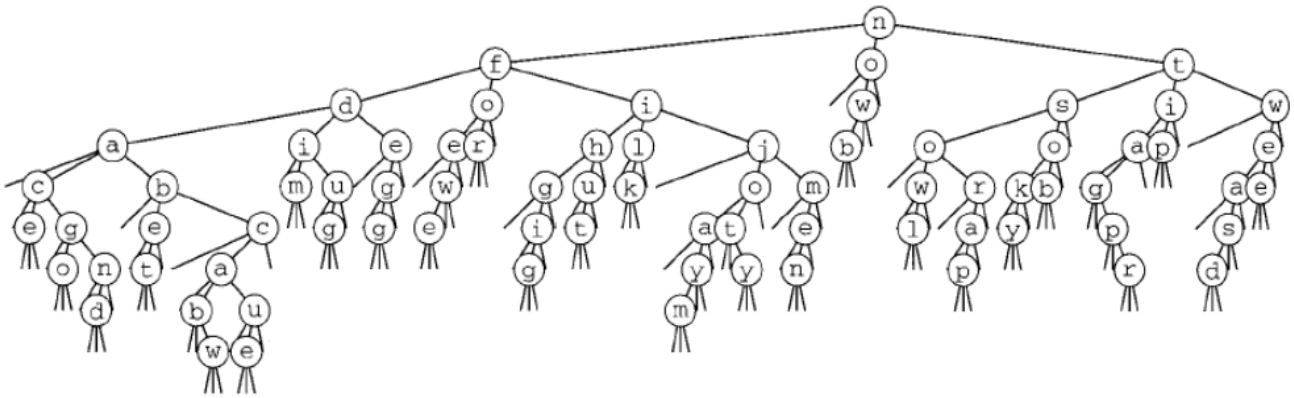
Projet d'Algorithmique Avancée
Les Tries



Lima Gorito Félix
Margarido Julien
Etudiants en M1 STL
UPMC - 2015

Dans le cadre de l'unité d'enseignement d'Algorithmique Avancée (AlgAv – 4I500), nous avons eu à réaliser un projet portant sur les structures de tries : les tries "Hybrides" et les tries "de la Briandais" (aussi appelé « Arbre de la Briandais »).

Pour réaliser ce projet, nous avons choisi d'utiliser le langage Java. Celui-ci étant un langage orienté objet, parfaitement adapté pour l'implémentation et l'étude de nos deux structures. De plus, les deux tries ayant des méthodes similaires à implémenter, Java nous offre la possibilité de mettre une couche d'abstraction, à l'aide d'interfaces, dans le but d'avoir un projet facilement maintenable et réutilisable.

Sommaire

1. Structures

Pour l'implémentation des deux structures de tries, nous avons choisi de créer une interface ITrie. Celle-ci déclare la signature des 8 fonctions demandées dans l'énoncé du projet, plus quelques autres que nous avons trouvées pertinentes :

```
void insererMot(String mot)
void insererPhrase(String phrase)
boolean rechercherMot(String mot)
int comptageMots()
int comptageNil()
List<String> listeMots()
int hauteur()
double profondeurMoyenne()
int prefixe(String str)
void suppression(String mot)
IArbre conversion()
```

De cette manière, chaque structure de Trie devra implémenter l'interface ITrie, et possèdera ces fonctions, adaptées à la structure du Trie en question.

a) Les arbres de la Briandais

Nous avons choisi de définir un arbre de la Briandais de cette façon : un nœud (de type ArbreBriandais) possède comme attributs

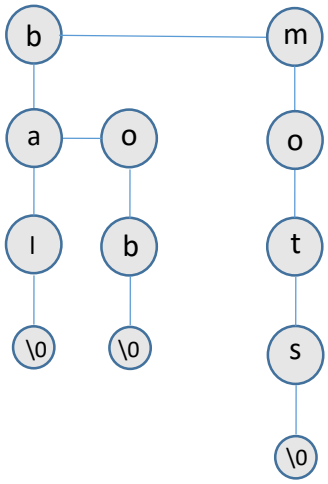
- une clef (caractère ASCII)
- un frère droit
- un fils

} de type ArbreBriandais

Le frère droit possède une clef ayant une valeur supérieure à celle de son frère gauche.
Le fils possède une clef qui indique le caractère qui suit celui du père.

Ainsi, le fils d'un nœud est toujours celui qui a la clef de plus petite valeur parmi ses frères, ce qui implique que certaines fonctions, comme l'ajout d'un mot devront respecter cette règle, et parfois modifier le fils d'un nœud.

Si le frère d'un nœud est "null", c'est que celui-ci n'en possède pas, et réciproquement pour le fils (le nœud de fin de mot ayant, de ce fait, un fils toujours null).
Pour représenter la fin d'un mot nous avons décidé d'utiliser le caractère '\0' de la table ASCII. Ce caractère est la première valeur de la table ASCII, représentant la valeur NULL. Cela facilite ainsi le repérage dans l'arbre, notamment lorsque l'on doit compter le nombre de mots (il suffit de compter le nombre de nœuds ayant '\0' comme clef), ou encore pour la recherche d'un mot.



En plus des 8 fonctions avancées, il nous était demandé qu'un Arbre de la Briandais possède une fonction fusion, qui effectue la fusion de deux Briandais pour n'en faire qu'un seul.
Cet arbre résultant doit bien sûr posséder tous les mots des deux arbres à fusionner, et avoir toutes les caractéristiques d'un Briandais (clef d'un nœud inférieure à la clef de son frère droit, et chaque nœud pointant vers son fils ayant la plus petite valeur s'il existe).

Pour cela, nous avons donc, pour une instance d'un ArbreBriandais (le nom de la classe), que nous pouvons nommer 'ab', la possibilité d'appeler sur ab la fonction fusion qui prend en paramètre un autre ArbreBriandais, nommé 'other'. La fonction va transformer ab en un nouvel arbre contenant ab et other.

Celle-ci fonctionne ainsi :

Si le l'arbre est non null, on compare les 2 clefs (celle de ab et celle de other).

- Si la clef de ab est inférieure à celle de other, alors le frère de ab fusionne avec other.

- Sinon si la clef est supérieure, on crée une copie temporaire de ab, on affecte la clef et le fils de other à ab et on affecte comme frère de ab sa copie, sur laquelle on fusionne le frère droit de other.

- Sinon (dans le cas où les 2 clefs sont les mêmes), on fusionne le fils de ab avec le fils de other, et réciproquement pour les frères.

b) Les Tries Hybrides

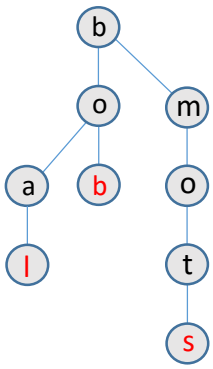
Nous avons décidé d'implémenter les Tries Hybrides ainsi :

- une clef (caractère ASCII)
- un booléen isFinDeMot
- deux frères sup et inf
- un fils eq

} de type TrieHybride

A la différence d'un Briandais, nous avons décidé qu'aucun nœud n'aura comme clef '\0', mais plutôt un booléen 'isFinDeMot' qui indiquera qu'un nœud représente la fin d'un mot. Ceci permet ainsi de réduire le nombre de nœud dans l'arbre, et ainsi de réduire légèrement les complexités des fonctions.

A chaque nœud, le frère 'inf' représentera la lettre qui précède la clef actuelle, et inversement pour le frère 'sup'. Le fils 'eq' est la suite directe du mot (équivalent au fils d'un Briandais).



De même que pour l’arbre de la Briandais, nous avons implémenté les 8 fonctions avancées demandées, mais aussi d’autres fonctions particulières concernant l’équilibrage :

void equilibre()
boolean isEquilibre()
void insererPhrasePuisEquilibre()
void insererMotPuisEquilibre()
void insererListeMotsPuisEquilibre()

Les 3 fonctions d’insertions fonctionnent comme les fonctions de même nom sans l’équilibrage, à la différence où, lorsque l'exécution de l’insertion est terminée, si l’arbre est déséquilibré, la fonction equilibre est appelée.

La fonction isEquilibre retourne vrai si le Trie Hybride est déséquilibré (c’est-à-dire si la différence de hauteur entre le fils inf et le fils sup est supérieure ou égale à 2).

Enfin, la fonction equilibre effectue des rotations droites et gauches sur les parties déséquilibrées de l’arbre (de manière récursive).

2. Complexités

Les fonctions comptageMots, comptageNil, Hauteur et ProfondeurMoyenne ont une complexité en O(n), où n représente le nombre de nœud dans l’arbre. Leurs complexités sont en O(n) car dans le meilleur comme dans le pire cas, il faudra parcourir tous les nœuds de l’arbre pour définir le nombre de mots, le nombre de nœuds null, la hauteur de l’arbre ou encore sa profondeur moyenne.

Les fonctions ListeMots et conversion ont également une complexité en O(n) car pour lister tous les mots d’un arbre ou pour convertir un Briandais en un trie Hybride (ou inversement), on doit obligatoirement parcourir tous les nœuds de l’arbre, par conséquent leurs complexités sont bien en O(n).

Les méthodes Recherche(mots) et Suppression(mots) ont une complexité en O(L), où L représente la longueur mots, car dans le pire cas (pour la recherche ou la suppression d’un mot), on doit parcourir L caractères dans l’arbre.

Pour avoir le nombre de mots préfixé par un mot dans un arbre, on doit d'abord rechercher le mot dans l'arbre puis parcourir le reste de l'arbre pour calculer le nombre de mots qu'il reste dans l'arbre. Par conséquent, la complexité de la méthode prefixe(mot) est en O(L + ln(n)), où L représente la longueur mots et ln(n) le nombre de nœuds restant dans l'arbre.

Pour fusionner deux arbres de la Briandais en un seul on doit dans le pire cas parcourir tous les nœuds de l'arbre 1 et de l'arbre 2 par conséquent la complexité de la fusion(Arbre1, Arbre2) est en O(n1+n2), où n1 représente la nombre de nœud de l'arbre 1 et n2 le nombre de nœud de l'arbre 2.

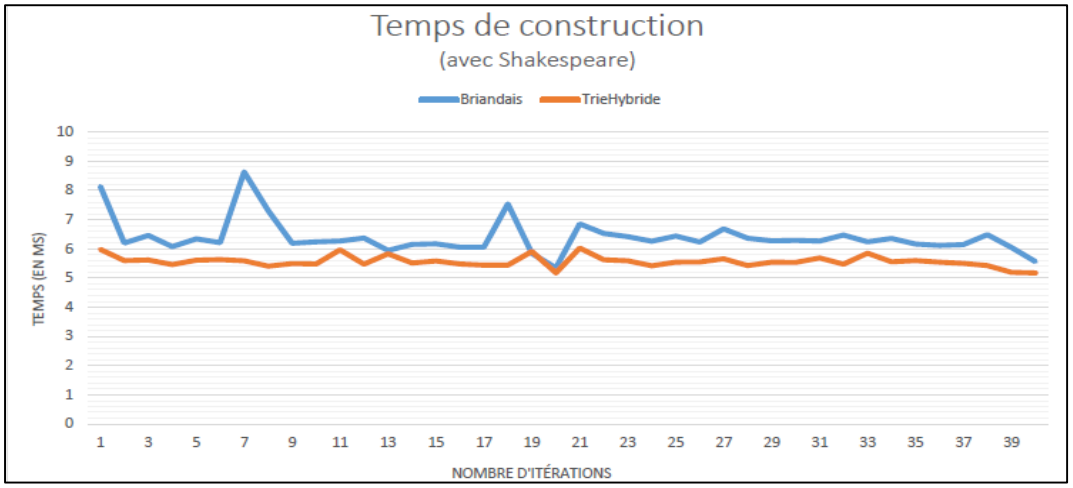
Pour équilibrer un arbre, nous devons, dans tous les cas, parcourir entièrement celui-ci pour vérifier si il est équilibré ou non. Par conséquent, la complexité de la fonction equilibrer est en O(n).

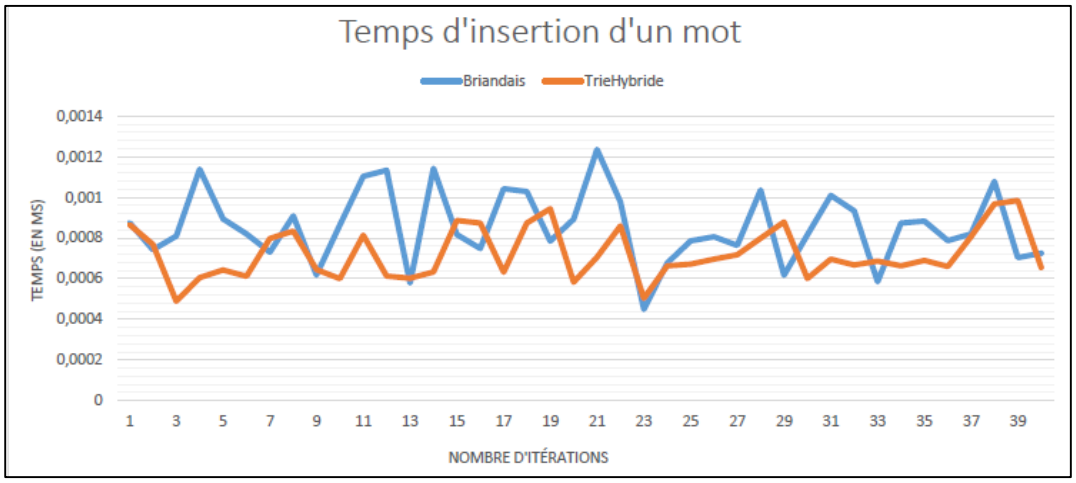
3. Etude des temps

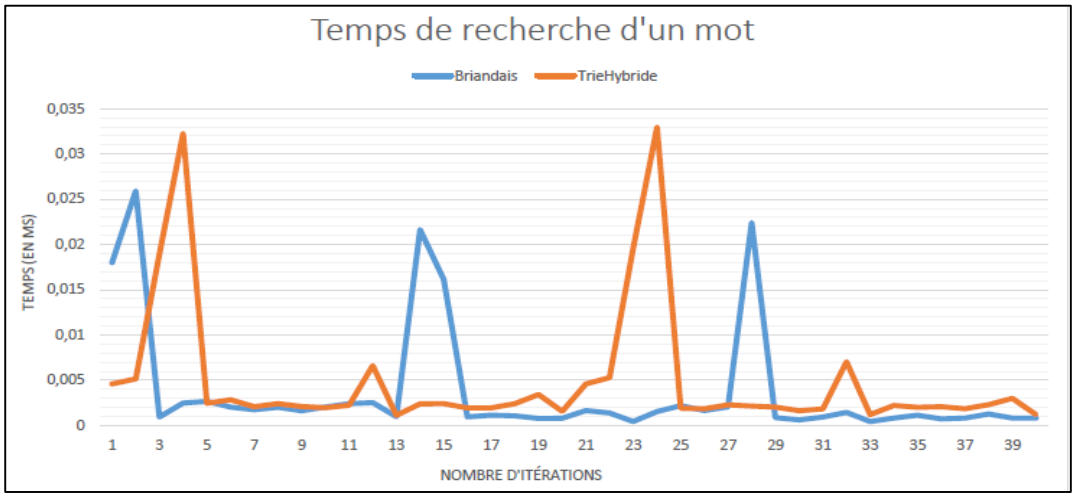
Dans le but d'étudier et comparer les 2 structures, nous avons décidé d'afficher des comparatifs des temps d'exécution de certaines méthodes communes aux 2 structures, à la manière d'un "benchmark".
Ceci a été possible grâce à la fonction System.nanotime, qui donne un temps précis à un temps T. Il suffit de déclencher cette fonction juste avant et après l'exécution d'une méthode, et de prendre la différence entre ces 2 temps, puis de convertir ce temps de nanosecondes à millisecondes, et de l'enregistrer dans un fichier texte.

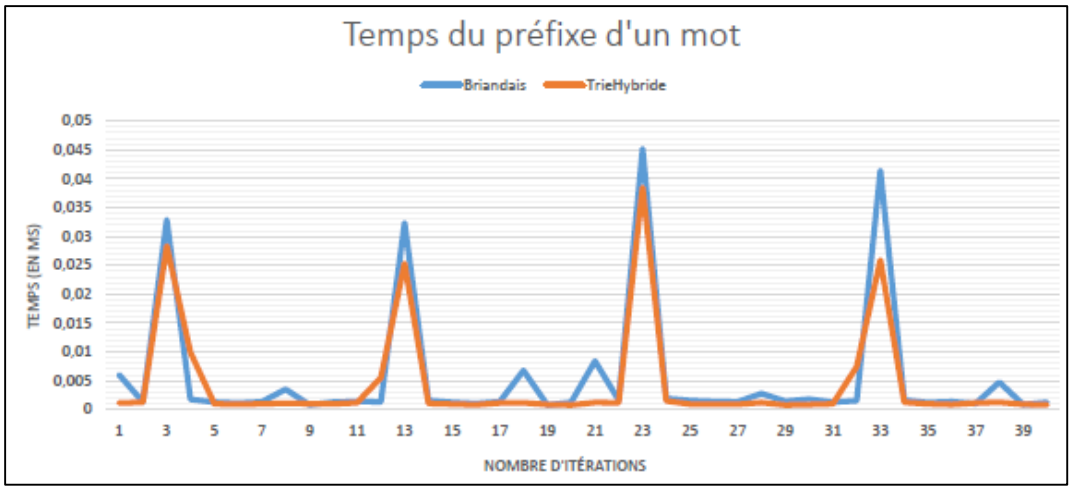
Afin d'obtenir des résultats pertinents, nous répétons 40 fois d'affilés cette opération, en changeant éventuellement le mot en paramètre si c'est une fonction qui en prend un (comme insererMot ou préfixe).
Toutes les fonctions sont effectuées sur un arbre possédant tous les mots de l'œuvre de Shakespeare, dans le but d'étudier ces fonctions dans un environnement proche de l'utilisation normale d'un dictionnaire, par exemple par plusieurs utilisateurs différents, chaqu'un utilisant des mots différents.

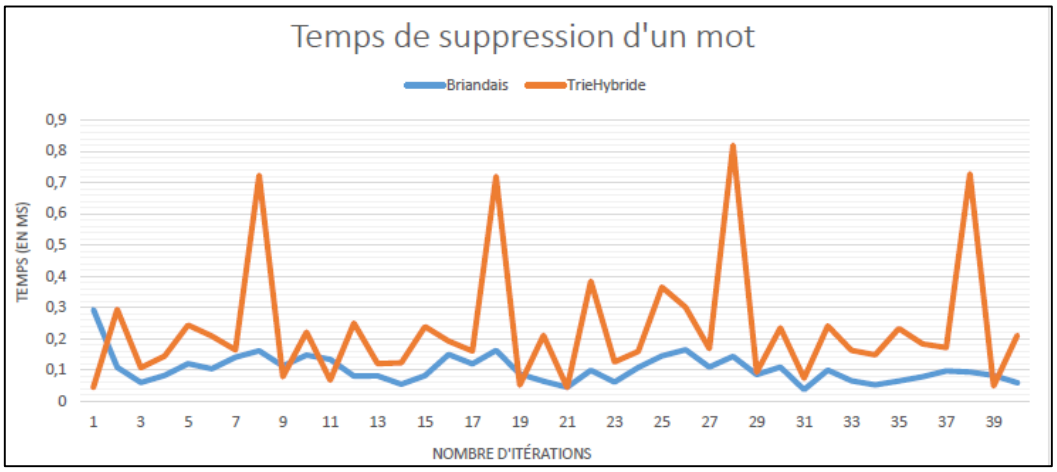
De ce fait, nous avons pu obtenir suffisamment de temps pour comparer les arbres de la Briandais et les Tries Hybrides sur 6 fonctions :

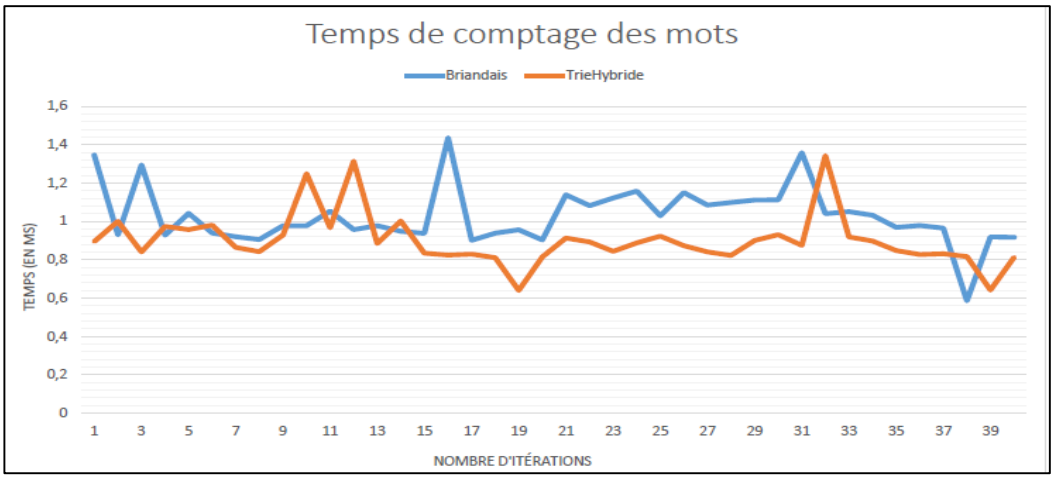



Temps d'insertion d'un mot
Briandais
TrieHybride
TEMPS (EN MS)
0,0014
0,0012
0,001
0,0008
0,0006
0,0004
0,0002
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
NOMBRE D'ITÉRATIONS


Temps de recherche d'un mot
Briandais
TrieHybride
TEMPS (EN MS)
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
NOMBRE D'ITÉRATIONS


Temps du préfixe d'un mot
Briandais
TrieHybride
TEMPS (EN MS)
0,05
0,045
0,04
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
NOMBRE D'ITÉRATIONS





En moyenne, on peut observer que le Briandais est plus performant que le Trie Hybride sur la suppression, légèrement sur la recherche d'un mot, mais dans les autres cas, notamment sur l'insertion d'un mot (la construction avec l'œuvre de W. Shakespeare étant une suite d'insertion de tous les mots contenus dans celle-ci).

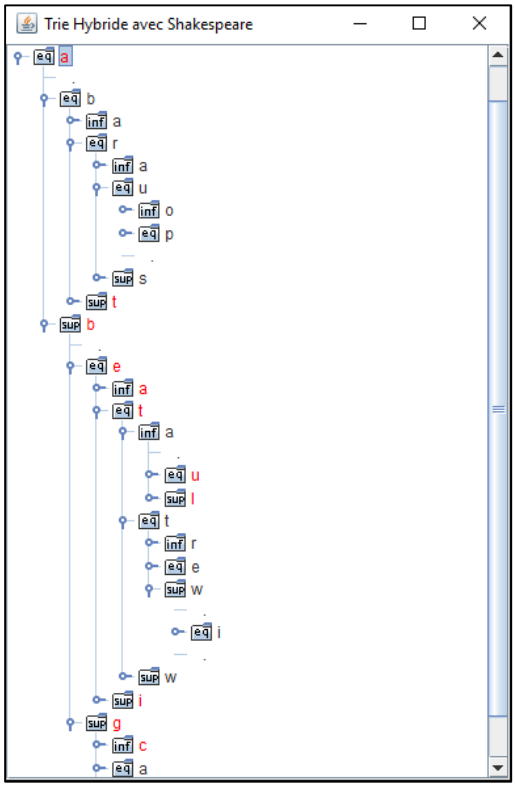
Cependant, ces résultats sont à nuancer avec le fait que ce n'est pas forcément le temps exact d'exécution, car ce temps dépend aussi de la machine sur lequel est lancé le programme (OS, processus en parallèle, matériel, etc…).

4. Affichage des structures

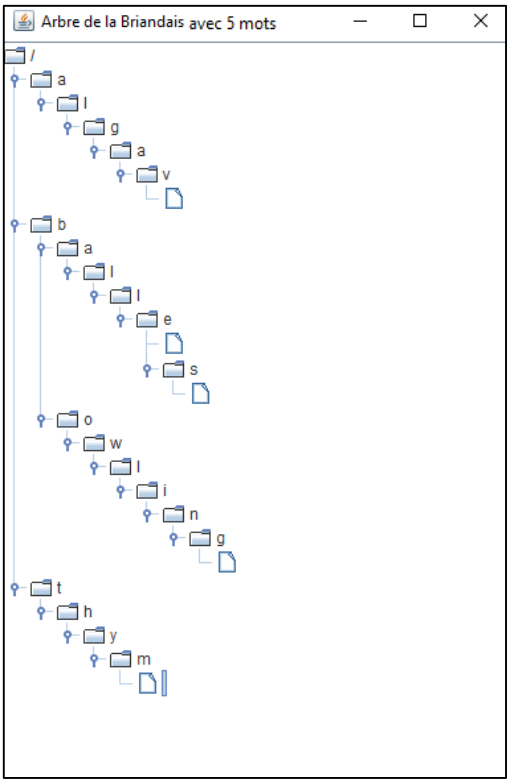
Pour visualiser certaines modifications sur nos structures, nous voulions avoir la possibilité de les afficher, de manière statique ou dynamique.
Nous avons d'abord pensé à l'utilisation de la console, simple pour de petits arbres, mais impossible pour de grandes structures.
Il y avait aussi la possibilité d'afficher l'arbre dans un fichier texte, ou sur un logiciel comme GraphViz, mais ces solutions ne nous paraissaient pas correctement adaptées à notre cas.

En cherchant, nous avons découvert la possibilité d'afficher des arbres à l'aide de la bibliothèque Swing de Java, notamment grâce à la classe JTree.
Cependant, cette classe ne peut pas directement charger un Arbre de la Briandais ou un Trie Hybride comme tel, il a donc fallu créer une classe adaptateur par structure (implémentant l'interface TreeModel), découlant directement du Design Pattern MVC (où le Modèle est la classe de Trie, la Vue le Jtree, et le Contrôleur est l'adaptateur).
Ainsi, nous avons pu obtenir différents affichages :

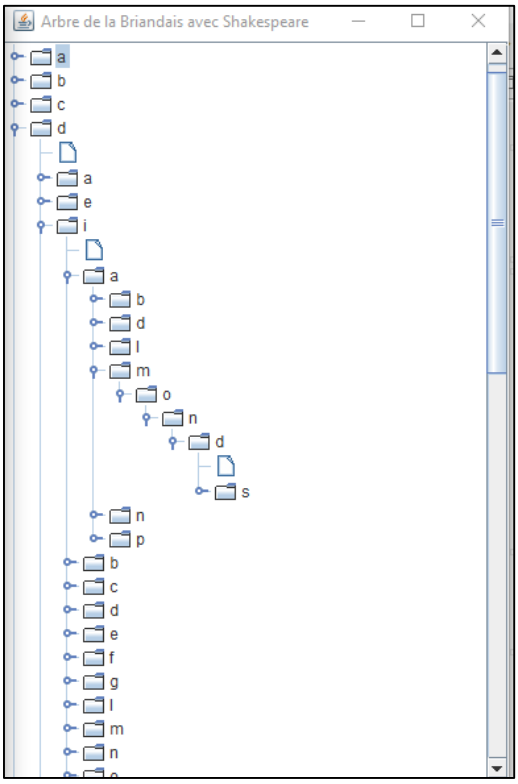
Affichage d'un TrieHybride contenant 5 mots



Affichage d'un TrieHybride contenant l'ensemble des œuvres de W. Shakespeare



Affichage d'un ArbreBriandais contenant 5 mots



Affichage d'un ArbreBriandais contenant l'ensemble des œuvres de W. Shakespeare

Conclusion

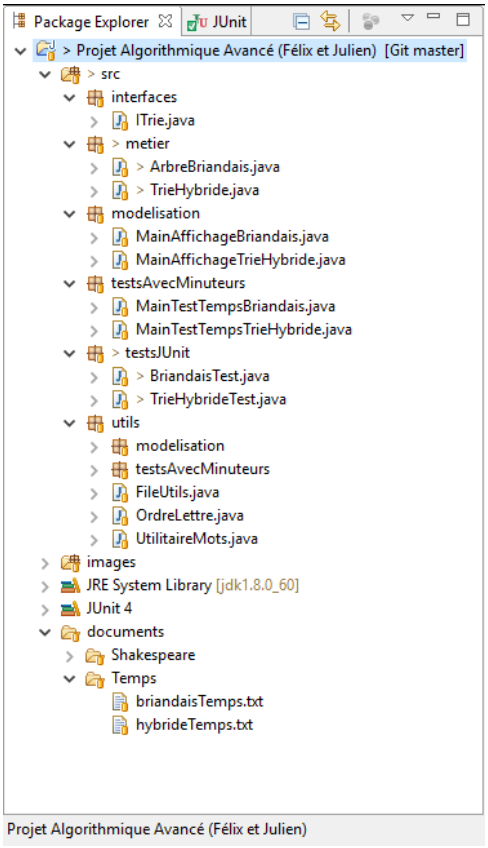
Dans l'ensemble, nous sommes satisfaits de notre projet, dans lequel nous avons réussi à implémenter toutes les fonctions et méthodes demandées, répondre avec un maximum de justesse aux questions de l'énoncé, ainsi qu'effectuer les questions facultatives.

Nous sommes conscients que certaines fonctions ne sont pas implémentées de la façon la plus optimale possible. En effet, pour améliorer la rapidité, nous aurions pu coder ce projet en C ou C++, nous aurions pu rajouter dans nos structures d'arbre un lien pour avoir accès au père/frère gauche du nœud courant et nous faciliter grandement l'implémentation de méthodes comme la suppression, tout en améliorant la complexité. Cependant, nous voulions rester fidèles aux descriptions des structures dans le cours, quitte à avoir des performances légèrement réduites.

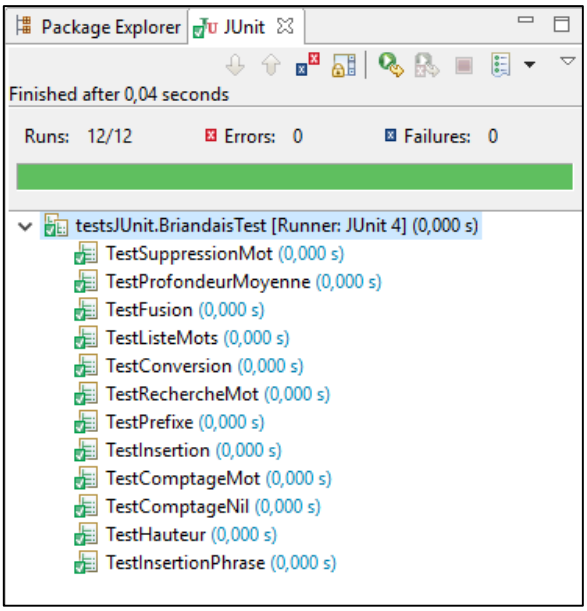
Cependant, même avec ces implémentations, nos fonctions s'exécutent dans des temps tout à fait raisonnable, y compris sur l'exécution des fonctions demandées ou sur les méthodes facultatives sur l'ensemble des œuvres de Shakespeare.

Nous tenons à remercier nos enseignantes Madame Pelletier et Madame Soria pour les connaissances qu'elles nous ont apportées et pour leur aide durant l'élaboration du projet.

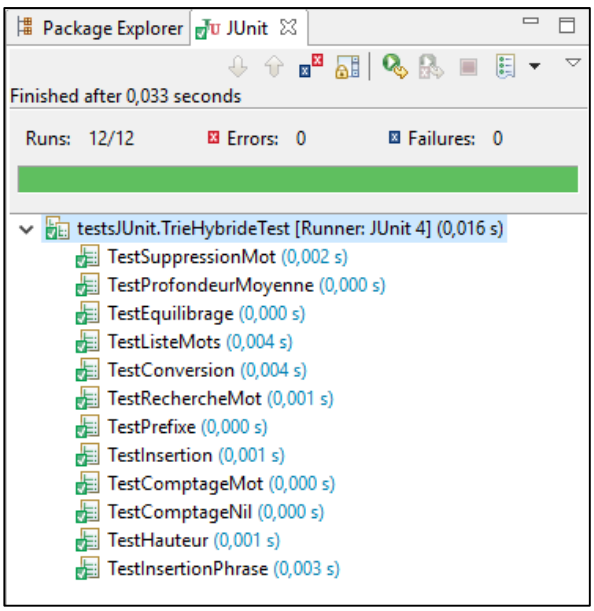
Annexes



Workspace complet du projet, sous Eclipse



Tests JUnit des fonctions de ArbreBriandais



Tests JUnit des fonctions de TrieHybride