Programmation avancée
Projet 2: Arbre binaire de recherche

Pierre Geurts – Jean-Michel Begon

7 Novembre 2014

Énoncé

L'objectif du projet est d'implémenter une structure d'arbre binaire de recherche et de l'utiliser pour trouver des (co)occurrences de mots au sein d'une même phrase d'un texte donné. Concrètement, on souhaite lister l'ensemble des phrases (par ordre d'occurrence dans le texte) où un ou plusieurs mots apparaissent. L'ordre relatif des différents mots n'a pas d'importance tant qu'ils apparaissent effectivement dans une même phrase.

Le texte support sera une version prétraitée de *The Time Machine* (H. G. Wells, 1895) basée sur la version eBook gratuite du projet Gutenberg (n'hésitez pas à aller y jeter un oeil, il y a de très bons livres).

1 Implémentation

L'idée générale de la solution à implémenter est d'utiliser un arbre binaire de recherche pour stocker un index inversé du texte dans lequel on souhaite faire la recherche. Cet arbre binaire de recherche contiendra tous les mots du texte (qui seront les clés de l'arbre) et à chaque mot/nœud de l'arbre sera associé la liste des positions d'occurrence du mot dans le texte, c'est-à-dire l'indice des phrases (dans l'ordre du texte) dans lesquelles ce mot apparaît. La recherche des phrases dans lesquels plusieurs mots apparaissent consistera alors à faire l'intersection des listes de positions associées à chacun des mots dans l'arbre binaire de recherche. L'arbre binaire de recherche et les listes de positions pour chaque mot seront créées en parcourant séquentiellement le texte, phrase par phrase et mot par mot.

Il y a donc deux structures à implémenter :

(a) `SequentialSet` : un ensemble "séquentiel". C'est à dire, un ensemble d'entiers (les positions) auquel on va toujours fournir les valeurs par ordre croissant.

(b) `BinarySearchTree` : un arbre binaire de recherche (dont les clés seront des chaînes de caractères et les valeurs associées des ensembles séquentiels).

Dans les deux cas, un fichier *header* expose une structure opaque ainsi que l'interface des structures de données. La structure est dite opaque car elle n'expose pas son implémentation. Celle-ci est donc libre.

1.1 Ensemble séquentiel

La structure `SequentialSet` stocke des éléments de type `size_t`. Une même valeur ne peut être présente qu'une seule fois dans la structure. Celle-ci peut insérer les valeurs en tirant parti du fait que les valeurs sont fournies par ordre croissant. Si cette condition n'est pas rencontrée, il est impossible de garantir le comportement correct de la structure.

Les différentes opérations sur cette structure sont :

`createSequentialSet` qui crée un nouvel ensemble séquentiel ;

`freeSequentialSet` qui libère la mémoire allouée par/pour l'ensemble séquentiel ;

`getNumItems` qui renvoie le nombre d'éléments contenus par ensemble séquentiel ;

`insertValue` qui insère un nouvel élément dans l'ensemble séquentiel ;

`contains` qui teste si un élément est présent dans l'ensemble séquentiel ;

`setAsArray` qui renvoie une vue instantanée de l'ensemble séquentiel sous forme d'un tableau ;

`intersect` qui réalise l'intersection de plusieurs ensembles séquentiels ;

Etant donné l'application visée, on veillera à optimiser en priorité la complexité en temps de l'opération d'intersection et de l'opération d'insertion, ainsi que l'espace mémoire occupé par la structure.

1.2 Arbre binaire de recherche

Il s'agit d'implémenter un dictionnaire sous la forme d'un arbre binaire de recherche dont :

Les clés sont des chaînes de caractères (`char*`). Dans le cadre de ce projet, il s'agira de mots.

Les valeurs sont des `SequentialSet` contenant des positions. Dans le cadre de ce projet, il s'agira des numéros de phrases où le mot apparaît.

A noter que c'est à l'arbre de gérer le stockage des positions dans les bons `SequentialSet`. En effet, l'interface ne permet de soumettre à l'arbre que des paires mot-position.

Nous vous demandons d'implémenter les opérations de base suivantes :

`createBinarySearchTree` qui crée un nouvel arbre binaire de recherche vide ;

`freeBinarySearchTree` qui libère la mémoire allouée par/pour la structure de l'arbre ;

`getNumString` qui retourne le nombre de mots différents dans l'arbre ;

`getTotalNumString` qui retourne le nombre de paires mots-phrases différentes dans l'arbre ;

`insert` qui permet d'ajouter une nouvelle paire mot-position dans l'arbre :
- Si le mot n'est pas présent dans l'arbre, celui-ci crée un nouveau noeud contenant le mot ainsi qu'un nouveau `SequentialSet` pour y stocker l'index de la phrase ;
- Si le mot est déjà présent, l'arbre ajoute sa position dans le `SequentialSet` correspondant ;
- Remarque : par définition, si un même mot apparaît plusieurs fois au sein d'une même phrase, on ne stocke qu'une seule fois l'index correspondant ;

`find` qui permet de retrouver tous les indices d'occurrence d'un mot (par ordre croissant) ;

`findCooccurrences` qui permet de retrouver tous les indices de cooccurrence d'un groupe de mots (par ordre croissant) ;

`isBalanced` qui détermine si l'arbre est équilibré ou non. Un arbre est équilibré si pour chacun de ses nœuds, la hauteur du sous-arbre de droite de ce nœud ne diffère pas de la hauteur du sous-arbre de gauche de plus d'une unité ;

`getHeight` qui renvoie la hauteur de l'arbre ;

`getAvgDepth` qui renvoie la profondeur moyenne de l'arbre (i.e. la moyenne de la profondeur de chaque noeud de l'arbre).

Il n'est pas nécessaire d'implémenter la suppression de clés pour ce problème. En ce qui concerne la fonction `getTotalNumString`, si un même mot apparaît plusieurs fois dans une même phrase, il ne doit compter que pour un. Par exemple, l'insertion mot par mot de la phrase :

"foo bar foo baz"

dans un arbre vide conduit à une seule insertion de *foo* (et de *bar* et *baz*). La méthode `getTotalNumString` sur ce même arbre renverra 3.

1.3 Fichiers fournis

Nous vous fournissons les fichiers suivants :

`time_machine.txt` contient la version primaire du livre *The Time Machine* telle que fournie par le projet Gutenberg ;

`time_machine_formated.txt` contient une version formatée du livre. C'est avec ce fichier là qu'on va travailler ;

`SequentialSet.h` contient l'interface de la structure du même nom ;

`BinarySearchTree.h` contient l'interface de la structure du même nom ;

`main.c` contient un programme de test.

Une fois compilé avec la commande :

```
gcc BinarySearchTree.c SequentialSet.c main.c --std=c99 -o find_occurrences
```

le programme s'utilise comme suit :

```
find_occurrences time_machine_formated.txt geometry misconception school
```

afin de trouver s'il y a une phrase dans laquelle les trois mots *geometry*, *misconception* et *school* apparaissent. Le programme peut accepter plus que trois mots. Si aucun mot n'est fourni, le programme se contentera d'afficher quelques statistiques sur le texte.

2 Rapport et analyse théorique

Il vous est demandé de fournir un rapport épousant le canevas suivant :

1. Ensemble séquentiel :
   (a) Décrivez et justifiez votre implémentation de l'ensemble séquentiel.
   (b) Expliquez le fonctionnement de la fonction `intersect`.
   (c) Identifiez le pire cas et sa complexité pour :
      i. l'opération d'insertion d'une position (en fonction de n, le nombre d'éléments dans l'ensemble).
      ii. l'opération de test de présence d'une position (en fonction de n, le nombre d'éléments dans l'ensemble).
      iii. l'opération d'intersection de deux ensembles (en fonction de m, le nombre total d'éléments dans les deux ensembles).
2. Arbre binaire de recherche :
   (a) Décrivez vos choix d'implémentation pour l'arbre binaire de recherche.
   (b) Pour les deux fonctions `getAvgDepth` et `isBalanced` :
      i. Donnez le pseudo-code en vous basant sur l'implémentation d'arbre binaire utilisée dans le cours théorique.
      ii. Etudiez leur complexité dans le meilleur et le pire cas.
   (c) Tracez l'évolution de la hauteur et de la profondeur moyenne des nœuds de l'arbre en fonction du nombre de clés qu'il contient. Commentez ces deux graphes. Pour recueillir les informations nécessaires, vous pouvez modifier le fichier `main.c`.
3. Complexité de `FindCooccurrences` :
   (a) En supposant que vous avez implémenté une insertion dans l'arbre binaire sans équilibrage, étudiez la complexité de la fonction `FindCooccurrences` dans le cas d'une recherche de deux mots en fonction du nombre de mots n dans l'arbre et en supposant qu'un mot apparaît au plus dans k phrases différentes. Expliquez le meilleur et le pire cas.
   (b) Que deviennent ces complexités si on suppose que l'arbre est équilibré ?

3 Deadline et soumission

Le projet est à réaliser **individuellement** pour le **30 novembre 2014** à **23h59** au plus tard. Le projet est à remettre via la plateforme *cicada* : http://cicada.run.montefiore.ulg.ac.be. Il doit être rendu sous la forme d'une archive `tar.gz` contenant :

(a) Votre rapport (4 pages maximum) au format PDF. Soyez bref mais précis et respectez bien la numération des (sous-)questions.

(b) Un fichier `BinarySearchTree.c` contenant l'implémentation de la structure correspondante.

(c) Un fichier `SequentialSet.c` contenant l'implémentation de la structure correspondante.

Respectez bien les extensions de fichiers ainsi que leur nom pour les fichier `*.c` (en ce compris la casse). Seule la dernière archive soumise sera prise en compte.

Vos fichiers seront évalués sur les machines ms8** avec les commandes :

```
gcc testSeqSet.c SequentialSet.c --std=c99 --pedantic -Wall -Wextra -Wmissing-prototypes
-o test
```

```
gcc testWhole.c BinarySearchTree.c SequentialSet.c --std=c99 --pedantic -Wall
-Wextra -Wmissing-prototypes -o test
```

(Vous ne devez pas fournir les fichiers `testSeqSet.c` et `testWhole.c`.)

Ceci implique que :

— Le projet doit être réalisé dans le standard C99.

— La présence de *warnings* impactera négativement la cote finale.

— Un projet qui ne compile pas avec cette commande recevra une cote nulle pour le code.

Un projet non rendu à temps recevra également une cote nulle. En cas de plagiat avéré, l'étudiant se verra affecter une cote nulle à l'ensemble des projets.

Les critères de correction sont précisés sur la page web du cours.

Bon travail !