ALGAV PROJECT

Devoir de programmation : Tries

Guillaume HIVERT && Darius MERCADIER

I. Introduction

Ce rapport présente le travail que nous avons fait dans le cadre du devoir de programmation de d'ALGAV.

L'objectif de ce devoir est de comparer les performances de deux structures de dictionnaires différentes : les Trie Hybrides d'une part (aussi appelés Arbres Ternaires de Recherche ou Ternary Search Trees), et les arbres de la Briandais d'autre part.

Tout d'abord, quelques mots par rapport au code qui accompagne ce rapport. Celui-ci est écrit dans le langage de programmation Python. Cela présente notamment l'avantage d'être assez lisible et compréhensible, dans le mesure où ce langage ressemble fort à du pseudo-code, mais permet malgré tout de s'abstraire de bon nombres de problèmes (gestion mémoire facilitée, bibliothèque standard étoffée, gestion des listes et des objets, etc.). De plus, un certain nombre de commentaires y sont présents pour faciliter sa lecture.
Nous avons choisi de ne pas copier-coller de code dans ce rapport puisque nous l'avons fourni avec. En revanche, nous nous y réfèreront lorsque ce sera nécessaire, et expliqueront les fonctions qui ont besoin de l'être.

Quant aux mots ajoutés dans les structures, nous avons dû les formater légèrement avant de les ajouter dans les structures : seuls les caractères alphabétiques et les tirets sont gardés. Ce qui fait un total de 27 caractères différents possibles (c'est un chiffre que nous retrouverons lorsque nous parleront des complexités plus tard dans ce rapport).

Quel caractère ASCII choisir pour encoder la fin d'un mot?

Le caractère 00 représente le caractère 'NUL' dans le codage ASCII. Les mots étant composé exclusivement de lettres (et éventuellement quelques caractères spéciaux comme des tirets), ce caractère ne peut donc pas faire partie d'un mot, et est donc tout indiqué pour représenter la fin d'un mot.

En réalité, presque n'importe quel caractère non-alphabétique aurait pu convenir : #, *, +, ~, etc.

Cependant, nous avons opté pour une implémentation un peu différente : chaque nœud contient un champ booléen 'final', qui vaut 'True' si le nœud en question est la fin d'un mot, et 'False' sinon.
On peut noter qu'un booléen occupe théoriquement 1 bit (bien que ce ne soit probablement pas le cas en python), là où un objet avec ses pointeurs (8 octets en 64 bits) sur son frère, son fils, et sa lettre occupe à priori 8 + 8 + 2 octets (soit 18*8 = 144 bits). Donc utiliser un booléen en plus par nœuds est rentable, à condition d'avoir moins de 144 nœuds de fin de mot pour chaque nœud à l'intérieur des mots, ce qui est le cas.

II. Fonctions de base

Un arbre de la Briandais est représenté par un ensemble de nœuds. Chacun de ces nœuds est représenté par un objet, contenant les champs suivants :

key : le caractère stocké dans le nœud
final : un booléen valait vrai si un mot finit sur ce nœud, faux sinon
brother : le frère du nœud (None si le nœud n'a pas de frère)
child : le fils du nœud (None si le nœud n'a pas de fils)

Les Trie Hybrides sont également formés par un ensemble de nœuds. Les champs des objet-nœuds sont les suivants :

key : le caractère stocké dans le nœud
final : un booléen valait vrai si un mot finit sur ce nœud, faux sinon
middle : le pointeur vers le nœud du milieu
left : le pointeur vers le nœud de gauche
right : le pointeur vers le nœud de droite

Les primitives de manipulation de ces deux structures ont les mêmes signatures, et sont les suivantes :

BriandaisTree `: ( ) -> BriandaisTree`
Renvoie un nouvel Arbre de la Briandais vide.
TernaryTree `: ( ) -> TernaryTree`
Renvoie un nouveau Trie Hybride vide.

add_word `: Tree * Word -> ( )`
Ajoute un mot dans un arbre. (Fonction utilisant des effets de bord, donc pas de valeur de retour, excepté si le mot passé en paramètre est vide, auquel cas *None* est renvoyé)

is_empty `: Tree -> Bool`
Renvoie vrai si et seulement si l'arbre est vide.

L'implémentation de ces fonctions est assez basique, et le code python que nous avons joins au rapport devrait vous renseigner si vous le souhaitez.

Ces primitives permettent de créer un arbre, puis de le remplir par ajout successif de mots. Nous avons ainsi pu créer l'arbre de la Briandais, et le Trie Hybride contenant tous les mots de l'`exemple de base`. Les 2 pages suivantes montrent des représentations visuelles des arbres ainsi obtenus.

A- Arbre de la Briandais

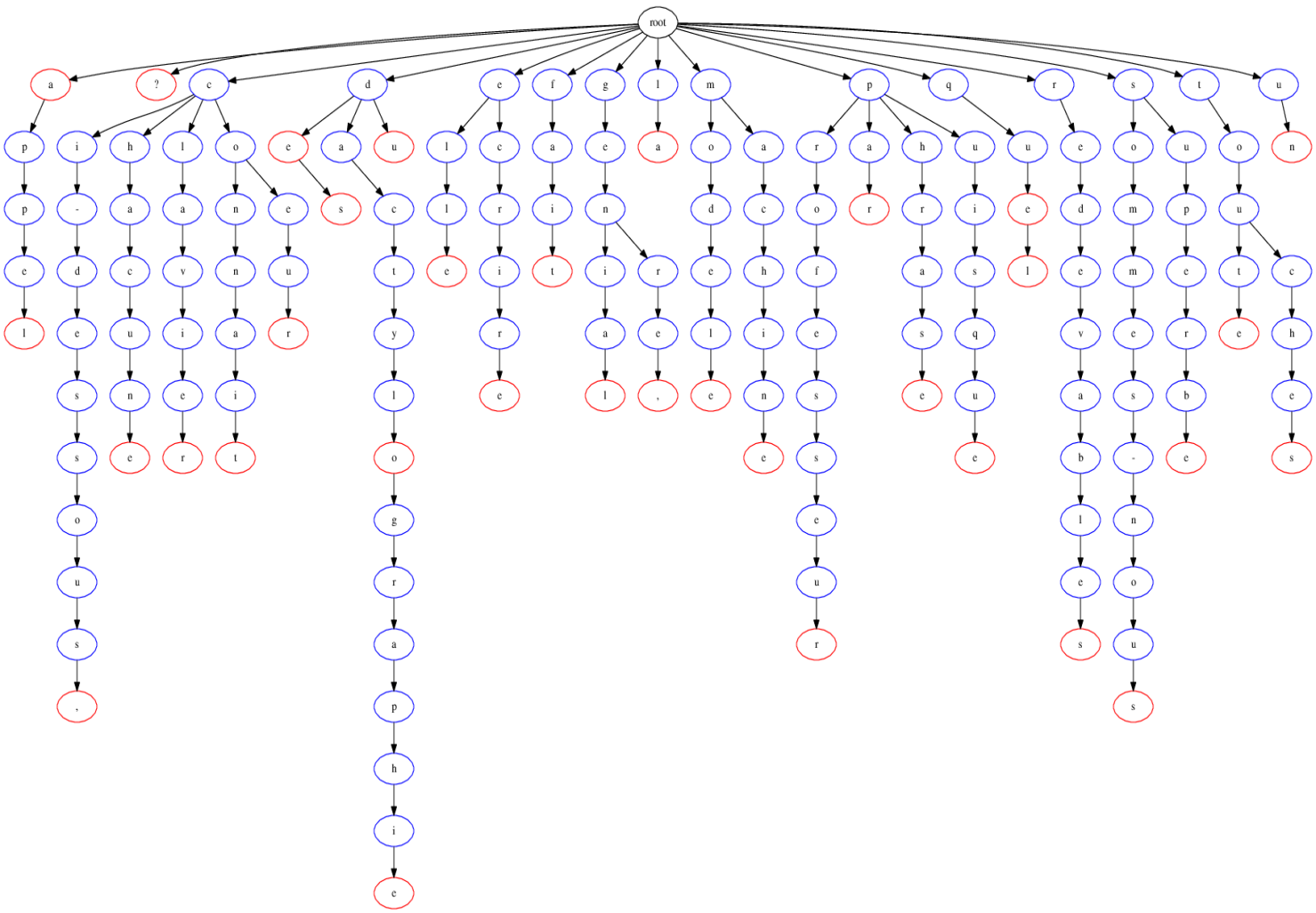


Figure 1. Représentation schématique de l'arbre de la Briandais obtenu par ajout successif de tous les mots de l'`exemple de base`.

Les nœuds entourés en rouge représentent la fin d'un mot, tandis que les bleus sont des nœuds à l'intérieur de mots.

Il faut noter que cette représentation visuelle diffère un peu de la représentation interne : en interne, les frères sont dans une liste chainée, alors que sur le schéma, ils sont représentés comme étant les fils d'un même nœud.

B- Trie Hybride

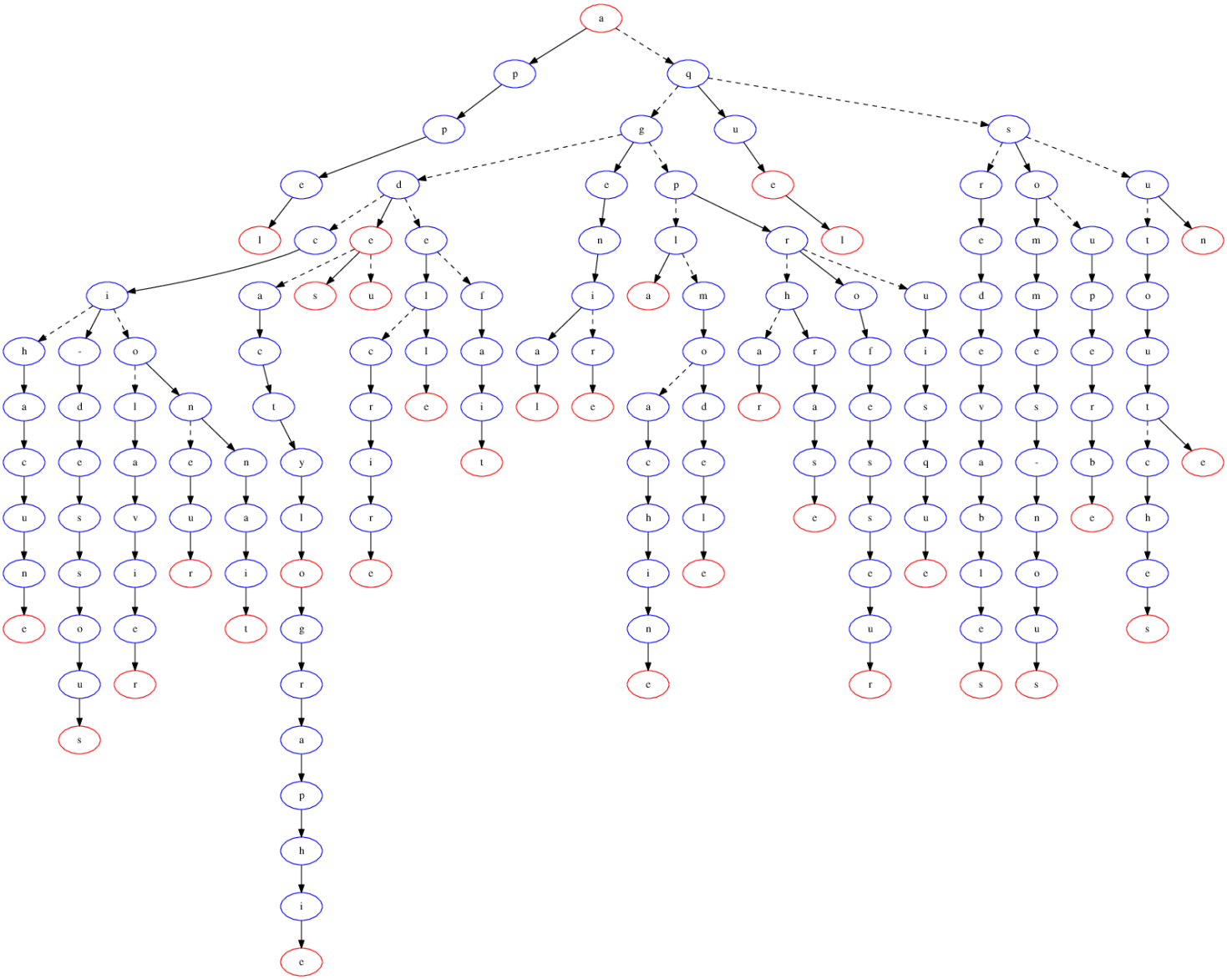


Figure 2. Représentation schématique du Trie Hybride obtenu par ajout successif de tous les mots de l'`exemple de base`.

Les nœuds entourés en rouge représentent la fin d'un mot, tandis que les bleus sont des nœuds à l'intérieur de mots.

Les liens en pointillés représentent les pointeurs sur les sous-arbres gauches et droits, tandis que les liens pleins représentent les pointeurs vers les sous-arbres centraux.

Remarque :
Ce schéma ne fait pas ressortir le déséquilibre de l'arbre au premier coup d'œil. En regardant bien, on constate cependant que le nœud-racine n'a pas de fils gauche par exemple. Nous reviendrons sur ce déséquilibre plus tard.

III. Fonctions avancées

1. ***Recherche*** : *Arbre * Mot -> Booléen*

A- Arbre de la Briandais

Le principe de la recherche d'un mot dans l'arbre de la Briandais est :

`Si le mot est vide, on renvoie` **`'Vrai'`**`.`
`On recherche la première lettre du mot dans la liste des frères du premier nœud.`
`Si on la trouve, on reprend la même recherche dans le fils du nœud trouvé, avec la lettre suivante.`
`Sinon on renvoie` **`'Faux'`**`.`

Complexité :

On parcourt donc au plus autant de niveau dans l'arbre que le mot que l'on recherche a de lettres. Et à chaque niveau, on effectue au plus 27 comparaisons de caractères.
La recherche d'un mot de m lettres est donc en $O(27m) = O(m)$.

B- Trie Hybride

Le principe de la recherche d'un mot dans un Trie Hybride est :

`Si le mot est vide, on renvoie` **`'Vrai'`**`.`
`Si la lettre de la racine est la première lettre du mot, on descend dans le sous-arbre du milieu, et on répète l'opération en passant à la lettre suivante du mot.`
`Sinon, si la lettre est plus petite (resp. Plus grande) que celle de la racine, on passe au sous-arbre gauche (resp. droit), sans passer à la lettre suivante du mot, puis on recommence.`
`Si l'on arrive à un point où l'on veut descendre dans un sous-arbre, mais qu'on ne peut pas car celui-ci est vide, on renvoie` **`'Faux'`**`.`

Complexité :

Le pire cas est celui d'un arbre où les mots insérés l'ont été dans l'ordre alphabétique, ce qui aura pour conséquence qu'aucun nœud n'aura de fils gauche, mais uniquement des fils droits. Dans ce cas, pour chaque lettre du mot, on effectuera au plus 27 comparaisons de caractères.
La recherche d'un mot de m lettres est donc en $O(27m) = O(m)$.

2. ***ComptageMots*** : *Arbre -> Entier*

A- Arbre de la Briandais

Le principe du comptage de mot consiste en un parcours complet de l'arbre, lors duquel on compte le nombre de champ *final* à vrai :

`Depuis chaque nœud (la racine en premier), si le champ final vaut` **`'Vrai'`**`, on ajoute 1 au total. On appelle ensuite récursivement la fonction sur son frère et son fils (s'ils existent).`

Complexité :

On passe une fois dans chaque nœud, et on effectue 3 tests : on compare le champ *final* à 'True', et on compare les champs *child* et *brother* à 'None'. Le nombre de nœuds de l'arbre est dans le pire des cas le n, nombre de caractères total ajouté dans l'arbre (dans le cas où tous les mots ajoutés n'ont aucun préfixe commun).
La complexité est donc en $O(3n) = O(n)$, avec n le nombre de caractères ajouté dans l'arbre au total.

B- Trie Hybride

Cette fois aussi, il suffit d'un parcours complet du Trie Hybride, lors duquel on compte le nombre de champ *final* à vrai :

`Depuis chaque nœud (la racine en premier), si le champ final vaut` **`'Vrai'`**`, on ajoute 1 au total. On appelle ensuite récursivement la fonction aux sous-arbres gauche, droit, et central (s'ils existent).`

Complexité :

On passe une fois dans chaque nœud, et on effectue 4 tests : on compare le champ *final* à 'True', et on compare les champs *left*, *right* et *middle* à 'None'. Le nombre de nœuds de l'arbre est dans le pire des cas le *n*, nombre de caractères total ajouté dans l'arbre (dans le cas où tous les mots ajoutés n'ont aucun préfixe commun).

La complexité est donc en O(4*n*) = O(*n*), avec *n* le nombre de caractères ajouté dans l'arbre au total.

3. ***ListeMots*** *: Arbre -> Liste[Mots]*

Pour les deux structures, la liste des mots se fait grâce à un parcours semblable à celui effectué pour compter les mots. Il suffit juste de retenir un préfixe auquel on ajoute la lettre courante lorsque l'on descend en profondeur (fils pour les arbres de la Briandais, et sous-arbre central pour les Trie Hybrides), afin de construire des mots au fur et à mesure. A noter donc que lorsque l'on passe au frère pour un arbre de la Briandais, ou dans les sous-arbres gauches et droits pour les Trie Hybrides, on ne rajoute pas la lettre du nœud courant au préfixe (c'est assez logique compte tenu de la structure de l'arbre).

La complexité de cette fonction est la même que celle de la fonction *ComptageMots*, soit en O(*n*) pour les deux structures, avec n le nombre total de caractères ajoutés.

4. ***ComptageNil*** *: Arbre -> Entier*

Le principe de cette fonction va être le même que la fonction de comptage du nombre de mots.
Pour les arbres de la Briandais, les pointeurs vers Nil seront dans des nœuds qui n'ont pas de frère ou bien pas de fils.
Quant aux Trie Hybrides, les pointeurs vers Nil seront dans des nœuds qui n'ont soit pas de sous-arbre gauche, droit, ou central.

La complexité de cette fonction est la même que la fonction *ComptageMots*, soit en O(*n*) pour les deux structures, avec n le nombre total de caractères ajoutés.

5. ***Hauteur*** *: Arbre -> Entier*

Le calcul de la hauteur d'un arbre se fait par un parcours du même genre de celui des fonctions précédentes :
Pour les arbres de la Briandais, la hauteur d'un nœud est égale au max de la hauteur de son frère et de 1 + la hauteur de son fils.
Pour les Trie Hybrides, la hauteur est égale à 1 + le max de la hauteur des sous-arbre gauches, droits et centraux.

La complexité reste le même que pour les fonctions précédentes : O(*n*) pour les deux structures, avec n le nombre total de caractères ajoutés.

6. ***ProfondeurMoyenne*** *: Arbre -> Entier*

Cette fonction calcule simplement la profondeur de chaque feuille puis en fait une moyenne. Ceci est fait avec toujours le même genre de parcours.

Sa complexité est en O(*n*) pour les deux structures, avec n le nombre total de caractères ajoutés.

7. ***Prefixe*** *: Arbre * Mot -> Entier*

Le principe du fonctionnement de cette fonction est le suivant : on effectue une recherche du mot dans l'arbre. Si on ne le trouve pas, on renvoie 0. Sinon, On renvoie le nombre de mots dans l'arbre duquel le nœud contenant la dernière lettre du mot est la racine, à l'aide de la fonction *ComptageMots*.

La complexité de cette fonction, qui effectue une *recherche* puis un *comptageMots* est également en O(*n*) avec *n* le nombre de caractères ajouté dans l'arbre au total.

8. ***Suppression*** *: Arbre * Mot -> Arbre*

Cette fonction va devoir rechercher le mot, et supprimer les nœuds qui le composent à condition qu'ils ne fassent pas partie d'un autre mot.

Sa complexité est en O(*m*), avec *m* le nombre de lettres du mot que l'on veut supprimer.

IV. Fonctions complexes

Fusion de deux arbres de la Briandais

La fusion de deux arbres de la Briandais consiste en un parcours simultané des deux arbres. On compare dans un premier temps les deux nœuds 'de départ' de chacun des deux arbres. On compare les clefs de ces nœuds, puis :

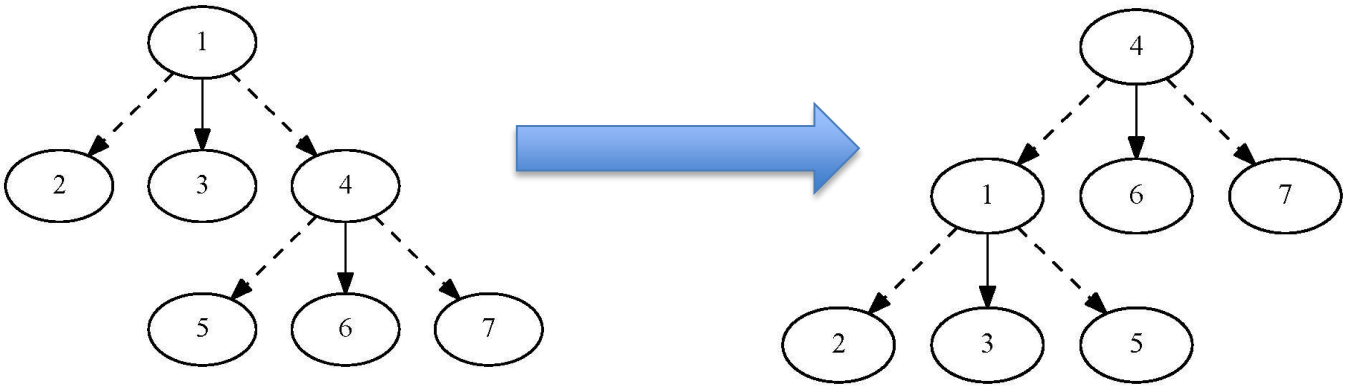
- On fusionne récursivement les fils des deux nœuds si les clefs sont identiques, puis on fusionne récursivement les frères des deux nœuds.
- On fusionne récursivement le frère du nœud le plus petit ainsi que le nœud le plus grand, si les clefs sont différentes.
- On garde le reste de l'arbre si l'un des nœuds est nul.

La récursion conserve un arbre trié, et assure simultanément une fusion des fils et des frères.
Exemple : On applique la fonction Fusion sur deux arbres A et B, contenant pour le premier 'alors' et 'bete', et pour le deuxième 'arbre'. On compare le premier nœud de A et B. Les clefs sont identiques (la lettre 'a'). On garde alors l'un des deux nœuds, et l'on applique Fusion sur les fils de A et de B (qui deviennent les fils du nœud que l'on garde). On applique ensuite Fusion sur le frère de A et de B, qui devient alors frère du nœud que l'on garde. Cette fusion compare un nœud plein (clef : 'b') avec un nœud nul : on conserve donc le nœud plein.
Si les deux arbres contiennent respectivement *m* et *n* nœuds, cette fonction est en Θ(*m*+*n*).

Trie Hybride équilibré

On peut maintenir un Trie hybride équilibré à tout moment en utilisant le même principe de rotation que pour les AVL : si la différence de hauteur de sous arbres gauches et droits est supérieure à 1, on effectue une rotation (gauche si le sous-arbre droit est plus profond, et droite sinon).
Il faut noter que le sous-arbre du milieu n'intervient pas dans le calcul de la hauteur : un mot n'ayant aucun préfixe commun avec un autre entrainera forcément un arbre d'une profondeur de la taille de ce mot, et aucune rotation ne pourrait changer ça.

La figure suivante montre une rotation gauche :



Les rotations peuvent augmenter un peu le nombre d'opérations nécessaires à l'insertion d'un mot, mais la complexité de l'insertion d'un mot reste cependant la même, à savoir O(*m*) (avec *m* le nombre de lettres du mot). De plus, la complexité de la recherche d'un mot dans le pire des cas reste également la même. En effet, même si une lettre plus 'centrale' de l'alphabet devrait se trouver sur tous les nœuds ayant des sous-arbres gauches et droits, on ne peut pas conclure de la présence d'un mot avant avoir dans certains cas parcouru un certain nombre de nœuds en suivant des pointeurs *left* ou *right*. Ainsi, la complexité dans le pire des cas de la recherche reste en O(*m*), avec *m* le nombre de lettre du mot.
Cependant, en moyenne, l'arbre équilibré devrait diviser par presque 2 le nombre de pointeurs *left* ou *right* suivis. La complexité théorique reste donc la même, mais la complexité pratique est presque 2 fois meilleure.

Les deux figures suivantes montrent les différents arbres obtenus en faisant des insertions sans équilibrage (figure 4), et avec équilibrage (figure 5). On constate que l'arbre équilibré est naturellement légèrement moins profond, et qu'il contient moins de nœuds ayant un fils droit mais sans fils gauche ou l'inverse. Cependant, comme expliqué précédemment, le fait que le sous-arbre du milieu n'intervient pas dans le calcul de la hauteur laisse un déséquilibre apparent dans l'arbre, mais on ne peut pas obtenir de meilleurs résultats.

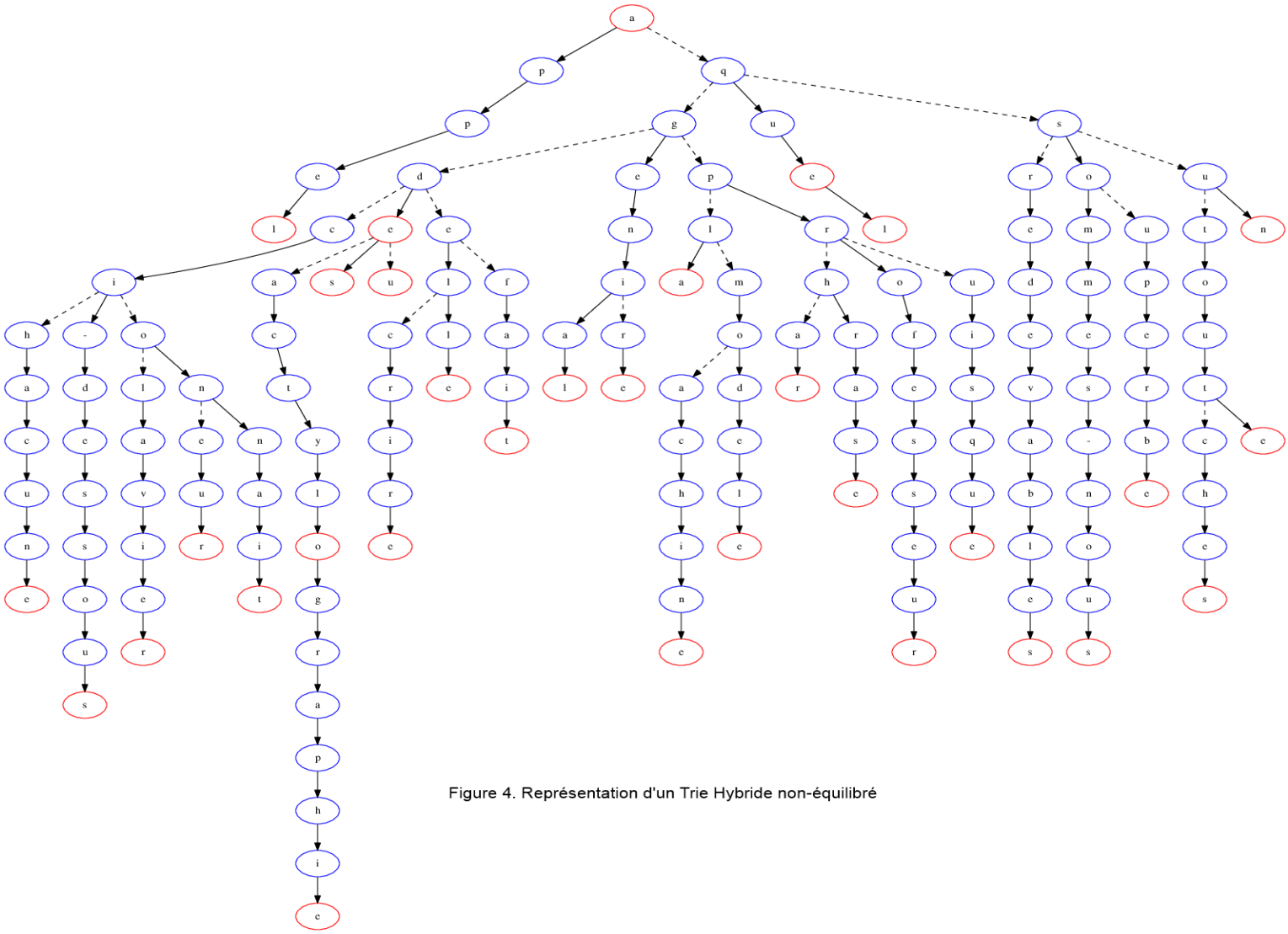


Figure 4. Représentation d'un Trie Hybride non-équilibré

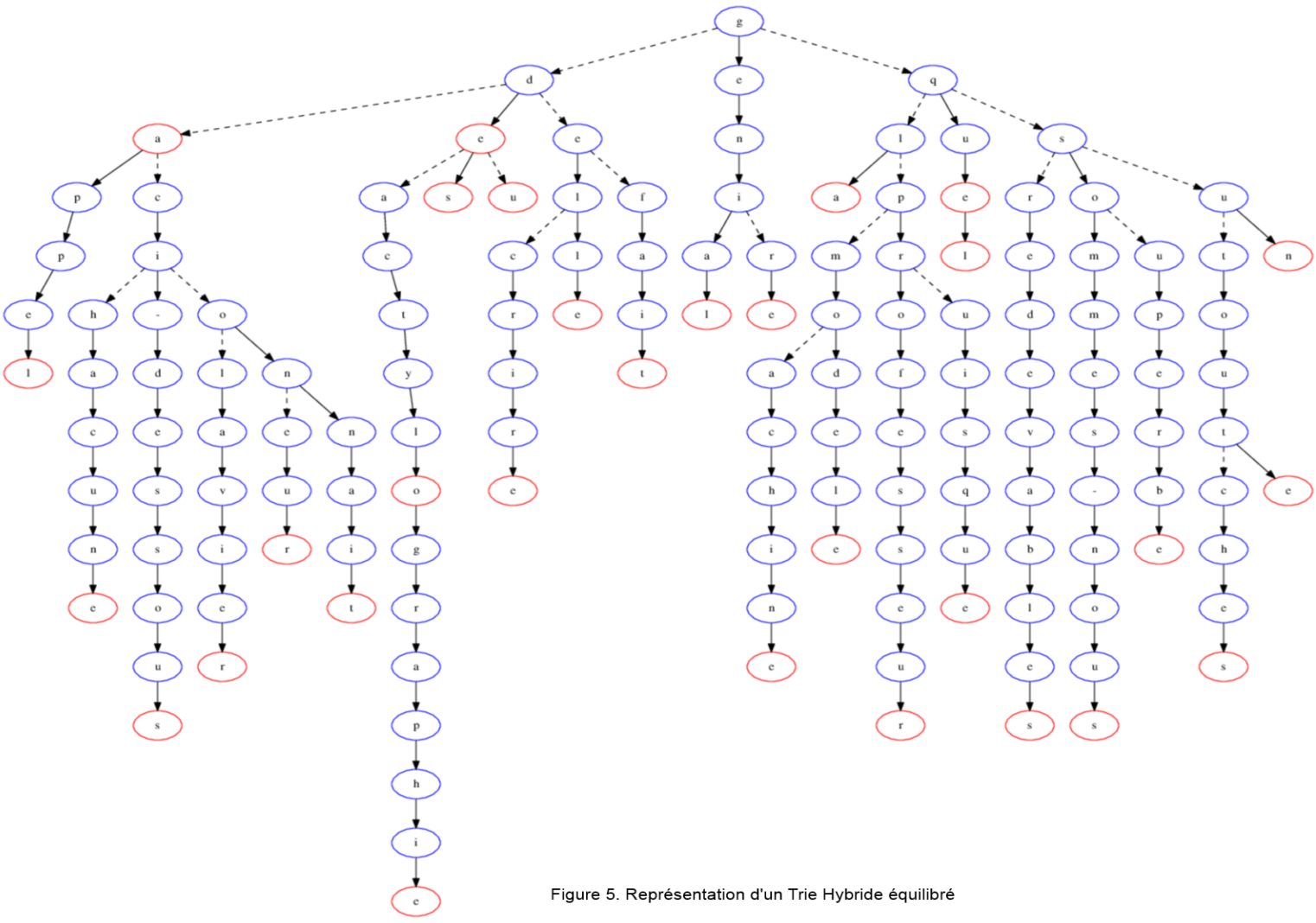


Figure 5. Représentation d'un Trie Hybride équilibré

V. Etude expérimentale.

L'étude expérimentale a été faite en manipulant, avec les deux structures différentes, l'ensemble des mots de l'oeuvre de Shakespeare.
Vous pouvez retrouver la plupart de valeurs grace à la fonction *shakespeare_test* du fichier tests.py. (Elle devrait s'exécuter en moins d'une minute sur une machine raisonnable).
Le comportement par défaut du fichier tests.py est de lancer une fonction de tests des fonctions de base sur l'exemple de base, puis de lancer la fonction *shakespeare_test*. Il suffit donc d'exécuter le fichier tests.py si vous souhaiter constater par vous-même ces résultats.

Voici donc les résultats obtenus.

- Temps d'ajout des mots :
  - Briandais : 18 s
  - Trie Hybride : 10 s
- Temps de suppression :
  - Briandais : 0,3 s
  - Trie Hybride : 0,4 s
- Nombre de pointeurs Nil :
  - Briandais : 56 392
  - Trie Hybride : 112 783
- Hauteurs :
  - Briandais :
    - Max : 27
    - Moyenne : 7
  - Trie Hybride :
    - Max : 36
    - Moyenne : 18

Nous pouvons voir que le temps d'ajout global est environ deux fois plus long pour un arbre de la briandais. En effet, un trie hybride a tendance à regrouper les mots ayant un préfixe commun de manière plus efficace : à l'aide des comparaisons, l'arbre ternaire détermine plus rapidement le préfixe commun. En effet, si l'on tendance d'insérer une des dernières lettres de l'alphabet (par exemple 'z'), nous allons devoir parcourir toute la liste chaînée des frères afin de savoir si la lettre existe ou non. Un arbre ternaire équilibré devrait normalement pouvoir le déterminer en temps logarithmique. L'insertion est donc plus rapide sur un arbre ternaire de recherche que pour un arbre de la Briandais.

Les temps de suppressions sont quasiment équivalents : en effet, dans notre implémentation en Python, la suppression consiste à passer le pointeur vers l'objet à 'None'. L'objet est donc perdu pour l'utilisateur. Le garbage collector de Python s'occupera, lorsque le besoin se fera sentir (ou de manière discrète, lors d'un temps d'attente d'entrée utilisateur par exemple), de vider la mémoire inutilisé. Passer le pointeur à 'None' est donc effectivement une suppression, puisque les données sont perdues et la mémoire virtuellement accessible. Mais nous n'avons aucune certitude quant à l'instanéité de la suppression. Afin d'avoir des valeurs correctes, il faudrait utiliser un langage s'assurant de vider la mémoire lors de la perte d'un pointeur (à l'aide de Pointeurs Intelligents, ou en gérant manuellement la mémoire, comme en C).

Le nombre de pointeurs Nil varie d'un facteur 2 entre les deux arbres. Un trie hybride comprendra 2 foix plus de pointeurs nuls qu'un arbre de la Briandais.
En effet, chaque nœud d'un Trie Hybride a un pointeur vers un sous-arbre droit, et un autre vers le gauche, tandis qu'un nœud d'un arbre de la Briandais n'a qu'un seul pointeur vers l'équivalent, à savoir vers son frère. Ce qui veut dire que les nœuds qui n'ont pas de frère dans le cas des arbres de la Briandais, ou pas de sous-arbres gauches/droits dans le cas des Trie Hybrides, et qui représentent la majorité des nœuds (il s'agit en effet de tous les nœuds n'étant pas dans un préfixe, ou bien tous les nœuds dans un préfixe mais n'étant pas au point où les mots se 'séparent' sur ce préfixe), auront 2 pointeurs Nil dans le cas des Trie Hybrides, et seulement 1 dans le cas des arbres de la Briandais.

Enfin, les hauteurs moyenne des arbres : en moyenne un arbre de la Briandais est nettement moins, avec 7 niveaux, là où un trie hybride en fait 18.
Dans un arbre de la Briandais, la profondeur du dernier nœud d'un mot est simplement le nombre de lettres de ce mot. La hauteur moyenne des feuilles d'un arbre de la Briandais est donc la longueur moyenne des mots de l'œuvre de Shakespare.
En revanche, dans les Trie Hybrides, les descentes dans les sous-arbres gauches ou droits entrainent une profondeur plus importante, même deux fois plus importante comme les montrent les chiffres obtenus.

Aperçu global et visuelle des structures

Vous avez déjà pu constater au cours de ce rapport le rendu des fonctions d'affichage graphique.
Nous nous sommes basés sur un outil open source appelé GraphViz. Contrairement à ce que le nom pourrait laisser penser, il est assez aisé de tracer des arbres avec ce logiciel : il suffit de créer un fichier .dot, où l'on définit les noms des nœuds, puis les relations entre les nœuds. GraphViz déduit de ce fichier la hiérarchie entre les nœuds et trace les arbres que vous avez pu voir.

Pour pouvoir nous servir de GraphViz dans notre cas, il nous a suffi de parcourir l'arbre de la Briandais ou le Trie Hybride, et de créer au fur et à mesure un fichier contenant les nœuds et les liens entre eux.

Il faut noter que forcer la direction d'un lien (c.-à-d. forcer un lien à aller vers la gauche, ou bien tout droit vers le bas ou bien sur la droite) est assez complexe, voir pas faisable selon les cas. De plus, tracer des liens horizontaux requière de regrouper les nœuds au même niveau dans un cluster, ce qui une fois de plus, peut être délicat. Par conséquent, nous avons choisis de représenter les structures avec les différences à l'implémentation réelle suivantes :

- pour les arbres de la Briandais, les frères ne sont pas représentés comme une liste chainée, mais comme étant tous les fils d'un même père.
- pour les Trie Hybrides, les arcs ne sont pas nécessairement dans la bonne direction mais ont les caractéristiques suivantes : les arcs vers les sous-arbres gauches et droits sont représentés en pointillé, et les arcs vers les sous-arbres centraux sont pleins.

Ceci nous a permis de représenter même les arbres contenant l'ensemble des mots de l'œuvre de Shakespeare. Ils ne sont pas présents dans ce rapport car leur taille est trop importante, et leur format peu adapté (ces arbres sont peu profond (pas au-delà de 36 de profondeur), mais très larges : plusieurs centaines voire milliers de nœuds de largeur au maximum).
(Décommentez les lignes 101 et 142 du fichier tests.py pour pouvoir observer ces arbres. Leur génération peut prendre quelques minutes.)