TD 1

Exercices d'application

TD d'informatique du Lycée Louis Legrand – Jean-Pierre Becirspahic
http://info-llg.fr/

Savoirs et compétences :

❒ *Alg – C15 : Récursivité : avantages et inconvénients.*

Exercice 1

Soit l'algorithme suivant :

```python
def mult(n, p):
    if p == 0:
        return 0
    else :
        return n+mult(n,p-1)
```

Question 1 *Énoncer un variant de boucle et montrer la terminaison de l'algorithme.*

Question 2 *Énoncer un invariant de boucle et montrer la correction de l'algorithme.*

Correction
- Soit $\mathscr{P}$ la propriété d'invariance : à l'itération p, on a $mult(n,p) = n \cdot p$.
- À l'instant 0, on a : d'une part : $\forall n,\ n \cdot 0 = 0$. D'autre part, `mult(n,0)` renvoie 0. La propriété de récurrence est vraie.
- À l'instant p, on considère la propriété de récurrence est vraie à l'instant p : $mult(n,p) = n \cdot p$.
- À l'instant $p+1$, on applique l'algorithme : p étant différent de 0, l'algorithme retourne $mult(n,p+1) = n + mult(n,p)$. D'après la propriété de récurrence, on a donc $mult(n,p+1) = n + n \cdot p = n(p+1)$. La propriété est donc vraie au rang $p+1$.
- L'algorithme calcule donc le produit np.

Question 3 *Donner et justifier la complexité temporelle de la fonction* `mult`.

Correction On note $C(p)$ le nombre d'appels récursifs : $C(p) = 1 + C(p-1) = 1 + 1 + C(p-2) = p + T(0)$. On a donc $C(p) = \mathcal{O}(p)$. La complexité temporelle est linéaire.

Question 4 *Donner et justifier la complexité spatiale de la fonction* `mult`.

Correction On stocke une valeur à chaque appel récursif. Si ce stockage est à coût constant, étant donné qu'il y a n appels récursifs, la complexité spatiale est en $\mathcal{O}(n)$.

Exercice 2

Soit l'algorithme suivant :

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

Question 1 *Énoncer un variant de boucle et montrer la terminaison de l'algorithme.*

Question 2 *Énoncer un invariant de boucle et montrer la correction de l'algorithme.*

Correction
- Soit $\mathscr{P}$ la propriété d'invariance : à l'itération p, on a $puiss(x,n) = x^n$.
- À l'instant 0, on a : d'une part : $\forall x > 0$, $x^0 = 1$. D'autre part, `puiss(x,0)` renvoie 1. La propriété de récurrence est vraie.
- À l'instant n, on considère la propriété de récurrence est vraie et à l'instant p : $puiss(x,n) = x^n$.
- À l'instant $n+1$, on applique l'algorithme : n étant différent de 0, l'algorithme retourne $puiss(x,n+1) = x * mult(x,n)$. D'après la propriété de récurrence, on a donc $puiss(x,n+1) = x * x^n = x^{n+1}$. La propriété est donc vraie au rang $n+1$.
- L'algorithme calcule donc le produit x^n.

Question 3 *Donner et justifier la complexité temporelle de la fonction* `puiss`.

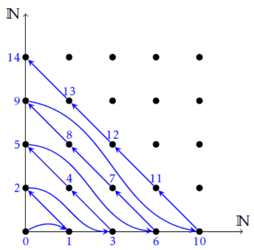
Correction On note $C(p)$ le nombre d'appels récursifs : $C(p) = 1 + C(p-1) = 1 + 1 + C(p-2) = p + T(0)$. On a donc $C(p) = \mathcal{O}(p)$. La complexité temporelle est linéaire.

Question 4 *Donner et justifier la complexité spatiale de la fonction* `puiss`.

Correction On stocke une valeur à chaque appel récursif. Si ce stockage est à coût constant, étant donné qu'il y a n appels récursifs, la complexité spatiale est en $\mathcal{O}(n)$.

Exercice 3

On démontre que sur l'ensemble $\mathbb{N} \times \mathbb{N}$ est dénombrable en numérotant chaque couple $(x, y) \in \mathbb{N}^2$ suivant le procédé suggéré par la figure ci-dessous.



Question 1 *Rédiger une fonction récursive qui retourne le numéro du point de coordonnées (x, y).*

Correction
```
def numerote(x, y):
    if x == 0 and y == 0:
        return 0
    if y > 0:
        return 1 + numerote(x+1, y-1)
    return 1 + numerote(0, x-1)
```

Question 2 *Rédiger la fonction réciproque, là encore de façon récursive.*

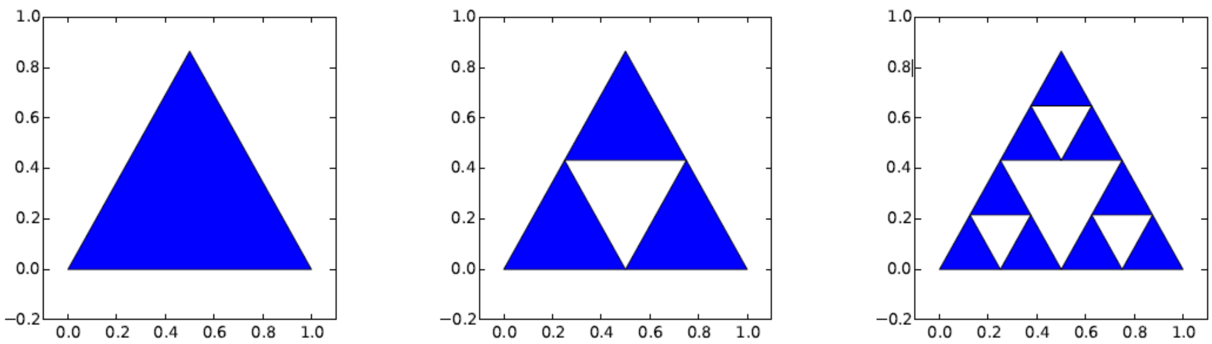
Correction

```python
def reciproque(n):
    if n == 0:
        return (0, 0)
    (x, y) = reciproque(n-1)
    if x > 0:
        return (x-1, y+1)
    return (y+1, 0)
```

Exercice 4

On suppose disposer d'une fonction polygon((xa, ya), (xb, yb), (xc, yc)) qui trace le triangle plein dont les sommets ont pour coordonnées $(x_a; y_a)$, $(x_b; y_b)$, $(x_c; y_c)$.

Question 1 *Définir une fonction récursive permettant le tracé présenté figure suivante (tous les triangles sont équilatéraux).*



Correction

```python
import numpy as np
import matplotlib.pyplot as plt

def triangle(A,B,C):
    """
    Entrées :
     * A,B,C : couples de coordonnées des points A B et C
    Sortie  :
     * Rien (plot)
    """
    X,Y = [],[]
    X.append(A[0])
    X.append(B[0])
    X.append(C[0])
    Y.append(A[1])
    Y.append(B[1])
    Y.append(C[1])
    plt.fill(X,Y,'b')

#triangle((0,0),(1,0),(.5,np.sqrt(3)/2))

def trace(n,A,B,C):
    if n==1:
        triangle(A,B,C)
    else :
        a = (.5*(B[0]+C[0]),.5*(B[1]+C[1]))
        b = (.5*(C[0]+A[0]),.5*(C[1]+A[1]))
        c = (.5*(A[0]+B[0]),.5*(A[1]+B[1]))
        trace(n-1,A,c,b)
        trace(n-1,c,B,a)
        trace(n-1,b,a,C)

trace(4,(0,0),(1,0),(.5,np.sqrt(3)/2))
```

Exercice 5

Question 1 *Écrire une fonction récursive qui calcule a^n en exploitant la relation : $a^n = a^{n/2} \times a^{n/2}$.*

Correction
```python
def power(a, n):
    if n == 0:
        return 1
    elif n == 1:
        return a
    return power(a, n//2) * power(a, n-n//2)
```

Question 2 *Écrire une fonction qui utilise de plus la remarque suivante :* $n/2 = \begin{cases} n/2 & \text{si } n \text{ est pair} \\ n/2+1 & \text{sinon} \end{cases}$.

Correction
```python
def power(a, n):
    if n == 0:
        return 1
    elif n == 1:
        return a
    x = power(a, n//2)
    if n % 2 == 0:
        return x * x
    else:
        return x * x * a
```

Question 3 *Déterminer le nombre de multiplications effectuées dans le cas où n est une puissance de 2, et majorer simplement ce nombre dans le cas général.*

Correction On note $C(n)$ le nombre multiplications.

Dans le premier cas, on note $n = 2^k$ et on conjecture que $\forall n > 2, C(n) = k-1 = \ln_2(n)-1$ et on raisonne par récurrence.

Dans le second cas, on conjecture que $\forall n > 2, C(n) \leq 2\ln_2 n$ et on raisonne par récurrence. En effet, le variant est $\ln_2 n$, qui diminue d'au moins 1 à chaque appel récursif. Il y a donc au plus $\ln_2 n$ appels récursifs. Or, il y a au plus 2 multiplications par appel.

Exercice 6 – Fonction 91 de McCarthy

On considère la fonction récursive suivante :

■ **Python**
```python
def f(n) :
    if n>100 :
        return n-10
    return f(f(n+11))
```

Question *Prouver sa terminaison lorsque $n \in \mathbb{N}$ et déterminer ce qu'elle calcule (sans utiliser l'interpréteur de commande).*

Correction Distinguons plusieurs cas.

- Si $n \geq 101$, l'algorithme se termine et renvoie $n-10$.
- Si $n \in [\![90, 100]\!]$, montrons par récurrence descendante que le calcul de $f(n)$ termine et renvoie 91.
  C'est immédiat pour $n = 100$: $f(100) = f(f(111)) = f(101) = 91$.
  Soit $n \in [\![91, 100]\!]$ tel que $f(n) = 91$. Alors $f(n-1) = f(f(n+10)) = f(n)$ car $n+10 > 100$ donc $f(n+10) = n$.
  La propriété est donc héréditaire.
  On raisonne ensuite par récurrence descendante de 10 en 10 : si $n \in [\![80, 89]\!]$, alors $n+11 \in [\![90, 100]\!]$, donc $f(n) = f(f(n+11)) = f(91) = 91$ d'après le cas précédent. Et ainsi de suite pour les intervalles $[\![70, 79]\!] \cdots [\![0, 9]\!]$.