Algorithmique Avancée **Devoir de Programmation : Tries**

Devoir à faire en binôme. Soutenance en séance de TD/TME 10 (entre le 16/12 et 19/12).
Rapport (au format pdf, d'une dizaine de pages), Code Source, et transparents de Présentation à déposer dans une archive compressée (format zip ou tar.gz) sur `moodle` *au plus tard le 19/12 à 23h59.*

Langage de programmation libre.

1 Présentation

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. En plus des implantations des structures de données et des primitives de base, nous envisageons une fonction avancée pour chacun des modèles.

Les deux modèles envisagés sont (1) les Patricia-tries (cf. TD 2 exercice 2.3) et (2) les tries hybrides (cf. cours). On rappelle que dans une telle structure, chaque mot encodé n'apparaît qu'une seule fois.

Le dictionnaire que nous considérons est constitué de mots construits sur l'alphabet du code ASCII. Celui-ci est composé de 128 caractères, chacun étant encodé sur 8 bits.

1.1 Structure 1 : Patricia-Tries

Dans l'exercice de TD, l'alphabet utilisé contenait 4 lettres, ainsi qu'un caractère indiquant la fin d'un mot.

Question 1.1 Déterminer judicieusement un caractère parmi les 128 du code ASCII qui sera utilisé pour indiquer la fin d'un mot. Par conséquent, on ne pourra représenter que des mots construits sur les 127 caractères restants.

Question 1.2 Encoder les primitives de base concernant les Patricia-tries.

Question 1.3 Construire par ajouts successifs l'arbre représentant la phrase suivante, appelée par la suite `exemple de base` :
A quel genial professeur de dactylographie sommes nous redevables de la superbe phrase ci dessous, un modele du genre, que toute dactylo connait par coeur puisque elle fait appel a chacune des touches du clavier de la machine a ecrire ?
Quelques erreurs se sont glissées dans la phrase, afin de la rendre compatible avec le code ASCII.

1.2 Structure 2 : Tries Hybrides

Question 1.4 Définir et encoder les primitives de base concernant les tries hybrides.

Question 1.5 Construire par ajouts successifs le trie hybride représentant l'`exemple de base`.

2 Fonctions avancées pour chacune des structures

Question 2.6 Écrire les algorithmes suivants pour chacun des deux modèles d'arbres.

- une fonction de recherche d'un mot dans un dictionnaire : `Recherche`(arbre, mot) → booléen
- une fonction qui compte les mots présents dans le dictionnaire : `ComptageMots`(arbre) → entier
- une fonction qui liste les mots du dictionnaire dans l'ordre alphabétique : `ListeMots`(arbre) → liste[mots]
- une fonction qui compte les pointeurs vers Nil : `ComptageNil`(arbre) → entier
- une fonction qui calcule la hauteur de l'arbre : `Hauteur`(arbre) → entier
- une fonction qui calcule la profondeur moyenne des feuilles de l'arbre : `ProfondeurMoyenne`(arbre) → entier
- une fonction qui prend un mot A en argument et qui indique de combien de mots du dictionnaire le mot A est préfixe. Ainsi pour l'`exemple de base`, le mot *dactylo* est préfixe de deux mots de l'arbre (dactylographie et dactylo). Noter que le mot A n'est pas forcément un mot de l'arbre : `Prefixe`(arbre, mot) → entier
- une fonction qui prend un mot en argument et qui le supprime de l'arbre s'il y figure : `Suppression`(arbre, mot) → arbre

3 Fonctions complexes

Question 3.7 Étant donnés deux Patricia-tries, expliquer les étapes importantes de la fusion de ces deux arbres en un seul Patricia-trie. Puis encoder un algorithme permettant de fusionner deux Patricia-tries en un seul.

Question 3.8 Après plusieurs ajouts successifs dans un trie hybride, ce dernier pourrait être plutôt déséquilibré. Après avoir expliqué les idées clé pour rendre un tel arbre mieux équilibré, et donné une idée du seuil permettant de dire si un arbre est déséquilibré ou non, encoder un algorithme d'ajout de mot suivi d'un rééquilibrage si nécessaire.

4 Complexités

Question 4.9 Pour chacune des questions de 2.6. à 3.8, donner, en la justifiant, la complexité de la fonction (après avoir indiqué quelle était la mesure de complexité judicieuse, par exemple, le nombre de comparaisons de caractères, ...)

5 Format du rendu du code

Les entrées/sorties de votre code doivent avoir un format strict pour permettre de lancer des tests automatiques.

- Si vous utilisez un langage compilé vous devez fournir un MakeFile qui permet de compiler votre code avec la commande "make". Si vous utilisez un langage interprété le MakeFile n'est pas nécessaire.
- Les arbres de Patricia et Tries Hybrides seront représentés sous format json (voir les Annexes pour un exemple). Veillez à respecter scrupuleusement le format indiqué.
- Dans ce qui suit nom_fichier.txt est une variable représentant le nom d'un fichier textuel qui contient un seul mot par ligne (un exemple est donné en Annexes).
- Veillez à indiquer dans le README de votre projet le langage que vous avez utilisé, sa version et le compilateur utilisé ainsi que sa version.

Votre code doit pouvoir s'exécuter avec un script bash comme décrit ci-dessous :

```
inserer x nom_fichier.txt
```

Si $x = 0$ la commande crée un arbre de Patricia en insérant successivement les mots de `nom_fichier.txt` et ensuite l'arbre est enregistré en format json dans un fichier nommé `pat.json`.
Si $x = 1$ la commande crée un trie hybride en insérant successivement les mots de `nom_fichier.txt` et ensuite l'arbre est enregistré en format json dans un fichier `trie.json`.

```
suppression x nom_fichier.txt
```

Si $x = 0$ on lit l'arbre `pat.json` puis on supprime successivement les mots de `nom_fichier.txt` et enfin on enregistre le nouvel arbre dans `pat.json`.
Si $x = 1$ on lit l'arbre `trie.json`. On fait le même travail que précédemment, puis on enregistre le nouvel arbre dans `trie.json`.

```
fusion x arbre_1.json arbre_2.json
```

où `arbre_1.json` et `arbre_2.json` sont des fichiers au format json.
Si $x = 0$ on fait une fusion des arbres de Patricia contenus dans les fichiers `arbre_1.json` et `arbre_2.json` et on écrit le résultat dans `pat.json`.
Si $x = 1$ on fait une fusion des Tries Hybrides contenus dans les fichiers donnés en paramètres et on écrit le résultat dans `trie.json`.

```
listeMots x arbre.json
```

où `arbre.json` est une variable représentant le nom d'un fichier au format json. Si $x = 0$ on lit l'arbre de Patricia contenu dans le fichier json et on écrit la liste des mots contenus dans l'arbre dans le fichier `mot.txt` (on mettra un mot par ligne, même format que pour `nom_fichier.txt`). Si $x = 1$ on fait la même chose à partir d'un Trie Hybride.

```
profondeurMoyenne x arbre.json
```

où `arbre.json` est une variable représentant le nom d'un fichier au format json. Si $x = 0$ on lit l'arbre de Patricia contenu dans le fichier json et on écrit la valeur de la profondeur moyenne dans le fichier `profondeur.txt` (le fichier contiendra uniquement cette valeur sans retour à la ligne, cette valeur est un flottant).
Si $x = 1$ on fait le même chose à partir du Trie Hybride contenu dans le fichier json.

```
prefixe x arbre.json s
```

où `arbre.json` est une variable représentant le nom d'un fichier au format json Si $x = 0$ on lit l'arbre de Patricia contenu dans le fichier json arbre et on écrit l'entier correspondant au nombre de mots de l'arbre tels que le mot s en est un préfixe. Le résultat sera écrit dans le fichier `prefixe.txt` (le fichier contiendra uniquement cet entier sans retour à la ligne).
Si $x = 1$ on fait la même chose sur le Trie Hybride contenu dans le fichier json.

Exemple, si vous écrivez en python votre fichier bash pour la commande "inserer" pourrait ressembler à :

```
#!/bin/bash

# Check if exactly two arguments are provided
if [ "$#" -ne 2 ]; then
    echo "Usage: $0 <x> <y>"
    exit 1
fi

x=$1
y=$2

# Check if x is either 0 or 1
if [ "$x" -eq 1 ]; then
    echo "Running script1.py with argument $y"
    python3 inserer_patricia.py "$y"
elif [ "$x" -eq 0 ]; then
    echo "Running script2.py with argument $y"
    python3 inserer_trie.py "$y"
else
    echo "Error: x must be 0 or 1"
    exit 1
fi
```

6 Étude expérimentale

La dernière partie du devoir consiste à comparer expérimentalement les structures.

Question 6.10 Construire le Patricia-trie et le trie hybride encodant l'ensemble des mots de l'œuvre de Shakespeare [1].

Question 6.11 En utilisant les fonctions définies dans les parties précédentes (et éventuellement d'autres), comparer les deux types d'arbres encodant les œuvres de Shakespeare. La comparaison consiste par exemple en :

- temps de construction de la structure complète
- temps d'ajout d'un nouveau mot (éventuellement n'existant pas en anglais) dans chacune des structures
- temps de la suppression d'un ensemble de mots des structures
- profondeur des structures
- ... et toute autre comparaison qui semble pertinente.

Question 6.12 Toujours, sur l'ensemble de vos fonctions, mettre en place un compteur dans les fonctions afin de mesurer expérimentalement la complexité des fonctions que vous avez proposée en Section 4. Par exemple pour la fonction d'insertion, la mesure de complexité adéquate est le nombre de comparaisons. Compter cette valeur pour la construction de la structure complète. Recommencer pour chaque suggestion de la Question 5.11.

Question 6.13 Peut-on mettre en rapport ces expérimentations avec l'approche théorique de la Section 4 ? Retrouve-t-on les résultats théoriques attendus ?

Question 6.14 (facultative) Comparer les temps de construction des arbres soit par ajouts successifs soit par fusions d'arbres encodant chacun une œuvre (cette approche permet des constructions en parallèle).

1. Une archive de chaque œuvre est disponible sur la page `moodle` de l'UE.

7 Annexes

Example d'un fichier nom_fichier pour les commandes "inserer" et "suppression"

```
car
cat
cart
dog
bat
```

Exemple des structures sur l'ensemble de mots {car, cat, cart, dog, bat}. L'arbre de Patricia sera :

```
{
    "label": "",
    "is_end_of_word": false,
    "children": {
        "b": {
            "label": "bat",
            "is_end_of_word": yes,
            "children": {}
        },
        "c": {
            "label": "ca",
            "is_end_of_word": false,
            "children": {
                "r": {
                    "label": "rt",
                    "is_end_of_word": true,
                    "children": {}
                },
                "t": {
                    "label": "t",
                    "is_end_of_word": true,
                    "children": {}
                }
            }
        },
        "d": {
            "label": "dog",
            "is_end_of_word": yes,
            "children": {}
        }
    }
}
```

Et le trie Hybride sera :

```
{
    "char": "c",
    "is_end_of_word": false,
    "left": {
        "char": "b",
        "is_end_of_word": false,
        "left": null,
        "middle": {
            "char": "a",
            "is_end_of_word": false,
            "left": null,
            "middle": {
                "char": "t",
                "is_end_of_word": true,
                "left": null,
                "middle": null,
                "right": null
            },
```

```
            "right": null
        },
        "right": null
    },
    "middle": {
        "char": "a",
        "is_end_of_word": false,
        "left": null,
        "middle": {
            "char": "r",
            "is_end_of_word": true,
            "left": null,
            "middle": {
                "char": "t",
                "is_end_of_word": true,
                "left": null,
                "middle": null,
                "right": null
            },
            "right": {
                "char": "t",
                "is_end_of_word": true,
                "left": null,
                "middle": null,
                "right": null
            }
        },
        "right": null
    },
    "right": {
        "char": "d",
        "is_end_of_word": false,
        "left": null,
        "middle": {
            "char": "o",
            "is_end_of_word": false,
            "left": null,
            "middle": {
                "char": "g",
                "is_end_of_word": true,
                "left": null,
                "middle": null,
                "right": null
            },
            "right": null
        },
        "right": null
    }
}
```