Dictionnaires

Samy Jaziri

Sujet de khôlle inspiré du sujet du Concours Communs Polytechniques 2013

Préambule

Tous les algorithmes de ce sujet devront être implémentés en ***C****.*

Pour vous mettre dans les conditions du concours, vous n'avez pas le droit d'accéder aux ressources en ligne.

En ce qui concerne les questions orales de la khôlle, lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de fa'c'on schématique, courte et précise. Vous ne devez pas expliquer votre code ligne par ligne! Quand la complexité d'un algorithme est demandée en temps ou en mémoire en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, donné en notation de Landau ($\mathcal{O}(n), \mathcal{O}(\log(n)), \dots$). Prenez des notes lorsque vous préparez une question orale pour retrouver plus rapidement les grandes lignes de votre explication lorsque l'examinateur passe vous voir.
Vous devrez tester votre programme sur des exemples bien choisis permettant de vérifier rapidement que votre code est fonctionnel.
Il vous est demandé d'aborder les questions dans l'ordre et de noter vos difficultés à répondre à une question avant de passer à la suivante. Vous pourrez alors les aborder avec l'examinateur.
Attention, une attention particulière sera portée aux problèmes de fuite de mémoire lorsque vous utiliserez l'allocation dynamique
Les deux parties sont indépendantes.

1 Introduction

Motivation

Le but de ce sujet est d'implémenter une structure de *tableau associatif* ou *dictionnaire* en **C**, dont les clés sont des chaînes de caractères alphanumériques. On souhaite que l'insertion et la recherche s'exécutent avec une complexité linéaire en la longueur de la plus longue clé. On utilisera pour cela une implémentation par les arbres `PATRICIA` introduit dans la section suivante.

Préliminaires

La structure de dictionnaire implémentée dans ce sujet, possède des clés sous forme de *mots* et des valeurs sous forme de *chaînes de caractères quelconques*. Un *mot* est une séquence *éventuellement vide* de caractères alphanumériques. Une chaîne de caractère est un tableau de `char` qui se termine par la caractère '\0' (code `ASCII` 0). On définit le type suivant pour représenter les mots en **C**.

```c
typedef unsigned int uint;
typedef struct mot {
    char* chaine;
    uint longueur;
} mot;
```

On définit une relation d'ordre $<$ entre les mots non vides. Soit deux mots $m_1 = "a_1 \dots a_n"$ et $m_2 = "b_1 \dots b_m"$, on dit que $m_1 < m_2$ si et seulement si le code `ASCII` de a_1 est strictement plus petit que le code `ASCII` de b_1.

Question 1

Implémentez une fonction `bool distinguer(mot* m1, mot* m2)` qui renvoie vrai si `m1` $<$ `m2` et faux sinon. *On pensera à inclure la librairie* `stdbool.h`.

On aura besoin de deux fonctions pour ajouter ou enlever un préfixe d'un mot.

Question 2

Implémentez les qutres fonction suivantes :

- `mot* prefixer(mot* m, mot* prefixe)` qui renvoie le mot `m` auquel on aura ajouter le préfixe `prefixe`.
- `mot* prefixe(mot* m, uint i)` qui renvoie le prefixe de taille `i` du mot `m`.
- `mot* suffixe(mot* m, uint i)` qui renvoie le suffixe de `m` dont le premier caractère est à la position `i`.
- `mot* copie(mot* m)` qui renvoie une copie de `m`

2 Arbres `PATRICIA`

Un arbre `PATRICIA` A est composé de nœuds qui peuvent être terminaux ou non-terminaux. Chaque nœud est composé d'une séquence éventuellement vide de fils. Chaque branche reliant un nœud à son fils est étiquetée par un mot non vide. Un arbre `PATRICIA` est valide si et seulement si :

- La racine de l'arbre est un nœud non-terminal.
- un nœud, différent de la racine, qui ne possède pas de fils est terminal.
- les étiquettes de chaque nœud doivent être distinctes et ordonnées de manière croissante selon la relation $<$.

Un dictionnaire sera implémenté comme un arbre `PATRICIA` **valide** où chaque nœud contient une chaîne de caractère, `char* s`, telle que

- si le nœud est terminal `s` est une chaîne de caractère (éventuellement vide)
- si le nœud est non-terminal `s` est un pointeur `NULL`.

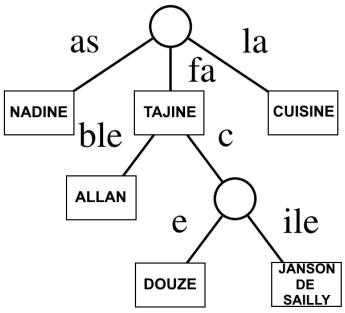

Figure 1: Implémentation d'un dictionnaire

Question 3

Définir un le type dictionnaire en **C**. La liste des branches d'un nœud sera implémentée par une liste chaînée. On mettra à disposition les fonctions suivantes :

- `dictionnaire* creer_dictionnaire_vide();` qui renvoie un pointeur vers un dictionnaire vide dont la mémoire a été allouée dynamiquement.
- `void detruire_dictionnaire(dictionnaire*);` qui libère la mémoire occupée par le dictionnaire passé en paramètre.

3 Recherche dans un dictionnaire

Dans un arbre `PATRICIA`, le chemin reliant la racine à un nœud défini un mot par concaténation successives des étiquettes de chaque branche rencontrée dans le chemin. Soit A un arbre `PATRICIA`

et n un nœud internet de A on notera $k_A(n)$ le mot défini par la chemin de A reliant la racine au nœud n.
Dans un dictionnaire, l'accès se fait par un système de clé. Un clé ici sera un mot non vide. L'ensemble des clés du dictionnaire est l'ensemble des $k_A(n)$ pour n un nœud **terminal** de A. La valeur associée à une clé k d'un dictionnaire est donc la chaîne de caractère contenu dans le nœud terminal n telle que $k = k_A(n)$.

Question 4

Implémenter la fonction d'accès à un élément du dictionnaire par sa clé : `char* get(dictionnaire* dict, mot* cle, char* defaut);`.

- si `cle` est une clé de `dict` renvoyer la valeur associé à `cle` dans `dict`
- sinon renvoyer `defaut`.

Préparer une réponse à donner à l'oral

Faites l'analyse détaillée de la complexité de votre algorithme.

4 Insertion dans un dictionnaire

Préparer une réponse à donner à l'oral

Détailler les étapes de l'ajout des clés `facteur, fac, lampe` et `lame` dans le dictionnaire représenté ci-dessous (qu'importe les valeurs associées). En précisant toutes les étapes et les arbres intermédiaires.

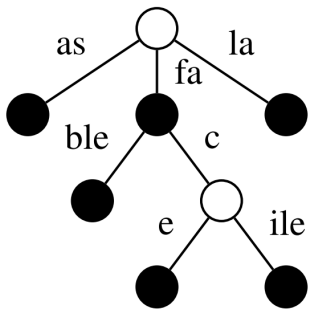

Figure 2: Dictionnaire dont les clés sont `{as, fable, facile, face, la}`.

Question 5

Implémenter la fonction d'insertion d'un couple clé-valeur dans le dictionnaire : `void set(dictionnaire* dict, mot* cle, char* valeur);`.

- si `cle` est une clé de `dict` changer la valeur associé à `cle` dans `dict` pour qu'elle soit égale à `valeur`
- sinon créer une nouvelle clé du dictionnaire `cle` dont la valeur est `valeur`.

Préparer une réponse à donner à l'oral

Faites la preuve de correction détaillée de votre algorithme.

Question 6

Vérifier que vos fonctions `get` et `set` fonctionnent en créant un dictionnaire adéquat et en accédant à ses éléments.

5 Liste des éléments

Question 7

Définir un type `liste_cle_valeur` représentant une liste de couple de mot et de chaîne de caractère. Vous utiliserez une liste doublement chaînée. Vous implémenterez les fonctions suivantes :

- `liste_cle_valeur* creer_liste_cle_valeur_vide();` qui renvoie un pointeur vers une liste vide dont la mémoire a été allouée dynamiquement.
- `void detruire_liste_cle_valeur(liste_cle_valeur*);` qui libère la mémoire occupée par une liste.
- `void prefixer_liste(liste_cle_valeur*, mot*);` qui préfixe toutes les clés de la liste par le mot passé en paramètre.
- `void concatener(liste_cle_valeur*, liste_cle_valeur*);` qui prend deux listes, concatène la seconde à la première et détruit la deuxième liste.

Question 8

Implémenter une fonction `liste_cle_valeur* elements(dictionnaire*);` qui renvoie la liste des couples clé et valeur du dictionnaire passé en paramètre.