DS d'informatique : corrigé

PTSI Lycée Eiffel

27 novembre 2015

Exercice 1

1. On peut par exemple citer Windows, MacOS et Linux, mais rien n'interdit d'aller chercher du côté des systèmes d'exploitation dédiés aux smartphones (Android par exemple). Donner plusieurs versions d'un même système d'exploitation (Windows 95 et Windows 7, ou même Ubuntu et Linux) ne répond pas contre pas vraiment à la question.
2. Il s'agit de l'ENIAC, créé à la fin de la seconde guerre mondiale à des fins militaires (décryptage de messages codés notamment).
3. La mémoire vive est beaucoup plus rapide d'accès (en gros un facteur 10 par rapport à la mémoire morte), elle sert à stocker des informations de façon temporaire, et elle est en particulier effacée lors de la mise hors tension de l'ordinateur. Au contraire, la mémoire morte est une mémoire à long terme, utilisée notamment pour stocker tous les fichiers de données et et autres programmes installés sur l'ordinateur.

Exercice 2

1. On peut fonctionner en conservant trois valeurs successives de la suite dans trois variables u, v et w :

```
def calculsuite(n) :
    u,v,w=1,1,1
    for i in range(n-2) :
        u,v,w=v,w,2*u+v
    return w
```

2. Je n'ai mis que $n-2$ passages dans la boucle for puisqu'on connait initialement u_2, il suffit donc de calculer $n-2$ nouveaux termes de la suite pour aller jusqu'à u_n. Si on oublie les affectations de variables, les seuls calculs effectués sont donc $n-2$ multiplications par 2, et $n-2$ additions.
3. On utilise le même range, et on prend les bons éléments dans la liste pour le calcul de u_{n+3} : u_{n+1} se trouve alors en avant-dernière position, et u_n juste avant.

```
def listetermes(n) :
    l=[1,1,1]
    for i in range(n-2) :
        l.append(2*l[-3]+l[-2])
    return l
```

4. Il suffit de remplacer la dernière ligne du programme précédent par le bloc suivant :

```
s=0
for i in l :
        s=s+i
return s
```

   En plus des $n-2$ additions et multiplications effectués dans la première boucle for, ce programme fera encore $n+1$ additions pour calculer la somme, soit $2n-1$ additions et $n-2$ mutliplications par 2 au total.

5. Allons-y :

```
def miaou(n) :
        s=0
        l=listetermes(n)
        for i in range(n+1) :
                s=s+l[i]*l[-1-i]
        return s
```

Exercice 3

1. On ne peut pas faire plus simple pour créer des listes indépendantes car Python se contente de copier les adresses des variables si on fait une simple copie de listes. Même la commande $l1 = l[:]$ ne fonctionnerait pas car les éléments de l seraient bien copiés dans l1, mais comme ces éléments sont eux-mêmes des listes (on est en présence d'une liste de listes), cette copie ne serait pas correctement effectuée.
2. Les deux premières lignes supposent déjà une liste de listes puisqu'il y a des doubles crochets. Les append à l'intérieur des deux boucles for aussi, puisqu'ils supposent que les éléments des listes l1 et l2 (matérialisés par les variables i et j), sont eux-mêmes des listes.
3. Elles se contentent d'ajouter des 0 (respectivement des 1), au bout de chacune des listes constituant la liste l1(respectivement l2).
4. On inverse l'ordre des éléments de la liste l2, puis on renvoit un résultat obtenu en mettant bout à bout les listes l1 et l2. En particulier, on aura toujours à la sortie une liste deux fois plus longue qu'au départ, et chacune des listes internes aura un élément de plus qu'au départ (à cause des append dans les boucles for).
5. On obtient dans un premier temps la liste $[[0,0],[1,0],[1,1],[0,1]]$, puis dans un deuxième temps la liste $[[0,0,0],[1,0,0],[1,1,0],[0,1,0],[0,1,1],[1,1,1],[1,0,1],[0,0,1]]$.
6. Il suffit de faire une petite boucle for :

```
def gray(n) :
        l=[[0],[1]]
        for i in range(n-1) :
                l=mystere(l)
        return l
```

7. Le résultat obtenu sera toujours une liste constitué de 2^n listes, elles-même contenant n éléments. On démontre en fait facilement par récurrence que ces 2^n listes sont toutes les listes distinctes de n chiffres égaux à 0 ou 1. C'est le cas pour $n = 1$ (le programme sort juste la liste l), et si c'est vrai au rang n, chaque mot binaire de n bits apparaitra exactement une fois dans

la liste gray(n), et sera donc utilisé deux fois pour le calcul de gray(n+1)=mystere(gray(n)) : une fois dans la liste l1 (on lui ajoutera un 0) et une fois dans la liste l2 (on lui ajoutera un 1). On retrouvera donc dans gray(n+1) exactement deux mots de $n+1$ bits commençant par un mot de n bits donnés, l'un terminé par un 0 et l'autre par un 1. Cela prouve que tous les mots de $n+1$ bits apparaissent dans gray(n+1). Si on est un peu plus courageux, on peut même prouver par récurrence que deux éléments consécutifs de la liste gray(n) ne différent que d'un bit (autrement dit, les listes ont les mêmes éléments aux même endroits, sauf pour un unique indice), et que c'est également le cas pour le premier et le dernier élément de la liste, ce qui prouve qu'on obtient une liste correspondant à la numérotation binaire appelée codage de Gray et évoquée dans le cours. Cette preuve n'est pas si dure : prenons deux listes éléments consécutifs de gray(n+1), si elles sont toutes les deux dans la première moitié de la liste, elle sont issues de deux éléments de gray(n) auxquels on a ajouté un 0 (éléments de l1 dans le programme mystere), donc elles ont le même bit d'écart que ces deux éléments dont elles sont issues. Même chose si elles sont dans la deuxième moitié (on a ajouté un 1 au lieu du 0 et inversé l'ordre, ça ne change rien). Si on prend maintenant les deux listes « au milieu » de gray(n+1) (la dernière de la première moitié et la première de la deuxième moitié), elles sont respectivement issues de la dernière liste de gray(n) augmentée d'un 0, et de la **dernière** liste de gray(n) augmentée d'un 1 (puisqu'on a retourné l'ordre dans la deuxième moitié), et seul leur dernier bit diffère. Même chose pour le premier et le dernier élément de gray(n+1), qui sont issus de la première liste de gray(n), augmentée respectivement d'un 0 ou d'un 1.

8. Le résultat obtenu sera toujours une liste constitué de 2^n listes, elles-même contenant n éléments. On démontre en fait facilement par récurrence que ces 2^n listes sont toutes les listes distinctes de n chiffres égaux à 0 ou 1. C'est le cas pour $n = 1$ (le programme sort juste la liste l), et si c'est vrai au rang n, chaque mot binaire de n bits apparaitra exactement une fois dans la liste gray(n), et sera donc utilisé deux fois pour le calcul de gray(n+1)=mystere(gray(n)) : une fois dans la liste l1 (on lui ajoutera un 0) et une fois dans la liste l2 (on lui ajoutera un 1). On retrouvera donc dans gray(n+1) exactement deux mots de $n+1$ bits commençant par un mot de n bits donnés, l'un terminé par un 0 et l'autre par un 1. Cela prouve que tous les mots de $n+1$ bits apparaissent dans gray(n+1). Si on est un peu plus courageux, on peut même prouver par récurrence que deux éléments consécutifs de la liste gray(n) ne différent que d'un bit (autrement dit, les listes ont les mêmes éléments aux même endroits, sauf pour un unique indice), et que c'est également le cas pour le premier et le dernier élément de la liste, ce qui prouve qu'on obtient une liste correspondant à la numérotation binaire appelée codage de Gray et évoquée dans le cours. Cette preuve n'est pas si dure : prenons deux listes éléments consécutifs de gray(n+1), si elles sont toutes les deux dans la première moitié de la liste, elle sont issues de deux éléments de gray(n) auxquels on a ajouté un 0 (éléments de l1 dans le programme mystere), donc elles ont le même bit d'écart que ces deux éléments dont elles sont issues. Même chose si elles sont dans la deuxième moitié (on a ajouté un 1 au lieu du 0 et inversé l'ordre, ça ne change rien). Si on prend maintenant les deux listes « au milieu » de gray(n+1) (la dernière de la première moitié et la première de la deuxième moitié), elles sont respectivement issues de la dernière liste de gray(n) augmentée d'un 0, et de la **dernière** liste de gray(n) augmentée d'un 1 (puisqu'on a retourné l'ordre dans la deuxième moitié), et seul leur dernier bit diffère. Même chose pour le premier et le dernier élément de gray(n+1), qui sont issus de la première liste de gray(n), augmentée respectivement d'un 0 ou d'un 1.