Cours 3. Partie 1: Quelques complexités courantes. Diviser pour règner.

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

Aujourd'hui

Quelques complexités "courantes"
- L'exercice
- Temps linéaire
- Temps $O(n \log n)$
- Temps quadratique: $O(n^2)$
- Temps cubique: $O(n^3)$
- Temps polynomial: $O(n^k)$
- Au delà du temps polynomial

Diviser pour règner
- Le cas du tri fusion
- D'autres récurrences
- Exemple: Multiplication entière

Complexité du problème du tri

Plan

But de l'exercice

- Pour analyser un algorithme il peut être utile d'avoir une vision du "paysage" des complexités courantes pour les algorithmes:
  - $O(n)$, $O(n^2)$, $O(n \log n)$, $O(n^3)$, ...

Rappels

- Pour un processeur capable d'effectuer un million d'instructions élémentaires par seconde.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	4 s
$n = 30$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	18 m	10^{25} a
$n = 50$	< 1 s	< 1 s	< 1 s	< 1 s	11 m	36 a	∞
$n = 10^2$	< 1 s	< 1 s	< 1 s	1s	12.9 a	10^{17} a	∞
$n = 10^3$	< 1 s	< 1 s	1s	18 m	∞	∞	∞
$n = 10^4$	< 1 s	< 1 s	2 m	12 h	∞	∞	∞
$n = 10^5$	< 1 s	2 s	3 h	32 a	∞	∞	∞
$n = 10^6$	1s	20s	12 j	31710 a	∞	∞	∞

- Notations: ∞ = le temps dépasse 10^{25} années, s= seconde, m= minute, h = heure, a = an.

Plan

Temps linéaire

- Exemple 1: Calcul du maximum:

```
int max = T[1];
for (int j=2; j<= T.length; j++) {
      if (T[j] > max)
        max = T[j];}
```

Temps linéaire

- Exemple 2: Fusionner deux listes triées.

```
// maintenir une référence (un pointeur) sur chaque liste, initialisé
comme pointant sur la tête de chaque liste.

Tant que les deux listes ne sont pas vides
  soit a l'élement pointé par le pointeur sur la première liste
  soit b l'élement pointé par le pointeur sur la première liste
  ajouter min(a,b) à la liste en sortie
  avancer le pointeur de la liste qui contenait min(a,b) à l'élément
  suivant.
Fin tant que

Une fois que l'une des listes est vide, ajouter le reste de l'autre
liste à la sortie.
```

En JAVA: récursif

```
static Liste Fusion(Liste a, Liste b) {
  if (a == null)
    return b;
  if (b == null)
    return a;
  if (a.contenu < b.contenu)
    return new Liste(a.contenu, Fusion(a.suivant, b));
  else
    return new Liste(b.contenu, Fusion(a, b.suivant));
}
```

- Complexité: $O(longueur(a) + longueur(b))$.

Fusion: version itérative

Suppose que $T[p \ldots q]$ et $T[q+1 \ldots r]$ sont triés dans l'ordre croissant.

```
 int[] a = new int[M+1],
       b = new int[N+1],
       c = new int[M+N];
int i = 0, j = 0;

a[M+1] = b[N+1] = Integer.MAX_VALUE;

for (int k = 0; k < M+N; ++k)
         if (a[i] <= b[j]) {
           c[k] = a[i];
           ++i;
         } else {
           c[k] = b[j];
           ++j;
}
```

(on utilise deux sentinelles $a_{M+1} = \infty$ et $b_{N+1} = \infty$ pour ne pas compliquer la structure du programme).

Plan

Temps $O(n \log n)$

- Souvent obtenu par un algorithme du type "Diviser pour règner":
  - Diviser pour règner:
    - pour résoudre un problème de taille n, on résout deux problèmes de taille $n/2$, que l'on recombine.
  - Le $O(n \log n)$ vient du fait suivante:
    - Toute récurrence du type $C(n) \leq cn + 2C(n/2)$ donne une complexité en $O(n \log n)$

      (en supposant $C(2) \leq c$).

Exemple: tri fusion

- Une liste de 0 ou 1 élément est triée.
- Toute autre liste L
  - peut se découper en deux sous-listes L_1 et L_2 de même taille (à 1 élément près).
  - Les sous-listes L_1 et L_2 sont triées récursivement,
  - puis fusionnées.

$$TriFusion(L) = Fusion(TriFusion(L_1), TriFusion(L_2)).$$

Tri fusion: version récursive

```
static Liste MergeSort(Liste l) {
  Liste l1 = null,l2 = null;
  boolean even = true ;
  for ( ; l ≠ null ; l = l.suivant ) {
    if (even) {
      l1 = new Liste (l.contenu, l1) ;
    } else {
      l2 = new Liste (l.contenu, l2) ;
    }
    even = !even ;
  }
  if (l2==null) return l1;
  return Fusion(MergeSort(l1),MergeSort(l2));
  }
```

- Note: `L1` et `L2` sont en fait ici dans l'ordre inverse des éléments pairs et impairs, par commodité.
- Note: le tri fusion fait *toujours* de l'ordre de $n \log n$ comparaisons, et pas seulement au pire cas.

Version itérative

```
procedure TriFusion (g, d: integer);
    var i, j, k, m: integer;
    begin
    if g < d then
        begin
        m := (g + d) div 2;
        TriFusion (g, m);
        TriFusion (m + 1, d);
        for i := m downto g do b[i] := T[i];
        for j := m+1 to d do b[d+m+1-j] := T[j];
        i := g; j := d;
        for k := g to d do
            if b[i] < b[j] then
                begin T[k] := b[i]; i := i + 1 end
            else
                begin T[k] := b[j]; j := j - 1 end;
        end;
    end;
```

- La recopie pour faire l'interclassement se fait dans un tableau b de même taille que T.
- Il y a une petite astuce en recopiant une des deux moitiés dans l'ordre inverse, ce qui permet de se passer de sentinelles pour l'interclassement, puisque chaque moitié sert de sentinelle pour l'autre moitié.

Plan

Temps quadratique: $O(n^2)$

- Exemple: on se donne des points $(x_1, y_1), \ldots, (x_n, y_n)$ du plan, et on veut déterminer la distance entre les deux points les plus proches.

- Algorithme naïf:

```
min = +infini
pour chaque point (x_i,y_i)
  pour chaque autre point (x_j,y_j)
    calculer d= racine ( (x_i-x_j) ^2 + (y_i-y_j)^2)
    si d<min alors min=d
```

- Peut se faire aussi en $O(n \log n)$ (diviser pour règner).

Plan

Temps cubique: $O(n^3)$

- Exemple: on se donne des ensembles $S_1, S_2, \ldots, S_n$, chaque $S_i \subset \{1, 2, \ldots, n\}$ et on veut savoir si l'une des paires (S_i, S_j) n'a pas d'élement en commun.

```
Pour chaque ensemble S_i
 Pour chaque ensemble S_j
  Pour chaque element p de S_i
   Déterminer si p est dans S_j
  Fin pour
  Si aucun élement de S_i est dans S_j
   Retourner que S_i et S_j sont disjoints
 Fin pour
Fin pour
```

Plan

Temps cubique: $O(n^k)$

- On se donne un graphe G a n sommets, et on veut savoir s'il possède un "ensemble indépendant" S de taille k:
  - c'est-à-dire un sous-ensemble S de sommets tels qu'aucune arête de G ne possède ses deux extrémités dans S.

```
Pour chaque sous-ensemble S de k sommets
  Vérifier si S est un ensemble indépendant
  Si S est un ensemble indépendant
    s'arrêter et déclarer un succès
  Fin si
Fin pour
Déclarer un échec.
```

- Le nombre de parties à k éléments parmi n est en $O(n^k/k!) = O(n^k)$ si k est considéré comme constant.

Plan

Au dela du temps polynomial

- On se donne un graphe G a n sommets, et on veut déterminer la taille du plus grand ensemble indépendant.

```
Pour chaque sous-ensemble S de sommets
  Vérifier si S est un ensemble indépendant
  Si S est un ensemble indépendant
    plus grand que ceux vus jusque là
      alors mémoriser la taille de S
  Fin si
Fin pour
```

- Le nombre total de parties de $\{1, 2, \ldots, n\}$ est 2^n. Chaque itération fait $O(n^2)$ opérations. Au final $O(2^n n^2)$.

- Aucun algorithme polynomial connu à ce jour.

Aujourd'hui

Quelques complexités "courantes"
- L'exercice
- Temps linéaire
- Temps $O(n \log n)$
- Temps quadratique: $O(n^2)$
- Temps cubique: $O(n^3)$
- Temps polynomial: $O(n^k)$
- Au delà du temps polynomial

Diviser pour règner
- Le cas du tri fusion
- D'autres récurrences
- Exemple: Multiplication entière

Complexité du problème du tri

Plan

Récurrence $T(n) \leq 2T(n/2) + cn$, $T(2) \leq c$.

Plusieurs méthodes pour analyser cette récurrence:

1. "Dérouler la récurrence":
2. "Par substitution d'une solution":

Méthode: Dérouler la récurrence

- Dessin des appels récursifs: au pire cas:
  - au niveau 0: 1 sommet, cn.
  - au niveau 1: 2 sommets, $cn/2 + cn/2 = cn$ au total.
  - au niveau 2: 4 sommets, $4cn/4 = cn$ au total.
  - au niveau k, 2^k sommets, $2^k cn/2^k = cn$ au total.
- Chaque niveau de récursion contient au plus cn opérations.
- Il y a au plus $\log_2 n$ niveaux.

Méthode: Par substitution d'une solution

- On prouve par récurrence que $T(n) \leq cn \log_2(n)$.
- Hypothèse de récurrence:
  - HR_n: $T(m) \leq cm \log_2 m$, pour tout $m \leq n$.
- On a bien HR_2 car $cn \log_2 n = 2c$ et $T(2) \leq c$.
- Pour prouver HR_n à partir de HR_{n-1}, on écrit

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \\ &\leq 2c(n/2)\log_2(n/2) + cn \\ &= cn[(\log_2 n) - 1] + cn \\ &= (cn \log_2 n) - cn + cn \\ &= c_n \log_2 n \end{aligned}$$

Plan

Le cas $T(n) \leq qT(n/2) + cn$ $T(2) \leq c$, $q > 2$.

- Dessin des appels récursifs: au pire cas:
  - au niveau 0: 1 sommet, cn au total.
  - au niveau 1: q sommets, $cn/2 + \cdots + cn/2 = q/2cn$ au total.
  - au niveau 2: q^2 sommets, $q^2 cn/4$ au total.
  - au niveau k, q^k sommets, $q^k cn/2^k$ au total.
- Il y a au plus $\log_2 n$ niveaux.
- $T(n) \leq \sum_{k=0}^{\log_2 n - 1} (q/2)^k cn = cn \sum_{k=0}^{\log_2 n - 1} (q/2)^k \leq cn \frac{r^{\log_2 n}}{r-1}$
  en majorant la suite géométrique $1 + r + r^2 + \cdots + r^{\log_2 n - 1}$ par $\frac{r^{\log_2 n}}{r-1}$.
  On obtient $T(n) \leq \frac{c}{r-1} n r^{\log_2 n}$.
  Or $r^{\log_2 n} = n^{\log_2 r} = n^{\log_2 (q/2)} = n^{\log_2 q - 1}$.
  Ce qui donne

$$T(n) \leq \frac{c}{r-1} n^{\log_2 q} = O(n^{\log_2 q}).$$

Le cas $q = 1$

Donne $O(n)$!!

Plan

Exemple: Multiplication entière

- On veut multiplier deux entiers: exemple 12 * 13 en décimal, 1100 * 1101 en binaire.
  - si on compte pour unitaire une multiplication chiffre par chiffre, cela fait $O(n)$ opérations pour chaque produit partiel, et $O(n)$ pour combiner avec la somme des produits partiels jusque là.
  - Au total $O(n^2)$.
- Peut-on faire mieux?
- Diviser pour règner:
  - $x = x_1 2^{n/2} + x_0$, $y = y_1 * 2^{n/2} + y_0$.
  - On écrit:

$$xy = x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0.$$

  - On réduit le problème sur n bits à 4 problèmes sur $n/2$ bits (pour calculer $x_1 y_1$, $x_1 y_0$, $x_0 y_1$, $x_0 y_0$).
  - Récurrence: $T(n) \leq 4T(n/2) + cn$.
  - On obtient: $T(n) \leq O(n^{\log_2 4}) = O(n^2)$: aucune amélioration !!

Astuce

- Le produit $(x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0$ permet d'obtenir les 4 produits avec une unique multiplication récursive.
- si on détermine x_1y_1 et x_0y_0 par récurrence, on obtient ces termes extrêmes explicitement, et ceux du milieu en les soustrayant à $(x_1 + x_0)(y_1 + y_0)$.
- Bref:

```
Recursive-multiply(x,y)
  Write x= x_1 2^{n/2} + x_0
  Write y = y_1 2^{n/2} + y_0
  Calculer x_1 + x_0
  Calculer y_1 + y_0
  p = Recursive-multiply(x_1+x_0,y_1+y_0)
  x_1y_1 = Recursive-multiply(x_1,y_1)
  x_0y_0 = Recursive-multiply(x_0,y_0)
  Return x_1y_1 2^n + (p-x_1y_1-x_0y_0).2^{n/2} + x_0y_0
```

- (en ignorant que $x_1 + x_0$ et $y_1 + y_0$ peuvent avoir $n + 1$ bits au lieu de n) .

$$T(n) \leq 3T(n/2) + cn.$$

- Soit

$$T(n) \leq n^{\log_2 3} = n^{1.59}.$$

Aujourd'hui

Quelques complexités "courantes"
- L'exercice
- Temps linéaire
- Temps $O(n \log n)$
- Temps quadratique: $O(n^2)$
- Temps cubique: $O(n^3)$
- Temps polynomial: $O(n^k)$
- Au delà du temps polynomial

Diviser pour règner
- Le cas du tri fusion
- D'autres récurrences
- Exemple: Multiplication entière

Complexité du problème du tri

Complexité du problème du tri

- Tout algorithme de tri
  - qui fonctionne par comparaisons,
  - qui n'a pas d'autres informations sur les données,

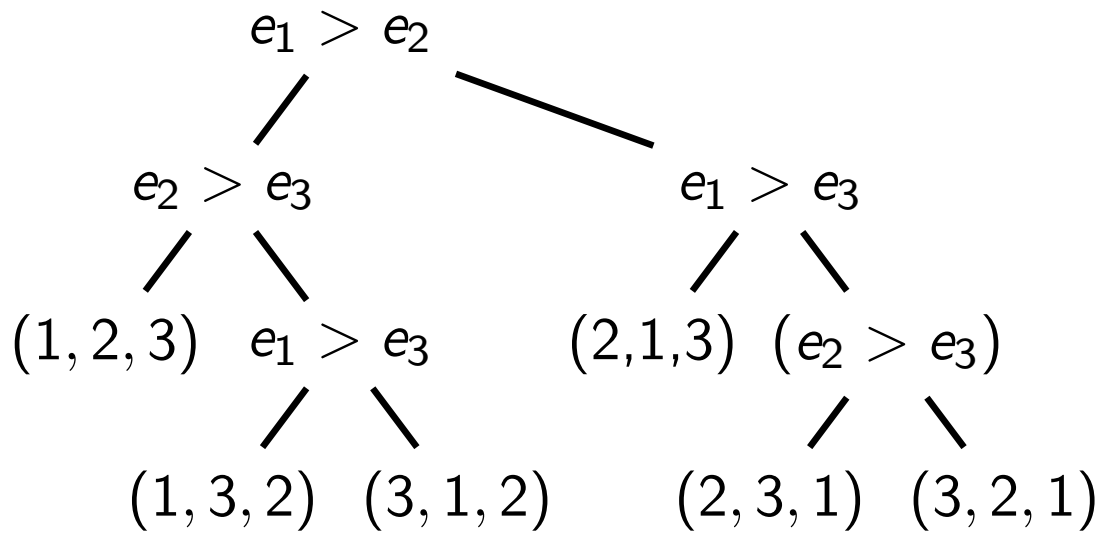
  effectue $\Omega(n \log n)$ comparaisons dans le pire des cas.

- Autrement dit, la complexité du problème du tri (avec ces hypothèses) est en $\Theta(n \log n)$.

- Le tri par fusion est donc parmi les tris optimaux en $O(n \log n)$.

Preuve de la borne inférieure $\Omega(n \log n)$: 1/2

- A un algorithme on peut associer un arbre de décision:



  - En chaque noeud qui n'est pas une feuille: une comparaison effectuée par l'algorithme.
    - La racine est la première comparaison.
    - Fils gauche: récursivement ce qui se passe alors pour un résultat positif.
    - Fils droit: récursivement ce qui se passe alors pour un résultat négatif.
  - En chaque feuille, le résultat produit.

Preuve de la borne inférieure $\Omega(n \log n)$: 2/2

- Un arbre binaire de hauteur h a au plus 2^h feuilles.
  - voir cours sur les arbres (ou se prouve par récurrence).
- Les feuilles doivent contenir au moins les $n!$ permutations, et donc $h \geq \log(n!) = \Omega(n \log n)$ par la formule de Stirling.