Fonctions Récursives

Compétences

- Écrire un algorithme avec une méthode récursive
- Visualiser la notion d'environnement et de pile d'exécution.

Une série de petits exercices

Pour tester vos fonctions et mieux comprendre les appels récursifs il est conseillé d'utiliser Python tutor

1. Rendre récursive la fonction somme suivante :

```
1 def sumInt(n):
2     somme = 0
3     for i in range(1, n) :
4         somme += i
5     return somme
```

2. Rendre récursive la fonction somme suivante :

```
1 def sumArray(t, n):
2     somme = t[0]
3     for i in range(1, n) :
4         somme += t[i]
5     return somme
```

3. Écrire une fonction récursive `oneComplement(t, n)` qui prend en argument un nombre binaire sous la forme d'un tableau `t` d'entiers courts, son nombre d'éléments `n` et qui renvoie le nombre complémentaire (0 devient 1 et 1 devient 0) dans le même tableau.

```
Exemple:
t = array('i', [1,1,1,0])
oneComplement(t, len(t))
Renvoie:
array('i',[0,0,0,1])
```

4. Écrire une fonction récursive `reverse(t, n, i=0)` qui prend en argument un tableau de caractères `t`, le nombre d'éléments `n`, un indice `i` qui vaut 0 par défaut et qui renvoie le tableau avec les caractères ordonnés en sens inverse.

```
Exemple:
t = array('u', ["p", "e", "t", "i", "t"])
reverse(t, len(t), 0)
Renvoie:
array('u', 'titep')
```

5. Écrire une fonction récursive qui prend en arguments un tableau, sa taille, un indice et qui renvoie la somme des entiers positifs entre l'indice inclu et la fin du tableau. Le tableau `[2, 3, -1, 4]` avec l'indice 1 renvoie 7
6. Écrire une fonction récursive qui prend en argument un tableau, sa taille et qui renvoie `True` si le tableau est trié et `False` sinon.
7. Un nombre N est pair si $(N-1)$ est impair, et un nombre N est impair si $(N-1)$ est pair. Ecrire deux fonctions récursives croisées `pair(N)` et `impair(N)` permettant de savoir si un nombre N est pair ou impair.
8. Écrire une fonction récursive qui prend en argument un tableau, sa taille et qui renvoie le maximum de ce tableau. Pensez à diviser pour régner.
9. Il s'agit de distinguer les nombres premiers. Le test le plus simple est le suivant : pour tester un nombre entier N, on vérifie s'il est divisible par l'un des entiers compris entre 1 et N (bornes non comprises). Si la réponse est négative, alors N est premier. Écrire une fonction récursive `primeNumber(N, i=0)` qui prend en argument un nombre entier, un indice qui vaut 0 par défaut et qui renvoie `True` si celui-ci est premier et `False` sinon.

Retour sur la fraction égyptienne

À l'aide de l'énoncé du TD3 proposer une solution en implémentant la fonction suivante : `fractionEgyptienneV2 ( a, b, t, i=0)`. Cette fonction prend en paramètres le numérateur et le dénominateur de la fraction, un tableau de type `array`, un indice de position qui vaut 0 par défaut et renvoie un tableau avec les différents dénominateurs de la décomposition égyptienne.

```
Exemple:
t = array('l', [0]*10)
fractionEgyptienneV2 ( 3, 5, t, 0)
Renvoie:
[5, 6, 7, 42, 30, 31, 930, 0, 0, 0]
```

La suite de Fibonnaci

Considérons la suite numérique ainsi définie :

- $F_0 = 0$
- $F_1 = 1$
- $\forall n \in \mathbb{N}, F_{n+2} = F_{n+1} + F_n$

On a donc :
$F_2 = 0+1 = 1$
$F_3 = F_2 + F_1 = 1+1 = 2,$
$F_4 = F_3 + F_2 = 2+1 = 3, \ldots$

1. Écrire une fonction `fiboRecursive` qui calcule à l'aide d'une méthode recursive le n-ième terme de la suite de Fibonnaci.
2. Observer en détail la pile d'exécution lors du calcul de `fiboRecursive(4)` avec PythonTutor
3. Écrire une fonction `fiboIterative` qui calcule à l'aide d'une méthode itérative le n-ième terme de la suite de Fibonnaci.
4. Comparer les temps moyens d'exécution pour le terme de rang $n = 20$ à l'aide des instructions suivantes.

```
%%timeit
fiboRecursive(20)
%%timeit
fiboIterative(20)
```

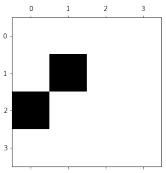
5. Quelle est la fonction qui a la meilleure complexité temporelle ?
6. Même question avec la complexité spatiale.

Labyrinthe

Un labyrinthe peut-être représenté à l'aide d'une matrice carrée de la forme : $\mathscr{M}_{n,n}(\{0,1\})$ avec $n \in \mathbb{N}^*$. On décide que la valeur 1 indique un couloir et que la valeur 0 indique un mur. À partir d'une cellule donnée, nous sommes autorisés à passer aux cellules $(i+1, j)$ et $(i, j+1)$ uniquement

Exemple d'un labyrinthe de dimension (4×4)

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$



1. Tracer à la main tous les chemins possibles pour aller de la case $(0,0)$ à la case $(3,3)$
2. Écrire une fonction récursive permettant de calculer tous les chemins possibles en partant de la case $(0,0)$ pour atteindre la case $(n-1, n-1)$