DS d'informatique n°2 : Suite de Champernowme

Durée : **2h**. Calculatrices non autorisées.
Le soin et la clarté pourront faire varier la note de ±1 point. Soyez vigilants à respecter l'indentation, les notations de l'énoncé, et à mettre des commentaires pour les étapes complexes. **Vous pouvez réutiliser les fonctions des questions précédentes, même si vous n'avez pas su y répondre.**

Un exercice indépendant

On considère l'algorithme suivant :

```
Entrée : a,b
# Retourne pgcd(a,b), avec a,b deux entiers positifs et (a,b)≠(0,0)
Tant que a*b > 0 faire :
    Si a > b :
        a ← a-b
    Sinon :
        b ← b-a
Retourner le maximum entre a et b
```

1) On applique cet algorithme avec les arguments $a = 72$ et $b = 30$. Faire un tableau qui recense les valeurs de a et b à la fin de chaque itération de la boucle "Tant que". Quelle est la valeur retournée ?

2) Montrer la terminaison de cet algorithme.

Annexe et définitions utiles pour le problème

- On rappelle que `str` est la fonction de conversion en chaine de caractères. Par exemple `str(14)` renvoie `'14'`.
- Si `S` est une chaine, `S[i]` est l'élément d'indice `i` de la chaine `S`. On peut utiliser le slicing, avec les syntaxes comme `S[d:f]`, `S[:f]`, `S[d:]`, etc.
- Pour tester si deux chaines `S1` et `S2` sont identiques, on pourra utiliser le test `S1 == S2`.
- On dit que `y` est une *sous-chaine* de `x` si les caractères de `y` figurent de manière consécutive dans `x`. Par exemple `'format'` est une sous-chaine de `'informatique'`.
- (La suite de cette annexe concerne uniquement la partie C) Un *dictionnaire* est un conteneur avec des accolades qui contient des couples sous la forme "clé: valeur". Dans ce devoir on se restreint au clés qui sont des entiers et aux valeurs qui sont des booléens :

```
>>> D = { 1:  True, 2:  True, 4:  False, 7:  True }
```

- On peut facilement lire la valeur d'une clé, et affecter une valeur à une clé (déjà existante ou pas). Toutefois la syntaxe utilise des crochets comme pour les chaine de caractères :

```
>>> D[7]     # lecture
True
>>> D[3]     # lecture
KeyERROR : 3 is not a key
```

```
>>> D[2] = False     # modification
>>> D[3] = True     # ajout d'une clé non existante
>>> D
{ 1: True, 2: False, 3: True, 4: False, 7: True }
```

Problème : suite de Champernowme

Une suite univers (en base 10) est une suite u de chiffres (de 0 à 9) telle que toute séquence finie de chiffres apparaît comme sous-suite formée de termes consécutifs de cette suite. Par exemple on soupçonne que les décimales de π forment une suite univers (mais ceci n'a jamais été prouvé). Si c'est le cas alors toute séquence, quelle que soit sa longueur, se trouve dans les décimales de π. Prenons par exemple la séquence 419 :

$$3141592653589793238462643383279502884197169399375 \ldots$$

La suite de *Champernowme* offre un exemple de suite univers. On l'obtient par concaténation des chiffres composant les entiers naturels, pris dans l'ordre croissant :

$$01234567891011121314151617181920 21\ldots$$

On définit ainsi la suite de Champernowme notée C :

$$C = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 0, 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9, 2, 0, 2, 1, 2 \ldots).$$

On comprend aisément que C est une suite univers, par construction.
Pour $i \in \mathbb{N}$ on note $x_i \in [\![0, 9]\!]$ le i-ème terme de C. Par exemple $x_9 = 9$, $x_{10} = 1$, $x_{15} = 2\ldots$

Dans ce devoir, nous décidons de travailler avec des chaînes de caractères. Ainsi la séquence $(x_0, x_1, \ldots, x_{20})$ sera codée par la chaine `'012345678910111213141'`.

Partie A : suite de Champernowme

1) Écrire une fonction `champ(N:int) -> str` qui prend en argument un entier naturel N et qui renvoie la chaine de caractères obtenue en concaténant les entiers de 0 à N (inclus).

2) Écrire une fonction `champ2(N:int) -> str` qui prend en argument un entier naturel N et renvoie $(x_0, x_1, \cdots, x_N)$ sous forme de chaine de caractères.

Partie B : suite de Champernowme réduite, algorithme naïf

On peut remarquer que la séquence `'12'` (par exemple) apparaît dès le début de la séquence C, mais qu'elle a quand même été réécrite un peu plus loin, ce qui n'est pas nécessaire pour obtenir une suite univers :

$$1234567891011121314151617181920 2$$

La suite de *Champernowme réduite* notée C_r est définie de manière similaire à celle de Champernowme. Pour chaque entier $n \in \mathbb{N}^*$, la séquence de ces chiffres est concaténée, mais uniquement si cette séquence ne figure pas déjà dans les termes précédents :

$$C_r = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 0, 1, 1, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9, 2, 0, 2, 1, 2, 2, 2, 4, 2, 5,$$

$$2, 6, 2, 7, 2, 8, 2, 9, 3, 0, 3, 2, \ 3, 3, 3, 5, 3, 6, 3, 7, 3, 8, 3, 9, 4, 0, 4, 3, 4, 4, \ldots)$$

Ci-dessus, 23 n'est pas ajouté car la séquence `'23'` existe déjà dans les termes précédents, à savoir (x_2, x_3). Par contre la séquence `'24'` n'est pas préexistante, et donc elle est ajoutée.

On souhaite écrire une fonction `champred(N:int) -> str` qui prend en argument un entier naturel N et qui renvoient les termes de C_r en allant jusqu'à l'écriture (éventuelle) des chiffres de l'entier N.

```
>>> champred(12)
'01234567891011'

>>> champred(200)
'012345678910111314151617181920212224252627282930323335363738394043444647484950545557585990991001021031041051061071081091101121141151161171181191606566686970767779808788909910010210310410510610710810911011211411511611711811916065666869707677798087889099100102103104105106107108109110112114115116117118119120124125126127128129130132133134135136137138139140142143144145146147148149150152153154155156157158159160162163164165166167168169170172173174175176177178179180182183184185186187188189190193194195196197198199200'
```

3) Écrire une fonction `cherche(mot:str, texte:str) -> bool` qui renvoie `True` si la chaine `mot` est une sous-chaîne de la chaine `texte`, et `False` sinon.
Par exemple, `cherche('lire','tirelire')` renvoie `True`, mais `cherche('pire','tirelire')` renvoie `False`.

4) On considère dans ce problème qu'un test d'égalité entre deux chaînes constitue une seule opération élémentaire, quelles que soient les longueurs des chaînes.
Justifier que la complexité de la fonction `cherche` est d'ordre n, où n est la longueur de la chaîne `texte`.

5) Réécrire et compléter la fonction suivante pour qu'elle renvoie le résultat attendu :

```
def champred(N):
    C = ......
    for k in ...... :
        if ...... :
            C = C + str(k)
    return C
```

6) Montrer que le nombre d'opérations élémentaires lors de l'appel de `champred(N)` est au moins égal à N^2.

Partie C : suite de Champernowme réduite, deuxième algorithme

On cherche maintenant à écrire une fonction `champred2(N:int) -> str`, qui renvoie le même résultat que `champred`, mais de manière plus efficace. On propose de stocker, au cours de l'algorithme, les séquences déjà écrites dans une variable dédiée `X`, de manière à pouvoir tester **rapidement** si la séquence des chiffres d'un entier donné a déjà été écrite ou pas.

7) Expliquer pourquoi dans la fonction `champred2(N)`, si N est de longueur ℓ, alors on peut se contenter de stocker dans `X` les séquences déjà écrites de longueur inférieure ou égale à ℓ uniquement.

On envisage alors deux possibilités :

a) `X` est une liste, on stocke simplement les séquences déjà écrites comme éléments de `X`.

b) `X` est un dictionnaire, et `X[i]` est une valeur booléenne qui indique si le nombre `i` a déjà été écrit. On renvoie à l'annexe en début de sujet pour des détails sur les dictionnaires.

Par exemple pour $N = 100$, après avoir écrit les nombres jusqu'à 3, i.e. $C_r = (0, 1, 2, 3)$, l'état de ces variables serait

a) `X = ['0','1','01','2','12','012','3','23','123']`

b) `X = [ '0':True, '1':True, '2':True, '3':True, '4':False, ..., '9':False,`
`'10':False, '11':False, '12':True, '13':False, ... , '100':False ]`

8) Quel est le coût de la recherche d'une séquence `s` dans une liste `L` selon la longueur de `L` : constant, linéaire ou quadratique ? Quel est le coût pour obtenir la même information avec un dictionnaire `D` ?

On décide dans la suite de retenir la solution b) : la variable `X` est donc un dictionnaire `D`. Au début de l'algorithme, les valeurs dans `D` doivent toutes être `False`.

9) Écrire une fonction `initdico(N:int) -> dict` qui renvoie cette valeur initiale du dictionnaire. Par exemple :

```
>>> initdico(100)
{'0': False, '1': False, '2': False, '3': False, ... , '100': False}
```

10) Écrire une fonction `longueur(N:int) -> int`, qui renvoie le nombre de chiffres qui composent l'entier N. Dans la suite ce nombre de chiffres sera noté M. Par exemple `longueur(2000)` renvoie `4`.

11) Montrer qu'il existe une constante α telle que $M \leq \alpha \ln(N)$.

Dans notre algorithme, pour chaque entier k qu'on concatène (si on le concatène), la variable `D` doit être actualisée. Pour cela il faut déclarer comme déjà écrites toutes les séquences de longueur $\leq M$ qu'on fait apparaître après la concaténation de chaque chiffre composant l'entier k.
Exemple. Pour $N = 100$ (donc $M = 3$), lors de la concaténation de $k = 25$:

$$012345678910111314151617181920212224\underline{\mathbf{25}}$$

il faut déclarer écrites les séquences suivantes :

- `'2'`, `'42'`, `'242'` après concaténation du chiffre 2,
- `'5'`, `'25'`, `'425'` après concaténation du chiffre 5.

12) Compléter la fonction suivante, qui effectue le travail décrit ci-dessus :

```
1  def champred2(N):
2      M = longueur(N)
3      C = ''
4      D = ......
5      for k in ...... :
6          if D[str(k)] == ...... :
7              for chiffre in str(k):
8                  C = ...... # ajout de chiffre dans C
9                  for j in range(1,M+1):
10                     # déclarer la séquence des j derniers
11                     # caractères de C comme déjà vue
12                     D[ ...... ] = ......
13     return C
```

13) Montrer que le nombre d'opérations élémentaires effectués par la fonction `champred2(N)` est majoré par $\beta N \ln^2 N$, avec $\beta > 0$ une constante indépendante de N.