INFORMATIQUE MP DEVOIR 1

04 NOVEMBRE 2016

Le soin et la clarté, ainsi que le respect de la syntaxe en Python sont partie intégrante de la notation finale.

EXERCICE 1 - TRIS ET ÉTUDES

Dans le but dé réaliser des études statistiques, on souhaite se doter d'une fonction de tri. On se donne la fonction **tri** suivante :

```
def tri(L):
    n = len(L)
    for i in range(1,n):
        j = i
        x = L[i]
        while 0 < j and x < L[j-1] :
            L[j] = L[j-1]
            j = j-1
        L[j]=x
```

1. Détailler l'appel `tri(L)` lorsque `L` est la liste `[5,2,3,1,4]`.
2. Soit L une liste non vide d'entiers ou de flottants. Montrer que ≪ la liste `L[0:i+1]` (en convention Python) est triée par ordre croissant à l'issue de l'itération `i` ≫ est un invariant de boucle. En déduire que `tri(L)` tri la liste `L`.
3. Évaluer la complexité dans le meilleur et le dans le pire des cas de l'appel `tri(L)` en fonction du nombre d'éléments `n` de `L`.
4. Citer un algorithme de tri plus efficace dans le pire des cas. Quelle en est sa complexité dans le meilleur et le pire des cas ?

On souhaite, partant d'une liste constituée de couples (chaîne, entier), trier la liste par ordre croissant de l'entier associé suivant le fonctionnement suivant :

```
>>> L = [['Bresil', 76], ['Kenya', 26017], ['Ouganda', 8431]]
>>> tri_chaine(L)
>>> L
[['Bresil', 76], ['Ouganda', 8431], ['Kenya', 26017]]
```

5. Écrire une fonction python `tri_chaine` réalisant cette opération.

On considère toujours une liste constituée de couples (chaîne, entier). On souhaite déterminer quelques indicateurs sans avoir à trier la liste.

6. Écrire une fonction python `max_chaine` qui renvoie la chaîne de caractère dont l'entier correspondant est le maximum.
7. Écrire une fonction python `moy_chaine` qui renvoie la moyenne des entiers des couples de la liste.
8. Écrire une fonction python `pays_moy_chaine` qui renvoie la liste des chaînes de caractères dont l'entier correspondant a un écart à la valeur moyenne des entiers de plus ou moins 10%.

EXERCICE 2 - PILES

On souhaite réaliser une structure de piles de taille borné en Python.
Pour simuler une pile capacité `N` on considère un tableau `p = [n,a1,a2,a3,...,an,....]` de taille $N+1$ tel que :
- N soit la taille maximale de la pile
- n, la valeur de $p[0]$, soit le nombre d'éléments dans la pile.
- $a1, a2, a3, ..., an$ sont les n éléments de la pile
- l'espace restant est considéré comme vide dans la pile, quelque soit les valeurs du tableau.

Par exemple `p=[3,6,1,5,2,4,0]` représente la pile :

On considère que les fonctions suivantes sont déjà saisies dans Python :

```
def creer_pile(N):
   p = (N + 1) * [None]
   p[0] = 0
   return(p)


def taile(p):
  return(p[0])
```

1. Ecrire en Python les fonctions suivantes :
   ▷ `est_vide(p)` : indique si la pile p est vide.
   ▷ `depiler(p)` : dépile et renvoie le sommet de la pile p.
   ▷ `empiler(p,v)` : empile la valeur v sur la pile p.

On considère que toutes ces fonctions sont maintenant écrites en Python.

2. Écrire une fonction `intervert(p)` qui échange les deux éléments en haut de la pile `p`.
3. Écrire une fonction `renverse(p)` qui renvoie une pile constituée des mêmes éléments dans l'ordre inverse.
4. On considère deux piles `A` et `B`. On souhaite mélanger les deux piles de la manière suivante :

   ```
   tant que B n'est pas vide:
       si A n'est pas vide
         on renverse A
         on dépile le sommet de A que l'on stocke
         on empile le sommet de B sur A
         on rempile le sommet stocké sur A
   ```

   Écrire une suite d'instructions réalisant ce processus.

EXERCICE 3 - AUTOUR DE LA RÉCURSIVITÉ

Les parties sont indépendantes.

Partie A

On s'intéresse d'abord au calcul de la puissance k-ième d'un entier naturel n.

1. On considère l'algorithme naïf :

   ```
   def puissance(x,n):
       if n == 0:
          return 1
       else :
          return x * puissance(x,n-1)
   ```

   Quelle est la complexité de cette algorithme ?

2. On utlise maintenant l'algorithme :

   ```
   def puissance_rapide(x,n):
       if n == 0:
          return 1
       else :
          r = puissance_rapide(x,n // 2)
          if n % 2 ==0:
              return r * r
           else :
              return x * r * r
   ```

(a) Décrire l'exécution de l'appel `puissance_rapide(2,7)`.

(b) Décrire l'exécution de l'appel `puissance_rapide(2,8)`.

(c) Déterminer la complexité de cet algorithme.

Partie B

On considère l'algorithme d'Euclide pour calculer le pgcd de deux entiers :

```
def euclide(a,b):
    """ Données: a et b deux entiers naturels
    résultat : le pgcd de a et b, calculé par l'algorithme d'Euclide """
    u = a
    v = b
    while v != 0:
        r = u % v
        u = v
        v = r
   return u
```

Proposer une version récursive `euclide_rec` de cette fonction.

Partie C

On considère le couplage de Cantor, qui établit une bijection entre $\mathbb{N} \times \mathbb{N}$ et $\mathbb{N}$.
On a par exemple $f(1,2) = 8$ et $f(3,1) = 11$.
Écrire une fonction récursive qui permet de calculer $f(x,y)$ pour tout couple (x,y) d'entier naturels