Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 6

Code UE : SPO1U57

Année 2023–2024

Algorithmes numériques

Exercice 1 (Calcul de la puissance d'un entier)

L'élévation d'un nombre à une puissance entière positive est une opération de base cruciale lorsqu'on veut calculer avec des valeurs numériques. C'est même l'étape de base dans la méthode cryptographique RSA. Étant donné un nombre x (cela peut être un entier ou un réel, qu'on représente en machine à l'aide d'un flottant) et un entier naturel n, on souhaite donc calculer le nombre $x^n = \underbrace{x \times x \times \cdots \times x}_{n \text{ fois}}$. On a vu en cours une façon efficace de calculer x^n à l'aide de l'algorithme suivant :

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

1. Exécutez cet algorithme lors du calcul de 2^{10}, en précisant la valeur des variables lors de chaque étape de la boucle. Combien de multiplications sont effectuées ?
2. Dans le cas général, combien de multiplications effectue cet algorithme pour calculer x^n, en fonction de x et n ?

Voici une façon plus efficace de calculer x^n à l'aide de l'algorithme dit d'exponentiation (c'est le nom qu'on donne à l'*élévation à la puissance*) rapide :

```
def fastexp(x, n):
    a = 1
    b = x
    m = n
    while m > 0:
        if m % 2 == 1:
            a = a * b
        b = b * b
        m = m // 2
    return a
```

1. Exécutez cet algorithme lors du calcul de 2^{10}, en précisant la valeur des variables lors de chaque étape de la boucle.
2. Donnez une preuve du fait que l'algorithme termine toujours. (*Indication : commencer par étudier pour quelle raison cet algorithme pourrait ne pas terminer*)
3. Pour prouver que l'algorithme est correct, c'est-à-dire qu'il renvoie bien x^n, une méthode consiste à vérifier qu'à tout moment de l'algorithme, on a $x^n = a \times b^m$. Montrez que c'est vrai avant de rentrer dans la boucle, puis que si c'est vrai au début d'une itération de la boucle, alors c'est vrai à la fin de cette itération. Concluez alors à l'aide d'un raisonnement par récurrence.
4. Les opérations élémentaires coûteuses d'un algorithme d'exponentiation sont les multiplications. Combien de multiplications sont effectuées par l'algorithme lors du calcul de 2^{10}. Par souci de généralité, donnez un ordre de grandeur du nombre de multiplications effectuées pour calculer x^n par l'algorithme, en fonction de n.

Exercice 2 (Aire sous la courbe) Soit $f\colon \mathbb{R} \to \mathbb{R}$ une fonction d'une variable réelle, par exemple $f(x) = 3x^2 - 3$, avec son implémentation en Python :

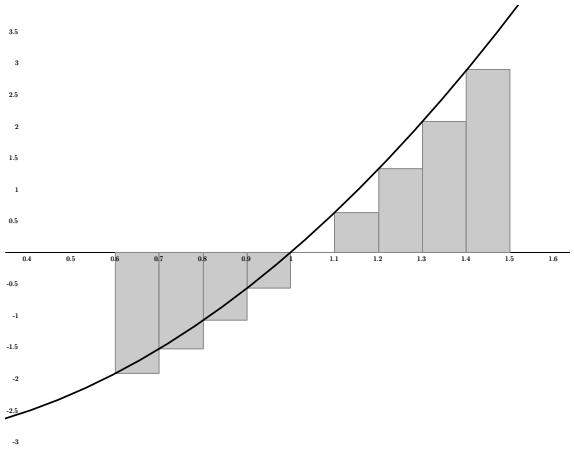
```
def f(x):
    return 3 * x**2 - 3
```

Il est possible de calculer une approximation de l'aire sous la courbe $f(x)$ entre les points a et b avec la méthode des rectangles. D'abord, on choisit une petite valeur pour ϵ, par exemple $\epsilon = 0,1$, ce qui donne en Python :

```
EPSILON = 0.1
```

Ensuite, on construit pour chaque point $x = a, a + \epsilon, a + 2\epsilon, a + 3\epsilon, \ldots$ jusqu'à b non compris un rectangle de largeur ϵ et de hauteur $f(x)$, dont il est simple de calculer l'aire. Si $f(x) < 0$, alors l'aire sera compté en négatif.

L'approximation de l'aire sous la courbe est obtenue comme somme des aires des rectangles. Voici un exemple avec $a = 0,6$ et $b = 1,5$



Écrivez une fonction `aire(a, b)` qui prend en entrée deux nombres réels a et b (avec $a < b$) et calcule l'aire sous la courbe $f(x)$ entre a et b.

Exercice 3 L'algorithme suivant calcule la partie entière du logarithme en base 2 de l'entier strictement positif `n`, c'est-à-dire le plus grand entier i tel que $2^i \leq$ `n`.

```
def log2(n):
    i = 0
    m = n
    while m > 1:
        i = i + 1
        m = m // 2
    return i
```

1. Exécutez l'algorithme `log2` sur l'entier `n` $= 8$, en donnant toutes les valeurs des variables pendant l'exécution, le résultat et en comptant le nombre de tours de boucle effectués.
2. Même question pour l'entier `n` $= 7$.
3. Expliquez pourquoi l'algorithme `log2` termine pour toute entrée `n` > 0.
4. Calculez, en justifiant, le nombre d'instructions effectuées par l'algorithme `log2` en fonction de la valeur de `n`, prise quelconque. Vous pourrez simplifier le résultat en utilisant la notation « grand $\mathcal{O}$ ».