DS 2

Informatique de tronc commun, classe de PC

Julien Reichert

Ce devoir consiste en une partie de programmation, une partie sur une bases de données et un problème.

Les parties sont indépendantes et les exercices ainsi que les parties ne sont pas forcément de difficulté croissante.

Partie 1 : Programmation

Toutes les questions de cette section sont à traiter en Python. L'utilisation de dictionnaires est recommandée en général mais n'est pas obligatoire.

Dans les quatre premiers exercices, on suppose (inutile de le vérifier) que les arguments des fonctions à écrire seront des listes de chaînes de caractères qui sont toutes des mots en minuscules, et on appellera ces arguments des phrases pour raccourcir les consignes.

Les autres exercices sont indépendants.

Exercice 1.1 : Écrire une fonction prenant en argument une phrase et renvoyant le nombre de mots **différents** commençant par une voyelle dans la liste.

Exercice 1.2 : Écrire une fonction prenant en argument une phrase et renvoyant la liste des voyelles par lesquelles au moins un mot commence.

Exercice 1.3 : Écrire une fonction prenant en argument une phrase et renvoyant la liste des lettres apparaissant dans au moins un mot.

Exercice 1.4 : Écrire une fonction prenant en argument une phrase et renvoyant la lettre apparaissant le plus souvent au total dans l'ensemble des mots. En cas d'égalité on renverra au choix la liste des lettres à égalité ou n'importe laquelle d'entre elles.

Exercice 1.5 : Écrire une fonction prenant en argument un entier et renvoyant la taille de la chaîne de caractères qui représente cet entier en chiffres romains selon la convention usuelle. On pourra supposer sans le vérifier que l'entier est compris entre `1` et `3999`.

Exercice 1.6 : Écrire une fonction prenant en argument un entier positif (pas à vérifier) noté `n` et renvoyant le plus grand entier qui soit inférieur ou égal à `n` et qui soit par ailleurs le carré ou le cube d'un entier.

Exercice 1.7 : Écrire une fonction prenant en argument une liste et renvoyant le nombre de « phases de monotonie » dans la liste, en définissant une phase de monotonie comme une sous-liste monotone d'éléments consécutifs de la liste qu'on ne peut pas prolonger sans perdre la monotonie (ou parce qu'on est à un bout de la liste). Attention, la liste elle-même pourra avoir des éléments consécutifs égaux !

Partie 2 : Base de données « F1 » [1]

Depuis la création des Grands Prix, la Fédération Internationale de l'Automobile conserve l'ensemble des résultats des différentes courses. Utiliser une base de données permet notamment de réaliser les classements lors de chacune des saisons mais également de réaliser des statistiques historiques. La base de données que nous allons utiliser est très allégée et comporte trois tables.

La table `SAISONS` contient les colonnes :
- `date` : date du Grand Prix (de type date, affichée au format `aaaa-mm-jj`) ;
- `grand_prix` : nom du Grand Prix (de type chaîne de caractères) ;
- `pays` : nom du pays où le Grand Prix est organisé (de type chaîne de caractères) ;
- `continent` : continent où le Grand Prix est organisé (de type chaîne de caractères) ;

date	grand_prix	pays	continent
1950-05-13	Silverstone	Royaume-Uni	Europe
1950-05-21	Monaco	Monaco	Europe
...	...	...	...
2017-11-12	São Paulo	Brésil	Amérique du Sud
2017-11-26	Abou Dabi	Émirats arabes unis	Asie

FIGURE 1 – Table SAISONS

La table `GP` contient les données de la saison en cours uniquement (2017 pour notre sujet), avec les colonnes :
- `grand_prix` : nom du Grand Prix (cf. la table précédente) ;
- `pilote` : nom du pilote (de type chaîne de caractères) ;
- `cl_qualif` : classement à l'issue de la phase des qualifications (entier) ;
- `cl_course` : classement à l'issue de la course (entier) ;
- `points` : nombre de points obtenu lors du Grand Prix (entier).

grand_prix	pilote	cl_qualif	cl_course	points
Melbourne	Vettel	1	1	25
Melbourne	Hamilton	3	2	18
Melbourne	Bottas	2	3	15
...	...	...	...	...

FIGURE 2 – Table GP

La table `TOURS` contient les données concernant les tours d'un Grand Prix : à chaque fois qu'un pilote termine un tour (de numéro entre 1 et le nombre de tours du Grand Prix), une entrée est ajoutée dans cette table avec le temps mis depuis la fin du tour précédent (ou le départ pour le premier tour). Si un pilote subit une pénalité, une entrée est également ajoutée, avec le numéro 0, contenant le temps de pénalité cumulé sur la course. **S'il abandonne la course, il manquera des enregistrements dans la table.** Les colonnes sont :
- `date` : date du Grand Prix ;
- `pilote` : nom du pilote (cf. table précédente) ;
- `num_tour` : numéro du tour accompli (entier) ;
- `temps_tour` : temps en secondes mis pour le tour (de type flottant) avec 1 ms de précision.

date	pilote	num_tour	temps_tour
2017-03-26	Vettel	1	`88.542`
2017-03-26	Vettel	2	`87.048`
2017-03-26	Vettel	3	`86.845`
...	...	...	...

FIGURE 3 – Table TOURS

Exercice 2.1 : Écrire une requête SQL qui renvoie l'ensemble des Grands Prix, ainsi que leur date, s'étant déroulés sur le continent européen.

1. La base de données ci-après avait été écrite par mon co-auteur pour le sujet de Centrale 2022. Elle n'a pas été retenue dans la version finale du sujet, et il serait temps qu'elle soit utilisée (à de légères modifications près) vu que son écriture date de 2017...

Exercice 2.2 : Que réalise la requête SQL suivante ? Donner un exemple de résultat.

```
SELECT pilote, SUM(points) FROM GP GROUP BY pilote
```

Exercice 2.3 : Écrire une requête SQL qui renvoie les dates et noms des Grands Prix correspondant à des courses lors desquelles le pilote ayant commencé en pole position a fini en tête, ces courses devant avoir eu lieu en 2017 sur le continent asiatique. On rappelle que le terme de pole position signifie que le pilote a réalisé le meilleur temps de la phase de qualifications.

Exercice 2.4 : Écrire une requête SQL qui renvoie le nom du pilote en tête du classement de la saison 2017, en ne tenant compte que des points. En cas d'égalité, un nom suffit.

Exercice 2.5 : Écrire une requête SQL qui renvoie le nom du pilote qui a gagné la course du 1er mai 1994 en ne se servant que de la table `TOURS`.

Partie 3 : Subset-Sum

Le problème de la sous-somme, bien connu en théorie de la complexité sous son nom anglais de *Subset-Sum*, consiste à déterminer s'il existe une sous-séquence d'une séquence d'entiers dont la somme est égale à un entier donné. Ce problème est NP-complet, mais là n'est pas le sujet. Une version du problème consiste à exhiber en cas de réponse positive une sous-séquence de somme la valeur souhaitée.

Une approche possible pour résoudre le problème de la sous-somme est de faire intervenir la mémoïzation, mais avant d'en arriver là quelques questions préliminaires permettront de broder autour du problème et en douceur.

Exercice 3.1 : Écrire une fonction prenant en argument une liste d'entiers et renvoyant sa somme.

Exercice 3.2 : Écrire une fonction prenant en argument une liste d'entiers `seq` ainsi qu'une autre liste `indseq` contenant des entiers différents deux à deux et compris entre `0` et la taille de `seq` moins un (ceci ne sera pas à vérifier) et renvoyant la somme des entiers de `seq` aux indices figurant dans `indseq` et seulement ceux-ci.

Exercice 3.3 : Quelle est la complexité de la fonction précédente ?

Exercice 3.4 : Proposer une méthode naïve de résolution du problème de la sous-somme à l'aide de la fonction précédente et estimer la complexité en temps dans le pire des cas si cette méthode est mise en œuvre. **Attention, il ne s'agit pas de programmer ici !**

Pour résoudre le problème par mémoïzation, la possibilité proposée est de considérer chaque élément de la liste, en découvrant de nouvelles sommes possibles à chaque fois à partir des sommes déjà trouvées.

En admettant qu'on ne veuille pas utiliser de dictionnaires, dans une version simple de l'algorithme, il faudrait créer un tableau avec un décalage d'indices de sorte de pouvoir couvrir toutes les sommes possibles, de la plus petite (somme de tous les éléments négatifs de la liste) à la plus grande (somme de tous les éléments positifs de la liste).

Exercice 3.5 : Écrire une fonction qui prend en argument une liste et qui renvoie une liste de taille deux contenant les deux sommes en question (celle des négatifs et celle des positifs).

Pour éviter d'emprunter cette route risquée du point de vue des bugs, on se forcera à créer un dictionnaire indexé par les sommes possibles. Afin de rentabiliser l'utilisation de dictionnaires plutôt que des ensembles, on prendra soin d'associer à ces clés des valeurs qui permettent de produire la clé en tant que somme d'une sous-liste de la liste.

Exercice 3.6 : Résoudre alors par la méthode exposée ci-avant le problème de la sous-somme.

Exercice 3.7 : Déterminer la complexité de la fonction précédente.