Interrogation d'ITC n°3

Semaine du 10/03/2025

Durée : 30 minutes

Nom :		Prénom :	

Consignes

- Les codes doivent être présentés en mentionnant explicitement l'indentation, au moyen par exemple de barres verticales sur la gauche.
- Pour qu'un code soit compréhensible, il convient de choisir des noms de variables les plus explicites possibles.
- La performance du code proposé à chaque question entrera dans l'évaluation, de-même que l'utilisation de boucles appropriées.
- Vous pouvez bien sûr utiliser une feuille de brouillon en parallèle.

/20

Exercice 1

On considère la fonction suivante :

```
1  def mystere(a:int,b:int)->int:
2      assert a > 0 and b > 0, "donner des entiers > 0"
3      r = 0
4      while a > 0:
5          if a%2 == 0:
6              a = a//2
7              b = b*2
8          else:
9              a -= 1
10             r += b
11     return r
```

1. /1 Que se passe-t-il lors de l'appel `mystere(-3, 4)` ?

2. /2 On désigne par a_i, b_i et r_i les contenus des variables a, b et r à l'issue de la i-ème itération. Donner les valeurs des suites $(a_i)_i$, $(b_i)_i$ et (r_i) lors de l'appel `mystere(13, 4)`. *On précisera le résultat sous forme d'un tableau.*

3. /3 Montrer la terminaison de l'algorithme.

4. /3 On note alors n le nombre d'itérations du `while`. Soit $i \in [\![0, n]\!]$. On pose $\mathscr{P}(i)$ « $a \times b = a_i \times b_i + r_i$ ». Montrer que $\mathscr{P}(i)$ est un invariant de boucle. Que renvoie `mystere(a, b)` ? Prouver alors la correction à l'aide de l'invariant.

Exercice 2

On considère la fonction :

```python
def recherche(L:list,m:int)->bool:
    Trouve = False
    n = len(L)
    for k in range(n):
        if L[k] == m:
            Trouve = True
    return Trouve
```

1. /2 Préciser (grossièrement) la docstring de cette fonction. *La réponse devra surtout mentionner les parties principales attendues dans une docstring.*

2. /2 Soit $k \in [\![0, n]\!]$. On pose $\mathscr{P}(k)$ « `m ∈ L[:k]` **ou** $\texttt{Trouve}_k = \texttt{False}$ ». Montrer que $\mathscr{P}(k)$ est un invariant de boucle.

5. /3 On désigne par $C(p)$ le nombre total d'opérations lors de l'appel `mystere(2^p,b))`. Exprimer $C(p)$ en fonction de p. *On comptera : les affectations, tests, sommes, produits et différences*

3. /2 Déterminer la complexité asymptotique (notation « grand o ») de cette fonction. *On commencera par calculer la complexité dans le pire et le meilleur des cas, n'hésitez pas à faire appel précisément aux numéros de lignes mentionnés à gauche du programme*

4. /2 Quel est le principal défaut (en terme de complexité) de cette fonction ? Proposez-en une amélioration, et analysez la complexité nouvelle associée.

SOLUTIONS DES EXERCICES

Solution 1

1. L'appel `mystere(-3, 4)` renvoie un erreur de type `AssertionError` ci-dessous :

```
>>> mystere(-3, 4)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "<input>", line 2, in mystere
AssertionError: donner des entiers > 0
```

2. Voici les contenus des variables au cours de la fonction :

Itération	a_i	b_i	r_i
0	13	4	0
1	12	4	4
2	6	8	4
3	3	16	4
4	2	16	20
5	1	32	20
6	0	32	52

La boucle s'arrête ensuite puisque $a_6 = 0$

On peut vérifier le contenu de `r` en exécutant :

```
>>> mystere(13, 4)
52
```

Le résultat semble être le produit $a \times b$, nous le prouverons plus tard à l'aide d'un invariant.

3. Supposons que la boucle ne termine pas, alors la suite $(a_i)_{i\in\mathbb{N}}$ vérifie : $\forall i \in \mathbb{N},\quad a_i > 0$. Montrons que $(a_i)_{i\in\mathbb{N}}$ est strictement décroissante. Soit $i \in \mathbb{N}$. Alors :

- si a_i est pair, $a_{i+1} = \dfrac{a_i}{2} < a_i$ puisque $a_i \neq 0$.
- Si a_i est impair. Alors $a_{i+1} = a_i - 1 < a_i$.

Dans tous les cas : $a_{i+1} < a_i$ et donc $(a_i)_{i\in\mathbb{N}}$ est strictement décroissante, et est à valeurs entières (récurrence évidente). Ainsi, $a_i \leqslant 0$ à partir d'un certain rang, ce qui est contradictoire.

Conclusion : la boucle termine.

4. Montrons que $\mathscr{P}(i)$ « $a \times b = a_i \times b_i + r_i$ » est un invariant de boucle.

Initialisation. On a $a_0 = a, b_0 = b$ et $r_0 = 0$, on a bien $ab = ab + 0$, l'invariant est vérifié.

Hérédité. Supposons l'invariant vrai à une itération $i \in [\![0, n-1]\!]$, il y a donc une itération $i+1$ de sorte que $a_i > 0$. Montrons que $ab = a_{i+1}b_{i+1} + r_{i+1}$.

- si a_i est pair, $a_{i+1} = \dfrac{a_i}{2}$, $b_{i+1} = 2b_i$ et $r_{i+1} = r_i$. On a bien :
$$a_{i+1}b_{i+1} + r_{i+1} = \frac{a_i}{2} \times 2b_i + r_i = a_ib_i + r_i = ab$$
d'après l'invariant.
- Si a_i est impair. Alors $a_{i+1} = a_i - 1$, $b_{i+1} = b_i$ et $r_{i+1} = r_i + b_i$. On a bien :
$$a_{i+1}b_{i+1} + r_{i+1} = (a_i - 1)b_i + (r_i + b_i) = a_ib_i + r_i = ab$$
d'après l'invariant.

Conclusion : $\mathscr{P}(i)$ est un invariant de boucle. Puisque $a_n = 0$, en prenant $i = n$ on a : $ab = 0 + r_n$ et donc la fonction renvoie bien $r_n = ab$.

5. Si $a = 2^p$, alors $a_k = 2^{p-k}$ pour tout $k \in [\![0, p]\!]$ (que l'on justifie sans peine en prouvant que c'est un invariant). Ainsi $a_p = 1$ et donc $a_{p+1} = 0$. En résumé, la boucle `while` possède $p+1$ itérations, et on entre p-fois dans le `if`, puis 1 fois dans le `else`. Au bilan :

$$\mathrm{C}(p) = \underbrace{3}_{\text{affectation init + assert}} + \underbrace{5}_{\substack{\text{2 tests+1 produit}\\ \text{+2 affectations}}} \times p + \underbrace{6}_{\substack{\text{2 tests+2 affectations}\\ \text{+2 opérations}}} \times 1 + \underbrace{1}_{\text{test final de sortie de boucle}} = \boxed{10 + 5p}.$$

Solution 2

1.

```
def recherche(L:list, m:int)->bool:
    """
    Renvoie True si m est présent dans L, False sinon

    Parameters
    ----------
    L : list of floats
    m : float

    Returns
    -------
    boolean
        True si m est dans L
        False sinon

    Examples
    --------
    >>> recherche([1, 2, 3], 2)
    True
```

```
    >>> recherche([1, 2, 3], 0)
    False
    """
    Trouve = False
    n = len(L)
    for k in range(n):
        if L[k] == m:
            Trouve = True
    return Trouve
```

2. Soit $k \in [\![0, n]\!]$. On pose $\mathscr{P}(k)$ « `m ∈ L[:k]` **ou** $\texttt{Trouve}_k = \texttt{False}$ ».

Initialisation. Pour $k = 0$, comme $\texttt{Trouve}_0 = \texttt{False}$, l'invariant est vérifié.

Hérédité. Supposons l'invariant vrai à une itération $k \in [\![0, n-1]\!]$, il y a donc une itération $k+1$.

- Si `m = L[k+1]`, alors $\texttt{Trouve}_{k+1} = \texttt{True}$, ce qui signifie que m est dans `L[:k+1]`, donc $\mathscr{P}(k+1)$ est vérifié.
- Sinon, `Trouve` garde sa valeur précédente, donc $\texttt{Trouve}_{k+1} = \texttt{Trouve}_k$. Puisque l'on suppose $\mathscr{P}(k)$, on a deux cas :
  - ◇ si $m \in$ `L[:k]`, alors $m \in$ `L[:k+1]` aussi et donc $\mathscr{P}(k+1)$ est vrai.
  - ◇ Si $\texttt{Trouve}_k = \texttt{False}$, alors $\texttt{Trouve}_{k+1} = \texttt{Trouve}_k = \texttt{False}$. Donc $\mathscr{P}(k+1)$ est vrai.

3.
- Dans le pire des cas (liste contenant que des `m`), on a 2 affectations (lignes 2 et 3), puis n itérations comportant 1 test (ligne 5) et 1 affectation (ligne 6). Au total : $C_{\text{pire}}(n) = 2 + 2n = 2(n+1)$.
- Dans le meilleur des cas (liste avec `m` absent), on a 2 affectations (lignes 2 et 3), puis n itérations comportant 1 test (ligne 5). Au total : $C_{\text{meilleur}}(n) = n + 2$.

On en déduit que : $\boxed{C(n) = O(n)}$.

4. Le problème de cette fonction c'est que la boucle ne s'arrête pas lorsque `m` a été trouvé. On peut proposer plutôt :

```
def recherche(L:list, m:int)->bool:
    """
    Renvoie True si m est présent dans L, False sinon

    Parameters
    ----------
    L : list of floats
    m : float

    Returns
    -------
    boolean
        True si m est dans L
        False sinon

    Examples
    --------
    >>> recherche([1, 2, 3], 2)
    True

    >>> recherche([1, 2, 3], 0)
    False
    """
    Trouve = False
    n = len(L)
    k = 0
    while k < n and not Trouve:
        if L[k] == m:
            Trouve = True
        k += 1
    return Trouve
```

```
>>> recherche([1, 2, 3], 2)
True
>>> recherche([1, 2, 3], 0)
False
```

Ici, dans le meilleur des cas on justifie sans peine qu'on est en $O(1)$ puisque la boucle ne possèdera qu'une seule itération.