Exemples d'algorithmes

Sont explicitement au programme (donc à savoir écrire) les algorithmes de

- recherche dans un tableau : renvoyer l'indice de la première apparition de `x` dans `T`, -1 s'il n'apparaît pas.
- recherche de l'indice du maximum d'un tableau `T`.
- calcul de la moyenne et de la variance :
  Rappel : $\mathrm{Var}(T) = \frac{1}{n}\sum_{i=1}^{n} T_i^2 - \overline{T}^2$ si $T = (T_1, \ldots, T_n)$.
- Recherche dichotomique dans un tableau trié (version exacte pour les entiers et à précision ε pour les flottants)
- Recherche de motif dans un texte

1 Indice du maximum

```
indiceMax(T)
    n ← taille(T)
    iMax ← 0
    max ← T_0
    Pour i entre 1 et n−1 Faire
        Si T_i > max Alors
            iMax ← i
            max ← T_i
        FinSi
    FinPour
    Retourner iMax
```

Invariant de sortie : max contient le maximum de $T_0, ..., T_i$ et $iMax$ l'indice de sa première apparition.

Complexité temporelle : au mieux $n + O(1)$ (max en première position), au pire $3n + O(1)$ (tableau strictement croissant). Donc $\Theta(n)$.

Complexité spatiale : $O(1)$.

2 Recherche dans un tableau

```
recherche(T,x)
    i ← 0
    n ← taille(T)
    Tant que i < n et T_i ≠ x Faire
        i ← i + 1
    FinTq
    Si i > n Alors
        i ← −1
    FinSi
    Retourner i
```

Invariant de sortie : x n'apparaît pas dans les $i+1$ premières cases du tableau.

Terminaison : stricte croissance de l'entier i majoré par n.

Complexité temporelle en comparaisons : au mieux 2 (tableau vide) ou 3 (x en première position), au pire $2n+2$ (x n'apparaît pas). Donc $O(n)$.
Complexité spatiale : $O(1)$.

3 Moyenne et variance

MoyVar(T)

$n \leftarrow \text{taille}(T)$
$som \leftarrow 0$
$somCar \leftarrow 0.$
Pour i *entre* 0 *et* $n-1$ **Faire**
  $som \leftarrow som + T_i$ $somCar \leftarrow somCar + T_i^2$
FinPour
$moy \leftarrow som/n$
Retourner $moy, somCar/n - moy^2$

Invariant de sortie :
$som = T_0 + \cdots + T_i$ et
$somCar = T_0^2 + \cdots + T_i^2$.

Complexité temporelle (+,*,-,/) :
$3n+4 = \Theta(n)$ opérations.

Complexité spatiale : $O(1)$.

4 Recherche dichotomique dans un tableau trié

rechercheDichotomie(T,x)

$n \leftarrow \text{taille}(T)$
$imin \leftarrow 0$
$imax \leftarrow n-1$
$imilieu \leftarrow \left\lfloor \frac{imin+imax}{2} \right\rfloor$
Tant que $imin < imax$ *et* $T_{imilieu} \neq x$ **Faire**
  Si $T_{imilieu} < x$ **Alors**
    $imin \leftarrow imilieu + 1$
  Sinon
    $imax \leftarrow imilieu - 1$
  FinSi
  $imilieu \leftarrow \left\lfloor \frac{imin+imax}{2} \right\rfloor$
FinTq
Si $imin \neq imax$ **Alors**
  $reponse \leftarrow imilieu$
Sinon
  $reponse \leftarrow -1$
FinSi
Retourner $reponse$

Invariant : $\mathscr{P}(k)$: « à la fin de l'étape k (donc si l'on n'est pas déjà tombé sur x),

$$imax - imin < \frac{n}{2^k}$$

et **si** x **est dans le tableau**,

$$T_{imin} \leqslant x \leqslant T_{imax}.»$$

Démonstration

Par récurrence :

- À la fin de l'étape 0, donc juste avant la boucle, si x est dans le tableau (trié), on a bien

$$T_0 = T_{imin} \leqslant x \leqslant T_{imax} = T_{n-1}$$

  et

$$imax - imin = n - 1 < \frac{n}{2^0}.$$

- Si c'est vrai à la fin de l'étape k, appelons a la valeur de min, b celle de max tels que $b - a < \frac{n}{2^k}$.
  Alors, $milieu$ devient

$$\left\lfloor \frac{imin+imax}{2} \right\rfloor = \left\lfloor \frac{a+b}{2} \right\rfloor.$$

Si $T_{imilieu} \neq x$, on ne sort pas de la boucle, et

- ⋆ Soit $T_{imilieu} > x$, alors $imin$ prend la valeur $imilieu + 1 = \left\lfloor \frac{a+b}{2} \right\rfloor + 1$ et $imax$ reste à b. **Si x est dans le tableau** (trié), on a alors bien $T_{imin} \leqslant x \leqslant T_{imax}$, et comme $\frac{a+b}{2} < imilieu + 1 \leqslant b$,
$$imax - imin < \frac{b-a}{2} < \frac{n}{2^{k+1}}$$
- ⋆ Soit $T_{imilieu} < x$, alors $imax$ prend la valeur $imilieu - 1 = \left\lfloor \frac{a+b}{2} \right\rfloor - 1$ et $imin$ reste à a. **Si x est dans le tableau** (trié), on a alors bien $T_{imin} \leqslant x \leqslant T_{imax}$, et comme $a \leqslant imilieu - 1 < \frac{a+b}{2}$,
$$imax - imin < \frac{b-a}{2} < \frac{n}{2^{k+1}}$$

ce qui établit la récurrence. □

Terminaison : $imax - imin \in \mathbb{N}$ et $imax - imin < \frac{n}{2^k}$ donc si k assez grand, $max - imin = 0$.

Correction : Soit on tombe sur x à un moment, soit on arrive à $imin = imax$, et si x est dans le tableau, $T_{imin} \leqslant x \leqslant T_{imax}$. Le résultat renvoyé est bien le bon.

Complexité temporelle (comparaisons) : 3 par étapes, avec un nombre d'étape maximal égal au premier entier k tel que $\frac{n}{2^k} < 1$, soit $2^k > n$ donc au plus $\lfloor \log_2(n) \rfloor + 1$ étapes, soit $T(n) = 3\lfloor \log_2(n) \rfloor + 3$ comparaisons. Cela se produit si x n'est pas dans le tableau.

Soit en général $\boxed{O(\ln n).}$

Complexité spatiale : $O(1)$.

Remarque

C'est mieux que l'algorithme naïf mais le tableau doit être trié... Le tri se faisant au minimum en $O(n \ln n)$ en général.

5 Recherche dichotomique dans un tableau trié de flottants

Il suffit de rajouter une précision ε en argument et de remplacer $T_{imilieu} \neq x$ par $|T_{imilieu} - x| > \varepsilon$.

6 Recherche d'un mot dans une chaîne de caractères

On écrit un algorithme « naïf » de recherche d'un mot dans une chaîne de caractères : on passe par une fonction `estIci(mot, chaine, i)` testant si `mot` se trouve en position `i` dans `chaîne`.

estIci(mot,chaine,i)

$j \leftarrow 0$

$m \leftarrow$ taille(mot)

Tant que $j < m$ *et* $mot[j] = chaine[i+j]$ **Faire**

  $j \leftarrow j+1$

FinTq

Retourner $j = m$

rechercheMot(mot,chaine)

$n \leftarrow$ taille($chaine$)

$m \leftarrow$ taille(mot)

$pos \leftarrow 0$

$pasTrouvé \leftarrow Vrai$

Tant que $i \leqslant n-m$ *et* $pasTrouvé$ **Faire**

  $pasTrouvé \leftarrow$ non($estIci(mot, chaine, i)$)

FinTq

Si $pasTrouvé$ **Alors**

  $i \leftarrow -1$

FinSi

Retourner i

Pour rechercheMot :

Invariant de sortie : mot ne se trouve pas en positions $0, 1, ..., i$ dans chaine.

Complexité (comparaisons de caractères) : au plus m pour estIci. Donc au mieux m ou $n-m$, au pire majoré par $m(n-m+1)$. Donc $T(n) = O(m(n-m))$.

7 Complément : l'exponentiation rapide

Pour calculer x^n, on remarque la chose suivante $x^{2^{k+1}} = \left(x^{2^k}\right)^2$: il suffit donc d'une multiplication pour passer de x^{2^k} à $x^{2^{k+1}}$.

L'idée est alors de décomposer n en base 2 : si n s'écrit

$$n = b_0 + b_1 \cdot 2 + b_2 2^2 + \cdots + b_k 2^k$$

avec $b_0, \cdots, b_k \in \{0,1\}$ (écriture en base 2) alors

$$x^n = x^{b_0} \left(x^2\right)^{b_1} \cdots \left(x^{2^k}\right)^{b_k},$$

le calcul de x^{2^p} s'effectuant à partir de celui de $x^{2^{p-1}}$.

Par exemple, $19 = 1+2+2^4$ et $x^{19} = x \times x^2 \times \left(\left(\left(x^2\right)^2\right)^2\right)^2$ (6 multiplications nécessaires), $39 = 1+2+2^2+2^5$ et

$$x^{39} = x \times x^2 \times \left(x^2\right)^2 \times \left(\left(\left(\left(x^2\right)^2\right)^2\right)^2\right)^2$$

(8 multiplications nécessaires).

```python
def exprap(x, n):
    """calcul de x puissance n par expon. rapide"""
    puis = 1
    xpuis2 = x
    m = n
    while m != 0:
        if m % 2 == 1:
            puis *= xpuis2
        xpuis2 *= xpuis2
        m //= 2
    return puis
```

Terminaison : `m` strictement décroissant minoré.

Invariant d'entrée : au début de la i^{e} étape, `m` contient $\left\lfloor \dfrac{n}{2^{i-1}} \right\rfloor$, `xpuis2` contient $x^{2^{i-1}}$, et `puis` contient

$$x^{b_0}\left(x^2\right)^{b_1}\cdots\left(x^{2^i}\right)^{b_i} = x^{b_0+b_1 2+\cdots+b_i 2^i}$$

où $b_i, \ldots, b_0$ sont les $i+1$ derniers chiffres de l'écriture en base 2 de n.

Correction : dernière étape : lorsque $m = 0$, on a bien obtenu tous les chiffres de l'écriture en base 2 de n et on renvoie bien x^n.

Complexité temporelle : Il y a autant d'étape que le nombre de chiffres N de l'écriture en base 2 de n, et au plus 2 produits à chaque étape.

Comme $2^{N-1} \leqslant n < 2^N$, on a $N = \lfloor \log_2 n \rfloor + 1$, donc la complexité $T(n)$ est telle que $T(n) \leqslant 2\log_2 n + 2$ et on a donc une complexité en $O(\ln n)$, logarithmique, ce qui est beaucoup (beaucoup) mieux que la méthode naïve linéaire.