Application Twitter

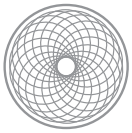
Préparé pour:

- Ioan Todinca, Encadrant
- Antony Perez, Encadrant

Préparé par:

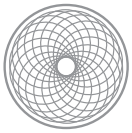
- Hubert Simon
- Mchaakah Alaa
- Boulealf Soufiane

18 mars 2013, Orléans



Entreprise

Table des matières



Groupe14
STIC

Introduction

Afin de pouvoir valider notre L3, nous (Groupe14) avons un module intitulé projet, qui traite le problème des «capitalistes sociaux

sur Twitter», le but étant de pouvoir détecter ces individus inscrits sur twitter à partir des informations de leurs Followers et Followees.

D'après la description du sujet, les capitalistes sociaux adoptes, deux techniques différentes, et donc peuvent se diviser en deux parties distinctes :

- **«FOLLOW ME I FOLLOW YOU» :** ou FMIFY, ces individus promettent à leurs ''Followees" (personnes qui les suivent), de les suivre en retour. Ils ont clairement plus de Followees que de Followers. Donc le ratio de Followers et plus important que celui de Followees.
- **«I FOLLOW YOU FOLLOW ME» :** IFYFM, à l'inverse de l'autre catégorie, ces individus suivent des utilisateurs et espère qu'ils les suivent en retour. Donc le ratio de Followees et plus important que celui de Followers.

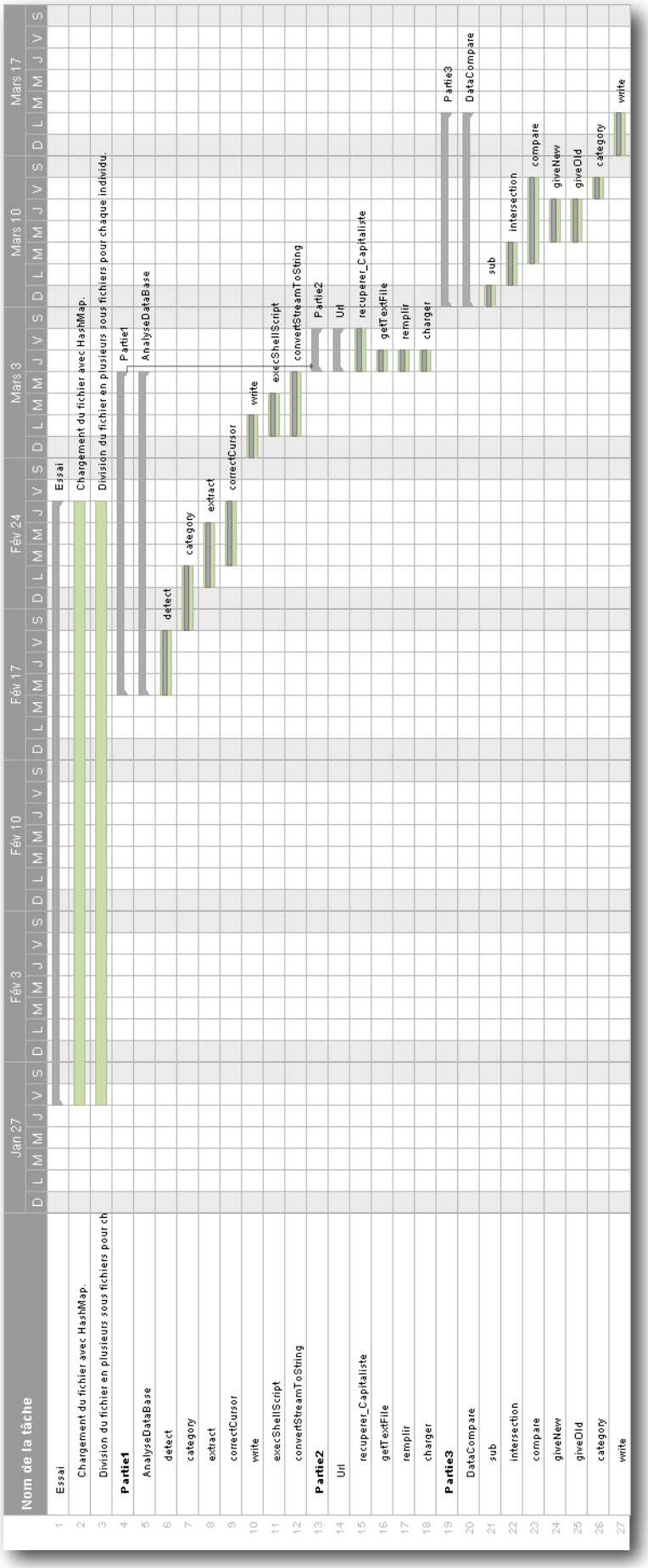
Les fichiers que nous traitons tout au long de ce rapport sont de la forme suivante :

Source1	Destination1
Source2	Destination2
Source3	Destination3
....	....

Où «source» et «destination» représente l'ID d'individus et symbolise la relation : «source» follow «destination»,
donc «source» est un Follower de «destination» et «destination» est un Followee de «source».

Il faudra noter avant tout que nous avons travaillé avec Eclipse, sous Mac OS X , avec le langage de programmation Java.

Organisation


Nom de la tâche
Jan 27
Fév 3
Fév 10
Fév 17
Fév 24
Mars 3
Mars 10
Mars 17
1 Essai
2 Chargement du fichier avec HashMap.
3 Division du fichier en plusieurs sous fichiers pour ch
4 Partie1
5 AnalyseDataBase
6 detect
7 category
8 extract
9 correctCursor
10 write
11 execShellScript
12 convertStreamToString
13 Partie2
14 Url
15 recuperer_Capitaliste
16 getTextFile
17 remplir
18 charger
19 Partie3
20 DataCompare
21 sub
22 intersection
23 compare
24 giveNew
25 giveOld
26 category
27 write
Essai
Chargement du fichier avec HashMap.
Division du fichier en plusieurs sous fichiers pour chaque individu.
Partie1
AnalyseDataBase
detect
category
extract
correctCursor
write
execShellScript
convertStreamToString
Partie2
Url
recuperer_Capitaliste
getTextFile
remplir
charger
Partie3
DataCompare
sub
intersection
compare
giveNew
giveOld
category
write

La partie d'essai consistait à prendre plusieurs pistes en parallèle pour l'algorithme de détection, et tester pour pouvoir juger les différences, les pistes qui ont était prise n'ont rien donner, les algorithmes mettaient des heures pour s'exécuter.

Après avoir trouver une piste (celle du changement des colonnes des fichiers cf plus bas) nous avons lancé en parallèle le développement de la partie 2.

Le développement de la dernière partie n'a pas pris beaucoup de temps vu qu'on a repris les mêmes algorithmes que la partie 1.

Part.1 : Détection de Capitaliste

Nous devons extraire les capitalistes sociaux à partir d'un fichier «grapheProjet.txt» contenant des relations entre les individus inscris sur Twitter. Le fichier n'est pas trié.

Classe AnalyseData

Nous avons crée une classe utilitaire, qui permet d'avoir des fonctions «static» permettant de traiter les fichiers et les informations des Followers et Followeens. (Pour plus d'informations cf. javadoc.)

Préparation de fichiers

Grâce à un script Shell, nous trions le fichier «grapheProjet.txt», ensuite nous inversons les deux colonnes avec la commande awk, et nous re-trions le fichier, ainsi nous trouverons :

- GrapheProjet : la deuxième colonne correspond au Folloween de la première colonne, donc notre fichier de Folloween
- Fichier inverse : la deuxième colonne correspond au Followers de la première colonne, donc notre fichier de Followers

L'exécution du Script se fait avec la fonction ExecShellScript, qui permet de créer un nouveau Processus externe grâce à la Class ProcessBuilder. (pour plus d'informations voir le code source (commenté) et la javadoc.)

Algorithme de Détection

Cet algorithme permet de charger les informations de chaque identifiant figurant dans le fichier de Folloween, ainsi pouvoir effectuer le traitement nécessaire et déterminer si l'individu est un capitaliste (cf. Introduction).
Le fichier de Folloween, étant pris comme référence, implique que les individus qui ne figurent pas dans ce fichier et qui existent dans le fichier de Followers, sont automatiquement ignorés. (l'intersection est égal 0, on traite que les ID dont l'intersection est supérieure à 1000).

Avant de commencer la lecture du programme "detect" il faut noter les choses suivantes : Nous utilisons :

- Un BufferedReader pour lire dans un fichier, il faut noter que pendant tout le traitement de chaque fichier, nous lisons chaque ligne une seule fois.
- Plusieurs HashSet qui contiennent les informations suivantes :
  - Les Followers de l'ID traité
  - Les Folloween de l'ID traité
  - Les individus de type FMIFY
  - Les individus de type IFMFY
  - Les individus des deux types
- Une fonction qui permet d'extraire les informations correspondant à un utilisateur donné, il s'agit de la fonction extraire (cf. javadoc). Pendant le traitement, il faut noter que si l'utilisateur donné est inférieur à l'utilisateur lu, le programme saute au prochain ID donc avec la fonction nextID (cf. javadoc). Aussi que cette fonction renvoi la prochaine ligne donc le prochain user, sinon nulle.
- Une fonction qui permet de classifier l'utilisateur traité :
  - si l'intersection de Folloween et Followers est supérieur à 1000.
  - si le ratio de Followers > 0.8 donc c'est un FMIFY
  - si le ratio de Folloween > 0.8 donc c'est un IFYFM
  - si on a les deux c'est un both
  - sinon on traite pas

  il s'agit de la fonction category (cf. javadoc).
- Les scripts Shell sont exécutés avec ProcessBuilder, les fichiers des scripts se trouvant dans la racine du projet (cf. javadoc).

Tant qu'on a pas finit de lire le fichier de Folloween (grapheProjet) {

- On lit une ligne du fichier Folloween et on extrait les informations de l'ID traité (c'est -à-dire la première colonne), on ajoute ensuite son Folloween (2ème colonne) à la liste de Folloween.
- On extrait du fichier de Folloween (grapheProjet), Followers (fichier inverse), les information de l'ID (premiere colonne).
- On analyse la catégorie de l'ID.
- Vérification du cursor des Buffers, dans le cas où le cursor est déréglé nous utilisons la fonction CorrectCurosr pour corriger le dérèglement, pour qu'il commence au même ID.

}

Écriture des Résultats (c-à-d les HashSet des trois catégories) dans le fichier de sortie, et trie de ce fichier un script shell. (Pour plus détailles voir le code source (commenté), et javadoc.)

Partie 2 : Mise à jour du graphe Twitter

Lecture

Chargement des requêtes via l'API Tweeter

Nous avons décidé une fois de plus d'exécuter cette tâche dans l'environnement java. Pour cela, nous nous sommes tout d'abord intéressé à l'API de Tweeter, suite à quelques recherches nous avons décidé d'utiliser la REST API v1 et toutes ses ressources. L'utilisation que nous en faisons est simple, elle consiste à exécuter les requêtes suivantes comme URL : **https://api.twitter.com/1/type/ids.json?user_id=tweetos** où *type* correspond au type choisi pour la requête (Followers ou Following) et *tweetos* correspond à l'id de l'utilisateur dont on souhaite les informations. Pour réaliser cela en java nous avons simplement écrit une méthode permettant de générer une URL sur internet et de charger son contenu en type String.

Récupération des données

Ayant récupérer le contenu des requêtes via la méthode décrite précedement ils nous faut maintenant les analyser et extraire seulement les donneés qui nous intéressent. Le contenu étant de type String nous avons simplement supprimé les caractéres inutiles à l'aide des méthodes *replace* et *replaceAll* de la Class String. Ensuite nous récupérons la valeur *next_cursor* du contenu qui nous permet de savoir si la liste d'utilisateurs obtenue est compléte ou non. Car chaque requête nous donne une liste d'utilisateurs limitée à 5 000 et comme nous en voulons au maximum 25 000 par Capitalistes nous devons exécuter au maximum 5 requêtes sur le même utilisateur. Cette valeur *next_cursor* nous permet donc de savoir où reprendre les prochaines requêtes pour obtenir la suite de la liste des followers (ou des following) de cet utilisateur. Nous exécuterons donc une nouvelle requête en spécifiant le curseur :

https://api.twitter.com/1/type/ids.json?user_id=tweetos&cursor=next_cursor

Et enfin pour chaque requête, après avoir récupéré le *next_cursor*, nous récupérons à l'aide d'une boucle la liste d'utilisateurs situé entre crochets à l'aide de la méthode *sc.nextBigInteger( )* où *sc* est un Scanner déclaré sur notre contenu.

Écriture

L'écriture est tout d'abord exécutée dans deux fichiers distincts : Followers14.txt et Followees14.txt.

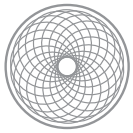
Pour effectuer cette écriture, nous passons tout d'abord par notre algorithme qui exécute une boucle sur la liste des Capitalistes de notre intervalle. Cette liste est obtenue grâce à une sélection faite sur le fichier idsCapitalistesSociaux.txt. Et à chaque fois que l'on récupère les données d'une requête, grâce aux méthodes de la lecture, on écrit directement dans le fichier de sorti correspondant sous le format : **idCapSoc follower** ou **idCapSoc followee** suivant le fichier.

Part.3 : Comparaison

Après avoir récupéré les informations actuelles des capitalistes sociaux détecté en 2010 sur twitter, nous devons comparer ces informations, avec celles récupérés en 2010 (grapheProjet.txt).

Les Encadrants ont pu créer un fichier de Followers et de Folloween à partir de la récupération de chaque groupe.

Remarques : Avant toute opération sur les fichiers, nous les trions grâce au script «sort.sh» qui exécute la commande «sort -n» .



Groupe14
STIC

Objectif

Comparer les fichiers de 2010 et 2013 implique la récupération des informations suivantes :

- Le nouveau type de l'utilisateur, incluant le fait que ces ID étaient tous des capitalistes sociaux en 2010 et peuvent donc avoir décroché entre temps.
- Le nombre de nouveau Followers, et Followees.
- Le nombre d'anciens Followers, Followees qui n'existe plus actuellement dans la liste des individus.

Algorithme de Comparaison

Nous avons créer une Class DataCompare utilitaire, utilisant des fonctions de AnalyseData. Voici les principales idées visant à répondre aux objectifs cité au dessus :

- Nous avons déjà une fonction qui récupère le type dans la class AnalyseData (category cf. javadoc), nous allons juste changer le type de retour de cette fonction et reprendre le même traitement.
- Nous reprendrons la même idée de l'algorithme de détection de la 1ère partie pour pouvoir charger les informations d'un individu dans les 4 fichiers (en utilisant la fonction extract). Ensuite il ne reste plus qu'à effectuer la comparaison entre les nouveaux et les anciens éléments.
- Une fonction qui permet de faire la soustraction entre deux espaces (cf. la fonction sub).
  Soit "A" l'ensemble de Followees de 2010 et B l'ensemble de Followees de 2013 (même chose pour les Followers) :
  - A - B : Anciens éléments (cf. giveOld javadoc).
  - B - A : Nouveaux éléments (cf. giveNew javadoc).

Notre algorithme parcourt les 4 fichiers en même temps, en prenant comme référence cette fois si le fichier de Followees de 2013. Il faut noter que les informations apparaissant dans le fichier de sortie, ne concerne que les individus du fichier Followees de 2013, ceux qui n'apparaissent que dans le fichier de Followers 2013 uniquement sont ignorés, car ils ne sont plus des capitalistes sociaux (intersection 0).
Pour économiser la mémoire, nous faisons l'extraction et la comparaison de Followees en premier. Ensuite, nous supprimons l'espace de Followees2010 (qui ne nous serre plus à grand chose), avant d'extraire les Followers des deux années et refaire une comparaison.

Test, Difficulté ...

Afin de pouvoir optimiser le temps d'exécution de l'algorithme de détection de capitalistes nous avons effectué plusieurs changements :

- Utilisation du BufferedReader qui est nettement plus rapide que le FileInputStream, Pour la lecture d'un même fichier, FileInputStream s'exécute à l'ordre de 4 minutes, alors que BufferedReader s'exécute en 1minute.

- Pour extraire les informations de chaque ligne lut du fichier, nous utilisions un Scanner. Après réflexion, nous avons remplacé le Scanner par une expression régulière (cf. String :: Split), cela nous a permis de gagner 30 minutes avec le fichier «grapheProjet».

Conclusion

Les programmes de détection et de comparaison fonctionnent correctement. Les temps d'exécutions sur les fichiers fournit par les encadrants sont les suivants:

- Détection :

    - Préparation du fichier : 4 min.

    - Détection des capitalistes : 8 min.

    - Nous pouvons détecter 12500 capitalistes dans ce fichier.

- Comparaison :

    - Le trie des 4 fichiers avec commande «sort» : 2 à 4 min par fichier.

    - La lecture et comparaison de données : 5 à 7 min.

Grâce a ce projet nous avons pu mettre en oeuvre différentes capacités, notamment l'utilisation de subversion, ssh, shell, sous différentes système d'exploitation, Mac OS X, Ubuntu, Debian... Prendre le temps de bien réfléchir à des méthode, exigence au niveau de la qualité du code.