

# Map-Reduce : un cadre de programmation parallèle pour l'analyse de grandes données

Stéphane Genaud  
ENSIIE

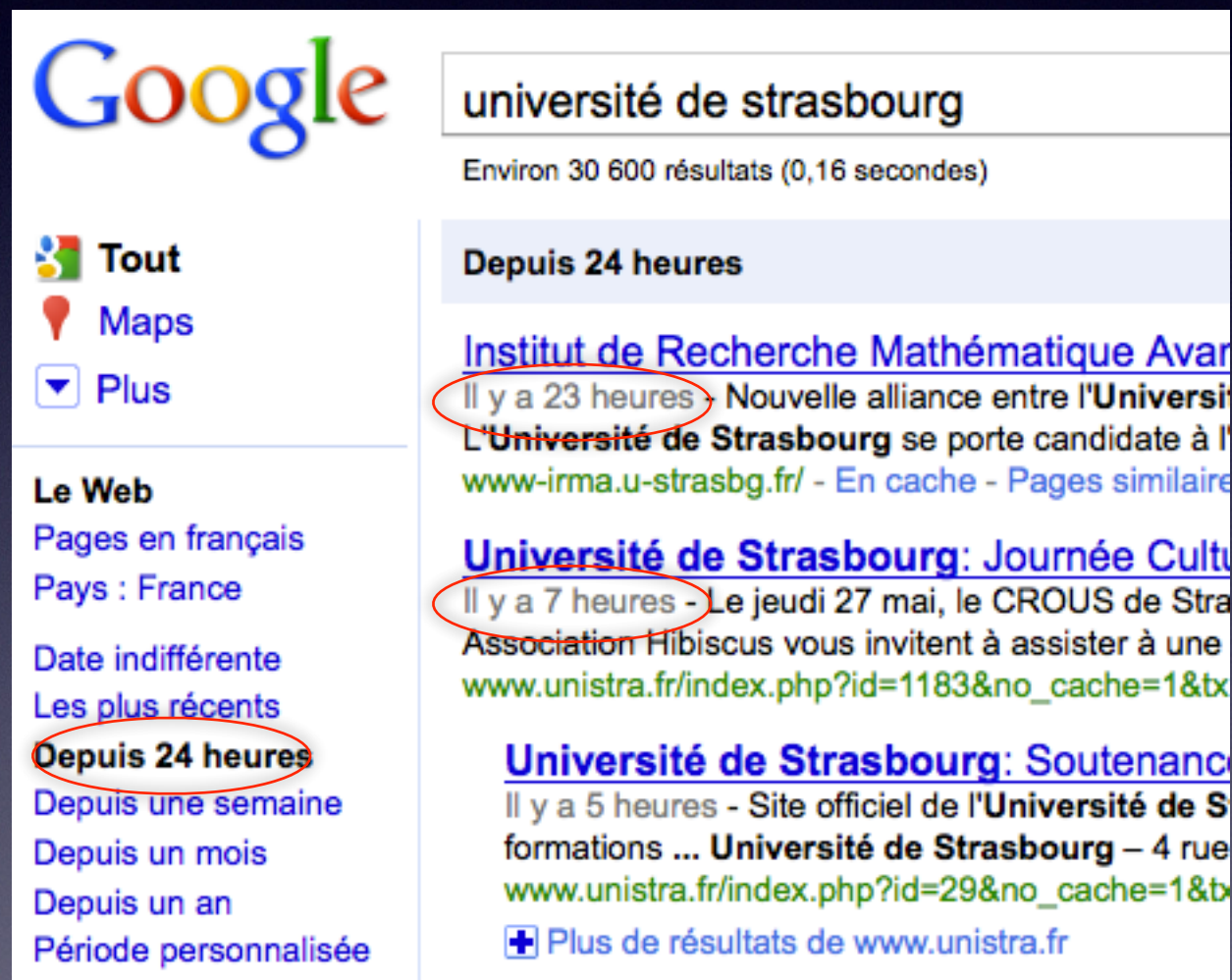


# Traitement de données distribuées

- ✦ Google a introduit **Map-Reduce** [Dean and Ghemawat 2004]
- ✦ Ils s'en servent de manière intensive:
  - utilisation en sept 2007: 400+ TB de données traitées, temps et nombre moyen de machine par traitement Map-Reduce : 6 min et 436
  - indexation web traite 20+ TB de données brutes
  - analyse d'images satellites
  - calculs statistiques pour Google translate
  - clustering des Google news
  - ....



# Traitement de données



- analyser les données toujours plus vite (avant 2007, 2 jours d'ancienneté)
- ... alors que le volume des données augmente toujours

27 mai 2010, 14:00



# Traitement de données

supelec

Web Images Maps Shopping Discussions Plus Outils de recherche

Tous les pays Toutes les langues Moins de 24 heures Tri par pertinence Tous les résultats

Les cookies assurent le bon fonctionnement de nos services. [OK](#) [En savoir plus](#)

**Supélec**  
[www.asso-supelec.org/gene/main.php](http://www.asso-supelec.org/gene/main.php)  
Il y a 20 heures - Bienvenue sur le site de [supelec.org](http://supelec.org). ... 24/01 : [Supélec Lyon] Vis Valbonne.

**Ingénieur ESE Supélec - Etnoka**  
[www.etnoka.fr/formations/one-formation.tcl?training\\_id=384](http://www.etnoka.fr/formations/one-formation.tcl?training_id=384)  
Il y a 22 heures - Ingénieur ESE **Supélec**. Spécialisation : radio communications, intelligence artificielle... Diplôme préparé : Diplôme Ingénieur. Matière principale : Electronique/ ...

**interview supelec sport h264 1920x1080 - YouTube**  
[www.youtube.com/watch?v=YpDuQPt1NyY](http://www.youtube.com/watch?v=YpDuQPt1NyY)  
Il y a 6 heures - Ajouté par voilesdeseinetv  
interview **supelec** sport h264 1920x1080. voilesdeseinetv-79  
videos. SubscribeSubscribedUnsubscribe 2 ...

Date indifférente  
Moins d'une heure  
✓ Moins de 24 heures  
Moins d'une semaine  
Moins d'un mois  
Moins d'un an  
Période personnalisée

16 jan 2013, 15:46



# Traitement de données distribuées

## ♦ Autres applications possibles:

- New York Times : 4TB d'images TIFF transformées en 11 millions de PDFs (durée 24h, coût 240 \$ sur 100 machines EC2)
- caractériser les activités des utilisateurs Facebook
- analyser les fichiers de logs d'un serveur web
- fouille de données sur logs des caisses enregistreuses super-marché
- recherche de séquences d'ADN dans un génôme



# Map-Reduce

- Un modèle de programmation ...
- ... avec un schéma très contraint.
- Mais permet
  - ✦ parallélisation automatique
  - ✦ de l'équilibrage de charge
  - ✦ des optimisations sur les transferts disques et réseaux
  - ✦ de la tolérance aux pannes



# Schéma général

1. Lecture des données
2. **Map** : pour chaque élément de données, construire un couple (clé,valeur)
3. Trier selon les clés
4. **Reduce**: agréger, résumer, filtrer ou transformer les données
5. Écrire les données



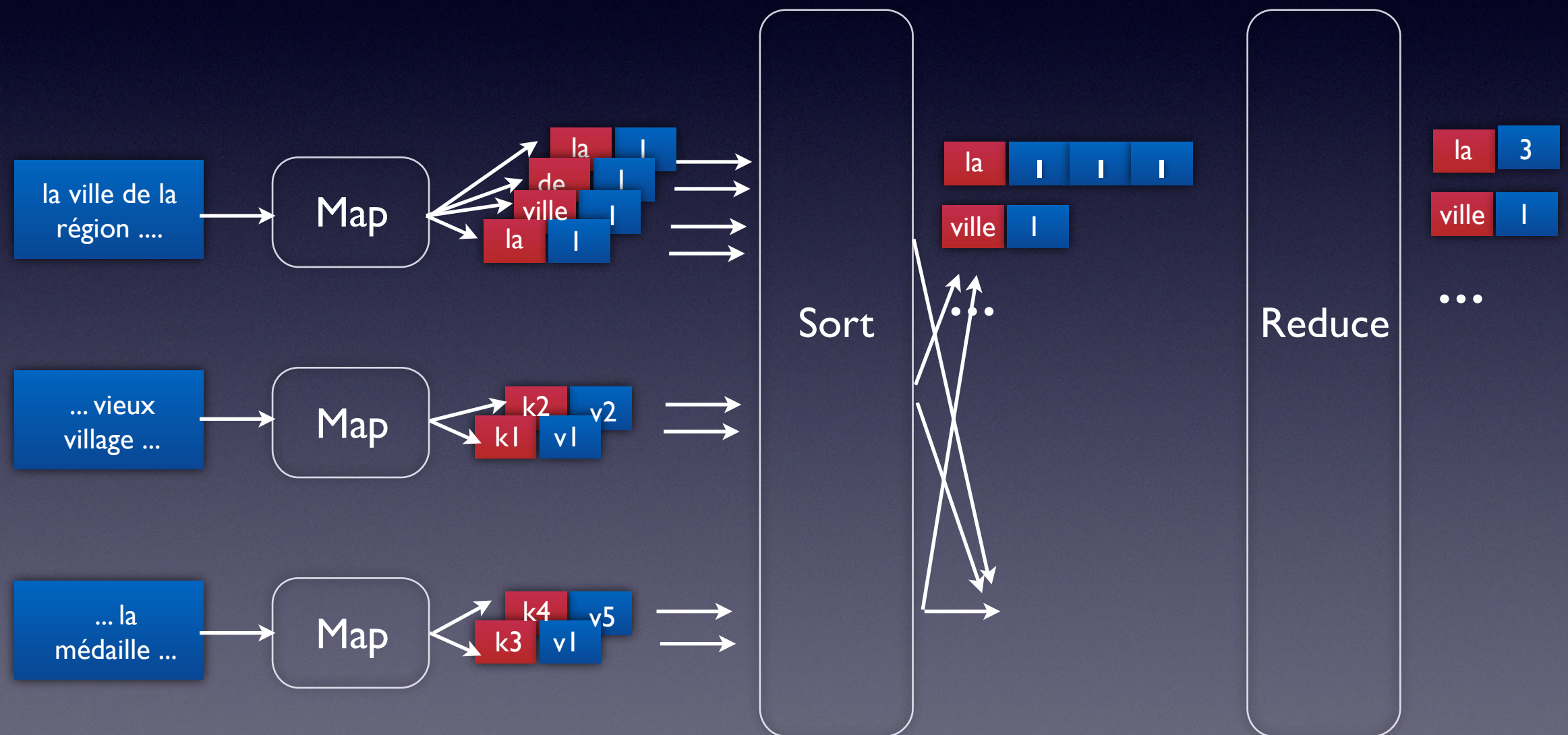
# Exemple: word count

Problème: *parmi un ensemble de textes, compter le nombre d'occurrences de chaque mot.*

- Données: un ensemble de fichiers textes
- **Map** : sur chaque texte, décomposer en mots, et à chaque mot  $m$ , ajouter à la liste  $[(m, 1), \dots]$
- Sort : le système trie/regroupe les paires selon la clé  $m$ , dans une liste  $[(m, [1, 1, \dots]), \dots]$
- **Reduce** : les valeurs  $1$  sont additionnées pour chaque mot :  $[(m, 2), \dots]$



# Schéma général




...devrait s'appeler Map-Sort-Reduce ...



# Pseudo-code word count

N'est pas utilisé ici. Contient  
généralement des informations  
relatives à input\_values :  
position des données dans le  
fichier, nom du fichier lu, ...



```
Map(Long input_key, String input_values) :  
    foreach word w in input_values:  
        EmitIntermediate (w, «1»);
```

```
Reduce (String key, Iterator intermediate_values):  
    int result=0;  
    foreach v in intermediate_values:  
        result += ParseInt( v );  
    Emit (key, String( result ));
```



# Exemple: index renversé

*Problème: soit un ensemble de fichiers contenant des mots. Etablir une liste des mots, et pour chaque mot une liste des fichiers où il apparaît.*



```
F1 : Strasbourg Nantes Paris  
F2 : Mulhouse Colmar Strasbourg
```

```
Colmar : F2  
Nantes : F1  
Mulhouse: F2  
Paris : F1  
Strasbourg : F1 F2
```



# Pseudo-code

## index renversé

```
Map(String filename, String line) :  
    foreach word w in line:  
        EmitIntermediate( w, filename );
```

```
Reduce (String key, Iterator intermediate_values):  
    // key=word, intermediate_values=filenames  
  
    foreach f in intermediate_values:  
        result += f + ' ';  
  
    Emit( key, result );
```



# Implémentations Map-Reduce

- Des bibliothèques MapReduce existent pour C++, C#, Erlang, Java, Python, Ruby, R
- La plus connue (hors celle propriétaire de Google) est Hadoop (projet Apache).





# Signatures

En hadoop, les fonctions map et reduce :

- **map** :  $(k_1, v_1) \mapsto [(k_2, v_2)]$ 
  - *Pour chaque paire clé-valeur reçue, on émet un ou plusieurs couple clé-valeur.*
- **reduce** :  $k_2, [v_2] \mapsto [(k_3, v_3)]$ 
  - *Pour chaque clé unique et sa liste de valeurs associées, on émet un ou plusieurs couple clé-valeur.*



# Map in Hadoop

```
public class Mapper <KEYIN, VALUEIN, KEYOUT, VALUEOUT>{  
  
    public class Context extends MapContext<KEYIN, VALUEIN, KEYOUT,  
VALUEOUT> {  
        // ...  
    }  
  
    public void map(KEYIN key, VALUEIN value, Context context)  
        throws IOException, InterruptedException {  
        // ...  
    }  
}
```



# Reduce in Hadoop

```
public class Reducer <KEYIN, VALUEIN, KEYOUT, VALUEOUT> {  
  
    public class Context extends ReducerContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {  
        // ...  
    }  
  
    public void reduce(KEYIN key, Iterable<VALUEIN> values, Context context)  
        throws IOException, InterruptedException {  
        // ...  
    }  
}
```



# wc avec Hadoop

```
public static class TokenizerMapper
    extends Mapper<LongWritable, Text, Text, IntWritable>{

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        StringTokenizer itr = new StringTokenizer(value.toString());

        while (itr.hasMoreTokens()) {
            context.write(new Text(itr.nextToken(), IntWritable(1)));
        }
    }
}
```



On écrit une instance la classe abstraite Mapper  
en donnant le type de (k,v) entrée et sortie

(k,v) entrée: On recevra l'offset dans le fichier (un long)  
et un morceau des données

```
public static class TokenizerMapper
    extends Mapper<LongWritable, Text, Text, IntWritable>{

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        StringTokenizer itr = new StringTokenizer(value.toString());

        while (itr.hasMoreTokens()) {
            context.write(new Text(itr.nextToken(), IntWritable(1));
        }
    }
}
```

emit(key,val)

par défaut, 1 ligne de fichier

(k,v) sortie: on sort une (k,v) composée  
d'un mot et de la valeur 1



# wc avec Hadoop

```
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```



# Structure

```
public class WordCount {
```

```
    public static class TokenizerMapper
        extends Mapper<LongWritable, Text, Text, IntWritable>{

        public void map(LongWritable key, Text value, Context context);

        /* Votre code pour map() ici */

    }
}
```

Classe interne héritant  
de Mapper dont on  
redéfinit la fonction map

```
    public static class IntSumReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {

        public void reduce(Text key, Iterable<IntWritable> values, Context context);

        /* Votre code pour reduce() ici */

    }
}
```

Classe interne  
héritant de Reducer

```
    public static void main(String[] args) {
        Job job = new Job(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(IntSumReducer.class);
    }
}
```

Configuration des jobs  
avant lancement



# InputSplit et Map

- Un *InputSplit* (ou split) représente un morceau des données traitée par une opération map.
- Chaque map traite un seul split.
- Chaque split est divisé en *records* de type (key,value) et la map traite successivement chaque record.



# InputFormat

- Le programmeur n'utilise pas *InputSplit* car les splits sont créés par une classe abstraite *InputFormat*.
- *FileInputFormat* est ensuite la classe de base pour toutes les implémentations de *InputFormat*.
- *FileInputFormat* fournit essentiellement les méthodes *addInputPath()* et *addInputPaths()*



# Compiling and Running

(Single Node Setup, Hadoop >= 2.4.x)

- Set \$JAVA\_HOME and \$HADOOP\_CLASS

```
export HADOOP_VERSION=2.4.0
export HADOOP_HOME=/usr/local/Cellar/hadoop/${HADOOP_VERSION}
export PATH=$HADOOP_HOME/bin:$PATH
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar:.
```

- Compile your project

```
hadoop com.sun.tools.javac.Main WordCount.java
```

- Make a JAR

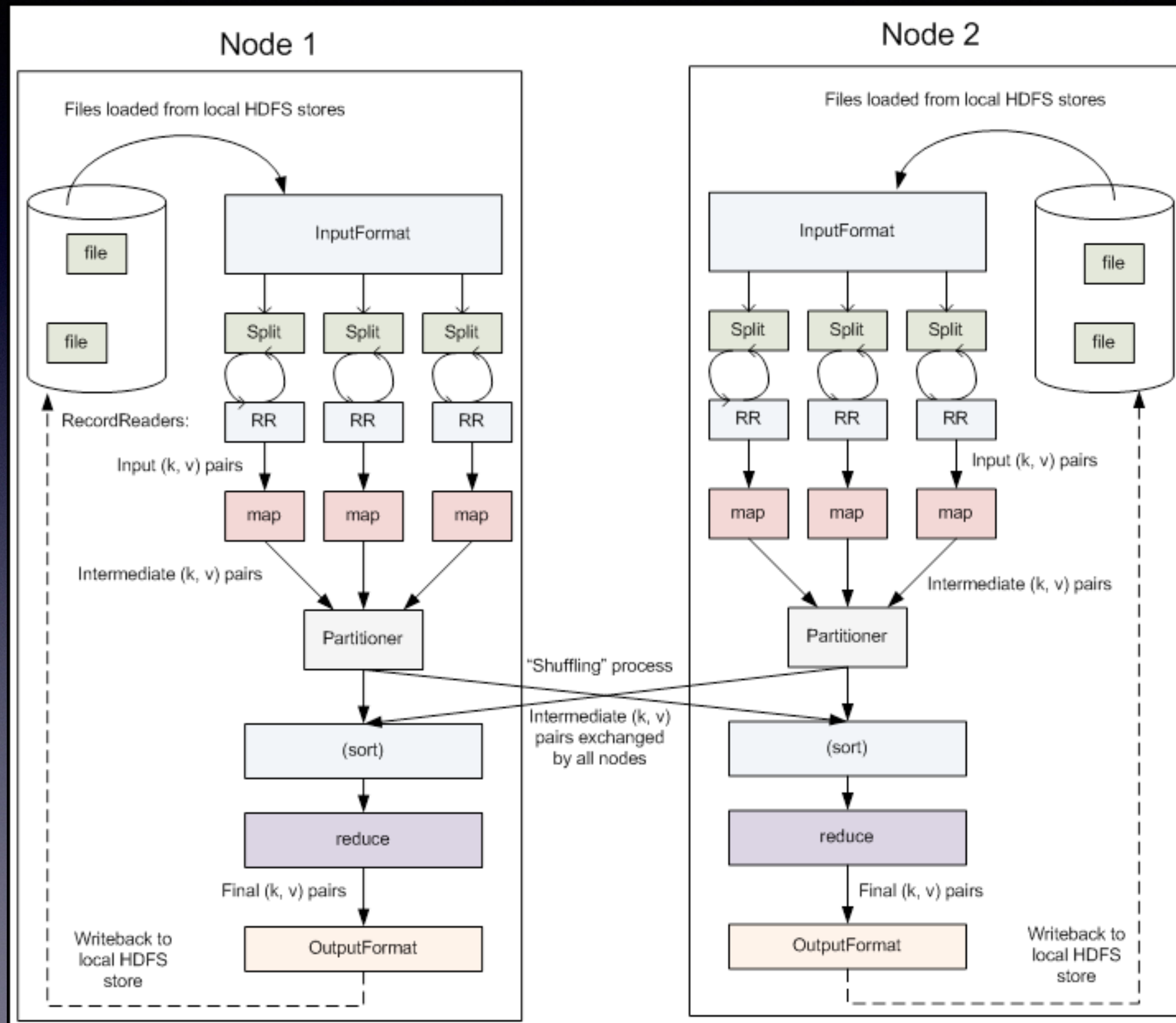
```
jar -cvf wc.jar *.class
```

- Run

```
hadoop jar wc WordCount input output
```



# Schéma général d'exécution



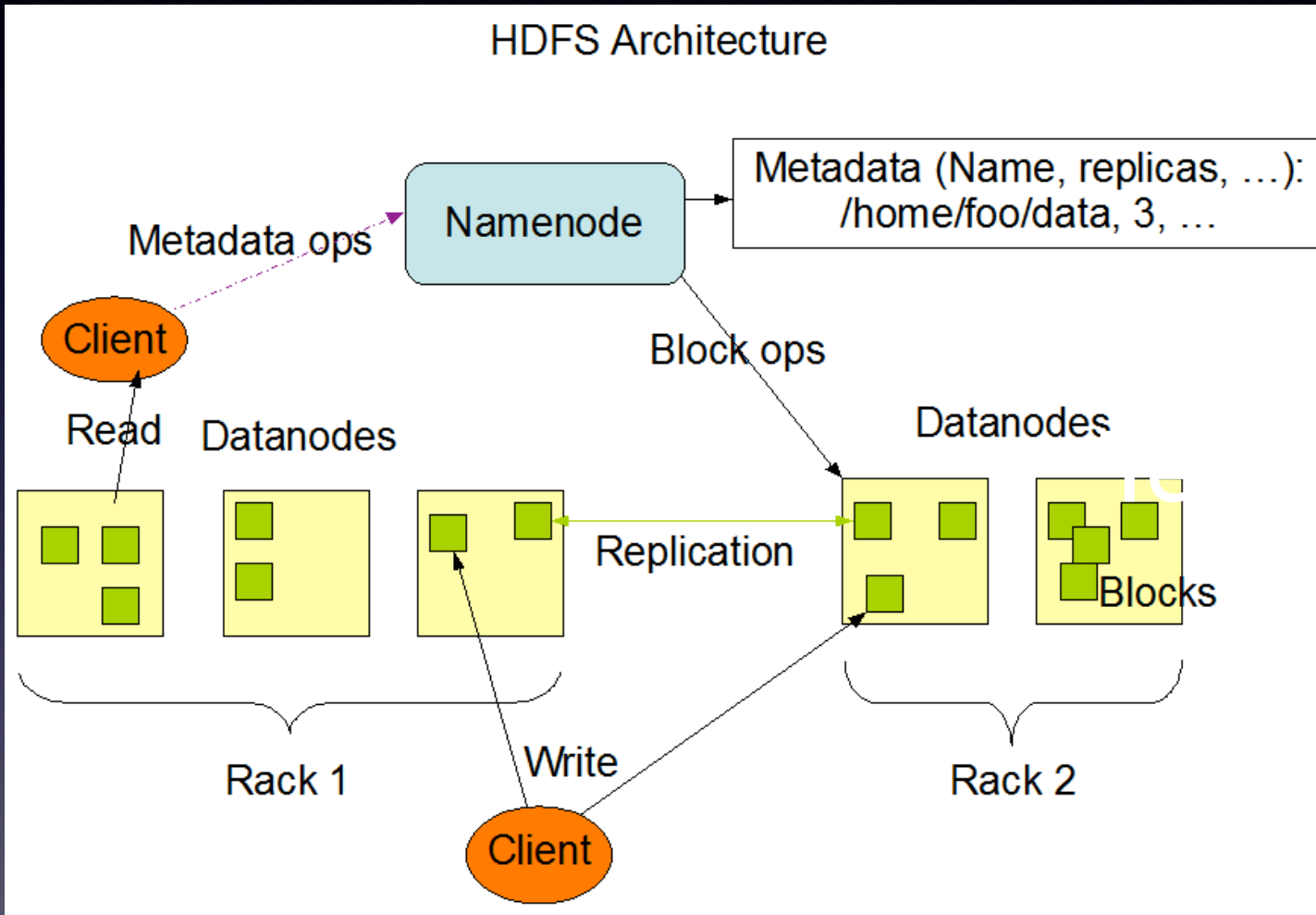


# Systeme de fichiers

- Les lectures et les communications entre processus (une JVM par process) se font par écriture de fichiers.
- Hadoop propose le pendant du Google File System (GFS): le Hadoop File System (HDFS)
- Caractéristiques:
  - non-posix,
  - write-once, read-many
  - block-size : par défaut 16 MB



# Systeme de fichiers



# Réplication des blocs de données, pour plus de disponibilité et fiabilité.

*Namenode* centralise les meta-données, et surveille vitalité des blocs par battement de coeurs.



Map-Reduce pour des algorithmes de graphes ?

Google l'utilise pour calculer PageRank.



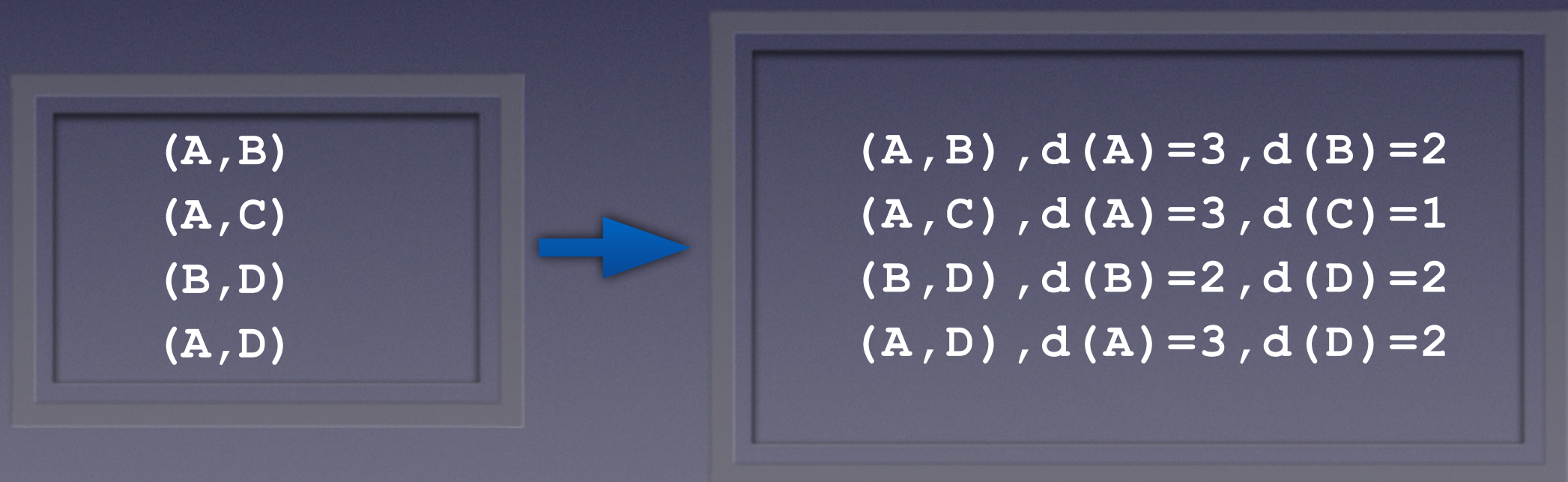
- Représentation répandue des graphes : la matrice d'adjacence :  $M_{i,j}=v$ ,  $v \neq 0$  quand il y a un lien du sommet  $i$  à  $j$ .
- Pour les grands graphes représentant la plupart des phénomènes réels, les matrice d'adjacence sont creuses (presque que des 0, lignes inutilement longues).
- Utiliser une représentation matrice creuse.
- Ex: sommet: (voisin,distance)  
A: (B, 4), (C, 3), (D, 1)



# Exemple : degrés des sommets

Problème: soit la liste des arrêtes d'un graphe exprimées par des relation binaires:  $(A,B)$  pour une relation non-orientée entre sommets A et B.

Etablir pour chaque arrête, le degré de chacun des deux sommets de la relation.

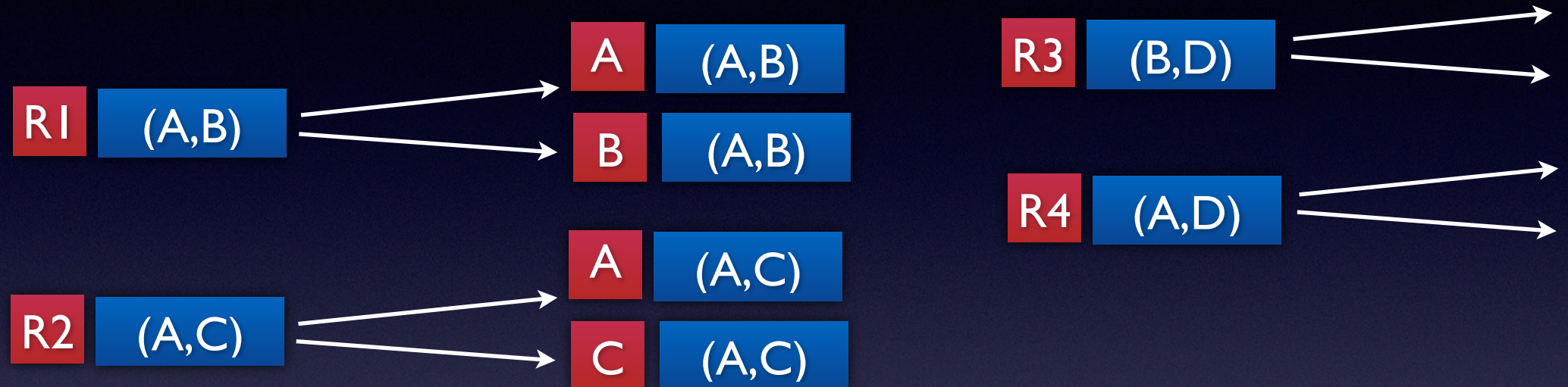




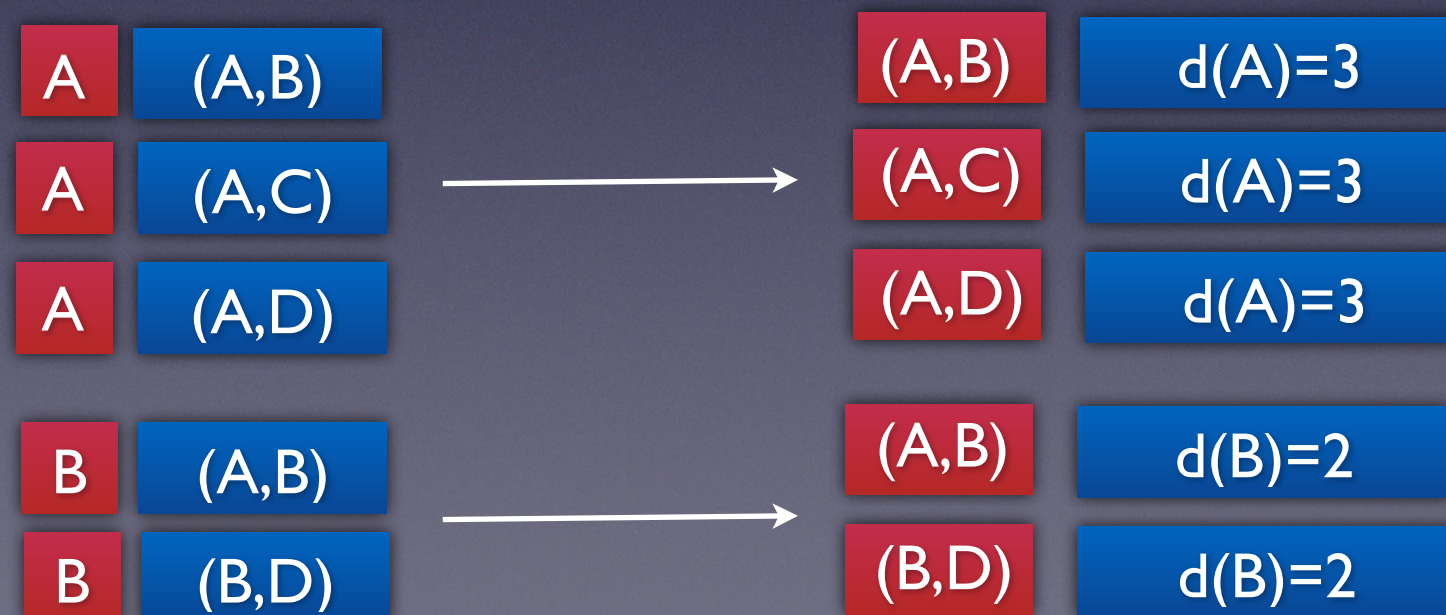
# Exemple : degrés des sommets

Nécessite 2 passes:

Map I : éclater les 2 sommets



Reduce I : compte le nombre d'enregistrements avec même clé, et ré-émet la relation comme clé avec le degré comme valeur.



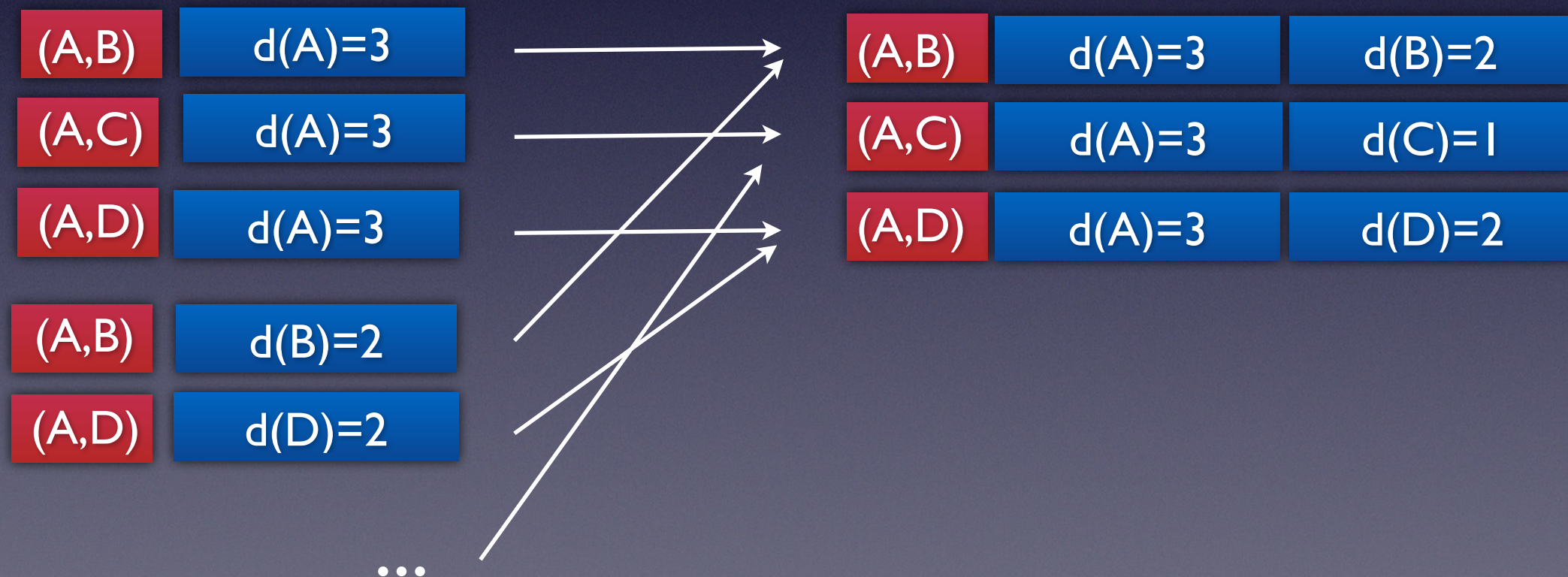


# Exemple : degrés des sommets

Passe 2:

**Map2 : identité** (quand appariement map et reduce obligatoire comme avec Hadoop)

**Reduce2 : agrège les degrés trouvés pour une même relation**

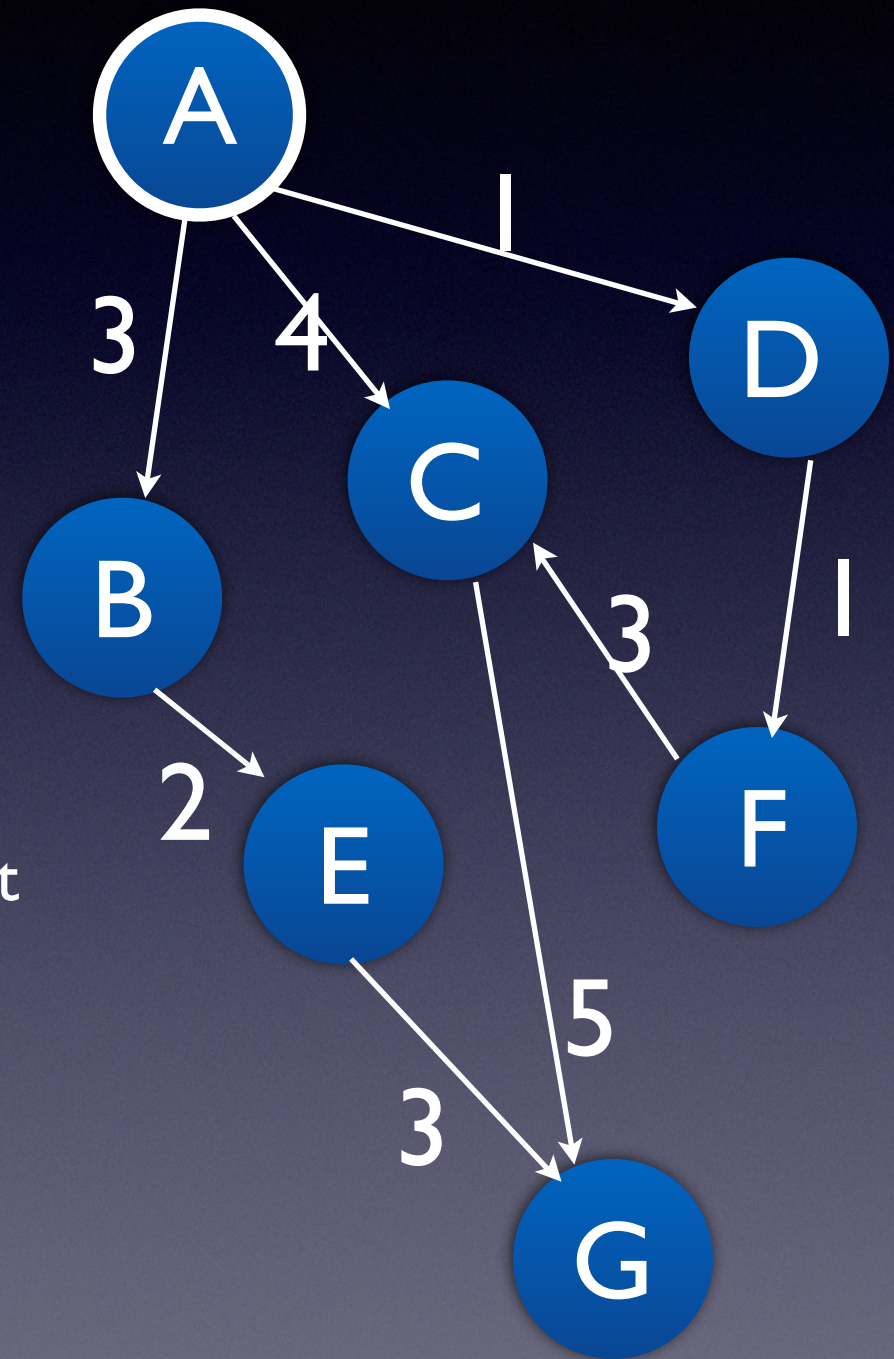




# Plus court Chemin

## Formulation du problème

- Un noeud initial est donné (ici A)
- On connaît les distances de voisin à voisin
- On veut connaître la distance minimum reliant le noeud initial à tous les autres noeuds.





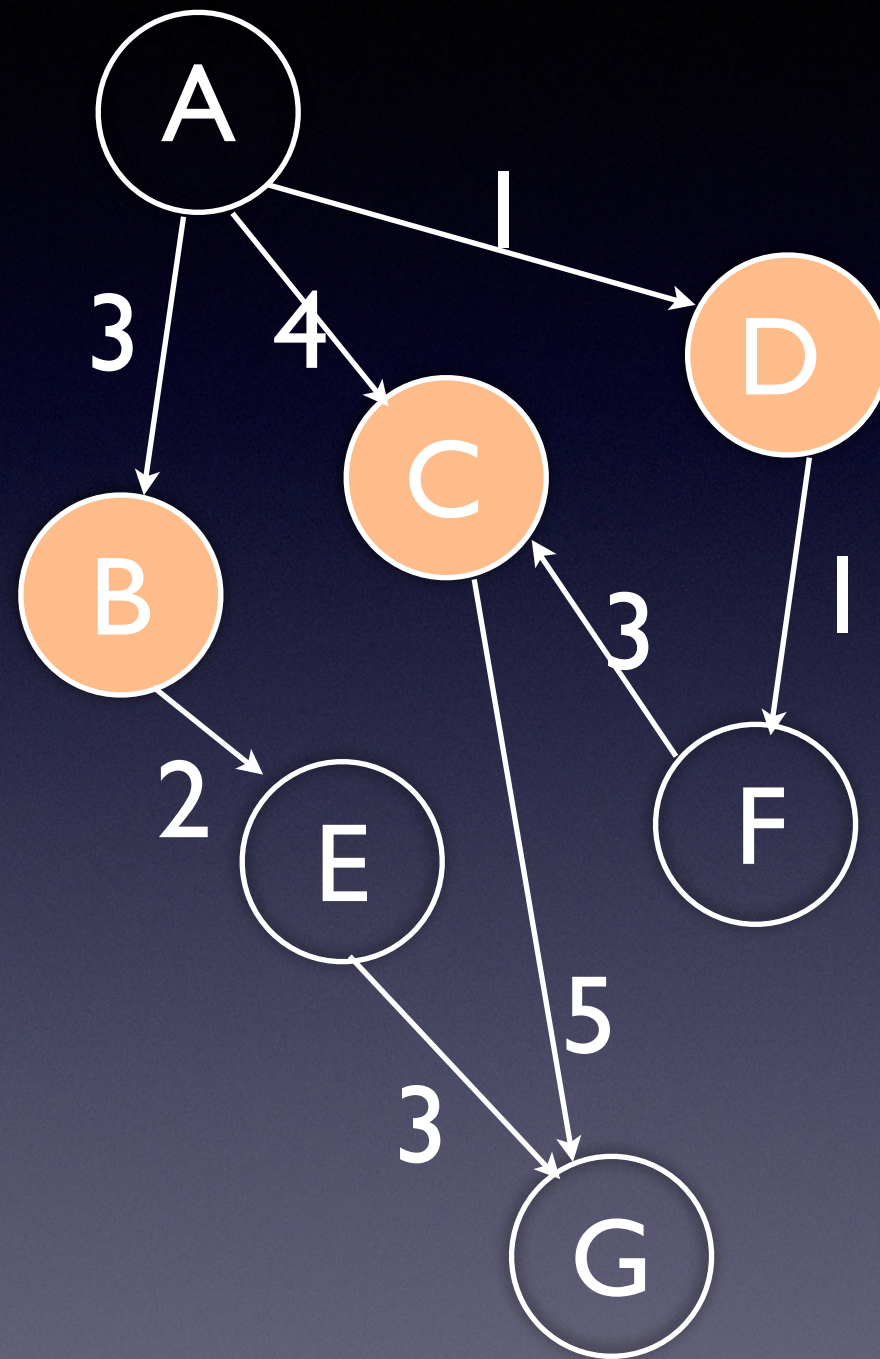
# Algorithme de Dijkstra

- $S$  : ensemble des sommets,
- $s_0$  : sommet initial
- $\text{neighbor}(a)$  rend l'ensemble des sommets connectés à un sommet  $a$ ,
- $\text{dist}(a,b)$  rend la distance entre 2 sommets adjacents  $a$  et  $b$
- $U$  : l'ensemble des sommets non visités, trié par ordre alpha
- $D$  : distance minimale de  $s_0$  aux autres sommets (le résultat)

```
/* initialization */
forall s ∈ S
    D[s] ← infinity
D[s0] = 0
U = S
/* computation */
while U ≠ {} do
    a ← head(U)
    U ← tail(U) /* removes a from unvisited */
    foreach b ∈ neighbor(a)
        D[b] ← min ( D[b] , D[a] + dist(a,b) )
    endfor
enddo
```

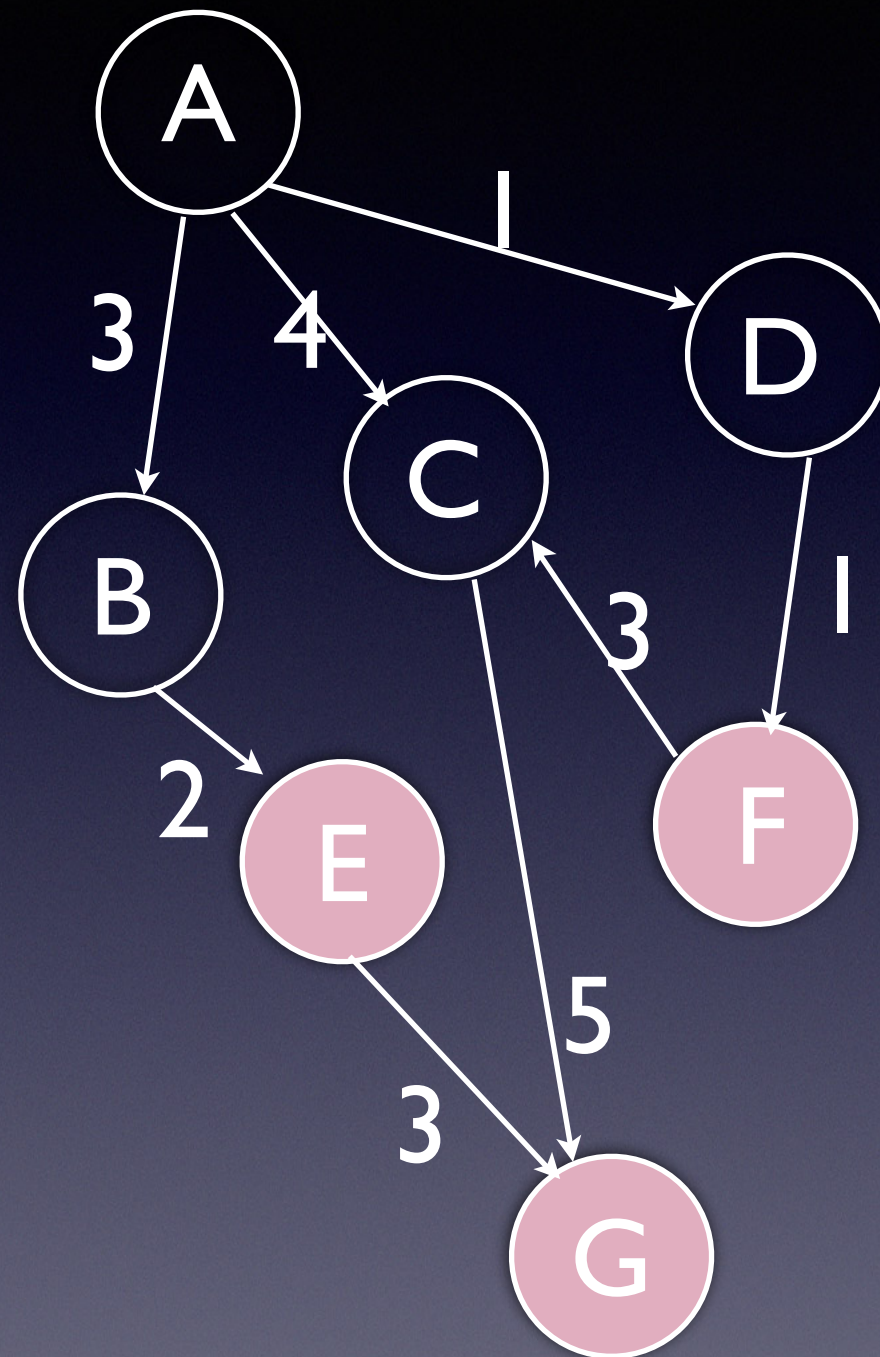


- ✦ Parcours en largeur: à partir d'un sommet, on examine d'abord tous les voisins.
- ✦ Dans le cas d'un graphe, nécessité de colorer les sommets pour l'arrêt.



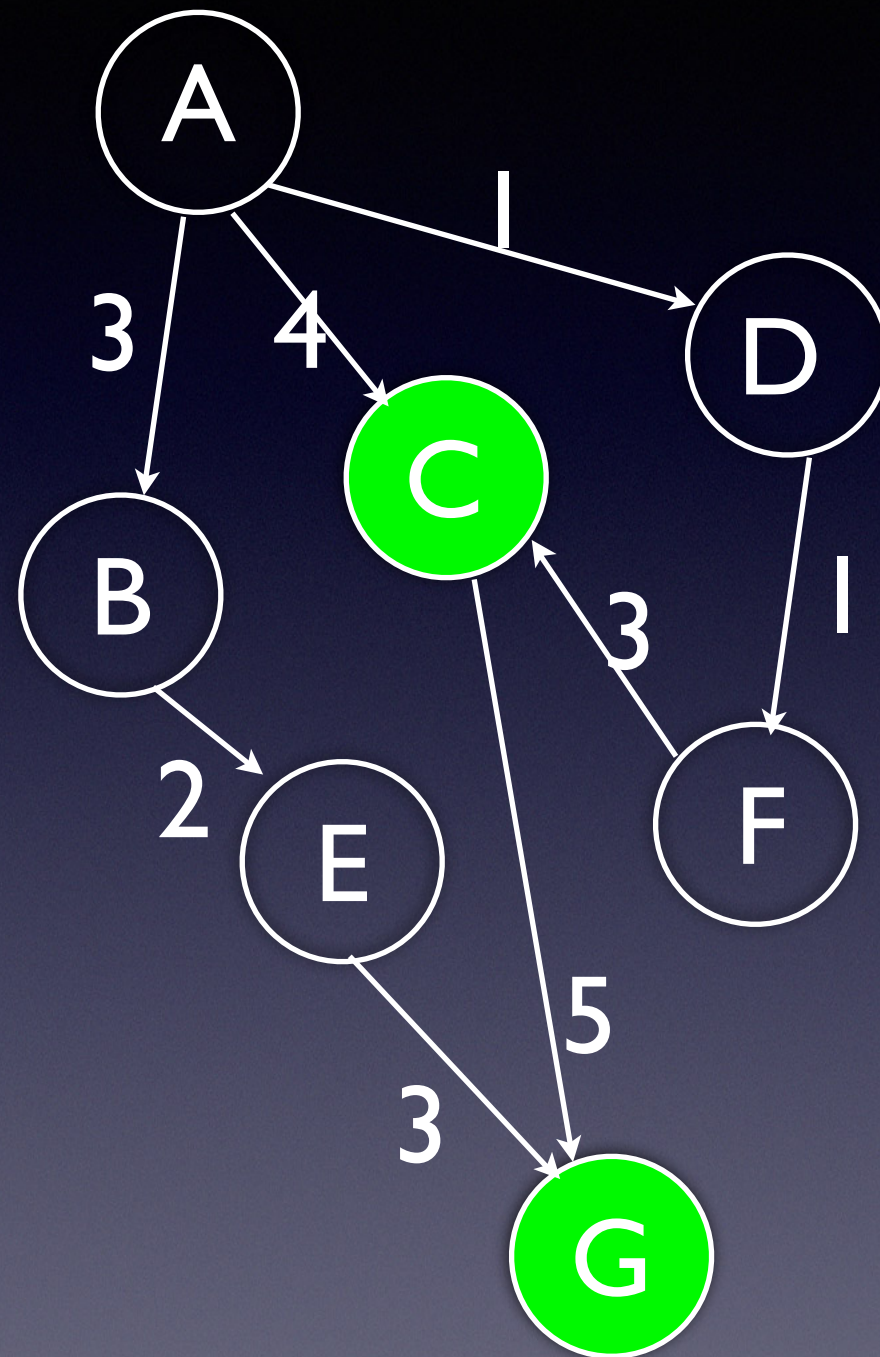


- ✦ Parcours en largeur: à partir d'un sommet, on examine d'abord tous les voisins.
- ✦ Il faut  $n$  itérations,  $n$  diamètre du graphe.
- ✦ Problème de l'arrêt.



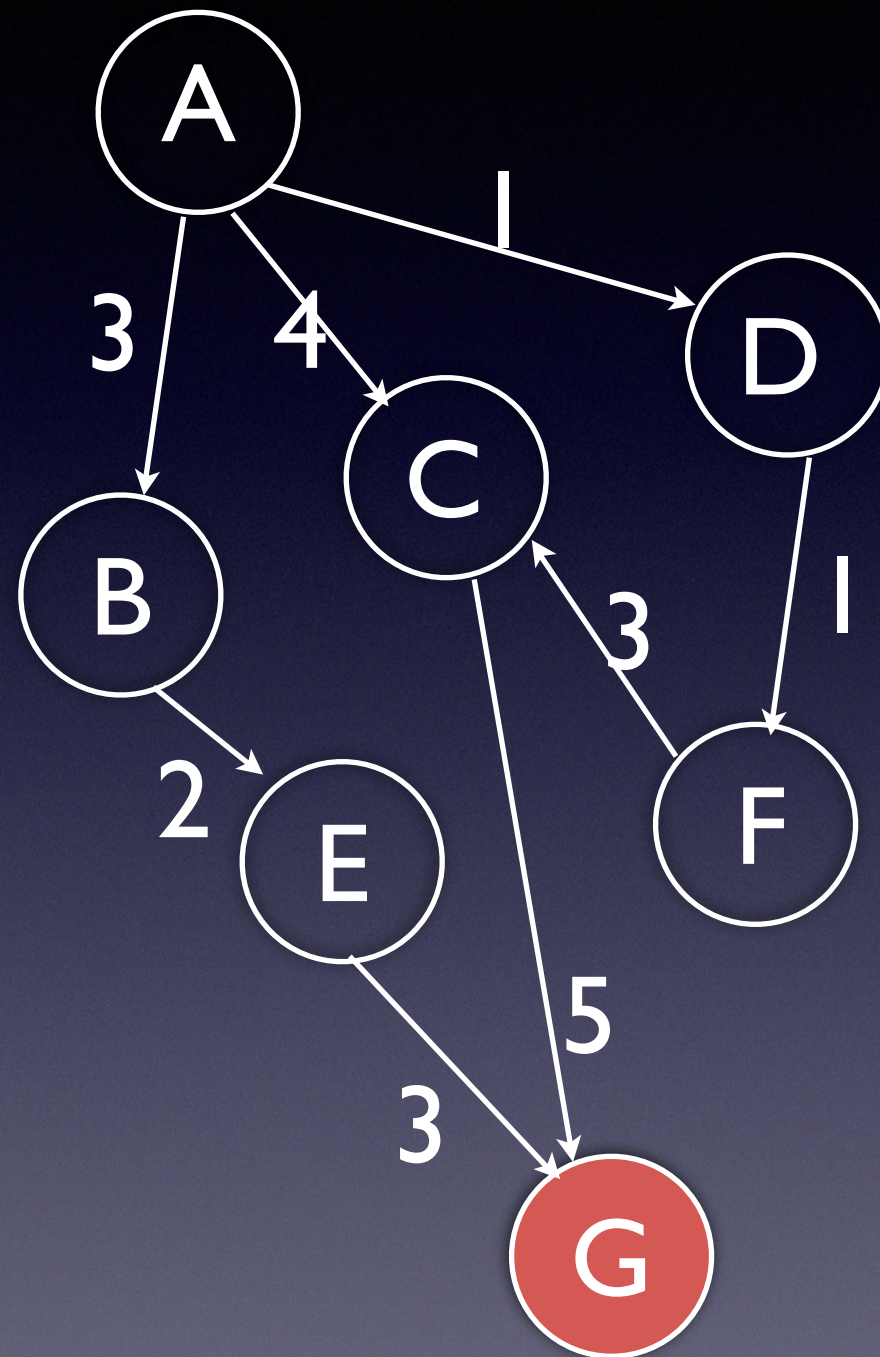


- ✦ Parcours en largeur: à partir d'un sommet, on examine d'abord tous les voisins.
- ✦ Il faut  $n$  itérations,  $n$  diamètre du graphe.
- ✦ Problème de l'arrêt.

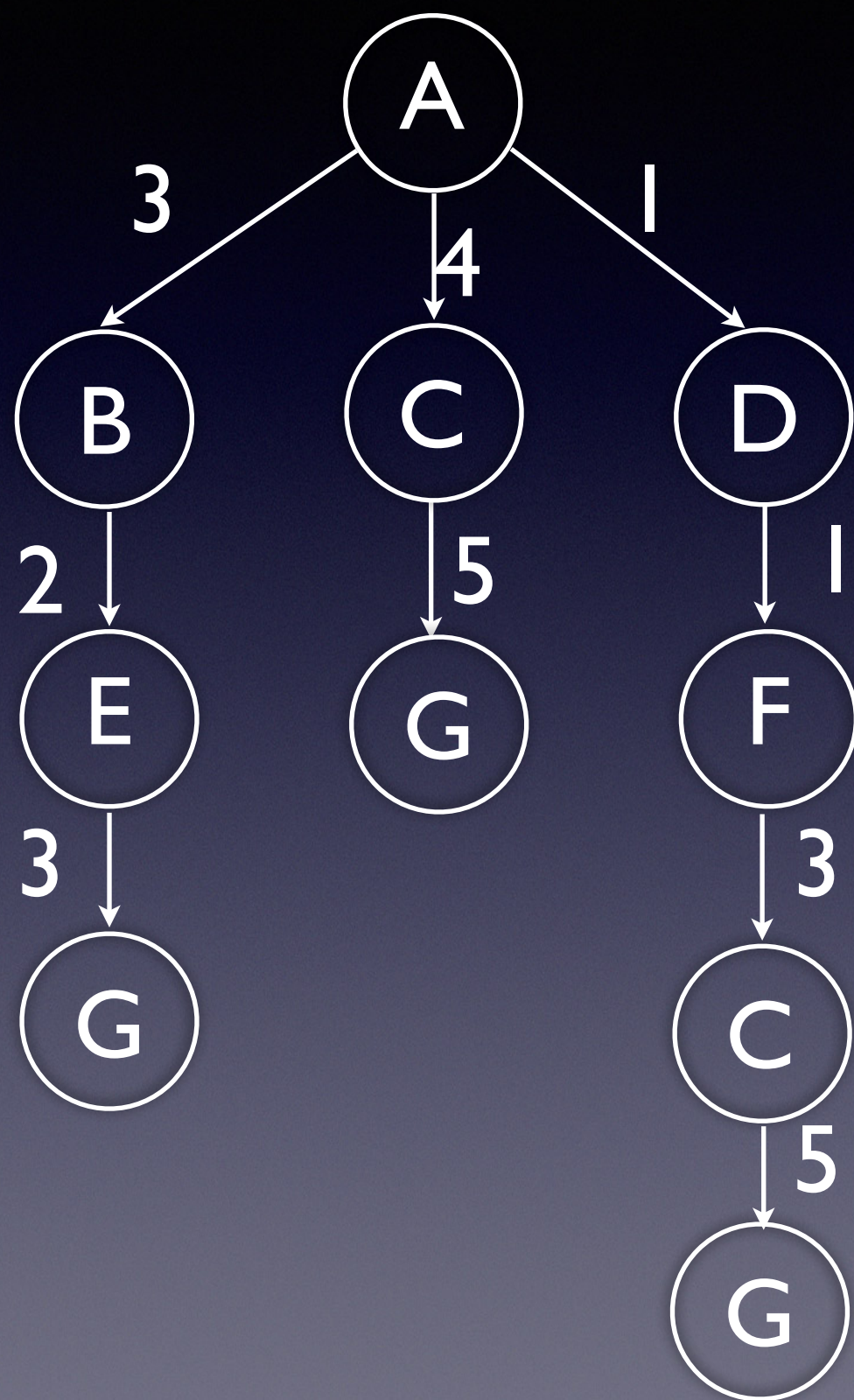




- ✦ Parcours en largeur: à partir d'un sommet, on examine d'abord tous les voisins.
- ✦ Il faut  $n$  itérations,  $n$  diamètre du graphe.
- ✦ Problème de l'arrêt.









# Problème de l'arrêt

- Idées possibles:
  - Nombre connu d'itérations (diamètre du graphe)
  - Constater la stabilité de la solution
  - Utiliser une variable globale - Hadoop Counters pour établir un consensus
  - Diffuser en permanence l'information actualisée concernant les noeuds visités



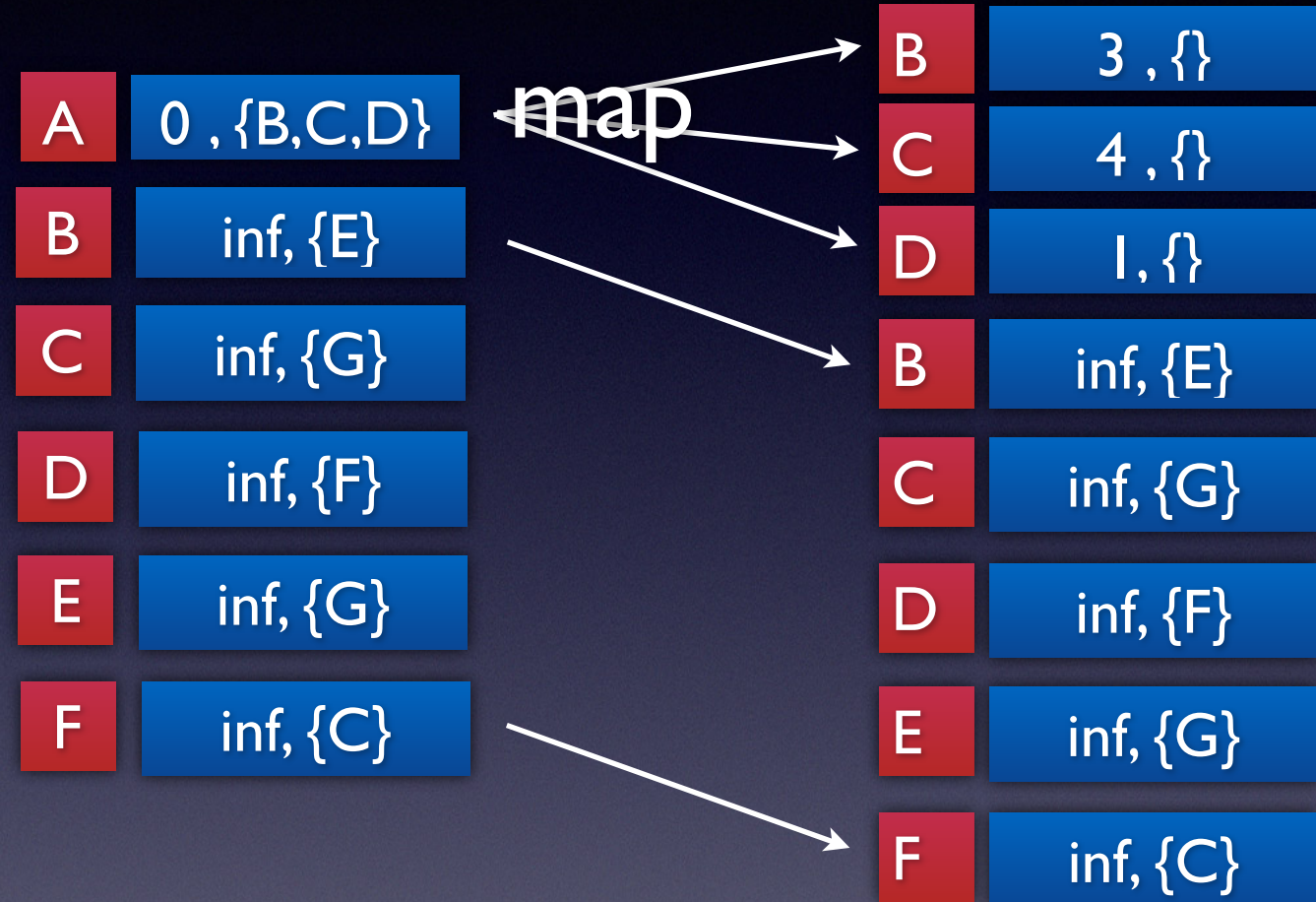
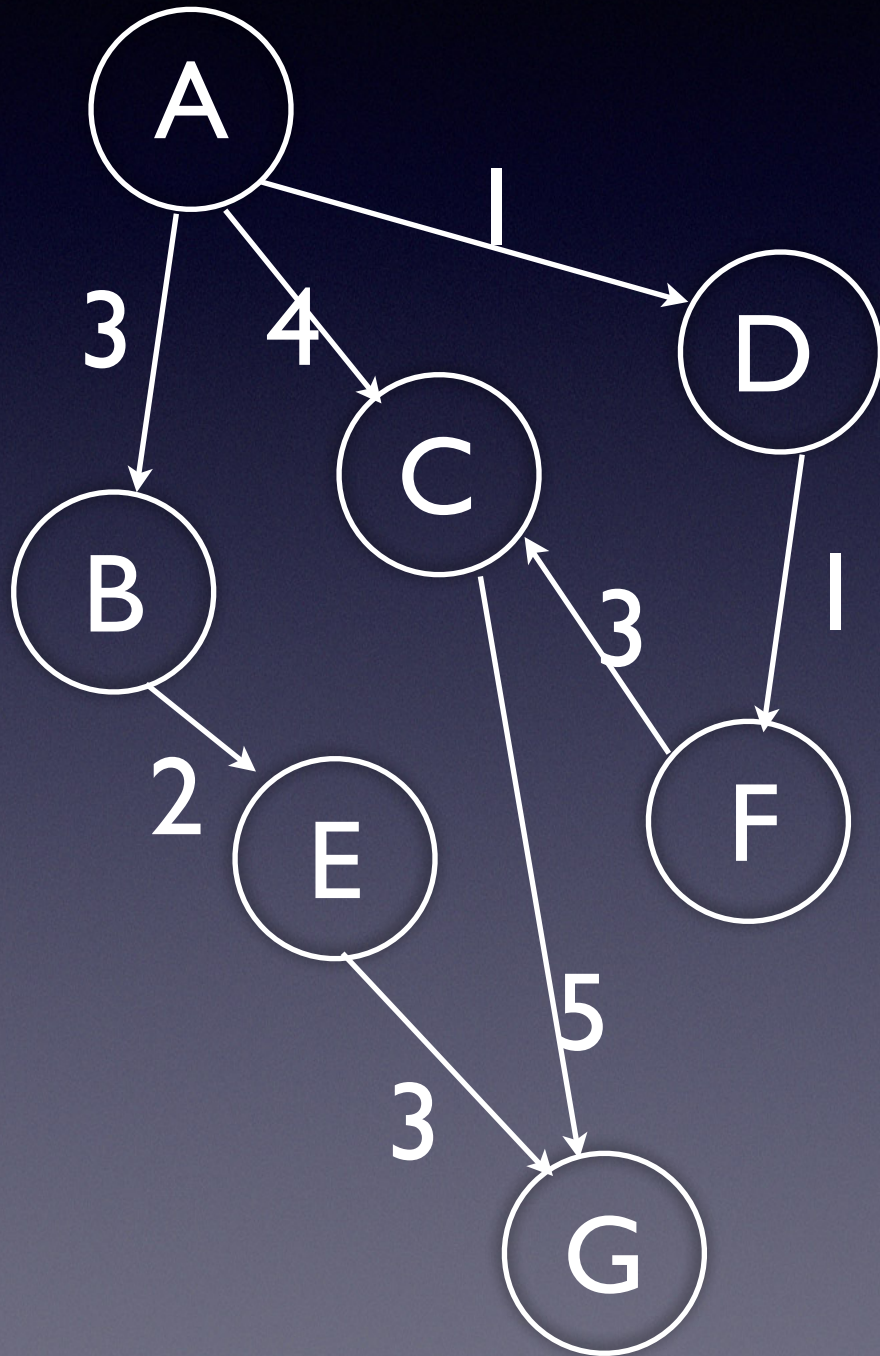
# Algorithme Map-Reduce

- ✦ La fonction **Map** reçoit :
  - clé : un sommet  $a$ ,
  - valeur :  $(D, voisins)$ 
    - \*  $D$  : distance du sommet à partir du début
    - \*  $voisins$  : liste des sommets accessibles depuis  $a$
  - $\forall b \in voisins, \text{émet } (b, \text{dist}(a,b))$
- ✦ La fonction **Reduce** traite toutes les distances possibles vers un  $b$  donné et sélectionne le plus petit.



# Algorithme Map-Reduce

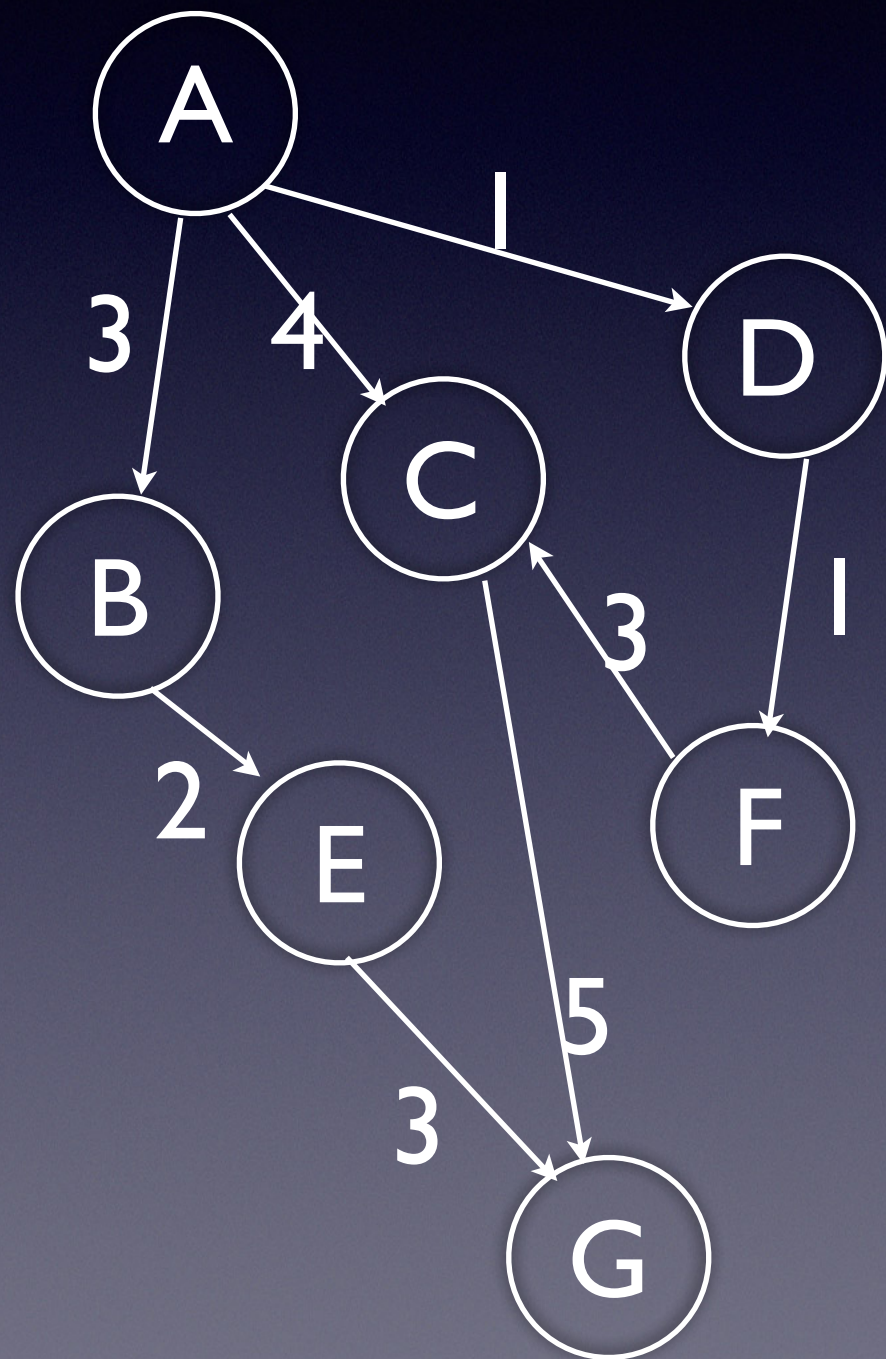
On part de A: la distance du début est 0. Pour les autres, initialement infinie.



Ce map génère des paires, comme (B,(3,{ })), que l'on peut interpréter comme : à partir de A, on atteint B, en ayant parcouru une distance de 3.



On réduit en prenant pour une même clé, la distance minimum.



B	3, {}
C	4, {}
D	1, {}
B	inf, {E}
C	inf, {G}
D	inf, {F}
E	inf, {G}
F	inf, {C}

Reduce

B	3, {E}
C	4, {G}
D	1, {F}
E	inf, {G}
F	inf, {C}

passee I



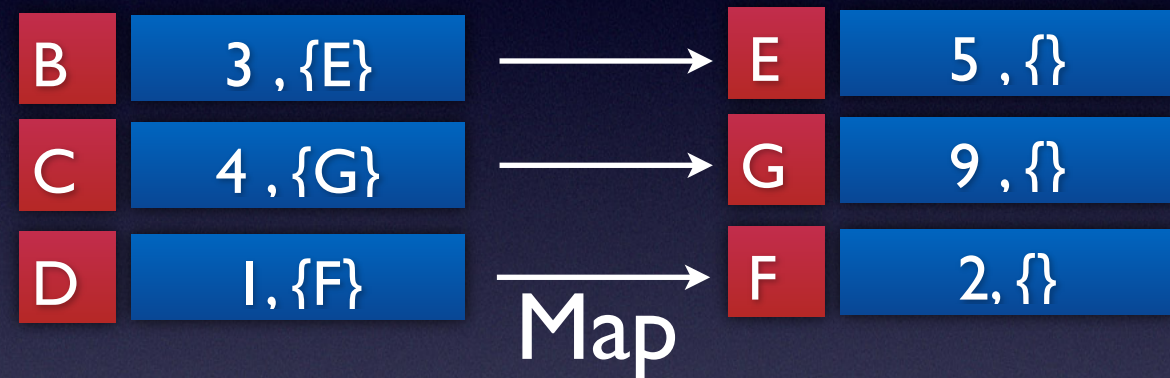
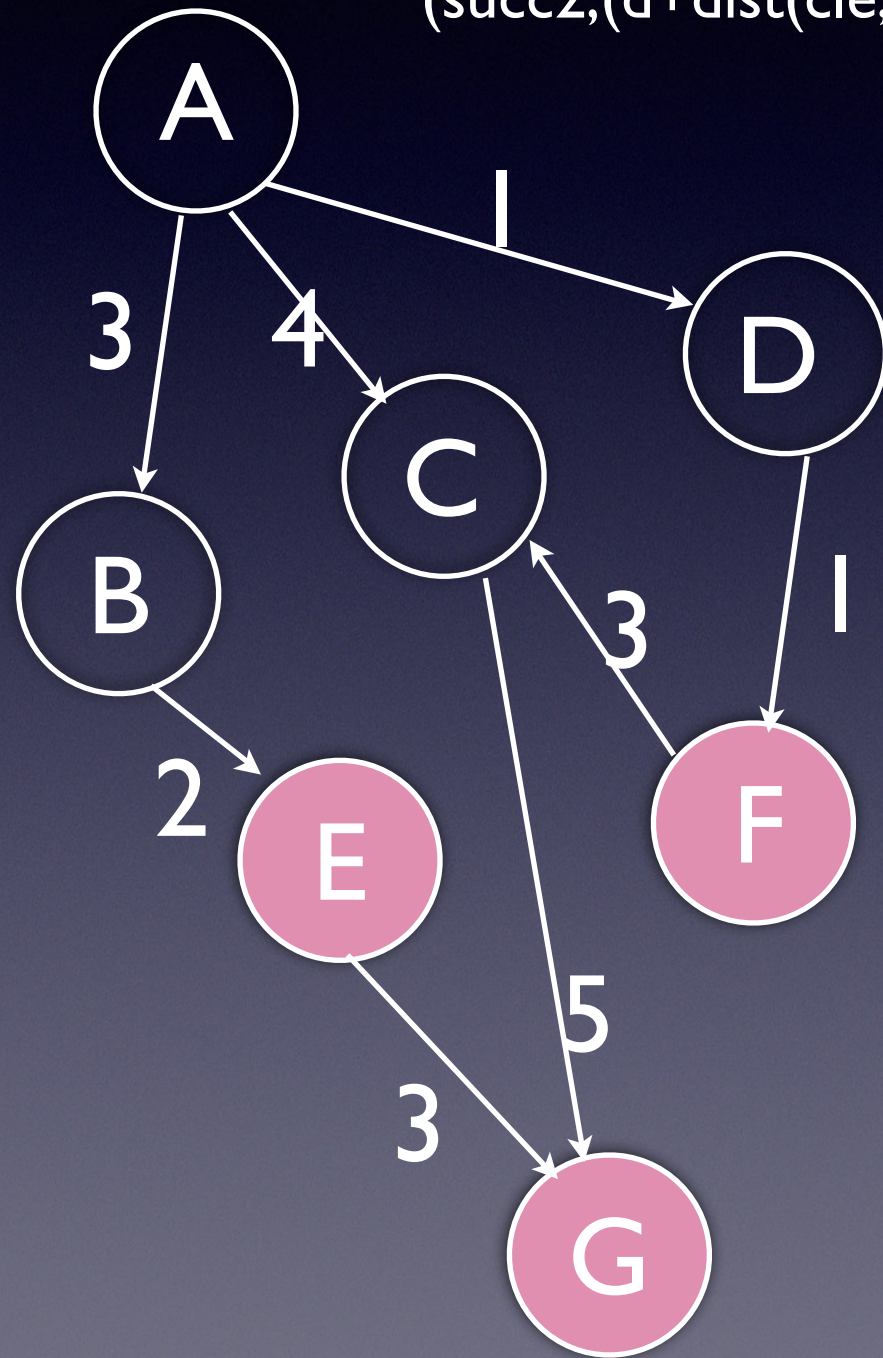
Le deuxième map génère de nouvelles clés 'sommet' en substituant

$(\text{clé}, (d, \{\text{succ1}, \text{succ2}, \dots\}))$

par

$[(\text{succ1}, (d + \text{dist}(\text{clé}, \text{succ1}), \{\}));$

$(\text{succ2}, (d + \text{dist}(\text{clé}, \text{succ2}), \{\})); \dots]$

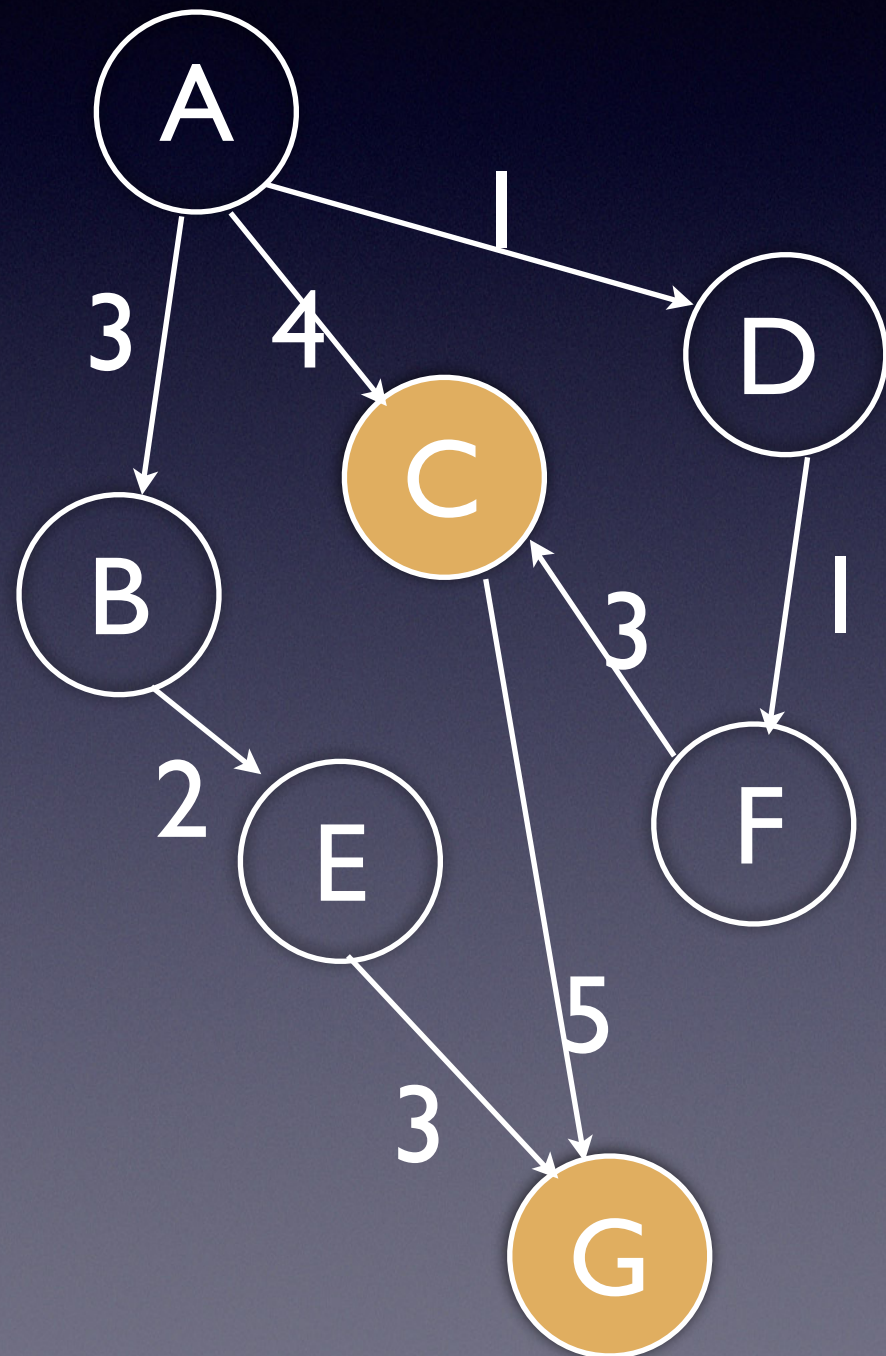


...

pas 2



On réduit en prenant pour une même clé, la distance minimum

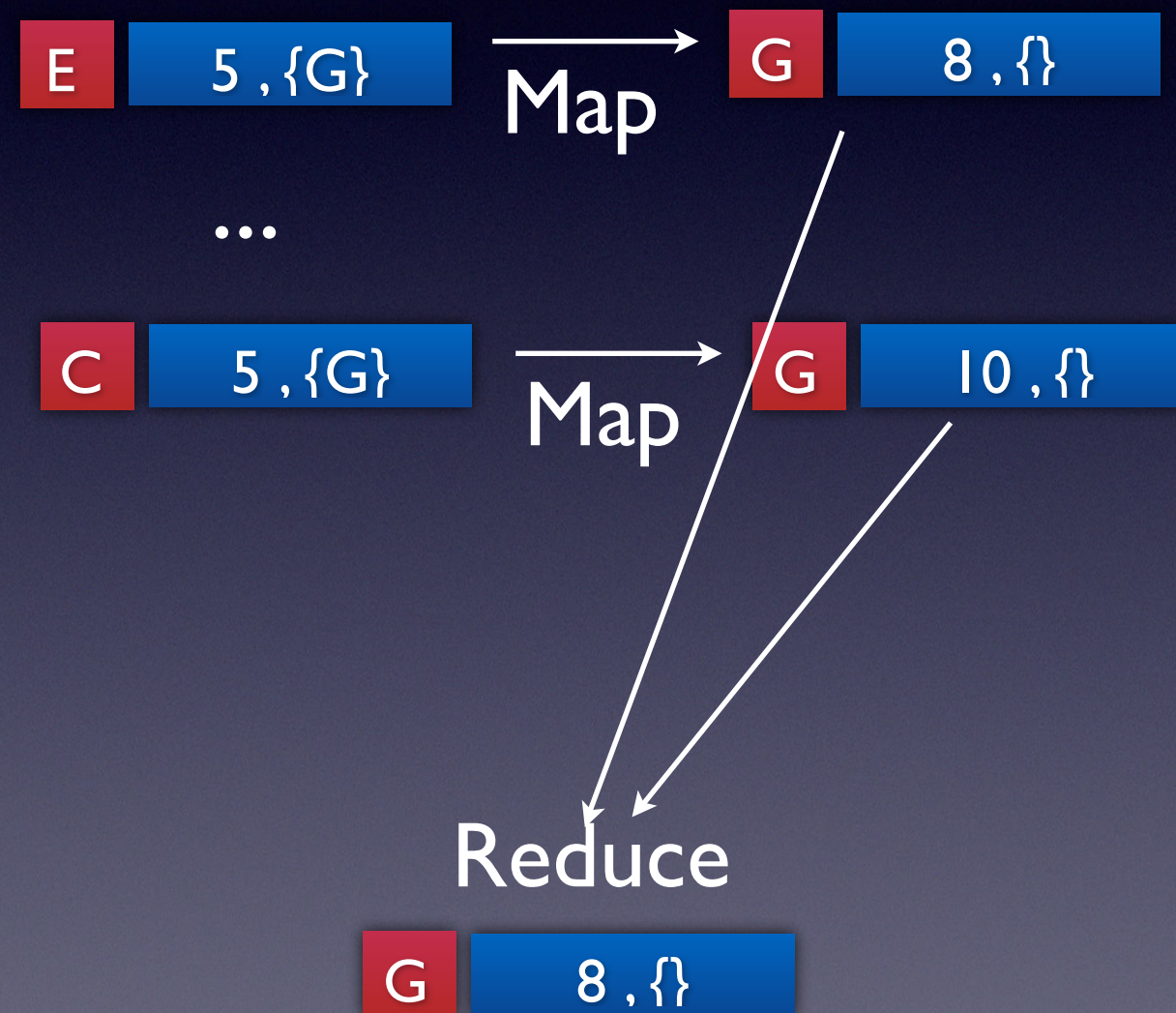
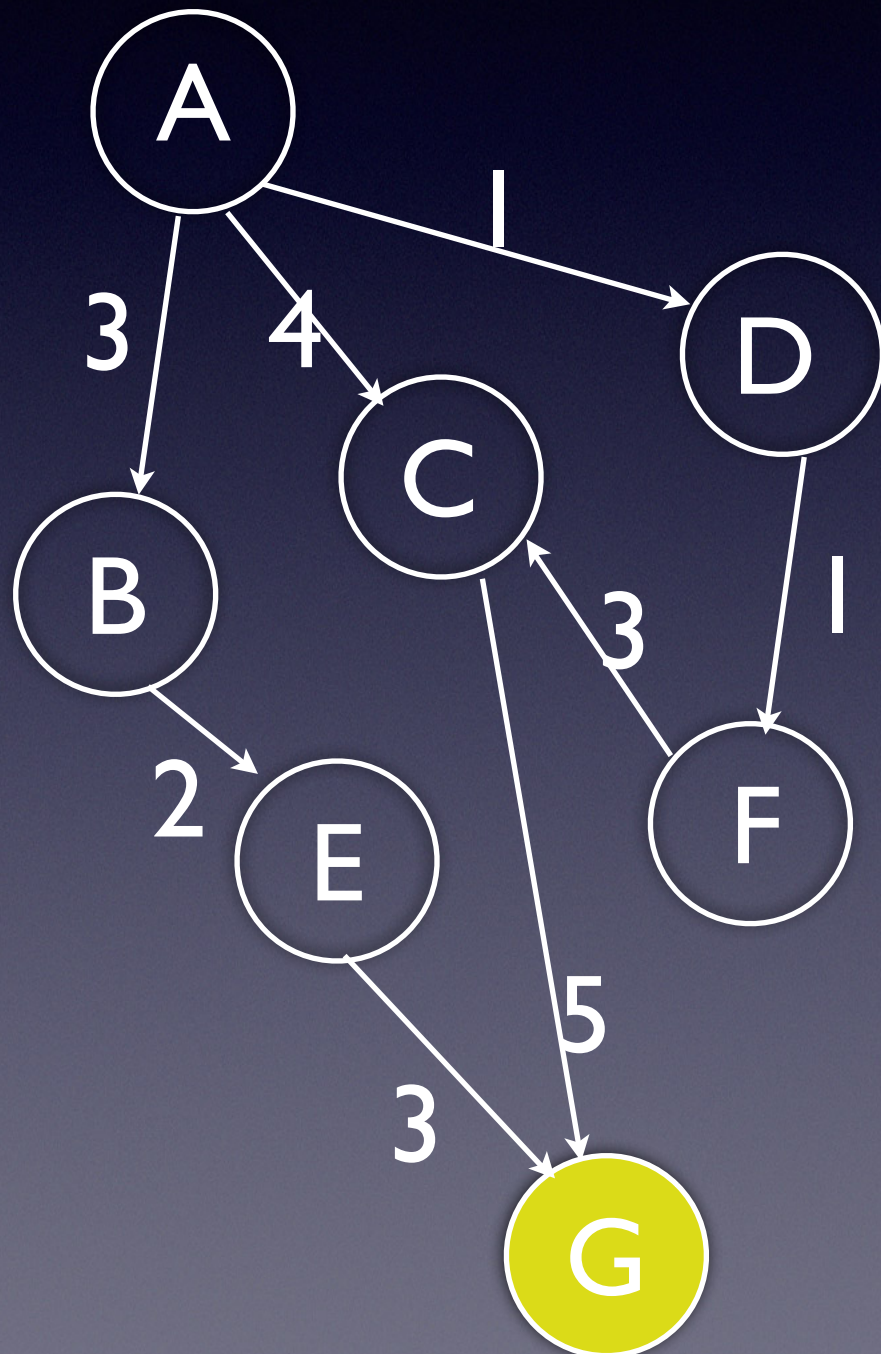


...

pas 3



On réduit en prenant pour une même clé, la distance minimum





# Input File Format

```
% cat inputSP
A 0 B,3;C,4;D,1
B 999 E,2;
C 999 G,5;
D 999 F,1;
E 999 G,3;
F 999 C,1;
G 999 {}
```



# Produit de matrices

A			B				C		
$a_{11}$	$a_{12}$	$a_{13}$	*	$b_{11}$		=	$c_{11}$		
				$b_{21}$					
				$b_{31}$					

La matrice résultat:  $c_{ij} = \sum_k a_{ik} * b_{kj}$



# Produit de matrices

$a_{11}$  est utilisé pour  $c_{11}, c_{12}, c_{13}$

$a_{12}$  est utilisé pour  $c_{11}, c_{12}, c_{13}$

...

$a_{21}$  est utilisé pour  $c_{21}, c_{22}, c_{23}$

**Idée 1** : regrouper les valeurs de A selon leur utilisation dans C

on lit :  $a_{11}, a_{12}, a_{13} \Rightarrow \text{emit}(c_{11}, a_{11}); \text{emit}(c_{11}, a_{12}); \text{emit}(c_{11}, a_{13})$

...

on lit :  $a_{21}, a_{22}, a_{23} \Rightarrow \text{emit}(c_{21}, a_{21}); \text{emit}(c_{21}, a_{22}); \text{emit}(c_{21}, a_{23})$

...



# Produit de matrices

Idem pour B

$b_{11}$  est utilisé pour  $c_{11}, c_{21}, c_{31}$

$b_{12}$  est utilisé pour  $c_{12}, c_{22}, c_{32}$

...

$b_{21}$  est utilisé pour  $c_{11}, c_{21}, c_{31}$

on lit :  $b_{11}, b_{12}, b_{13} \Rightarrow \text{emit}(c_{11}, b_{11}); \text{emit}(c_{21}, b_{11}); \text{emit}(c_{31}, b_{11})$

...

on lit :  $b_{21}, b_{22}, b_{23} \Rightarrow \text{emit}(c_{11}, b_{21}); \text{emit}(c_{21}, b_{21}); \text{emit}(c_{31}, b_{21})$

...



# Produit de matrices

Problème : *reduce* récupère les valeurs nécessaires pour calculer un  $c_{ij}$  mais dans un ordre quelconque.

Exemple :

$c_{11}$

$a_{11}; b_{11}; b_{21}; a_{12}; b_{31}; a_{13}$

...

$c_{21}$

$b_{11}; b_{21}; a_{21}; a_{22}; b_{31}; a_{23}$



# Produit de matrices

**Idée 2** : ajouter une information dans les valeurs indiquant comment apparier les produits.

Exemple :

**c<sub>11</sub>** (a<sub>11</sub>, 1); (b<sub>21</sub>, 2); (a<sub>12</sub>, 2); (b<sub>11</sub>, 1); (b<sub>31</sub>, 3); (a<sub>13</sub>, 3)

match !

...

match !

**c<sub>21</sub>** (b<sub>11</sub>, 1); (b<sub>21</sub>, 2); (a<sub>12</sub>, 1); (a<sub>22</sub>, 2); (b<sub>31</sub>, 3); (a<sub>23</sub>, 2)



# Input File Format

```
% cat matrices.in.dat
```

```
A : [0 1 2 3 4] [5 6 7 8 9]
```

```
B : [0 1 2] [3 4 5] [6 7 8] [9 10 11] [12 13 14]
```

Par défaut, chaque ligne lue déclenche un map



# Exemple supplémentaire: Clustering

- Classification non-supervisée: trouver  $k$  groupes de points tels que les points dans chaque groupe soient plus «proches» de ceux de leur groupe que des autres groupes.
- K-means :
  1. prendre au hasard  $k$  points initiaux, désignés comme «centres».
  2. pour chaque point, définir quel est le centre le plus proche. Le point rejoint le groupe associé au centre.
  3. pour chaque groupe, définir un nouveau centre, barycentre des points appartenant au groupe.
  4. Recommencer l'étape 2. Si aucun point ne change de groupe, fin de l'algorithme, sinon recommencer à l'étape 3.