Compte rendu TP PLP

Bruno Finel & Cassandre Chatard

Répartition du travail

La répartition du travail s'est faite de manière équitable.

Pour la partie Hadoop - MapReduce, nous avions tous les 2 fini individuellement les questions 2.7 et 2.8 en TP. Nous avons travaillé ensemble sur les questions restantes.

Pour la partie Spark, nous n'avons pas pu commencer le projet pendant les séances de TP à cause de problèmes d'installation. Nous avons fait l'ensemble des questions ensemble.

Table des matières

Partie Hadoop – Map Reduce

Toutes les questions ont été traitées dans un premier temps sur un ordinateur avec Hadoop et Virtual Box. Cependant après de nombreuses tentatives, nous n'avons jamais réussi à lancer les fichiers .jar depuis le terminal. Les questions ont donc été traitées sur Eclipse.

Question 2.7

Pour cette question nous nous sommes servis de l'architecture utilisée dans CompterLigneFile.java en y ajoutant une classe Tree et une fonction DisplayYearHeight qui prend en argument une ligne et renvoie les différents arguments des arbres (entre autres l'année et la hauteur). Le MainTree lit le fichier en entrée et applique la fonction DisplayYearHeight pour chaque ligne.

Ci-dessous un aperçu de ce que renvoie Eclipse dans la console (dernières lignes) :

```
<terminated> MainTree [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java (Dec 19, 2017, 10:56:12 PM)
N/A - 15.0
1814 - 20.0
1865 - 20.0
1939 - 25.0
1852 - 30.0
N/A - 20.0
1907 - 9.0
1847 - 40.0
1859 - 25.0
N/A - 12.0
1852 - 30.0
1863 - 12.0
1896 - 16.0
1918 - 32.0
1860 - 22.0
1870 - 15.0
```

Question 2.8

Cette question a été traitée de manière très similaire à la question 2.7. La classe Station définit les arguments pour chaque station. La fonction DisplayInfo prend en argument un eligne et renvoie les différents arguments des stations. Le MainStation lit le fichier en entrée et applique la fonction DisplayInfo pour chaque ligne.

Ci-dessous un aperçu de ce que renvoie Eclipse dans la console (dernières lignes) :

```
<terminated> MainStation [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java (Dec 19, 2017, 10:58:23 PM)
USAFCode: 99999 - name: DETROIT METRO AP              - country: US - elevation: +0202.4
USAFCode: 99999 - name: ALPENA PHELPS COLLINS FIELD   - country: US - elevation: +0211.2
USAFCode: 99999 - name: GRAND RAPIDS KENT COUNTY ARPT - country: US - elevation: +0244.8
USAFCode: 99999 - name: CHICAGO MEIGS FIELD           - country: US - elevation: +0180.1
USAFCode: 99999 - name: FRANKLIN                      - country: US - elevation: +0469.1
USAFCode: 99999 - name: MILWAUKEE TIMMERMAN           - country: US - elevation: +0227.1
USAFCode: 99999 - name: DUBUQUE MUNICIPAL AP          - country: US - elevation: +0329.2
USAFCode: 99999 - name: WATERLOO MUNICIPAL AP         - country: US - elevation: +0267.6
USAFCode: 99999 - name: GRAND FORKS AF                - country: US - elevation: +0277.7
USAFCode: 99999 - name: HIBBING CHISHOLM-HIBBING AP   - country: US - elevation: +0413.6
USAFCode: 99999 - name: LINCOLN 8 ENE                 - country: US - elevation: +0362.4
USAFCode: 99999 - name: LINCOLN 11 SW                 - country: US - elevation: +0418.2
USAFCode: 99999 - name: TOK 70 SE                     - country: US - elevation: +0609.6
USAFCode: 99999 - name: RUBY 44 ESE                   - country: US - elevation: +0078.9
USAFCode: 99999 - name: SELAWIK 28 E                  - country: US - elevation: +0006.7
USAFCode: 99999 - name: DENALI 27 N                   - country: US - elevation: +0678.2
```

Question 5.1

Le but de cet exercice est de calculer pour chaque mot de chaque document le TF-IDF (Term Frequency – Inverse Document Frequency). Cette valeur permet de connaître l'importance d'un mot dans un texte par rapport à un ensemble de texte. On utilise trois architectures MapReduce à la suite l'une de l'autre.

Approche Théorique

Le premier MapReduce prend en argument l'ensemble des textes constitués de mots. Le Mapper1 traite les textes, ie enlève les caractères spéciaux et met le texte en minuscule puis associe à chaque couple (mot, document) la valeur 1. Puis le Reducer1 renvoie en valeur le nombre d'occurrence d'un mot par document.

Sources :

callwild.txt
a b a d b
defoe-robinson-103.txt
b a e e

TFIDFMapper1 :

(Mot ; docID)	NbrMots
a ; callwild	1
b ; callwild	1
a ; callwild	1
d ; callwild	1
b ; callwild	1
b ; defoe-robinson-103	1
a ; defoe-robinson-103	1
e ; defoe-robinson-103	1
e ; defoe-robinson-103	1

TFIDFReducer1 :

(Mot ; docID)	NbrMots
a ; callwild	2
a ; defoe-robinson-103	1
b ; callwild	2
b ; defoe-robinson-103	1
d ; callwild	1
e ; defoe-robinson-103	2

Le MapReduce2 permet de réunir toutes les informations nécessaires pour calculer le TF-IDF d'un couple (mot, document) grâce à la formule suivante :

$$TFIDF\ (mot,\ doc) = TF(mot,\ doc)\ * IDF(\ doc)$$

$$TFIDF\ (mot,\ doc) = \frac{nbr\ occurence\ mot\ dans\ doc}{nbr\ mots\ dans\ doc} * ln(\frac{nbr\ de\ docs}{nbr\ de\ docs\ avec\ mot})$$

La seule information manquante est le nombre de mots par documents qui est calculée dans le Reducer2.

Source :	TFIDFMapper2 :	TFIDFReducer2 :

(Mot ; docID)	NbrMots
a ; callwild	2
a ; defoe-robinson-103	1
b ; callwild	2
b ; defoe-robinson-103	1
d ; callwild	1
e ; defoe-robinson-103	2

docID	(Mot; NbrMots)
callwild	a;2
defoe-robinson-103	a;1
callwild	b;2
defoe-robinson-103	b;1
callwild	d;1
defoe-robinson-103	e;2

(Mot ; docID)	(NbrMots ; MotParDoc)
a ; callwild	2;5
a ; defoe-robinson-103	1;4
b ; callwild	2;5
b ; defoe-robinson-103	1;4
d ; callwild	1;5
e ; defoe-robinson-103	2;4

Enfin, le MapReduce3 calcule le TFIDF pour chaque couple (mot, document).

Source :	TFIDFMapper3 :	TFIDFReducer3 :

(Mot ; docID)	(NbrMots ; MotParDoc)
a ; callwild	2;5
a ; defoe-robinson-103	1;4
b ; callwild	2;5
b ; defoe-robinson-103	1;4
d ; callwild	1;5
e ; defoe-robinson-103	2;4

Mot	(Doc ;NbrMots ; MotParDoc)
a	callwild ; 2 ; 5
a	defoe-robinson-103 ; 1 ; 4
b	callwild;2;5
b	; defoe-robinson-103;1;4
d	callwild ;1;5
e	defoe-robinson-103 2;4

(Mot ; docID)	TF-IDF
a ; callwild	0.000
a ; defoe-robinson-103	0.000
b ; callwild	0.000
b ; defoe-robinson-103	0.000
d ; callwild	0.0602
e ; defoe-robinson-103	0.1505

Exemple simple

Afin de réaliser des tests, l'exemple du corpus ci-contre a été traité. L'ensemble des étapes des MapReduce correspondants au calcul des tf-idf de ce réseau sont explicitées dans les tableaux de la partie *Approche théorique.*

(Mot ; docID)	TF-IDF
a ; callwild	0.000
a ; defoe-robinson-103	0.000
b ; callwild	0.000
b ; defoe-robinson-103	0.000
d ; callwild	0.0602
e ; defoe-robinson-103	0.1505

Les résultats obtenus sont donnés dans le tableau ci-contre. Ils peuvent être retrouvés dans le fichier reponse/TFIDF_test/.
On observe que les mots a et b ne sont pas spécifiques à un certain document, d'où le tf-idf nul. Au contraire, le mot e est très spécifique au document callwild parce qu'il y est présent 2 fois alors que non présent dans le reste du corpus, c'est pourquoi il a un grand tf-idf.

Exemple concret

Il s'agit maintenant de déterminer l'ensemble des tfidf pour un corpus constitué de 2 textes très longs. Les fichiers txt obtenus à chaque MapReduce sont disponibles dans le fichier reponses/TFIDF/.

Les 20 mots avec les plus grands scores TF-IDF sont présentés dans le tableau ci-contre.

On y retrouve sans grande surprise les noms des personnages de CallWild.

Classement	Mot ; doc	TF-IDF
1	buck ; callwild	0,00335
2	dogs ; callwild	0,00109
3	thornton ; callwild	9,51E-04
4	myself ; defoe-robinson-103	7,21E-04
5	spitz ; callwild	6,06E-04
6	sled ; callwild	5,87E-04
7	francois ; callwild	5,59E-04
8	friday ; defoe-robinson-103	4,57E-04
9	trail ; callwild	3,82E-04
10	john ; callwild	3,73E-04

Question 5.2

Le but de cette question est de déterminer pour un réseau web donné (pages et liens) la probabilité, le PageRank, de se retrouver sur une certaine page web en naviguant aléatoirement de page en page. On utilise trois architectures MapReduce à la suite l'une de l'autre. Pour expliquer les différentes étapes, on prendra les abréviations suivantes :

PF	Page From
PT	Page To
PRF	Page Rank From
PRT	Page Rank To
C	Count Page To From Page From

Approche Théorique

Le premier MapReduce prend en argument les arêtes orientées du graphe du réseau. Le Mapper1 crée en clé des couples (page, pagerank) des pages web et attribue en valeur pour chaque arête le couple (page, pagerank) correspondant. On initialise le pagerank à un divisé par le nombre de nœuds dans le réseau. Le Reducer1 regroupe pour chaque couple, l'ensemble des informations nécessaires pour appliquer la formule du PageRank à une page donnée :

$$PR(A)_{i+1} = 0.15 * PR(A)_i + 0.85 * \sum_{liens \to A} \frac{PR(lien)_i}{nbr\ liens\ sortant\ (lien)}$$

Source :

PF	PT
0	1
0	2
1	0
2	1

PageRankMapper1 :

(PF, PRF)	(PT, PRT)
(0, 0.33)	(1, 0.333)
(0, 0.33)	(2, 0.333)
(1, 0.33)	(0, 0.333)
(2, 0.33)	(1, 0.333)

PageRankReducer1 :

(PF, PRF)	C ; (PT, PRT)
(0, 0.33)	2; (1, 0.333)#(2, 0.333)
(1, 0.33)	1; (0, 0.333)
(2, 0.33)	1; (1, 0.333)

Cependant le pagerank se calcule par itérations, quand on calcule une itération il faut eméttre les informations nécéssaires au calcul du prochain pagerank. Le deuxième Mapper prend donc en argument la sortie du Reducer1 et renvoie en clé la page et trois différents types de valeurs :
- le pagerank précédent
- la liste des liens sortant de la page
- la liste des pagerank entrants de la page avec le nombre de liens sortants associés à la page

Dans le Reducer2, on calcule le nouveau pagerank. La sortie du Reducer2 doit être de la même forme que son entrée pour pouvoir itérer.

Source :

(PF, PRF)	C ; (PT, PRT)
(0, 0.33)	2; (1, 0.33)#(2, 0.33)
(1, 0.33)	1; (0, 0.33)
(2, 0.33)	1; (1, 0.33)

PageRankMapper2 :

Page	PR	Page	PT	Page	PRF/C
1	0.333	1	;0	1	0.333/2
2	0.333	2	;1	2	0.333/1
0	0.333	0	;1;2	0	0.333/2
1	0.333	1	;0	1	0.333/2

PageRankReducer2 :

Page	PR	Page	PT	Page	PRF/C
0	0.333	0	;1;2	0	0.475/1
1	0.192	1	;0	1	0.192/1
2	0.475	2	1	2	0.333/2

Puis on fait un troisième MapReduce, qui permet de faire le Reducer2 plusieurs fois à la suite, le Mapper3 renvoie l'identité.

Source :

Page	PR	Page	PT	Page	PRF/C
0	0.333	0	;1;2	0	0.475/1
1	0.192	1	;0	1	0.192/1
2	0.475	2	1	2	0.333/2

PageRankMapper3 :

PageRankReducer2 :

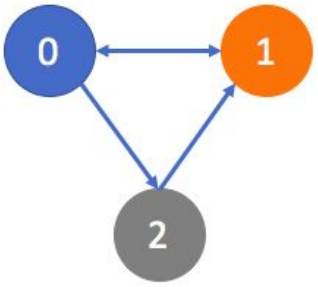
Page	PR	Page	PT	Page	PRF/C
0	0.454	0	;1;2	0	0.376/1
1	0.376	1	;0	1	0.170/1
2	0.170	2	1	2	0.454/2

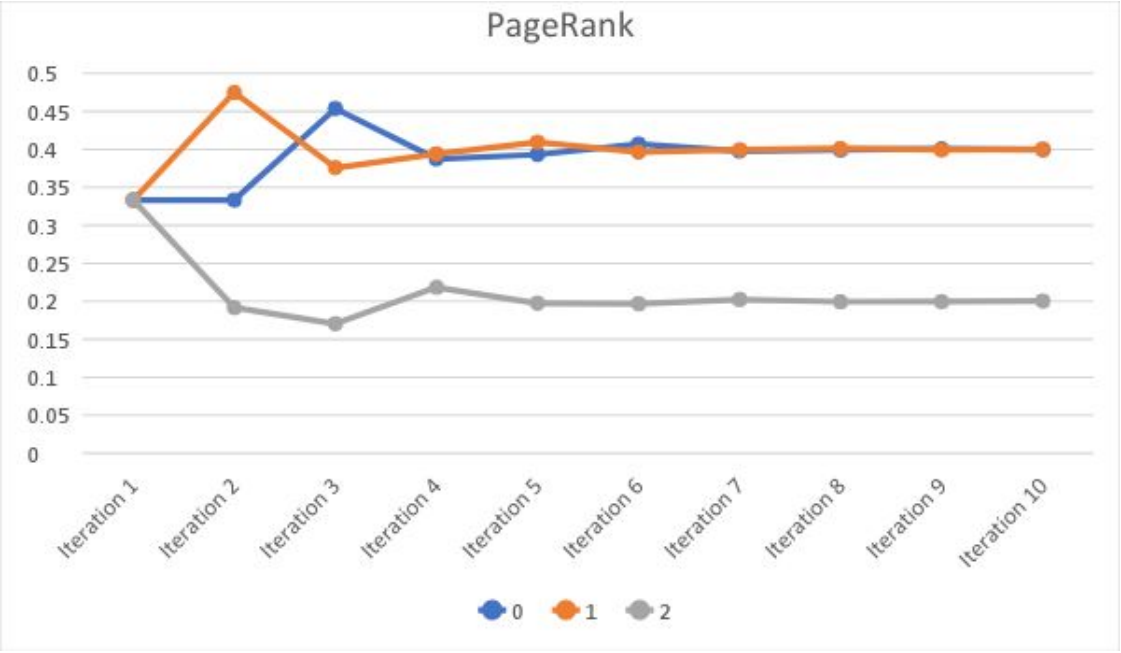
Exemple simple

Afin de réaliser des tests, l'exemple de réseau ci-dessous a été traité. L'ensemble des étapes des MapReduce correspondants au calcul des pagerank de ce réseau sont explicitées dans les tableaux de la partie *Approche théorique.*

Nous avons fait 10 itérations. Les fichiers txt obtenus à chaque fois sont disponibles dans le fichier reponses/PageRankTest. Le graphe ci-dessous présente les résultats obtenus.

On observe que l'algorithme converge rapidement (dès la 4eme itération). De plus, la somme de l'ensemble des pagerank du réseau fait bien 1.





Exemple concret

Il s'agit maintenant de déterminer l'ensemble des pagerank pour un réseau constitué de 75879 pages web et de 508837 liens. Nous avons fait 20 itérations. Les fichiers txt obtenus à chaque fois sont disponibles dans le fichier reponses/PageRankEpinions.

Les résultats obtenus sont donnés dans le tableau ci-contre.

On observe que les 10 plus grandes probabilités sont cohérentes avec une somme de 1.

Logiquement, on s'aperçoit aussi que les sites suivent une loi de Pareto : seulement un dix millièmes des pages représente plus d'un centième des probabilités. Cela serait peut-être encore plus flagrant si on faisait plus d'itérations.

	Page	PageRank
Compte	10	1,67%
Total	75879	100,00%
%	0,013%	1,67%

Améliorations

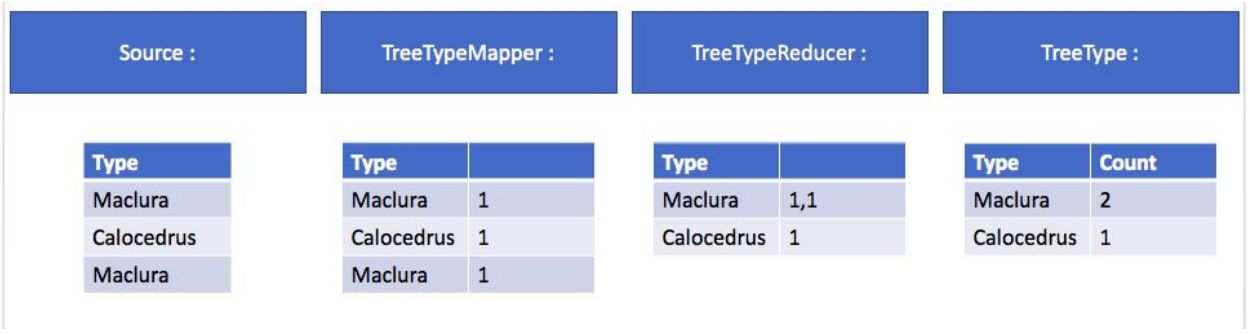
Afin d'améliorer le programme, il serait utile de :

- Mettre une clé de la forme Couple(page, pagerank), on pourrait alors grâce au compareTo() ordonner notre sortie de reducer en fonction des pagerank. Le problème est que nous n'avons pas réussi à définir un nouvel OutputFormat, nous n'avons pas réussi à ce que les clés et valeurs des Reducer soient autre chose qu'un Text
- Nous pourrions aussi ne pas lancer les MapReduce manuellement un nombre fixer de fois mais plutôt les lancer tant que les pagerank ne convergent pas.

Question 5.3

Compute the number of trees by type

Le but de cette question est de compter le nombre d'arbres dans Paris par type. On utilise donc une architecture MapReduce. Le template utilisé est celui fourni dans CompterLigneFile. La seule différence est que le mapper lit seulement la 2eme colonne du fichier source et non pas la ligne en entier. La clé de chaque ligne est le type de l'arbre auquel on attribue une valeur 1. La logique du MapReduce est expliquée ci-dessous :

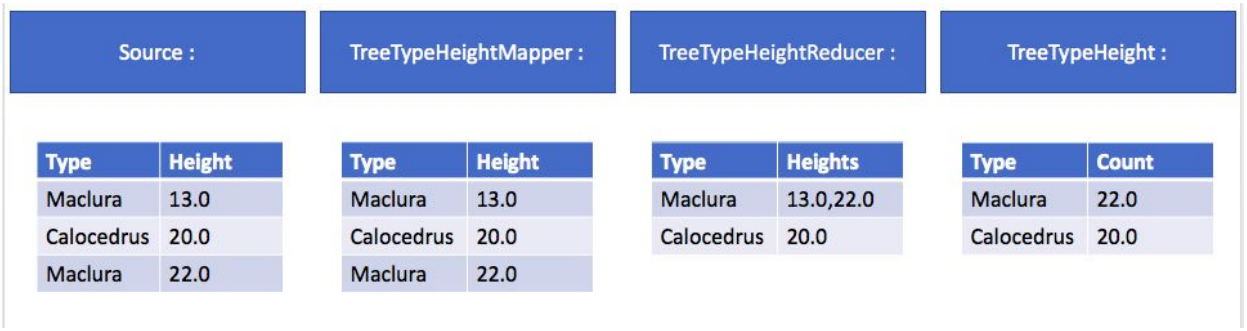


Le fichier obtenu est TreeType.txt, il peut être trouvé dans le fichier reponses/TreeType.

Compute the height of the highest tree of each type

Le but de cette question est de connaitre la taille du plus grand arbre de chaque type. On utilise aussi une architecture MapReduce. Le modèle utilisé est celui fournit dans l'exercice ci-dessus. Il y a deux différences majeures. D'abord, le type de l'arbre est toujours la clé mais la valeur attribuée n'est plus 1 mais la taille de l'arbre correspondant. C'est donc un FloatWritable et non pas un IntWritable. Puis au lieu de sommer l'ensemble des 1 obtenus

pour un type d'arbre, on choisit la hauteur maximale. La logique du MapReduce est expliquée ci-dessous :



Source :

Type	Height
Maclura	13.0
Calocedrus	20.0
Maclura	22.0

TreeTypeHeightMapper :

Type	Height
Maclura	13.0
Calocedrus	20.0
Maclura	22.0

TreeTypeHeightReducer :

Type	Heights
Maclura	13.0,22.0
Calocedrus	20.0

TreeTypeHeight :

Type	Count
Maclura	22.0
Calocedrus	20.0

Le fichier obtenu est TreeTypeHeight.txt, il peut être trouvé dans le fichier reponses/TreeTypeHeight.