Conception d’algorithmes

Diviser pour régner

V. Chevrier

Diviser pour régner

- Divide and conquer
- À chaque étape,
  - on décompose le problème en sous problèmes similaires au problème d'origine
  - On résout chaque sous problème indépendamment
  - On combine les sous-solutions pour construire la solution globale

Exemple 1

- Soit T un tableau trié de E
- Soit x : E, trouver l'indice de x dans T

- Alors ?
- Complexité ?

  Théorème maitre

Exemple 2

- Nombre de fibinnacci
  - $f_0=0$
  - $f_1=1$
  - $f_n=f_{n-1}+f_{n-2}$
  - Complexité ?

Fibonacci

- Solution divide-and-conquer

- Pour n > 1 on a:

$$F_{2n-1} = (F_n)^2 + (F_{n-1})^2$$
$$F_{2n} = (F_n)^2 + 2F_{n-1}F_n$$

- On a donc

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ (F_{\lceil n/2 \rceil})^2 + (F_{\lceil n/2 \rceil - 1})^2 & n >= 2 \quad et\ impair \\ (F_{\lceil n/2 \rceil})^2 + 2F_{\lceil n/2 \rceil}F_{\lceil n/2 \rceil - 1} & n >= 2\ et\ pair \end{cases}$$

Exemple : maximum dans un tableau

- Methode Maxtab(T: tableau d'element;
  left,right: indice): element

```
Si left = right alors
    retourner T[left]
Sinon
    m←(left+right)/2
    u ←Maxtab(T,left,m)
    v ← Maxtab(T,m+1,right)
    retouner max (u,v)
fsi
```

Exemple: tri par fusion

- Divide = ?
- Conquer

Principe

- Diviser
  - Si solution triviale alors solution directe
    sinon diviser le problème $S(n)$ en k sous problèmes $S(n_i)$ de taille $n_i < n$ ($i = 1..k$)
- Conquérir /Résoudre (récursivement)
  - $S(n_i)$ pour $i = 1..k$
- Combiner : fusionner les solutions

- Complexité:
  - Théorème maitre

Théorème maitre

Enoncé

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

- a >= 1, b> 1
- On divise un problème de taille ***n*** en a sous problèmes chacun de taille ***n/b***, le coût de diviser et combiner le problème est de ***f(n) (O(n^d))***

$$T(n) = \begin{cases} O\left(n^{d}\right) & si\ d > \log_b a \\ O\left(n^{d}\log_b n\right) & si\ d = \log_b a \\ O\left(n^{\log_b a}\right) & si\ d < \log_b a \end{cases}$$

- n/b peut être soit l'entier inférieur soit supérieur

Tri par fusion

```
static int[] triFusion(int[] t){
   if(t.length == 1) return t;
   int m = t.length / 2;
   int[] tg = sousTableau(t, 0, m);
   int[] td = sousTableau(t, m, t.length);
// on trie les deux moitiés
   tg = triFusion(tg);
   td = triFusion(td);
 // on fusionne
   return fusionner(tg, td);
}
```

```
// on crée un tableau contenant t[g..d[
  static int[] sousTableau(int[] t, int g, int d){
     int[] s = new int[d-g];

     for(int i = g; i < d; i++)
        s[i-g] = t[i];
     return s;
  }
```

Mise en oeuvre

- Tri par fusion
  - $T(n) = 2T(n/2) + O(n^d)$
  - $a=2$, $b=2$
  - $d = 1 = \log_b a = \log_2 2$
  - Donc $O(n\log_2 n)$

Mise en oeuvre

- Recherche dichotomique
  - $T(n)=1/T(n/2)+O(1)$
  - 1 sous problème
  - Sous problème de taille n/2
  - Diviser et combiner constant
  - Etc
  - $0 = \log_2 1$ donc $k = 0$
  - Donc $O(\log_2 n)$

Fibonnaci

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ (F_{\lceil n/2 \rceil})^2 + (F_{\lceil n/2 \rceil - 1})^2 & n >= 2 \quad et\ impair \\ (F_{\lceil n/2 \rceil})^2 + 2F_{\lceil n/2 \rceil}F_{\lceil n/2 \rceil - 1} & n >= 2\ et\ pair \end{cases}$$

Fibonnaci

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ (F_{\lceil n/2 \rceil})^2 + (F_{\lceil n/2 \rceil - 1})^2 & n >= 2 \quad et\ impair \\ (F_{\lceil n/2 \rceil})^2 + 2F_{\lceil n/2 \rceil}F_{\lceil n/2 \rceil - 1} & n >= 2\ et\ pair \end{cases}$$

- Complexité
  - T(n) = 2T(n/2)+ O(1)
  - d =0 < $\log_2(2)$
  - Cas 3
  - Donc O(n)

Application divide and conqueer

- Hauteur d'un arbre
- Nombre de feuille d'un arbre

ALGORITHM *Height(T)*

//Computes recursively the height of a binary tree
//Input: A binary tree T
//Output: The height of T
if $T = \varnothing$ **return** -1
else return $\max\{Height(T_{left}), Height(T_{right})\} + 1$

Pur aller plus loin

- Cormen et ali p62
- https://fr.wikipedia.org/wiki/Master_theorem

Exemple

- Produit de matrice

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Et

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Approche

- Algorithme naïf:
  - $O(n^3)$
- Diviser pour régner (V1)
  - On considère les matrices nxn comme une matrice 2x2 de matrice (n/2)x (n/2)

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} . \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

  - Avec $r = e.a + b.g;\ s = af+bh;\ t = ce+dg;\ u = cf+dh$

    Soit 8 produits récursifs de matrices n/2 x n/2

    4 additions de matrices n/2 x n/2
- Complexité ? (théorème maitre)

Complexité

- Théorème maitre
  - T(n) = 8 T(n/2) + O(n²)

Additions

8 produits de matrices n/2

$$T(n) = aT\left(\frac{n}{b}\right) + O\left(n^d\right)$$

$$T(n) = \begin{cases} O\left(n^d\right) & si\, d > \log_b a \\ O\left(n^d \log_b n\right) & si\, d = \log_b a \\ O\left(n^{\log_b a}\right) & si\, d < \log_b a \end{cases}$$

  - Soit d = 2, a = 8 et b =2
    - $\log_2 8 = 3 > 2$ (cas 3) => $O(n^3)$
- Donc ?

Algorithme de Strassen

- N'utiliser que 7 multiplications de matrices
- P1= a.(f-h)
- P2 = (a+b).h
- P3 = (c+d).e
- P4 = d.(g-e)
- P5=(a+d).(e+h)
- P6 = (b-d).(g+h)
- P7=(a-c).(e+f)

r = P5 + P4-P2+P6
s= P1+P2
t= P3+P4
u= P5+P1-P3-P7

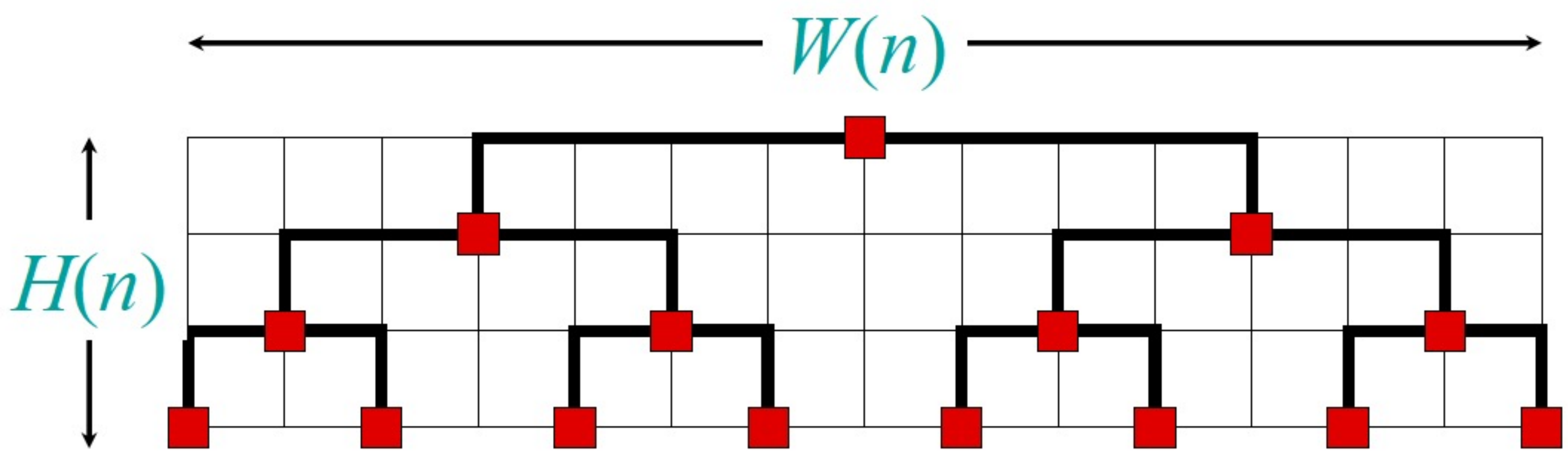
Complexité

- $T(N) = 7T(N/2) + O(N2)$
- Soit $T(N) = O(n^{\lg 7}) = O(n^{2.81})$

- Comme c'est un exposant, la différence est sensible ..
- Attention au calcul sur réel et précision

Représentation d'un arbre

- Soit une grille h x l
- Comment représenter un arbre à n feuilles en minimisant

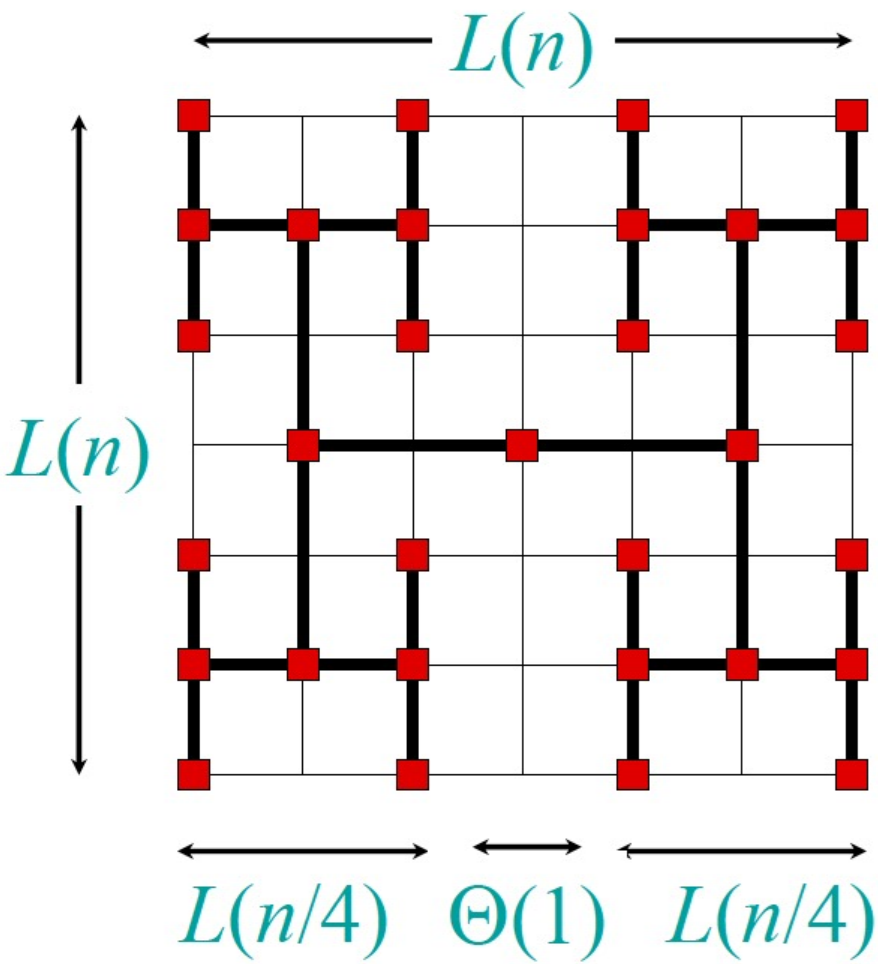


$$H(n) = H(n/2) + \Theta(1) = \Theta(\lg n)$$

$$W(n) = 2\,W(n/2) + \Theta(1) = \Theta(n)$$

$$\text{Area} = \Theta(n \lg n)$$

Représentation d'un arbre



$$
\begin{aligned}
L(n) &= 2L(n/4) + \Theta(1) \\
&= \Theta(\sqrt{n})
\end{aligned}
$$

$$\text{Area} = \Theta(n)$$

Exemple :Plus proche voisin

- Soit un ensemble de point dans R^2, trouver les deux plus proches
- Approche force brute : complexité ?

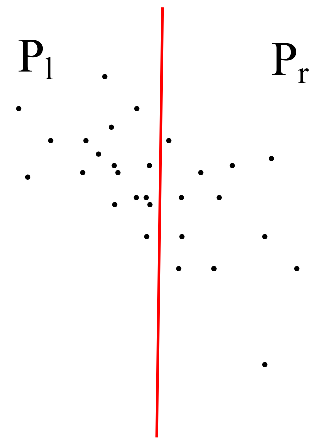
Plus proche voisin

- Principe
  - On travaille avec un ensemble de point (P), et deux tableaux X et Y *chacun contenant tous les points,*
  - X est trié par abscisse croissante,
  - Y est trié par ordonnée croissante

  - Si le nombre de points est 3 on applique « force brute »
  - Sinon divide/conqueer/combine
  - Voir p 192 Levitin

Divide

- Trouver une limite verticale qui sépare P en deux sous ensembles P_l et P_r tel que il y ait autant de points dans chacun (à 1 près)



Divide

- Trouver une limite verticale qui sépare P en deux sous ensembles P_l et P_r tel que il y ait autant de points dans chacun (à 1 près)
- X est divisé en 2: X_l et X_r qui contiennent les points de P_l et P_r (resp) triés par ordre croissant sur l'abscisse

Divide

- Trouver une limite verticale qui sépare P en deux sous ensembles P_l et P_r tel que il y ait autant de points dans chacun (à 1 près)
- X est divisé en 2: X_l et X_r qui contiennent les points de P_l et P_r (resp) triés par ordre croissant sur l'abscisse

- On fait de même sur les ordonnées Y: Y_l et Y_r qui contient les points de P_l et P_r (resp) triés par ordre croissant sur l'ordonnée

Conquer

- On cherche les deux plus proches dans P_l et P_r

  → 2 appels récursifs

  1. avec P_l, X_l et Y_l
  2. avec P_r, X_r et Y_r

  → résultat delta est le min de la plus distance entre deux points dans P_l ($delta_l$) et P_r ($delta_r$)

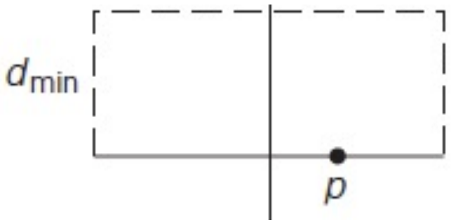
Combine

- Les deux point les plus proches sont
  - Les deux points correspondant à delta (a)
  - Un point de P_l et un point de P_r (b)

  → il faut calculer cela

  Considérer un rectangle delta x 2 delta autour de la limite verticale

  Nombre borné de point à considérer



  - On retourne les deux plus proches soit de (a) soit de (b)
- Complexité annoncée: n log n
  - Il faut un algorithme correspondant à 2T(n/2) + O(n)
  - « divide » doit être linéaire (pas de tri à chaque étape)

Figure p 910 Cormen

Détails du calcul

- Créer un tableau Y’ à partir de Y en enlevant tous les points à plus de 2 delta de la limite verticale
  Y’ est trié par ordonnée croissante
- Pour chaque point p dans Y’, on cherche dans Y’ un point à distance delta (il n’y en a que 7 possibles)
  on garde le couple où la distance est la plus petite

Détails d'implantation

- Verrou : avoir à chaque appel récursif des tableaux X_l et Y_l, X_r et Y_r triés et Y' aussi
- Remarque:

Si X est trié, séparer P en P_l et P_r se fait en temps linéaire

Si on dispose de P_l et P_r, répartir les points en Y_l et Y_r triés se fait en parcourant Y (trié !) et en ajoutant à chaque étape le point soit à Y_l et Y_r selon qu'il appartient à P_l ou P_r (test sur la limite l !)

- Il suffit de trier une fois au début (n log n)

ALGORITHM *EfficientClosestPair*(P, Q)

//Solves the closest-pair problem by divide-and-conquer
//Input: An array P of $n \geq 2$ points in the Cartesian plane sorted in
//   nondecreasing order of their x coordinates and an array Q of the
//   same points sorted in nondecreasing order of the y coordinates
//Output: Euclidean distance between the closest pair of points
if $n \leq 3$
  return the minimal distance found by the brute-force algorithm
else
  copy the first $\lceil n/2 \rceil$ points of P to array P_l
  copy the same $\lceil n/2 \rceil$ points from Q to array Q_l
  copy the remaining $\lfloor n/2 \rfloor$ points of P to array P_r
  copy the same $\lfloor n/2 \rfloor$ points from Q to array Q_r
  $d_l \leftarrow EfficientClosestPair(P_l, Q_l)$
  $d_r \leftarrow EfficientClosestPair(P_r, Q_r)$
  $d \leftarrow \min\{d_l, d_r\}$
  $m \leftarrow P[\lceil n/2 \rceil - 1].x$
  copy all the points of Q for which $|x - m| < d$ into array $S[0..num - 1]$
  $dminsq \leftarrow d^2$
  for $i \leftarrow 0$ **to** $num - 2$ **do**
    $k \leftarrow i + 1$
    while $k \leq num - 1$ **and** $(S[k].y - S[i].y)^2 < dminsq$
      $dminsq \leftarrow \min((S[k].x - S[i].x)^2 + (S[k].y - S[i].y)^2, dminsq)$
      $k \leftarrow k + 1$
return $sqrt(dminsq)$

- https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/