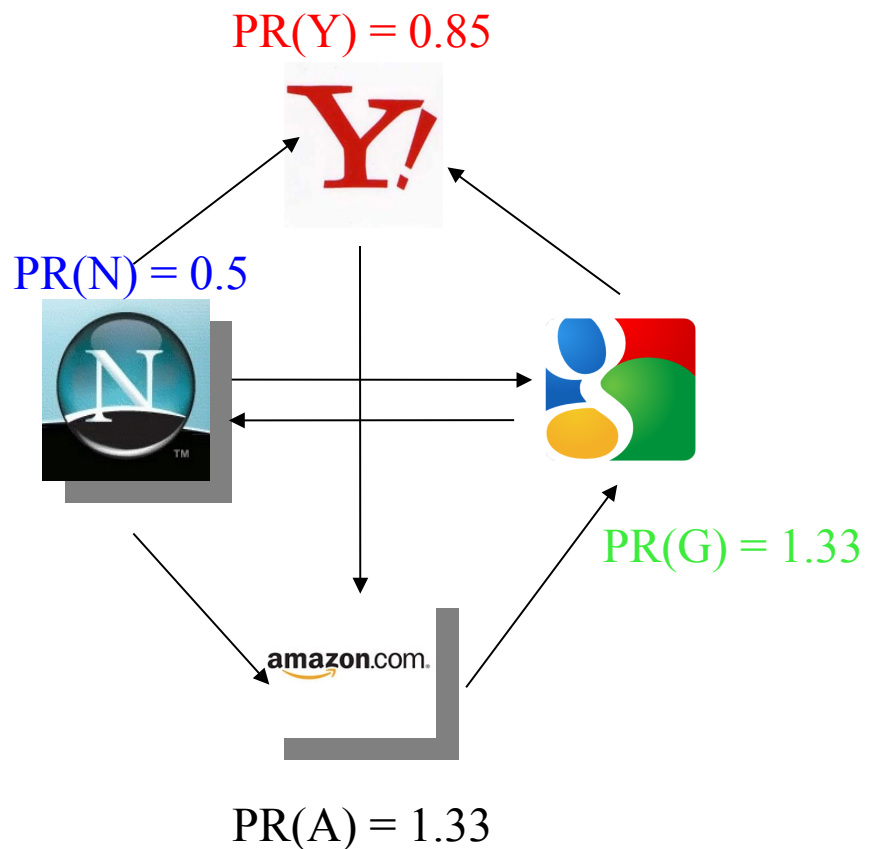
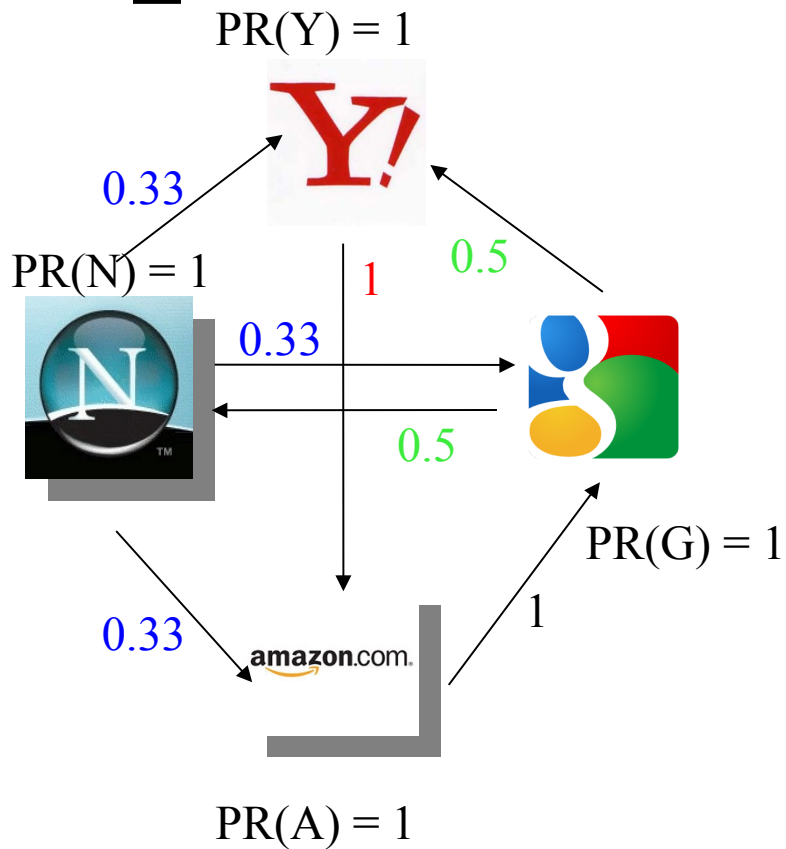
- Arrêt du calcul (convergence) :

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

Le vecteur PR obtenu à la convergence satisfait la condition :

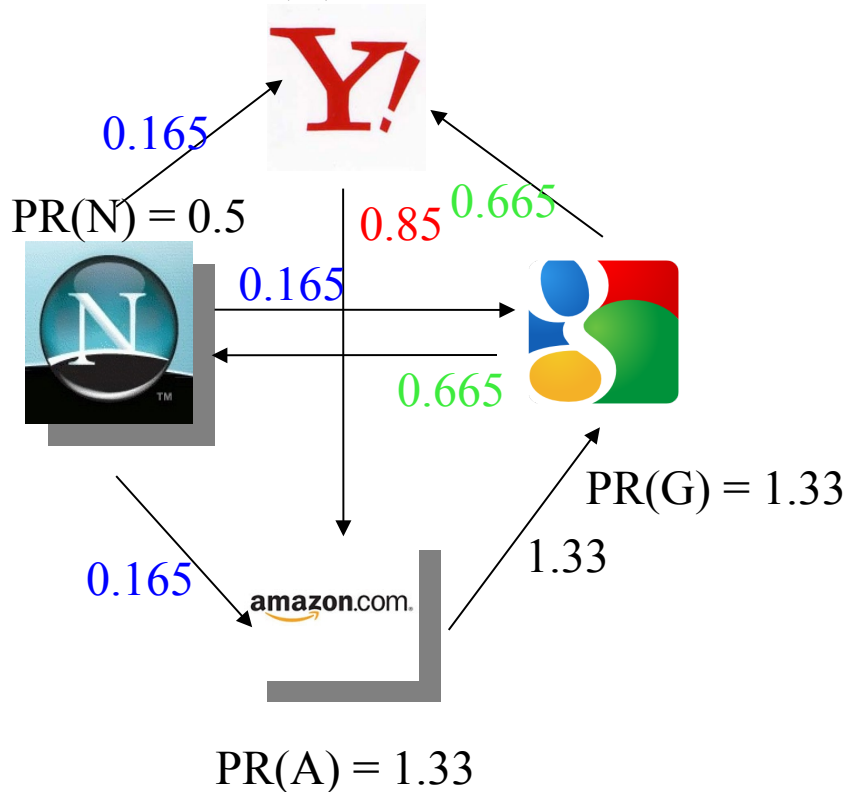
$$PR = M * PR$$

# Exemple – Itération 1

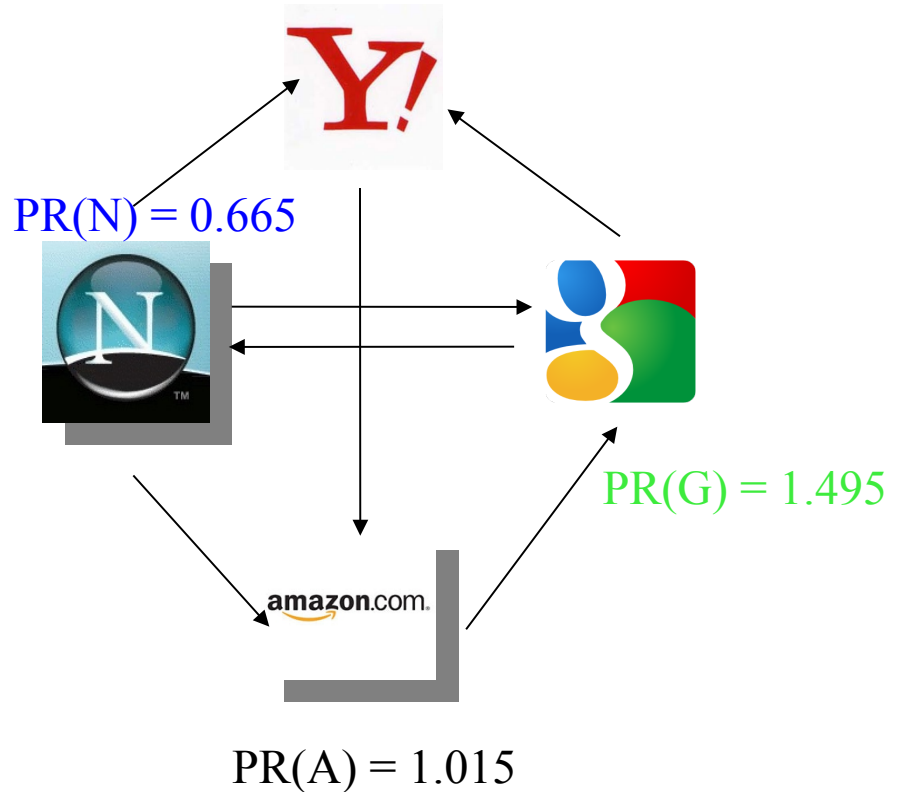


# Exemple – Itération 2

$$PR(Y) = 0.85$$



$$PR(Y) = 0.83$$



# [ Algorithme itératif complet ]

Un graphe avec  $N$  nœuds

■ Initialisation :  $PR^0 = [1/N, \dots, 1/N]$

■ Vecteur ajouté à chaque itération :  $P = [1/N, \dots, 1/N]$

■ À chaque itération  $k$ , recalculer  $PR^{(k)}$

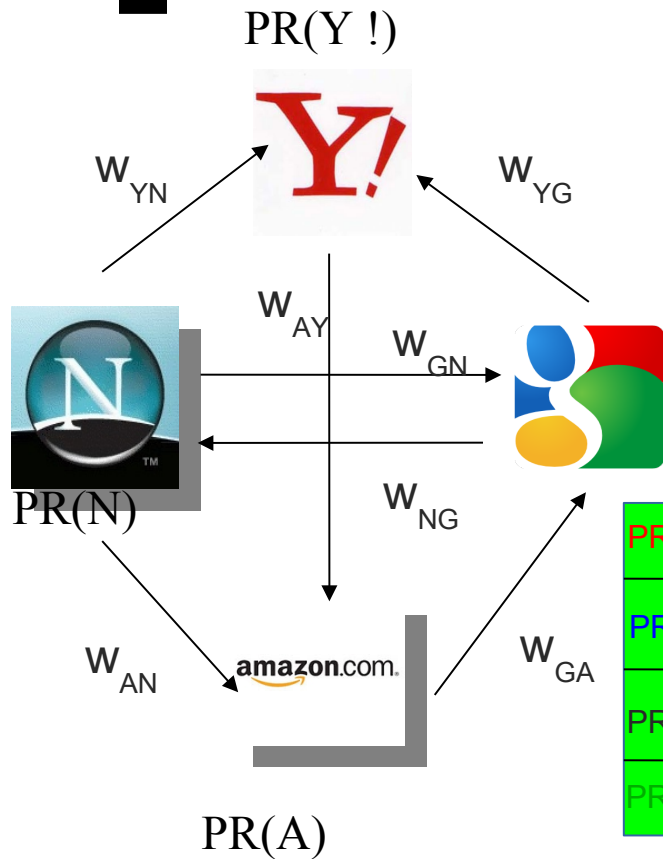
$$PR^{(k)} = d * M * PR^{(k-1)} + (1-d) * P$$

(ou équivalent :  $\forall i : PR_i^k = d * \sum_j PR_j^{k-1} * w_{ij} + (1-d) * p_i$  )

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

- 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
- Personnalisation (importance spécifique à un ensemble  $E$  de noeuds):
  - $P = [p_1, \dots, p_N]$  ( $p_i = 1/E$  si  $i$  dans  $E$ ,  $p_i = 0$  sinon)

# [Exemple]



$$PR(A) = d * (PR(N) * w_{AN} + PR(Y) * w_{AY}) + (1-d) * p_A$$

$$PR(Y) = d * (PR(N) * w_{YN} + PR(G) * w_{YG}) + (1-d) * p_Y$$

$$PR(N) = d * (PR(G) * w_{GN}) + (1-d) * p_N$$

$$PR(G) = d * (PR(A) * w_{GA} + PR(N) * w_{GN}) + (1-d) * p_G$$

$$\begin{array}{c}
 \begin{array}{c} PR(Y) \\ PR(N) \\ PR(A) \\ PR(G) \end{array} = d * \begin{array}{c} Y \\ N \\ A \\ G \end{array} \begin{array}{c|c|c|c} Y & N & A & G \\ \hline & w_{YN} & & w_{YG} \\ \hline & & & w_{NG} \\ \hline w_{AY} & w_{AN} & & \\ \hline w_{GN} & & w_{GA} & \end{array} * \begin{array}{c} PR(Y) \\ PR(N) \\ PR(A) \\ PR(G) \end{array} + (1-d) * \begin{array}{c} PY \\ PN \\ PA \\ PG \end{array}
 \end{array}$$

$PR = d * M * PR + (1-d) * P$

# [ PageRank en MapReduce ]

## ► SPLIT

- Fragmentation en ensembles de données homogènes
- Distribution sur un ensemble de nœuds (mémoire, disque)

## ► MAP et REDUCE

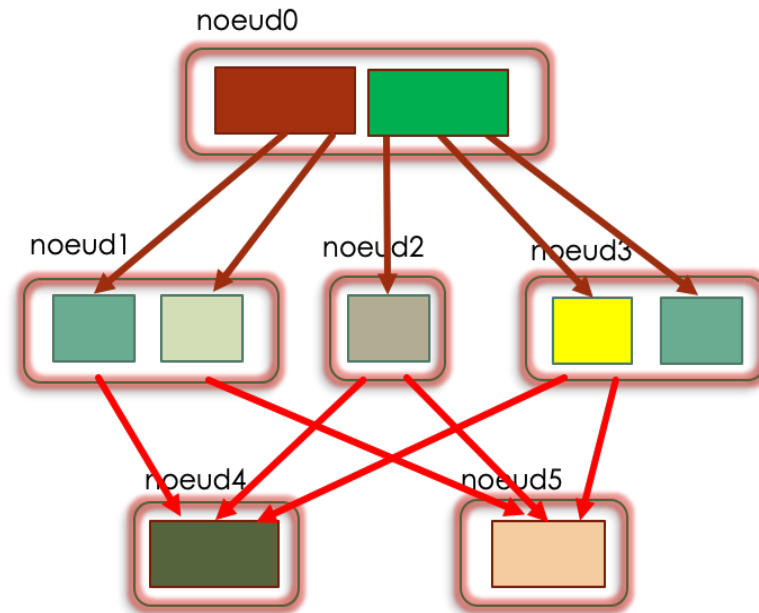
- Tâches homogènes
- Données isolées

## ► SHUFFLE

- Synchronisation

## ► Optimisation

- Stratégies de fragmentation et de placement de données
- Degré de parallélisme dépend de la taille de données



SPLIT

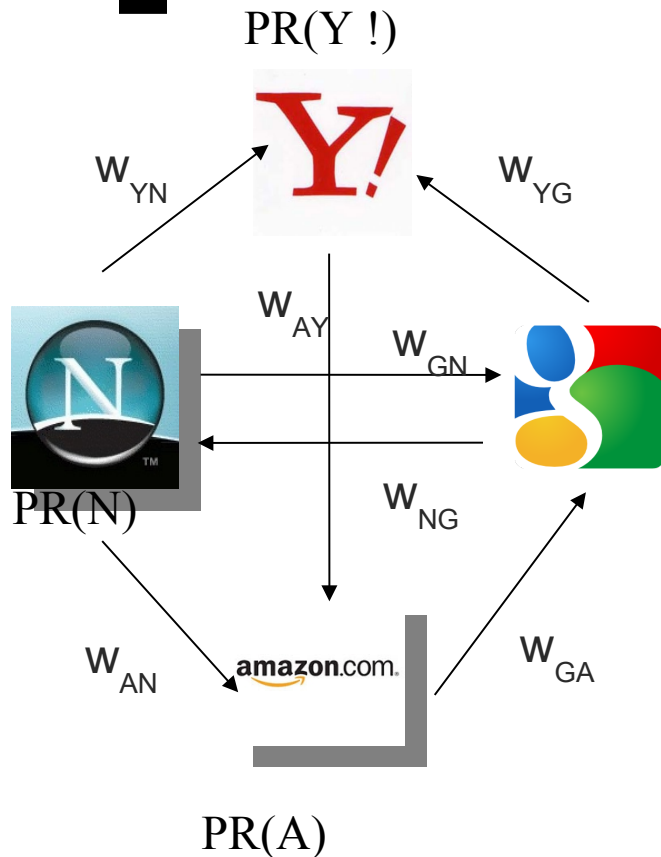
MAP

SHUFFLE

REDUCE



# PageRank en MapReduce



Décomposition du calcul à chaque itération **K** en deux étapes :

MAP : calcul des valeurs  $d * PR_j^k * w_{ij}$

REDUCE : Somme des valeurs précédentes et ajout des valeurs de personnalisation

$PR(G)$

$$PR(A) = d * PR(N) * w_{AN} + d * PR(Y) * w_{AY} + (1-d) * p_A$$

MAP

MAP

REDUCE

# Calcul de PageRank en M/R

## Map :

- Un Map task travaille sur une portion de la matrice  $M$  et du vecteur  $PR^{(k-1)}$  et produit une partie de  $PR^{(k)}$
- La fonction Map s'applique à un seul élément  $w_{ij}$  ( $1/|out(j)|$ ) de la matrice  $M$  et produit la paire  $(i, d * w_{ij} * PR_j)$  => tous les termes de la somme qui permet de calculer le nouveau  $PR_i$  auront la même clé
- On utilise également un combiner pour agréger localement les valeurs produites par le même Map task

=> Définir des **stratégies de partitionnement de la matrice et des vecteurs** qui tiennent compte de la capacité mémoire des nœuds de calcul exécutant les tasks

## Reduce :

- La fonction Reduce additionne les termes avec la même clé  $i$ , et produit la paire  $(i, PR_i)$

# [PageRank en MapReduce]

*method* **MAP** (nodeid n, vertex N)

EMIT(n, N)

**for** all nodeid m in N.OUT **do**

$p \leftarrow d * w_{nm} * N.PAGERANK$

EMIT(m, p)

*method* **REDUCE** (nodeid m, [p<sub>1</sub>, p<sub>2</sub>...])

M ← null, s ← 0

**for** all p in [p<sub>1</sub>, p<sub>2</sub>...] **do**

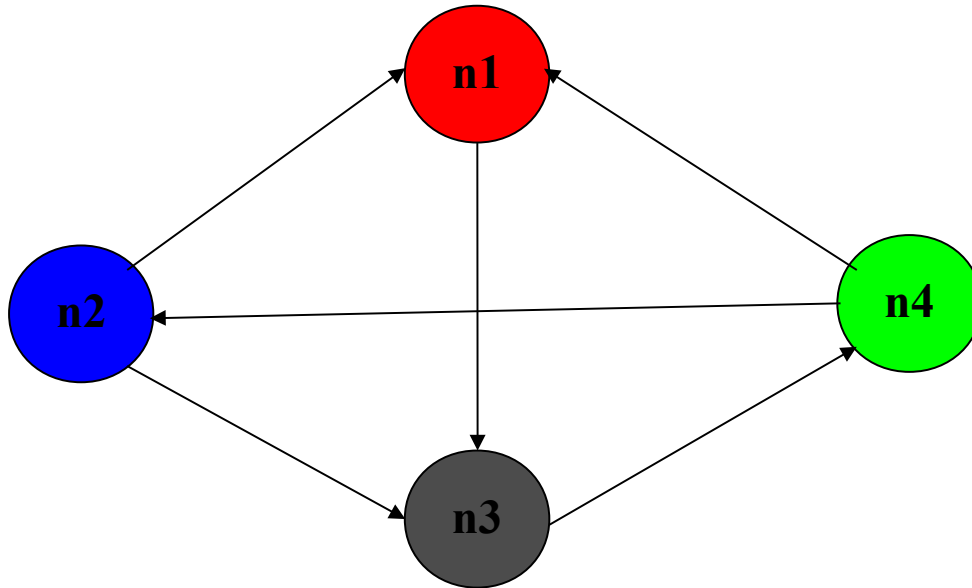
**if** ISVERTEX(p) **then** M ← p

**else** s ← s + p

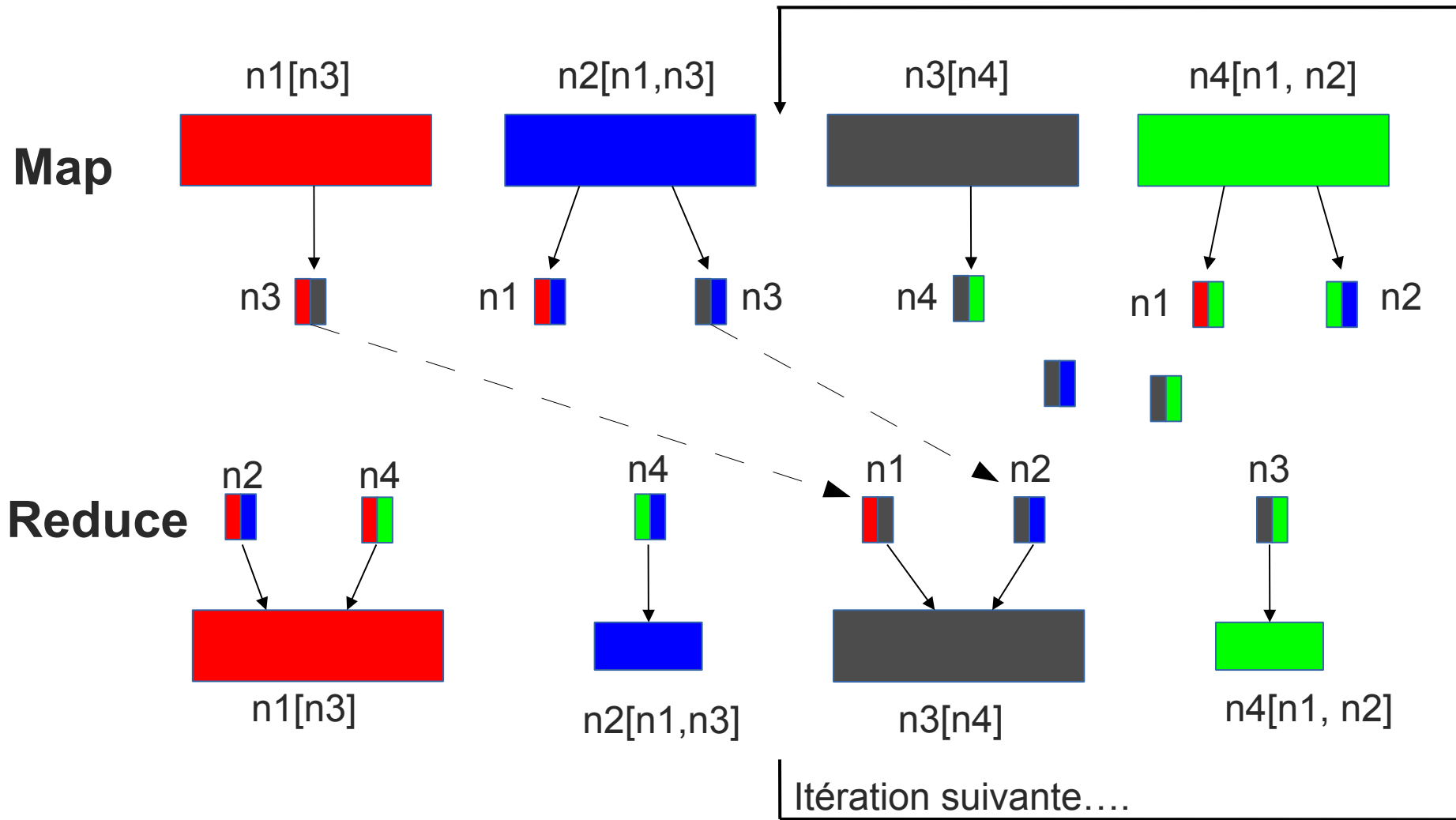
M.PAGERANK ← s+(1-d)\*M.P

EMIT(m, M)

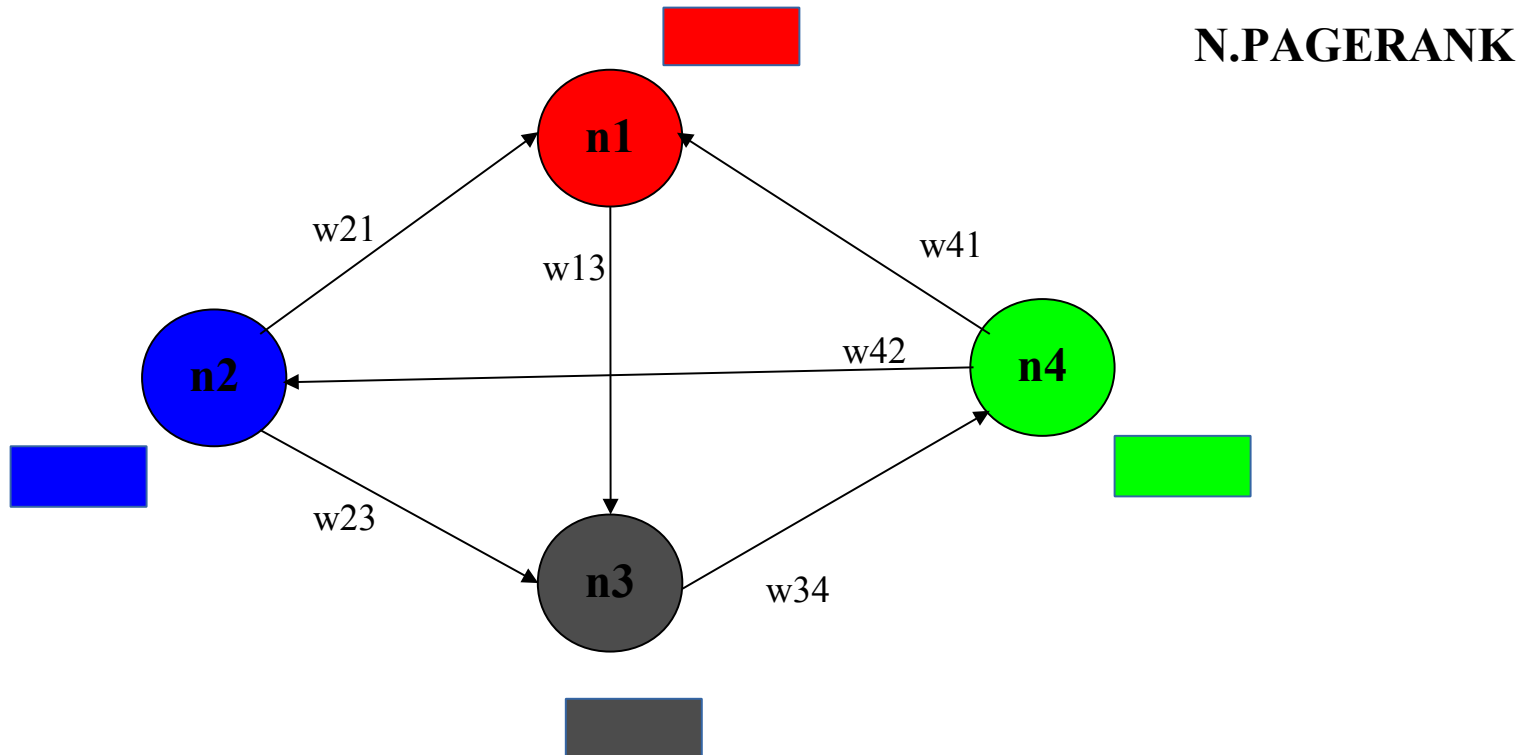
# Exemple



# [ PageRank en MapReduce ]

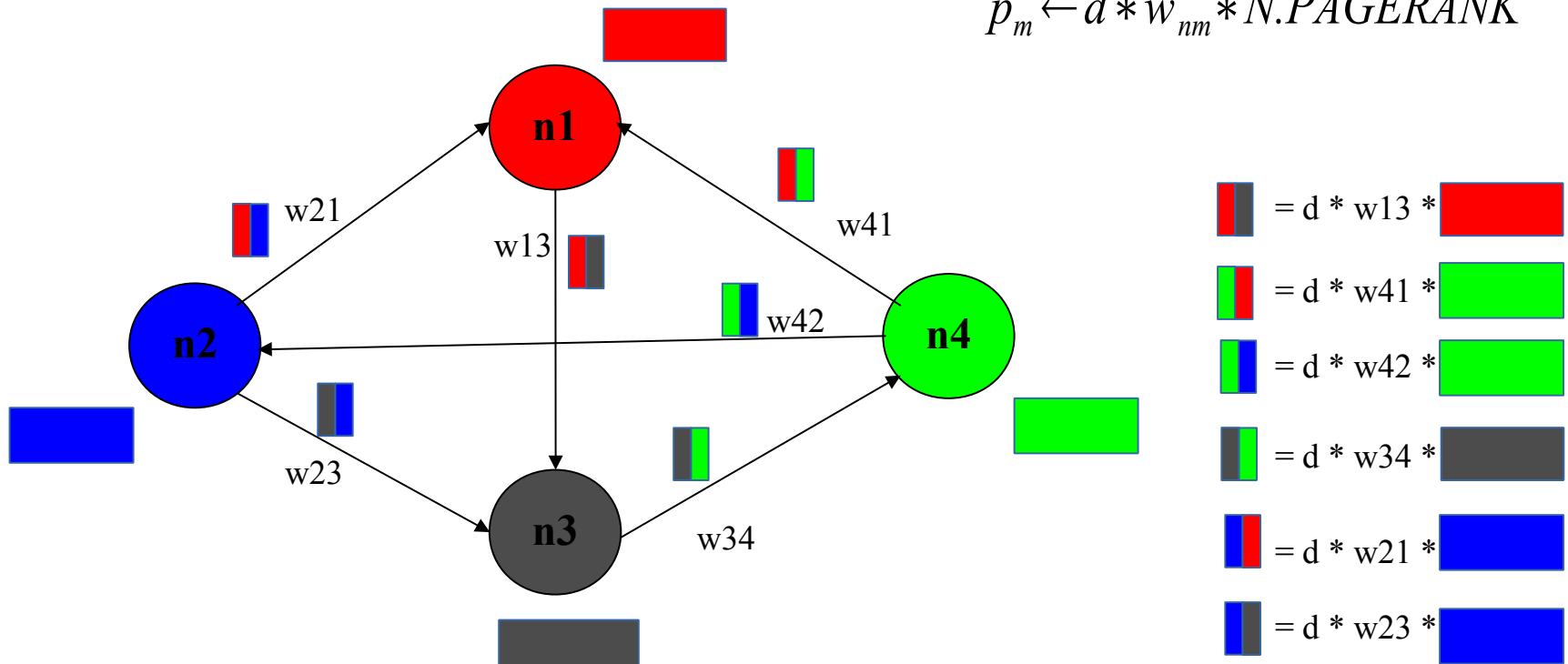


# Exemple: MAP



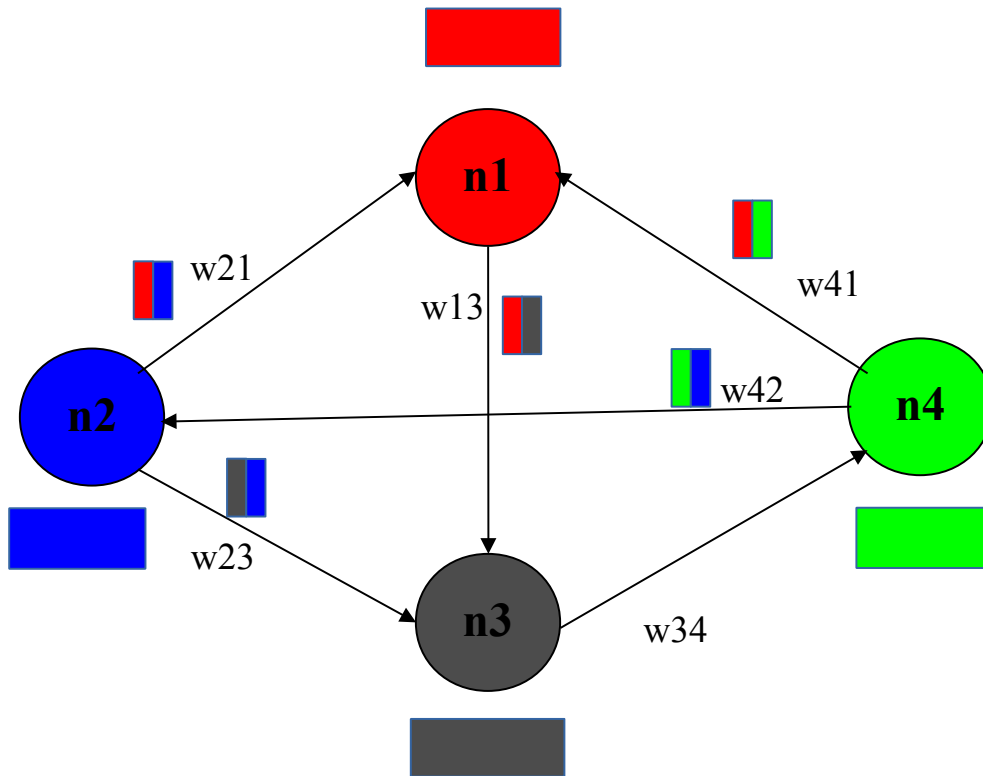
# Exemple: MAP

$$p_m \leftarrow d * w_{nm} * N.PAGERANK$$



# Exemple: REDUCE

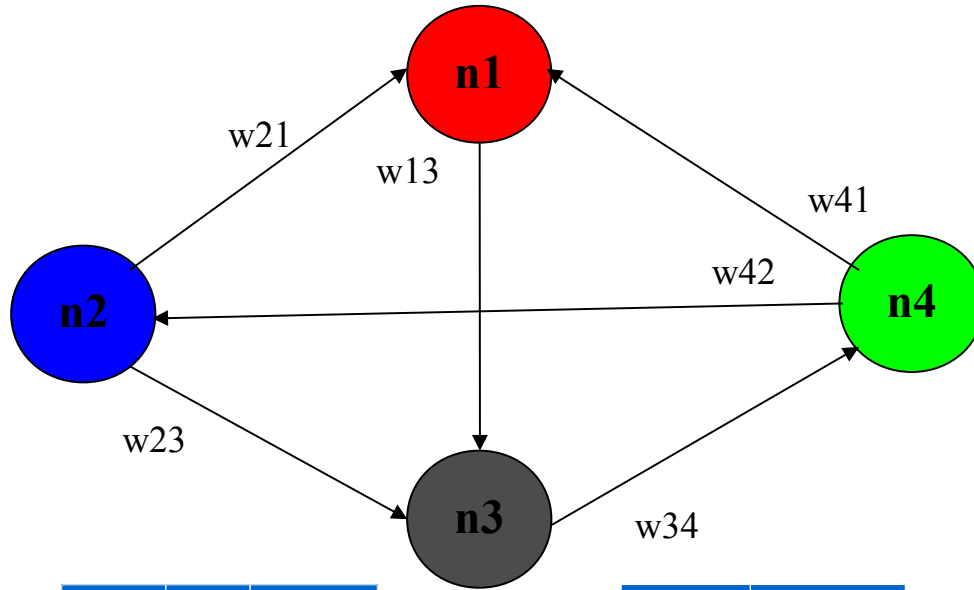
$$M.PAGERANK \leftarrow s + (1-d) * M.P$$



$$\begin{aligned} \text{Red Rectangle} &= ( \text{Red/Blue} + \text{Red/Green} ) + (1-d) * n1.PERS \\ \text{Blue Rectangle} &= ( \text{Green/Blue} ) + (1-d) * n2.PERS \\ \text{Green Rectangle} &= ( \text{Dark Grey/Green} ) + (1-d) * n4.PERS \\ \text{Dark Grey Rectangle} &= ( \text{Dark Grey/Blue} + \text{Red/Dark Grey} ) + (1-d) * n1.PERS \end{aligned}$$



# Exemple: PR personnalisé



$d = 0.85$

PR = Importance à calculer

P = Vecteur de personnalisation  
(personnalisation pour n2)

S	D	w
1	3	w13
2	1	w21
2	3	w23
3	4	w34
4	1	w41
4	2	w42

graphe

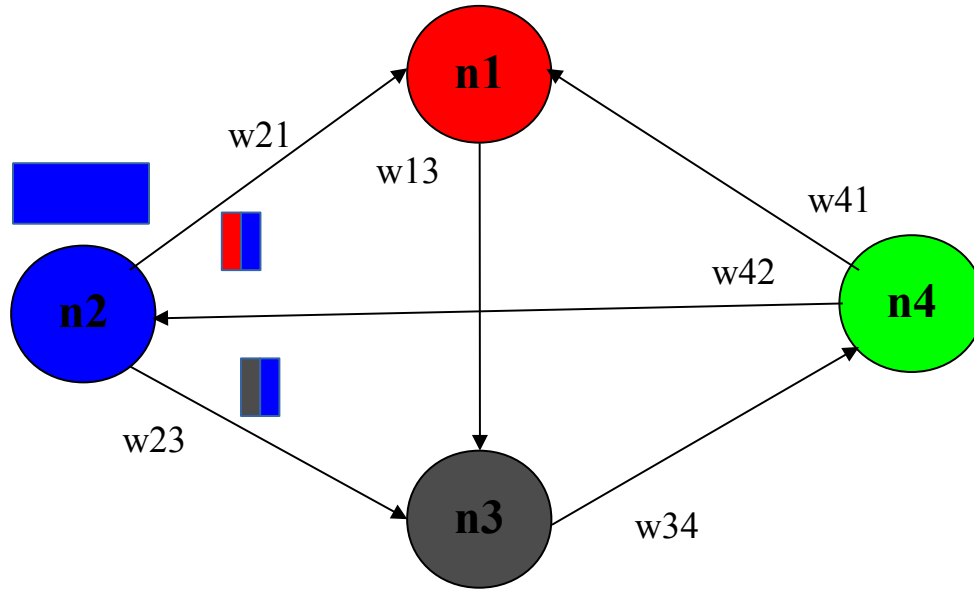
id	p
2	1

P

id	i
1	i1
2	i2
3	i3
4	i4

PR

# Exemple: Première itération



$$= i2\_0 * d * w21$$

$$= i2\_0 * d * w23$$

S	D	w
1	3	w13
2	1	w21
2	3	w23
3	4	w34
4	1	w41
4	2	w42

graphe

id	i
2	i2_0
= i2_0 = 1	

PR=P

Iteration 1

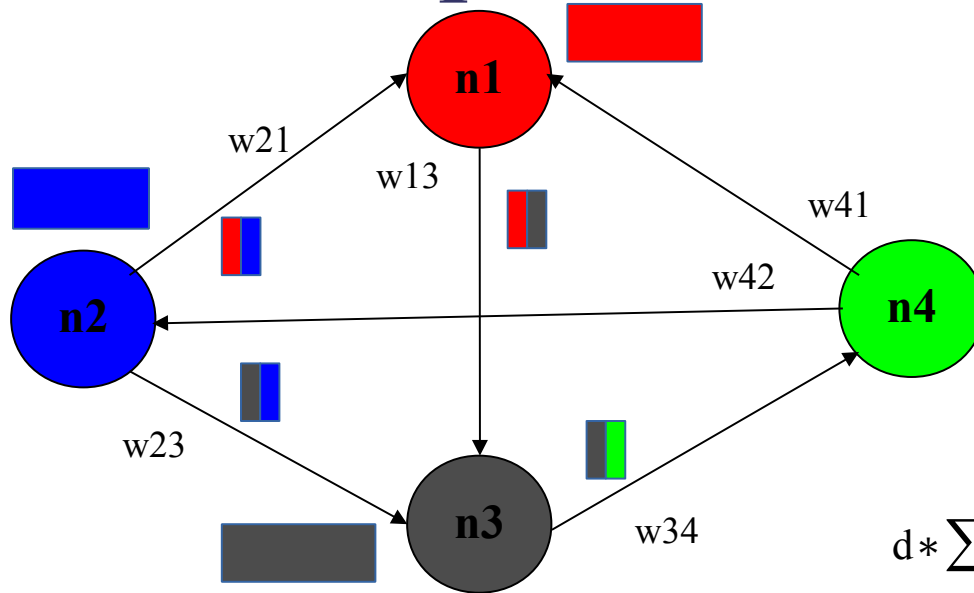
D	next-i
1	$i2\_0 * d * w21$
3	$i2\_0 * d * w23$





$$d * \sum i_j^0 * w_{ji}$$

id	i
1	$i2\_0 * d * w21$
3	$i2\_0 * d * w23$
2	$(1-d) * 1$

$$PR_i^1 = d * \sum i_j^0 * w_{ji} + (1-d) * P_i$$

# Exemple: deuxième itération





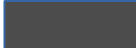
  $= i2\_1 * d * w21 = (1-d) * d * w21$   
  $= i2\_1 * d * w23 = (1-d) * d * w23$   
  $= i3\_1 * d * w34 = (1 * d * w23) * d * w34$   
  $= i1\_1 * d * w13 = (1 * d * w21) * d * w13$

$$d * \sum i'_j * w_{ji}$$

$$PR_i^2 = d * \sum i'_j * w_{ji} + (1-d) * P_i$$

S	D	w
1	3	w13
2	1	w21
2	3	w23
3	4	w34
4	1	w41
4	2	w42





graphe

id	i
1	i1_1
3	i3_1
2	i2_1
	i2_1
	i1_1
	i3_1

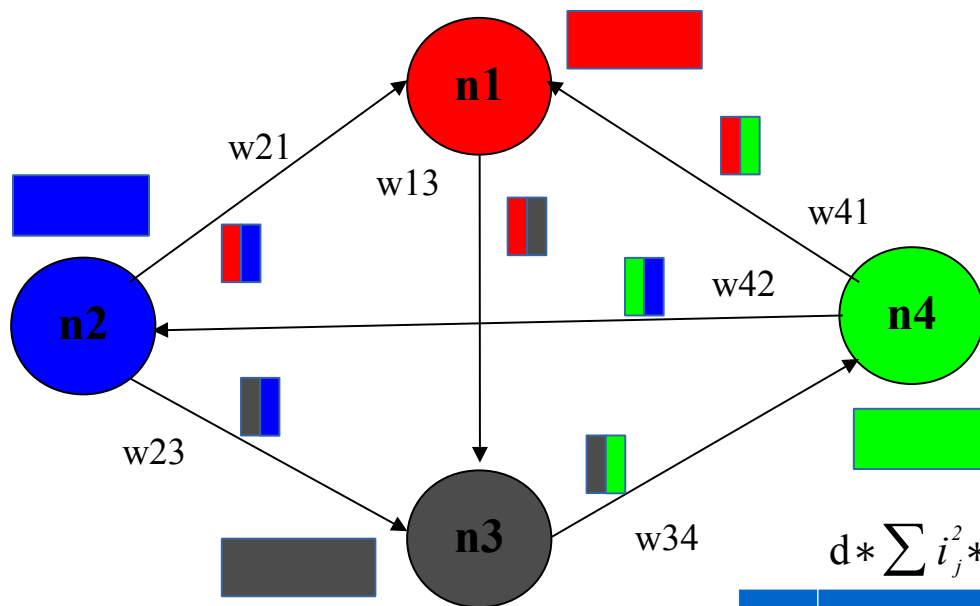
Itération 2  
→







D	next-i
1	$i2\_1 * d * w21$
3	$i2\_1 * d * w23 + i1\_1 * d * w13$
4	$i3\_1 * d * w34$

id	i
1	$i2\_1 * d * w21$
3	$i2\_1 * d * w23 + i1\_1 * d * w13$
4	$i3\_1 * d * w34$
2	$(1-d) * 1$

  $i2\_2 = 1-d$   
  $i1\_2 = i2\_1 * d * w21$   
  $i3\_2 = i2\_1 * d * w23 + i1\_1 * d * w13$   
  $i4\_2 = i3\_1 * d * w34$

# Exemple: troisième itération



-  =  $i2\_2 * d * w21$
-  =  $i2\_2 * d * w23$
-  =  $i3\_2 * d * w34$
-  =  $i1\_2 * d * w13$
-  =  $i4\_2 * d * w41$
-  =  $i4\_2 * d * w42$

$$d * \sum i_j^2 * w_{ji}$$

$$PR_i^3 = d * \sum i_j^2 * w_{ji} + (1-d) * P_i$$

S	D	w
1	3	w13
2	1	w21
2	3	w23
3	4	w34
4	1	w41
4	2	w42





graphe

id	i
1	i1_2
3	i3_2
2	i2_2
4	i4_2

Itération 2

D	next-i
1	$i2\_2 * d * w21 + i4\_2 * d * w41$
3	$i2\_2 * d * w23 + i1\_2 * d * w13$
4	$i3\_2 * d * w34$
2	$i4\_2 * d * w42$

id	i
1	$i2\_2 * d * w21 + i4\_2 * d * w41$
3	$i2\_2 * d * w23 + i1\_2 * d * w13$
4	$i3\_2 * d * w34$
2	$i4\_2 * d * w42 + (1-d) * 1$

-   $i2\_3 = i4\_2 * d * w42 + (1-d) * 1$
-   $i1\_3$
-   $i3\_3$
-   $i4\_3$

# Algorithme distribué: convergence

- $\epsilon$ : Seuil de convergence
- Convergence **locale** de l'importance du noeud ***nj*** à l'itération ***k***:
  - $|i_j^k - i_j^{k-1}| / i_j^{k-1} < \epsilon$
  - La condition pourrait ne plus être satisfaite à l'itération suivant
- Convergence **globale** de l'importance à l'itération ***k***:
  - Convergence locale pour tous les noeuds

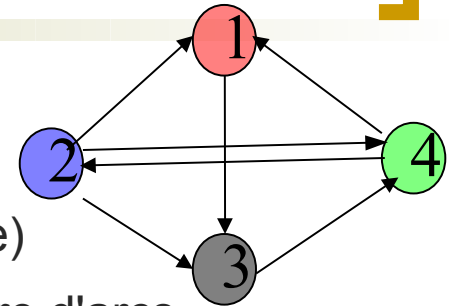
→ Arrêt de l'algorithme

- *Comment modifier le calcul M/R* pour prendre en compte la convergence?
- **Implémentation TME**: liste de noeuds encore ***actives*** à chaque itération ***k***:
  - Noeuds mis à jour à ***k*** sans convergence locale
  - Peut varier d'une itération à une autre
  - Arrêt de l'algorithme lorsque la liste ***actives*** est vide

# Calcul de PR par bandes/blocks

*Encodage de la matrice :*

- Matrice creuse (beaucoup d'entrées sont 0)
- Stocker les entrées non nulles (listes d'adjacence)
- L'espace de stockage est proportionnel au nombre d'arcs
- Pour chaque nœud on stocke également le nombre de liens sortants
- Le vecteur PR est également de grandes dimensions (prop. au nombre de noeuds)



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



Nœud source	degré	Nœuds destination
1	1	3
2	3	1, 3, 4
3	1	4
4	2	1, 2

# [ Partitionnement en bandes ]

- B Map tasks

## Map

- La matrice  $M$  est partitionnée en  $B$  bandes verticales (une bande correspond à un ensemble de nœuds source)
- Le vecteur  $PR^{(k-1)}$  est partitionné en  $B$  bandes horizontales  
 $\Rightarrow$  chaque task reçoit une bande  $M_b$  de  $M$  et la bande correspondante de  $PR_b^{(k-1)}$
- Chaque task produit une version locale du vecteur  $PR^k$  (de même taille que le vecteur  $PR^k$ ) :  $PR_b^k = ((1, PR_b^k(1)), \dots, (n, PR_b^k(n)))$

## Reduce

- Aggrégation somme des vecteurs  $PR_b^k$  en fonction des clés

**Avantage de cette méthode :** on stocke seulement une partie(bande) de  $M$  et de  $PR^{(k-1)}$  dans la mémoire locale d'un nœud de calcul

**Inconvénient :** on doit stocker le vecteur  $PR_b^k$  entièrement( peut avoir la même taille que  $PR^k$ )  $\Rightarrow$  problème si pas assez de mémoire

# [Exemple]

Map task 1

source	degré	destination
1	1	3
2	3	1, 3, 4

$M_1$

source	degré	destination
3	1	4
4	2	1, 2

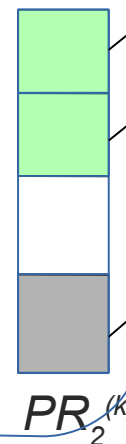
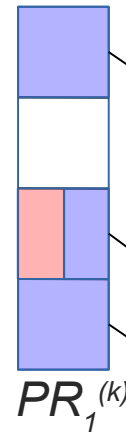
$M_2$

$$* \begin{array}{|c|} \hline \text{red} \\ \hline \text{blue} \\ \hline \end{array} = \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \text{ + blue} \\ \hline \text{blue} \\ \hline \end{array}$$

$PR_1^{(k-1)}$

$$* \begin{array}{|c|} \hline \text{grey} \\ \hline \text{green} \\ \hline \end{array} = \begin{array}{|c|} \hline \text{green} \\ \hline \text{green} \\ \hline \text{white} \\ \hline \text{grey} \\ \hline \end{array}$$

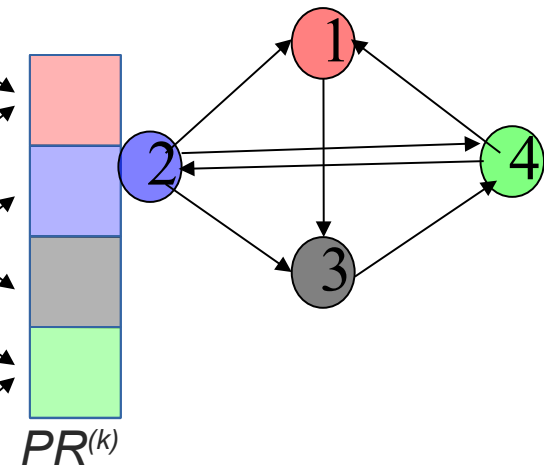
$PR_2^{(k-1)}$



0	1/3	0	1/2
0	0	0	1/2
1	1/3	0	0
0	1/3	1	0

$M_1$        $M_2$

$$* \begin{array}{|c|} \hline \text{blue} \\ \hline \text{blue} \\ \hline \text{white} \\ \hline \text{white} \\ \hline \end{array} \begin{array}{l} PR_1^{(k-1)} \\ PR_2^{(k-1)} \end{array}$$



Reduce

Map task 2



# Partitionnement en blocks

## B\*B Map tasks

### Map

$M_{11}$	....	$M_{1B}$
$\vdots$	$\vdots$	$\vdots$
$M_{B1}$	....	$M_{BB}$

- La matrice M est partitionnée en **B\*B blocks**
- Le vecteur  $PR^{(k-1)}$  est partitionné en **B bandes horizontales**

=> chaque task reçoit un block  $M_{ib}$  de M et une bande de  $PR_b^{(k-1)}$  ( $PR_b^{(k-1)}$  est transmise B fois : à chaque task qui reçoit un block  $M_{ib}$ , pour i de 1 à B)

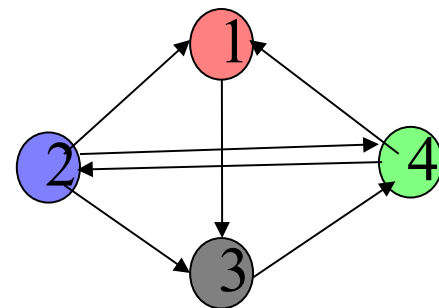
- Chaque task produit une version locale du vecteur  $PR_i^k$  :
- $PR_{ib}^k = ((1, PR_{ib}^k(1)), \dots, (i, PR_{ib}^k(i)))$

**Reduce :** somme des vecteurs  $PR_{ib}^k$  en fonction des clés

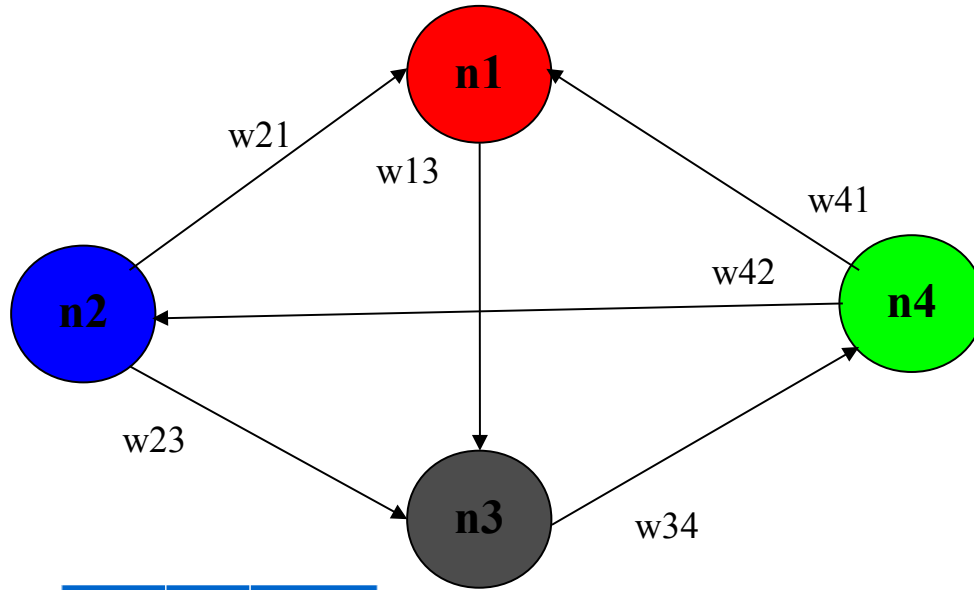
- **Avantage de cette méthode :** on stocke seulement une partie(block) de M et de  $PR^{(k-1)}$  dans la mémoire locale d'un nœud de calcul, ainsi qu'une partie du vecteur final => tout peut être stocké en mémoire

**Inconvénient :** chaque bande  $PR_b^{(k-1)}$  du vecteur  $PR^{(k-1)}$  doit être répliquée plusieurs fois

## ]



# Calcul de PR personnalisé en bandes?



$d = 0.85$

PR = Importance à calculer

P = Vecteur de personnalisation  
(personnalisation pour n2)

S	D	w
1	3	w13
2	1	w21
2	3	w23
3	4	w34
4	1	w41
4	2	w42

graphe

id	p
2	1
1	0
3	0
4	0

P

id	i
1	i1
2	i2
3	i3
4	i4

PR

	n1	n2	n3	n4
n1		$w_{21}$		$w_{41}$
n2				$w_{42}$
n3	$w_{13}$	$w_{23}$		
n4			$w_{34}$	

M

# [ Calcul M/R : Problèmes ]

- Le graphe est transmis (shuffled) à chaque itération (objet **vertex N** transmis comme paramètre aux deux méthodes, il inclut la liste d'adjacence OUT)
- On veut pouvoir transmettre uniquement la nouvelle valeur d'importance et non pas la structure du graphe
- On doit contrôler les itérations en dehors de M/R (conditions de terminaison et logique du programme)

=> **Pregel** (calcul sur des graphes de grande taille)

- Proposé par Google
- Implémentations open source : Apache Giraph, Stanford GPS, Apache Hama

[

]

Compter les triangles

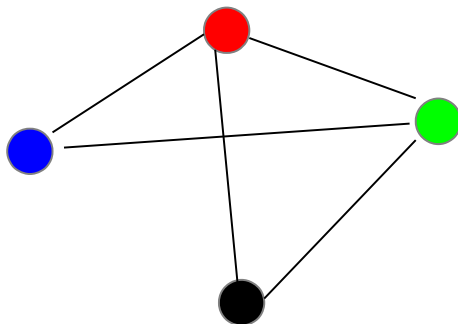
# Pourquoi compter les triangles ?

*Coefficient de clustering* pour un graphe non dirigé  $G=(V,E)$  :

- mesure de regroupement des nœuds dans un graphe, utilisé dans les réseaux petit monde (réseaux sociaux), détection de communautés
- $cc(v)$  = fraction des voisins de  $v$  qui sont eux-mêmes des voisins

$$= \frac{|\{(u, w) \in E | u \in \Gamma(v) \wedge w \in \Gamma(v)\}|}{\binom{d_v}{2}}$$

$\binom{d_v}{2}$  est le nombre total d'arcs possibles entre les voisins de  $v$



$$cc(\text{blue}) = 1/1$$

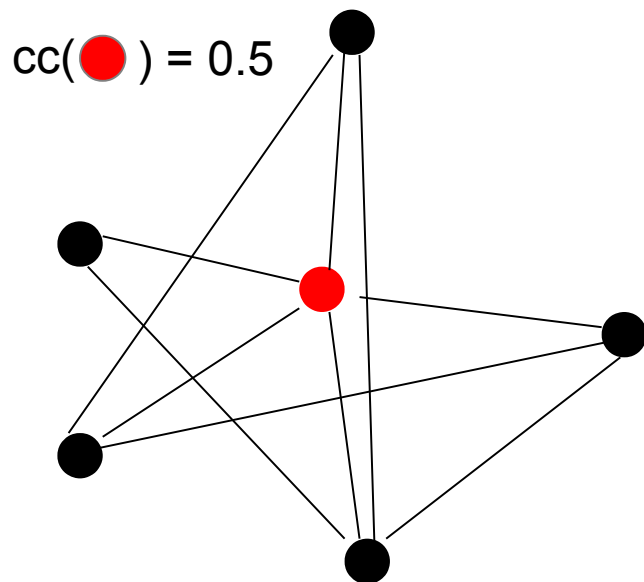
$$cc(\text{red}) = 2/3$$

$$cc(\text{black}) = 1/1$$

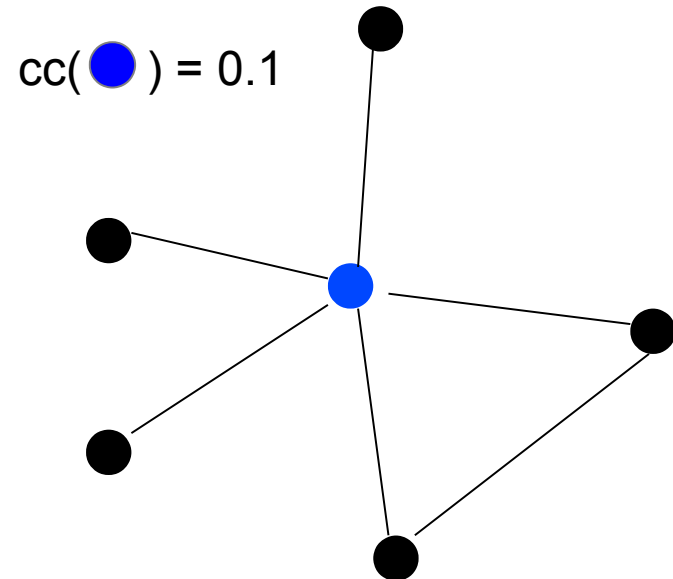
$$cc(\text{green}) = 2/3$$

# [ Coefficient de clustérisation ]

Montre la densité de la connectivité autour d'un noeud



vs.



# [ Comment compter les triangles ? ]

*Algorithme séquentiel* (graphe non dirigé) :

NbTriangles  $\leftarrow 0$

foreach nœud  $v$  :

    foreach couple  $u, w$  in  $\Gamma(v)$

        if  $(u, w)$  est une arête

            NbTriangles  $+= 1/2$

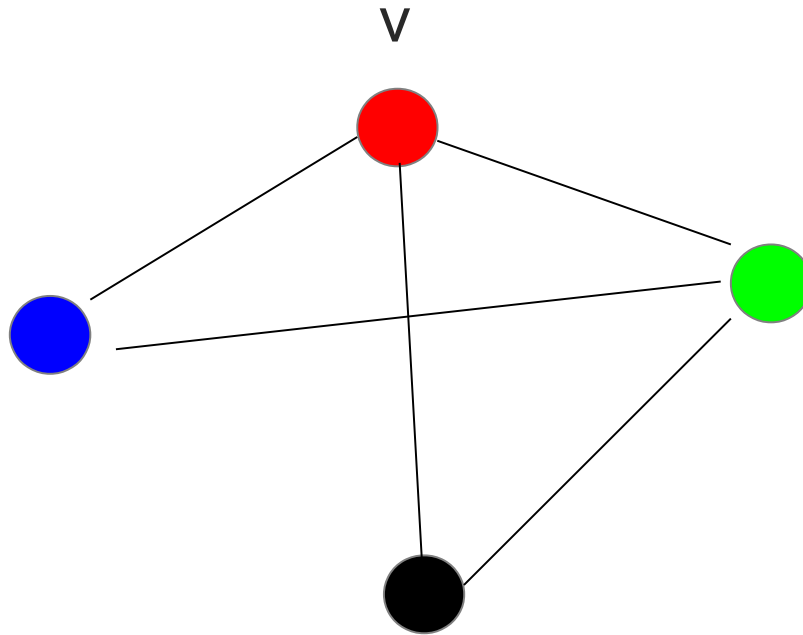
return (NbTriangles / 3)

Complexité de l'algorithme :  $O\left(\sum d_v^2\right)$

Chaque triangle est compté 3 fois (une fois par noeud)



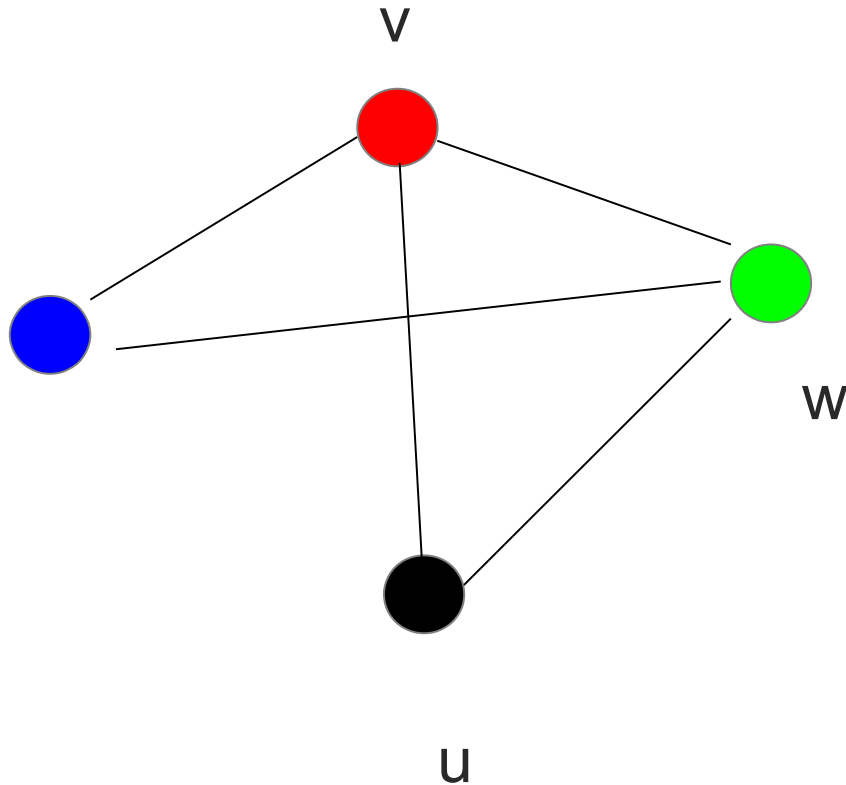
# [ Exemple : calcul des triangles ]



À partir du nœud v

NbTriangles = 0

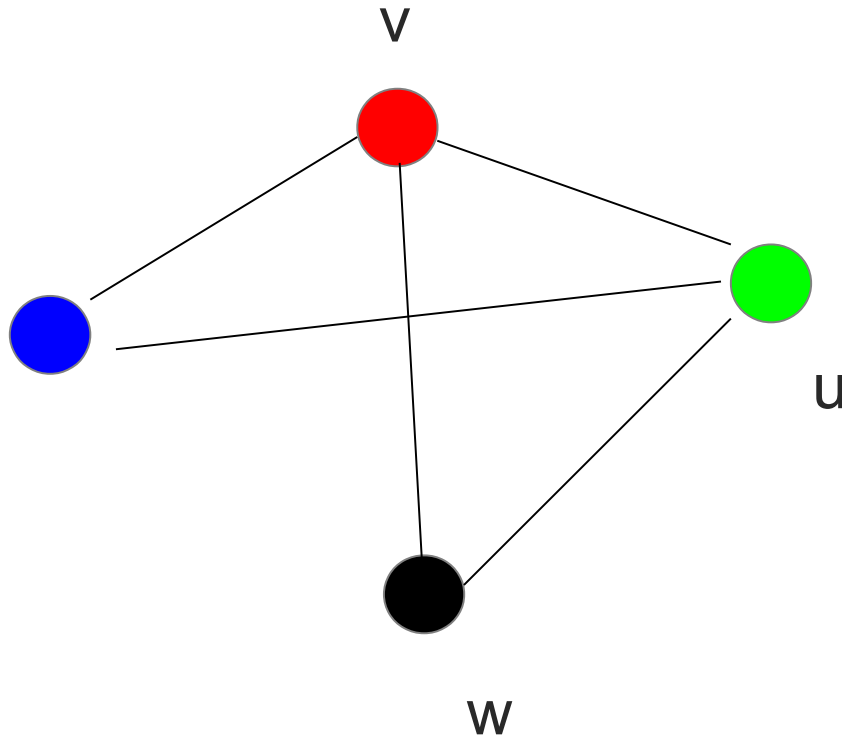
# [Exemple]



À partir du nœud  $v$

NbTriangles =  $1/2$

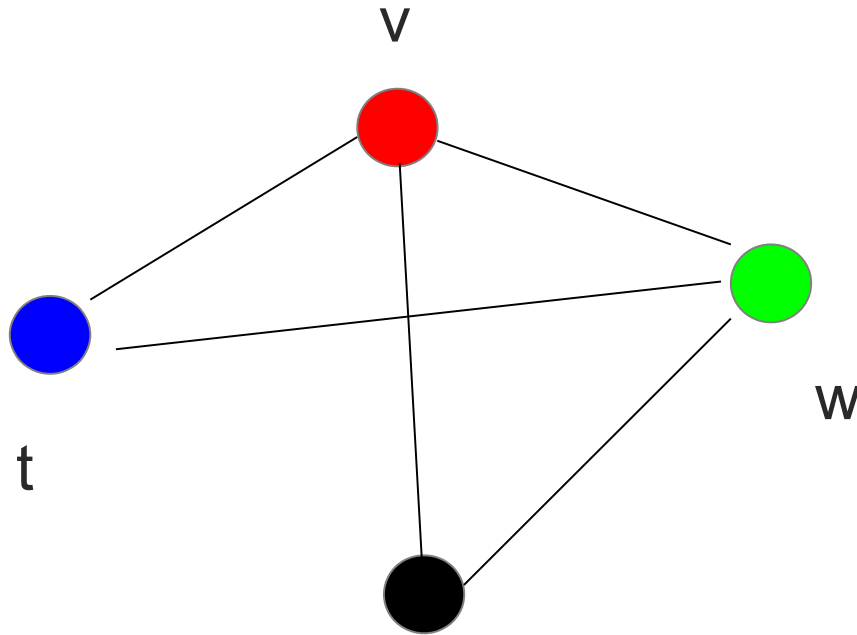
# [Exemple]



À partir du nœud v

NbTriangles = 1

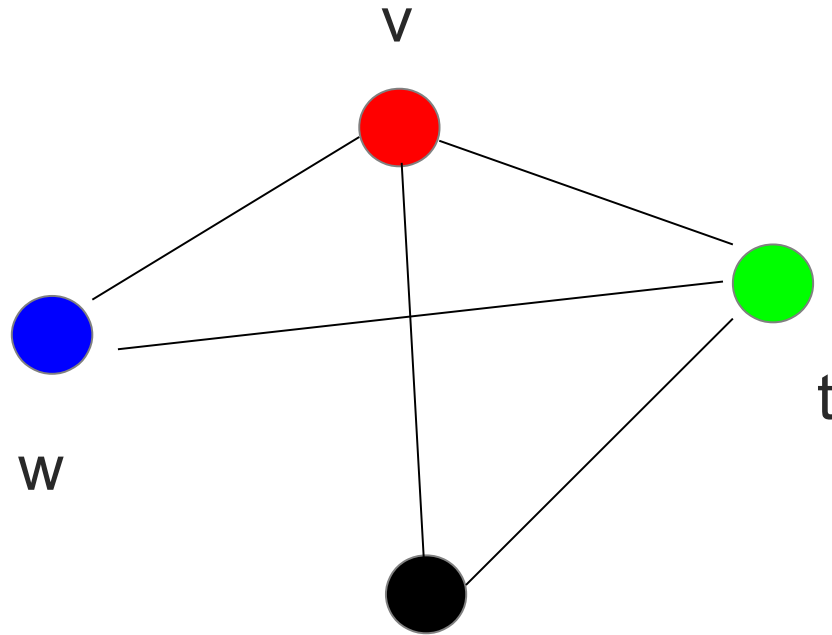
# [Exemple]



À partir du nœud  $v$

NbTriangles = 1.5

# [Exemple]



À partir du nœud v

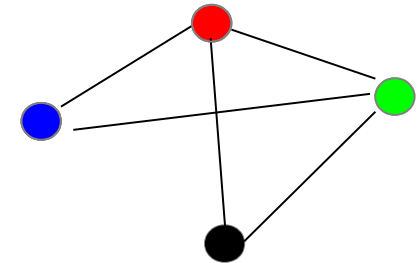
NbTriangles = 2

# [ Algorithme MapReduce ]

**Map 1** : Input :  $\{(v,u) \mid u \in \Gamma(v)\}$

foreach  $u \in \Gamma(v)$  emit  $\{(v,u)\}$

Exemple : (●●)(●●)(●●)(●●)(●●)(●●)(●●)



**Reduce 1** : Input :  $\{(v,u) \mid u \in \Gamma(v)\}$

foreach  $(u,w) : u,w \in \Gamma(v)$

emit  $\{((u,w), v)\}$  //tous les arcs possibles dans le graphe

Exemple : ((●●)●) ((●●)●) ((●●)●),.....

**Map 2** : emit  $\{((u,w), \$) \mid w \in \Gamma(u)\}$

**Reduce 2** : Input :  $\{(u,w) \mid v_1, v_2, \dots, v_k, \$?\}$

foreach  $(u,w)$  if  $\$$  is part of the input, then :

NbTriangles[ $v_i$ ] += 1/2

((●●)●)  $\rightarrow$  NbTriangles(●) + 1/2

(●●)  $\rightarrow \emptyset$

# [Adaptation de l'algorithme]

- On génère tous les chemins à vérifier en parallèle, le temps d'exécution est  $\max_{v \in V} (\sum d_v^2) \Rightarrow$  pour les nœuds avec beaucoup de voisins (millions) les reducer tasks correspondants peuvent être très lents

*Amélioration :*

- ordonner les nœuds par leur degré (pour ceux qui ont le même degré par leur identifiant)
- compter chaque triangle une seule fois, à partir du nœud minimum
- complexité :  $O(m^{3/2})$  (m = nombre d'arcs dans le graphe)

# [ Algorithme amélioré ]

Algorithme séquentiel :

NbTriangles  $\leftarrow$  0

foreach  $v$  in  $V$

    foreach  $u, w$  in Adjacency( $v$ )

        if  $u > v \ \&\& \ w > u$

            if  $(u, w)$  in  $E$

                Triangles++

Return Triangles



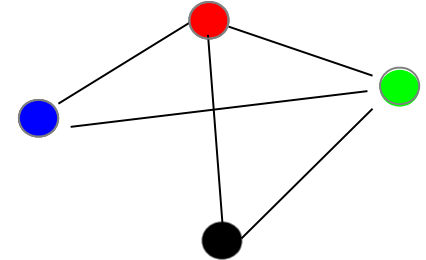


# [ Algorithme parallèle ]

**Map 1** : Input :  $\{(v,u) \mid u \in \Gamma(v)\}$

if  $u > v$  then emit  $\{(v,u)\}$

Exemple : (● ●) (● ●) (● ●) (● ●) (● ●)



**Reduce 1** : Input :  $\{(v,u) \mid u \in S \subset \Gamma(v)\}$

foreach  $(u,w) : u,w \in S$  //  $u,w$ :voisins de  $v$  ( $v < u$  et  $v < w$ )

if  $w > u$  then emit  $\{((u,w), v)\}$

Exemple : ((● ●)●) ((● ●)●) ((● ●)●)

● < ● < ● < ●

**Map 2** : emit  $\{((u,w), \$) \mid w \in \Gamma(u), u < w\}$

**Reduce 2** : Input :  $\{(u,w) \mid v_1, v_2, \dots, v_k, \$\}$

foreach  $(u,w)$  if  $\$$  is part of the input, then :  $\text{NbTriangles}[v_i]++$

Exemple : ((● ●)●)  $\rightarrow \text{NbTriangles}(\text{●})++$

(● ●)●  $\rightarrow \emptyset$

# Calcul DataFrame

**Map 1** : Input :  $\{(v,u) \mid u \in \Gamma(v)\}$

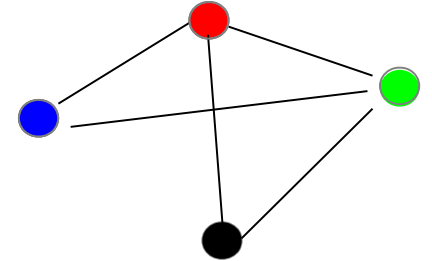
if  $u > v$  then emit  $\{(v,u)\}$

Exemple : (● ●) (● ●) (● ●) (● ●) (● ●)

v	u
●	●
●	●
●	●
●	●
●	●
●	●
●	●
●	●
●	●
●	●



v	u
●	●
●	●
●	●
●	●
●	●
●	●



● < ● < ● < ●

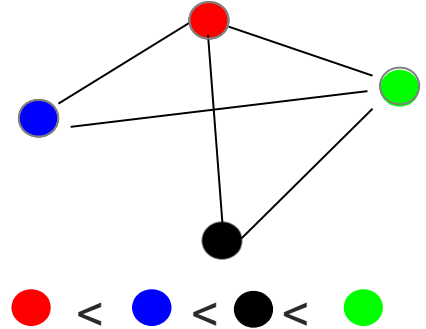
# Calcul DataFrame

**Reduce 1** : Input :  $\{(v,u) \mid u \in S \subset \Gamma(v)\}$

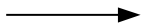
foreach  $(u,w) : u,w \in S // u,w$ :voisins de  $v$  ( $v < u$  et  $v < w$ )

if  $w > u$  then emit  $\{((u,w), v)\}$

Exemple :  $((\text{blue}, \text{black}), \text{red}) ((\text{black}, \text{green}), \text{red}) ((\text{blue}, \text{green}), \text{red})$



v	u
red	blue
red	black
red	green
blue	green
black	green



v	liste_u
red	blue black green
blue	green
black	green



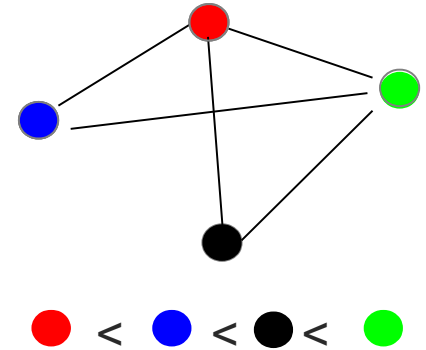
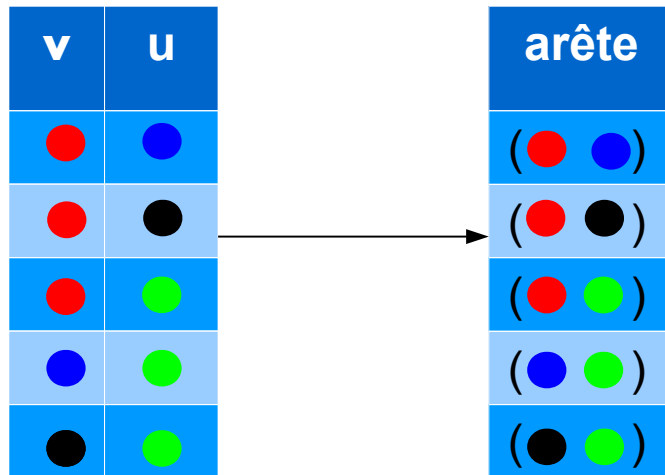
v	liste_u
red	(blue black)(blue green)(black green)



v	couple
red	(blue black)
red	(blue green)
red	(black green)

# [ Calcul DataFrame ]

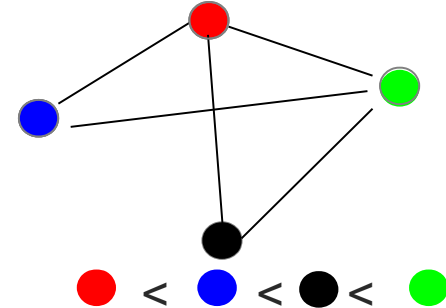
**Map 2 :** emit  $\{ ((u,w), \$) \mid w \in \Gamma(u), u < w \}$



# [ Calcul DataFrame ]

**Reduce 2** : Input :  $\{(u,w) \mid v_1, v_2, \dots, v_k, \$\}$

foreach (u,w) if \$ is part of the input, then :  $\text{NbTriangles}[v_i] ++$



arête
(red blue)
(red black)
(red green)
(blue green)
(black green)

arête=couple

v	couple
red	(blue black)
red	(blue green)
red	(black green)

v	couple
red	(blue green)
red	(black green)

v	triangles
red	2

[

]

Plus court chemin

# [ Problème ]

- Trouver la longueur du plus court chemin à partir d'un nœud donné  $s$  vers les autres nœuds
- *Centralisé*: l'algorithme de Dijkstra
- *MapReduce* : BFS en parallèle à partir du nœud initial  $s$

Intuition (pour un graphe non pondéré):

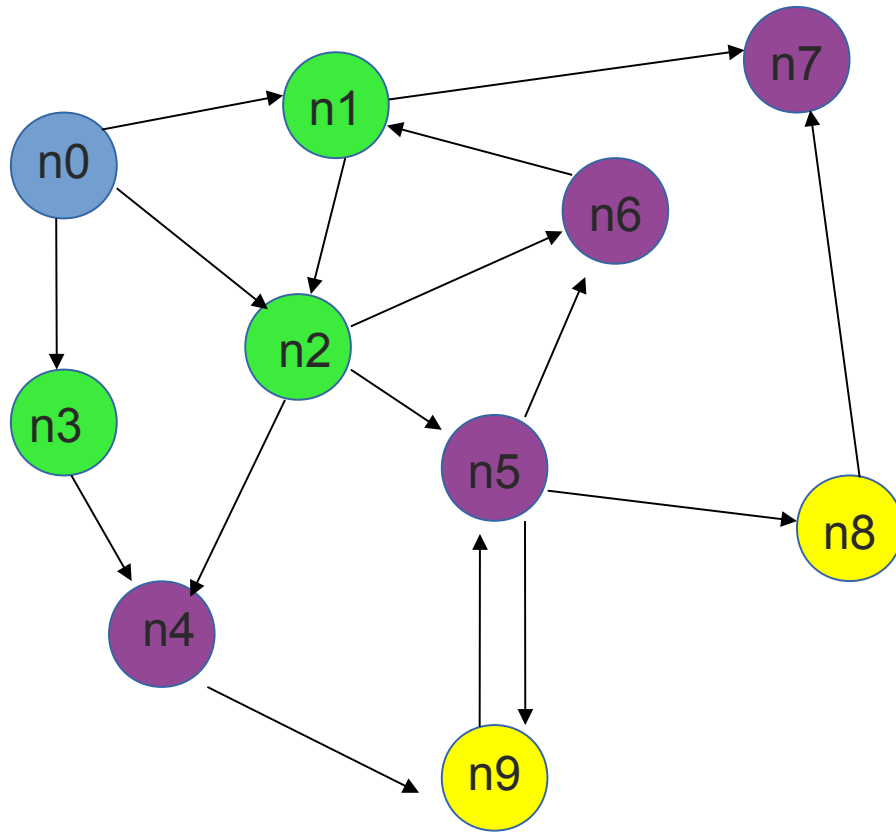
- Pour tous les voisins  $p$  de  $s$  :

$$\text{DISTANCE}(p) = 1$$

- Pour tous les nœuds  $n$  atteignables à partir d'un ensemble de nœuds  $M$

$$\text{DISTANCE}(n) = 1 + \min_{m \in M} (\text{DISTANCE}(m))$$

# [ BFS ]





# [Algorithme]

Données :

- Clé : nœud  $n$
- Valeur : distance  $d$  à partir de  $s$ , liste de voisins AdjacencyList  $\Rightarrow$  node  $N$
- Initialisation :  $d = \infty$  pour tous les nœuds sauf  $s$

```
1: class MAPPER
2:   method MAP(nid  $n$ , node  $N$ )
3:      $d \leftarrow N.DISTANCE$ 
4:     EMIT(nid  $n$ ,  $N$ )                                ▷ Pass along graph structure
5:     for all nodeid  $m \in N.ADJACENCYLIST$  do
6:       EMIT(nid  $m$ ,  $d + 1$ )                            ▷ Emit distances to reachable nodes

1: class REDUCER
2:   method REDUCE(nid  $m$ , [ $d_1, d_2, \dots$ ])
3:      $d_{min} \leftarrow \infty$ 
4:      $M \leftarrow \emptyset$ 
5:     for all  $d \in \text{counts } [d_1, d_2, \dots]$  do
6:       if ISNODE( $d$ ) then
7:          $M \leftarrow d$                                 ▷ Recover graph structure
8:       else if  $d < d_{min}$  then                          ▷ Look for shorter distance
9:          $d_{min} \leftarrow d$ 
10:     $M.DISTANCE \leftarrow d_{min}$                         ▷ Update shortest distance
11:    EMIT(nid  $m$ , node  $M$ )
```

# [ BFS ]

- Plusieurs itérations sont nécessaires pour explorer tout le graphe( à chaque itération on découvre de nouveaux nœuds)
- Lorsqu'un nœud est trouvé pour la première fois on obtient la plus courte distance (pas vrai pour des graphes pondérés où les arcs peuvent avoir des poids différents)
- *Arrêt de l'algorithme* : lorsque la distance de tous les nœuds est différente de  $\infty$  (on suppose que le graphe est connecté). Nombre d'itérations : le diamètre du graphe
- *Comparaison:*
  - BFS : À chaque itération calcule les distances vers tous les nœuds => beaucoup de calculs inutiles, pas de structure globale
  - Dijkstra : plus efficace, à chaque fois qu'un nœud est exploré son plus court chemin a déjà été trouvé, nécessite une structure globale

# [ 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](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/>)