ENSTA Paris
2023-2024

Résolution de problèmes algorithmiques – IN103

Alexandre Chapoutot et Marine Saint-Mézard

Feuille d'exercices 2

Objectif(s)

★ Prendre en main les structures de données linéaires : liste, pile, file et ensemble.

PRÉPARATION

Cette première partie permet de préparer votre environnement 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`, 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.

Matériel pour le TP

Récupérez l'archive associé à cette séance de TP à l'adresse :

`https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/in103-td2.tar.gz`

EXERCICES

Exercice 1 – Prise en main des listes chaînées

L'objectif de cet exercice est de prendre en main la partie de la bibliothèque qui concerne les listes chaînées.

Fonctions utiles pour cet exercice

- `integer_list_init`
- `integer_list_destroy`
- `integer_list_ins_next`
- `integer_list_rem_next`
- `integer_list_size`
- `integer_list_head`
- `integer_list_tail`
- `integer_list_next`
- `integer_list_data`

Remarque : Les données de cet exercice sont dans le répertoire `exo1`.

Question 1

Écrivez le fichier `Makefile` pour compiler le programme `list-printer.x` à partir du fichier source `list-printer.c`. Pensez à ajouter une règle `clean` pour supprimer les fichiers générés lors de la compilation.

Explication(s):

- ✿ La principale difficulté est d'utiliser les options du compilateur C pour trouver les fichier `.h` et la bibliothèque `libin103.a`.
- ✿ Suivant les instructions d'installation de la bibliothèque `libin103` : l'ajout du répertoire contenant les fichiers `.h` se réalise avec l'option `-I$(HOME)/Library/libin103-1.4/include`
- ✿ Suivant les instructions d'installation de la bibliothèque `libin103` : l'ajout de la bibliothèque`libin103.a` se réalise avec l'option `-L$(HOME)/Library/libin103-1.4/source` et l'option `-lin103` qui est le raccourci pour ajouter le fichier `libin103.a`.

Question 2

Écrivez le code associé à la fonction `print_list` dans le fichier `list-printer.c`.

Exercice 2 – Problème de couverture par ensembles

Les problèmes de couvertures d'ensemble sont des problèmes d'optimisation combinatoires qui apparaissent dans divers domaines comme les problèmes de logistiques, par exemple, placement d'un nombre minimum de centres de distribution maximisant la couverture de population, ou le placement de caméras pour couvrir une zone.

Ces problèmes de couvertures d'ensembles entre dans la classe des problèmes algorithmiques difficiles (classe NP) pour lesquels des solutions approchées peuvent être calculées par des algorithmes gloutons. L'objectif de cet exercice est de mettre en œuvre un tel algorithme.

Fonctions utiles pour cet exercice (en plus de celles sur les listes)

- `integer_set_init`
- `integer_set_destroy`
- `integer_set_insert`
- `integer_set_remove`
- `integer_set_union`
- `integer_set_difference`
- `integer_set_intersection`
- `integer_set_size`

Formulation mathématique du problème est étant donné
— un ensemble U à couvrir ;
— un ensemble d'ensembles S ;
on cherche le plus petit $S' \subseteq S$ tel que $\forall u \in U, u \in S'$.

Par exemple, si $U = \{1,2,4,6,8,9\}$ et $S = \{\{1\},\{2\},\{2,4\},\{4,6\},\{1,6,8\},\{8\},\{9\}\}$ on a :
— $S_1 = \{\{1\},\{2\},\{4,6\},\{8\},\{9\}\}$ est une couverture de U de taille 5 ;
— $S_2 = \{\{2,4\},\{4,6\},\{1,6,8\},\{9\}\}$ est aussi une couverture de U mais de taille 4 ;
— $S_3 = \{\{4,6\},\{1,6,8\},\{9\}\}$ n'est pas une couverture de U car $2 \in U$ mais $2 \notin S_3$.

En s'appuyant sur l'exemple ci-dessus, nous allons développer un algorithme glouton qui trouve une couverture d'ensembles mais qui en général ne sera pas la plus petite solution mais une bonne approximation.

Remarque : Les données de cet exercice sont dans le répertoire `exo5`.

Question 1

Développer une idée d'algorithme glouton pour calculer une couverture d'ensemble.

Question 2
Quelle structure de donnée peut être utilisée pour représenter l'ensemble d'ensembles S ?

Question 3
Écrivez une fonction `glouton` dont le prototype est

```
integer_list_t glouton(integer_set_t* S, int size, integer_set_t* U)
```

— Le type de retour `integer_list_t` représente une liste contenant les indices du tableau S des ensembles qui couvrent U ;
— Les entrées sont :
  — Le tableau S des ensembles ainsi que sa taille `size` ;
  — L'ensemble à couvrir U.

Question 4 – Application
La RATP souhaite sécuriser les couloirs d'une station de métro. Dans un objectif de réduction des coûts, elle souhaite utiliser le moins de caméra possible. On considère que les caméras ont une vision à 360° et une vision infinie.

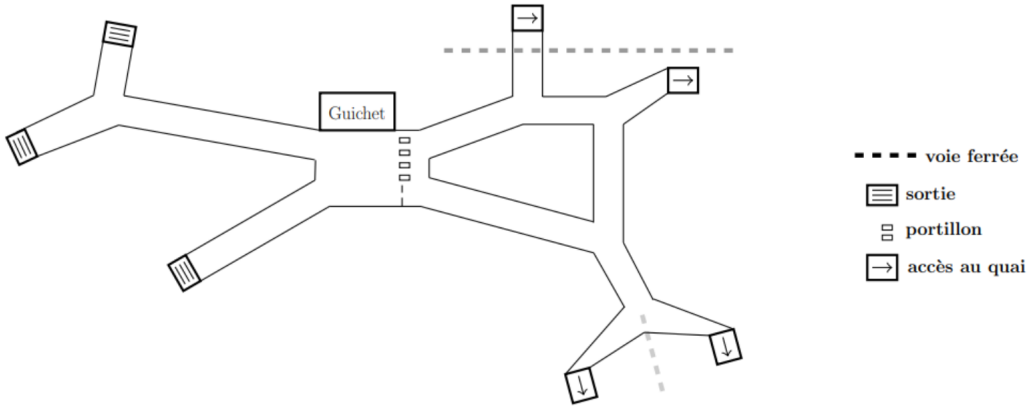

FIGURE 1 – Plan d'une station de métro

1. Comment peut-on modéliser ce problème pour se ramener au problème de couverture d'ensemble ?
2. Combien de caméra sont nécessaires pour couvrir la station de métro ? et où devons-nous les placer ?

Exercice 3 – Un peu de réflexion avec les piles et les files

Un grand classique des problèmes algorithmiques données dans les entretiens d'embauche. En supposant que vous n'avez accès qu'à des structures de données de type pile, donnez une mise en œuvre d'une structure de données de type file. Il faut donner une mise en œuvre des fonctions :
— `enqueue`
— `dequeue`
Remarque : Les données de cet exercice sont dans le répertoire `exo3`.

Question 1
Sur quelle structure de données présentée en cours sont fondées les piles ?

Question 2
Combien de variables de type pile allez-vous utiliser ?

Question 3
Donnez le nouveau type mettant en œuvre une file avec des piles.

Question 4
Donnez la définition de la fonction `enqueue`.

Question 5
Donnez la définition de la fonction `dequeue`.

Question 6
Donnez la définition de la fonction `size`.

Question 7
Discuter de la complexité des opérations `enqueue` et `dequeue` et dire pourquoi ce n'est pas une bonne solution.

POUR S'ENTRAÎNER À LA MAISON

Exercice 1 – Propriété d'unicité dans les ensembles
Cet exercice a pour objectif de manipuler la structure de données des ensembles (sets).

Remarque : Les données de cet exercice sont dans le répertoire `exo4`.

Question 1
Sur quelle structure déjà présentée en cours sont fondées les ensembles (sets) ?

Question 2
L'objectif de cet exercice est d'utiliser un ensemble (set) pour lister les différents caractères alphabétiques qui composent une phrase (on ne considérera donc pas les espaces et les symboles de ponctuation) et on normalisera les caractères pour ne conserver que les minuscules.

Décrivez un algorithme pour faire cela s'appuyant sur la structure de données d'ensembles (sets) et mettez le en œuvre.

Exercice 2 – Tri radix
Il existe de nombreux algorithmes de tri de données comme *bubble sort*, *merge sort*, *quick sort*, etc. La complexité de ces algorithmes oscille entre $\mathcal{O}(n^2)$ et $\mathcal{O}(n\log(n))$ où n est le nombre d'éléments à trier.

Dans cet exercice nous allons explorer une autre méthode de tri, nommée *tri radix* ou *tri par base*. Cet algorithme consiste à trier les nombres chiffres par chiffre. Le principe est le suivant :

— On trie tous les nombres par rapport aux chiffres des unités
— Puis on trie les nombres par rapport aux dizaines, puis par rapport aux centaines, etc.

Remarque : Les données de cet exercice sont dans le répertoire `exo2`.

Question 1
Écrivez une fonction `find_max` dont le prototype est :

```c
int find_max(integer_list_t *list);
```

Explication(s):

- ✿ On reprend la structure de l'itérateur sur les listes utilisé dans l'exercice concernant l'affichage d'une liste.
- ✿ Pour utiliser la constante `INT_MIN` il faut inclure le fichier `limits.h`.

Question 2
Mettez en œuvre l'algorithme du tri par base dans une fonction dont le prototype est :

```c
void radix_sort (integer_list_t* initial, integer_list_t* sorted, int max)
```

Question 3
Discuter de la complexité pire cas de cet algorithme et comparez la par rapport aux autres algorithmes.

APPROFONDISSEMENT

Exercice 1
L'objectif de cet exercice est de mettre en oeuvre une extension de la bibliothèque `libin103` pour gérer les tables de hachage. Pour corser le tout, nous allons considérer la version générique de cette structure de données.

Nous supposons travailler dans le répertoire `exo6` pour cet exercice.

Eléments sur les tables de hachage Les listes chaînées sont une structure de données interressantes pour stocker et manipuler des données dynamiquement sans connaissance *a priori* du nombre d'éléments à stocker. Cependant, la complexité linéaire de la recherche d'un élément ou au mieux une complexité logarithmique dans le cas d'une liste chainée triée, peut rendre son utilisaton peu aisée dans certains cas.

Les tableaux sont une structure de données qui permet un adressage direct aux éléments mais ne permettent d'indexer des éléments que par des entiers.

Les tables de hachages permettent de représenter des ensembles de couples (clef, valeur) avec un accès rapide. L'ensemble des clefs étant grand, l'idée est de plongée cet espace dans un espace plus petit. On utilise un tableau pour représenter des tables de hachage mais les indices du tableau sont calculés à partir de la clef initiale qui est "hachée", associée à un entier. Une problématique dans les tables de hachage est que plusieurs clefs peuvent être associées au même indice du tableau. On parle alors de collision, plusieurs solutions sont possibles pour gérer ces conflits. La solution que nous adopterons pour cet exercice est l'utilisation de liste chaînée pour accumuler au niveau d'un même indice du tableau l'ensemble des couples (clefs, valeurs) qui dont la clef est "hachée" de la même manière.

Pour plus d'explications, vous pouvez regarder le cours de M. Pessaux sur les tables de hachage

`https://perso.ensta-paris.fr/~chapoutot/teaching/in103/slides/cours-table-hachage.pdf`

Travail à faire L'API que nous proposons de développer pour les tables de hachage est donnée dans le fichier `exo6/generic_hashtable.h`. Il consiste en des définitions de types, une fonction d'initialisation et de destruction ainsi que des fonctions d'ajout et de suppression.

Question 1

Programmez les fonctions associées à cette API dans fichier nommé `generic_hashtable.c`