Analyse d'algorithmes

Jean-Marc Vincent[1]

[1] Laboratoire LIG
Equipe-Projet MESCAL
Jean-Marc.Vincent@imag.fr

Organisation

Problématique

Exemples de problèmes

Rechercher un mot dans un texte ;

Compresser un signal audio ;

Calculer une trajectoire de robot ;

Construire un emploi du temps ;

Extraire les contours d'une image ;

Gérer d'une base de donne ;

Résoudre des équations mathématiques ;

Simuler des systèmes complexes ;

Contrôler un environnement distribué ;

...

Problème général

Résoudre un problème (traitement de données)

1. Spécification du problème
2. Implémentation (écriture d'un algorithme)
3. Validation (preuve)
4. Implantation (écriture d'un programme)
5. Validation (test)
6. Mise en production

Hypothèse sous-jacente :

ON DISPOSE DE RESSOURCES (machine)

Ressources : niveaux d'abstraction

Architecture matérielle

mémoire,
registres, compteurs,
circuits, opérateurs,...

Systèmes d'exploitation

Processus, processus légers,
caches, pages mémoire,
I/O, clavier, écran,...

Fonctionnalités du langage de programmation

opérations élémentaires,
types abstraits, piles, files, tables de hashage,
librairies,...

Questions

- Est-ce-que je peux construire un programme qui résout mon problème sur une machine donnée ?

  Spécification du problème et construction d'un algorithme

- Est-ce-que l'excution de mon programme sur la machine donne bien le résultat souhaité ?

  Vérification de propriétés qualitatives

- Est-ce que mon programme me fournit le résultat en un temps acceptable ?

  Validation de propriétés quantitatives

Modèle de machine

Une machine est constituée d'un processeur, d'une **mémoire** et d'un ensemble d'**opérations**.

- **Mémoire**
  - infinie (un calcul utilise un espace fini),
  - types élémentaires (finis) `int, float,...`
- **Opérations**
  - nombre fini d'opérations ;
  - chaque opération a un nombre fini de paramètres (nombre fini de variables lues et écrites).
- **Processeur**
  - processeur unique ;
  - effectue les opérations en temps constant (1 top = 1 opération)

Exécution d'un programme

Calcul

Un **calcul** est une séquence d'opérations qui, partir d'une configuration initiale de la mémoire, produit, en un temps fini, une configuration terminale.
- exécution d'un algorithme = séquence d'opérations
- coût (en temps) de l'algorithme à partir d'une configuration initiale d = nombre d'opérations dans la séquence d'exécution à partir de d
- coût en espace mémoire = nombre maximal de variables utilisées simultanément au cours de l'excution

Remarques

- Analogie avec le modle RAM (Random access machine)
- Lien avec le modèle de machine de Turing

Exemple (1)

Multiplication de 2 entiers

```
int mul(int a, int b) { entiers de taille fixe}
{
int c ;
c=a*b ;
return(c) { résultat entier de taille fixe}
}
```

Coût de l'algorithme

Coût en nombre d'opérations arithmtiques sur les entiers = 1
Coût en espace mémoire = 3 (entiers)

Coût constant, indépendant de la valeur de la donnée

Exemple (2)

Puissance : calcul de p^n

```
int puissance_1 (int p, int n) {entiers de taille fixe}
{
int r,k ;
r=1 ; k=0 ;
while (k !=n) do
   {
   r=r*p ;
   k=k+1 ;
   }
end while
return(r) {résultat entier de taille fixe}
}
```

Coût de l'algorithme

- Coût en nombre d'opérations arithmétiques sur les entiers = n
- Coût en espace mémoire = 4 (entiers)

Coût variable, dépend de la valeur des données

Taille des données en entre : n (en fait $\log_2(n)$)

$\Rightarrow$ Expliquer le coût en fonction de la taille

Améliorer l'algorithme

Exemple (3)

Maximum d'un tableau : T

```
int maximum (int * T, int n)
{T tableau d'entiers distincts,}
{n taille du tableau }
{
int max,i ;
max= int_minimal_value ;
for (i=0 ; i < n ; i++) do
   if (T[i] > max)
   {
   max = T[i] ;
   Traiter(max) ;
   }
end for
return(max)
}
```

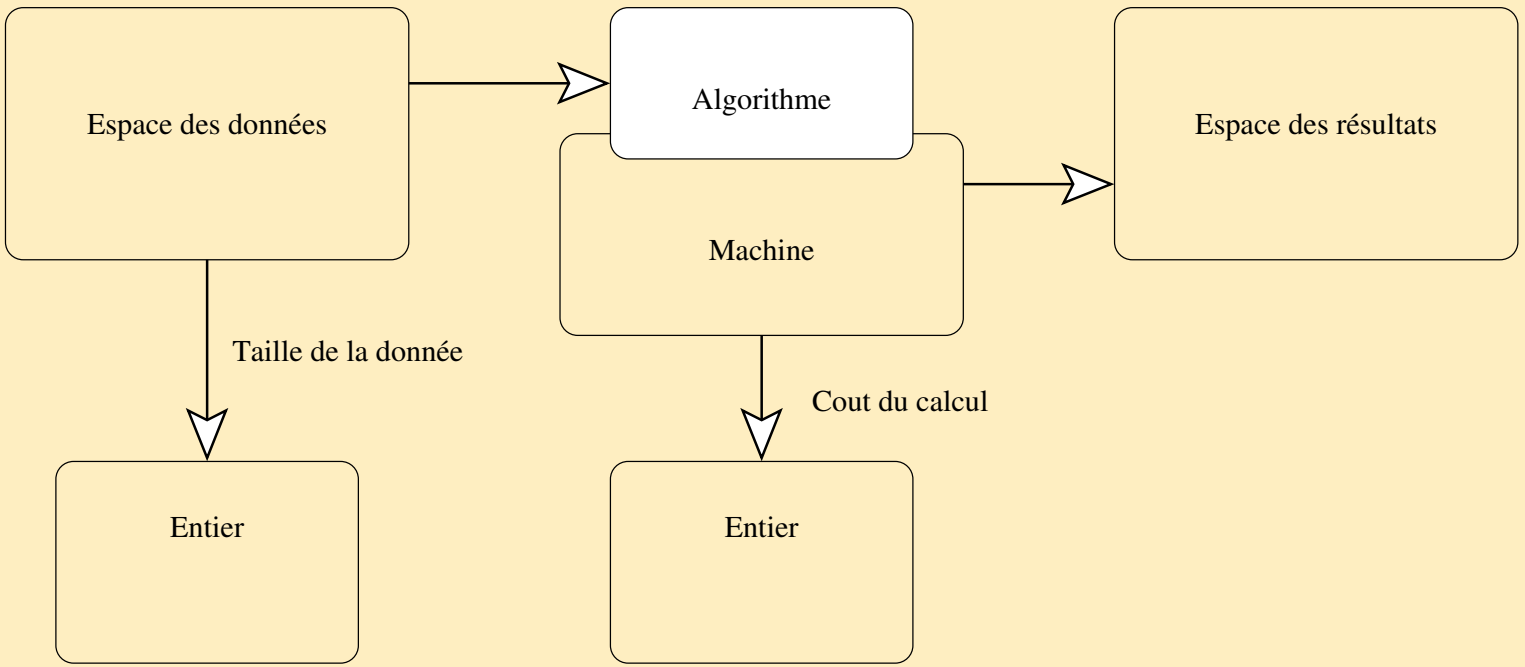
Coût de l'algorithme

Nombre d'appels **Traiter**

- minimum : 1
  exemple : T=[n,1,2,...,n-1]
  nb cas min : $(n-1)!$
- maximum : n
  exemple : T=[1,2,...,n]
  nb cas min : 1

Borné par une fonction linéaire

Synthèse



Trouver la relation entre la taille des données et le coût de l'algorithme sur une machine donnée

Complexité d'un algorithme

Complexité

La complexité d'un algorithme $\mathcal{A}$ en fonction de la taille des données :

$$C_{\mathcal{A}}(n) = \max(\text{coût}(d); d \text{ donne de taille } n)$$

- Suppose avoir fixé la notion de **taille**
- Maximum : complexité au pire
- Donner l'exemple

Cas le plus favorable :

$$C_{\mathcal{A}}^{min}(n) = \min(\text{cot}(d); d \text{ donne de taille } n)$$

complexité au mieux (utile car on sait que l'on ne peut pas mieux faire !)

Idéal : complexité au mieux et au pire proches et petites ?

Complexité des structures de base

Instructions en squence

Le coût de

```
a= calcul(machin) ;
b= traitement(bidule) ;
```

est la somme des coûts de calcul et de traitement

Composition des cots

La complexité d'une itération est donc la somme des coûts des opérations de l'itération

Instructions **FOR, WHILE, REPEAT,...**

Complexité des structures de base (2)

Instructions conditionnelles

if Condition **then**
  a= calcul(machin) ;
else
  b= traitement(bidule) ;
end if

coût de l'une des branche plus le coût d'évaluation de condition

Majoration du coût

La complexité au pire sera donc majorée par le maximum des complexités au pire de chaque branche.

$$C(\textit{if Condition alors A sinon B}) \leq C(\textit{evaluation}(\textit{Condition}) + \max\{C(A), C(B)\}$$

Instructions **IF, SWITCH,...**

Complexité des structures de base (3)

Appel de procédure

Le coût de l'appel d'une procédure est le coût du corps de la procédure pour ses paramètres d'appel plus le coût de l'évaluation de ses paramètres.

Méthode de calcul de complexité

Le calcul de la complexité d'un algorithme s'obtient donc en composant les complexités des différentes opérations composant l'algorithme.

- assemblage et reconstruction
- méthode par composition
- se fait à la **conception de l'algorithme**

Ordres de grandeur

- La complexité est une prédiction du temps d'exécution du programme codant l'algorithme.
- Mais dépend de l'architecture de la machine, donc c'est une abstraction (approximation)
- Passage l'échelle des algorithmes

Ce qui est important c'est :

1. l'ordre de grandeur ;
2. de pouvoir comparer les algorithmes

Exemples

- somme des éléments d'un tableau de taille n
  $\Rightarrow$ n opérations
- tri par insertion des éléments d'un tableau de taille n
  $\Rightarrow$ n^2 opérations
- énumération des vecteurs de bits de taille n
  $\Rightarrow$ 2^n opérations

Définir des ordres de grandeur comparables

Borne supérieure asymptotique

Notation $\mathcal{O}$

Pour une fonction donnée g on note $\mathcal{O}(g)$ l'ensemble de fonctions :

$$\mathcal{O}(g) = \{f \text{ telles que } \exists c \geq 0, \exists n_0 \geq 0, \forall n \geq n_0, f(n) \leq c.g(n)\}\,;$$

on écrit $f = \mathcal{O}(g)$ pour $f \in \mathcal{O}(g)$ et on dit que g est une borne supérieure asymptotique pour f.

$$x = \mathcal{O}(x^3);\quad 1250x^2 = \mathcal{O}(2^x); \cdots$$

$$x + \log x = \mathcal{O}(x);\quad x\sqrt{x} + x\log x = \mathcal{O}(x\sqrt{x});$$

...

Autres encadrements

Borne inférieure asymptotique (notation Ω)

Pour une fonction donnée g on note $\Omega(g)$ l'ensemble de fonctions :

$$\Omega(g) = \{f \text{ telles que } \exists c \geq 0, \exists n_0 \geq 0, \forall n \geq n_0, c.g(n) \leq f(n)\};$$

on écrit $f = \Omega(g)$ pour $f \in \Omega(g)$ et on dit que g est une borne inférieure asymptotique pour f.

Borne asymptotique approchée : (notation Θ)

Pour une fonction donné g on note $\Theta(g)$ l'ensemble de fonctions :

$$\Theta(g) = \{f \quad \text{telles que} \quad \exists c_1, c_2 \geq 0, \exists n_0 \geq 0, \forall n \geq n_0, \\ c_1.g(n) \leq f(n) \leq c_2.g(n)\};$$

on écrit $f = \Theta(g)$ pour $f \in \Theta(g)$ et on dit que g est une borne asymptotique approchée pour f.

Echelles de comparaison

n : taille des données

- Echelle polynomiale : x^n

  $$m \leq n \;\; x^m = \mathcal{O}(x^n)$$

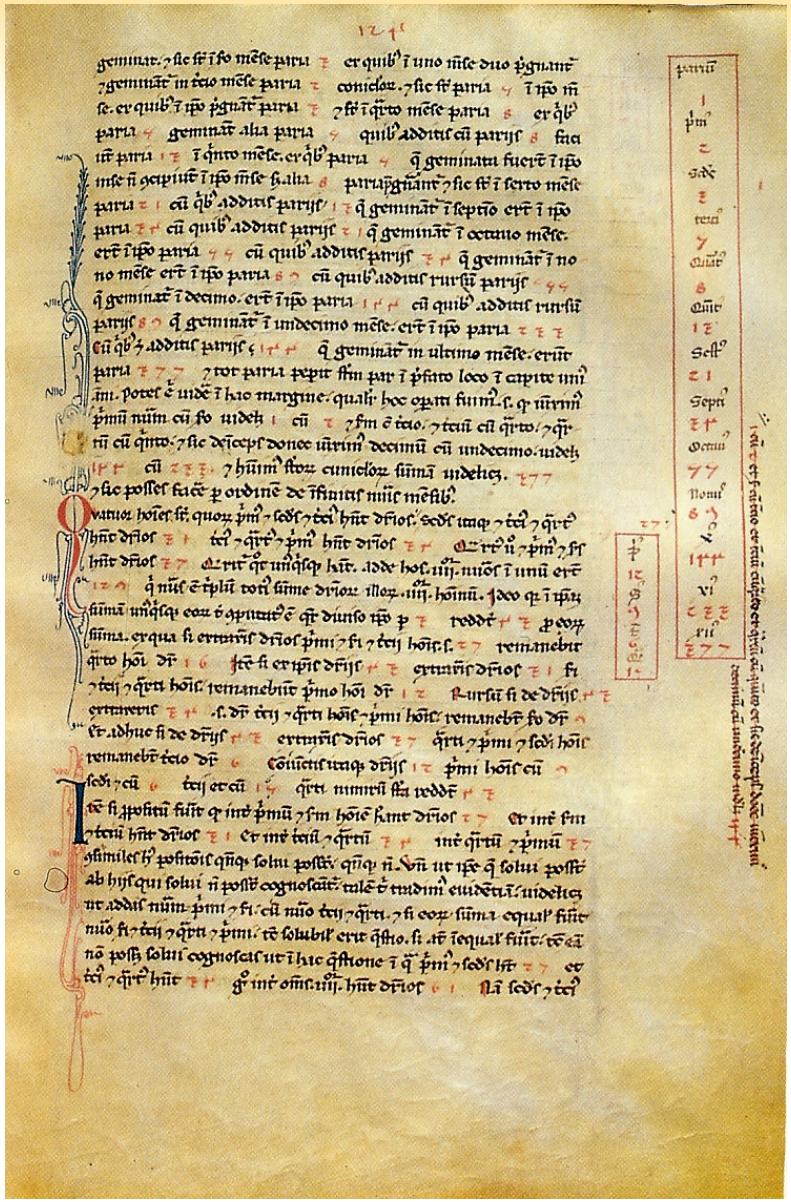
- Echelle logarithmique : $\log(x),\ \log(\log(x)), \cdots$
- Echelle exponentielle : $e^n,\ 2^n, \cdots$

  Pause courbes

Les lapins de Fibonnacci (1170-1250)

Leonardo Pisano dit Fibonnaci rédige le Liber abaci vers 1202

Possédant initialement un couple de lapins, combien de couples obtient-on en douze mois si chaque couple engendre tous les mois un nouveau couple à compter du second mois de son existence ?



Les lapins de Fibonnacci (suite)

$F_n =$ population de couples de lapins au n-ième mois

$F_0 = F_1 = 1$ Condition initiale

$F_2 = 2$

$F_3 = 3$

$F_4 = 5$

$\vdots$

$F_n = F_{n-1} + F_{n-2}$

$\vdots$

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,$

$10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040,$

$1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, \cdots$

The On-Line Encyclopedia of Integer Sequences

Les lapins de Fibonnacci (suite)

Le calcul de F_n

```
number fibo(number n)
{Version récursive}
if n=0 or n=1 then
   return 1
else
   return fibo(n-1)+fibo(n-2)
end if
```

Complexité en nombre d'opérations + notée $C(n)$

$$C(n) = C(n-1) + C(n-2) + 1$$

$$C(0) = C(1) = 0$$

Ordre de grandeur ?

0,0,1,2,4,7,12,19,31,...

Les lapins de Fibonnacci (suite)

Ordre de grandeur

$$C(n) = C(n-1) + C(n-2) + 1; \quad C(0) = C(1) = 0$$

$$2C(n-2) \leq C(n) \leq 2C(n-1)$$

$$2^{n/2-1} C(2) \leq C(n) \leq 2^{n-1} C(2)$$

$$\log_2 C(n) = \Theta(n)$$

Peut-on faire mieux ?

Les lapins de Fibonnacci (suite)

Le calcul de F_n

```
number fibo(number n)
{Version itérative avec mémoire}
if n=0 or n=1 then
    return 1
else
    a=1 ; b=1 ;
    for i =2 to n do
        c=a+b ; a=b ; b=c
    end for
end if
```

Complexité en nombre d'opérations + notée $C'(n)$

$$C'(n) = n - 1 = \Theta(n)$$

Peut-on faire mieux ?