Algorithmique Avancée

Devoir de Programmation :

Tries

Guittonneau Nicolas

Kruissel Rudy

I - Présentation du projet:

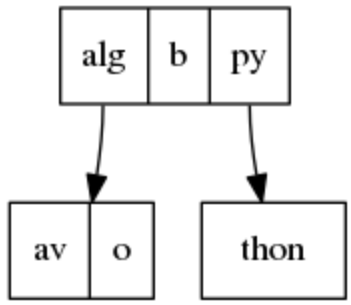
1 - Langages utilisés

Nous avons pris la décision d'utiliser le langage python pour coder l'ensemble du projet, sa syntaxe simpliste permet de faire beaucoup avec peu de lignes de codes (et force la lisibilité), sa gestion de la mémoire et des pointeurs est totalement transparente, et est en plus orienté objet. En revanche, du fait qu'il soit interprété, ses performances sont nettement moindre vis-à-vis de ses homologues compilés.

Pour afficher les deux structures d'arbre, nous utilisons le langage de description de graphe DOT (outil de graphviz) qui va s'occuper de tout l'agencement de l'arbre à partir d'une simple description.

Exemple:

```
digraph graphname { node [shape=record];
node1[label = "<f1> alg|<f2> b|<f3> py"];
node2[label = "<f1> av|<f2> o"];
node3[label = "<f1> thon"];
node1:f1 -> node2
node1:f3 -> node3
}
```



2 - Structure 1 : Patricia Trie

L'objet PatriciaTrie est composé d'un tableau de string "Key" qui fait la taille de l'alphabet, ainsi que d'un second tableau de références vers d'autres objets PatriciaTrie "children", de taille identique au premier. De plus, l'objet contient un entier "WordCount" mis à jour si nécessaire à chaque modification de l'arbre.

La caractère de fin de mot choisi est l'underscore '_' non présent dans l'alphabet et visuellement intéressant pour les affichages. Il n'est pas possible de choisir '\0' qui d'une part, n'est pas imprimable mais en plus est déjà utilisé pour détecter le fin d'une chaîne de caractère.

3 - Structure 2 : Hybrid Trie

La structure du Trie Hybride comporte un tuple "Key" de type [char, int] qui contient le caractère du noeud et un int correspondant à l'identifiant du mot se terminant à ce noeud, laissé à None si le noeud n'est pas une fin de mot; un tableau de 3 références vers 3 autres Tries Hybrides : [FilsGauche, FilsCentral, FilsDroit]; un int WordCount, contenant le nombre de mot contenus dans l'arbre.

Sur l'affichage des Tries Hybrides, une fin de mot est indiquée par une case rouge.

II - Execution

Pour exécuter les fonctions il suffit simplement d'exécuter les fonctions du jeu de test mis à disposition en exécutant une ou plusieurs des fonctions suivantes, toutes les fonctions sont écrites dans la section MAIN du code, il suffit juste de les dé-commenter (les noms sont plutôt explicites) :
-Création des arbres et affichage avec la phrase d'exemple
    PatriciaTrieExample()
    HybridTrieExample()
-Création des arbres et affichage avec tous les fichiers Shakespeare (les fichiers doivent se trouver dans un sous répertoir "Shakespeare") ⚠les temps d'exécution peuvent être longs
    PatriciaTrieAllFiles()
    HybridTrieAllFiles()
-Création des arbres et affichage avec Shakespeare/comedy_errors.txt (ce fichier est le plus léger)
    PatriciaTrieOneFile()
    HybridTrieOneFile()
-Création des Patricia-Tries par fusions sur tous les fichiers
    PatriciaTrieMerging()
    PatriciaTrieMergingMultiProcessing()
-Création et affichage de l'Hybrid-Trie équilibré de la phrase de l'exemple
    HybridTrieBalancedExample()

Le temps d'exécution total est toujours affiché à la fin de l'exécution du programme.
Il est possible d'interrompre la création d'un pdf par un Ctrl+C si le temps d'exécution est trop long, la suite s'exécutera alors.

III - Réponse aux questions du projet

Question 3.7: Merge Patricia

Soit deux Patricia tries A et B, l'appel de la méthode `A.merge(B)` fusionne directement dans A le Patricia-Trie B.

L'algorithme parcourt chaque clef de A et la compare avec la clef du même indice de B, à partir de là nous avons 6 cas différents:

- B n'a pas de clefs à cet indice, on passe au tour de boucle suivant
- B contient une clef mais pas A, on copie la clef et son fils correspondant dans A
- A et B ont exactement la même clef, on appelle récursivement `fils(A).merge(fils(B))`

Si aucune des conditions précédentes n'est vérifiée, on calcule le préfixe commun des deux clefs ainsi que leur suffixe, l'on crée aussi un nouvel arbre vide C et on remplace la clef actuelle de A par le préfixe à la fin:

- Si la clef de A correspond au préfixe, on ajoute dans C le suffixe de B ainsi que le fils de B correspondant à l'indice actuel, on merge ensuite le fils de A avec C
- Si la clef de B correspond au préfixe, on ajoute dans C le suffixe de A ainsi que le fils de A correspondant à l'indice actuel, on merge ensuite le fils de B avec C, le résultat de cette fusion donnera le nouveau fils de A
- Si le préfixe forme une nouvelle clef, on ajoute dans l'arbre C les deux suffixes calculés auxquels on rattache à chacun leur fils respectifs, le nouveau fils de A à cet indice est C, pas d'appel récursif

Question 3.8: Conversion

Nous n’avons pas réussi à implémenter de façon récursive les conversion d’une structure vers l’autre, nous avons toutefois apporté une optimisation dans l’algorithme itératif de la conversion des Patricia Tries vers les Tries Hybrides. Afin que celui-ci soit équilibré nous ajoutons pour une liste de mots donnée, le mot au milieu de celle-ci, et appelons récursivement deux fois notre fonction avec respectivement la première et la seconde moitiée de la liste (tronquée du mot central). Il faut donc initialement appeler ListeMots() sur notre Patricia Trie.

Pour la conversion dans l’autre sens, l'ordre d’ajout dans un patricia n’ayant aucune répercussion sur sa structure interne, on ajoute chaque mots présents dans ListeMots().

Question 3.9: Equilibrage des Hybrid Tree

L’algorithme utilisé pour l'équilibrage est le même que celui décrit dans la conversion des Patricia vers les Hybrides

L’algorithme optimal aurait été de faire des rotations comme dans les AVL, en se basant non pas sur la hauteur des deux fils mais sur le nombre de mots qu’ils contiennent.

Question 4: Complexités

Soient :

- m le nombre total de mots
- n le nombre total de noeuds
- h la profondeur maximale
- c nombre total de caractères dans le mot
- a taille de l'alphabet
- k nombre de clefs communes aux deux arbres
- nil le nombre de feuilles

	Patricia Tries	**Hybrid Tries**
Ajout()	Dans le pire cas, il faudra parcourir un noeud pour chaque char du mot O(c)	Dans le pire des cas, il faudra atteindre la hauteur de l'arbre, puis ajouter le mot O(h + c)
Recherche()	Dans le pire cas, il faudra parcourir un noeud pour chaque char du mot O(c)	Dans le pire des cas, le mot est le mot le plus profond, on doit donc parcourir toute la hauteur O(h)
ComptageMots()	Stocké dans la structure O(1)	Stocké dans la structure O(1)
ListeMots()	Parcours de chaque noeud O(n)	Parcours de chaque noeud O(n)
ComptageNIL()	Parcours de chaque noeud O(n)	Parcours de chaque noeud O(n)
Hauteur()	Parcours de chaque noeud O(n)	Parcours de chaque noeud O(n)
ProfondeurMoyenne()	Parcours de chaque noeuds + calcul de la moyenne (une donnée par nil) O(n + nil)	Parcours de chaque noeuds + calcul de la moyenne (une donnée par nil) O(n + nil)
Prefixe()	Revient à faire une recherche suivit d'un comptage de mot (récursif) O(c) + O(n)	Revient à faire une recherche suivit d'un comptage de mot (récursif) O(c) + O(n)
Suppression()	Revient à faire une recherche suivit de suppression à la remontée en cas de réussite O(c*2) = O(c)	Revient à faire une recherche suivit de suppression à la remontée en cas de réussite O(c*2) = O(c)
Merge()	O(k*n)	**X**
PatriciaToHybrid	O(n + m*(h + c))	**X**
HybridToPatricia	**X**	On liste les mots, puis on les ajoute O(n + m*c)
Equilibrage()	**X**	On liste les mots, puis on les ajoute O(n + m*c)

IV - Etude experimentale

Question 5.13: Fusion experimentale

En construisant le patricia trie fichiers par fichiers, nous arrivons à un temps s'approchant des 10 secondes. La premier essai de fusion s'est effectué avec un seul processus, chaque fichier est dans un premier temps inséré dans un patricia trie, ceci fait nous avons merge tous les patricia tries entre eux, soit autant d'appel à `merge()` que de fichiers moins un. Temps total de cette opération: 16 secondes, soit 6 secondes pour fusionner tous les PatriciaTries.

La méthode `addMergeAllfilesMultiprocessing()` va créer un processus pour chaque fichier existant. Chaque fils va construire l'arbre à partir du fichier donné en argument, et le placer dans une file partagée avec le père. Le programme principal pourra alors faire un appel à merge à chaque ajout dans la file. Cette fonction s'exécute en 10 secondes et fournit exactement le même arbre que les deux précédentes méthodes. La fonction merge prend trop de temps et ne permet pas avec cette implémentation un gain de temps conséquent.

Question 5.14: Aperçu visuel des arbres

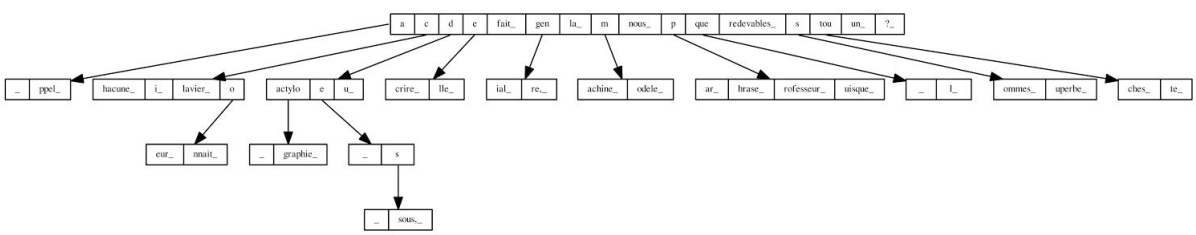
Il est possible de générer un aperçu des deux structures en PDF à l'aide de la fonction toDot(), qui va générer le code DOT dans un fichier et créer le pdf. En revanche si les arbres sont trop grands, il est impossible de les ouvrir avec Adobe Acrobat Reader ou Evince, il faut passer par les navigateurs Google Chrome ou Firefox.

Attention, les tries Hybrid sont très long à générer.

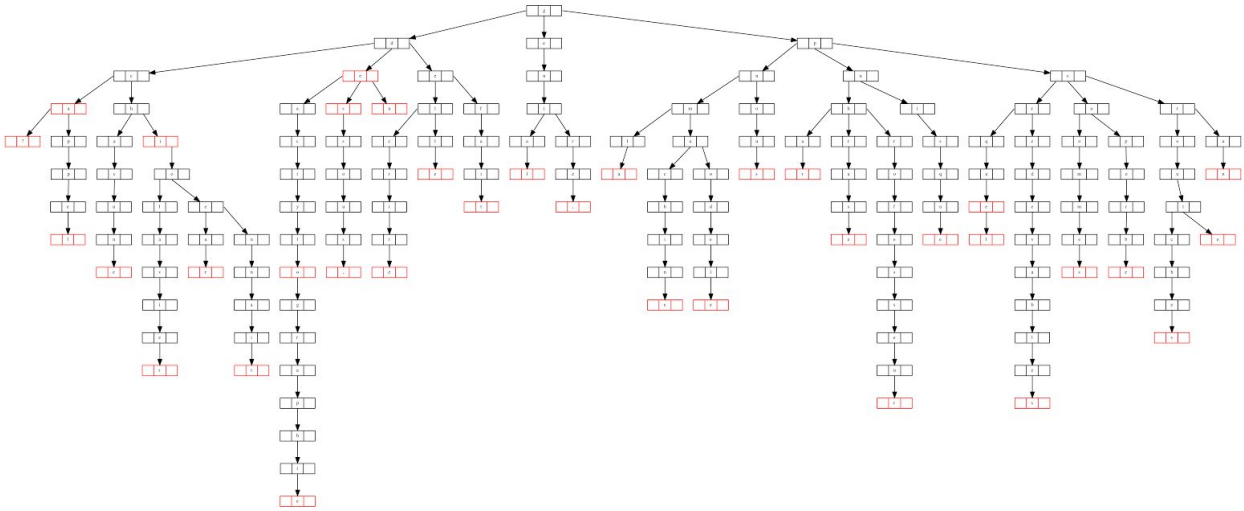
V - Conclusion: Patricia VS Hybrid

Aperçu des arbres (sur la phrase de l'exemple) :

PatriciaTrie :



HybridTrie (balanced) :



Statistiques:

	Patricia Tries	Hybrid Tries
Nombre total de noeuds	11 472	170 287
Nombre de NIL	1 456 945	340 575
Hauteur max	10	38
Hauteur moyenne	5,04	17,76
Temps de generation	~10 sec	~6 sec
Temps recherche*	~11.5 sec	~1.9 sec
Utilisation mémoire	35.5Mo	108.9Mo

*Recherches successives de tous les mots de tous les fichiers

En conclusion, avec notre implémentation des deux structures, les Hybrid tries sont incontestablement les plus rapides, aussi bien en temps de recherche qu'en temps de génération, c'est donc la structure à choisir si vous souhaitez les meilleures performances. En revanche, les Patricia Tries demandent près de 6 fois moins de mémoire et sont donc à privilégier pour des environnements limités en mémoire. Les données qui ne sont pas écrites en mémoire d'un côté doivent calculées de l'autre et donc prend plus de temps.

Détail de l’utilisation mémoire:

Guppy est un module python permettant notamment d'afficher l'état de la mémoire à n’importe quelle étape du programme. Il fonctionne de la façon suivante:

```
from guppy import hpy
### Some code ###
h = hpy()
print h.heap()
```

Voici le résultats pour les deux structures après l’ajout de l’ensemble de l’oeuvre de Shakespeare.

HybridTries:

Partition of a set of 730846 objects. Total size = 108855536 bytes.

Index	Count	%	Size	%	Cumulative	%	Kind
0	170287	23	47680360	44	47680360	44	dict of __main__.HybridTrie
1	340752	47	46344576	43	94024936	86	list
2	170287	23	10898368	10	104923304	96	__main__.HybridTrie
3	12210	2	967216	1	105890520	97	str
4	23255	3	558120	1	106448640	98	int
5	6049	1	488904	0	106937544	98	tuple
6	325	0	281080	0	107218624	98	dict (no owner)
7	70	0	219280	0	107437904	99	dict of module
8	1682	0	215296	0	107653200	99	types.CodeType
9	202	0	214768	0	107867968	99	dict of type

PatriciaTries:

Partition of a set of 98349 objects. Total size = 35475864 bytes.

Index	Count	%	Size	%	Cumulative	%	Kind (class / dict of class)
0	23153	24	26646600	75	26646600	75	list
1	11472	12	3212160	9	29858760	84	dict of __main__.PatriciaTrie
2	35262	36	2076472	6	31935232	90	str
3	11472	12	734208	2	32669440	92	__main__.PatriciaTrie
4	7329	7	597048	2	33266488	94	tuple
5	340	0	292960	1	33559448	95	dict (no owner)
6	92	0	283040	1	33842488	95	dict of module
7	2035	2	260480	1	34102968	96	types.CodeType
8	251	0	248456	1	34351424	97	dict of type
9	1975	2	237000	1	34588424	97	function


Evolution des Tries pour les 10000 premiers mots
800000
600000
400000
200000
0
Nil Patricia
Nil Hybrid
Node Patricia
Node Hybrid
Mots