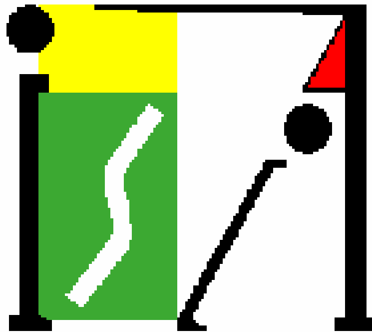
Introduction à l'Algorithmique

I.S.I.A.

Ce que l'on conçoit bien s'énonce clairement et
les mots pour le dire arrivent aisément [Boileau]

Franck.Nielsen@sophia.inria.fr

http://www.inria.fr:/prisme/personnel/nielsen/ISIA/isia.html



Année 1995-1996

INRIA SOPHIA ANTIPOLIS

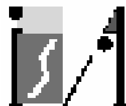
Introduction et Historique

- **Algorithmique**: science des algorithmes.
  - Antiquité: Euclide décrit le $pgcd(a, b)$. Archimède: calcul de π à une précision arbitraire.
  - Muhamed Ibn Al Khou Warizmi: représentation décimale et règles de calcul.
  - ALGORITHME : moyen de résoudre un problème
  - Géometrographie introduite par Lemoine (1902): code les primitives euclidiennes. Notion de simplicité.

Exercice 1 *Donner des exemples d'algorithmes de la vie de tous les jours*

copyright 1996 Nielsen Franck

version page html



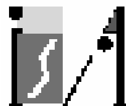
Homme/Ordinateur

Execution fastidieuse d'un "algorithme" par un être humain (calculer à 10^{-6} près $e \simeq 2.71...$).
→Règles répétitives
→Grand nombre d'étapes

L'avénement de l'ordinateur a favorisé le developpement de l'algorithmique. De nos jours, l'algorithmique a une croissance exponentielle.

- Temps d'execution d'un algorithme?
- Efficacité d'un algorithme?
- Difficulté (complexité) d'un problème?

version page html



Temps&Efficacité

Exercice 2 *Donner une méthode de calcul pour calculer* x^i *pour* $x \in \mathbf{R}$ *et* $i \in \mathbf{N}$.

- *Evaluer votre méthode.*
- *Donner une autre méthode*
- *Comparer vos deux algorithmes*
- *Temps de calcul sur une machine qui réalise* 10^6 *mult./s?*

version page html

Définition

Un algorithme construit une solution à partir de données

– Execution automatique. Pas d'ambiguité dans les régles (→sémantique)

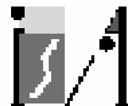
– Entrée d'un algorithme: format défini et taille finie.

– L'algorithme peut être considéré comme une suite d'opérations sur une chaîne de symboles (machine de Turing). Automate.

– Pas de langage associé à un algorithme.

Exercice 3 *Définissez un problème. Décrire ses entrées, le moyen de le résoudre et la(les) sortie(s).*

version page html



Algorithmes/Programmes

Un programme est écrit dans un langage de programmation (PASCAL, C, C++, LISP, ...) et s'execute sur un ordinateur disposant d'une mémoire.

$$\text{Programme} \Rightarrow \text{Algorithme}$$

- L'ensemble des programmes est dénombrable ($|\Sigma|^{Longueur}$).
- $\mathbf{N}^{\mathbf{N}}$ n'est pas dénombrable.

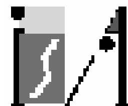
"Tout algorithme est il programmable?"

Réponse inconnue!!! Conjecture d'Alonzo Church:

> Les fonctions $\mathbf{N}^{\mathbf{N}}$ **calculables** sont programmables.

Exercice 4 *Donner des fonctions de* $\mathbf{N} \rightarrow \mathbf{N}$ *et (si possible) leurs algorithmes.*

version page html



Correction Partielle/Totale

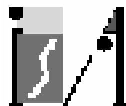
→Vérifiez que l'algorithme résoud bien le problème donné: **Correction Partielle**.

→Vérifiez que pour une entrée fixée l'algorithme résoud le problème et termine en un temps fini: **Correction Totale**.

Exercice 5 *Donner un algorithme juste qui ne termine jamais?*

Exercice 6 *Peut-on déterminer en avance la terminaison d'un algorithme? Existence d'un tel algorithme?*

version page html



Programme A qui vérifie la terminaison?

- Entrée: Un programme P, des données D.
- Sortie: 0 le programme P ne termine pas sur le jeu de données D. 1 le programme termine sur les données D

- Soit le programme P' suivant:

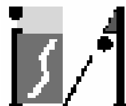
1. $resultat \leftarrow A(P, d)$
2. si $resultat = 1$ alors aller en 2 (on boucle)
3. renvoyer $resultat$

Équivalence des propositions

- P' termine sur P'
- P' ne termine pas sur P'

$\rightarrow$contradiction.

version page html



Pseudo-langage

– Alphabet de l'ordinateur: 0 ou 1. Peu convivial!

– On cherche à cerner les grandes étapes d'un algorithme

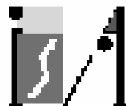
– Programmation ne dépendant pas de la machine (C, C++, LISP, FORTRAN, ...)

– Langage de programmation $\simeq$ 1960 mais le crible d'Erathostène est bien plus ancien (Erathostène 284-192 av. J.C.)!

– →Description formelle d'un algorithme sans entrer trop dans les détails et compréhensible.

– →Organigramme: peu lisible dû aux nombreuses flêches.

version page html



Qualités requises d'un algorithme

– Lisibilité

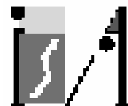
– Décomposition modulaire

– Description des données et du résultat de l'algorithme

→Style "PASCAL" sans la syntaxe.

Les **structures de contrôle** assurent le séquencement des **instructions**. Les données, les résultats intermédiaires et le résultat sont stockés dans des **variables**.

version page html



Pseudo-langage

- La **séquence**: $Instruction_1; Instruction_2$
- La **conditionnelle**:

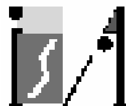
  si C **alors** A_1 **sinon** A_2 **fsi**;
- L'**itération**: 3 formes essentielles
  - Tant que: **Tant que** C **faire**

    début
    A_1;
    A_2;
    ⋮
    A_n;
    fin
  - Répeter jusqu'à: **répeter**

    A_1;
    A_2;
    ⋮
    A_n;
    jusqu'à C;

copyright 1996 Nielsen Franck

version page html



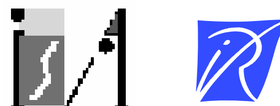
Suite&Exercices

– pour i de $debut$ à fin faire:
pour i **de** $debut$ **à** fin **faire**
début
A_1;
A_2;
⋮
A_n;
fin

Exercice 7 *Illustrer la redondance des structures d'itérations. Donner quelques équivalences de programmes. Y-a t'il des ruptures de structure? Est-ce que la structure* **aller à ...** *apporte de la puissance au pseudo-langage?*

Exercice 8 (pgcd d'Euclide) *Soit* a, b *deux entiers positifs. Calculer le pgcd*(a, b)*, le plus grand commun diviseur de* a *et* b*. Calculer pgcd*$(35, 42)$.

version page html



Fonctions

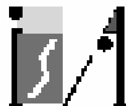
→Afin de rendre les programmes plus lisibles, nous utilisons deux catégories de “boîte englobante”: les **fonctions** et les **procédures**.
Une fonction est définie par son nom suivi d'une liste de paramètres formels typés entre parenthéses. Une fonction renvoit une valeur typée.

fonction $PgcdIteratif(m, n$:**entier**):entier;

Exercice 9 *Écrire la fonction pgcd dont le prototype est donné ci-dessus. Écrire la fonction* **produit_scalaire**.

copyright 1996 Nielsen Franck

version page html



Procédures

Une procédure fait “quelque chose” mais ne renvoit pas de résultats.

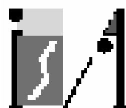
$$\boxed{\textbf{procédure}\ Initialisation(m, n{:}\textbf{entier});}$$

– Un bloc est défini par les mots clefs **début** ... **fin**.

– scope d'une variable

– Affectation $i := 0$ ou $i \leftarrow 0$.

version page html



Passage par valeur/variable

A l'appel d'une fonction ou d'une procédure, les paramètres formels sont soient passés par **valeur** ou par **variable**.

Passage par valeur. Les paramètres formels sont "lus" à l'appel de la fonction/procédure puis leurs valeurs sont affectées à des variables temporaires locales.

→Avantage du passage par valeur?

Passage par paramètres. Les variables des entrées de la procédure sont liées aux paramètres formels. A la sortie de la fonction/procédure, les variables peuvent avoir leur contenu changé!

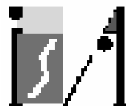
procédure incrémente(**var** i:**entier**);
début
i:=i+1;
fin;

→Incrémente(0)?

```
E 16 - Expression given (variable required) for var parameter
p of incremente Compilation failed
```

copyright 1996 Nielsen Franck

version page html



Programmes récursifs

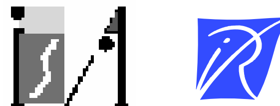
Un programme est dit **récursif** s' il s'auto-appelle afin de construire la solution. Les programmes récursifs se construisent naturellement à partir d'une méthode constructive elle-même récursive. Deux problèmes majeurs se posent: déterminer le cas simple et assurer que l'on arrive à ce cas simple.
Par exemple... la factorielle, le PGCD:

---

fonction pgcd_récursif(m, n):**entier**;
début
si $n > m$ **alors** pgcd_récursif(n, m) **fsi**
si $m = n$ **ou** $n = 1$ **alors** pgcd_récursif$\leftarrow n$
sinon pgcd_récursif←pgcd_récursif(n,m-n);
fsi
fin

Exercice 10 *Quelles sont les propriétés du* $PGCD(a, b)$*? Prouver que l'algorithme précedent termine et qu'il renvoit le PGCD de m et n. Améliorez l'algorithme... N'oubliez pas qu'Euclide (Eudoxus 375 av. J.C?) fit mieux en 300 avant J.C. Étudier sa complexité est toujours au goût du jour...*

copyright 1996 Nicolson Franck
version page html

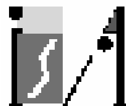


Implantation en C

```
#include "stdio.h"
#include "stdlib.h"

int m,n;

int pgcd(m,n)
int m,n;
{
#ifdef PGCD
fprintf(stdout,"pgcd(%d,%d)\n",m,n);
#endif
if (n>m) return pgcd(n,m);
if (n==0) return m;
if ((n==m)||(n==1)) return n; else return pgcd(m%n,n);
}


main(argc,argv)
int argc;
char**argv;
{
if (argc!=3) {fprintf(stderr,"Appel pgcd(m,m): %s m n\n",argv[0]);
exit(-1);}
    m=atoi(argv[1]);n=atoi(argv[2]);
fprintf(stdout,"pgcd(%d,%d)=%d\n",m,n,pgcd(m,n));

}
```

copyright 1996 Nielsen Franck

version page html



Résultat

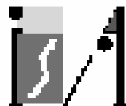
```
pgcd(40902,24140)
pgcd(16762,24140)
pgcd(24140,16762)
pgcd(7378,16762)
pgcd(16762,7378)
pgcd(2006,7378)
pgcd(7378,2006)
pgcd(1360,2006)
pgcd(2006,1360)
pgcd(646,1360)
pgcd(1360,646)
pgcd(68,646)
pgcd(646,68)
pgcd(34,68)
pgcd(68,34)
pgcd(0,34)
pgcd(34,0)
pgcd(40902,24140)=34
```

Exercice 11 *Montrer comment on peut dérecursiver le programme précédent en gérant soi-même la pile.*
La fonction d'Ackermann $A_n(n)$ est définie récursivement comme suit:

- $A_1(n) = 2n$
- $A_k(n) = A_{k-1}^{(n)}(1)$ *pour tout* $k \geq 2$.

Calculer $A(1), A(2), A(3), A(4)\ldots$ *et donner un algorithme pour calculer* $A_n(n)$.

version page html

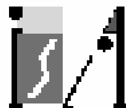


Notion de complexité

- Notion de simplicité (Lemoine 1902).
- Mesurer le temps d'execution d'un algorithme. Mesurer les **ressources** utilisées par un ordinateur (temps, mémoire, I/O, ...)
- Prédire le comportement d'un algorithme. Faut-il 0.01s, 1 min., 1 jour, 1 an?
- Permet de comparer plusieurs algorithmes qui résolvent le même problème, de les classer dans une certaine mesure
- Établir sur un **modéle de machine** précis le nombre minimal d'opérations de base requises.

→Définir un modéle de machine.
→Donner des bornes inférieures pour des problèmes et des bornes supérieures pour des algorithmes.

version page html



“Programme= Algorithme+Structure de données”[Wirth]

- Formule traditionnelle de Niklaus Wirth qui lie les régles algorithmiques aux variables (le stockage de résultats et la **manipulation** efficace de ceux-ci).
- Types scalaires: entier, floatant, booléen, caractère, string, ...
- Types composés: record, $T \times T$.
- Les opérations sur les structures de données... sont eux mêmes des algorithmes.
- Tableau, liste chaînée, ensemble, arbre, ...

copyright 1996 Nicolas Pioch

version page html

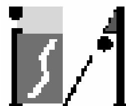


Illustration par la recherche séquentielle

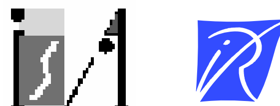
Soit $S = \{a_i\}_{i=1..n}$ une suite de réels et x un élément (réel). On veut savoir si $x \in S$. Les élements sont stockés dans un flux, pour accéder à l'élément a_i, il faut avoir visité les éléments $a_1, ..., a_{i-1}$.

Exercice 12 *Écrire un algorithme de recherche séquentielle qui renvoit -1 si l'élément x n'est pas dans le tableau et i si $x = a_i$.*
Essayer de caractériser le nombre d'instructions réalisées par l'algorithme:

- *Dans le meilleur des cas.*
- *Dans le pire des cas.*
- *En moyenne.*

Illustrer vos résultats sur $n = 2^{10}$, $n = 2^{20}$.
Peut-on améliorer votre algorithme? Que se passe-t-il si les éléments sont stockés dans un tableau trié par ordre croissant (l'accés à l'élément a_i se fait par $a_i := A[i]$?

version page html



Complexité

Dans le cas le pire ($C_M(n)$). On cherche à maximiser le nombre d'opérations possibles par un algorithme sur une taille des données fixée.

- Avantage: l'algorithme "finit" toujours avant $C_M(n)$ opérations.
- Inconvénient: ne refléte pas les situations (c'est-à-dire les données) de votre application (exemples?)

Dans le cas en moyenne ($C_{\overline{m}}(n)$). On veut déterminer en moyenne le temps d'execution de l'algorithme pour une taille n des données.

- Avantage: l'algorithme marche bien en moyenne si les données sont réparties suivant le modèle utilisé pour calculer la moyenne. Analyse randomisée.
- Inconvénient: pour des applications temps réels, le cas le pire peut se produire.

→Calcul des moments de la fonction complexité; variance, écart-type. Probabilité que le temps de calcul dépasse k fois la moyenne $C_{\overline{m}}(n)$.

Dans le meilleur des cas. Comportement invraisemblable mais qui donne une borne inférieure pour le cas le pire et en moyenne de l'algorithme.

version page html

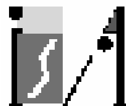
Recherche Séquentielle

fonction RechercheSéquentielle(S, x):entier;
var
i:entier;

début
$i \leftarrow 1$;
RechercheSéquentielle$\leftarrow -1$;
tant que $i \leq n$ **faire**
si $a_i = x$ **alors** RechercheSéquentielle$\leftarrow i$; **fsi**;
$i \leftarrow i + 1$;
fin

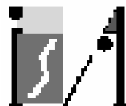
Exercice 13 *Prendre en compte le cas de plusieurs éléments identiques. Améliorer l'algorithme précedent.*

version page html



```
program RechercheSequentielle;
(* Un algorithme bien naif pour la recherche sequentielle *)
const  n=500;

type element=integer;

var
   x : element;
   S : array[1..n] of element;

procedure Initialisation;
var
   i : integer;
begin
   for i:=1 to n do S[i]:=i;
end; { Initialisation }

function Recherche(x : integer):integer;
var
   i : integer;

begin
   Recherche:=-1;
   i:=1;
   while(i<=n) do begin
      if (x=S[i]) then Recherche:=i;
      i:=i+1;
   end;
end; { Recherche }

begin
   writeln('Recherche sequentielle dans un flux');
   Initialisation;
   writeln('Cherche element:');
   readln(x);
   writeln('Resultat:',Recherche(x));

end.
```

version page html

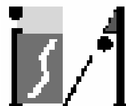


Recherche séquentielle...

Algorithme avec coupure dés qu'un élément x est trouvé et sentinelle.

fonction RechercheSequentielle(S, x):**entier**;
var
i:entier;
début
$i \leftarrow 1$;
$a_{n+1} \leftarrow x$;
tant que $a_i \neq x$ **faire** $i \leftarrow i + 1$ **ftq**;
RechercheSequentielle$\leftarrow i$;
fin

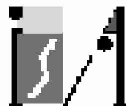
copyright 1996 Nielsen Franck
version page html



En C++

```
#include<iostream.h>

#define n 500

typedef  int element;

int i;
element x, S[n+1];

void Initialisation()
{
for(i=0;i<n;i++) S[i]=i+1;
}



int Recherche(element x)
{
i=0;
S[n]=x;
while(S[i]!=x) i++;

if (i<n) return i+1; else return -1;
}



main()
{
cout << "Recherche Sequentielle avec sentinelle"<<endl;
Initialisation();
cout <<"Recherche x:";
cin >>x;
cout << "Recherche:"<<Recherche(x)<<endl;
}
```

version page html



Analyse en moyenne

Soit q la probabilité que $x \in S$ et $\mathcal{D}$ l'ensemble des données de taille n. On partitionne les données en $n+1$ paquets $\mathcal{D}_0, ..., D_n$ suivant que $x \notin \mathcal{D}$, $rank(x) = i$ pour $1 \leq i \leq n$.

$$C_{\overline{m}}(n) = \sum_{d \in \mathcal{D}} \Pr(d) c(d)$$

$$C_{\overline{m}}(n) = \sum_{d \in \mathcal{D}_0} \Pr(d) c(d) + \sum_{i=1}^{n} \sum_{d \in \mathcal{D}_i} \Pr(d) c(d)$$

$$C_{\overline{m}}(n) = (n+1) \sum_{d \in \mathcal{D}_0} \Pr(d) + \sum_{i=1}^{n} i \sum_{d \in \mathcal{D}_i} \Pr(d)$$

Toutes les places sont équiprobables à partir du moment où l'élément est dans S.

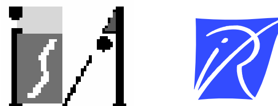
$$\sum_{d \in \mathcal{D}_i} \Pr(d) = \frac{q}{n}$$

$$C_{\overline{m}}(n) = (n+1)(1-q) + \frac{q}{n} \sum_{i=1}^{n} i$$

$$C_{\overline{m}}(n) = (n+1)(1 - \frac{q}{2})$$

Exercice 14 *Commenter la fonction $C_{\overline{m}}(n)$. Ajouter les affectations. Analysez l'algorithme dans le cas d'une recherche dichotomique (tableau trié)*

version page html



Déviation de $C_{\overline{m}}(n)$

Soit X la variable aléatoire décrivant la complexité en nombre d'opérations d'un algorithme. On cherche à caractériser l'écart entre X et $\mathbf{E}[X] = C_{\overline{m}}(n)$.

Inégalité de Markov. Pour tout $\alpha > 0$, on a

$$\Pr(X \geq \alpha) \leq \frac{\mathbf{E}[X]}{\alpha}$$

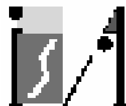
Ainsi $\Pr(X \geq kC_{\overline{m}}(n)) \leq \frac{1}{k}$.

Inégalité de Chebychev. La variance de X est l'espérance $\mu = \mathbf{E}[(X - \mathbf{E}[X])^2]$. La déviation standard est $\sigma = \sqrt{\mu}$. On a:

$$\Pr(|X - \mu| \geq t\sigma) \leq \frac{1}{t^2}$$

Technique de Chernoff. technique utilisée afin d'avoir de meilleurs estimateurs de la déviation à la moyenne. Basée sur le principe de $X = \Sigma_{i=1}^{k} X_i$.

version page html



Complexité asymptotique

On utilise la théorie de la complexité afin de comparer des algorithmes qui résolvent le même problème. On peut alors compararer leur comportement dans un intervalle donné afin de choisir le meilleur (entendez le plus rapide!) pour une taille connue des données.

Exemple: Soit A et B deux algorithmes pour un problème P donné de complexité respective $c_A(n) = 500 \times n$ et $c_B(n) = 10 \times n^2$. B est plus rapide si $n < 50$ et plus lent si $n > 50$.

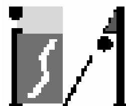
– Importance des constantes.

– En général on ne peut pas calculer exactement la complexité mails seulement des bornes qui l'encadre.

– Algorithme performant cherché pour traiter un grand nombre de données. Basé sur le fait que si $n <$ constante alors on peut essayer de trouver pour chaque n un algorithme optimisé pour cette valeur.

$\rightarrow$**Complexité aymptotique** ($n \rightarrow +\infty$).
Complexité du problème P.

version page html



Notations asymptotiques

→Estimer un ordre de grandeur. Classer les fonctions de complexité c modulo les relations d'équivalence définies ci-dessous.

Borne supérieure On note c(n)=O(f(n)) si:

$$\exists n_0 \in \mathbf{N},\ \exists A \in \mathbf{R}^+,\ \forall n \geq n_0\ c(n) \leq A \times f(n)$$

On parle de borne supérieure d'un **algorithme**.

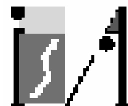
Borne inférieure. On dit que le problème P a une borne inférieure $\Omega(f(n))$ ssi tout algorithme qui résout P a une complexité $c(n)$ vérifiant

$$\exists n_0 \in \mathbf{N},\ \exists A \in \mathbf{R}^+,\ \forall n \geq n_0\ c(n) \geq A \times f(n)$$

Optimalité. Enfin un algorithme est **optimal** si sa complexité $c(n)$ atteint la complexité du problème $p(n)$ ($c(n) = O(p(n))$). On note $\Theta(c(n))$ la complexité de cet algorithme.

Exercice 15 *Ordonner les fonctions suivantes à l'aide de la classe $O(\cdot)$ ou $\Omega(\cdot)$.*
$f \in \{1, n, n\log n, n\log^k n, n^2, n^k, n^n, n!, e^n, 2^n\}$.

version page html



Ordre de grandeur

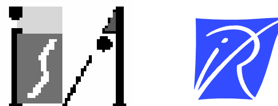
Un jargon spécifique pour la complexité:

$\log n$	complexité logarithmique
n^2	complexité quadratique
n^3	complexité cubique
n^k	complexité polynomiale
2^n	complexité exponentielle
2^{2^n}	complexité doublement exponentielle

Une opération élémentaire (la $\times$) se fait sur un SPARCstation IPX (4/50) en 0.000000883298 seconde. Si le coût d'une opération est 10^{-6} alors nous obtenons la grille suivante des temps d'execution:

Complexité	$\log n$	n	$n \log n$	n^2	n^3	2^n
10^2	$6.6\mu s$	$0.1ms$	$0.66ms$	$10ms$	$1s$	4×10^{16}année
10^3	9.9μ	$1ms$	$9.9ms$	$1s$	$16.6mn$	$+\infty$
10^4	$13.3\mu s$	$10ms$	$0.13s$	$1.5mn$	$11.5j$	$+\infty$
10^5	$16.6\mu s$	$0.1s$	$1.64s$	$2.7h$	$31.7a$	$+\infty$
10^6	$19.9\mu s$	$1s$	$19.9s$	$11.5j$	$31.7 \times 10^6 a$	$+\infty$

copyright 1995 Nicolas Franck



Produit de matrices

Exercice 16 *Donner un algorithme pour le produit scalaire de deux vecteurs d'entiers* $\vec{v_1} = (v_{1,i})_{i=1..d}$ *et* $\vec{v_2} = (v_{2,i})_{i=1..d}$*. Sa complexité? En déduire un algorithme pour le calcul du produit de deux matrices* $A \times B$*. Donnez la complexité de votre algorithme.*

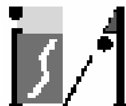
fonction ProduitScalaire(v_1, v_2):**entier;**
var
i, r:**entier**;

début
$r \leftarrow 0$;
pour i **de** 1 **à** d **faire**
$r \leftarrow r + v_{1,i} * v_{2,i}$;

ProduitScalaire$\leftarrow r$;
fin

Soit $A = (a_{ij})_{i,j}$ une matrice de taille (n_A, m) et $B = (b_{i,j})_{i,j}$ une matrice de taille (m, n_B) à coefficients entiers.

copyright 1996 Nicolson Franck
version page html



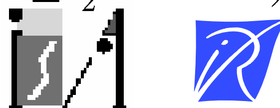
Produit de matrices(suite)

fonction ProduitMatrice(A, B):**matrice**;
var
i, j:**entier**;
C:**matrice**;
début pour i **de** 1 **à** n_A **faire**
  début
  pour j **de** 1 **à** n_B **faire**
  c_{ij} =ProduitScalaire(a_{i*}, b_{*j});
  fin
fin

- Algorithme "naïf" en $O(n^3)$ multiplications.
- Strassen proposa en 1969 un algorithme en $O(n^{2.81})$ multiplications (mais $O(n^3)$ opérations arithmétiques). Coppersmith et Winograd en 1986 donna un algorithme en $O(n^{2.52})$-$\times$.
- Temps d'une multiplication peu différent du temps d'une addition.
- Borne inférieure en $\Omega(n^2)$.

Exercice 17 *Étudiez la complexité de l'algorithme d'Euclide (note: $a \bmod b \leq \frac{a}{2}$ si $a > b$).*

version page html



Programme&Ordinateur

– Définir un modèle de machine qui colle aux machines existantes. Plusieurs modèles.

– Problème de précisions numériques:

$$a_n = \frac{6^{n+1} + 5^{n+1}}{6^n + 5^n}$$

la suite $(a_n)_n$ converge vers 6 mais sur les ordinateurs utilisant le format arithmétique flottant, elle converge vers 100.

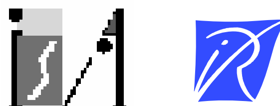
$\rightarrow$On suppose la précision infinie dans nos algorithmes mais il faut avoir conscience du réel problème en pratique.

– Vérification de programme grâce à la sémantique qui permet de valider l'implantation. Construction automatique à partir d'une preuve. Spécification en λ-calcul.

L'algorithmique est une discipline extrêmement vivante!

copyright 1996 Nielsen Franck

version page html



Que fait un programme?

Problème de terminaison (conjecture de Syracuse):

fonction f(n:**entier**;):**entier**;
début
si $n = 1$ **alors** $f \leftarrow 1$ **sinon**
    si $(n \bmod 2 = 0)$ **alors** $f \leftarrow f(\frac{n}{2})$
    sinon $f \leftarrow f(3n + 1)$ **fsi**
    fsi
fin;

Problème du résultat (fonction 91 de Mc Carthy):

fonction f(n:**entier**;):**entier**;
début
si $n > 100$ **alors** $f \leftarrow n - 10$ **sinon**
    $f \leftarrow f(f(n + 1))$;
fsi
fin;

version page html