# Projet ”Quel bazar !”

## Algorithmique et structures de données 3

**Bois Cédric, groupe 501A**

**01/12/2014**

# Sommaire

# Introduction

Le but de ce projet est de faire des groupes pages selon leur contenu. En effet nous disposons d'un dictionnaire, si une page a un nombre donné de mots appartenant au dictionnaire, en commun avec une autre page, alors ces pages font parti du même groupe. Nous allons voir par la suite comment ce problème a été traité.

# Compilation et exécution du programme

Le projet compile et s'exécute grâce à un script nommé compilExec.sh. Il est important de préciser, que pour un soucis d'optimisation, les pages doivent être numéroté de 1 à n[1], et doivent être ordonnées en ordre croissant dans la ligne de commande.

## Compilation

Dans le dossier du projet, la compilation s'effectue avec la ligne de commande suivante :

Javac src/ClasseUnion.java src/AVL.java src/Main.java src/Livre.java Dictionnaire.java src/Page.java src/Mot.java –d bin/

## Exécution

Pour exécuter le programme, il faut se placer dans le dossier bin, puis rentrer une ligne de commande sur le modèle suivant :

java Main k FichierDictionnaire FichierPage1 FichierPage2 …

[1] n étant la nombre de pages

# Les structures de données

## AVL

L'implémentation de l'AVL contient les méthodes d'ajout et de suppression, d'équilibrage, du calcul de hauteur, de renvoie des éléments et de recherche d'un élément.

La fonction d'équilibrage agit uniquement sur le nœud qui vient d'être modifié. Elle regarde la balance de ce nœud. Si elle est supérieure à 1, on regarde la balance du fils droit. Si elle est positive, on effectue une rotation à gauche, si elle est négative, ont effectue une double rotation à gauche. Si la valeur de la balance du nœud courant est inferieur à -1, ont fait symétriquement de même sur le fils gauche. Cette méthode agit de manière linéaire, mis à part ses appels à la fonction « balance » qui opère en O(log(n)). La fonction d'équilibrage a donc une complexité en O(log(n)). Cette fonction renvoie la nouvelle racine de l'arbre.

La fonction d'ajout cherche l'emplacement où ajouter l'élément, ceci s'effectue en une complexité de log(n), puis équilibre le nœud qui a ajouté l'élément, toujours en log(n). La complexité de cette méthode est donc en O(log(n)).

La méthode de suppression fonctionne de la même manière que la méthode d'ajout, mis à part qu'elle recherche le maximum de son sous arbre gauche, ou le minimum de son sous arbre droit pour remplacer l'élément supprimé, ceci en log(n). Cette fonction a donc aussi une complexité en O(log(n)).

La méthode qui renvoie la liste des éléments contenus dans l'arbre fait un parcours symétrique. Elle a donc une complexité en O(n), c'est pourquoi elle est utilisée le moins possible dans le traitement des pages du livre.

## Classe-Union

La classe-union a pour attribut un arrayList de Pages, ainsi qu'un tableau de pages représentant les classes éléments présents dans la liste. Comme expliqué précédemment, les pages doivent être rentrées dans l'ordre car la classe de l'élément situé à l'index i dans la liste a sa classe située aussi à l'index i dans le tableau. De plus, les pages doivent être numérotés de 1 à n car la classe d'une page, se trouve donc dans le tableau à l'index id[2]-1. La classe union fonctionne de cette manière pour éviter d'avoir à utiliser la fonction indexOf de l'ArrayList qui a une complexité en O(n).

La classe-union a une méthode union qui permet d'unir deux Pages p1, et p2. Pour ce faire, elle va regarder la classe de p2, si celle-ci est null ou égale à p2, elle va regarder la classe de p1, si elle est null, la classe de p2 devient p1, sinon, la classe de p2 devient la classe de p1. Si l classe de p2 est différente de null ou d'elle-même, alors on uni p1 avec la classe de p2.

Une fonction classe permet de renvoyer une page qui sert d'identifiant à un groupe de page. Si la classe de cette page est null ou égale à elle-même, c'est soit qu'elle sert d'identifiant pour son groupe, soit qu'elle est seule, c'est donc elle que l'on va retourner. Sinon, on fait un appel récursif pour retourner la classe de sa classe.

[2] Identifiant de la page

# Les différentes classes

## Mot

La classe Mot contient un attribut, qui est le mot en chaine de caractères. Cette classe implémente l'interface Comparable, elle contient donc la méthode CompareTo qui permet de comparer des mots, ainsi de créer un ordre entre eux.

## Page

Une page contient k[3], un AVL de Mots, ainsi qu'un identifiant en entier. Le constructeur prend le chemin du fichier texte en paramètre, et le dictionnaire. Il scanne chaque mot de la page. Les caractères de ponctuation étant situés en fin de mot sont supprimés, puis si le mot courant est présent dans le dictionnaire, on l'ajoute au contenu de la page, sinon, il n'est pas retenu car il n'est pas indispensable dans le traitement de la page par la suite. La classe contient aussi une méthode listeMots qui utilise la fonction listeElts de l'AVL. Une méthode unir prend en paramètre une page et renvoie un booléen. Cette fonction permet de savoir si les deux pages peuvent être unies. Elle récupère donc la liste des mots de la page passer en paramètre, et va regarder le nombre de mots qu'elles ont en commun. Si ce nombre est supérieur ou égal à k, la fonction renvoie vrai, sinon, faux. Cette fonction a donc une complexité en O(n*log(m)), avec n et m le nombre de mot pour chaque page.

## Livre

Le Livre a en attribut, une classe-union de pages,  et le dictionnaire. Le constructeur prend la liste des arguments passés au programme, en paramètre. Il créée le dictionnaire et toutes les pages.

Le livre contient une méthode unirPages qui permet de réunir les pages par chapitre. Pour chaque page, elle teste si l'union est possible avec la méthode de la classe Page, avec chaque page ayant un identifiant supérieur au sien. Si la méthode unir renvoie vrai, elle va alors appeler la méthode union de la classe-union. Cette méthode est peu efficace puisqu'elle opère en $O(n^2)$.

[3] Nombre de mot communs qu'il faut pour que 2 pages puissent être réunies dans un chapitre

## Jeux de données

On teste notre programme avec la liste des Pokémons de la première génération, organisés par type. On met dans le dictionnaire la liste des Pokémons appartenant à deux types différents. Voici un tableau montrant le nombre de Pokémons appartenant à deux types (dans l'ordre des numéros de pages).

| | plante | feu | eau | insecte | poison | normal | sol | roche | psy | glace | vol | électrique | dragon |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| plante | | | | 2 | 9 | | | | 2 | | | | |
| feu | | | | | | | | | | | 2 | | |
| eau | | | | | | | | 4 | 3 | 3 | 1 | | |
| insecte | 2 | | | | 5 | | | | | | 2 | | |
| poison | 9 | | | 5 | | | 2 | | | | 2 | | |
| normal | | | | | | | | | | | 9 | | |
| sol | | | | | 2 | | | 6 | | | | | |
| roche | | | 4 | | | | 6 | | | | 1 | | |
| psy | 2 | | 3 | | | | | | | 1 | | | |
| glace | | | 3 | | | | | | 1 | | 1 | | |
| vol | | 2 | 1 | 2 | 2 | 9 | | 1 | | 1 | | 1 | 1 |
| électrique | | | | | | | | | | | 1 | | |
| dragon | | | | | | | | | | | 1 | | |

D'après le tableau, avec k=4, on devrait obtenir les résultats suivants :

| | |
|---|---|
| groupe 1 | plante(1), poison(5), insecte(4) |
| groupe 2 | feu(2) |
| groupe 3 | eau(3), roche(8), sol(7) |
| groupe 4 | normal(6), vol(11) |
| groupe 5 | psy(9) |
| groupe 6 | glace(10) |
| groupe 7 | électrique(12) |

| | |
|---|---|
| groupe 8 | dragon(13) |

Le programme nous renvoie ceci :

page :1 , chapitre : 4
page :2 , chapitre : 2
page :3 , chapitre : 7
page :4 , chapitre : 4
page :5 , chapitre : 4
page :6 , chapitre : 6
page :7 , chapitre : 7
page :8 , chapitre : 7
page :9 , chapitre : 9
page :10 , chapitre : 10
page :11 , chapitre : 6
page :12 , chapitre : 12
page :13 , chapitre : 13

Les résultats concordent bien avec ce qui était prévu.

## Problèmes rencontrés

J'ai eut des soucis, plus liés à la compréhension du sujet,  qu'à son implémentation. En effet, lors de la présentation de mon travail, celui-ci ne correspondait pas à ce qui était attendu, puisque mon programme groupait les pages en chapitres, en fonction des mots clés contenu dans le chapitre.

## Conclusion

Pour conclure ce rapport, il n'aura pas été facile de faire ce programme de manière efficace, à cause d'une mauvaise lecture du sujet, et donc par manque de temps. Par ailleurs j'ai tout de même trouvé le sujet intéressant, car il permet concrétiser l'utilisation de structures de données vu en cours en les implémentant.