# Introduction a la programmation

David Wiedemann

## Table des matières

# List of Theorems

**Lecture 1: Partie theorie** Fri 18 Sep

# 1 Information, Calcul et Communication

## 1.1 Introduction

Objectifs :

— Vous convaincre de l'importance de ce cours

— insister sur le role de l'informatique

Presenter l'info en tant que discipline scientifique.

Fonde sur 3 grands principes fondamentaux :

— representation discrete du monde

— representation entachee d'erreurs, mais controlee

— variabilite de la difficulte des problemes et des solutions

## 1.2 A quoi ca sert ?

— la simulation/ l'optimisation

— l'automatisation

— Gestions de donnees

### 1.2.1 Calcul scientifique

— Utilisation : simulation de systemes complexes

— Exigences : grande puissance de calcul.

### 1.2.2 La conduite de processus

— Utilisation : tres nombreuses applications : pilotage/surveillance de processus industriels

— Exigences : necessite d'un faible encombrement, consommation reduite, d un cout minimum et d'une grande fiabilite

### 1.2.3 La gestion d'information

— Utilisation : gestion de systemes bancaires ou boursiers, commerce electronique, fichiers de police

— Exigences : importantes de capacite, traitement efficace, controle de processus

# 2 Plan du cours

1. Fondement du calcul

2. Calcul et algorithme

3. Strategies de calcul

4. theorie du calcul

1. Information et communication

2. Echantillonage

3. Reconstruction

4. Entropie et information

5. Compression des messages/donnees

1. Fondements des systems

2. Architecture des ordinateurs

3. Stockage et reseaux

1. Secureite informatique

2. RSA

3. Problemes sociaux

**Lecture 2: Calcul et Algorithmes I**

Fri 18 Sep

# 3 Algorithmes

## 3.1 Formalisation d'algorithmes

**Definition 1 (Algorithme)**
*Un algorithme est une description abstraite des etapes conduisant a la solution d'un probleme*

**Exemple**
***Probleme :***
*Trouver la valeur maximale dans une liste*
*Une liste c'est un element du produit cartesien de $E^n$, n taille de la liste.*
*On pourrait ordonner la liste et retourner le dernier element.*

- *comparer les elements de la liste entre eux*
- *a chaque fois prendre le plus gd.*
- *au fur et a mesure*

### 3.1.1 Methodologie

But : Pour un probleme, trouver une sequence d'actions permettant de produire une solution acceptable en un temps raisonnable.

- Bien identifier le probleme
  - Quelle question ?
  - Quelles entrees ?
  - Quelles sorties ?
- Trouver un algorithme correct ?
  verifier qu'il est effectivement correct, qu'il se termine dans tous les cas.
- trouver l'algorithme le plus efficace possible

### 3.1.2 Qu'est-ce qu'un algorithme ?

Moyen pour un humain de representer la resolution d'un probleme donne

**Definition 2 (Algorithme)**
*Composition d'un ensemble fini d'operations elementaires bien definies operant sur un nombre fini de donnees et effectuant un traitement bien defini :*

— *suite finie de regles a appliquer*

— *dans un ordre determine*

— *a un nombre fini de donnees*

*Un algorithme peut etre*

— *sequentiel : operations s'executent en sequence*

— *parallele : certaines de ses operations s' executent en parallele : simultanement*

— *reparti : certaines de ses operations s executent sur plusieurs machines.*

### Lecture 3: Programmation :Variables et Expressions

Thu 24 Sep

# 4 Etude de cas

**But**

— Resumer ce qu'il faut retenir
— Etude de cas ( simple ici)
— Complements de cours
— Repondre a des questions

## 4.1 Le langage C++

Cest un language oriente-objet, compile et fortement type.

Parmi les avantages de C++ :

— tres utilise
— compile, applications efficaces
— typage fort

A retenir :

— variable = representation interne d'une donnee du probleme traite Une entite conceptuelle representee dans le programme.
— Chaque valeur est typee
— Les principaux types sont intt, double et bool

## 4.2 Valeurs litterales

— valeurs litterales de type int : 1,12, etc
— double : 1.23, ...
— valeurs litterales de type char : 'a', '!', ...

```
1  char x('B');
```

### 4.3 Expression

— un calcul a faire
— expression= combinaison d'expressions, valeurs, a l aide d'operateurs
— toute expression *fait* quelque chose et *vaut* quelquechose.

### 4.4 `auto`

En c++11, on peut laisse rle compilateur deviner le type d'une variable grave au mot-cle `auto`.
Par exemple :

```
auto val(2);
auto j(2*i+5);
```

Conseil : ne pas en abuser !
N'utilisez `auto` que dans les cas "techniques" qui seront vus plus tard.

### 4.5 `const` et `constexpr`

Par defaut, les variables sont modifiables.
si une variable est initialisee avec `const` mais elle est modifiable par pointeur.
`constexpr` est pas modifiable, doit etre connue au moment de la compilation et n'est pas modifiable par pointeur.

### Lecture 4: Ingredients de base des algorithmes

Fri 25 Sep

## 5 Definition formelle des algorithmes

Un algorithme travaille sur des donnees qu'il utilise et/ou moidifie. Il doit memoriser ces donnees en les associant a un nom. Les traitements sont associes a la notion d'instructions et d'expressions.

### 5.1 Instructions elementaires

Une instruction elementaire est une instruction dont le cout d'execution est constant ( negligeable par rapport a la taille des donnees).
Exemples :

— $delta = b^2 - 4ac$
— instruction non elementaire : compter le nombre de caracteres contenus dans une phrase ( depend de la longueur de la phrase)

### 5.2 Structures de controle

Pour pouvoir exprimer des traitements interessants/complexes, un algorithme ne peut se reduire a une sequence lineaire d'instructions.

Structures de controle : Il y en a 3

- les branchements conditionnels : `si... alors`
- les boucles conditionnelles : `tant que`
- les iterations : `pour ... allant de ... a ...` , ou alors : `pour ... parmi`

Note : on peut toujours faire des iterations avec des boucles conditionnelles.

## 5.3 Resolution d'une equation du second degre

entree : $b, c$

sortie : $\{xdansR : x^2 + bx + c = 0\}$

```
Delta = b^2 - 4c
Si Delta < 0
sortir:
Sinon
  si Delta= 0
    x= -b/2
    sortir: x
  Sinon
    x_1 = -b-sqrt(Delta)/2,
    x_2 = -b+sqrt(Delta) /2,
    sortir x1 et x2
```

## 5.4 Conclusion

On attend d'un algorithme qu'il se termine, produise le resultat correct ( solution du probleme) pour toute entree.

Difficulte de l'informatique ( science) : assurer que l'algorithme est correct pour toute entree.

On ne peut pas verifier par des essais ( empirisme) : il faut faire des preuves mathematiques.

## 5.5 Comment, a partir d'un probleme concret, trouver une solution ?

Pour des problemes non-connus, cf semain prochaine. Sinon, si algorithme connu :

- Recherche
- Tri
-

Exemple : recherche d'un element x dans une list $E$.

Avant tout : specifiaction claire du probleme et de l'algorithme voulu : $E$ peut il etre vide ?
$E$ varie-til pendant la recherche ?
$E$ est il ordonne ?

## 5.6 Recherche

Deux algorithmes de recherche :
`appartient1`
entree : $x, E$
Sortie $x \in E$?

```
i=1
Repeter
  Si x= E[1]
    Sortir oui
  i=i+1
  t = taille(E)
jusqua i>t
Sortir non
```

Deuxieme solution
`appartient2`

```
t = taille(E)
Pour i de 1 a t
  si x=E[i]
    Sortir: oui
Sortir : non
```

Dans le premier algorithme, on a suppose $E$ non vide, si $E$ est vide, l'algorithme n'est pas valable.
Le deuxieme algorithme est correct pour un ensemble vide.

Lequel des deux est le plus efficace ?
complexite : nombre d'instructions elementaires necessaires a un algorithme pour donner la reponse dans le pire des cas.

Notons $n$ la taille de $E$ et comptons combien d'instructions elementaires que chaque algorithme necessite dans le pire des cas : $C_1(n)$ et $C_2(n)$
Pour l'algorithme 1 :
Le pire des cas est si $x$ n'est pas dans $E$. On trouve :

$$C_1(n) = 1 + n \times (4 + T(n))$$

Pour l'algorithme 2 : Le pire cas est le meme que pour l'algorithme 1.

$$C_2(n) = T(n) + 1 + 4n$$

Supposons que le calcul de la taille de $E$ se fait en $T(n) = a + bn$ instructions ( $b \geq 0$). On aurait alors :

$$C_1(n) = 1 + (a+4)n + bn^2$$
$$C_2 = 1 + a + (4+b)n$$

## 5.7 Dichotomie

Algorithme : appartientD$(x, E_1)$

```
1  Si E vide
2    sortir non
3  Si E eest reduit a 1 seul element
4    Sortir x=e?
5
6  decouper E en deux sous-ensembles non vides
7  et disjoints E1 et E2
8  Si x< max(E1)
9    sortir appartientD(x,E1)
10 sinon
11   sortir appartientD(x,E1)
```

Cet algorithme est-il meilleur que l'algorithme `appartient2` ?
Si l'element recherche est au "milieu" du "milieu" du "milieu" ... de la liste, il faudra repeter la bouclede decoupage en deux autant de fois. Donc la complexite est le combien de fois on peut couper $E$ en 2.
Combien de fois qu'on peut diviser $n$ par 2 ?

$$\log_2(n)$$

**Definition 3 (Complexite de l'algorithme)**
*La complexite d'un algorithme est le nombre d'instructions elementaires utilisees par l'algorithme dans le pire des cas ( celui qui demande le plus d'instructions).*

C'est une fonction de la taille de l'entree.
Ce qui nous interesse c'est comment le temps pris par l'algorithme evolue au cours du temps.
Ce qui nous interesse est le terme dominant.

## 5.8 Complexite : notation $\Theta$

**Definition 4**
*Pour deux fonctions $f$ et $g$ de $\mathbb{R}$ dans $\mathbb{R}$, on ecrit*

$$f \in \Theta(g)$$

*si et seulement si*

$$\exists c_1, c_2 c_2 \geq c_1 \geq 0$$

## 5.9 differens types de complexite

- constante : $\Theta(1)$
- logarithmique $\Theta(\log n)$
- lineaire $\Theta(n)$

## 5.10 Les tris

Les methodes de tris sont tres importantes en pratique.
Le probleme du tri est egalement un probleme interessant en tant que tel et un bon exemple pour lequel il existe de nombreux algorithmes

On considere une structure de donnees avec une relation d'ordre dessus

**Remarque**
*Un tri ne supprime pas les doublons*
*Quelque soit l'algorithme de tri , un ensemble de donees vide ou reduit a un seul element est deja trie*

On parle de tri interne ( ou "sur place" )

**Exemple (Tri par insertion)**
*entree : un tableau(liste)*
*sortie : le tableau trie*

```
1 Tant que il ya un element mal place
2   on cherche sa place dans le tableau.
3   on deplace lelement labas
```

## 5.11 Probleme de plus court chemin

*Exemples*

- *Calcul du chemin le plus rapide entre toutes les gares du reseau CFF ( 2 a 2)*
- *calcul du chemin le plus rapide entre une gare donnee et toutes les autres gares*
- *calcul du chemin le plus rapide entre deux gares donnees*

*mais aussi*

- *resoudre le rubiks cube*
- *transcrire un texte lu ( reconnaissance de la parole)*
- *corriger les erreurs dans une communication satellite ( codes de convolution)*

## Lecture 5: Programmation

Thu 01 Oct

Comment calculer l'expression suivante sans produire d'erreur

$$\frac{\sqrt{20+7x-x^2}\log(\frac{1}{x+5})}{\frac{x}{10}-\sqrt{\log(x^3-3x+7}-\frac{x^2}{5}}$$

Il faut decomposer

```
1 if ( x + 5.0 ==0){
2 cerr << "expressions invalide pour x=" << x<<endl;
3 return 1;
4 }
```

etc...

```
1 if(x + 5.0 < 0.0) {
2 cerr << "expressions invalide pour x=" << x<<endl;
3 return 1;
4 }
```

Attention, ceci est du copier colle, il auriat fallu poser

```
1 double aux(x+5.0);
2 if (aux ==0){
3 cerr << "expressions invalide pour x=" << x<<endl;
4 return 1;
5 }
6 if(aux < 0.0) {
7 cerr << "expressions invalide pour x=" << x<<endl;
8 return 1;
9 }
10 double resultat(log(1.0/aux));
11 ...
```