Corrigé Sujet 7 épreuve pratique NSI

Michel Billaud (michel.billaud@laposte.net)

1er février 2022

Table des matières

1 Licence

2 Le sujet

Se trouve sur la page https://eduscol.education.fr/2661/banque-des-epreuves-pratiques-de-specialite-nsi, dans https://eduscol.education.fr/document/33196/download

- conversions d'un entier positif en binaire
- tri à bulles

3 Conversion en binaire

Résumé du sujet écrire une fonction qui prend un entier positif `n` et retourne un couple formé d'une liste `b` d'entiers représentant `n` en binaire, et de `bit` le nombre de bits. L'exemple fourni

```
>>> conv_bin(9)
([1,0,0,1],4)
```

est particulièrement mal choisi, parce que la liste est un palindrome.

Les explications données ne précisent pas la question de l'ordre, mais on peut soupçonner qu'il va être utile d'employer la méthode `reverse` décrite, après avoir largement dévoilé le pot aux roses

> On remarquera qu'on récupère la représentation binaire d'un entier n en partant de la gauche en appliquant successivement les instructions :

```
b = n%2
n = n//2
```

répétées autant que nécessaire.

Critiques :

- L'indication présentée oublie le cas du zéro, codé sur *un* bit. Dans la première phrase du sujet, il aurait fallu insister sur **strictement** positif
- le nom `b` était déjà utilisé pour la liste de bits. Une fois de plus, on constate le manque de soin dans le choix des identificateurs.

3.1 Solution

Le nombre de bits correspond à la longueur de la liste, il n'y a pas de raison de ne pas utiliser `len` lors de la construction du couple résultat :

```
# pour n entier strictement positif
def conv_bin(n):
    bits = []
    while n != 0:
        bit = n % 2
        n = n // 2
        bits.append(bit)
    bits.reverse()
    return (bits, len(bits))


print ("Tests :")
assert conv_bin(1) ==  ([1], 1)
assert conv_bin(2) ==  ([1, 0], 2)
assert conv_bin(9) ==  ([1, 0, 0, 1], 4)
assert conv_bin(10) == ([1, 0, 1, 0], 4)
print("Ok!")
```

3.2 Cas du zeo

Le cas particulier de l'entier nul peut être traité en début de fonction

```
# pour n entier positif ou nul

def conv_bin(n):
    if n == 0:
        return ([0], 1)
    ....
```

4 Exercice 2, l'infâme tri à bulles

Le tri à bulles est un exemple pas spécialement facile à comprendre d'algorithme de tri intrinséquement inefficace (quadratique). On lira avec intérêt l'article d'Owan Astrakan https://users.cs.duke.edu/~ola/bubble/bubble.html à propos de son histoire, ses caractéristiques et l'échec malheureux jusqu'ici de son éradication dans l'enseignement (malgré les recommandations de nombreux auteurs, dont D. Knuth, par exemple).

Bref, on le retrouve ici.

Sujet : compléter la fonction

```
def tri_bulles(T):
    n = len(T)
    for i in range(...,...,-1):  #1
        for j in range(i):
            if T[j] > T[...]:    #2
                ... = T[j]       #3
                T[j] = T[...]    #4
                T[j+1] = temp
    return T
```

qui prend une liste d'entiers et renvoie la liste dans l'ordre croissant.

Remarque la fonction fait en réalité 2 choses, retourner une liste ordonnée, elle modifie son paramètre. En programmation, il est préférable de choisir entre les deux.

5 Résolution

Le principe du tri à bulles est de repérer un couple d'éléments voisins qui ne seraient pas dans l'ordre voulu, de les échanger pour arranger ça, et de recommencer jusqu'à ce que tout soit dans l'ordre.

En regardant les 3 lignes relatives à l'échange, on voit que

- les éléments concernés sont `T[j]` et `T[j+1]`,
- la variable d'échange est `temp`

Ce qui fournit assez d'information pour compléter d'un coup ce fragment

```
            if T[j] > T[j+1]:     #2
                temp = T[j]       #3
                T[j] = T[j+1]     #4
                T[j+1] = temp
```

En regardant la boucle `for j in range(i):`

- `j` va varier de 0 à `i-1`
- la première comparaison qu'elle fera sera entre `T[0]` et `T[1]`, la dernière entre `T[i-1]` et `T[i-1 + 1]`, c'est à dire `T[i]`
- pour que `T[i]` soit un accès légal au tableau `T`, il faut que `i` soit entre `0` et `n-1 = len(T)-1`

La boucle sur `i` est descendante (le troisième paramètre de range est -1) - elle doit partir de `n-1`, la plus grande valeur pour `i` - la boucle intérieure est exécutée `i-1` fois. C'est à dire 0 fois quand `i` vaut 1. - il est donc inutile de rentrer dans le corps de la boucle extérieure pour cette valeur. - on choisit donc les paramètres du `range` pour qu'il parcoure les valeurs de `n-1` à 2.

Le second paramètre de `range` indiquant la valeur à ne pas atteindre, on écrit :

```
  for i in range(n-1, 1, -1):  #1
```

5.1 Code complété

```
def tri_bulles(T):
    n = len(T)
    for i in range(n-1, 1, -1):  #1
        for j in range(i):
            if T[j] > T[j+1]:     #2
                temp = T[j]       #3
                T[j] = T[j+1]     #4
                T[j+1] = temp
    return T


print("Test tri")
assert tri_bulles([4, 1, 5, 2, 3]) ==  [1, 2, 3, 4, 5]
print("Ok")
```