Corrigé Sujet 4 épreuve pratique NSI

Michel Billaud (michel.billaud@laposte.net)

31 janvier 2022

Table des matières

1 Licence

- Les notes sont publiées dans https://www.mbillaud.fr/notes/
- Sources dans https://github.com/MichelBillaud/notes-diverses

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/33187/download

- liste des couples d'entiers consécutifs dans une liste
- remplissage d'une composante connexe

3 Exercice 1 : liste des couples d'entiers consécutifs successifs

Sujet résumé : Exemples fournis

```
>> recherche([1, 4, 3, 5])
[]
>>> recherche([1, 4, 5, 3])
[(4, 5)]
>>> recherche([7, 1, 2, 5, 3, 4])
[(1, 2), (3, 4)]
>>> recherche([5, 1, 2, 3, 8, -5, -4, 7])
[(1, 2), (2, 3), (-5, -4)]
```

Remarque : le nom `recherche` n'est pas particulièrement bien choisi, comme expliqué dans un corrigé précédent. On fera avec.

3.1 Résolution

Bien réfléchir aux termes employés :

- il y a le fait que deux nombres soient **voisins dans la liste**, c'est-à-dire que le couple est formé de `(liste[i], liste[i+1])` pour un certain entier `i`
- et aussi qu'ils se suivent, c'est-à-dire que le couple est de la forme `(n, n+1)`.

On peut se demander si un couple `(n+1, n)` est acceptable : c'est réfuté par le premier exemple, le couple `(4,3)` ne fait pas partie du résultat.

3.2 Solution 1

La fonction sera de la forme

```
def recherche(liste):
  ...
```

Une solution évidente est de parcourir tous les couples de nombres voisins, et de stocker dans une liste `resultat` ceux qui nous conviennent.

Pour parcourir l'ensemble de ces couples, on peut faire varier `i` de `0 à len(liste) - 2` :

```
def recherche(liste):
    resultat = []
    for i in range(0, len(liste) - 1):        # attention
        if liste[i] + 1 == liste[i+1] :
            resultat.append((liste[i], liste[i+1]))
    return resultat
```

ou de `1` à `len(liste) - 1`, avec un décalage d'indices

```
def recherche(liste):
    resultat = []
    for i in range(1, len(liste)):
```

```
        if liste[i-1] + 1 == liste[i] :
            resultat.append((liste[i-1], liste[i]))
```

3.3 Tests

Pour travailler sur un sujet d'examen, un environnement de test facilite la vie, en permettant de vérifier que les résultats obtenus sont bien ceux présentés dans le sujet

```
def test(liste):
    print ("recherche(" + str(liste) + ") = " \
           + str(recherche(liste)))

test([1, 4, 3, 5])
test([1, 4, 5, 3])
test([7, 1, 2, 5, 3, 4])
test([5, 1, 2, 3, 8, -5, -4, 7])
```

Affichage à l'exécution

```
recherche([1, 4, 3, 5]) = []
recherche([1, 4, 5, 3]) = [(4, 5)]
recherche([7, 1, 2, 5, 3, 4]) = [(1, 2), (3, 4)]
recherche([5, 1, 2, 3, 8, -5, -4, 7]) = [(1, 2), (2, 3), (-5, -4)]
```

3.4 Solution 2, avec listes en intension

La solution ci-dessous est plus proche de l'analyse du problème : produire une liste des couples, puis sélectionner.

```
def recherche(liste):
    couples = [ (liste[i-1], liste[i]) for i in range(1, len(liste))]
    return [ (a,b) for (a,b) in couples if a+1 == b ]
```

on peut aussi combiner les deux

```
def recherche(liste):
    return [ (liste[i-1], liste[i])
                for i in range(1, len(liste))
                if liste[i-1] + 1 == liste[i]
           ]
```

4 Exercice : remplissage d'une composante

```
def propager(M, i, j, val):
    if M[i][j]== ...:                                  # 1
        return

    M[i][j]=val

    # l'élément en haut fait partie de la composante
    if ((i-1) >= 0 and M[i-1][j] == ...):              # 2
```

```
        propager(M, i-1, j, val)

    # l'élément en bas fait partie de la composante
    if ((...) < len(M) and M[i+1][j] == 1):          # 3
        propager(M, ..., j, val)                     # 5

    # l'élément à gauche fait partie de la composante
    if ((...) >= 0 and M[i][j-1] == 1):              # 5
        propager(M, i, ..., val)                     # 6

    # l'élément à droite fait partie de la composante
    if ((...) < len(M) and M[i][j+1] == 1):          # 7
        propager(M, i, ..., val)                     # 8
```

Remarque ; le sujet oublie de préciser que **le tableau est carré**. On le déduit du fait qu'en 3, on compare la coordonnée verticale avec `len(M)` pour tester si on peut aller vers le bas, et en 7 la coordonnée horizontale pour aller à droite.

On pourrait avoir un tableau simplement rectangulaire, et dans ce cas la comparaison 6 se ferait avec `len(M[i])`.

Mauvaise rédaction : "Une composante d'une image est un sous-ensemble de l'image constitué uniquement de 1 et de 0" : l'emploi du masculin implique de c'est le sous-ensemble qui est composé de 0 et de 1. Mais le dessin montre qu'une composante est composée `soit de` zéros, `soit de` uns.

4.1 Résolution

Il s'agit d'un algorithme de "flooding". Il faut repeindre la composante qui commence à un certain endroit et est marquée par des 1.

Algorithme : Pour repeindre à partir d'un certain endroit

- si cet endroit ne fait pas partie de la composante (valeur 0) rien à faire
- sinon on repeint ce point, et on recommence à partir de ses voisins si ils n'ont pas déjà été repeints.

4.2 Solution

- la phrase "ne fait rien si vaut 0" incite à comparer avec 0 dans le trou 1 (`if M[i][j] == 0: # 1`)
- pour le trou 2, on peut deviner à partir des lignes 3, 5, 7 et du commentaire "fait partie de la composante", qu'il faut comparer à 1 `if ((i-1) >= 0 and M[i-1][j] == 1):`
- ensuite, il s'agit de voir en bas `(i-1,j)`, à gauche `(i,j-1)` et à droite `(i, j+1)` :
  - pour finir, il est clair qu'il s'agit de propager à partir d'un point après avoir vérifié qu'il fallait le faire, on reporte donc les coordonnées dans l'appel à `propager`

Code, en enlevant les parenthèses inutiles qui encadrent les conditions, et en remplaçant les commentaires trompeurs :

```python
def propager(M, i, j, val):
    if M[i][j] == 0:
        return

    M[i][j] = val

    # haut
    if (i-1) >= 0 and M[i-1][j] == 1 :
        propager(M, i-1, j, val)

    # bas
    if (i+1) < len(M) and M[i+1][j] == 1 :
        propager(M, i+1, j, val)

    # gauche
    if (j-1) >= 0 and M[i][j-1] == 1 :
        propager(M, i, j-1, val)

    # droite
    if (j+1) < len(M) and M[i][j+1] == 1 :
        propager(M, i, j+1, val)
```