Rapport
Projet ALGAV Trie
Implantation du Trie Hybride et du Patricia Trie

Amel Arkoub 3301571
Ling-Chun SO 3414546

22 décembre 2017

Table des matières

0 Introduction **3**

1 Trie Hybride **4**
1.1 Implantation 4
1.1.1 Structure 4
1.2 Description des algorithmes 5
1.2.1 Algorithmes triviaux 5
1.2.2 AjoutMot 5
1.2.3 Suppression 6
1.3 Complexité 6
1.3.1 Recherche 6
1.3.2 Comptage Mots 6
1.3.3 Liste Mots 6
1.3.4 Comptage Nil 6
1.3.5 Hauteur 7
1.3.6 Profondeur Moyenne 7
1.3.7 Préfixe 7
1.3.8 Suppression 7

2 Patricia Trie **8**
2.1 Implantation 8
2.2 Complexité 8
2.2.1 Recherche 8
2.2.2 Comptage Mots 8
2.2.3 Liste Mots 8
2.2.4 Comptage Nil 8
2.2.5 Hauteur 8
2.2.6 Profondeur Moyenne 8
2.2.7 Préfixe 8
2.2.8 Suppression 8

3 Fonctions complexes **9**
3.1 Implantation 9
3.1.1 Fusion de Patricia Trie 9
3.1.2 Conversion de Patricia Trie en Trie Hybride 9
3.1.3 Conversion de Trie Hybride en Patricia Trie 9
3.1.4 Rééquilibrage de Trie Hybride 9
3.2 Complexité 10
3.2.1 Fusion de Patricia Trie 10
3.2.2 Conversion de Patricia Trie en Trie Hybride 10
3.2.3 Conversion de Trie Hybride en Patricia Trie 10

3.2.4 Rééquilibrage de Trie Hybride 10

4 Etude expérimentale **11**
4.1 Temps de construction 11
4.2 Temps d'ajout d'un mot 11
4.3 Temps de suppression 11
4.4 Profondeur des structures 11
4.5 Hauteur 11

Chapitre 0

Introduction

Présentation

Nous souhaitons représenter un dictionnaire de mots, c'est-à-dire implanter une structure de données éfficace stockant des mots. Pour cela, nous allons nous servir de la structure de Trie. Dans cette optique, nous proposons l'implantation de deux structures de tries concurrentes que sont le Trie Hybride et le Patricia Trie. De cette implantation, une étude expérimentale sera analysées afin de mettre en exergue les avantages et inconvénients de chacune de ces structures. Le langage choisit pour l'implantation de ces deux structures est JAVA.

Trie

Un Trie est une représentation arborescente d'un ensemble de clés en évitant de répéter les préfixes communs.

Trie Hybride

Un Trie Hybride est un arbre ternaire dont chaque noeud contient un caractère et une valeur (non vide lorsque le noeud représente une clé). Chaque noeud a 3 pointeurs : un lien Inf (resp. Eq, res. Sup) vers le sous arbre dont le premier caractère est inférieur (resp. égal, resp. supérieur) à son caractère. Il permet en outre de réduire le nombre de pointeurs vide par rapport à un R-Trie.

Patricia Trie

Un Patricia Trie est un arbre dont le but est de réduire la taille des R-Trie tout en conservant une recherche efficace. Pour ce faire, plutôt que chaque noeud interne permette de distinguer une lettre, il permet de distinguer la plus longue sous-chaîne de lettres communes à plusieurs mots.

Chapitre 1

Trie Hybride

1.1 Implantation

1.1.1 Structure

Dans notre implantation JAVA, un Trie Hybride est un objet qui contient 5 éléments :

— un char “letter”, qui correspond à la lettre stockée.

— une valeur “value”, qui correspond à la représentation de fin de mot (-1 le noeud n’est pas la fin d’un mot, sinon la valeur correspond à l’ordre d’ajout croissant).

— un pointeur vers un Trie Hybride “fc”, ce Trie Hybride correpond au fils central, c’est-à-dire on parcours ce Trie Hybride si le caractère recherché au Trie Hybride courant est correct.

— un pointeur vers un Trie Hybride “fg”, ce Trie Hybride correpond au fils gauche, c’est-à-dire on parcours ce Trie Hybride si le caractère recherché au Trie Hybride courant est plus petit dans l’ordre alphabétique.

— un pointeur vers un Trie Hybride “fd”, ce Trie Hybride correpond au fils droit, c’est-à-dire on parcours ce Trie Hybride si le caractère recherché au Trie Hybride courant est plus grand dans l’ordre alphabétique.

Voici une représentation ci-dessous du Trie Hybride résultant des l’ajouts successifs des mots :

— don

— le

— a

— dort

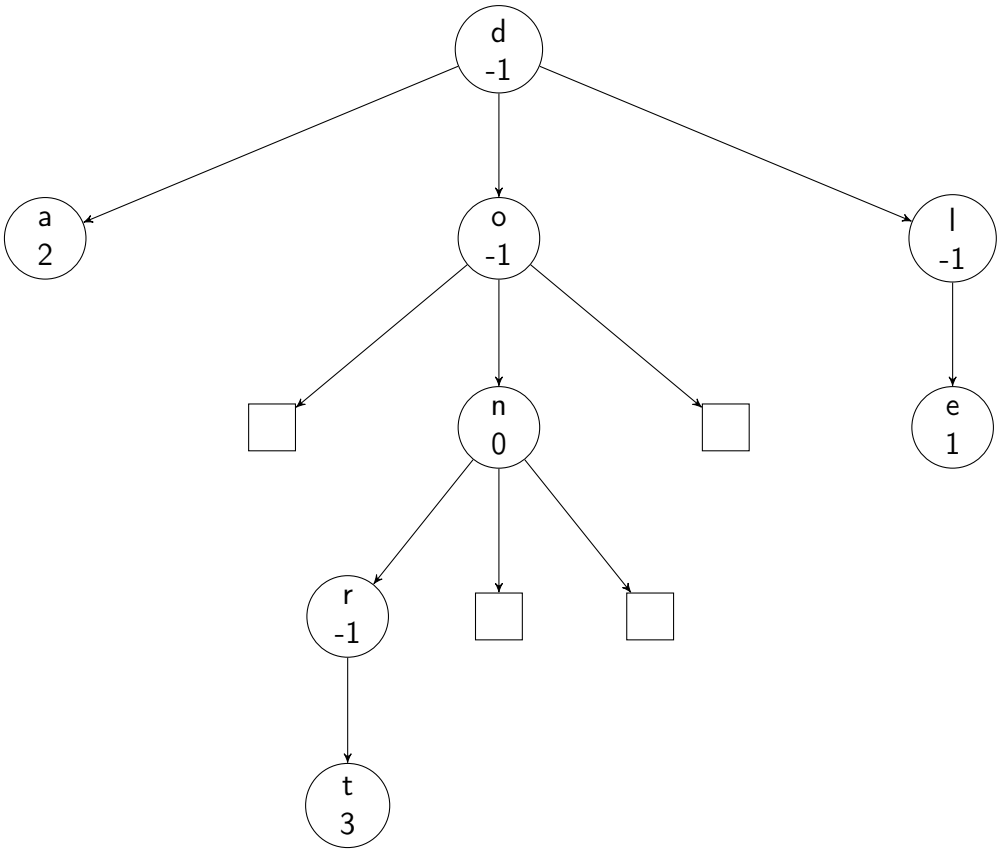

FIGURE 1.1 – Représentation d'un Trie Hybride

1.2 Description des algorithmes

1.2.1 Algorithmes triviaux

La plupart des algorithmes de l'implantation du Trie Hybride sont relativements triviaux, notamment les fonctions :

- Recherche(arbre, mot) → booléen
- ComptageMots(arbre) → entier
- ListeMots(arbre) → liste[mots]
- ComptageNil(arbre) → entier
- Hauteur(arbre) → entier
- ProfondeurMoyenne(arbre) → entier
- Prefixe(arbre) → entier

Elles correpondent pour la plupart d'un simple parcours de la structure du Trie Hybride, dont le traitement dépend de la fonction.

1.2.2 AjoutMot

La fonction ajoutMot permet d'ajouter un mot dans le Trie Hybride et donc aussi la sa construction dont la signature est : ajoutMot(mot, arbre) → void Les étapes sont donc :

- le cas d'arrêt lorsque le mot à été ajouté.
- si on est dans la racine qui est encore vide alors on ajoute le mot.
- si le mot est de longueur un, alors on verifie si le Trie Hybride courant correspond à la bonne lettre sinon on fait des appels récursifs sur le fils gauche ou droit.

— dans le cas général, on compare le caractère du mot et le caractère du Trie Hybride courant, si les lettres concordent alors on appel récursivement sur le fils central, s'ils sont différents on fait un appel récursif sur le fils gauche ou droit suivant l'ordre alphabétique des caractères.

1.2.3 Suppression

La fonction ajoutMot permet la suppression d'un mot dans le Trie Hybride et maintient le Trie Hybride cohérent en supprimant les noeuds dont il n'existe pas de mot, la signature est : suppression(arbre, mot) → void Les étapes sont donc :

— si le Trie Hybride courant est un pointeur null alors on a un cas d'arrêt
— si le mot est réduit à une lettre alors on vérifie que le caractère du mot et du Trie Hybride courant, s'ils ne sont pas égaux alors on fait un appel récursif sur le fils gauche ou droit suivant la comparaison de l'ordre alphabétique des lettres
— dans le cas général, on compare le caractère du mot et le caractère du Trie Hybride courant, si les lettres concordent alors on appel récursivement sur le fils central, s'ils sont différents on fait un appel récursif sur le fils gauche ou droit suivant l'ordre alphabétique des caractères.
— on calcul le nombre de mots issus de chaque fils, s'il existe un fils dont le nombre de mots est égal à 0, alors on détruit le fils en question.

1.3 Complexité

1.3.1 Recherche

Pour la recherche d'un mot de L caractères et un Trie Hybride de taille n, en prenant la comparaison de caractère comme mesure, on obtient une complexité dans le cas général :

— en Θ(L), si $L < n$;
— en Θ(n) sinon.

Sinon on a une complexité en Θ(n) dans le pire cas.

1.3.2 Comptage Mots

Le comptage de mots revient à faire un parcours de l'arbre en entier. Soit un arbre de taille n, en prenant la comparaison de la valeur du noeud comme mesure, on a donc une complexité Θ(n).

1.3.3 Liste Mots

La récupération des mots dans un Trie Hybride correspond aussi à un parcours de l'arbre en entier en gardant le *préfixe* en argument dans les appels de fonctions. Soit un arbre de taille n, en prenant la comparaison de la valeur du noeud comme mesure, on a donc une complexité en Θ(n).

1.3.4 Comptage Nil

Le comptage de pointeur vers null correspond à un parcours de tout les noeuds pour compter le nombre de fils null. Soit un arbre de taille n, en prenant la comparaison de la valeur du noeud comme mesure, on a une complexité en Θ(n) puisque pour n noeuds, on a un nombre de comparaison $\leq 3n$.

1.3.5 Hauteur

La détermination de la hauteur du Trie Hybride est un parcours complet de l'arbre, en prenant l'existence de fils comme mesure , on obtient une complexité en Θ(n).

1.3.6 Profondeur Moyenne

La profondeur moyenne d'un Trie Hybride est un parcours jusqu'aux feuilles dont on ajoute la profondeur dans une liste et on effectue une division. En prenant la comparaison d'existence de fils comme mesure, on obtient une complexité de Θ(n).

1.3.7 Préfixe

La recherche du préfixe peut dans le pire des cas en Θ(n) puisque le pire des cas est atteint lorsque le Trie Hybride revient à être une "liste chaînée" et donc à hauteur h=n.

1.3.8 Suppression

La suppression d'un mot est une recherche dans le Trie Hybride, ce qui correspond à une complexité en Θ(n) comparaison dans le pire cas, cependant il y a aussi 3 appels à la fonction comptageMots de complexité Θ(n). On a donc une complexité en $\mathcal{O}(n^2)$.

Chapitre 2

Patricia Trie

2.1 Implantation

2.2 Complexité

2.2.1 Recherche

2.2.2 Comptage Mots

2.2.3 Liste Mots

2.2.4 Comptage Nil

2.2.5 Hauteur

2.2.6 Profondeur Moyenne

2.2.7 Préfixe

2.2.8 Suppression

Chapitre 3

Fonctions complexes

3.1 Implantation

3.1.1 Fusion de Patricia Trie

3.1.2 Conversion de Patricia Trie en Trie Hybride

3.1.3 Conversion de Trie Hybride en Patricia Trie

3.1.4 Rééquilibrage de Trie Hybride

Après plusieurs ajouts successifs dans un Trie Hybride, ce dernier pourraitre être plutôt déséquilibré. Nous avons donc implantée une fonction permettant d'identifier si un Trie Hybride est équilibré et le rééquilibrage de celui-ci dans le cas échéant. Nous remarquons dans un Trie Hybride qu'il n'est pas modifiable dans la profondeur des fils centraux mais qu'il l'est, à partir d'un noeud courant, pour ses fils gauche et droit. En effet, ceux-ci sont "intervertibles" entre eux, nous pouvons donc considérer comme un équilibrage d'AVL dont le fils gauche et droit correspondent respectivement au fils gauche et droit d'un AVL. La fonction checkBalance(arbre) $\rightarrow$ booléen, permet d'identifier si un Trie Hybride est équilibré ou non, celui-ci calcul dans tout le Trie Hybride s'il existe un noeud dont le nombre de successions de fils gauche ou droit dans le fils gauche ou droit du noeud courant diffère de plus de 1. S'il diffère alors le Trie Hybride est déséquilibré.

La fonction de réequilibrage est balanceTrieHybride(arbre) $\rightarrow$ void, elle se décompose en ces étapes :

- le cas de base si le Trie Hybride est null
- on calcul si le Trie Hybride est déjà équilibré, s'il est équilibré alors on récupère tout les fils centraux de la successions de fils gauche et droit du noeud courant et on effectue des appels recursifs sur ces fils
- on extrait la succession des fils gauche et droit du noeud courant et on effectue un tri dans l'ordre alphabétique
- on retire dans ces noeuds les liens fils gauche et droit
- on calcul le noeud qui sera au milieu, l'élément qui coupe en deux sous tableaux de taille égal
- étant donné que nous manipulons des pointeurs et ne possédant pas de pointeur sur le père, nous remplaçons le contenu dans l'ancien noeud courant par le nouveau noeud milieu et vice-versa
- on effectue un appel à la fonction splitting(arbre,arbre[],arbre[]), celui-ci permet pour un noeud courant d'attribuer le fils gauche et droit. le premier tableau correspond a l'en-

semble des fils gauche que peut prendre le noeud courant et le second tableau correspond à l'ensemble des fils droit que peut prendre le noeud courant. Ainsi cette fonction récursive, calcul par dichotomie, les fils gauche et droit des noeuds.
— on récupère tout les fils centraux du chemin de fils gauche et droit pour faire des appels récursifs

Il est important de noter que le nombre maximal de noeuds d'un chemin de fils gauche et droit est majoré par une contante (26 pour les lettres de l'alphabet).

3.2 Complexité

3.2.1 Fusion de Patricia Trie

3.2.2 Conversion de Patricia Trie en Trie Hybride

3.2.3 Conversion de Trie Hybride en Patricia Trie

3.2.4 Rééquilibrage de Trie Hybride

checkBalance La fonction qui vérifie si un Trie Hybride équilibré, s'appui sur la fonction checkBalanceAux qui lui même utilise countLRNode. Les complexités sont donc :

— countLRNode est en $\Theta(26)$ donc en $\Theta(1)$, car le plus long chemin de fils gauche et droit est majoré par 26
— checkBalanceAux est donc aussi en $\Theta(1)$, puiqu'il calcul pour le fils gauche et droit du noeud courant
— checkBalance est en $\Theta(\log_{26}(n))$ donc en $\Theta(\log(n))$, puisqu'il fait appel à checkBalanceAux pour tout les noeuds sauf les noeuds du chemin de fils gauche et droit.

balanceTrieHybride De la même manière que checkBalance, la fonction balanceTrieHybride est en $\Theta(\log(n))$, puisque toutes les opérations sont des opérations en $\Theta(1)$ sauf pour les appels récursifs.

Chapitre 4

Etude expérimentale

4.1 Temps de construction

Nous avons calculé, le temps de construction des différentes implantations réalisées. Nous avons chargé avec un pas de 1000, tout les mots contenu dans l'oeuvre de Shakespeare, en repartant d'un Trie vide à chaque pas.

FIGURE 4.1 – Courbe du temps de calcul en fonction du nombre d'ajouts

4.2 Temps d'ajout d'un mot

Nous avons aussi calculé le temps d'ajout d'un mot dans des Tries qui avaient les oeuvres de Shakespeare déjà chargé, nous avons ajouté des mots français de longueur croissante, dont le plus long mot est composé de 25 lettres.

FIGURE 4.2 – Courbe du temps de calcul en fonction de la longueur du mot

4.3 Temps de suppression

Le temps de suppression en chargeant dans les Tries les oeuvres de Shakespeare et en supprimant un ensemble de mot avec un pas de 1000.

FIGURE 4.3 – Courbe du temps de calcul en fonction du nombre de suppression

4.4 Profondeur moyenne des structures

Cette étude de la profondeur des structures permet de mettre en évidence l'évolution de la profondeur moyenne des structures suivant le nombre de mot contenu. Nous avons chargé les oeuvres de Shakespeare avec un pas de 1000 et en calculant la profondeur moyenne entre ces ajouts.

FIGURE 4.4 – Courbe de la profondeur moyenne en fonction du nombre d'ajouts

4.5 Hauteur

De la même manière que pour le calcul de la profondeur moyenne, nous avons chargé les oeuvres et Shakespeare avec un pas de 1000 et en calculant la hauteur entre ces ajouts.

FIGURE 4.5 – Courbe de la hauteur en fonction du nombre d'ajouts