Devoir Surveillé 3 – 1 heure

Algorithmique et programmation

Avant-propos – Calcul d'une puissance

On souhaite calculer la puissance b d'un nombre x : x^b avec $x \in \mathbb{R}$ et $b \in \mathbb{N}$. On utilise pour cela la fonction expo basée sur un algorithme naïf prenant comme argument un entier naturel b et un nombre réel x :

```python
def expo(x,b):
    res = 1
    j=b
    inv = x
    while j>=1:
        res = res * x
        j=j-1
    return res
```

Question 1
Proposer une autre formulation de l'algorithme de la fonction expo en utilisant une boucle for.

Corrigé

```python
def expo(x,n):
    res = 1
    for i in range(n):
        res =res*x
    return res
```

Question 2
*On conserve la fonction expo utilisant la boucle while. Montrer que j est un **variant** de boucle.*

Corrigé

La boucle while est conditionnée par $j >= 1$. Par ailleurs, j est toujours positif est décroit à chaque boucle. j est donc un variant de boucle. Il nous assure que l'algorithme se terminera.

Question 3
*On conserve toujours la fonction expo utilisant la boucle while. Montrer que la propriété $\mathscr{P}(n)$ $x^b = inv_n^{j_n} \cdot res_n$ est un **invariant** de boucle.*

Corrigé

1. Initialement, $res = 1$, $j = n$.
2. L'invariant de boucle suggéré est $x^b = inv_n^{j_n} \cdot res_n$.
3. Montrons la validité de notre invariant :
   - au rang 0 : $j_0 = b$, $inv_0 = x$, $res_0 = 1$. On a donc $inv_0^{j_0} \cdot res_0 = x^b \cdot 1 = x^b$. La propriété est donc vraie.
   - au rang n : supposons que la propriété $inv_n^{j_n} \cdot res_n$ vraie.
   - au rang $n+1$: $j = j_n - 1$, $res_{n+1} = res_n \cdot x$, $inv_n = x$. On a donc : $inv_{n+1}^{j_{n+1}} \cdot res_{n+1} = x^{j_n - 1} \cdot res_n \cdot x = x^{j_n} \cdot x^{-1} \cdot res_n \cdot x = x^{j_n} \cdot res_n = x^b$. La propriété est donc vérifiée au rang $n+1$.
4. La terminaison du programme est vérifiée par l'existence du variant de boucle j.
5. En sortie de boucle, $j = 0$, et $res_n = x^b$. En conséquence, l'invariant de boucle est encore vrai.

Question 4

On note C_e le coût d'une opération élémentaire (affectation, opération mathématique simple, incrémentation de boucle, comparaison). Évaluer la complexité temporelle de l'algorithme proposé dans la fonction expo.

Corrigé

La fonction exo est constituée :
- trois affectations de coût C_e (coût total $3_C e$) ;
- une boucle while qui doit s'exécuter b fois et qui est constituée :
  - de deux instructions composées de de 2 affectations et de deux opérations élémentaires (coût total $4_C e$) ;
- du coût du return de coût C_e.

Au final, le coût temporel est de :

$$C_T(b) = 3 \cdot C_e + b \cdot 4C_e + C_e$$

Ainsi, $C_T(b) \underset{+\infty}{\sim} 4C_e b$. La complexité algorithmique est donc linéaire (en $\mathcal{O}(n)$).

Question 5

Citer une méthode plus efficace permettant de calculer x^b. Détailler brièvement son fonctionnement et préciser sa complexité temporelle.

Corrigé

La méthode d'exponentiation rapide permet de calculer plus rapidement x^b. Sa complexité est en $\mathcal{O}(log(n))$. Pour rappel, x^b se calcule ainsi :

$$x^b \begin{cases} \text{si } b=0 \quad x^b = 1 \\ \text{si b est pair, } x^b = x^{\frac{b}{2}} \cdot x^{\frac{b}{2}} \\ \text{si b est impair, } x^b = x^{b-1} \cdot x \end{cases}$$

Calcul de polynômes

On cherche à évaluer un polynôme en différentes valeurs. On note :

$$\forall x \in \mathbb{R} \quad P(x) = \sum_{i=0}^{n} a_i x^i$$

Les coefficients a_i du polynôme sont des entiers positifs stockés dans un tableau a tels que $a = [a_0, a_1, a_2, ..., a_n]$. La fonction suivante appelée evaluer prend comme argument un nombre flottant x et un tableau a contenant les coefficients a_i du polynôme. Ainsi, si $a = [0,1,2,3]$, alors $a[0] = a_0$, $a[1] = a_1$, etc. alors $P(x) = x + 2x^2 + 3x^3$. La fonction evaluer retourne $P(x)$.

python

```python
def evaluer(a,x):
    for i in range(len(a)):
        res = res+a[i]*expo(x,i)
    return res
```

Question 6

La fonction evaluer a-t-elle l'effet désiré ? Si non, modifier le programme.

Corrigé

Il est nécessaire d'initialiser la variable res à 0.

Question 7

Estimer la complexité algorithmique de la fonction evaluer.

Corrigé

Pour un polynôme de degré n, la boucle for s'exécutera $n+1$ fois.
Au rang i, le coût de la fonction expo est $3\cdot C_e + i\cdot 4C_e + C_e$.
Le coût d'un incrément de boucle est donc $C(i) = 3\cdot C_e + i\cdot 4C_e + C_e + 4C_e$
On a donc un coût total $C(i) = \sum_{0}^{n+1} C(i)$.
On peut donc en conclure que la complexité sera en $\mathcal{O}(n^2)$.

Méthode de Horner

Afin de diminuer le coût temporel de l'évaluation d'un polynôme, il est possible d'utiliser la méthode de Horner. Elle consiste en une réécriture du polynôme $P(x)$:

$$P(x) = a_0 + x\left(a_1 + x\left(a_2 + x\left(a_3 + ...\right)\right)\right)$$

Ainsi le polynôme $P(x) = x + 2x^2 + 3x^3$ est réécrit ainsi : $P(x) = 0 + x(1 + x(2 + 3x))$.

python

```python
def horner(a,x):
    res=0
    n = len(a)-1
    while n>=0:
        res = a[n]+x*res
        n=n-1
    return res
```

Question 8

On prend $a = [0,1,2,3]$ et $x = 2$. En remplissant un tableau, donner l'évolution des variables res et n à chaque incrément de boucle.

Corrigé

n	res(x)	res
3	$res = 3 + x\cdot 0 = 3$	3
2	$res = 2 + x\cdot 3$	8
1	$res = 1 + x(2 + x\cdot 3) = 1 + 2x + 3x^2$	17
0	$res = 0 + x\left(1 + 2x + 3x^2\right) = x + 2x^2 + 3x^3$	34

Question 9

Expliquer en quoi l'algorithme proposé répond à la réécriture du polynôme $P(x)$ suivant la méthode de Horner ?

Corrigé

Cf question précédente.

Question 10

Estimer la complexité algorithmique de la fonction horner. Conclure sur l'intérêt de cet algorithme.

Corrigé

On constate directement que la complexité de l'algorithme est linéaire ce qui lui confère une plus grande rapidité que la méthode naïve.

Intégration numérique

On cherche maintenant à intégrer numériquement $P(x)$ sur l'intervalle $[u, v]$ par la méthode des rectangles à gauche :

$$I = \int_u^v P(x)\,\mathrm{d}x$$

Question 11

Écrire la fonction integrale_rectangle prenant comme argument le nombre d'échantillons n, le tableau a des coefficients du polynôme ainsi que u et v les bornes de l'intégrale et retournant la valeur I de l'intégrale.

Corrigé

Corrigé

python

```python
def integrale_rectangle(n,u,v,a):
    res = 0
    pas = (v-u)/n)
    for i in range(0,n):
        val = a+pas*i
        res = res + pas*horner(a,val)
    return res
```

Question 12

Quel est l'ordre de grandeur de l'erreur effectuée sur le calcul de l'intégrale.

Corrigé

Pour n échantillons, l'erreur peut être majorée par $\dfrac{M}{2n}$ avec M le sup de $P'(x)$ sur l'intervalle $[u, v]$.

Il est à noter qu'utiliser la méthode des rectangles pour calculer l'intégrale d'un polynôme n'est pas forcément judicieux. En effet, il est aisé de trouver une primitive de $P(x)$.