Cours d'algorithmique
Tris simples - EISTI - ING 1

Ecole Internationale des Sciences du Traitement de l'Information

Problème du tri

Description

- Entrée : séquence de n nombres $a_1, a_2, \cdots, a_n$
- Sortie : permutation $a'_1, a'_2, \cdots, a'_n$ de la séquence d'entrée
  telle que $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Problème du tri

Pourquoi trier ?

- Tris inhérents aux problèmes à traiter (classement des données)
- Tris utilisés comme sous-routines vitales (affichage et profondeur)
- Intérêt historique (méthodes identiques pour d'autres algorithmes)
- Permet des comparaisons asymptotiques pour déterminer les complexités
- Liens entre tris et problèmes techniques (résolus algorithmiquement)

Tri par insertion

Description

Illustration de la manière dont les gens tiennent les cartes à jouer : choix des cartes une à une et placement en fonction des cartes déjà classées.

- Choisir le premier élément non trié
- L'insérer à sa place parmi les éléments triés

Temps d'execution : $c_1\ n^2$ pour trier n éléments
(c_1 : constante indépendante de n)

Tri par insertion

Algorithme

```
procedure triInsertion(ES
    tableau tab(N):entier, taille
    :entier)
variables
  i:entier
  j:entier
  cle:entier
  pour j←2 a taille pas 1
    cle ← tab(j)
    i ← j-1
    tant que i>0 et tab(i) > cle
          faire
      tab(i+1) ← tab(i)
      i ← i-1
    ftq
    tab(i+1) ← cle
  fpour
finprocedure
```

5	2	4	6	1	3
2	5	4	6	1	3
2	4	5	6	1	3
2	4	5	6	1	3
1	2	4	5	6	3
1	2	3	4	5	6

Tri par insertion

Invariant de boucle

```
procedure trInsertion(ES
      tableau tab(N):entier,
      taille:entier)
variables
  i:entier
  j:entier
  cle:entier
  pour j←2 a taille pas 1
    cle ← tab(j)
    i ← j−1
    tant que i>0 et tab(i) >
          cle faire
      tab(i+1) ← tab(i)
      i ← i−1
    ftq
    tab(i+1) ← cle
  fpour
finprocedure
```

Invariant de boucle : `{tab(1)<=...<=tab(j-1)} //` le tableau est trié entre les indices 1 et $j-1$

Preuve :

- Initialisation : Si $j = 2$ le sous tableau d'une seule case est trié.
- Conservation : l'insertion préserve l'ordre donc $tab(1) <= ... <= tab(j)$
- Terminaison : si $j = taille$ on obtient bien $tab(1) <= ... <= tab(taille)$

Tri par insertion

Coût de l'algorithme

```
procedure triInsertion(ES
      tableau tab(N):entier,
      taille:entier)
variables
  i:entier
  j:entier
  cle:entier
  pour j←2 a taille pas 1
        //2 instructions
    cle ← tab(j) //1
    i ← j−1 //2
    tant que i>0 et tab(i) >
          cle faire //2
      tab(i+1) ← tab(i) //2
      i ← i−1 //2
    ftq  //∑_{j=2}^{taille}(j − 1) fois
    tab(i+1) ← cle //2
  fpour //n − 1 fois
finprocedure
```

Coût de l'algorithme dans le cas le plus défavorable (n=taille) :

$$C(n) = n-1*7+(\sum_{j=2}^{n}(j-1))*6)$$
$$C(n) = 7*n-7+(n*(n-1)/2)*6)$$
$$C(n) = 7*n-7+3*n^2-3*n$$
$$C(n) = 3*n^2+4*n-7$$

Le temps d'exécution s'exprime sous la forme $an^2 + bn + c$, avec a, b, c constants. Il s'agit d'une fonction quadratique de n.

Notions de complexité algorithmique

Pourquoi

Comme nous venons de le voir sur les exemples précédents, il est souvent difficile d'évaluer exactement le coût d'un algorithme :

- constantes multiplicatives négligeables pour grandes données
- se contenter de déterminer l'ordre de grandeur
- notion de complexité asymptotique (à la limite)

Croissance de fonctions

Notation asymptotique

- Notation Θ
  - $\Theta(g(n)) = \{f(n)$ il existe des constantes positives c_1, c_2 et n_0 telles que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ pour tout $n \geq n_0\}$
  - On notera $f(n) = \Theta(g(n))$
  - $g(n)$ est une borne asymptotiquement approchée de $f(n)$
- Notation O : borne supérieure asymptotique
- Notation Ω : borne inférieure asymptotique

Fonctions classiques

Notations standard

- Monotonie de f : si $m \leq (resp. \geq) n$ alors $f(m) \leq (resp. \geq) f(n)$
- Parties entières : $\forall x \in \mathbb{R},\ x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$
- Congruences : $\forall a \in \mathbb{N}, n \in \mathbb{N}^+,\ a \mod n = a - \lfloor a/n \rfloor n$
  $a \equiv b \mod n$: a est congru à b
- Polynômes :

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

est un polynôme en n de degré d (d entier)

Propriétés des fonctions

Exponentielles

$$
\begin{aligned}
a^0 &= 1 \\
a^1 &= a \\
a^{-1} &= 1/a \\
(a^m)^n &= a^{mn} \\
(a^m)^n &= (a^n)^m \\
a^m a^n &= a^{m+n}
\end{aligned}
$$

Propriétés des fonctions

Logarithmes

$$
\begin{aligned}
a &= b^{log_b a} \\
\log_c(ab) &= \log_c a + \log_c b \\
\log_b a^n &= n \log_b a \\
\log_b a &= \frac{\log_c a}{\log_c b} \\
\log_b(1/a) &= -\log_b a \\
\log_b a &= \frac{1}{log_a b} \\
a^{\log_b c} &= c^{log_b a}
\end{aligned}
$$

Complexités usuelles

Par ordre croissant

- $\Theta(1)$: complexité constante
- $\Theta(log(n))$: complexité logarithmique (recherche dichotomique)
- $\Theta(n)$: complexité linéaire (recherche classique)
- $\Theta(n * log(n))$: (tris rapides)
- $\Theta(n^2)$: complexité quadratique (tris simples)
- $\Theta(n^k)$: complexité polynomiale (k constant)
  - Limite théorique des algorithmes utilisables
  - En pratique, inutilisable pour $k > 5$
- Complexité exponenetielles :
  - $\Theta(2^n)$
  - $\Theta(n^n)$

Inutilisables en pratique, utilisation de méthodes approchées (cf ING2) : problèmes d'emploi du temps, du voyageur de commerce, ...