Agrégation- 2023/2024

Préparation à l'épreuve orale, Option C

Thierry Mignon
thierry.mignon@umontpellier.fr
Mars 2024

Travaux Pratique : Complexité algorithmique

Bibliographie : *Algorithmique*, Cormen, Leiserson, Rivest, Stein.

1 Avant de commencer

La complexité est un outil qui permet de mesurer l'efficacité d'un algorithme, indépendamment de la machine qui l'exécute. Elle permet de déterminer si un problème donné peut être résolu informatiquement (sous-entendu, en temps raisonnable !) et, dans le cas où plusieurs solutions sont proposées, laquelle est la plus rapide. Bien sûr, la notion de "raisonnable" dépend du problème étudié, mais vous pouvez considérer, en première approximation, que les algorithmes pires que polynomiaux ne sont pas intéressants en pratique.

Remarque : La complexité s'exprime en fonction de la taille de l'entrée. Si votre problème consiste à trier une liste de n élements, il est naturel de l'exprimer en fonction de n. S'agissant d'un problème d'arithmétique/cryptographie, la taille de l'entrée pertinente est la longueur du mot binaire permettant de stocker les entiers de départ (et non les entiers eux-mêmes) !

Temps	Type de complexité	Temps pour n = 5	Temps pour n = 50	Temps pour n = 250	Temps pour n = 1 000	Temps pour n = 10 000	Temps pour n = 1 000 000	Problème exemple
$O(1)$	complexité constante	10 ns	10 ns	10 ns	10 ns	10 ns	10 ns	Accès tableaux
$O(\log(n))$	complexité logarithmique	10 ns	20 ns	30 ns	30 ns	40 ns	60 ns	Recherche dichotomique
$O(n)$	complexité linéaire	50 ns	500 ns	2.5 μs	10 μs	100 μs	10 ms	Parcours de liste
$O(n\log(n))$	complexité linéarithmique	40 ns	850 ns	6 μs	30 μs	400 μs	60 ms	Tris dont le Tri fusion ou le Tri par tas
$O(n^2)$	complexité quadratique (polynomiale)	250 ns	25 μs	625 μs	10 ms	1 s	2.8 heures	Parcours de tableaux 2D
$O(n^3)$	complexité cubique (polynomiale)	1.25 μs	1.25 ms	156 ms	10 s	2.7 heures	316 ans	Multiplication matricielle non-optimisée.
$2^{\text{poly(n)}}$	complexité exponentielle	320 ns	130 jours	10^{59} ans	...	...	...	Problème du sac à dos par force brute.
$O(n!)$	complexité factorielle	1.2 μs	10^{48} ans	...	...	...	...	Problème du voyageur de commerce (avec une approche naïve).

2 L'exponentiation

2.1 Premières expériences

1. Essayer de calculer brutalement `(3^400000000).mod(123)`.
2. Essayer avec `power_mod`.

2.2 Implémentation de l'algorithme d'exponentiation naïf

```
def expo_naive(x,n):
  result = 1
  for k in range(n) :
     result = result*x
  return result
```

1. Que fait cet algorithme ? Quelle est sa complexité (en fonction de n) ?

2.3 Implémentation de l'algorithme d'exponentiation rapide

```
def expo_rapide(x,n) :
  if n==0 :
  return 1
  else :
  if n% 2==0 :
  return expo_rapide(x,n/2)^ 2
  else :
  return x*expo_rapide(x,(n-1)/2))^ 2
```

1. Prouver que cet algorithme calcule bien x^n.
2. Quelle est sa complexité ? Justifier.

2.4 Banc d'essai

Grâce à la commande `% time` placée en début de cellule, vous pouvez connaître le temps nécessaire au processeur pour effectuer votre calcul. (Comparer avec la fonction `timeit`).

1. *Sur les entiers :* Donner un ordre de grandeur de la puissance à laquelle vous pouvez élever 3, sans que le calcul par `expo_naive` ne dure plus de dix secondes. Pour ces valeurs, combien de temps prend `expo_rapide` ? Jusqu'où peut monter `expo_rapide` en dix secondes ?
2. *Sur $\mathbb{Z}/n\mathbb{Z}$:* Mêmes questions pour l'élément 7 de $\mathbb{Z}/29\mathbb{Z}$. (utiliser les commandes `G=IntegerModRing(29)`, `G(7)`)
3. *Sur $\mathbb{Q}[X]$:* Mêmes questions pour le polynôme $X + 1$.
4. *Sur $M_n(\mathbb{Q})$:* Mêmes questions pour la matrice

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

3 Crible d'Eratosthène

Description de l'algorithme : Eratosthène était un savant grec, à la fois mathématicien, géographe, astronome et poète. Il dirigeait la grande bibliothèque d'Alexandrie deux siècles et demi avant JC. On lui doit, notamment, la première estimation vraisemblable du rayon de la Terre, ainsi que la méthode du crible, qui permet d'obtenir la liste de tous les nombres premiers inférieurs à une valeur fixée, disons N. Pour ce faire :

1. On dresse la liste de tous les entiers inférieurs à N.
2. On se place au début de la liste et l'on raye 0 et 1.
3. On avance jusqu'au prochain nombre non rayé et l'on raye tous ses multiples stricts.

4. On recommence au point 3 jusqu'à atteindre la fin de la liste.

Implémentation

1. Dérouler la procédure à la main jusqu'à 99.
2. Écrire l'algorithme et l'implémenter.
3. Prouver que cet algorithme fournit bien la liste des nombres premiers inférieurs à N.
4. Quelles améliorations pouvez-vous apporter ?
5. Faire un banc d'essai et comparer avec `prime_range`.
6. Calculer la complexité de votre algorithme (on admettra le résultat suivant :

$$\sum_{p \text{ premier}, p \leq n} \frac{1}{p} = O(\ln(\ln(n)))$$

4 Arithmétique sur les entiers

On étudie ici la complexité de l'arithmétique usuelle (addition, multiplication) des entiers écrits en base deux.

4.1 Développement binaire

L'écriture du développement binaire d'un nombre peut-être utilisée pour certains algorithmes d'exponentiation rapides par exemple. La détermination des chiffres d'un développement binaire peut se faire à l'aide de deux méthodes différentes, suivant si l'on commence par le bit de poids le plus fort ou le bit de poids le plus faible. Voici un exemple avec

$$n = 105 = \sum a_i 2^i$$

Première méthode

On a : $2^6 = 64 < 105 < 2^7 = 128$ donc $a_6 = \lfloor 105/64 \rfloor = 1$. Maintenant $105 - 64 = 41$, donc $a_5 = \lfloor 41/32 \rfloor = 1$. Puisque $41 - 32 = 9, a_4 = \lfloor 9/16 \rfloor = 0$. On continue de la même façon : $a3 = \lfloor 9/8 \rfloor = 1; 9 - 8 = 1$ donc $a_2 = a_1 = 0$ et $a_0 = 1$. L'écriture binaire de n est donc 1101001.

Deuxième méthode

$$\begin{aligned} 105 &= 2.52 + 1 \\ 52 &= 2.26 + 0 \\ 26 &= 2.13 + 0 \\ 13 &= 2.6 + 1 \\ 6 &= 2.3 + 0 \\ 3 &= 2.1 + 1 \\ 1 &= 2.0 + 1 \end{aligned}$$

- Écrire deux fonctions `decimalTObinary` et `binaryTOdecimal` qui transforment un entier en la liste des bits de son écriture binaire, et vice versa. On fera le choix de placer les bits de poids fort à droite.

  Ex : `decimalTObinary(14) = [0,1,1,1]`

- Écrire une fonction addition qui, étant donnés deux listes de bits (telles que spécifiées ci-dessus), renvoit la liste des bits de la somme. Quelle est la complexité de l'algorithme ?

- En vous inspirant des multiplications telles qu'on a l'habitude de les "poser" à la main, écrire une fonction multiplication qui, étant donnés deux listes de bits (telles que spécifiées ci-dessus), renvoit la liste des bits du produit. Quelle est la complexité de l'algorithme ?