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

Amel ARKOUB 3301571
Ling-Chun SO 3414546

22 décembre 2017

Contents

0

Introduction

Présentation

Nous souhaitons représenter un dictionnaire de mots, c'est-à-dire implanter une structure de données efficace stockant des mots. Pour cela, nous allons nous servir de la structure de *trie*. Dans cette optique, nous proposons les implantations de deux structures de tries concurrentes : le trie hybride et le PATRICIA trie. Ensuite, nous effectuerons une étude expérimentale de laquelle nous analyserons les résultats, afin de mettre en exergue les avantages et inconvénients de chacune des structures. Nous avons choisi le langage de programmation Java pour nos implantations.

Trie

Un trie est une représentation arborescente d'un ensemble de clés qui permet d'éviter la répétition de préfixes communs des mots.

Trie Hybride

Un trie hybride est un arbre ternaire dont chaque nœud contient un caractère et une valeur (non vide lorsque le nœud représente une clé). Chaque nœud 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 vides par rapport à un R-Trie.

PATRICIA Trie

Un PATRICIA trie est un arbre dont le but est de réduire la hauteur des R-Trie tout en conservant une recherche efficace. Pour ce faire, plutôt que chaque nœud interne ait pour valeur une lettre, il a pour valeur le préfixe commun à un ensemble de mots.

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 nœud 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 correspond 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 correspond 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 correspond au fils droit, c’est-à-dire on parcourt ce trie hybride si le caractère recherché au trie hybride courant est plus grand dans l’ordre alphabétique.

Voici une représentation du trie hybride résultant des ajouts successifs des mots:

- don
- le
- a
- dort

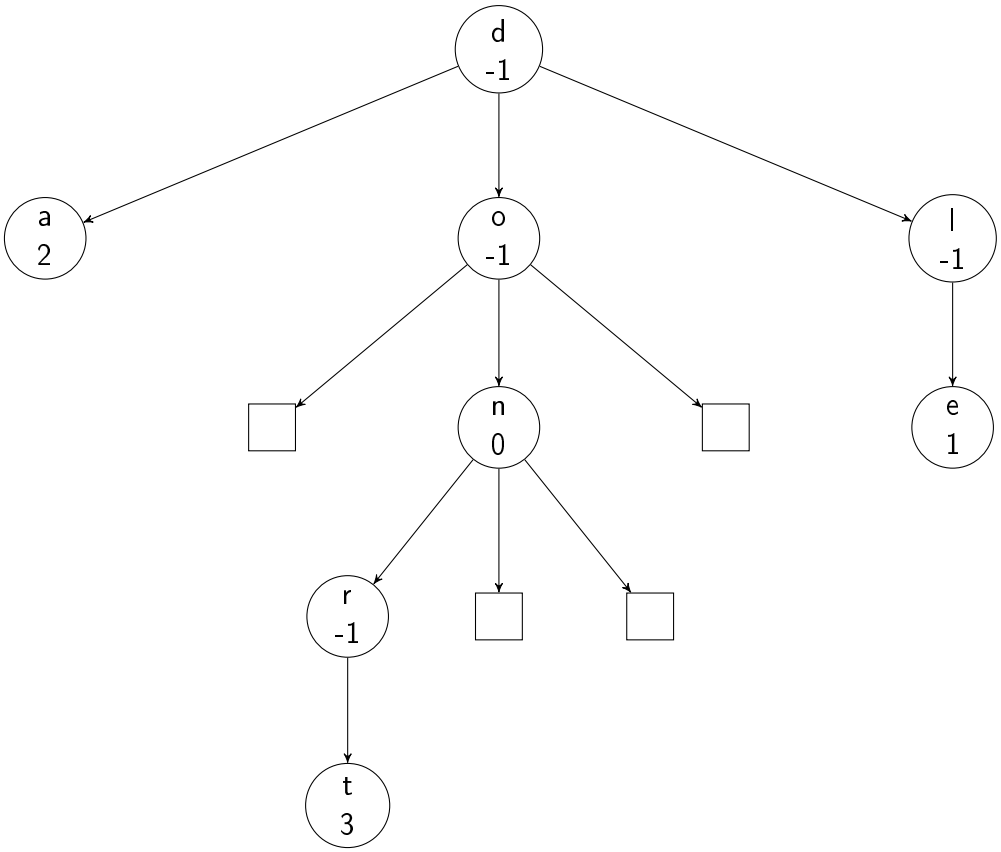

Figure 1.1: Représentation d'un Trie Hybride

1.2 Description des algorithmes

1.2.1 Algorithmes triviaux

La plupart des algorithmes de l'implantation du trie hybride sont relativement 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 correspondent 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 par conséquent la construction même du trie hybride, dont la signature est: ajoutMot(mot, arbre) $\rightarrow$ void. Les étapes sont donc:

- le cas d'arrêt lorsque le mot a été ajouté.
- si on est dans la racine qui est encore vide alors on ajoute le mot.
- si le mot est de longueur 1, alors on vérifie 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 appelle récursivement sur le fils central ; si elles sont différentes, on fait un appel récursif sur le fils gauche ou droit suivant l'ordre alphabétique des caractères.

1.2.3 Suppression

La fonction suppressionMot permet la suppression d'un mot dans le trie hybride et maintient le trie hybride cohérent en supprimant les nœuds dont il n'existe pas de mot, la signature est: suppression(arbre, mot) $\rightarrow$ 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 sont égaux. Dans ce cas, on ne considère plus le trie hybride comme un mot. Dans le cas contraire, c'est-à-dire 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 appelle récursivement sur le fils central ; si elles sont différentes, 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 nœud 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 nœud 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 tous les nœuds pour compter le nombre de fils null. Soit un arbre de taille n, en prenant la comparaison de la valeur du nœud comme mesure, on a une complexité en Θ(n) puisque pour n noeuds, on a un nombre de comparaisons $\leq 3n$.

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 être dans le pire cas en Θ(n). En effet, le pire cas est atteint lorsque le trie hybride est 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) comparaisons 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 $\mathcal{O}(n^2)$.

2

Patricia Trie

2.1 Implantation

Dans notre implantation JAVA, un PATRICIA trie est représenté par :

- une chaîne de caractères, nommée *valeur*, qui est le préfixe.
- un tableau de 27 PATRICIA tries fils *patTries*, un pour chaque lettre de l'alphabet, le 27ème représentant le caractère de fin de mot. Ce caractère est "{" .
- un indice, entier nommé *ind*, qui est le numéro de la prochaine lettre à évaluer dans les mots fils. En effet, les mots fils ont ce même préfixe, et diffèrent à partir de la lettre indiquée par l'indice.
- un booléen, *isFeuille* indiquant si ce nœud est une feuille ou non. Dans le cas où le nœud est une feuille, le préfixe est en fait le mot en entier.

AJOUTER SCHÉMA PAT TRIE

2.2 Description des algorithmes

2.2.1 Algorithmes triviaux

La plupart des algorithmes de l'implantation du trie hybride sont relativement 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 correspondent pour la plupart d'un simple parcours de la structure du trie hybride, dont le traitement dépend de la fonction.

2.2.2 Ajouter un mot

La fonction ajouterMot permet d'ajouter un mot qui n'est pas déjà présent dans le PATRICIA trie. La signature est: ajouterMot(arbre, mot) → booléen. Voici une description succincte de l'algorithme :

Si la taille du mot à ajouter équivaut à celle du préfixe du PATRICIA trie :

- s'il existe déjà un PATRICIA trie fils de fin de mot, c'est-à-dire celui qui se trouve à la case correspondant au caractère de fin de mot "{", on renvoie FAUX. En effet, cela signifie qu'il est déjà présent.
- sinon, on le crée puis on renvoie VRAI.

Sinon on regarde le PATRICIA trie fils se trouvant à la lettre indiquée par l'indice du PATRICIA trie courant.

- si la case indiquée est un pointeur vers NIL, alors on crée à cette case un nouveau PATRICIA trie, ayant pour valeur le mot, pour indice l'indice du père + 1 et est considéré comme une feuille. Ensuite, on renvoie VRAI.
- sinon un PATRICIA trie fils existe déjà à cette case, et alors il faut comparer sa valeur avec le mot qu'on veut ajouter.
  - si sa valeur, notée val, est correspond au préfixe du mot à ajouter, alors on fait un appel récursif sur ce PATRICIA trie.
  - sinon, on compare val avec le mot à ajouter, pour trouver l'indice à partir duquel ils diffèrent. En effet, on doit créer un PATRICIA trie qui contient le préfixe commun de val et du mot à ajouter, puis créer des PATRICIA tries fils à ce nouveau PATRICIA trie, qui contiendront val et le mot à ajouter.

  Finalement, on renvoie VRAI.

2.2.3 Supprimer un mot

La fonction supprimerMot permet la suppression d'un mot dans le trie hybride et maintient le trie hybride cohérent en supprimant les nœuds dont il n'existe pas de mot, la signature est: supprimerMot(arbre, mot) → booléen

Voici une description succincte de l'algorithme :

- Si la longueur du mot est strictement inférieure à l'indice du PATRICIA trie, le mot n'est pas dans le PATRICIA trie, on renvoie FAUX.
- Si la longueur du mot est égale à cet indice, on considère le PATRICIA trie fils de fin de mot. Sinon, on considère le PATRICIA trie fils se trouvant à la case correspondant à la lettre à considérer dans le mot. On trouve cette case en récupérant la lettre du mot à l'indice du PATRICIA trie. En effet, cet indice est aussi la taille du préfixe. Donc la lettre du mot qui se trouve à cet indice est la lettre qui suit le préfixe dans le mot.
- Si le PATRICIA trie fils considéré est un pointeur vers NIL, cela signifie que le mot n'est pas présent dans l'arbre, et on renvoie FAUX.
- Si le PATRICIA trie fils considéré est une feuille et que sa valeur équivaut au mot à supprimer, on fait pointer cette case vers NIL. En d'autres termes, le mot ne fait plus partie de l'arbre. Maintenant, on parcourt tous les fils du PATRICIA trie courant, pour

voir s'il existe un seul autre PATRICIA trie fils. Dans ce cas là, le PATRICIA trie courant prend la valeur de son fils, son indice et est une feuille. On fait donc pointer ce fils vers NIL, ensuite on renvoie VRAI.

- Sinon, on renvoie le résultat de l'appel récursif sur ce PATRICIA trie fils.

2.3 Complexité

Comme nous avons décidé d'implanter le PATRICIA trie avec un tableau pour ses fils, l'accès à ses fils se fait en $\Theta(1)$.

2.3.1 Recherche

La recherche d'un mot dans un PATRICIA trie de hauteur h est de complexité O(h). En effet, lors de la recherche, on est aiguillé par l'indice du PATRICIA trie, qui nous mène vers un de ses PATRICIA trie fils.

2.3.2 Comptage Mots

Soit un PATRICIA trie de taille n. Compter les mots qu'il contient revient à parcourir chaque nœud de l'arbre. Ainsi sa complexité est $\Theta(n)$.

2.3.3 Liste Mots

Soit un PATRICIA trie de taille n. Lister les mots qu'il contient revient à parcourir chaque nœud de l'arbre. Ainsi sa complexité est $\Theta(n)$.

2.3.4 Comptage Nil

Soit un PATRICIA trie de taille n. Compter le nombre de pointeurs à nil qu'il contient revient à compter le nombre de pointeurs à nil de chaque nœud de l'arbre. Ainsi sa complexité est $\Theta(n)$.

2.3.5 Hauteur

Soit un PATRICIA trie de taille n. Pour trouver sa hauteur, il faut parcourir chaque branche de l'arbre. Ainsi, on parcourt chaque nœud qu'il contient. Ainsi sa complexité est $\Theta(n)$.

2.3.6 Profondeur Moyenne

Soit un PATRICIA trie de taille n. Pour trouver sa profondeur moyenne, il faut parcourir chaque branche de l'arbre, et récupérer toutes les longueurs de ses branches, et ensuite en faire la moyenne (en O(1)). Ainsi, on parcourt chaque nœud qu'il contient. Ainsi sa complexité est $\Theta(n)$.

2.3.7 Préfixe

Soit un PATRICIA trie de taille n. Pour le nombre de mots qui ont en commun un certain préfixe, il faut parcourir la branche de l'arbre qui contient ce préfixe. Ainsi, on parcourt chaque nœud qu'il contient. Ainsi sa complexité est $\Theta(n)$.

2.3.8 Suppression

Soit un PATRICIA trie de hauteur h. Pour supprimer un mot, on parcourt dans le pire cas la hauteur de l'arbre. Comme les accès aux PATRICIA tries fils se font en $\Theta(1)$, la complexité est de la suppression est la hauteur de l'arbre O(h).

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 peut être plutôt déséquilibré. Nous avons donc implanté une fonction permettant d'identifier si un trie hybride est équilibré et le rééquilibrage de celui-ci dans le cas contraire. 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 nœud 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 les fils gauche et droit correspondent respectivement aux fils gauche et droit d'un AVL. La fonction checkBalance(arbre) $\rightarrow$ booléen, permet d'identifier si un trie hybride est équilibré ou non. Celle-ci calcule dans tout le trie hybride s'il existe un nœud dont le nombre de successions de fils gauche ou droit dans le fils gauche ou droit du nœud courant diffère de plus de 1. Si tel est le cas, alors le trie hybride est déséquilibré.

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

- le cas de base est le trie hybride est null.
- on calcule si le trie hybride est déjà équilibré : s'il est équilibré alors on récupère tous les fils centraux de la successions de fils gauche et droit du nœud courant et on effectue des appels recursifs sur ces fils.
- on extrait la succession des fils gauche et droit du nœud courant et on effectue un tri dans l'ordre alphabétique
- on retire dans ces nœuds les liens fils gauche et droit
- on calcule le nœud 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 nœud courant par le nouveau nœud milieu, et vice-versa.
- on effectue un appel à la fonction splitting(arbre,arbre[],arbre[]), celui-ci permet pour un nœud courant d'attribuer le fils gauche et droit. le premier tableau correspond a l'ensemble des fils gauches que peut prendre le nœud courant et le second tableau correspond à l'ensemble des fils droits que peut prendre le nœud courant. Ainsi cette fonction récursive calcule par dichotomie les fils gauche et droit des nœuds.
- on récupère tous 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 nœuds d'un chemin de fils gauche et droit est majoré par une constante (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 est équilibré s'appuie sur la fonction checkBalanceAux, qui elle-même fait appel à la fonction countLRNode. Les complexités sont donc:

- countLRNode est en $\Theta(26)$ donc en $\Theta(1)$, car le plus long chemin des fils gauche et droit est majoré par 26
- checkBalanceAux est donc aussi en $\Theta(1)$, puisque cette fonction calcule pour les fils gauche et droit du nœud courant
- checkBalance est en $\Theta(\log_{26}(n))$ donc en $\Theta(\log(n))$, puisqu'elle fait appel à checkBalanceAux pour tous les nœuds sauf les nœuds du chemin des 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.

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, tous les mots contenus dans l'œuvre de Shakespeare, en repartant d'un trie vide à chaque pas.

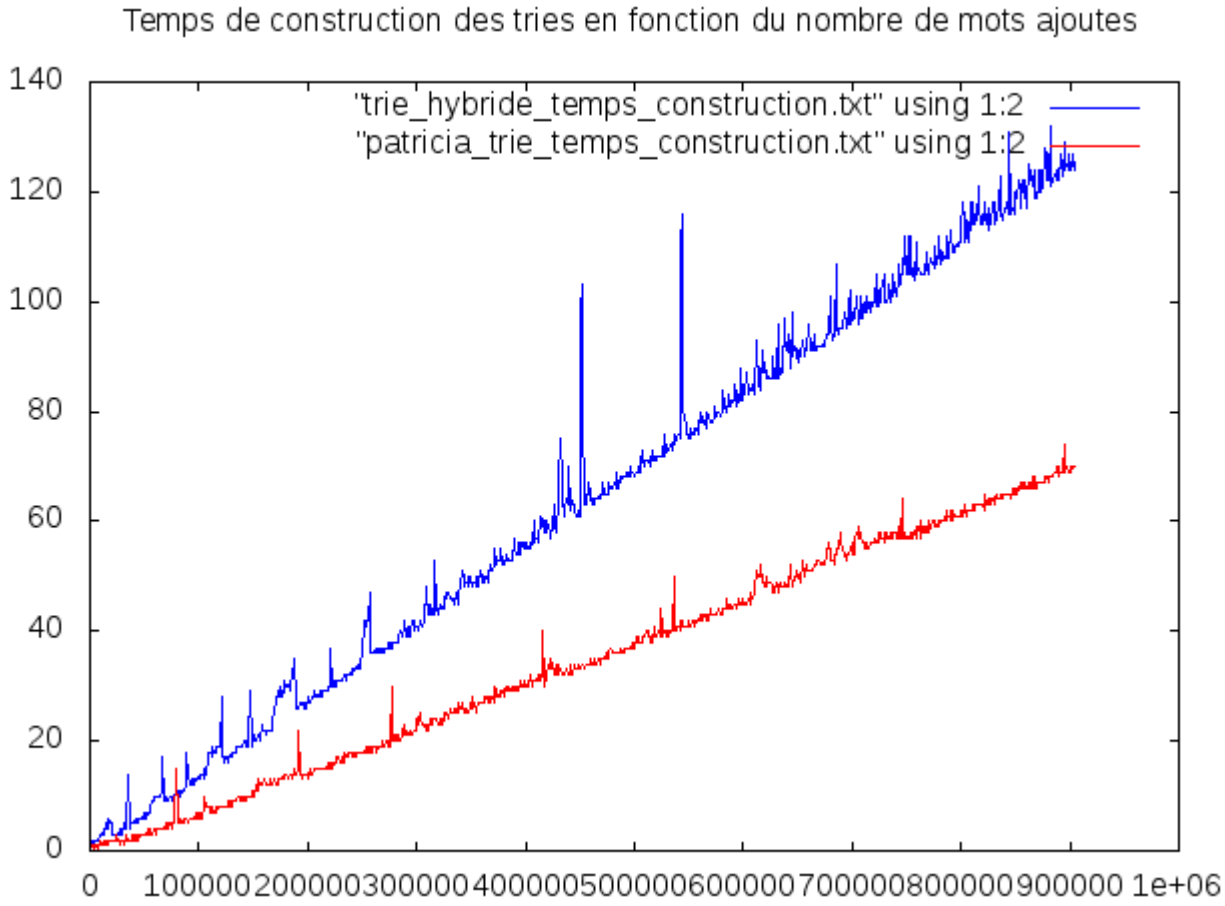

Figure 4.1: Courbe du temps de calcul en fonction du nombre d'ajouts

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 œuvres de Shakespeare déjà chargées. Nous avons alors ajouté des mots français de longueur croissante, dont le plus long mot est composé de 25 lettres.

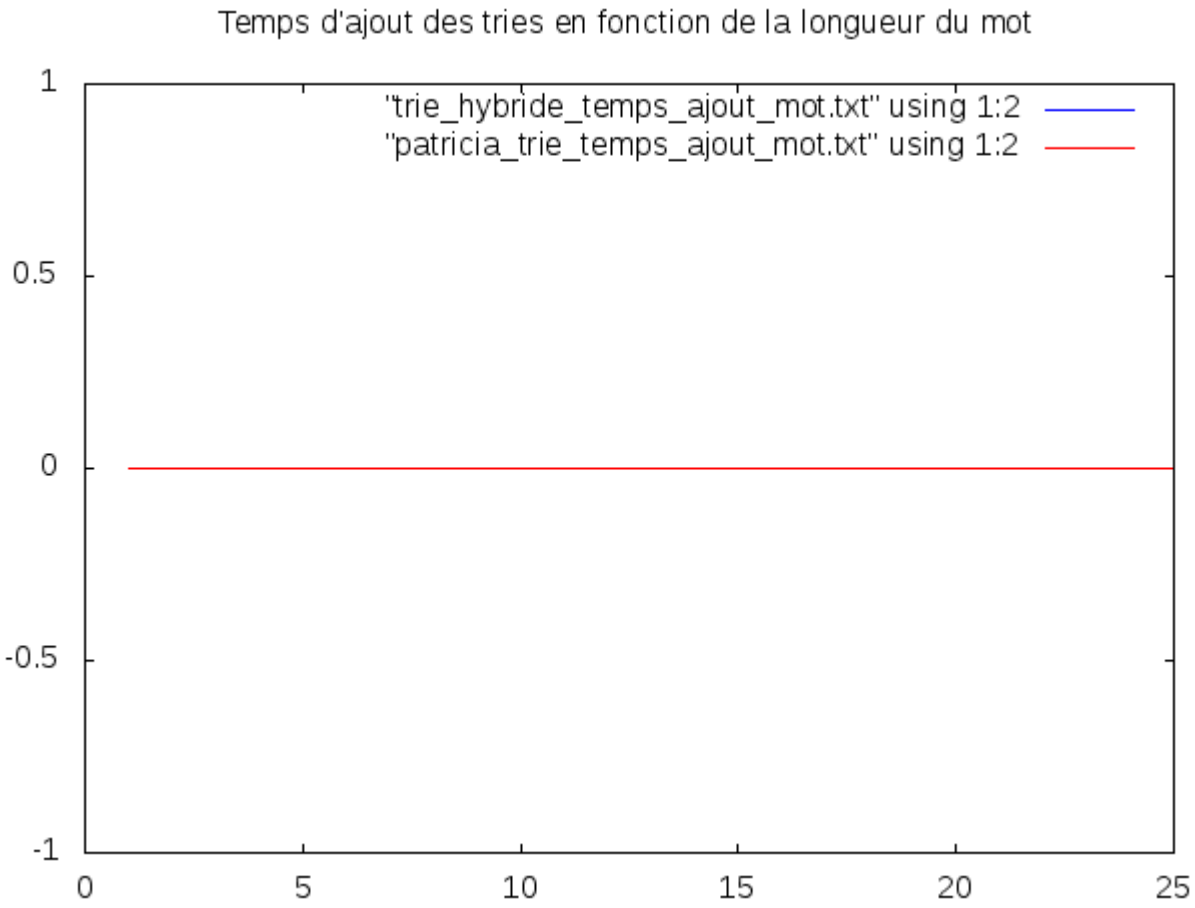

Figure 4.2: Courbe du temps de calcul en fonction de la longueur du mot

4.3 Temps de suppression

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

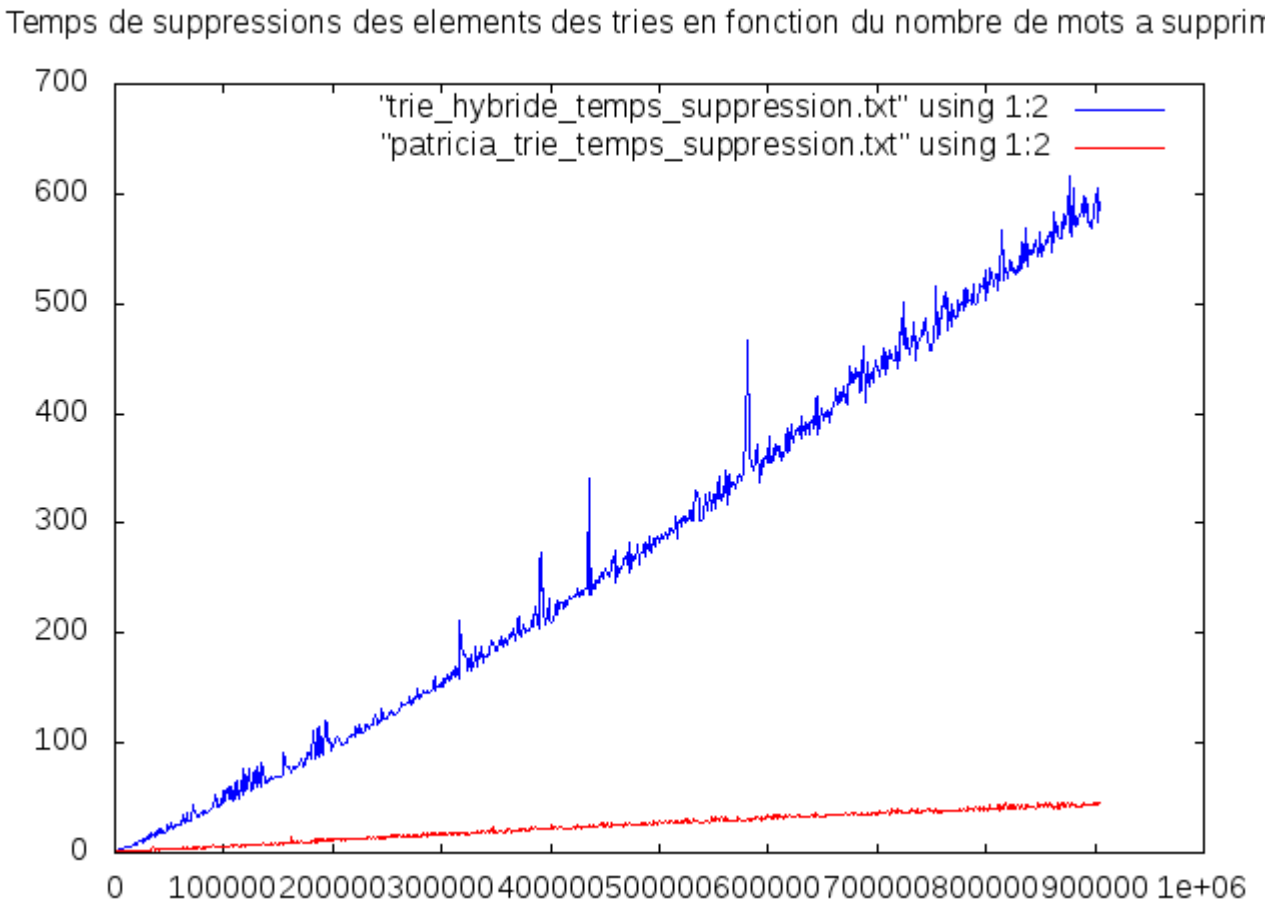

Figure 4.3: Courbe du temps de calcul en fonction du nombre de suppression

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 mots contenus. Nous avons chargé les œuvres de Shakespeare avec un pas de 1000 et en calculant la profondeur moyenne entre ces ajouts.

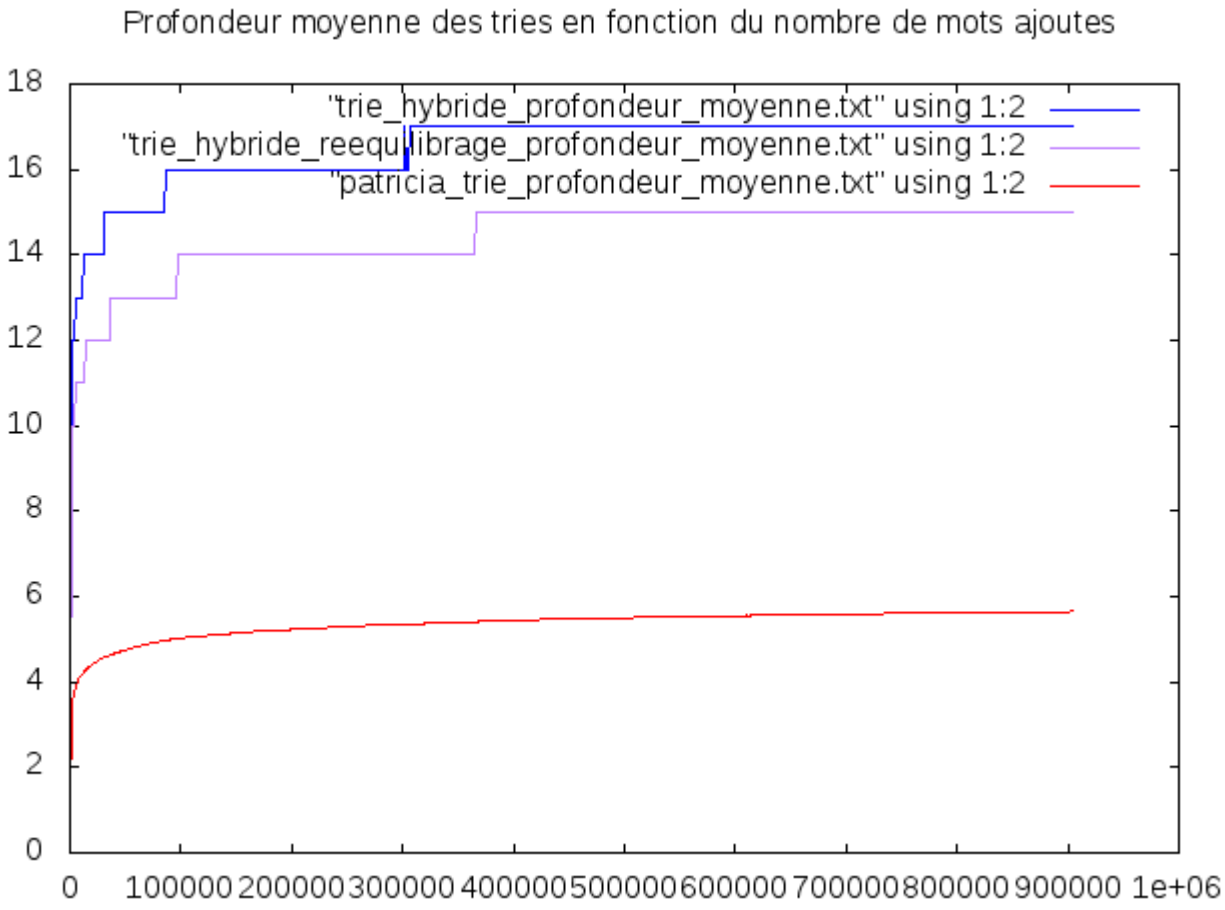

Figure 4.4: Courbe de la profondeur moyenne en fonction du nombre d'ajouts

4.5 Hauteur

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

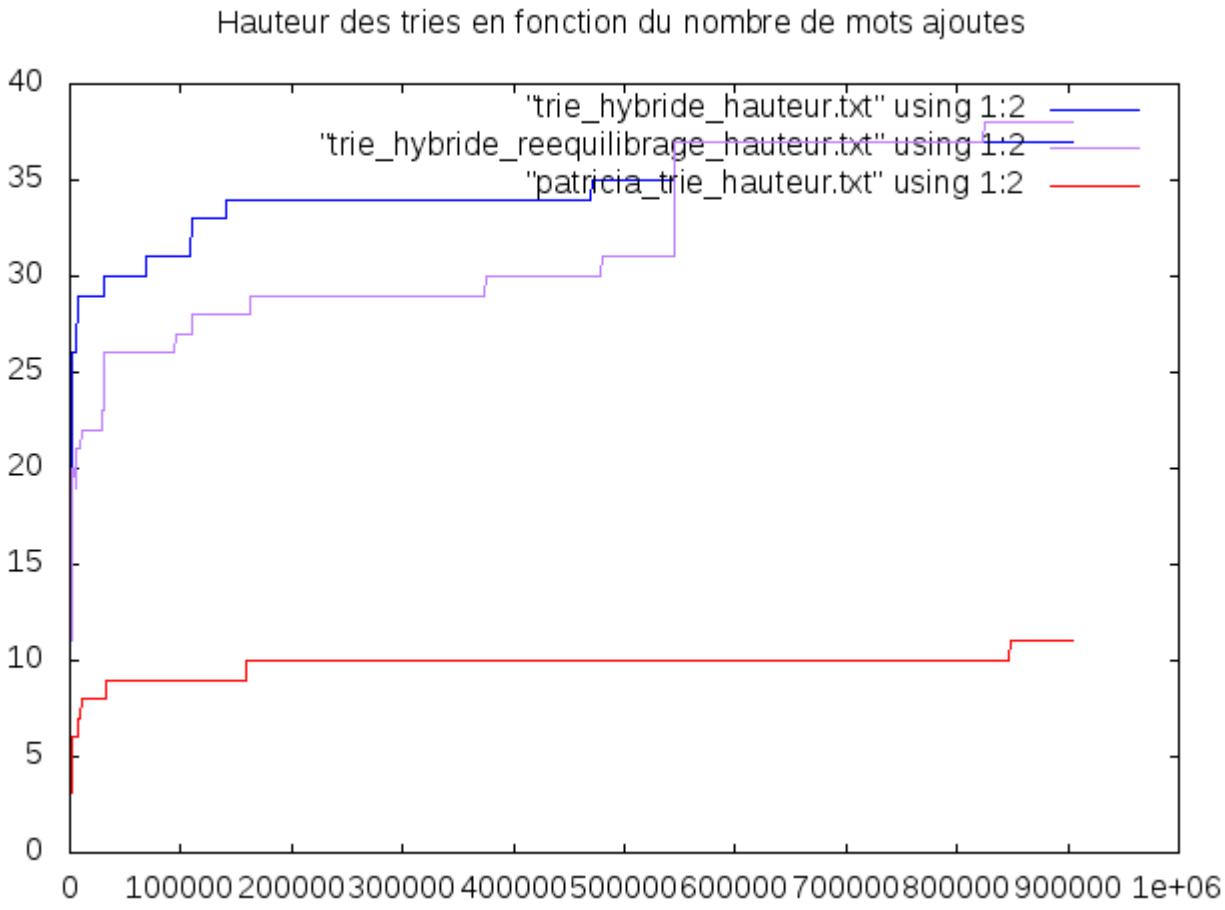

Figure 4.5: Courbe de la hauteur en fonction du nombre d'ajouts