Concours Blanc : corrigé

PTSI A et B Lycée Eiffel

28 mai 2015

Première partie : Code ISBN d'un ouvrage.

1. Chacun des caractères peut être codé sur 4 bits puisqu'il y a 10 ou 11 possibilités (donc moins que $2^4 = 16$). Puisqu'il y a 10 caractères à coder, 40 bits, soit 5 octets, seront suffisants. Si on tient vraiment à minimiser l'espace mémoire, on peut faire les choses globalement : il y a 11×10^9 codes possibles. On sait que 2^{10} est légèrement supérieur à 10^3, donc $4 + 3 \times 10$ bits seront suffisants, soit 4 octets et 2 bits.
2. On calcule $2+2\times2+3\times6+4\times6+5+6+7+5\times8+6\times9 = 2+4+18+24+5+6+7+40+54 = 160$. Comme $160 = 11 \times 14 + 6$, le reste de la division par 11 vaut 6, et le dernier chiffre du code est donc 6.
3. Une façon de faire est d'isoler le dernier chiffre du nombre (en prenant le reste de sa division par 10), puis de l'enlever (en faisant la division entière par 10), et de recommencer. Attention quand même, en procédant ainsi, on va récupérer les chiffres en commençant par le dernier, il faut les remettre dans l'ordre.

```
def listechiffres(n) :
    a=n
    l=[]
    while a!=0 :
        l=[a%10]+l
        a=a//10
    return l
```

4.
```
def clecode(n) :
    l=listechiffres(n)
    s=sum((i+1)*l[i] for i in range(10))
    s=s%11
    if s==10 :
        return 'X'
    else :
        return s
```

Il faut simplement faire attention à la numérotation des indices dans la liste qui se fait à partir de 0, et bien sûr gérer le cas particulier du caractère X. Ensuite, on va se contenter d'ajouter cette clé à la fin du nombre n, on va toutefois être obligés de toujours ressortir une chaine de caractères, et donc de convertir le nombre entier :

```
def completecode(n) :
    a=clecode(n)
    if a=='X' :
        return str(n)+a
```

```
    else :
        return str(n)+str(a)
```

5. Encore une fois, les seules difficultés sont dus à la gestion du caractère X :

```
def verifiecode(c) :
    n=int(c[0 :8])
    a=c[9]
    if clecode(n)==int(a) or clecode(n)==a :
        return True
    return False
```

   Il y a deux conditions différentes dans le if à cause du fait que clecode renvoie soit un entier, soit un caractère (on ne peut pas gérer les deux simultanément). Comme une fonction Python ne retourne qu'une seule valeur, si la condition du if est vérifiée, l'exécution du programme s'arrêtera ; sinon on retournera la valeur False.

6. Augmenter l'un des chiffres d'une unité va augmenter la clé d'une valeur égale au rang du chiffre dans le code ISBN (modulo 11). Ainsi, si c'est le sixième chiffre qui est augmenté accidentellement par exemple, la clé va être augmentée de 6 (si elle était égale à 9, elle vaudra désormais 15 modulo 11, soit 4). Il suffit donc de calculer le code correspondant au début de code incorrect, et de lui soustraire le vrai code. Le chiffre obtenu va nous donner le rang du chiffre incorrect, qu'il suffira de baisser d'une unité pour corriger l'erreur. Si on ne sait pas que l'erreur a été une augmentation d'une unité, on ne peut pas corriger : augmenter le deuxième chiffre de trois unités ou le sixième d'une unité aura le même effet sur le code. De même, on ne peut pas repérer deux erreurs simultanées : augmenter le premier et le sixième chiffres d'une unité aura le même effet qu'augmenter le troisième et le quatrième chiffres d'une unité.

7. (a) On complète avec le 978 puis on calcule $9+3\times7+8+3\times2+2+3\times6+6+3+1+3+5+3\times6 = 100$. La clé est donc 0, et le code ISBN13 complet est 9782266111560.

   (b) Il faut encore une fois faire attention à la numérotation des indices qui commence à 0 :

```
def conversioncode(c) :
    n=978*10**9+int(c[0 :8])
    l=listechiffres(n)
    s=sum(l[2*i]*3*l[2*i+1] for i in range(6))
    cle=(10-s%10)%10
    return 10*n+cle
```

   (c) Ah, une question triviale pour commencer, il suffit de regarder si le quatrième chiffre du code (une fois qu'on a ajouté 978 devant) est un 4 :

```
def livrejaponais(c) :
    if c[3]==4 :
        return True
    return False
```

Deuxième partie : Utilisation d'un tri par table de hachage.

1. Le Standard ASCII (**A**merican **S**tandard **C**ode for **I**nternational **Interchange**) est un code associant un nombre entier à tous les caractères usuels (128 caractères numérotés dans l'ASCII d'origine, qui ne contenait pas de caractères accentués). Ce code est essentiellement obsolète aujourd'hui puisqu'il ne permet bien sûr pas de coder des textes écrits avec autre chose qu'un alphabet latin réduit.

2. On suppose pour simplifier que tous les caractères seront non accentués. On peut par contre gérer les majuscules sans problème à l'aide de la méthode lower sur les chaines de caractères (qui passe tout en minuscule).

```
def rangalpha(c) :
    return ord(c.lower())-96
```

3. On prend les deux premiers caractères : P a pour rang alphabétique 16 et Y a pour rang 25, ce qui donne un code associé à ce titre égal à $26 \times 15 + 25 = 415$.
4. C'est très simple en utilisant la fonction déjà écrite plus haut :

```
def code(s) :
    a,b=ord(s[0].lower()),ord(s[1].lower())
    if 96 < a and 96 < b and a < 123 and b < 123 :
        return 26*rangalpha(s[0])+rangalpha(s[1])
    return 0
```

5. Il faut d'abord créer une liste de listes contenant autant de cases qu'il n'y a de codes possibles pour les différents titres, à savoir $26^2 + 1$ (un de plus pour tous ceux qui auront pour code 0, et 26^2 selon les deux premières lettres, on vérifie facilement que les codes utilisés sont bien les nombres entiers entre 0 et 26^2), puis on insère chaque titre dans la liste, à la bonne place :

```
def trihachage(liste) :
    l=[[]*(26**2+1)]
    for i in liste :
        l[code(i)].append(i)
    return l
```

6. C'est très simple : pour chaque élément de la liste, on a un calcul à faire (celui du code). Si on considère que les append vont prendre un temps négligable, le temps d'exécution est simplement proportionnel à la taille de la liste à trier, l'algorithme est donc linéaire. C'est nettement plus performant que les algorithmes classiques, mais on peut noter au moins deux défauts évidents :
   - il est très gourmand en place mémoire, puisqu'il faut créer une grosse liste avant de mettre les titres dedans. Bien sûr, on peut penser que, si la liste de livres contient des dizaines de milliers de titres, ce défaut deviendra négligeable, mais en fait ce n'est pas vrai : si on veut conserver un tri efficace, il faudra éviter d'avoir énormément de titres dans la même case de la liste (cf ci-dessous), et donc affiner le codage, ce qui reviendra à augmenter encore la taille de notre liste.
   - le tri effectué n'est pas complet, puisque tous les livres dont le titre commence par les deux mêmes premières lettres seront triés au même endroit. Si on voulait un tri exact, il faudrait refaire un tri sur chacune des sous-listes obtenues dans les cases, et on se retrouverait globalement avec un algorithme qui ne serait pas plus performant qu'un bon algorithme de tri classique.
7. C'est très simple : on calcule le code correspondant à son titre, et on va récupérer les titres se trouvant dans la case correspondante de notre grosse liste. Il reste une petit recherche à faire dans cette sous-liste pour y trouver notre titre.

Troisième partie : Exploitation du fichier de prêts.

1. C'est du cours ! On dispose du mode lecture ('r' pour read en Python), qui permet d'accéder au contenu du fichier sans le modifier ; du mode écriture ('w' pour write) qui permet d'écrire dans un fichier initialement vierge ; et du mode ajout ('a' pour add) qui permet d'ajouter du contenu à la fin d'un fichier déjà existant.

2.
```
f=open(prets.txt,'r')
l=[]
for i in f.readlines() :
    l.append(i.split(',')[1])
print l
f.close()
```

3. Il suffit de rajouter une condition sur la date :

```
f=open(prets.txt,'r')
l=[]
for i in f.readlines() :
    s=i.split(',')
    if s[4]<20150528 :
        l.append(s[1])
print l
f.close()
```

4. Je vais donc créer une liste constituée d'éléments de types différentes (ça ne pose pas de problèmes), les éléments d'indices pairs étant les noms des emprunteurs et ceux d'indices impairs les nombres de livres empruntés. À chaque fois qu'on tombe sur un emprunteur, on vérifie s'il est déjà dans la liste (avec count), on l'ajoute si ce n'est pas le cas et on modifie simplement son nombre de livres s'il est déjà présent.

```
f=open(prets.txt,'r')
l=[]
for i in f.readlines() :
    s=i.split(',')
    if l.count(s[1])==0 :
        l.append(s[1])
        l.append(1)
    else :
        j=l.index(s[1])
        l[j+1]=l[j+1]+1
print(l)
f.close()
```

Quatrième partie : Base de données.

1. Il faut prendre le couple (n°livre,n°lecteur).
2. CREATE TABLE lecteurs (n°lecteur : interger PRIMARY KEY, nom : varchar(20), prenom : varchar(20), datedenaissance : date)
3. Dans l'algèbre relationnelle :
   $\Pi_{titre}(\sigma_{nom=Knuth}((\text{livre} \bowtie_{livre.n^\circ livre=emprunt.n^\circ livre} \text{emprunt})) \bowtie_{lecteurs.n^\circ lecteur=emprunt.n^\circ lecteur}$ lecteurs)
   En SQL : SELECT titre FROM emprunt,livre,lecteur WHERE livre.n°livre=emprunt.n°livre AND lecteur.n°lecteur=emprunt.n°lecteur AND nom='Knuth'
4. ALTER TABLE livre ADD etat : varchar(20)
5. SELECT titre FROM emprunt,livre,lecteur WHERE livre.n°livre=emprunt.n°livre AND lecteur.n°lecteur=emprunt.n°lecteur AND auteur LIKE 'G%' AND nom LIKE 'L%'