Devoir ITC n°1

Recherche d'un élément unique

Le sujet suivant se compose de plusieurs parties indépendantes. Vous pouvez traiter les différentes parties dans l'ordre que vous souhaitez.

Lorsqu'une fonction est demandée par l'énoncé, même si vous ne proposez pas de code pour la définir, **vous pouvez supposer qu'elle est définie et vous en servir dans les questions suivantes**.

Dans tout le sujet, les fonctions sont données avec leur signature sous forme d'annotations : par exemple, `fonction(x:int, y:float) -> list` signifie que la fonction prend deux paramètres x (de type entier) et y (de type flottant) et renvoie une liste. **Il n'est pas demandé d'écrire les annotations dans la copie**.

Pour répondre aux questions de ce sujet, on dispose des fonctions suivantes de manipulation de listes :

- On peut obtenir la taille d'une liste `liste` avec `len(liste)`.
- Si `liste` est une liste de n éléments, on peut accéder au k^{e} élément (pour $0 \leq k < n$) avec `liste[k]`. On peut définir sa valeur avec `liste[k] = x`
- On peut ajouter un élément x dans une liste `liste` à l'aide de `liste.append(x)` et on considère qu'il s'agit d'une opération élémentaire.
- On peut concaténer deux listes en utilisant l'opération `liste1 + liste2`.
- On peut dupliquer n fois une liste en utilisant l'opération `liste * n`.

On dispose aussi du sclicing mais les autres fonctions ou méthodes sur les listes (`pop` , `sort` , `index` , `max` , etc...) sont interdites à moins de les réécrire explicitement.

Les points suivants seront pris en compte à la notation :

- vous attacherez la plus grande importance à la clarté, à la précision et à la concision de la rédaction ; 1 point pourra être attribué en bonus ou en malus selon la lisibilité de la copie ;
- de même, il est demandée de **tirer un trait entre deux questions** afin de les délimiter clairement ;
- la syntaxe sera évaluée de façon bienveillante, des erreurs mineures ne seront pas sanctionnées ; cependant, une syntaxe Python complètement fantaisiste fera perdre des points de présentation ;
- pour chaque fonction, un point sera attribué si celle-ci a une approche naturelle pour répondre au problème ;
- il est fortement recommandé de délimiter les niveaux d'indentation avec des traits verticaux ;
- si vous êtes amené à repérer ce qui vous semble être une erreur d'énoncé, vous le signalerez sur votre copie et devrez poursuivre votre composition en expliquant les raisons des initiatives que vous avez prises ;
- il est essentiel de respecter le temps imparti : les stylos seront levés à la fin de la composition ;
- le sujet ne sera **pas** rendu avec la copie.

Sur ce, bon courage ☺

On dispose d'une liste de $2n-1$ entiers naturels non nuls. Chaque élément de la liste est représenté exactement 2 fois à l'exception d'un entier présent une unique fois. Le but de ce problème est de trouver l'élément de la liste présent une unique fois.

I. Création de la liste

La seule importation autorisée est la fonction `random` du module `random`. L'instruction `random()` renvoie un réel tiré aléatoirement selon une distribution uniforme dans l'ensemble $]0;1[$.

1 – Rédiger une fonction `randentier(a:int, b:int)->int` qui renvoie un entier tiré aléatoirement dans l'ensemble $[\![a;b]\!]$.
On crée une liste de n éléments de la façon suivante : le premier élément est tiré dans l'ensemble $[\![1;p]\!]$ avec $p \in \mathbb{N}$; le deuxième dans l'ensemble $[\![1+p;2p]\!]$, et ainsi de suite jusqu'à l'ensemble $[\![1+(n-1)\,p;np]\!]$.

2 – Quelle liste obtient-on si $n=4$ et $p=1$?

3 – Proposer une fonction `alea(n:int, p:int)->list` renvoyant une liste de n éléments créés selon ce principe.

Le mélange de Knuth est un algorithme pour générer une permutation aléatoire d'un ensemble fini, c'est-à-dire pour mélanger un ensemble d'objets. Dans le cas d'une liste de n éléments (indicés de 0 à $n-1$), l'algorithme s'écrit en pseudo-code :

```
pour i allant de 0 à n-1 faire :
|    j = entier aléatoire entre 0 et i
|    échanger liste[i] et liste[j]
```

4 – Proposer une procédure `permute(liste:list)` qui permute la liste donnée en argument selon le principe de Knuth. L'application de cette procédure modifie la liste donnée en argument sans avoir besoin d'un `return`.

5 – Sans utiliser la méthode `pop`, proposer une fonction `suppr1list(liste:list)->list` qui renvoie une liste identique à la liste d'entrée mais avec un élément supprimé, choisi aléatoirement.

6 – À l'aide des fonctions précédentes, écrire une fonction `creation(n:int, p:int)->list` qui renvoie une liste mélangée de $2n-1$ éléments. Les entiers obtenus sont représentés exactement 2 fois à l'exception d'un entier présent une unique fois.

II. Recherche naïve

7 – À l'aide d'un parcours de liste à double boucle imbriquées, proposer une fonction python naïve d'en-tête `uniquenaif(liste:list)->int` permettant de trouver l'entier unique dans la liste des entiers donnée en argument.

8 – Exprimer la complexité de ce code en fonction de n et/ou p

III. Recherche par analyse des fréquences

9 – Donner une fonction `maximum(liste:list)->int` renvoyant le plus grand entier de la liste d'entiers naturels donnée en argument.

10 – Rédiger une fonction `indice(liste:list, x:int)->int` qui renvoie l'indice (de la première occurrence) de l'entier `x` dans la liste en argument. On pourra supposer que la présence de x est garanti dans la liste au moins une fois.

11 – Écrire une fonction `zeros(taille:int)->list` qui renvoie une liste de taille taille contenant uniquement des zéros.

12 – Proposer une fonction `freq(liste:list)->list` qui prend en argument d'entrée une liste d'entiers $[a_0, a_1, \dots a_{n-1}]$ et renvoie la liste Fq des fréquences de ces entiers. La liste Fq est de taille $M+1$ où M est le maximum de liste. L'idée de cette liste Fq est de considérer les valeurs a_k comme des indices (entre 0 et M), ainsi $Fq[a_k]$ représente le nombre d'apparitions de l'élément a_k dans la liste `liste` .

Par exemple pour la liste `L = [2, 2, 5, 3, 3, 1, 1]` la fonction `freq` renvoie la liste des fréquences d'apparitions `Fq = [0, 2, 2, 2, 0, 1]` . L'élément d'indice 5 indique 1 ce qui signifie que la valeur 5 n'apparaît que 1 fois dans la liste `L` .

13 – À l'aide des fonctions précédentes, en déduire une fonction `uniquef(liste:list)->int` qui renvoie l'élément unique de la liste donnée en argument.

14 – Exprimer la complexité de ce code en fonction de n et/ou p

IV. Recherche avec un tri

15 – À l'aide de la fonction `sorted(liste)` qui renvoie une liste triée de la liste donnée en argument, proposer une fonction `uniquetri(liste:list)->int` permettant de renvoyer l'entier unique dans la liste des entiers donnée en argument.

16 – Quelle est la complexité de cette fonction ? On indique que la complexité de la fonction `sorted` est en $\mathcal{O}(n \log n)$ pour une liste de n éléments.

V. Recherche avec XOR

On utilise habituellement l'écriture en base 10 des entiers : par exemple, 2985 représente le nombre $2\times10^3+9\times10^2+8\times10^1+5\times10^0$. Mais plus généralement, pour tout entier $b\geq 2$ on peut définir la représentation en base b d'un entier.

On définit la représentation en base b d'un entier, en convenant que l'écriture $(a_p a_{p-1}\dots a_0)_b$ représente le nombre :

$$a_p\times b^p + a_{p-1}\times b^{p-1} + \dots + a_1\times b^1 + a_0\times b^0$$

Pour s'assurer de l'unicité de l'écriture d'un entier dans une base donnée, il est nécessaire en outre d'imposer : $\forall k\in[\![0;p]\!]$, $a_k\in[\![0;b-1]\!]$ et $a_p\neq 0$.

Ainsi, en base 2 (représentation binaire) seuls les chiffres 0 et 1 sont utilisés, par exemple le nombre $(101)_2$ représente l'entier $1\times2^2+0\times2^1+1\times2^0$, c'est à dire 5. Lorsqu'un nombre est écrit en base 10, par convention on omet de le préciser : $(x)_{10}=x$.

On rappelle ci-dessous les tables logiques du « et » (&), du « ou » (|) et du « ou exclusif » ($\wedge$) :

$a\,\&\,b$	$a=0$	$a=1$
$b=0$	0	0
$b=1$	0	1

$a\|b$	$a=0$	$a=1$
$b=0$	0	1
$b=1$	1	1

$a\wedge b$	$a=0$	$a=1$
$b=0$	0	1
$b=1$	1	0

On considère par exemple :

- $a=92$ dont la décomposition en binaire est $(1011100)_2$
- $b=21$ d'écriture binaire $(10101)_2$

En opérant bit par bit, les différentes opérations donnent :

- $a\,\&\,b=(10100)_2=20$
- $a|b=(1011101)_2=93$
- $a\wedge b=(1001001)_2=73$

La fonction « ou exclusif » se nomme également XOR, est une opération commutative, et on peut l'obtenir en python avec `a^b` .

17 – On considère un entier naturel N, que renvoie :

- $0 \wedge N$?
- $N \wedge N$?
- $N \wedge N \wedge N$?

18 – En se servant des propriétés du XOR évoquées ci-dessus, donner une fonction itérative `uniquexor(liste:list)->int` permettant de trouver l'entier unique dans la liste des entiers donnée en argument. Sa complexité doit être optimale.

19 – En admettant que l'opération logique XOR s'effectue en temps constant, quelle est la complexité de la fonction `uniquexor` ? Conclusion.