DS 2

Option informatique, première année

Julien Reichert

Exercice 1 : Dans une société futuriste, les citoyens sont évalués sur une base biennale, et du résultat de cette évaluation dépend leur poste et leur nom. [1] Ainsi, le meilleur citoyen est le « Président » et porte pour nom A pendant deux ans, s'ensuivent les vingt-cinq ministres dans l'ordre alphabétique, puis les « 2-lettrés » de AA à ZZ, et ainsi de suite pour l'ensemble de la population [2]. Écrire une fonction qui attribue un nom à un dividu [3] en fonction de son classement, et la fonction réciproque. Le nom sera une chaîne de caractères, constitué uniquement de lettres capitales de l'alphabet français.

Exercice 2 : Distance d'édition (Levenshtein).

Le but de cet exercice est d'écrire un algorithme déterminant la distance, selon une unité virtuelle, entre deux chaînes de caractères, avec comme application le meilleur moyen de « corriger » une saisie erronée, mais aussi et surtout la génétique et l'étude de brins d'ADN.

Transformer une chaîne de caractères peut se faire suivant trois opérations élémentaires : insérer un caractère, en supprimer un ou en remplacer un. Dans un premier temps, on considèrera que le coût de chacune de ces opérations est toujours d'un. [4]

Question 2.1 : Écrire en Caml un programme dynamique qui détermine le coût sous ces conditions pour passer d'une chaîne de caractères à une autre.

Le principe de l'algorithme est le suivant : pour deux chaînes de tailles respectives `m` et `n`, on crée une matrice à `m+1` lignes et `n+1` colonnes (ou vice-versa, bien entendu) telle qu'à la position `(i, j)` figure le coût pour passer de la tranche des `i` premiers caractères de la première chaîne à la tranche des `j` premiers caractères de la deuxième chaîne.

Cette matrice contient `k` (car le coût des insertions et suppressions est d'un pour commencer) sur la première ligne à toute colonne `k` et sur la première colonne à toute ligne `k`, puis elle se remplit suivant la relation de récurrence suivante : pour `i, j` strictement positifs, la valeur en `(i, j)` est la plus faible parmi la valeur en `(i-1, j)` plus un, la valeur en `(i, j-1)` plus un ou la valeur en `(i-1, j-1)` plus (un si le caractère d'indice `i-1` de la première chaîne est différent du caractère d'indice `j-1` de la deuxième chaîne, zéro sinon). Il s'agit en fait du meilleur choix entre insérer, supprimer ou (remplacer ou simplement avancer) dans la progression de la première chaîne à la deuxième.

On comprendra aisément comment trouver l'information demandée à partir de la matrice.

Question 2.2 : Adapter la fonction précédente à un coût non identique mais constant, donné en tant qu'argument supplémentaire sous la forme d'un triplet (insertion, suppression, remplacement).

Pour ceux qui composent sur une feuille, il est possible de baliser bien explicitement sur la fonction précédente les passages à remplacer et de donner la nouvelle version. Dans un fichier, le copier-coller est envisageable si la fonction précédente est correcte...

1. Le thème de cet exercice reprend le contexte d'un roman d'Alain Damasio.
2. les noms sont limités à 5 lettres dans le roman, pas dans l'exercice
3. notion qui a remplacé l'individu, mais tout ceci est dans le roman...
4. On notera qu'il n'est pas pertinent de fixer un coût pour le remplacement supérieur à la somme des deux autres coûts.

Exercice 3 : Réaliser une structure de table de hachage, à partir du type ci-dessous et de la spécification des cinq fonctions à écrire.

Une table de hachage est une structure analogue à un dictionnaire, permettant d'associer à une clé de n'importe quel type (qu'on impose cependant d'être immutable) une valeur (comme on programme en Caml, la valeur ne pourra être que d'un seul type dans une table de hachage précise).

Dans un souci d'optimisation de la complexité, au regard des deux implémentations de la structure de dictionnaire du cours, les données sont placées dans un tableau à un indice déterminé par une fonction dite « fonction de hachage », prenant comme argument n'importe quelle valeur du type des clés et retournant un entier naturel majoré par la capacité du tableau (souvent par une opération de modulo). En cas de « conflit », c'est-à-dire dans le cas, qui est en pratique rendu peu probable au regard de la taille du tableau par rapport au nombre de clés utilisées, où plusieurs clés ont la même valeur par la fonction de hachage, les données cohabitent à l'indice commun au sein d'une liste. Une fois de plus, le typage de Caml intervient et force à mettre dans ce cas des listes partout dans le tableau. Il s'agit alors, en cas de recherche, de localiser la clé dans une liste a priori de taille suffisamment restreinte pour que le temps passé demeure faible.

Le type retenu pour les tables de hachage est le suivant :

```
type ('a, 'b) table_hachage = { fonction : 'a -> int; donnees : ('a * 'b) list array};;
```

Attention : Écrire la fonction dans l'accolade au moment de créer une table de hachage peut déclencher une erreur de syntaxe en raison de l'ambiguïté du point-virgule. Mieux vaut la créer à part en cas de test.

Les cinq fonctions à écrire sont :

- Création d'une table de hachage, avec comme arguments une fonction et la taille des données. On ne vérifiera évidemment pas que la fonction est bien bornée.
  La signature est : `cree : ('a -> int) -> int -> ('a, 'b) table_hachage`.
  On notera que le type des données n'est pas fixé à ce stade.
- Ajout d'un couple (clé, valeur) à une table de hachage, en déclenchant une erreur si la clé y figure déjà.
  La signature est : `ajoute : ('a * 'b) -> ('a, 'b) table_hachage -> unit`.
- Modification de la valeur associée à une clé, en profitant du fait que, bien que les listes ne soient pas mutables, elles sont dans un tableau et donc remplaçables. Il faut cette fois déclencher une erreur si la clé ne figure pas dans la table de hachage.
  La signature est : `modifie : 'a -> 'b -> ('a, 'b) table_hachage -> unit`.
- Suppression de l'entrée correspondant à une clé, dans les mêmes conditions que la modification.
  La signature est : `supprime : 'a -> ('a, 'b) table_hachage -> unit`.
- Consultation de la valeur associée à une clé, toujours dans les mêmes conditions.
  La signature est : `consulte : 'a -> ('a, 'b) table_hachage -> 'b`.