TD3 : Lecture de fonctions récursives

Compétences

- Évaluer la terminaison d'une fonction récursive.
- Lire et prévoir le résultat d'une fonction récursive
- Décrire précisément la pile d'exécution d'une fonction récursive.

Exercices

* **Ex. 1 —** On considère les fonctions `foo1` et `foo2` suivantes :

```python
def foo1(n):
    if n == 0:
        print(0)
    else:
        print(n)
        foo1(n - 1)
```

```python
def foo2(n):
    if n == 0:
        print(0)
    else:
        foo2(n - 1)
        print(n)
```

Décrire précisément la pile d'exécution des fonctions `foo1` et `foo2` pour $n = 3$

* **Ex. 2 —** On considère les fonctions `foo3` et `foo4` suivantes :

```python
def foo3(n):
    if n == 0:
        print(0)
    else:
        print(n)
        foo4(n - 1)
```

```python
def foo4(n):
    if n == 0:
        print(0)
    else:
        foo3(n - 1)
        print(n)
```

Décrire précisément la pile d'exécution des fonctions `foo3` et `foo4` pour $n = 3$

* **Ex. 3 —** Un problème classique est d'écrire une fraction comme somme de fractions égyptiennes avec des dénominateurs tous différents, que l'on nomme développement en fractions égyptiennes ou plus simplement développement égyptien. Tous les nombres rationnels positifs peuvent être écrits sous cette forme et ce, d'une infinité de façons différentes. Par exemple

$$\frac{2}{5} = \frac{1}{5} + \frac{1}{6} + \frac{1}{30} = \frac{1}{5} + \frac{1}{8} + \frac{1}{20} + \frac{1}{40}$$

Un étudiant programme la fonction `fractionEgyptienne` suivante :

```python
def fractionEgyptienneV1 ( a, b) :
    if a == 1 :
        print(f"1/{b}", end="    ")
    fractionEgyptienneV1 ( 1, b)
    fractionEgyptienneV1 ( a-1, b+1)
    fractionEgyptienneV1 ( a-1, b*(b+1))
```

1. L'appel `fractionEgyptienne(2, 5)` ne semble pas fonctionner et l'étudiant vous sollicite pour lui expliquer ce qui ne va pas. Qu'en pensez-vous ?
2. Proposer à l'étudiant une correction possible.
3. Décrire précisément la pile d'exécution pour l'appel précédent.

* **Ex. 4 —** On considère la fonction `powRec` suivante :

```python
def powRec (a, n):
    if ...
        ...
    return a * powRec(a, n-1)
```

1. Completer la condition d'arrêt de cette fonction

2. Décrire précisément la pile d'exécution pour l'appel `powRec(2, 4)`

** **Ex. 5 —** Écrire une fonction `dichotomie_recursive(x, t, g, d)` qui prend en paramètres la valeur cherchée, un tableau, les indices de départ et de fin dans le tableau et qui retourne True si la valeur est présente et False sinon.

** **Ex. 6 —** On considère la fonction `mystere` suivante :

```
1  def mystere(tab, g, d):
2      if g == d:
3          return tab[g]
4
5      else:
6          milieu = (g + d) // 2 - 1
7          if milieu % 2 == 1:
8              milieu += 1
9          if tab[milieu] != tab[milieu + 1] :
10             return mystere(tab, g, milieu)
11         else:
12             return mystere(tab, milieu + 2, d)
```

Décrire précisément la pile d'exécution de l'appel suivant : `mystere(array('l', [1,1,6,6,7,9,9,3,3,0,0]), 0, 10)`. Donner un nom explicite à cette fonction

** **Ex. 7 —** On considère la fonction `checkNumber` suivante :

```
1  def checkNumbers(L, n, i, j) :
2      if i >= n - 1:
3          return True
4      else :
5          if j == n:
6              return checkNumbers(L, n, i+1, i+2)
7          else :
8              if L[i] == L[j]:
9                  return False
10             else:
11                 return checkNumbers(L, n, i, j+1)
```

1. Décrire précisément la pile d'exécution pour l'appel de fonction `checkNumbers(array('l', [2,3,6,2]), 4, 0, 1)`
2. Décrire précisément la pile d'exécution pour l'appel de fonction `checkNumbers(array('l', [2,3,6,1]), 4, 0, 1)`
3. Que fait cette fonction ?
4. Quelle est sa complexité ?
5. Donner la version itérative de `checkNumbers`
6. Quelle est sa complexité