Rapport Projet ALGAV 2016

Par

Marin Dupas

et

Alexandre Gaspard Cilia

Sommaire

1 Présentation

1.1 Structure 1 : Patricia-Tries

Question 1.1

Parmi les 128 caractères du code ASCII, le troisième semble parfaitement correspondre à ce que nous cherchons. Il est désigné comme caractère “End Of Text (EXT)”, “fin de texte” en français. Cependant dans mon implémentation, un PatriciaTrie possède un attribut de type booléen faisant office de caractère final.

Question 1.2

Les primitives sont les suivantes :

ajouterMot(p:PatriciaTrie, mot:String):PatriciaTrie

estFinal(p:PatriciaTrie):Booléen

setSubword(sw:String)

getSubword():String

getTries(p:PatriciaTrie):list[PatriciaTrie] retourne les “fils” de p.

Question 1.3

La phrase ne contient pas de caractère qui n’est pas dans la table ASCII. Donc tout les accents de la phrase ont été retirés.

1.2 Structure 2 : Tries Hybrides

Question 1.4

ajouterMot(h : TriesHybrides, mot : String) : TriesHybrides

suppression(h : TriesHybrides, mot : String) : TriesHybrides

recherche(h : TriesHybrides, mot : String) : boolean

estVide(h : TriesHybrides) : boolean

Question 1.5

cf : Main.java

2 Fonctions avancées pour chacune des structures

Question 2.6

cf. code source du projet.

3 Fonctions complexes

Question 3.7

Pour fusionner deux Patricia je compare tous les fils des deux arbres entre eux pour vérifier s'ils ont un préfixe commun. J'ajoute au résultat tous ceux qui n'ont rien en commun avec personne. En ce qui concerne les autres, j'identifie trois façons de les fusionner :

Si leurs préfixes sont identiques, je lance simplement la fusion récursivement sur les deux tries.

Si leur préfixe commun est plus petit que la taille des préfixes des tries, je crée un nouveau Trie qui a pour "subword" leurs préfixe en commun qui aura comme "fils" les deux anciens nœuds délestés de leur préfixe commun.

Et enfin, si un des deux Trie possède le préfixe commun en tant que "subword", j'ajoute un nouveau Trie qui a pour "fils" les deux anciens Tries délestés encore une fois du préfixe commun.

Question 3.8

Pour convertir un trie Hybride en Patricia trie, il faut simplement parcourir les fils "eq" jusqu'à tomber sur un nœud dont "inf" ou "sup" ne soit pas égale à null et d'ajouter le résultat de la conversion de "inf", "eq" et "sup " à la liste de fils du résultat. Pour plus de précision cf. code source du projet (PatriciaTries.fromHybride()).

Question 3.9

Pour rendre un arbre mieux équilibré, l'outil clé est la rotation : si un fils inférieur ou supérieur a un poids plus important que les autres fils (incluant le fils égal), on peut effectuer une rotation permettant de remplacer le père par un de ces fils et d'ajouter l'ancien père comme nouveau fils du nouveau père.

On peut imaginer, comme seuil à atteindre avant d'effectuer une rotation, que le nombre de mots d'un des fils représente plus de la moitié des mots auquel le père à accès. En effet, si une seule rotation est effectuée, le 2^e fils de l'ancien père serait pénalisé par cette rotation (il serait, après la rotation, toujours un fils de l'ancien père, donc sa hauteur

augmenterait de 1). Le défaut d'appliquer strictement ce seuil est la possibilité que l'ajout d'un mot dans la partie de l'ancien père, la fasse redevenir la partie possédant la moitié des mots et que des rotations successives soit effectuées.

4 Complexités

Question 4.10a

Patricia trie :

Les complexités calculées correspondent au pire cas et sur le nombre de comparaisons de String. m est le nombre de "fils" maximum des nœuds soit la taille de l'alphabet supporté et n le nombre de nœuds total.

ajouterMot(p:PatriciaTrie, mot:String) : PatriciaTrie

O(m log n), car il faut chercher le "fils" ayant un préfixe commun avec le mot à rechercher. Cette liste de fils peut contenir au maximum m tries.

suppressionMot(p:PatriciaTrie, mot:String) : PatriciaTrie

O(m log n), cette fonction est très semblable à l'ajout de mot.

recherche(p:PatriciaTrie, mot:String) : Boolean

O(m log n), cette fonctionne est aussi semblable à l'ajout de mot.

comptageMot(p:PatriciaTrie) : Integer

O(n), parcours tous les nœuds à la recherche d'indicateurs de fin de mot.

listeMots(p:PatriciaTrie) : Tableau[String]

O(n), semblable au comptage de mot mais retourne le mot quand il trouve un indicateur de fin de mot.

comptageNil(p:PatriciaTrie) : Integer

O(n), parcours aussi tous les nœuds. Un nil est représenté par une liste de "fils" vide ou null dans notre implémentation.

hauteur(p:PatriciaTrie) : Integer

O(n), tout l'arbre est parcouru à la recherche du nœud le plus "haut".

profondeurMoyenne(p:PatriciaTrie): Double

O(n), fait une moyenne de la profondeur de tous les nœuds feuille.

prefixe(p:PatriciaTrie, mot:String) : Integer

O((m log n) + l), où l correspond au nombre de nœuds de l'arbre préfixe du mot recherché. Ici, l'algorithme cherche le mot et retourne le nombre de mots contenu dans l'arbre résultat.

fusion(p:PatriciaTrie, q:PatriciaTrie) : PatriciaTrie

O(n * n'), dans le pire cas il y conflit entre les nœuds des arbres p et q à chaque comparaison à tous les niveaux de profondeur ce qui peut obliger l'algorithme à parcourir entièrement les deux arbres. Ce cas peut se produire si les deux arbres sont identiques par exemple.

Question 4.10b

Tries Hybrides :

Les complexités calculées correspondent au pire cas, dans un arbre contenant déjà *n* nœuds et d'une hauteur max de *h* en fonction du nombre de comparaison (de caractères, vérification d'un pointeur null).

h = (ln (nb de caractères différents dans le TriesHybrides)) * (nb de caractères du mot d'hauteur max).

Par exemple, si dactylo est le mot le plus long et qu'il existe 26 caractères possible après chacun des lettres, h = ln(26) * 7 = 5 * 7 = 35.

ajouterMot(h : TriesHybrides, mot : String) : TriesHybrides

O(5 * h) = O(h) : 5 comparaisons (au pire) sont effectuées à chaque nœud traversé (1 pour vérifier si l'arbre est vide, 2 pour connaître le fils, 1, si l'ajout se fait dans le fils égal, pour savoir si la fonction est terminal et 1 pour vérifier si le fils est null) et ceci h (+ taille du mot, si aucune lettre n'a été rentrée) fois.

suppression(h : TriesHybrides, mot : String) : TriesHybrides

O(h) : semblable à ajouterMot, 7 comparaisons (au pire) possible et cela h fois.

recherche(h : TriesHybrides, mot : String) : boolean

O(h) : semblable à ajouterMot, 4 comparaisons (au pire) possible et cela h fois.

comptageMot(h : TriesHybrides) : integer

O(n) : 4 comparaisons sont effectuées pour chaque nœud du TriesHybrides.

listeMots(h : TriesHybrides) : String[]

O(n) : pour le parcours de l'arbre, avec 4 comparaisons (semblable à comptageMot). Seulement, il faut rajouter le coût de la reconstruction des mots : une concaténation est effectuée pour chaque caractère de chaque mot (sans compter la recopie des listes des fils dans la liste du père dans l'algo utilisé). Soit *c*, le nombre total de caractères utilisés dans cette liste, le coût total de listeMots serait de O(n + c).

comptageNil(h:TriesHybrides) : integer

O(n) : semblable à comptageMot, 4 comparaisons (au pire) possible et cela n fois.

hauteur(h : TriesHybrides) : integer

O(n) : semblable à comptageMot, 4 comparaisons (au pire) possible et cela n fois.

profondeurMoyenne(h : TriesHybrides) : integer

O(n) : semblable à comptageMot, 3 comparaisons et cela n fois.

prefixe(h : TriesHybrides, mot : String) : integer

Soit h' la hauteur à laquelle se trouve le mot-préfixe, et n' le nombre de nœuds dans l'arbre contenant le mot-préfixe.

O(h' + n') : algo de recherche + algo de comptageMots.

5 Étude expérimentale

Question 5.11

cf. shakespeareTest.jar exécutable à l'aide du scripte fourni dans le répertoire "runnable".

Question 5.12

idem que pour 5.11.

Question 5.13

Question 5.14

Dans les illustrations 1 et 2 nous pouvons voir une représentation graphique de nos deux structures. Les deux contiennent les mots "autre", "test", "tester", "table", "tete", "testeurs" et "testeur". Veuillez noter que "#0" indique que le nœud se trouve au niveau 0 de l'arbre. En ce qui concerne le Patricia Trie, la présence de "FIN" signifie que le nœud est considéré comme final, c'est à dire que sont indicateurs de fin est à "Vrai" ou qu'il possède un "fils" vide avec un indicateur de fin à "Vrai". En revanche la présence de ":1" sur un nœud de la

représentation graphique du trie Hybride indique que ce premier est un nœud final d'un mot. Le nombre après ":" correspond à clef du mot.

Les représentations graphiques de ces deux structures permettent de mettre en évidence que le Patricia trie est beaucoup moins profond que le trie Hybride. Ce qui, en théorie, permet de rendre les recherches dans les Patricia tries plus rapides. D'autre part, nous pouvons observer d'un Patricia trie prend moins de place (= moins de nœuds).

6 Annexe

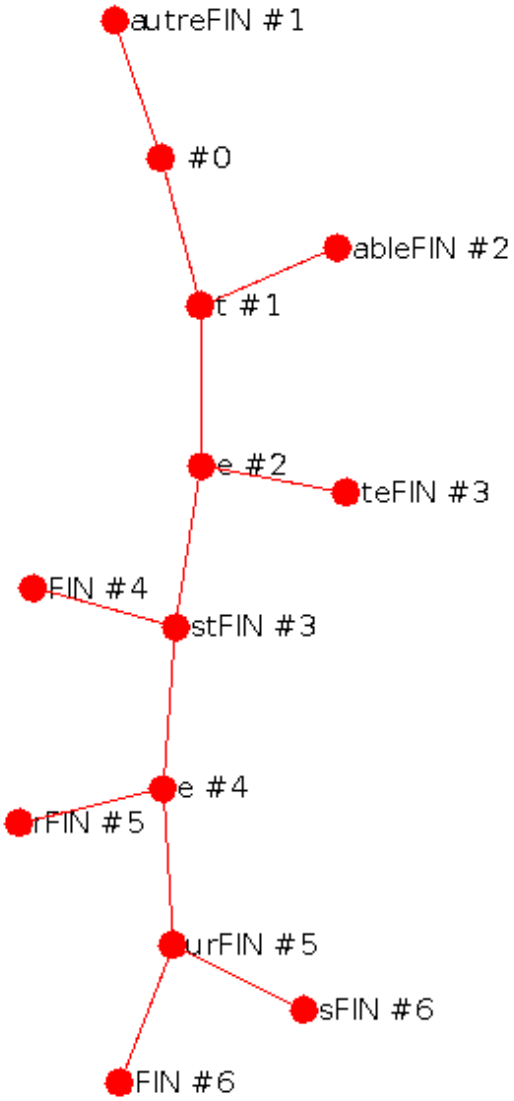

Illustration 1: Patricia trie

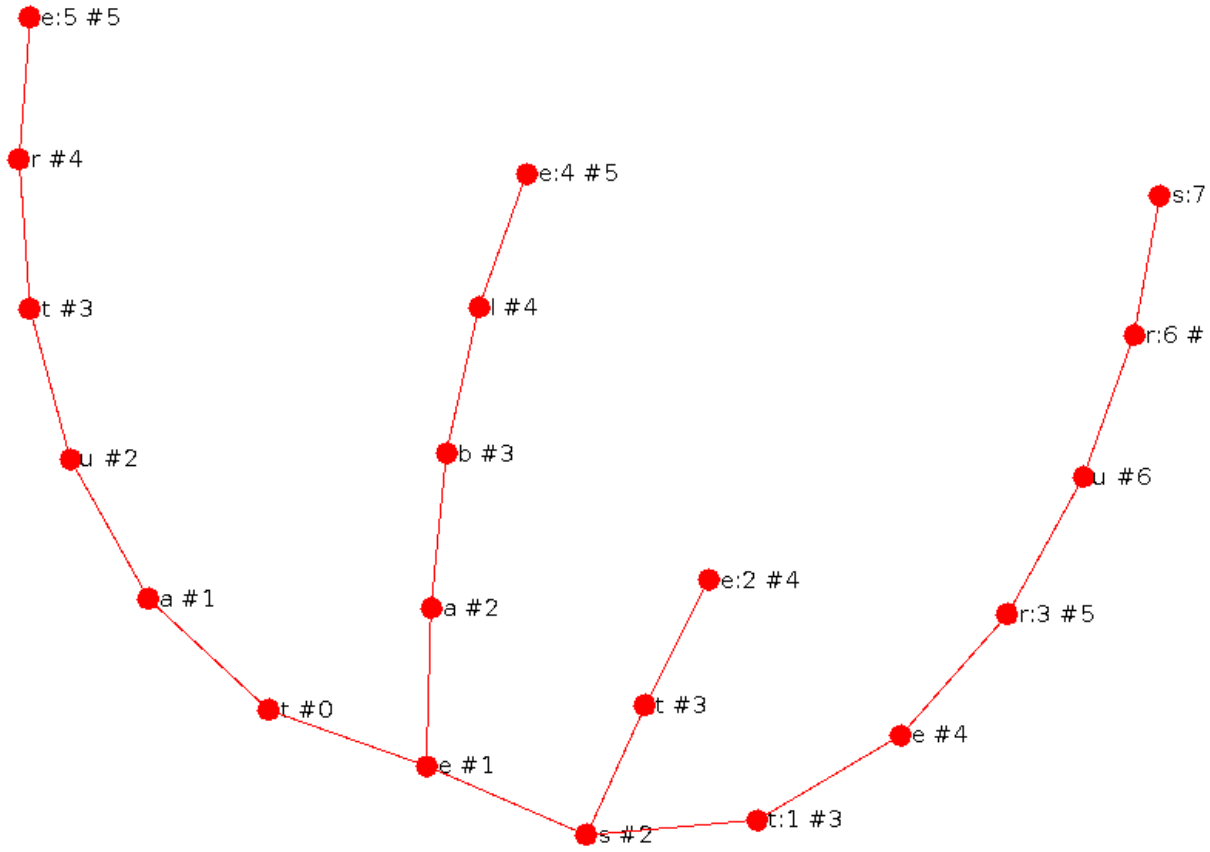

Illustration 2: trie Hybride