Complexité (Quatrième année)

TD4

Maud MARCHAL

5-6 avril 2011

Plan du cours de Complexité

- Concevoir un algorithme : trois paradigmes :
  - Diviser pour Résoudre
  - Programmation dynamique
  - Algorithmes gloutons
- Classes de complexité
- Problèmes d'optimisation :
  - Recherche basée sur l'exploration des arbres
  - Heuristiques et métaheuristiques

Algorithmes récursifs et programmation "Diviser pour résoudre"

5-6 avril 2011

Plan

1. Généralités
   - Stratégie "Diviser pour Résoudre"
   - Complexité des algorithmes "Diviser pour Résoudre"
     - Cas où la résolution d'un seul sous-problème suffit
     - Cas général : division des données puis regroupement
2. Exemples classiques
   - Multiplication de grands entiers
   - Multiplication des polynômes
   - Multiplication de matrices
3. Algorithmes d'enveloppe convexe
   - Définitions
   - Algorithme de l'enveloppe rapide
   - Algorithme de Jarvis
   - Scan de Graham

Plan

1. Généralités
   - Stratégie "Diviser pour Résoudre"
   - Complexité des algorithmes "Diviser pour Résoudre"
     - Cas où la résolution d'un seul sous-problème suffit
     - Cas général : division des données puis regroupement
2. Exemples classiques
3. Algorithmes d'enveloppe convexe

Description de la stratégie "Diviser pour Résoudre"

- **Stratégie "Diviser pour Résoudre"** (Divide and Conquer) : elle consiste à diviser un problème de taille n en sous-problèmes plus petits, dont la résolution permet de construire la solution du problème entier.
- Les différentes étapes :
  - Diviser le problème en un certain nombre de sous-problèmes ;
  - Résoudre récursivement les sous-problèmes ;
  - Combiner les solutions des sous-problèmes.
- Exemples : recherche dichotomique, tri fusion, tri rapide.
- Stratégie très répandue pour l'élaboration d'algorithmes, qui conduit le plus souvent à l'écriture d'algorithmes compacts et efficaces.

Applications de la stratégie "Diviser pour Résoudre"

Il y a 2 façons d'appliquer la stratégie "Diviser pour Résoudre" :

- Récursivité sur les données (ex. : tri fusion) :
  1. Séparer les données en plusieurs sous-ensembles ($O(1)$) ;
  2. Résoudre récursivement les sous-problèmes ;
  3. Effectuer un travail pour combiner les résultats ($O(n)$).
- Récursivité sur les résultats (ex : tri rapide) :
  1. Pré-traitement pour trouver le bon découpage des données ($O(n)$) ;
  2. Résoudre récursivement les sous-problèmes ;
  3. Les sous-résultats se combinent d'eux-mêmes, grâce à l'effort fait lors du découpage des données.

Premier exemple : Tri par fusion

- **Principe** : Diviser en deux chaque sous-élément du tableau, trier les sous-éléments puis les fusionner.
- Algorithme :

```
void TriFusion(int A[], int inf, int sup){
  int i;
  if (inf < sup){
    i = (inf + sup)/2;
    TriFusion(A,inf,i);
    TriFusion(A,i+1,sup);
    Fusionner(A,inf,i,sup);
  }
}
```


Premier exemple : Tri par fusion

- Exemple : A=[5,2,4,6,1,3,3,6]
- Questions :
  1. Arbre d'appel
  2. Evolution du tableau

Complexité du tri par fusion

Complexité des algorithmes "Diviser pour Résoudre"

- **Deux cas** :
  1. Cas où la résolution d'un seul sous-problème suffit ;
  2. Cas général : division des données puis regroupement.

- n : taille du problème
  T(n) : coût en nombre d'opérations fondamentales pour résoudre un problème de taille n.

Cas où la résolution d'un seul sous-problème suffit

Théorème

Si $T(n) = T(\frac{n}{2}) + g(n)$, avec $T(1) = C$, alors :

$$T(n) = C + \sum_{1}^{E(log_2(n))} g(2^i)$$

- Exemple : Recherche dichotomique :
  En deux comparaisons, on sait dans quel sous-tableau effectuer les recherches.

$$\begin{aligned} T(n) &= T(\frac{n}{2}) + 2 \\ &= \sum_{1}^{log_2(n)} 2 \\ &= O(log_2(n)) \end{aligned}$$

Démonstration du théorème

On suppose $n = 2^k$ (sinon on encadre n)
Par récurrence :

$$\begin{aligned} T(2n) &= T(n) + g(2n) \\ & \quad \text{avec } n = 2^k \Rightarrow 2n = 2^{k+1} \\ &= \sum_{i=1}^{E(log_2(n))} g(2^i) + c + g(2n) \\ &= \sum_{i=1}^{k} g(2^i) + c + g(2^{k+1}) \\ &= \sum_{i=1}^{k+1} g(2^i) + c \end{aligned}$$

Cas général : division des données puis regroupement

- **Cas général** : division des données en b morceaux puis regroupement.

- Exemple : tri fusion
  Division en 2 sous-problèmes de taille n/2

- Equilibrage des sous-problèmes :
  - Les algorithmes DPR sont d'autant plus efficaces que les sous-ensembles construits sur les données sont de même taille.
  - Exemple : tri par insertion vs tri par fusion.

Théorème dans le cas général

Le théorème s'applique à tous les algorithmes pour lesquels on divise les données en sous-ensembles de taille égale, et où le temps nécessaire au regroupement des résultats à chaque étape est proportionnel à n.

Théorème

Si $T(n) = aT(\frac{n}{b}) + cn$, avec $T(1) = C$, $a > 1$, $b > 1$ alors :

$$\begin{aligned} a < b &\Rightarrow T(n) = O(n) \\ a = b &\Rightarrow T(n) = O(nlog(n)) \\ a > b &\Rightarrow T(n) = O(n^{\log_b(a)}) \end{aligned}$$

Démonstration du théorème

$$\begin{aligned} T(n) &= aT(\frac{n}{b}) + c.n \\ aT(\frac{n}{b}) &= a^2T(\frac{n}{b^2}) + ac.\frac{n}{b} \\ \ldots &= \ldots \\ a^iT(\frac{n}{b^i}) &= a^{i+1}T(\frac{n}{b^{i+1}}) + a^ic.\frac{n}{b^i} \\ \ldots &= \ldots \\ a^{\log_b(n)}T(1) &= a^{\log_b(n)}.C \end{aligned}$$

On peut donc éliminer les termes en T à droite :

$$T(n) = cn\sum_{i=0}^{\log_b(n)-1}\left(\frac{a}{b}\right)^i + a^{\log_b(n)}.C$$

Démonstration du théorème (2)

- si $a < b$:
  la série $\sum_{i=0}^{\infty}\left(\frac{a}{b}\right)^i$ converge :
  $$\sum_{i=0}^{\log_b(n)-1}\left(\frac{a}{b}\right)^i < \sum_{i=0}^{\infty}\left(\frac{a}{b}\right)^i = \text{constante}$$
  Le premier terme de $T(n)$ est donc en $O(n)$. De plus, $a^{\log_b(n)} = n^{\log_b(a)}$ donc le second terme est négligeable.
- si $a = b$:
  Le premier terme est en $nlog(n)$ et le second en $O(n)$ est négligeable.
- si $a > b$:
  $$\sum_{i=0}^{\log_b(n)-1}\left(\frac{a}{b}\right)^i = \frac{\left(\frac{a}{b}\right)^{\log_b(n)} - 1}{\frac{a}{b} - 1} \approx \left(\frac{a}{b}\right)^{\log_b(n)} = \frac{a^{\log_b(n)}}{n}$$
  Les deux termes sont donc de même ordre :
  $$T(n) \approx a^{\log_b(n)} = n^{\log_b(a)}$$

Exemple : tri fusion

$T(n) = 2T(n/2) + n$

donc $a = b = 2 \Rightarrow T(n) = n\log(n)$

Plan

Généralités

2 Exemples classiques
- Multiplication de grands entiers
- Multiplication des polynômes
- Multiplication de matrices

Algorithmes d'enveloppe convexe

Description

- **Objectif** :
  Il s'agit de multiplier 2 entiers X et Y qui ont n chiffres en base 2.
- Méthode habituelle : On fabrique n produits partiels de taille n.
  Coût : $O(n^2)$ (nombre d'additions et de multiplications).
- Approche Diviser pour Résoudre :
  Elle consiste à scinder X et Y en deux entiers de $n/2$ chiffres.
  En supposant que n est une puissance de 2, on a :

Approche Diviser pour Résoudre

$T(1) = 1$

$T(n) = 4T(\frac{n}{2}) +$

Amélioration : diminution du nombre de sous-problèmes

- On peut reformuler l'équation de manière à **diminuer le nombre de sous-problèmes**, c'est à dire le nombre de multiplications entre entiers à $n/2$ chiffres :

Multiplication des polynômes : Description

- **Objectif** : effectuer la multiplication de 2 polynômes de degrés $n-1$ avec une complexité inférieure à n^2.

  Notations :
  P, Q $\in \mathcal{P}_{n-1}$:

$$\begin{aligned} P(X) &= p_0 + p_1X + \cdots + p_{n-1}X^{n-1} \\ Q(X) &= q_0 + q_1X + \cdots + q_{n-1}X^{n-1} \end{aligned}$$

- Exercice : proposer un algorithme analogue à la multiplication de grands entiers pour réaliser la multiplication de polynômes.

Multiplication des polynômes : Solution

$$\begin{aligned} P(X) &= X^4 + 3X^3 + X^2 - X + 1 \\ &= \underbrace{X^2(X^2+3X+1)}_{P_1(X)} \underbrace{-X+1}_{P_2(X)} \\ &= X^{\frac{n}{2}} P_1 + P_2(X) \end{aligned}$$

On a
$P(X) = P_g(X) + X^{\frac{n}{2}} P_d(X)$
$Q(X) = Q_g(X) + X^{\frac{n}{2}} Q_d(X)$

On reformule le produit à ce qu'il n'y ait que 3 multiplications de polynome de taille n/2.

Multiplication des polynômes : Remarque

- Il existe une méthode encore plus intéressante : la méthode de Schönhage et Strassen (1971) qui permet de multiplier des grands nombres en un temps $O(nlog(n))$.
- Pour cela, on remarque qu'un polynôme de degré n est entièrement défini par sa valeur en $n+1$ points.
  1. On utilise le schéma de Horner pour évaluer la valeur des deux polynômes pour les racines $2n-1$ième de l'unité.
  2. On multiplie ces valeurs.
  3. On applique la FFT (Fast Fourier Transform) pour calculer les coefficients du nouveau polynôme. L'algorithme de la FFT est de type Diviser pour Résoudre et son coût est $O(nlog(n))$.

Multiplication de matrices : Description

- **Objectif** : multiplication de grosses matrices.
- Jusqu'en 1968, on pensait qu'il était impossible de réaliser cette opération en moins de $O(n^3)$ multiplications (n multiplications pour chacun des n^2 termes).
  Strassen a proposé un algorithme de type diviser pour résoudre pour améliorer la complexité.
- Notations :
  Supposons qu'il s'agit de multiplier 2 matrices carrées de taille n avec $n = 2^k$.

Multiplication de matrices : Stratégie Diviser pour Résoudre

- Stratégie Diviser pour Résoudre : couper la matrices en 4 sous-matrices carrées :

$$P = M.N = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \cdot \begin{pmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{pmatrix}$$

- Si on multiplie simplement les sous-matrices, on a un coût en nombre de multiplications :

$$T(n) = 8T(\frac{n}{2}) = O(n^3)$$

Multiplication de matrices : Algorithme de Strassen

- Strassen a proposé de remplacer cette multiplication directe de sous-matrices par les calculs suivants :

$$\begin{aligned} P_{11} &= X_1 + X_4 - X_5 + X_7 \\ P_{12} &= X_3 + X_5 \\ P_{21} &= X_2 + X_4 \\ P_{22} &= X_1 + X_3 - X_2 + X_6 \end{aligned}$$

avec :

$$\begin{aligned} X_1 &= (M_{11} + M_{22})(N_{11} + N_{22}) \\ X_2 &= (M_{21} + M_{22})N_{11} \\ X_3 &= M_{11}(N_{12} - N_{22}) \\ X_4 &= M_{22}(N_{21} - N_{11}) \\ X_5 &= (M_{11} + M_{12})N_{22} \\ X_6 &= (M_{21} - M_{11})(N_{11} + N_{12}) \\ X_7 &= (M_{12} - M_{22})(N_{21} + N_{22}) \end{aligned}$$

Algorithme de Strassen : Calcul du coût

- On évalue le coût en terme d'opérations :
  on n'a plus que 7 multiplications de sous-matrices de taille $n/2$ et 18 additions au lieu de 8 multiplications et 4 additions.
- L'algorithme est intéressant si le temps d'une multiplication est beaucoup plus grand que celui d'une addition.
- Calcul du coût des multiplications :
  $T(n) = 7T(\frac{n}{2}) = O(n^{\log_2(7)}) = O(n^{2.81})$.

Algorithme de Strassen : Calcul du coût des additions

- Calcul des additions (on pose $n = 2^k$) :

$$\begin{aligned} a_k &= 7a_{k-1} + 18(n/2)^2 \\ &= 7a_{k-1} + \frac{9}{2}.4^k \end{aligned}$$

On pose alors : $a_k = 7^k y_k$

$$\begin{aligned} y_k &= y_{k-1} + \frac{9}{2}.(\frac{4}{7})^k \\ &= \frac{9}{2}\sum_{i=1}^{k}(\frac{4}{7}^i) \\ &\leq \frac{9}{2}\sum_{i=1}^{\infty}(\frac{4}{7}^i) = \frac{9}{2}.\frac{1}{1-4/7} = \frac{21}{2} \\ y_k &\leq \frac{21}{2} \end{aligned}$$

et on trouve donc : $n_{add} = O(7^k) = O(n^{2.81})$

Algorithme de Strassen : Conclusion

- Le produit de matrices (et la résolution d'un système linéaire) peut être fait en $O(n^{2.81})$ opérations.
- On pense que la valeur optimale de l'exposant est $2 + \epsilon$. De toute façon, la complexité minimale est en $O(n^2)$ car il faut pouvoir traiter $2n^2$ coefficients.
- Ce résultat présente surtout un intérêt théorique car on a négligé les coûts des opérations secondaires. Les constantes sont très importantes du fait des nombreuses additions et soustractions supplémentaires : le gain en exposant est si faible que l'algorithme est intéressant seulement si $n > 1000$.

Résolution des systèmes linéaires

- Les résultats précédents supposaient que le temps d'une muliplication est constant, ce qui n'est pas le cas. Le temps d'une multiplication dépend de la taille des nombres manipulés.
  Nous avons alors les résultats suivants :
  - Gauss en flottant : $O(n^3)$
  - Gauss en précision parfaite : $O(n^5 log^2(n))$
  - Méthode modulaire : $O(n^4 log(n))$
  - Méthode P-Adique : $O(n^3 log^2(n))$

Plan

Définition

- **Définition** : L'enveloppe convexe d'un ensemble de n points du plan est le plus petit polygone convexe qui les contient.
- Entrée : n points du plan, $E = \{P_1 \cdots P_n\} \in \mathbb{R}^2$.
  Sortie : $Conv(E) = [P_{1k} \cdots P_{ik}]$ séquence de k points qui définissent l'enveloppe convexe.

Propriétés

Propriétés de l'enveloppe convexe :

1. Les sommets de l'enveloppe convexe se trouvent parmi les n points.
2. Pour que $[P_iP_j]$ soit un segment de l'enveloppe convexe, il faut que tous les autres points de E soient du même côté de la droite (P_iP_j).
3. Soit Δ une droite quelconque qui partage le plan en deux parties. Dans chaque demi-plan, le point parmi l'ensemble E qui est le plus éloigné de la droite Δ est un sommet de l'enveloppe convexe.

Calcul de l'enveloppe convexe

- Calculer l'enveloppe convexe consiste à trouver les points qui forment son contour dans l'ordre.
- Algorithme naïf : Prendre toutes les droites possibles et tester pour chacune s'il y a des points que d'un seul côté : on a alors une arête de l'enveloppe convexe.
- Autres méthodes :
  - Stratégie Diviser pour Résoudre ;
  - Algorithme itératif.

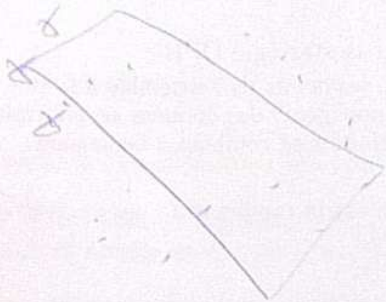
Algorithme de l'enveloppe rapide

- Deux manières d'appliquer la stratégie DPR :
  1. Diviser les données (i.e. les points de l'ensemble E)
  2. Effectuer un partage "intelligent" des données en pré-traitement de manière à ce que les résultats se combinent facilement.
- Question : quelle stratégie est la meilleure ? par résultat

Algorithme de l'enveloppe rapide

- **Algorithme** :
  1. Pour construire deux moitiés du résultat, on a besoin de connaître 2 sommets P et Q de l'enveloppe convexe : on prend les points d'abscisse minimale et maximale.
  2. La droite (PQ) partage les données E en deux parties E' et E'' contenant toutes deux P et Q. L'enveloppe convexe de chacune de ces parties est un polygone contenant l'arête $[PQ]$.
     Le résultat cherché est la concaténation de ces 2 résultats partiels, dont on retire l'arête $[PQ]$.
  3. Etape suivante : construction de l'enveloppe convexe des 2 sous-ensembles.
     En utilisant les propriétés de l'enveloppe convexe, on choisit le point S qui est le plus éloigné de la droite (PQ).
     On se sert alors des ensembles E_1 et E_2 pour continuer le calcul.

Algorithme de l'enveloppe rapide : illustration graphique



Algorithme de l'enveloppe rapide : pseudo-algorithme

Algorithme de l'enveloppe rapide : complexité

Algorithme de Jarvis

- Principe de l'algorithme de Jarvis (ou du paquet cadeau) : à partir d'un premier sommet P_1 (par exemple le point d'abscisse minimale) on trouve successivement les autres sommets de l'enveloppe en tournant autour des données.
- Plus précisément, si on a trouvé les sommets jusqu'à P_i, le point P_{i+1} est tel que la droite $(P_i P_{i+1})$ laisse tous les autres points du même côté.
- Illustration graphique :

Algorithme de Jarvis : complexité

- A chaque étape, le nouveau sommet est calculé en $O(n)$ (calcul de l'angle minimal).
- Le nombre d'étapes est égal au nombre de sommets de l'enveloppe convexe. Dans le cas le pire, on a n sommets : coût en $O(n^2)$.
- Complexité dans 3 cas typiques :
  - Distribution uniforme des points dans un polygone convexe : le nombre moyen de points de l'enveloppe convexe est $O(log(n))$. Le coût est alors en $O(nlog(n))$ comme l'enveloppe rapide.
  - Distribution uniforme des points dans un cercle : le nombre moyen de points de l'enveloppe convexe est $O(n^{\frac{1}{3}})$. Coût moyen : $O(n^{\frac{4}{3}})$, plus élevé que l'enveloppe rapide.
  - Distribution gaussienne : coût en $O(n\sqrt{log(n)})$.
- **Conclusion** : le principe DPR n'est pas toujours la meilleure solution.

Scan de Graham

- Algorithme :
  1. On numérote les points par balayage de droite à gauche par rapport à l'angle polaire.
  2. Le premier et le dernier point appartiennent à l'enveloppe convexe.
  3. Tant qu'on fait des "virages" à gauche, on continue. Si virage à droite, on rebrousse chemin.
- Illustration graphique :
- Complexité : faire le tri : $O(nlog(n))$.