Travaux pratiques de programmation

On s'attend à ce que les candidats traitent de manière au moins partielle chacune des deux parties : programmation en C et OCaml d'une part, audit de code d'autre part.

1 Programmation en C et OCaml

Dans cette partie de l'épreuve, il est demandé au candidat de concevoir, écrire et tester un programme répondant aux questions ci-dessous, dans le langage C pour la section 1.2 et dans le langage OCaml pour les sections 1.3 et 1.4. L'annexe à la fin de ce sujet donne des fonctions de bibliothèque pouvant être utiles.

On rappelle qu'il sera nécessaire, lors du rendu, de présenter le travail réalisé, les forces et faiblesses des approches adoptées, ainsi que d'expliciter et de commenter les jeux de tests utilisés.

1.1 Contexte : Modèle de N-grammes

Nous souhaitons développer un système permettant de prédire la suite d'une séquence de caractères. Ce genre de système est utilisé pour compléter les messages mails ou SMS par exemple. L'utilisateur commence à taper son message (une séquence de caractères) et le système propose de le compléter.

L'approche que nous considérons est basée sur les N-grammes par apprentissage sur un texte préexistant. Un N-gramme est une séquence de N caractères consécutifs apparaissant dans le texte. Nous expliquons cette approche sur le texte suivant (un réel apprentissage nécessite un texte bien plus grand) :

```
Bonjour, comment allez-vous ? Ca va, ca va aller bien mieux.
```

L'objectif de l'approche est

1. de construire les N-grammes contenus dans le texte ;
2. d'identifier les caractères qui leur succèdent en calculant les probabilités associées.

La première étape nécessite de choisir la valeur de N, la taille des N-grammes. Dans notre exemple, le texte contient, par ordre d'apparition :

— les 1-grammes suivants : `"B"`, `"o"`, `"n"`, `"j"`, `"u"`, `"r"`, `","`, `"c"`, etc.
— les 2-grammes suivants : `"Bo"`, `"on"`, `"nj"`, `"jo"`, `"ou"`, etc.
— les 3-grammes suivants : `"Bon"`, `"onj"`, `"njo"`, `"jou"`, etc.
— etc.

La deuxième étape vise à identifier les caractères qui suivent chaque N-gramme et compter leurs occurrences pour connaître leur probabilité. Par exemple,

— le 1-gramme `"a"` est suivi par les caractères `'l'`, `'␣'` et `','` avec respectivement 2, 3 et 1 occurrences, donc les probabilités 2/6, 3/6 et 1/6.
— le 1-gramme `"n"` est suivi par les caractères `'j'`, `'t'` et `'␣'` avec respectivement 1, 1 et 1 occurrences, donc les probabilités 1/3, 1/3 et 1/3.
— le 2-grammes `"en"` est suivi par les caractères `'t'` et `'␣'` avec respectivement 1 et 1 occurrences, donc les probabilités 1/2 et 1/2.

En choisissant une taille de N-grammes, c'est-à-dire en fixant N, on dispose alors de probabilités conditionnelles qui permettent de prédire le caractère qui fait suite à une séquence de caractères donnée. Ainsi,

— Avec un modèle de 1-grammes, si on considère que l'utilisateur tape le caractère `'a'` alors la probabilité que son caractère suivant soit `'l'` est de 2/6, `'␣'` est de 3/6 et `','` est de 1/6. Un prédicteur privilégiera donc `'␣'`.
— Toujours avec un modèle de 1-grammes, si l'utilisateur tape le caractère `'n'` alors la probabilité que son caractère suivant soit `'j'` est de 1/3, `'t'` 1/3 et `'␣'` 1/3. Un prédicteur choisira l'un des trois caractère au hasard.
— Avec un modèle de 2-grammes, si on considère les caractères `"en"` alors la probabilité que le caractère suivant soit `'t'` est de 1/2 et de même pour le caractère `'␣'`. On choisira là aussi l'un des deux au hasard.

1.2 Modèle de 1-grammes

Cette partie doit être réalisée en C.

Question 1. Proposer une structure de données pour encoder un modèle de 1-grammes. Un modèle de 1-grammes contient un ensemble de 1-grammes et, pour chacun, ses successeurs et leurs occurrences. Pour chaque 1-gramme présent dans le modèle, vous préciserez comment on accède aux successeurs, comment on connaît leur nombre d'occurrences et comment on connaît la probabilité conditionnelle associée à chaque caractère suivant.

Question 2. Proposer une fonction qui construit un modèle de 1-grammes à partir d'un texte. Cette fonction prend en entrée le texte. Elle renvoie le modèle de 1-grammes.

Question 3. Proposer une fonction qui prédit la suite d'une séquence de caractères. Cette fonction prend en entrée un modèle de 1-grammes, la séquence de caractères et un entier qui précise le nombre maximum de caractères contenus dans la suite prédite. Cette fonction renvoie la suite de caractères qui a été prédite à partir du modèle.

1.3 Modèle de N-grammes

Cette partie doit être réalisée en OCaml.

Question 4. Proposer une structure de données pour encoder un modèle de N-grammes avec un N donné. Vous ferez en sorte que cette structure permette de retrouver les modèles dont la taille est inférieure à N. Par exemple, un modèle de 2-grammes devra contenir les 1-grammes, un modèle de 3-grammes devra contenir les 2-grammes et les 1-grammes, etc.

Question 5. Proposer une fonction qui construit le modèle de N-grammes à partir d'un texte. Cette fonction prend en entrée le texte et un entier N.

Question 6. Proposer une fonction qui prédit la suite d'une séquence de caractères à partir d'un modèle de N-grammes. À la différence de la question 3, cette fonction prend en entrée un modèle de N-grammes qui contient les modèles de taille inférieure. De fait, pour chaque taille, le modèle peut contenir des probabilités différentes. Avec notre exemple et un modèle de 2-grammes, si l'utilisateur tape la séquence `"en"` alors on a le 2-gramme `"en"` avec `'t'` ou `'␣'` avec chacun 1/2 de probabilité, mais on a aussi le 1-gramme `'n'` qui a `'j'`, `'t'` et `'␣'` avec chacun la même probabilité 1/3. Vous proposerez alors une stratégie permettant de choisir parmi les différentes solutions.

1.4 Fils d'exécution (*threads*)

Cette partie doit être réalisée en OCaml.

La construction d'un modèle de N-grammes peut prendre du temps surtout si le texte d'entrée est très long. Pour optimiser la construction, on souhaite mettre en place une approche exploitant la concurrence, avec plusieurs fils d'exécution (*threads*).

Question 7. Proposer une approche pour répartir la construction du modèle en construisant plusieurs modèles, chacun étant construit sur une partie du texte.

Question 8. Proposer une approche qui permet de fusionner tous les modèles construits dans la question 7 afin de construire un unique modèle.

2 Audit de code C et OCaml

Dans cette partie de l'épreuve, il est demandé au candidat d'étudier un fichier source qui comporte des erreurs ou des maladresses, de qualité de code ou de fonctionnement, et il est demandé d'auditer ce fichier, c'est-à-dire :

- de comprendre et d'être capable d'expliquer le fonctionnement du code à l'oral ;
- de proposer des corrections en réécrivant certaines parties afin de corriger les erreurs ou maladresses éventuelles et de rendre le code plus clair, notamment dans une optique pédagogique ;
- de proposer des améliorations de la complexité en temps ou en mémoire, ou de la sûreté du code.

Le temps indicatif de préparation de cette partie est d'une heure. Le code ci-dessous est disponible dans votre compte, à la racine, sous les noms `~/audit1.c` et `~/audit2.ml` respectivement.

2.1 Code en C

```c
#include <stdbool.h>
#include <stdlib.h>
#include <assert.h>

typedef struct Cell { int value; struct Cell *next; } list;

// J'ai trouvé un algorithme vraiment merveilleux ! R. F.
bool list_cyclic(list *l) {
  list *tortoise = l, *hare = l->next;
  while (tortoise != hare) {
    if (hare == NULL) return 0;
    hare = hare->next->next;
    tortoise = tortoise->next;
  }
  return 1;
}

// Et j'ai bien vérifié qu'il fonctionne correctement.
int main() {
```

```
20    list l1, l2, l3;
21    l1.next = &l2;
22    l2.next = &l3;
23    l3.next = NULL;
24    assert(!list_cyclic(&l1));
25    l3.next = &l2;
26    assert(list_cyclic(&l1));
27    assert(list_cyclic(&l2));
28    assert(list_cyclic(&l3));
29  }
```

2.2 Code en OCaml

```
1   (* man pstree *)
2
3   open Printf
4
5   type t = N of string * t list
6
7   let rec print1 pref (N (s, tl)) =
8     printf "%s" s;
9     if tl <> [] then
10    let w = String.length s in
11    let pref' = pref ^ String.make w ' ' in
12    match tl with
13    | [t'] -> printf "---"; print1 (pref' ^ "␣␣") t'
14    | _ -> printf "-"; print2 pref' "+-" tl
15
16  and print2 pref start = function
17    | [s] ->
18        printf "`-"; print1 (pref ^ "␣␣") s
19    | s :: sons ->
20        printf "%s" start; print1 (pref ^ "|␣") s;
21        printf "\n"; printf "%s" pref;
22        print2 pref "|-" sons
23
24  let print t =
25    print1 "" t
```

3 Annexe : tirage aléatoire

En OCaml comme en C, on peut tirer un entier pseudo-aléatoire entre 0 inclus et `n` exclu, de la manière suivante :

— en OCaml, avec `Random.int n`,

— en C, avec `rand() % n`, en ayant chargé `stdlib.h`.

Dans le deux cas, l'entier `n` doit être strictement supérieur à 0.

* *
*