ENSTA Paris
2023-2024

Résolution de problèmes algorithmiques – IN103

Danil Berrah et Alexandre Chapoutot

Feuille d'exercices 3

Objectif(s)

- Manipuler une bibliothèque d'arbres binaires ;
- Manipuler une bibliothèque d'arbres binaires de recherche ;
- Découvrir les algorithmes d'apprentissage automatique fondés sur les arbres de décisions.

PRÉPARATION

Cette première partie permet de préparer votre environnement (**si vous ne l'avez pas fait au dernier TD**) de travail afin de pouvoir utiliser facilement la bibliothèque logicielle **libin103** spécialement développée pour cet enseignement.

1. A la racine de votre compte, créez un répertoire nommé `Library` s'il n'a pas déjà été créé, puis placez vous dans ce répertoire.

   ```
   mkdir ~/Library; cd ~/Library
   ```

2. Téléchargez l'archive `libin103-1.4.tar.gz` sur le site du cours

   ```
   wget https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/libin103-1.4.tar.gz
   ```

3. Désarchivez l'archive

   ```
   tar -xvzf libin103-1.4.tar.gz
   ```

4. Allez dans le répertoire `libin103-1.4` et compilez la bibliothèque. Quelle commande faut-il utiliser ?
   — Il faut utiliser la commande `make`. A la fin de la compilation vérifier la présence du fichier `libin103.a` dans le répertoire `source`.
   — Également, vous pouvez exécuter la commande `make check` pour compiler et exécuter les programmes de tests.

EXERCICES

Exercice 1 – Prise en main de la bibliothèque des arbres binaires

L'objectif de cet exercice est de prendre en main la partie de la bibliothèque qui concerne les arbres binaires. Pour cela nous allons construire pas à pas un arbre binaire dont l'information contenue dans les nœuds sera des entiers.

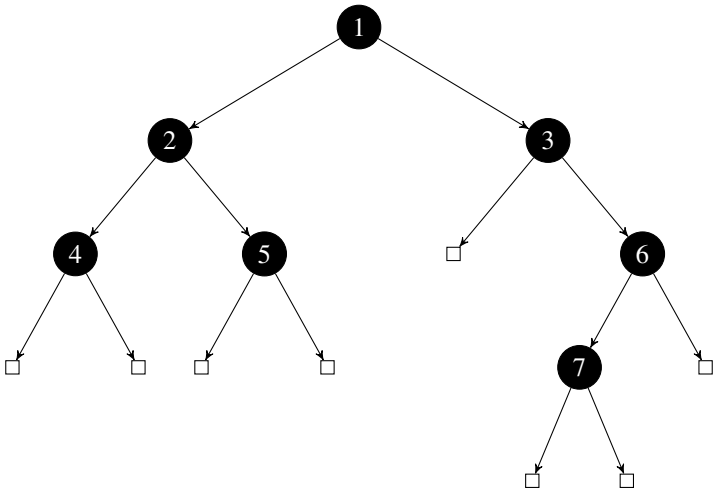
Fonctions utiles pour cet exercice

- `integer_bitree_init`
- `integer_bitree_destroy`
- `integer_bitree_ins_right`
- `integer_bitree_ins_left`
- `integer_bitree_size`
- `integer_bitree_root`
- `integer_inorder`
- `integer_preorder`
- `integer_postorder`
- `integer_list_init`
- `integer_list_head`
- `integer_list_tail`
- `integer_list_next`
- `integer_list_data`
- `integer_list_destroy`

Question 1
Dans quels fichiers peut-on trouver le prototype de ces fonctions ?

Question 2
Créez une variable, nommée `tree`, dont le type est un arbre binaire d'entiers et initialisez cette variable.

Question 3
Construisez l'arbre binaire qui a la structure suivante



Question 4
Nous voulons maintenant faire un parcours de l'arbre, dans l'ordre infixe.
- Quelle fonction faut-il utiliser pour cela ?
- Quel fichier faut-il inclure ?

Question 5
Mettez en œuvre le parcours infixe de l'arbre depuis la racine.

Question 6
Développez une fonction nommée, `print_integer_list` qui permet d'afficher le contenu d'une liste et dont le prototype est :

```
void print_integer_list (integer_list_t* list);
```

Utilisez cette fonction pour afficher l'ordre des nœuds de l'arbre dans l'ordre infixe.

Question 7
Affichez l'ordre des nœuds de l'arbre suivant un parcours préfixe et postfixe.

Question 8
Quelles fonctions faut-il appeler pour libérer la mémoire associée aux différentes structures de données utilisées ?

Exercice 2 – Un algorithme de tri avec des arbres binaires de recherche

Un arbre binaire de recherche permet de stocker des données en fonctions d'une relation d'ordre sur les valeurs. Nous allons considérer dans que des arbres avec valeurs entières.

Pour cet exercice, nous allons comparer la complexité d'un tri à partir d'un arbre binaire de recherche et un arbre binaire de recherche équilibré.

Fonctions utiles pour cet exercice

- `integer_bistree_init`
- `integer_bistree_destroy`
- `integer_bistree_insert`
- `generic_bitree_root`
- `generic_bitree_data`
- `generic_inorder`
- `generic_list_init`
- `generic_list_head`
- `generic_list_tail`
- `generic_list_next`
- `generic_list_data`
- `generic_list_destroy`
- `integer_list_init`
- `integer_list_head`
- `integer_list_tail`
- `integer_list_next`
- `integer_list_data`
- `integer_list_destroy`

Question 1
Rappelez la façon dont sont stockées les valeurs dans un arbre binaire de recherche (équilibré ou non).

Question 2
Quel type de parcours d'arbre peut être utilisé pour suivre l'ordre croissant des valeurs dans un arbre binaire de recherche ?

Question 3
(Version simple) Dans cette question, nous allons considérer la structure de données AVL des arbres binaires de recherche équilibrée fournie avec la bibliothèque `libin103` (`bistree`). Mettez en œuvre un algorithme de tri en vous appuyant sur cette structure de données.

Question 4
Discutez de la complexité de cet algorithme.

Question 5
Mettez en œuvre une fonction qui permet de calculer la hauteur maximale d'un arbre dont la racine est donnée en argument. Quel style de programmation allez-vous utiliser ?

Question 6
Nous allons maintenant mettre en œuvre un arbre binaire de recherche non équilibré. Définissez une fonction dont le prototype est

```
void add_node(integer_bitree_t* tree, integer_bitreenode_t* current, int elem);
```

Cette fonction permet d'ajouter un élément dans un arbre suivant la propriété des arbres binaires de recherche.

Avant de commencer le développement, quel style de programmation allez-vous utilisé ?

Question 7
Triez le tableau donné dans la fonction `main` avec les deux approches et comparer la hauteur des arbres résultats.

Exercice 3 – Arbres de décisions – Étude de l'algorithme ID3 (Optionnel)

Les arbres de décisions sont un type de méthodes d'apprentissage automatique qui permettent de résoudre des problèmes de classifications. Ces méthodes se positionnent dans le domaine de l'apprentissage supervisé, c'est-à-dire, à partir d'un ensemble de données annotées (valeurs et résultats, p. ex., une image et le contenu représenté) ont construit un modèle qui sera ensuite utilisé pour prédire une propriété sur une donnée n'appartenant pas à l'ensemble d'apprentissage.

Il existe plusieurs méthodes pour construire un arbre de décision, la plus ancienne et la plus simple est l'algorithme ID3 [1] décrit par J.R. Quinlan en 1986. Le principe est relativement simple.

1. On calcule le gain d'information pour chaque attribut
2. Considérant que chaque ligne n'appartiennent pas à la même classe, séparer l'ensemble d'apprentissage S en sous-ensembles en fonction de l'attribut ayant le plus de gain d'information.
3. Créer un nouveau nœud dans l'arbre de décision associé à l'attribut sélectionné avec le gain d'information le grand.
4. Si toutes les lignes du sous-ensemble sont dans la même classe, marquer ce nœud comme feuille avec l'étiquette associée à la classe représentée.

1. https://link.springer.com/article/10.1007/BF00116251

5. Répéter le processus avec les attributs restants jusqu'à que tous les attributs aient été traités ou que l'arbre ait toutes ses feuilles.

Le code proposé dans le répertoire `exo3` donne une mise en œuvre de cet algorithme ID3 (pas fidèlement). Le but de cet exercice est de comprendre comment cet algorithme a été implanté et quelles fonctions/structures de données ont été utilisées pour cela.

Remarque l'objectif n'est pas de comprendre toute la mise en œuvre de l'algorithme ID3 mais de souligner la différence qui peut exister entre une description informelle et une code effectif. Le second objectif est voir une autre utilisation des structures de données arbres dans le domaine de l'apprentissage automatique.

Question 1
Listez les différents types de structures de données qui sont utilisées dans cette mise en œuvre de l'algorithme ID3.

Question 2
L'algorithme ID3 induit beaucoup de manipulation de la base de données d'apprentissage, pour faire des calculs de statistiques ou pour filtrer les lignes de la base de données. Indiquez comme sont mis en œuvre ces manipulations de base de données et listez les différents traitements qui sont effectués.

Question 3
Pourquoi la structure de données arbres binaires est adaptée pour la construction d'arbre de décision pour l'exemple que nous considérons ? Quel impact aurait une autre nature des données sur la structure de données choisies pour représenter l'arbre de décision ?

Question 4
Pourquoi la phase d'apprentissage est construite avec une boucle `do-while` ?

Question 5
Dessinez l'arbre de décision qui est construit à partir de l'ensemble d'apprentissage situé dans le répertoire `flu`.

POUR S'ENTRAÎNER À LA MAISON

Exercice 1 – Retour sur les ensembles
L'objectif de cet exercice est de programmer une nouvelle version de la structure de données ensembles fondée sur les arbres binaires de recherche équilibrés.

Plus précisément, il faut donner une implémentation des fonctions associées aux ensembles d'entiers :
— initialisation
— destruction
— test d'appartenance
— test d'égalité
— test de sous-ensemble
— union
— intersection
— différence

Fonctions utiles pour cet exercice

- `integer_bistree_init`
- `integer_bistree_destroy`
- `integer_bistree_insert`
- `integer_bistree_remove`
- `integer_bistree_lookup`
- `integer_bistree_size`
- `generic_list_init`
- `generic_inorder`
- `generic_bitree_root`
- `generic_list_head`
- `generic_list_tail`
- `generic_list_next`
- `generic_list_data`

Question 1
Définissez un nouveau type nommé `set_t` qui s'appuie sur les arbres binaires de recherche équilibrés.

Question 2
Donnez la fonction d'initialisation `init_set` et la fonction de destruction `destroy_set` pour ce nouveau type d'ensemble.

Question 3
Définissez les fonctions d'insertion d'un élément `insert_set` et suppression d'un élément `remove_set` pour ce nouveau type d'ensemble.

Question 4
Définissez la fonction `size_set` qui donne la taille d'un ensemble.

Question 5
Définissez la fonction `is_member` qui vérifie si une valeur est présente dans l'ensemble.

Question 6
Définissez une fonction `is_subset` qui indique si un ensemble s_1 est sous-ensemble d'un second ensemble s_2.

Question 7
Définissez une fonction `is_equal` qui vérifie si deux ensembles s_1 et s_2 sont égaux (composé avec les mêmes éléments).

Question 8
Définissez la fonction `union_set` qui calcule l'union de deux ensembles s_1 et s_2. Le résultat est rendu par effet de bord (par un pointeur).

Question 9
Définissez la fonction `difference_set` qui calcule la différence de deux ensembles s_1 et s_2. Le résultat est rendu par effet de bord (par un pointeur).

Question 10
Définissez la fonction `intersection_set` qui calcule l'intersection de deux ensembles s_1 et s_2. Le résultat est rendu par effet de bord (par un pointeur).

Question 11
Discutez des complexités des fonctions `insert_set`, `remove_set`, `union_set`, `difference_set` et `intersection_set`.