

Algorithmique et Programmation avancée

Calcul de x^n

Énoncé du problème

On étudie le problème du calcul de x^n , étant donnés x et n (n étant un entier positif). x n'est pas nécessairement un nombre, il peut s'agir d'une matrice ou d'un polynôme à plusieurs indéterminées: si la multiplication a un sens, la division n'en a pas! On pose $y_0 = x$, et on utilise la règle suivante: si on a déjà calculé $y_1, y_2, \dots, y_{i-1}$, on peut calculer y_i comme produit de deux résultats précédents arbitraires :

$$y_i = y_j * y_k, \text{ avec } 0 \leq j, k \leq i-1$$

Le but est d'atteindre x^n le plus vite possible, i.e. de trouver un algorithme qui minimise i tel que $y_i = x^n$.

Algorithme naïf

Considérons l'algorithme dit "naïf" suivant: on multiplie n fois x par lui-même:

$$y_i = y_0 * y_{i-1}$$

On a $y_{n-1} = x^n$, le coût (nombre d'opérations) est donc de $n-1$.

Méthode binaire

On peut trouver un algorithme plus efficace :

$$x^n = \begin{cases} x^{n/2} * x^{n/2} & \text{si } n \text{ est pair} \\ x^{(n-1)/2} * x^{(n-1)/2} * x & \text{si } n \text{ est impair} \end{cases}$$

On peut aussi formuler l'algorithme de la façon suivante. Si l'on part de x l'algorithme ne fait que des "additions de 1" et des "multiplications par 2" de la puissance pour atteindre x^n , il peut alors être judicieux d'écrire n en binaire.

On remplace alors chaque "1" par SX et chaque "0" par S, et on enlève le premier SX (celui qui est à gauche). Le mot obtenu donne un façon de calculer x^n , en traduisant S par l'opération mettre au carré (squaring), et X par l'opération multiplier par x .

Par exemple, pour $n = 23$, la chaîne obtenue est SX S SX SX SX, en enlevant le premier SX, on obtient SSXSXSX. On calcule donc dans l'ordre $x^2, x^4, x^5, x^{10}, x^{11}, x^{22}, x^{23}$.

Le coût est borné par $2 \log_2 n$,

Cette méthode binaire n'est pas optimale : par exemple avec $n = 15$, on obtient la chaîne SXSXSX, d'où 6 multiplications alors qu'en remarquant que $15 = 3 * 5$, on a besoin de 2 multiplications pour trouver $y = x^3$, ($y = (x * x) * x$) puis de 3 autres pour calculer $x^{15} = y^5$ (on applique la méthode binaire : y^2, y^4, y^5).

Remarque. On pourra proposer une programmation récursive de cet algorithme.

Méthode des facteurs

La méthode des facteurs est basée sur la factorisation de n :

$$x^n = \begin{cases} (x^p)^q & \text{si } p \text{ est le plus petit facteur premier de } n, \\ x^{n-1} * x & \text{si } n \text{ est premier} \end{cases}$$

Remarque. Avec les puissances de 2, cette méthode est identique à la méthode binaire.

Cette méthode n'est pas optimale, par exemple pour $n = 33$ on a 7 multiplications avec la méthode des facteurs et seulement 6 avec la méthode binaire.

Il existe une infinité de nombres pour lesquels la méthode des facteurs est meilleure que la méthode binaire (prendre $n = 15 * 2k$), et réciproquement (prendre $n = 33 * 2k$).

Il faut souligner que le coût de la recherche de la décomposition de n en facteurs premiers n'est pas pris en compte dans notre formulation. C'est pourtant nécessaire pour quantifier correctement le coût de la méthode des facteurs. Le problème est qu'on ne sait pas, à ce jour, trouver la décomposition en temps polynomial en n .

Il existe d'autres façons de calculer les puissances d'un élément, l'arbre de Knuth est, par exemple, statistiquement plus intéressant.

Exercices

Exercice 1: Un peu de code

Proposer un pseudo code pour l'algorithme naïf, puis l'algorithme binaire comme proposé puis en récursif. Les coder en Python3.

Exercice 2: Min et Max d'une liste de nombres

On considère une liste de nombres:

1. Proposer un algorithme simple pour trouver le min et le max d'une liste, calculer le nombre de tests effectués par votre algorithme, puis le coder.
2. Déterminer un algorithme à priori plus efficace. (prendre les éléments de la liste 2 par 2)
3. Avec le module *time* de Python, comparer les temps d'exécution de ces 2 algorithmes et conclure.

Exercice 3: Hauteur fatale

Le problème est de déterminer à partir de quel étage d'un immeuble, sauter par une fenêtre est fatal. Vous êtes dans un immeuble à n étages (numérotés de 1 à n) et vous disposez de k étudiants. Il n'y a qu'une opération possible pour tester si la hauteur d'un étage est fatale: faire sauter un étudiant par la fenêtre. S'il survit, vous pouvez le réutiliser ensuite, sinon vous ne pouvez plus.

Vous devez proposer un algorithme pour trouver la hauteur à partir de laquelle un saut est fatal en faisant le minimum de sauts.

1. Si $k \geq \log_2 n$, proposer un algorithme en $O(\log_2 n)$ sauts (dichotomie).
2. Si $k = 1$, proposer une méthode en $O(n)$. En déduire, un algorithme en $O(k + \frac{n}{2^{k-1}})$ sauts si $k < \log_2 n$.
3. Si $k = 2$, proposer un algorithme en $O(\sqrt{n})$ sauts.

Corrections

Exercice 1: Un peu de code

Pseudo-codes:

```
puissance(x,n):
#Entrées: Element  $x$ , entier positif  $n$ .
#Sorties: Element  $x^n$ .
    res  $\leftarrow x$ 
    Pour  $i$  allant de 1 à  $n - 1$ :
        res  $\leftarrow res * x$  #on multiplie par  $x$ 
    Renvoie res
```

```
puissanceBinaire(x,n):
#Entrées: Element  $x$ , entier positif  $n$ .
#Sorties: Element  $x^n$ .
    res  $\leftarrow x$ 
    bin  $\leftarrow$  conversion de  $n$  en binaire
    li  $\leftarrow$  conversion de bin en liste de SX et S
    Suppression du premier SX dans li
    Lire li de gauche à droite:
        If S: res  $\leftarrow res * res$ 
        else: res  $\leftarrow res * x$ 
    Renvoie res
```

```
puissanceRec(x,n):
#Entrées: Element  $x$ , entier positif  $n$ .
#Sorties: Element  $x^n$ .
    If  $n = 1$ : renvoie  $x$  # Initialisation
    Else if  $n$  pair:
        renvoie puissanceRec(x,n/2)*puissanceRec(x,n/2)
    Sinon:
        renvoie puissanceRec(x,(n-1)/2)*puissanceRec(x,(n-1)/2)*x
```

Note: Il est judicieux de calculer puissanceRec(x,n/2) avant de le multiplier.

Exercice 2: min et max

Pseudo-codes:

```
minMax(l):
#Entrées: Liste  $li$  de  $n$  nombres.
#Sorties: Paire  $(m, M)$  du min  $m$  et du max  $M$  de  $li$ .
    m=M=li[0]
    Pour  $i$  allant de 1 à  $n - 1$ : # recherche du min
        If li[i]<m: m $\leftarrow$ li[i].
    Pour  $i$  allant de 1 à  $n - 1$ : # recherche du max
        If li[i]>M: M $\leftarrow$ li[i].
    Renvoie (m,M)
```

```
minMax(l):
#Entrées: Liste  $li$  de  $n$  nombres.
#Sorties: Paire  $(m, M)$  du min  $m$  et du max  $M$  de  $li$ .
    m=M=li[0]
    Pour  $i$  allant jusqu'à  $n - 1$  de 2 en 2: # Attention à la parité de  $n$ 
        If li[i] $\leq$ li[i+1]:
            If li[i]<m: m $\leftarrow$ li[i].
            If li[i+1]>M: M $\leftarrow$ li[i+1].
        Else
            If li[i+1]<m: m $\leftarrow$ li[i+1].
            If li[i]>M: M $\leftarrow$ li[i].
    Renvoie (m,M)
```

Exercice 3: hauteur fatale

1. La complexité en $O(\log_2 n)$ qui est indiquée nous aiguille vers une dichotomie. En effet, en supposant que l'on a $k \geq \log_2 n$, on obtient le résultat sur les étages de i à j en jetant un étudiant depuis le m -ème étage, où $m = \frac{i+j-1}{2}$, puis en itérant le procédé sur les étages de i à m si la chute a été fatale, et sur les étages de $m+1$ à j dans le cas contraire. La méthode dichotomique nous garantit alors que l'on obtient le bon résultat (lorsqu'il ne reste plus qu'un seul étage, c'est-à-dire lorsque l'on calcule pour les étages de $i = j$), et ce avec une complexité, logarithmique dans le pire des cas.
2. Puisqu'ici on ne dispose que de $k < \log_2 n$ étudiants, on ne peut pas appliquer directement la méthode dichotomique proposée précédemment. Cependant, on va remédier au problème de manière simple en appliquant une recherche dichotomique avec $k - 1$ étudiants, de manière à délimiter un intervalle d'étages dans lequel se trouve l'étage recherché. On se sert alors du dernier étudiant restant pour parcourir l'intervalle de façon linéaire, donc en le jetant de chaque étage en partant du plus bas de l'intervalle, jusqu'au plus haut. Après avoir jeté les $k - 1$ premiers étudiants, si l'on n'a pas encore trouvé le bon étage, il reste exactement $\frac{n}{2^{k-1}}$ étages dans l'intervalle de recherche, d'où une complexité dans le pire des cas en $O(k + \frac{n}{2^{k-1}})$ sauts.
3. Dans le cas particulier où l'on a $k = 2$, on ne veut pas avoir à tester chaque étage de façon linéaire, c'est pourquoi on va reprendre à notre compte les idées précédentes, et notamment celle qui consiste à délimiter un intervalle de recherche. Nous découpons donc l'ensemble des étages $\sqrt{n}$ en "tranches" de $\sqrt{n}$ étages, avant de jeter le premier étudiant de chacun des étages de début de tranche. Lorsque l'étudiant n'est pas en état de remonter, on se ramène au dernier étage n testé qui ne soit pas fatal, et on n'a plus qu'à parcourir de manière linéaire l'intervalle allant de l'étage $n+1$ à l'étage fatal trouvé précédemment. On a ainsi deux séries d'essais en $O(\sqrt{n})$, et donc une complexité finale dans le pire des cas également en $O(\sqrt{n})$ sauts.