INFO-H-100 - Programmation
TP 17 — Révisions

Ex. 1. Complexité (basé sur des exercices de la Faculté des Sciences)

Donnez la complexité au pire cas, en fonction de n et en utilisant la notation $\mathcal{O}$, des fonctions suivantes.

Pour effectuer vos calculs, vous pouvez considérer, même si ce n'est pas tout a fait exact, que les opérations atomiques (somme, différence, multiplication, division, assignation, `return`, ...) ont une complexité au pire cas de $\mathcal{O}(1)$.

Considérez également les complexités au pire cas suivantes pour les opérations sur la liste `l` de taille n :
- `len(l)` : $\mathcal{O}(1)$
- `i = l[j]` : $\mathcal{O}(1)$
- `l.append(elem)` : $\mathcal{O}(1)$
- `elem in l` : $\mathcal{O}(n)$
- `l.remove(elem)` : $\mathcal{O}(n)$

```python
def a(t):
    n = len(t)
    i = 0
    while i < n :
        t[i] = 0
        i += 3
```

```python
def b(t):
    n = len(t)
    for j in range(4):
        for i in range(n):
            t[i] *= t[i]
```

```python
def c(li):
    n = len(li)
    i = 0
    j = 0
    while i < n:
        while j < i:
            li[j] *= i - j
            j += 1
        i += 3
```

```python
def d(li):
    n = len(li)
    i = n - 1
    while i >= 0:
        li[i] = 0
        for j in range(i+1):
            li[i] += j
        i -= 1
```

```python
def e(li):
    n = len(li)
    for i in range(n):
        li[i] = 0
        j = n
        while j > 1:
            li[i] += i*j
            j /= 2
```

```python
def f(c, a, b, n):
    for i in range(n):
        for j in range(n):
            c[i][j] = 0
            for k in range(n):
                c[i][j] += a[i][k] * b[k][j]
```

où (`a`, `b` et `c` sont des matrices de taille `n*n`)

Ex. 2. Juin 2011 - Dictionnaire (Faculté des Sciences)

Nous vous demandons d'écrire une fonction `histogramme()` qui prendra deux paramètres : une liste `L` de valeurs réelles (éventuellement négatives) ainsi qu'une liste `I` d'au moins deux valeurs réelles distinctes et triées dans l'ordre croissant. La liste `I` donne les bornes d'intervalles successifs sur la droite réelle. Par exemple, si cette liste est `[-5,2,9.5]`, elle symbolise deux intervalles : les valeurs entre -5 (inclus) et 2 (non inclus) ainsi qu'entre 2 (inclus) et 9.5 (non inclus).

Votre fonction `histogramme()` doit renvoyer un dictionnaire où chaque clef est un string de la forme `"[a,b)"` si l'on suppose que a est la borne inférieure (incluse) de l'intervalle et que b est la borne supérieure (non incluse) ; la valeur associée à chaque clef sera un entier qui dira combien de valeurs de la liste L sont incluses dans l'intervalle considéré.

Dans un second temps, nous vous demandons d'écrire une fonction `intervalles()` qui prendra trois paramètres : deux réels `inf` et `sup` (il est supposé que `sup` > `inf`) ainsi qu'un entier strictement positif `n`. La fonction devra renvoyer une liste triée tel qu'utilisée par la fonction `histogramme()` et qui correspond à `n` intervalles de même taille telles que la borne inférieure du premier intervalle est `inf` et que la borne supérieure du dernier intervalle est `sup`.

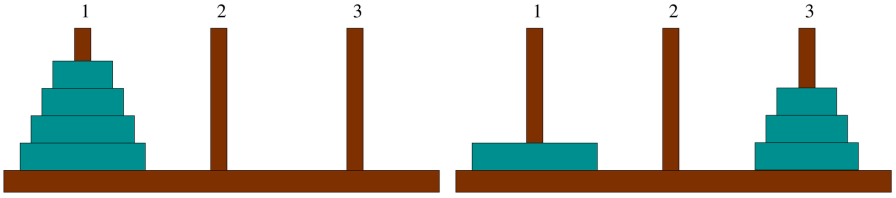
Un exemple possible d'exécution (en mode interactif) serait le suivant :

```
>>> L = [-1, 0, 3, -10, 0.5, 9, 8, 12]
>>> I = intervalles(-7, 9.5, 4)
>>> histogramme(L, I)
{'[-7,-2.875)': 0, '[-2.875,1.25)': 3, '[1.25,5.375)': 1, '[5.375,9.5)': 2}
```

Ex. 3. Aout 2011 - Récursivité (Faculté des Sciences)

Le problème des tours de Hanoi est un jeu qui consiste à déplacer des disques de diamètres différents d'une tour de départ à une tour d'arrivée en passant par une tour intermédiaire, et ceci en un minimum de coups, tout en respectant les règles suivantes :
– on ne peut déplacer plus d'un disque à la fois ;
– on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

Ce problème se prête très bien à une résolution récursive. Soient N le nombre des disques, et 1,2,3 les trois emplacements des tours. Si on veut déplacer la tour de N disques depuis l'emplacement 1 (Figure 1(a)) vers l'emplacement 2, il faut déplacer la tour composée des $N-1$ premiers disques de 1 vers 3 (Figure 1(b)), puis le disque N de 1 vers 2 et finalement la tour des $N-1$ disques de 3 vers 2.



(a) Situation initiale (b) Après déplacement de N-1 disques

On vous demande la fonction récursive `hanoi(N,orig,dest,inter)` qui prend en paramètres le nombre N de disques de la tour, l'emplacement du départ et de destination, et affiche à l'écran la suite des coups à faire pour resoudre le jeu.

Voici un exemple d'affichage à l'écran pour déplacer 4 disques de la tour 1 vers la tour 2 (le premier coup veut dire bouger un disque de l'emplacement 1 vers l'emplacement 3, etc.) :

```
>>> hanoi(4, 1, 2, 3)
1 --> 3   1 --> 2   3 --> 2   1 --> 3   2 --> 1
2 --> 3   1 --> 3   1 --> 2   3 --> 2   3 --> 1
2 --> 1   3 --> 2   1 --> 3   1 --> 2   3 --> 2
```

Ex. 4. Aout 2011 - Récursivité (Faculté des Sciences)

Soient x et y deux entiers positifs ; on voudrait calculer le quotient et le reste de leur division entière par récursivité.

On vous demande deux fonctions, `quotient(x,y)` et `reste(x,y)`, qui prennent deux valeurs x et y (vous pouvez considérer que ce sont des entiers positifs) et calculent le quotient et le reste de leur division entière de manière récursive. Bien évidemment, vous ne pouvez pas utiliser le modulo et la division entière, mais uniquement des sommes et des soustractions.

Pour calculer le quotient vous pouvez vous baser sur cette simple formule :

$$\frac{x}{y} = \frac{x-y+y}{y} = 1 + \frac{x-y}{y}.$$

Pour calculer le reste vous pouvez utiliser le résultat suivant :

$$\text{reste}(x,y) = \begin{cases} x & \text{si } x < y \\ 0 & \text{si } x = y \\ \text{reste}(x-y,y) & \text{si } x > y \end{cases}$$

INFO-H-100 - Programmation
TP 17 — Révisions
Corrections

Solution de l'exercice 1:

- La fonction `a(t)` est en $\mathcal{O}(n)$. Même si le nombre d'opérations s'approche de $n/3$, les règles de simplifications impliquent une complexité au pire cas en $\mathcal{O}(n)$.
- La fonction `b(t)` est également en $\mathcal{O}(n)$. Même si le nombre d'opérations s'approche de $4 * n$, les règles de simplifications impliquent une complexité au pire cas en $\mathcal{O}(n)$.
- La fonction `c(li)` est en $\mathcal{O}(n)$. La première boucle tourne n fois et la seconde boucle, imbriquée dans la première, tourne au maximum 3 fois.
- La fonction `d(li)` est en $\mathcal{O}(n^2)$. La première boucle tourne n fois et la seconde, imbriquée dans la première, tourne au maximum n fois.
- La fonction `e(li)` est en $\mathcal{O}(n \log n)$. La première boucle tourne n fois et la seconde tourne $\log n$ fois.
- La fonction `f(c, a, b, n)` est en $\mathcal{O}(n^3)$.

Solution de l'exercice 2:

```
def intervalles(begin, end, n):
  step = (end-begin)/float(n)
  return map(lambda x : begin+x*step ,range(n+1))

def histogramme(L, I):
    d = {}
    for i in range(len(I) - 1):
        inter = '[' + str(I[i]) + ',' + str(I[i+1]) + ')'
        d[inter] = 0
        for x in L:
            if I[i] <= x < I[i+1]:
                d[inter] += 1
    return d
```

Solution de l'exercice 3:

```
def hanoi(n, orig, dest, interm):
    if n > 0:
        hanoi( n - 1, orig, interm, dest )
        print orig, "-->", dest
        hanoi( n - 1, interm, dest, orig )
```

autre version :

```
def hanoi(n, orig, dest, interm):  #le dernier parametre ne sert a rien et devrait etre enleve
    if n > 0:
        inter = 6 - orig - dest
        hanoi(n-1, orig, inter)
        print orig, "-->", dest
        hanoi(n-1, inter, dest)
```

Solution de l'exercice 4:

```
def quotient(x,y):
    if x >= y :
        return 1 + quotient(x-y,y)
    else:
        return 0

def reste(x,y):
    if x < y :
        return x
    elif x == y :
        return 0
```

```
    else:
        return reste(x-y,y)
```