Rapport Projet d'Algorithmique Avancée - MU4IN500

Loïc Daudé Mondet
21304461

Réda Boudrouss

19/12/2024

Table des matières

I Introduction

Le but du problème consiste à représenter un dictionnaire de mots. Dans cette optique, nous proposons l'implémentation de deux structures de tries concurrentes puis une étude expérimentale permettant de mettre en avant les avantages et inconvénients de chacun des modèles.

II Présentation des deux structures de données

II.1 Patricia tries

Dans les patricia tries (Practical Algorithm To Retrieve Information Coded In Alphanumeric), chaque noeud interne ne permet pas de distinguer une lettre, mais la plus longue sous-chaîne de lettres commune à plusieurs mots

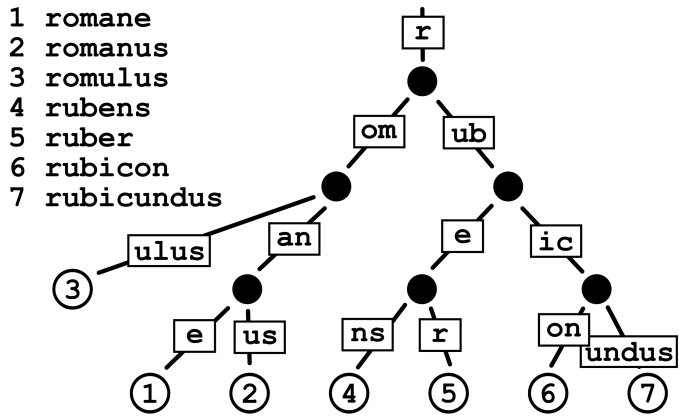

FIGURE II.1 – Exemple de Patricia Trie

II.2 Hybrid tries

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 vers le sous arbre dont le premier caractère est inférieur à son caractère, un lien Eq vers le sous arbre dont le premier caractère est égal à son caractère, un lien Sup vers le sous arbre dont le premier caractère est supérieur à son caractère.

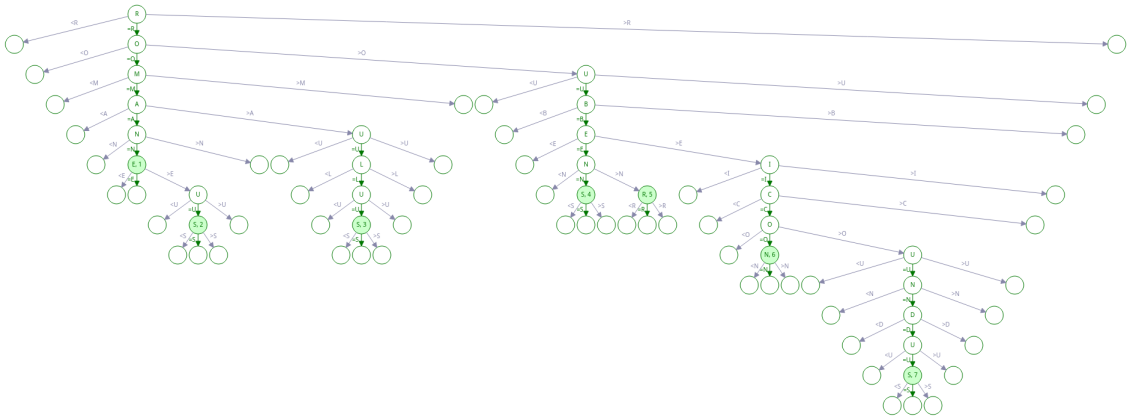

FIGURE II.2 – Exemple de Hybrid Trie pour l'insertion des mots : romane, romanus, romulus, rubens, ruber, rubicon, rubicundus

III Implémention des structures de données

III.1 Choix du langage

Nous avons choisi pour l'implémentation le langage JavaScript, malgré sa mauvaise réputation ce langage dispose d'une bibliothèque standard très étoffée notamment en ce qui concerne la manipulation des chaînes de caractères ou le parsing du JSON. Le fait qu'il soit un langage de script permet également un prototypage rapide, pour autant, les moteurs disponibles sont très performants ce qui est crucial quand il s'agit de traiter de grandes quantités de données (ici beaucoup de texte). Afin de rendre notre code plus sûr et correct, nous avons opté pour l'utilisation du sur-ensemble TypeScript qui permet comme son nom le laisse transparaître un typage des données. Aussi, nous utilisons le runtime Deno car il gère nativement le TypeScript.

III.2 Architecture du code

TypeScript est un langage orienté objet, nous avons donc implémenté les deux types de Trie en utilisant des classes. Plus précisement deux classes pour chaque Trie (une pour l'arbre et une pour les noeuds).

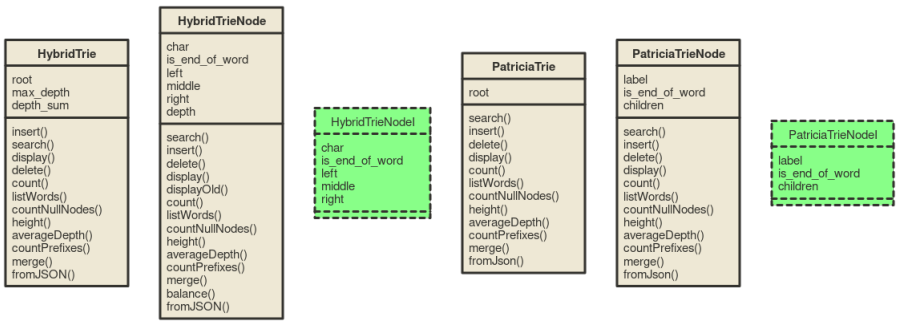

FIGURE III.1 – Classes et interfaces pour les différents Tries

IV Présentation

IV.1 Structure 1 : Patricia-Tries

IV.1.1 Question 1.1

Comme nous utilisons un langage de haut niveau qui traite les chaînes de caractère comme des objets, nous pouvons nous permettre d'utiliser un booléen et non un caractère ASCII pour marquer la fin d'un mot.

IV.1.2 Question 1.3

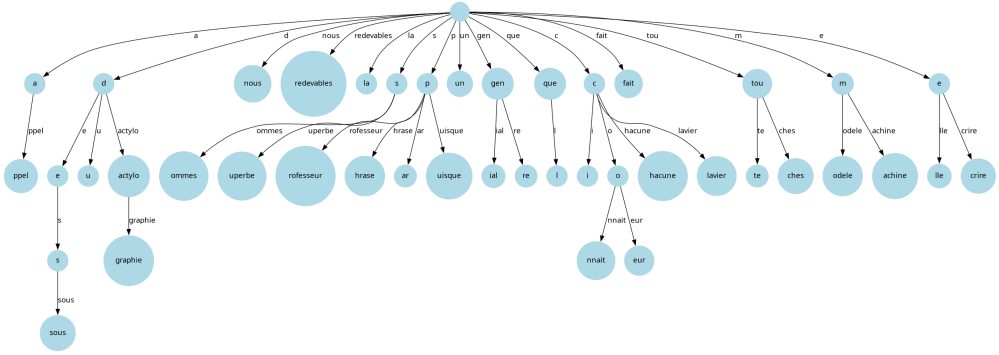

FIGURE IV.1 – Arbre représentant l'exemple de base avec un Patricia Trie

On remarque que les caractères spéciaux et les accents ont été retirés de la phrase de base afin d'être compatible avec le code ASCII.
NB : Notre implémentation utilise le code UTF-8 car c'est le standard en JavaScript/TypeScript, ce qui signifie que nous aurions pu nous passer de ces restrictions.

IV.2 Structure 2 : Tries Hybrides

IV.2.1 Question 1.5

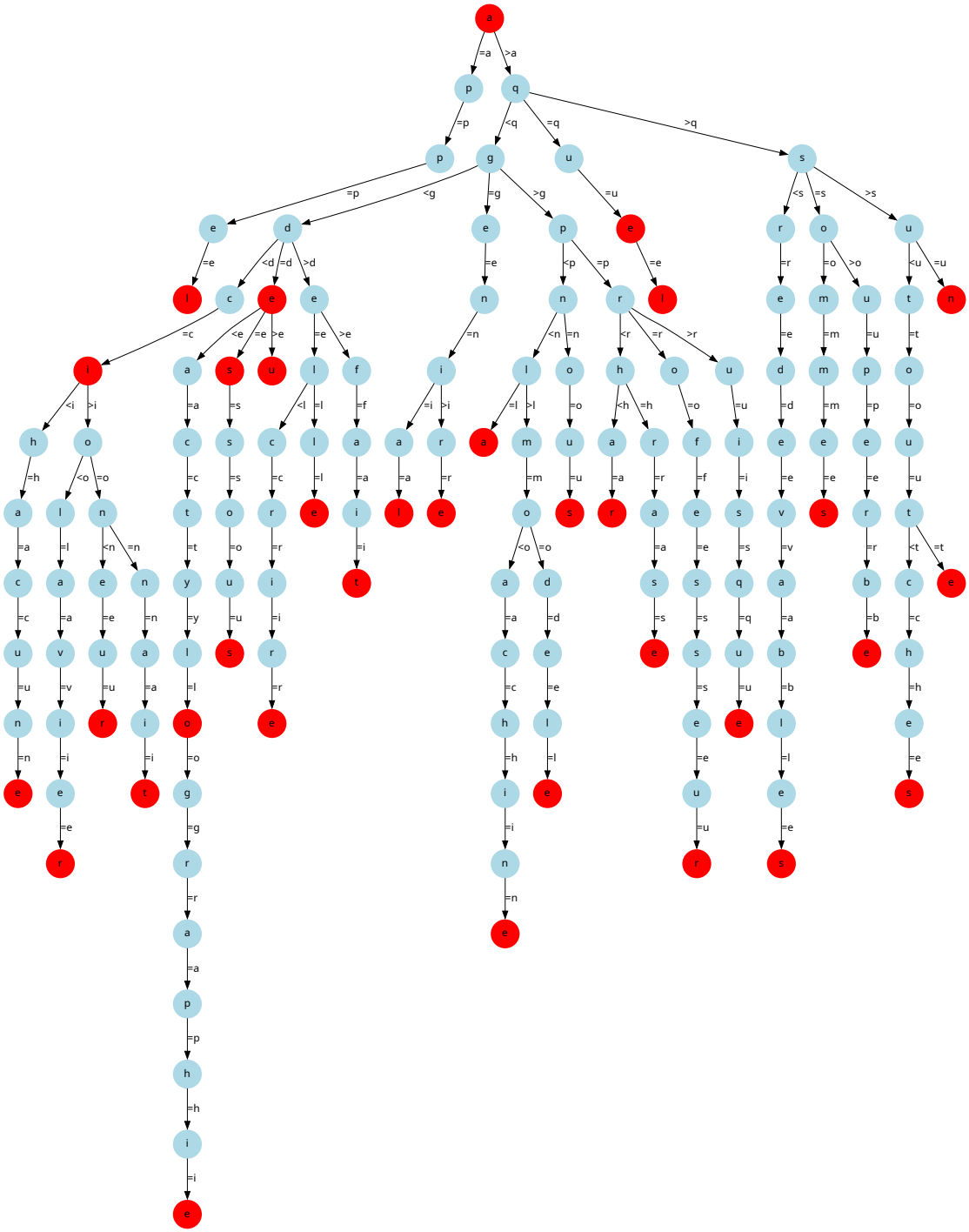

FIGURE IV.2 – Arbre représentant l'exemple de base avec un Hybrid Trie

V Fonctions complexes

V.1 Question 3.7

V.2 Question 3.8

Pour rendre un trie hybride plus équilibré on peut appliquer plusieurs stratégies, premièrement on pourrait essayer de faire des rotations comme avec des arbres rouge-noir, mais il faudrait pendre en compte le fait qu'ici c'est un arbre ternaire, on pourrait aussi extraire les mots de l'arbre, les triers par ordre alphabétique et en reconstruire un nouveau. Pour déterminer si un trie hybride est trop déséquilibré on pourrait comparer la profondeur moyenne des branches avec la profondeur max rencontrée, si ce rapport est supérieur à un seuil fixé, disons 2, alors on considère l'arbre comme étant trop déséquilibré. C'est cette approche couplée au tri par ordre alphabétique que nous avons choisi d'implémenter.

VI Complexités

VI.1 Question 4.9

VII Étude expérimentale

VII.1 Question 6.10

VII.2 Question 6.11

VII.3 Question 6.12

VII.4 Question 6.13

VII.5 Question 6.13