Première NSI - Algorithmie

Travaux dirigés : parcours séquentiel

qkzk

2020/07/05

Se tester

1. Appartenance

1. Quel est le rôle du mot clé `in` en Python ? Citez deux usages courants.
2. Que retournent ces instructions ?

```
>>> 3 in [1, 2, 7]
>>> 4 in range(12)
>>> '4' in range(12)
>>> "ZA" in "AZALEE"
```

3. On rappelle cet algorithme, vu en cours :

```
fonction (tableau T, objet x) ---> booléen:
  Pour chaque élément e de T,
    Si e = x, alors on retourne Vrai
  Si la boucle se termine, on retourne Faux.
```

   a. Quel est le rôle de cet algorithme ?
   b. Écrire un programme Python qui transcrive cet algorithme.
   c. Parmi les exemples présentés à la question 2, quels sont ceux pour lesquels votre programme Python obtient la même réponse que l'instruction avec `in` ?

4. Adapter l'algorithme précédent pour qu'il retourne soit :

   - le premier indice de x si x figure dans le tableau,
   - -1 si x ne figure pas dans le tableau.

2. Recherche d'un élément maximal

On rappelle l'algorithme vu en cours :

```
fonction maximum(tableau T, nombre x) ---> nombre:
  On affecte à max la valeur de l'élément d'indice 0 du tableau.
  Pour chaque élément e du tableau:
    si e > max:
      max = e
  retourner max
```

1. Traduire cet algorithme en Python.
2. Quel est la complexité de cet algorithme ?
3. Démontrer que l'algorithme se termine.
4. Déterminer un invariant pour la boucle de l'algorithme.

S'entraîner

3. Retourner un tableau

1. Proposer un algorithme qui prend un tableau et le retourne. Voici la signature attendue :

   ```
   retourner(tableau T) ---> tableau:
   ```

   Ainsi qu'un exemple :

   ```
   >>> retourner([1, 2, 3])
   [3, 2, 1]
   ```

2. Traduire cet algorithme en Python.
   On pourra (mais ce n'est pas la seule approche) utiliser l'opérateur `+` pour les objets `list` qui *concatène* deux listes :

   ```
   >>> [1, 2] + [3, 4]
   [1, 2, 3, 4]
   ```

4. Décompte d'éléments

1. Proposer un algorithme qui compte le nombre d'apparitions d'un élément dans un tableau.

   ```
   decompte(tableau T, element x) ---> int
   ```

   Exemple :

   ```
   >>> decompte([1, 1, 2, 1, 1, 2, 1], 1)
   5
   >>> decompte(['a', 'b', 'ab', 'a'], 'a')
   2
   ```

2. Traduire cet algorithme en Python.
3. Adapter votre fonction en une fonction `frequence` qui calcule la fréquence d'un élément dans un tableau. (fréquence = effectif de la valeur divisé par effectif total)

5. Fonction mystere

Voici une fonction mystère :

```
def mystere(mot: str) -> bool:
    i = 0
    j = len(mot) - 1
    while i <= j:
        if mot[i] != mot[j]:
            return False
        i = i + 1
        j = j - 1
    return True
```

1. Faire tourner cette fonction sur les mots `radar` et `radrar` (qui n'existe pas, je sais).
2. Quel est le rôle de cet algorithme ?
3. Démontrer que l'algorithme se termine.
4. Adapter le programme afin qu'il ne compte plus qu'une seule instruction `return`.
5. Proposer un algorithme alternatif réalisant la même chose utilisant une fonction introduite dans un exercice précédent.

6. Correction de l'algorithme `factorielle`

On rappelle la définition de la factorielle :

$0! = 1$ et $n! = 1 \times 2 \times 3 \times \cdots \times n$

Ainsi $1! = 1, 2! = 2, 3! = 6, 4! = 24$ etc.

On considère l'algorithme suivant :

```
factorielle(entier n) ---> entier:
    i = 0
    f = 1
    Tant que i < n:
        i = i + 1
        f = f * i
    retourner f
```

1. Vérifier les exemples cités plus haut pour cette fonction.
2. Justifier que la boucle n'est pas infinie.
3. Montrer, en utilisant la pré-condition $n \geq 0$, que la propriété :

$$f = i! \text{ et } i \geq n$$

   est un invariant de la boucle.

4. Formuler une post-condition qui établit la correction de cette algorithme.

7. Puissance de 2

1. Écrire un algorithme utilisant une boucle *tant que* permettant de déterminer si un entier n strictement positif est une puissance de 2.
2. Montrer que la boucle *tant que* se termine.
3. Montrer que l'algorithme produit le résultat attendu.

8. Le plus vieux

On dispose d'une liste de personnes et de leurs ages sous cette forme :

```
personnes = [('Joe', 16), ('Zoé', 17), ('Martin', 15)]
```

1. Proposer un algorithme qui détermine le plus âgé des individus.

   ```
   >>> plus_ages(personnes)
   'Zoé'
   ```

2. Déterminer un invariant pour la boucle de l'algorithme.
3. Traduire votre algorithme en Python.