

# Conception et analyse des algorithmes

Janvier 2020

Pr. Mohsine Eleuldj

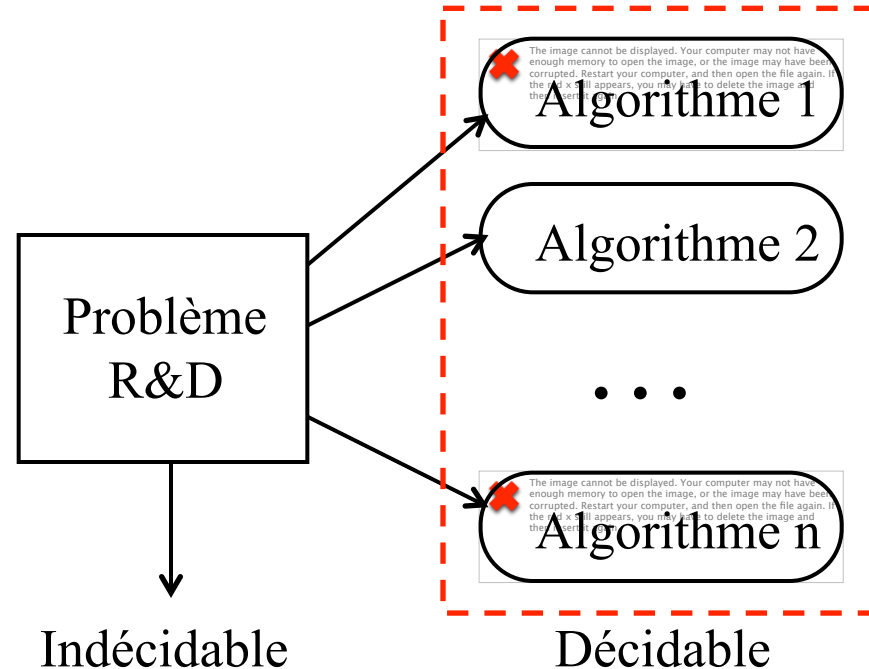
Département Génie Informatique

Ecole Mohammadia d'Ingénieurs

Université Mohammed V Rabat

[eleuldj@emi.ac.ma](mailto:eleuldj@emi.ac.ma)

# Résolution d'un problème



Complexité d'un algorithme : linéaire, quadratique, polynomial, exponentiel, ...

Classes de problèmes décidables (ou calculables) :  $\mathcal{P}$ ,  $\mathcal{NP}$ ,  $\mathcal{NP}$ -complet, ...

Problèmes indécidables (ou incalculables) : problème de l'arrêt, ...

# Objectifs et références

## Objectifs

- Étude des techniques de conception et d'analyse des algorithmes.
- comparaison et classification des algorithmes.
- Ce n'est pas un catalogue d'algorithmes pour la résolution de problèmes spécifiques.

## Références

A. V. Aho, J. E. Hopcroft and J. D. Ullman, «The Design and Analysis of Computer Algorithms»

G. Brassard et P. Bratly, « Algorithmique, conception et analyse »

## Evaluation des connaissances

- Contrôle avec documentation : 80%
- TD/TP : 20%

# Plan

## I Préliminaires

## II Analyse de l'efficacité des algorithmes

diviser 80 élèves sur 4 grp , et choisir 5 de chaque grp

## III Diviser pour régner

basée sur la taille des joueurs = les grandes tailles sont les meilleurs

## IV Algorithmes voraces

greedy algorithm /  
basée sur une heuristique

like diviser pour régner , mais lors du 1er test , on note chacun , ,

## V Programmation dynamique

aller de classe [domaine] > terrain , faire des matches ,  
revenir au class > afficher les résultats

pour ne pas  
répéter  
les tests

## VI Transformation du domaine

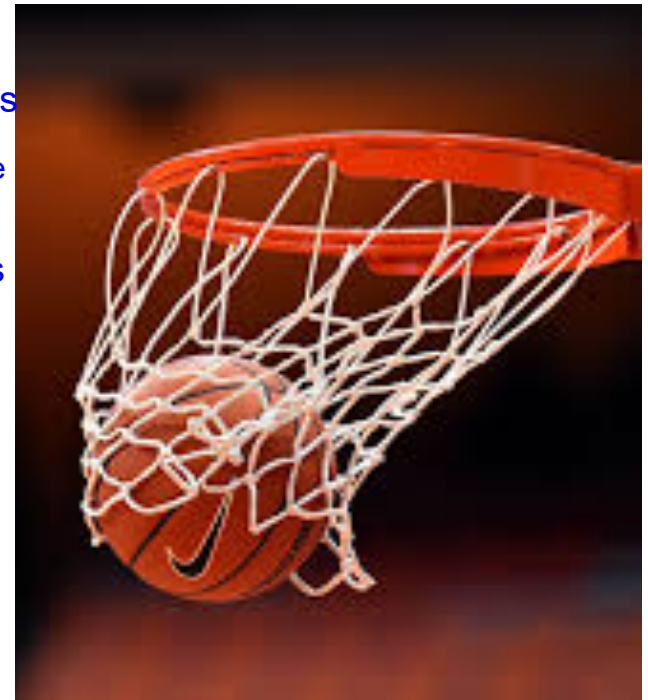
basée sur des probabilités, il peuvent être opposés de la réalité

## VII Algorithmes probabilistes

faire un test culturelle sur basket en premier pas [premier filtre]  
et après on va choisir les joueurs

## VIII Pré-conditionnement

BUT = choisir une équipe de  
basket pour la classe  
=> on a plusieurs algorithmes pour  
un pb donné



# Chapitre 1

## Préliminaires

### Contenu

- 1 Notion d'algorithme
- 2 Efficacité des algorithmes
- 3 Nature de l'analyse
- 4 Pourquoi des algorithmes efficaces ?
- 5 Calcul des nombres de Fibonacci

# Exemple : multiplication des nombres

	decalage a droie /2	*2 = decalage a gache	plus efficace et simple pour la machine = addaptee au machine
53	53	19	19
x 19	<del>26</del>	<del>38</del>	9
477	13	76	<del>4</del>
53	<del>6</del>	<del>152</del>	<del>2</del>
1007	3	304	1
	1	608	
		1007	

2- eliminer les nbr paires

## Analyse des besoins (ressources)

Méthode classique : tables de multiplication + addition

Méthode russe : multiplication par 2 et division par 2 (décalages à droite et à gauche dans une représentation en base 2) + addition

1-

3- des nbr restent sur la colonnes \*2

## Exercice : Décrire l'algorithme de multiplication russe

# 1 Notion d' algorithme

**Origine :** le mot "algorithme" est associé au célèbre auteur Perce Abou Jaafar Mohammed Ibn Moussa Al Khawarizmi connu pour son livre "Al Jabr Wa El Mokabala" écrit en l'an 825.

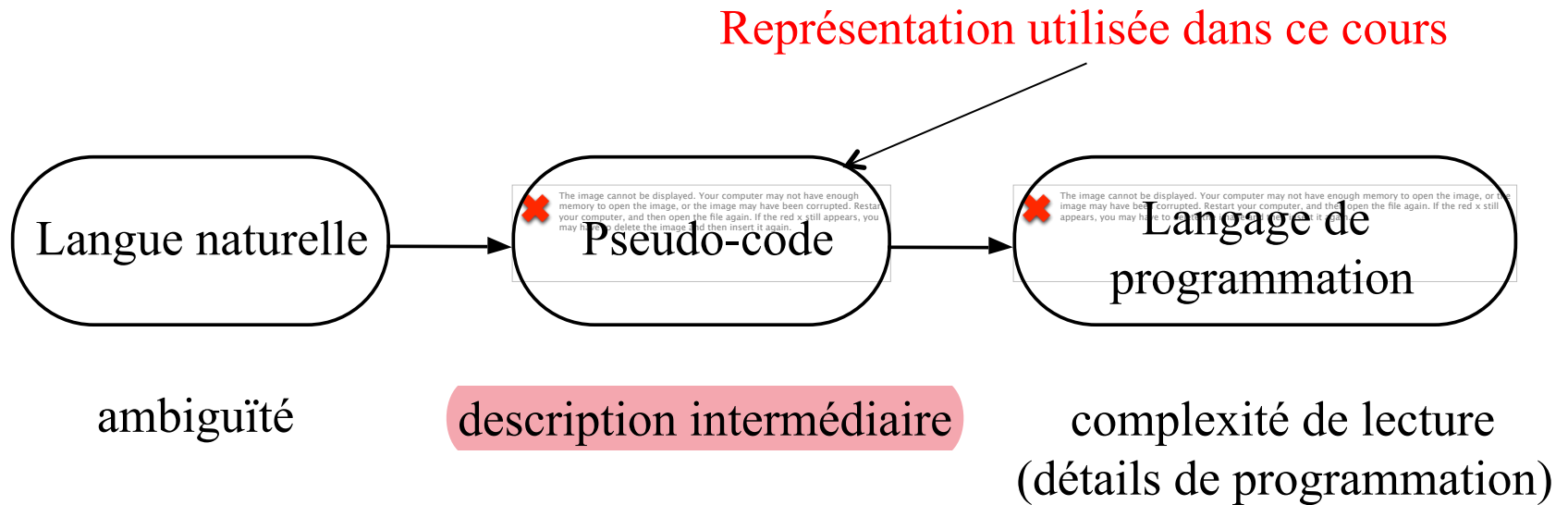
**Définition :** un algorithme est une méthode systématique pour résoudre un problème donné. L'exécution ne doit pas laisser la place à l'interprétation, l'intuition ni la créativité.

**Definition :** l'Algorithmique est l'étude des techniques de conception et d'analyse des algorithmes

## Exemples :

- Multiplication des nombres entiers
- Division
- Calcul du PGCD
- Certaines recettes de cuisines

# Représentation des algorithmes





# Algorithme de multiplication russe

**fonction** russe (a,b)

**tableau** X,Y

$X[0] \leftarrow a, Y[0] \leftarrow b, i \leftarrow 0$  // initialisation

**tant que**  $X[i] > 1$  **faire** // former les 2 colonnes

$X[i + 1] \leftarrow X[i] \text{ div } 2$

$Y[i + 1] \leftarrow Y[i] * 2$

$i \leftarrow i + 1$

$p \leftarrow 0, j \leftarrow 0$

**tant que**  $j < i$  **faire** // addition des entrées appropriées

**si**  $(X[j] \bmod 2 = 1)$  **alors**  $p \leftarrow p + Y[j]$

$j \leftarrow j + 1$   $X[j]$  impaire

**retourner** p

aditionner col de \*2

**Exercice :** Faire la trace pour l'exemple (53,19). Modifier cet algorithme pour avoir une seule boucle et utiliser des variables scalaires.

# Autre algorithme de multiplication russe

pour ne pas enregistrer les valeurs dans un tableau [certains valeurs sont inutiles]

**fonction** autre\_russe (a,b)

entier x,y

$x \leftarrow a, y \leftarrow b, p \leftarrow 0$

// initialisation

**tant que**  $x \geq 1$  **faire**

**si**  $(x \bmod 2 = 1)$  **alors**  $p \leftarrow p + y$  // ajouter la valeur appropriée

$x \leftarrow x \div 2$

$y \leftarrow y * 2$

    boucle infini si x est paire , dont il faut tjrs faire cette operation (faute indentation)

**retourner** p

**Exercice :** Faire la trace pour l'exemple (53,19).

# Autre algorithme de multiplication

**fonction** pas\_russe (A,B)

**tableau** X,Y

$X[1] \leftarrow A, Y[1] \leftarrow B, i \leftarrow 1$  // initialisation

**tant que**  $X[i] > 1$  **faire** // former les 2 colonnes

$X[i + 1] \leftarrow X[i] - 1$

$Y[i + 1] \leftarrow B$

$i \leftarrow i + 1$

il ressemble beaucoup a Russe

$P \leftarrow 0$

**tant que**  $i > 0$  **faire** // additionner les entrées appropriées

**si**  $X[i] > 0$  **alors**  $P \leftarrow P + Y[i]$

$i \leftarrow i - 1$

**retourner** P

**Exercice :** Faire la trace pour l'exemple (53,19). Quelle est la particularité de cette multiplication ? Ecrire l'algorithme classique de multiplication.

# Algorithme de multiplication classique

```
fonction mult_classique(tableau A,B)
    {ChiffresA, ChiffreB : nombre de chiffres de A et B}
    Produit  $\leftarrow$  0
    pour i = 1 à ChiffresA faire           { former les 2 colonnes }
        ProduitPartiel  $\leftarrow$  0
        pour j = 1 à ChiffresB faire
            ProduitPartiel  $\leftarrow$  ProduitPartiel + A[i]*B[j] *  $10^{j-1}$ 
            Produit  $\leftarrow$  Produit + ProduitPartiel* $10^{i-1}$ 
            c'est un peu nouveau
    retourner produit
```

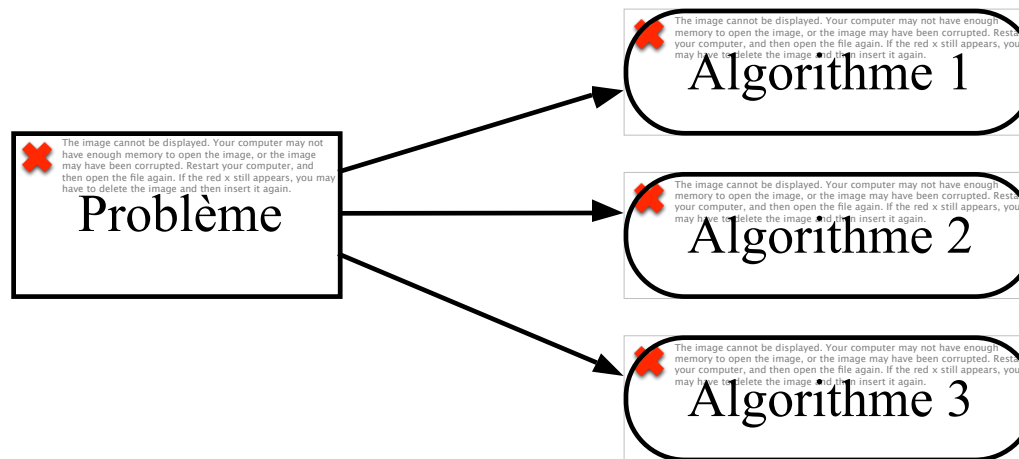
**Exercice :** Faire la trace pour l'exemple (53,17).

# Etapes de résolution d'un problème

projet bg data = optimisation memoire

les jeux - chats ,, temps de reponses

- 1 Trouver différents algorithmes (selon différentes méthodes de conception)
- 2 Analyser leur efficacité en fonction des ressources (temps de réponse, espace mémoire, communication, ...)
- 3 Choisir le meilleur algorithme (selon la taille de l'exemplaire du problème, espace mémoire, puissance de calcul, débit des communications,...)



## 2 Efficacité des algorithmes

**Définition :** Un exemplaire  $x$  est l'entrée (input) d'un algorithme,  $|x|$  est la taille de l'exemplaire  $x$

### Exemples

- Tri :  $|x|$  est le nombre d'entiers à ordonner
- Multiplication :  $|x|$  est le nombre de chiffres (ou bits) des facteurs

### Approches d'analyse

- Empirique : programmation + exécution avec plusieurs exemplaires
- Théorique : déterminer la quantité de ressources (temps, mémoire,...) en fonction de la taille des exemplaires

### Avantages de l'approche théorique

- Indépendance de l'ordinateur et langage de programmation
- Gain dans la programmation et l'exécution des algorithmes inefficaces
- Taille des exemplaires n'est pas une contrainte

# 3 Nature de l'analyse

## Efficacité d'un algorithme dépend

- taille de l'exemplaire [pire cas - meilleur cas - cas moyen]
- puissance de calcul
- espace mémoire
- débit des communication
- ...

## Comparaison des algorithmes selon différentes analyses

- meilleur cas (optimiste)
- moyenne
- pire cas (pessimiste)

# Tri par insertion

meilleur cas = 0 permutation

pire cas =  $n(n-1)/2$  permutation

## 3 Nature de l'analyse

on prend a chaque fois un element et on l'insere

Efficacité d'un algorithme dépend

- taille de l'exemplaire
- puissance de calcul
- espace mémoire
- débit des communication
- ...



**fonction** TriInsertion (T[1..n])

**pour**  $i \leftarrow 2$  jusqu'à  $n$  **faire**

$x \leftarrow T[i]$

$j \leftarrow i - 1$

**tant que**  $j > 0$  et  $T[j] > x$  **faire**

$T[j + 1] \leftarrow T[j]$

$j \leftarrow j - 1$  decrementer  $j$

$T[j + 1] \leftarrow x$

tant que l'elt suiv est superieur a l'elt courant => permuteer

Comparaison des algorithmes selon différentes analyses

- meilleur cas (optimiste)
- moyenne
- pire cas (pessimiste)

**Exercice :** Faire la trace pour

$T = [3, 1, 4, 0, 5]$ ,  $U = [0, 1, 3, 4, 5]$  et  $V = [5, 4, 3, 1, 0]$



# Trace de TriInsertion(T)

i	x	j	$j > 0$ et $T[j] > x$	T
-	-	-	-	3 1 4 0 5
2	1	1	oui	
		0	non	<b>1 3</b> 4 0 5
3	4	2	non	
4	0	3	oui	
		2	oui	1 3 <b>0 4</b> 5
		1	oui	1 <b>0 3</b> 4 5
		0	non	<b>0 1</b> 3 4 5
5	5	4	non	0 1 3 4 5

# Trace de TriInsertion(U) et (V)

i	x	j	U
-	-	-	0 1 3 4 5
2	3	1	
3	4	2	
4	6	3	
5	9	4	

i	x	j	V
-	-	-	5 4 3 1 0
2	6	1	
		0	<b>4 5</b> 3 1 0
3	4	2	
		1	4 <b>3 5</b> 1 0
		0	<b>3 4</b> 5 1 0
4	3	3	
		2	3 4 <b>1 5</b> 0
		1	3 <b>1 4</b> 5 0
		0	<b>1 3</b> 4 5 0
5	0	4	
		3	1 3 4 <b>0 5</b>
		2	1 3 <b>0 4</b> 5
		1	1 <b>0 3</b> 4 5
		0	<b>0 1</b> 3 4 5

# Analyse du Tri par insertion

## Trace

U : meilleur cas pour TriInsertion

V : pire des cas pour TriInsertion

## Conclusion

Nous allons effectuer une analyse au pire cas

Par la suite, nous allons voir que cet algorithme est quadratique (Ordre de  $n^2$ , où  $n$  est la taille de l'exemplaire  $x$ )

# 4 Pourquoi des algorithmes efficaces ?

## Hypothèse

A : un algorithme

M : une machine

A' : un algorithme plus efficace que A

M' : une machine plus puissante que M

**Question :** A sur M' ou A' sur M ?

## Autrement si

n : taille de l'exemple

t(n) : temps d'exécution de A sur M

t'(n) : temps d'exécution de A sur M'

t''(n) : temps d'exécution de A' sur M

**Question :**  $t'(n) < t''(n)$  ou  $t''(n) < t'(n)$  ?

deja avec un algorithme plus efficace , on est gagnant  
=> independamment de puissance de machine

selon la situation :  
algorithme si il est trouve est plus efficace generalement ,  
mais pas de garantit qu'on va le trouver  
[longe duree de recherche...  
machine plus fiable en terme d'optimisation ,  
mais il marche generalemnt pour les cas presse  
par le temos

# Application numérique

$$t(n) = 10^{-4} \times 2^n \text{ s}$$

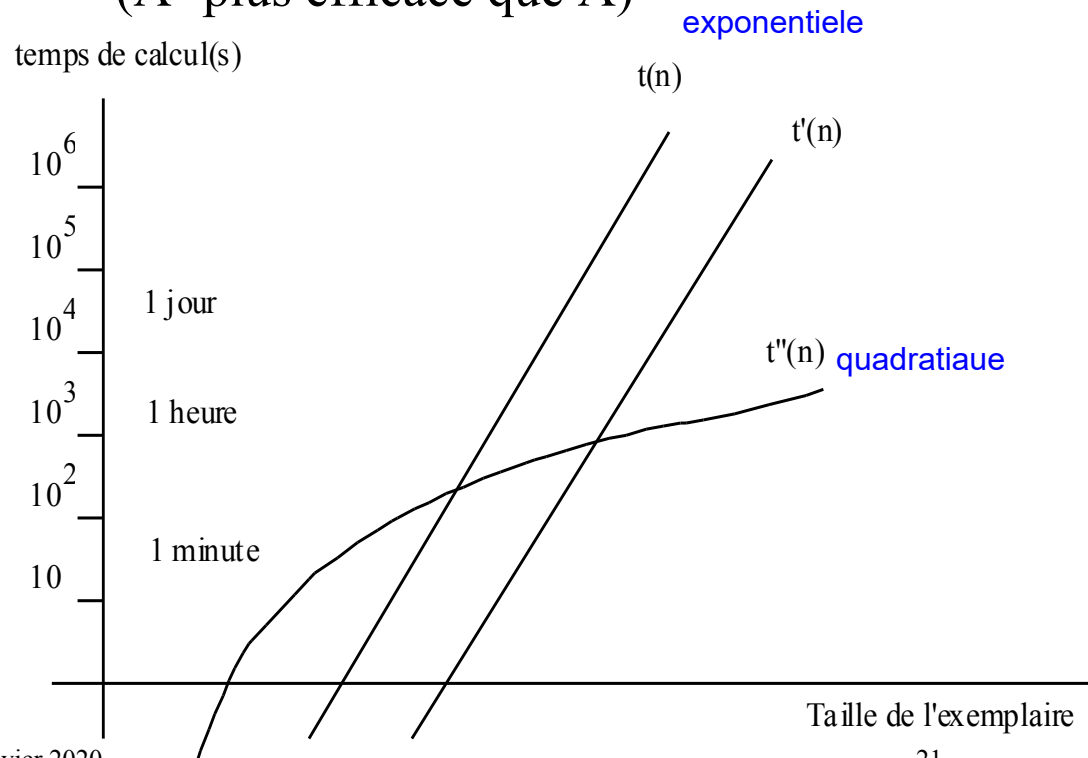
$$t'(n) = t(n) \times 10^{-2} = 10^{-6} \times 2^n \text{ s}$$

$$t''(n) = 10^{-2} \times n^3 \text{ s}$$

(M' 100 plus rapide que M)

(A' plus efficace que A)

n	t(n)	t'(n)	t''(n)
10	1/10 s	2 ms	10 s
20	2 mn	1 s	1mn
30	10 jours	3 heures	5 mn
38	1 année	4 mois	10 mn
45	-	1 année	20 mn
200	-	-	1 jour
1500	-	-	1 année



# Nombres de Fibonacci (1/3)

## Définition

$$f_0 = 0, f_1 = 1$$

$$f_n = f_{n-1} + f_{n-2} \text{ pour } n \geq 2$$

## De Moivre

$$f_n = 1/\sqrt{5} [ \Phi^n - (-\Phi)^{-n} ]$$

$$\text{où } \Phi = (1 + \sqrt{5})/2$$

appelé le nombre d'or

## Remarque

Cette formule n'est pas pratique pour calculer la valeur exacte de  $f_n$

# Nombres de Fibonacci (2/3)


**fonction fib1(n)**

**si**  $n < 2$  **alors retourner**  $n$

**sinon retourner**  $\text{fib1}(n-1) + \text{fib1}(n-2)$

formule de moivre

arbre  $\Rightarrow$  tmps exp

Fib1 base sur DPR  
il est recursive , et  
pour trouver une  
valeur dans la  
sommet :  
il descend 

**fonction fib2(n)**

$i \leftarrow 1, j \leftarrow 0$

**pour**  $k \leftarrow 1$  **jusqu'à**  $n$  **faire**

$j \leftarrow i + j$

$i \leftarrow j - i$

**retourner**  $j$

complexite  $n$

Fib2 base sur la  
programmation dynamique

on part des valeurs petites ,  
pour trouver des  
grand valeur

**Exercice :** Faire la trace de  $\text{fib1}(7)$  et  $\text{fib2}(7)$

# Calcul des nombres de Fibonacci (3/3)

**fonction** fib3(n)

$i \leftarrow 1, j \leftarrow 0, k \leftarrow 0, h \leftarrow 1$

**tant que**  $n > 0$  **faire**

**si**  $n$  est **impair** **alors**

$t \leftarrow jh$

$j \leftarrow ih + jk + t$

$i \leftarrow ik + t$

$t \leftarrow h^2$

$h \leftarrow 2kh + t$

$k \leftarrow k^2 + t$

$n \leftarrow n \text{ div } 2$

**retourner**  $j$

$n = 2^{\text{puissance}(k)}$   
 $C = \log(n)$   
logarithmique

Fib3 aussi basé sur la DPR, et il  
implémente la multiplication matricielle

**Exercice :** Faire la trace de fib3(7)



# Temps d'exécution\* de fib1, fib2 et fib3

n	10	20	30	40	$10^3$	$10^5$	$10^6$	$10^7$	$10^8$
<b>fib1</b>	208 $\mu$ s	9,9 ms	1,67 s	3mn 9s	(_) = impossible de le faire ici -	-	-	-	-
<b>fib2</b>	2,52 $\mu$ s	32 $\mu$ s	35 $\mu$ s	41 $\mu$ s	9,9 ms	1,38 s	1 mn 41 s	-	-
<b>fib3</b>	4,31 $\mu$ s	50 $\mu$ s	55 $\mu$ s	52 $\mu$ s	0,1 ms	29 ms	0,93 s	40,7 s	23 mn

\*programme écrit en Python 3.4.7 sous Windows 7 Pro exécuté sur i7

# Chapitre 2

## Analyse de l'efficacité des algorithmes

### Contenu

- 1 Notations asymptotiques
- 2 Analyse des algorithmes itératifs
- 3 Résolution d'équations de récurrences
- 4 Analyse des algorithmes récursifs

# 1 Notations asymptotiques

**Remarque :** L'analyse théorique de l'efficacité d'un algorithme se fait à une constante près pour ne pas tenir compte de :

- langage de programmation
- compilateur et système d'exploitation
- puissance de l'ordinateur

## Notation "l'ordre de"

ordre de  $t$  = ensemble de fct à partir d'un certain rang, ils vont être majorés par  $c \cdot f$

Soit  $f : \mathbb{N} \rightarrow \mathbb{R}^*$

$$O(f(n)) = \{t : \mathbb{N} \rightarrow \mathbb{R}^* / (\exists c \in \mathbb{R}^+) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) [t(n) \leq c f(n)]\}$$

$n$ =taille d'exemple

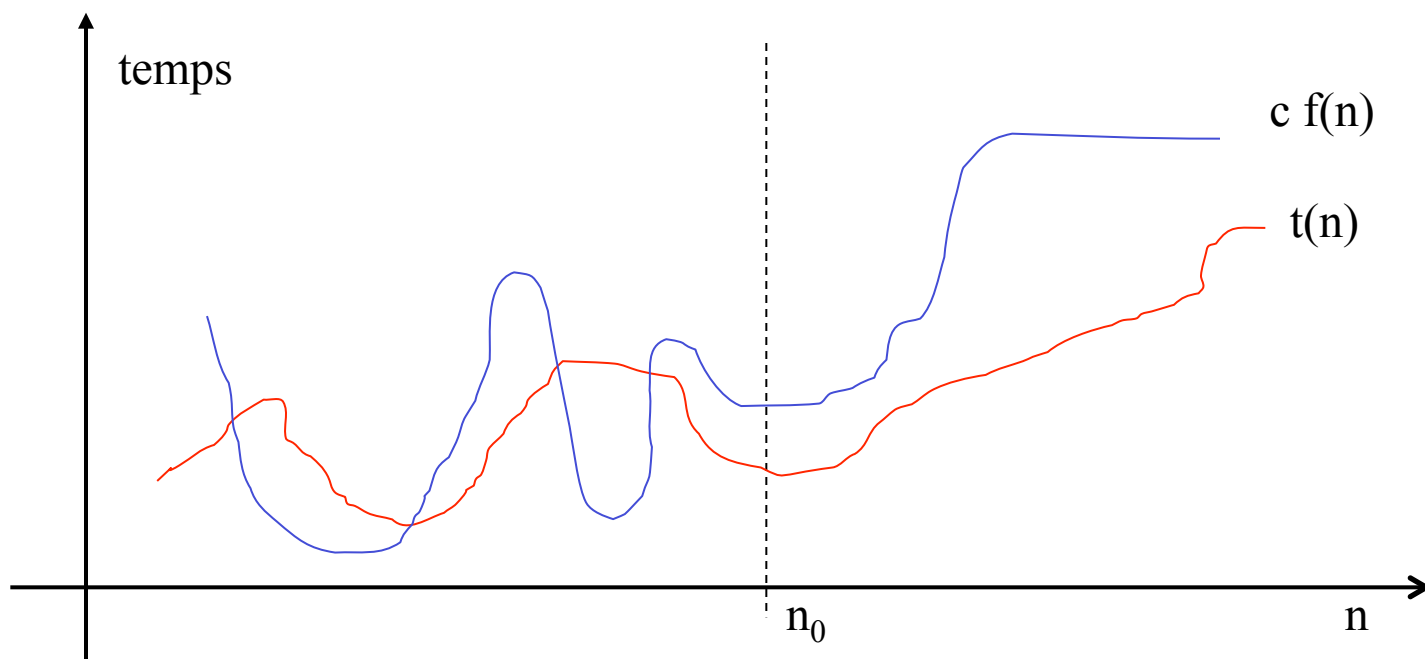
## Définitions

$O(f(n))$  est appelé l'ordre de  $f(n)$

$t(n) \in O(f(n)) \Rightarrow t(n)$  est dans l'ordre de  $f(n)$

$t(n)$  : temps d'exécution d'un algorithme  $\Rightarrow$  algorithme est de l'ordre de  $f(n)$

$$t(n) \in O(f(n))$$



# Exercices

1) Quel est l'ordre de l'algorithme qui prend un temps borné supérieurement par :

$$t(n) = 3 \text{ s} - 18n \text{ ms} + 27n^2 \mu\text{s}$$

2) Prouver que :  $f(n) \in O(g(n))$  et  $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$

3) Dédurre que  $g(n) \in O(h(n)) \Rightarrow O(g(n)) \subset O(h(n))$

4) Soient  $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$ , montrer que :

$$O(f(n) + g(n)) = O(\max(f(n), g(n)))$$

5) Soient  $f$  et  $g : \mathbb{N} \rightarrow \mathbb{R}^+$ , prouver que :

$$\lim_{n \rightarrow \infty} f(n) / g(n) = c \in \mathbb{R}^+ \Rightarrow O(f(n)) = O(g(n))$$

$$\lim_{n \rightarrow \infty} f(n) / g(n) = 0 \Rightarrow O(f(n)) \subset O(g(n))$$

rectif:  $\mathbb{R}^*$ , car pour l'inclusion inverse le  $1/c$ -A doit être positive, donc on doit choisir un  $c$  positif et pas nul

6) Prouver que :  $\log n \in O(\sqrt{n})$  et  $\sqrt{n} \notin O(\log n)$

7) Soit  $x \in \mathbb{R} / 0 < x < 1$ . Utiliser  $\subset$  et  $=$  pour mettre en rang les ordres des fonctions suivantes:

$$\underset{1}{n \log n}, \underset{4}{n^8}, \underset{2}{n^{1+x}}, \underset{6}{(1+x)^n}, \underset{4}{(n^2 + 8n + \log^3 n)^4} \text{ et } \underset{3}{n^2 / \log n} \quad \lim_{n \rightarrow +\infty} (n^{1+x}) / (n \log n) = +\infty$$

# Autres notations asymptotiques

**Remarque :** la notion d'ordre est l'estimation d'une limite supérieure du temps d'exécution d'un algorithme sur un exemplaire de taille donnée. Nous allons estimer une limite inférieure.

## Omega de $f(n)$

$$\Omega(f(n)) = \{t : N \rightarrow R^* / (\exists c \in R^+) (\exists n_0 \in N) (\forall n \geq n_0) [t(n) \geq c f(n)]\}$$

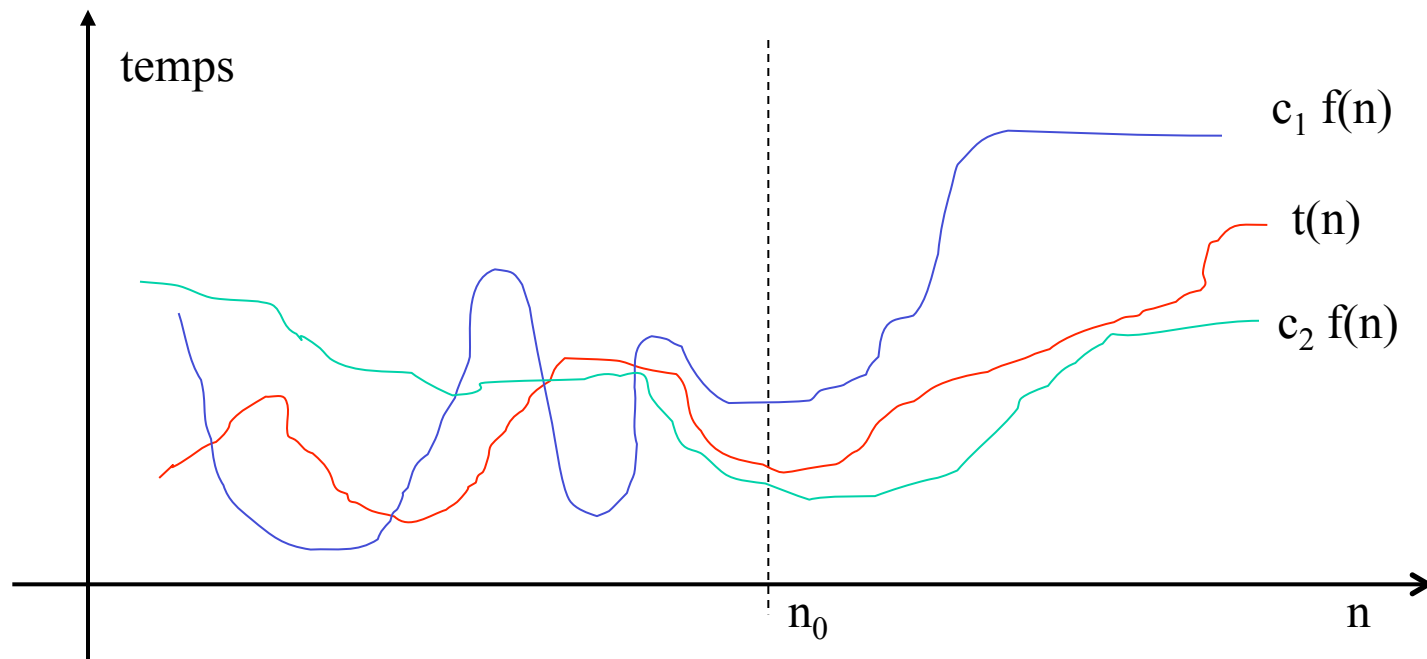
**Exercice :** Soient  $f, g : N \rightarrow R^*$ , montrer que :

$$f(n) \in O(g(n)) \Rightarrow g(n) \in \Omega(f(n))$$

## Ordre exact de $f(n)$

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

$$t(n) \in \Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$



## 2 Analyse des algorithmes itératifs

Soit  $t(n)$  : temps d'exécution de l'algorithme pour un exemplaire de taille  $n$

Algorithmes analysés

- Calcul de Fibonacci

- Tri par selection

- Calcul du PGCD



# Analyse de fib2

**fonction** fib2(n)

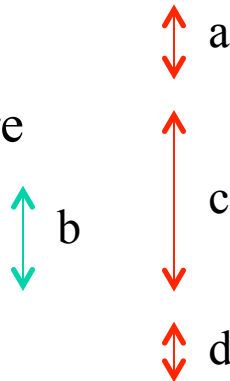
$i \leftarrow 1, j \leftarrow 0$

**pour**  $k \leftarrow 1$  **jusqu'à**  $n$  **faire**

$j \leftarrow i + j$

$i \leftarrow j - i$

**retourner**  $j$



$$t(n) = a + c + d = a + \left(\sum_{1 \leq k \leq n} b\right) + d = a + bn + d = a' + bn \quad \text{où } a' = a + d$$
$$\Rightarrow t(n) \in O(n)$$

# Analyse de fib3

**fonction** fib3(n)

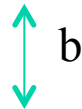
$i \leftarrow 1, \dots$

**tant que**  $n > 0$  **faire**

...

$n \leftarrow n \text{ div } 2$

**retourner** j



13=1101

on divise par 2 = eleminer un bit

13/2=6=110

6/2=3=11

3/2=1=1

1101 = 3 bit de 1

$\log_2(13)=3,7$

partie entiere =  $E(\log_2(13))=3$

$t(n) = a + c + d = a + bk + d$  où k est le nombre d'itérations

k = nombre de bits dans la représentation binaire de n  $\Rightarrow k = \lceil \log_2 n \rceil + 1$

$\Rightarrow t(n) \in O(\log n)$  c'est la partie entiere de k

# Analyse de select

**Procédure** select (T[1...n])

**pour** i ← 1 **jusqu'à** n-1 **faire**

    minj ← i, minx ← T[i]

**pour** j ← i+1 **jusqu'à** n **faire**

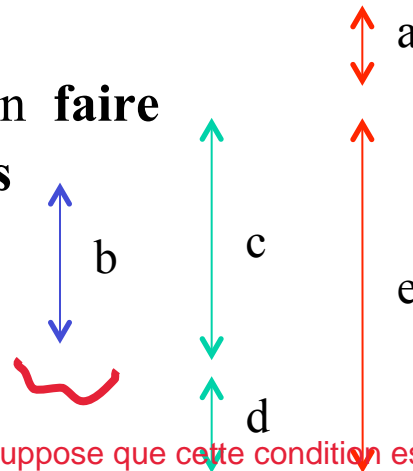
**si** T[j] < minx **alors**

            minj ← j

            minx ← T[j]

    T[minj] ← T[i]

    T[i] ← minx



tjrs on suppose que cette condition est vrai => pire cas

$$t(n) = \sum_{1 \leq i \leq n-1} [ a + \sum_{i+1 \leq j \leq n} (b) + d ] = (a + d + bn)(n - 1) - bn(n-1)/2$$

$$\Rightarrow t(n) \in O(n^2)$$

# Analyse de l'algorithme d'Euclide (1/2)

calcul du PGCD de deux nombres

```
fonction Euclide(m,n)
    tant que m > 0 faire
        t ← n mod m
        n ← m
        m ← t
    retourner n
```

Exercice : Faire la trace pour 8 et 12. Quel est l'ordre de cet algorithme ?

# Analyse de l'algorithme d'Euclide (2/2)

Montrons tout d'abord que : " $n, m / n \_ m \_ n \bmod m < n/2$

Cas 1 :  $m > n/2 \_ 1 \_ n/m < 2 \_ [n/m] = 1 \_ n \bmod m = n - m \_ n - n/2 = n/2$

Cas 2 :  $m \_ n/2 \_ n \bmod m < m$

Soit  $k$  le nombre d'itérations de la boucle pour l'exemplaire  $(m,n)$ . Soient  $m_i$  et  $n_i$  les valeurs de  $m$  et  $n$  après la  $i$ ème itération. On déduit que  $m_k = 0$  et le système suivant :

$$n_i = m_{i-1}$$

$$m_i = n_{i-1} \bmod m_{i-1}$$

$$n_0 = n \text{ et } m_0 = m$$

On peut vérifier que  $n_i > m_i$  pour  $i \_ 1. \Rightarrow m_i = n_{i-1} \bmod m_{i-1} < n_{i-1}/2 = m_{i-2}/2$

Supposons que  $k$  est impair. Soit  $d$  tel que  $k = 2d + 1$  alors :

$$m_{k-1} < m_{k-3}/2 < m_{k-5}/2^2 < \dots < m_0/2^d$$

Or  $m_{k-1} \_ 1 \_ m_0/2^d \_ 1 \_ d \in \log m > k \in 2\log m + 1$

le cas où  $k$  est pair est traité de la même  $\Rightarrow t(n) \in O(\log m)$ .

# Analyse de fib1

$t(n) = t(n-1) + t(n-2) + b$  on calcule le temps d'exécution  
 $t(0)=t(1)=a$

**fonction** fib1(n)

**si**  $n < 2$  **alors retourner** n

← a

condition d'arrêt

**sinon retourner** fib1(n-1) + fib1(n-2)

└──────────────────┘  
b

$t(n)$  est la solution du système de récurrence :

$$t(0) = t(1) = a$$

$$t(n) = t(n-1) + t(n-2) + b$$

# 3 Résolution de récurrences

## récurrence homogène

cad terme droite=null =0

$$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = 0 \quad (1)$$

cherchons une solution de la forme  $t_n = x^n$  (équation caractéristique)

$$a_0 x^n + a_1 x^{n-1} + \dots + a_k x^{n-k} = 0 \quad (2)$$

cherchons une solution non nulle  $\Rightarrow$  (2) devient :

$$a_0 x^k + a_1 x^{k-1} + \dots + a_k = 0$$

supposons que les  $k$  racines :  $r_1, r_2, \dots, r_k$  (réelles ou complexes) sont distinctes alors :

$$t_n = \sum_{1 \leq i \leq k} c_i (r_i)^n$$

où  $c_i$  ( $1 \leq i \leq k$ ) constantes déterminées par les conditions initiales

# Calcul du nombre de Fibonacci

$$f_0 = 0 \text{ et } f_1 = 1 \quad (1)$$

$$f_n = f_{n-1} + f_{n-2} \quad (2)$$

Posons  $f_n = x^n$ , alors (2) devient :  $x^n = x^{n-1} + x^{n-2} \Rightarrow x^2 - x - 1 = 0$

$$\Delta = 1 + 4 = 5 \Rightarrow r_1 = (1 - \sqrt{5})/2 \text{ et } r_2 = (1 + \sqrt{5})/2$$

$$\Rightarrow f_n = c_1 r_1^n + c_2 r_2^n$$

En utilisant les conditions initiales (1) on trouve :

$$c_1 + c_2 = 0$$

$$c_1 r_1 + c_2 r_2 = 0 \quad \Rightarrow \quad c_1 = -1/\sqrt{5} \text{ et } c_2 = 1/\sqrt{5}$$

$$\Rightarrow f_n = 1/\sqrt{5} [ -((1 - \sqrt{5})/2)^n + ((1 + \sqrt{5})/2)^n ] \quad (\text{Méthode de Moivre})$$



# Analyse de fib1

## Solution de l'EH + solution particulière

$$t(0) = t(1) = a \quad (1)$$

$$t(n) = t(n-1) + t(n-2) + b \quad (2)$$

D'après le calcul précédent on déduit que la solution de l'EH :

$$t(n) = c_1 r_1^n + c_2 r_2^n \quad \text{où } r_1 = (1 - \sqrt{5})/2 \text{ et } r_2 = (1 + \sqrt{5})/2$$



$t(n)$  appartient a l'ordre  $O(c_1 r_1^{**n} + c_2 r_2^{**n}) = \max(\text{des deux}) = O(c_2 r_2^{**n})$  si  $r_2 > 1 \Rightarrow$  croissante  
sinon vis versa

En utilisant les conditions initiales (1), on trouve le système :

$$c_1 + c_2 = a$$

$$c_1 r_1 + c_2 r_2 = a$$

$$\Rightarrow c_1 = -a (1 + \sqrt{5})/2\sqrt{5} \text{ et } c_2 = a (1 + 3\sqrt{5})/2\sqrt{5}$$

$\Rightarrow t(n) \in O(c_2^n)$  où  $|c_2| > 1 \Rightarrow$  la solution de l'EH est exponentielle

$\Rightarrow$  la solution générale est exponentielle

# Autre Analyse de fib1

$$t(0) = t(1) = a \quad (1)$$

$$t(n) = t(n-1) + t(n-2) + b \quad (2)$$

$$t(n) = t(n-1) + t(n-2) + b$$

$$t(n-1) = t(n-2) + t(n-3) + b \quad (2) \text{ pour } n-1 \Rightarrow (3)$$

$$t(n) - t(n-1) = \cancel{t(n-2)} + t(n-3) - t(n-1) - \cancel{t(n-2)} \quad (2)-(3)$$

$$t(n) - t(n-1) = \cancel{t(n-2)} + t(n-3)$$

$\Rightarrow$  equation caracteristique  $t(n)=x^{**}n ..$

$$X^3 - 2X^2 + 1 = 0$$

$$(X - 1)(X^2 - X - 1) =$$

$\Rightarrow t(n) \in O(\cancel{s_2}^n)$  où  $|\cancel{c_2}| > 1 \Rightarrow$  la solution de l'EH est exponentielle

### 3 Résolution de récurrences récurrences non homogènes

Illustrons ce type de récurrence à l'aide de l'exemple suivant où l'on se ramène à une récurrence homogène

$$t_n - 2t_{n-1} = 3^n \quad (1)$$

En multipliant (1) par 3 et en la considérant pour n+1 on trouve:

$$3t_n - 6t_{n-1} = 3^{n+1} \quad (2)$$

$$t_{n+1} - 2t_n = 3^{n+1} \quad (3)$$

on prend (1) en on passe au terme suivant [n+1]

En faisant la différence de (2) et (3), on trouve :

$$t_{n+1} - 5t_n + 6t_{n-1} = 0$$

L'équation caractéristique de cette équation est :

$$x^2 - 5x + 6 = 0 \Leftrightarrow (x - 2)(x - 3) = 0 \quad r1=2 \quad r2=3$$

$$\Rightarrow t_n = c_1 2^n + c_2 3^n$$

En utilisant (1), on trouve que  $t_n = c_1 2^n + 3^{n+1}$

# 3 Résolution de récurrences récurrences non homogènes

Illustrons cette technique à l'aide de l'exemple suivant où nous faisons une transformation de variable :

$$T(n) = 4T(n/2) + n, \quad n > 1 \text{ avec } n = 2^k \quad (1)$$

Posons  $t_k = T(2^k)$ . L'équation (1) devient : on fait ici une transformation de var

$$t_k = 4t_{k-1} + 2^k$$

En multipliant par 2 et en considérant l'équation pour  $n+1$ , on trouve :

$$2t_k = 8t_{k-1} + 2^{k+1} \quad (2)$$

$$t_{k+1} = 4t_k + 2^{k+1} \quad (3)$$

$$(2) - (3) \text{ donne : } t_{k+1} - 2t_k = 4t_k - 8t_{k-1} \Leftrightarrow t_{k+1} - 6t_k + 8t_{k-1} = 0$$

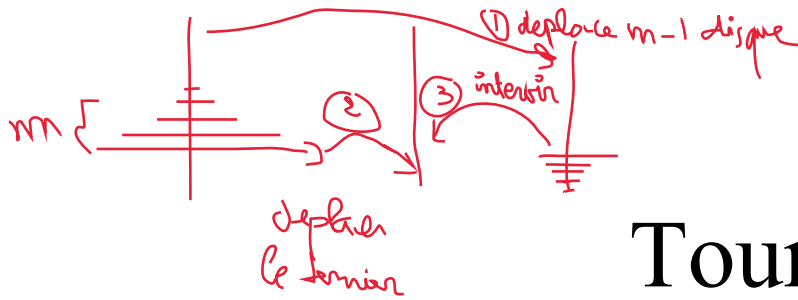
L'équation caractéristique de cette équation est :

$$x^2 - 6x + 8 = 0 \Leftrightarrow (x - 2)(x - 4) = 0 \text{ _ donc } t_k = c_1 2^k + c_2 4^k$$

$\Leftrightarrow$

$$\text{donc } T(n) = c_1 n + c_2 n^2$$

En utilisant (1) on trouve que  $c_1 = -1$  et par conséquent :  $T(n) = -n + c_2 n^2$



$$d(m) = d(m-1) + 1 + d(m-1)$$

① ② ③

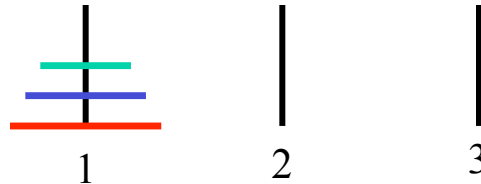
# Tours de Hanoï

[ *le genre de fin du monde* ]

## Problème

Soient trois aiguilles (1, 2 et 3) et  $m$  disques, tous de taille différente. Au départ tous les disques sont placés du plus grand au plus petit dans l'aiguille 1. Comment déplacer les disques à l'aiguille 2 sans jamais mettre de disque par-dessus un disque plus petit dans les aiguilles ?

## Exercice



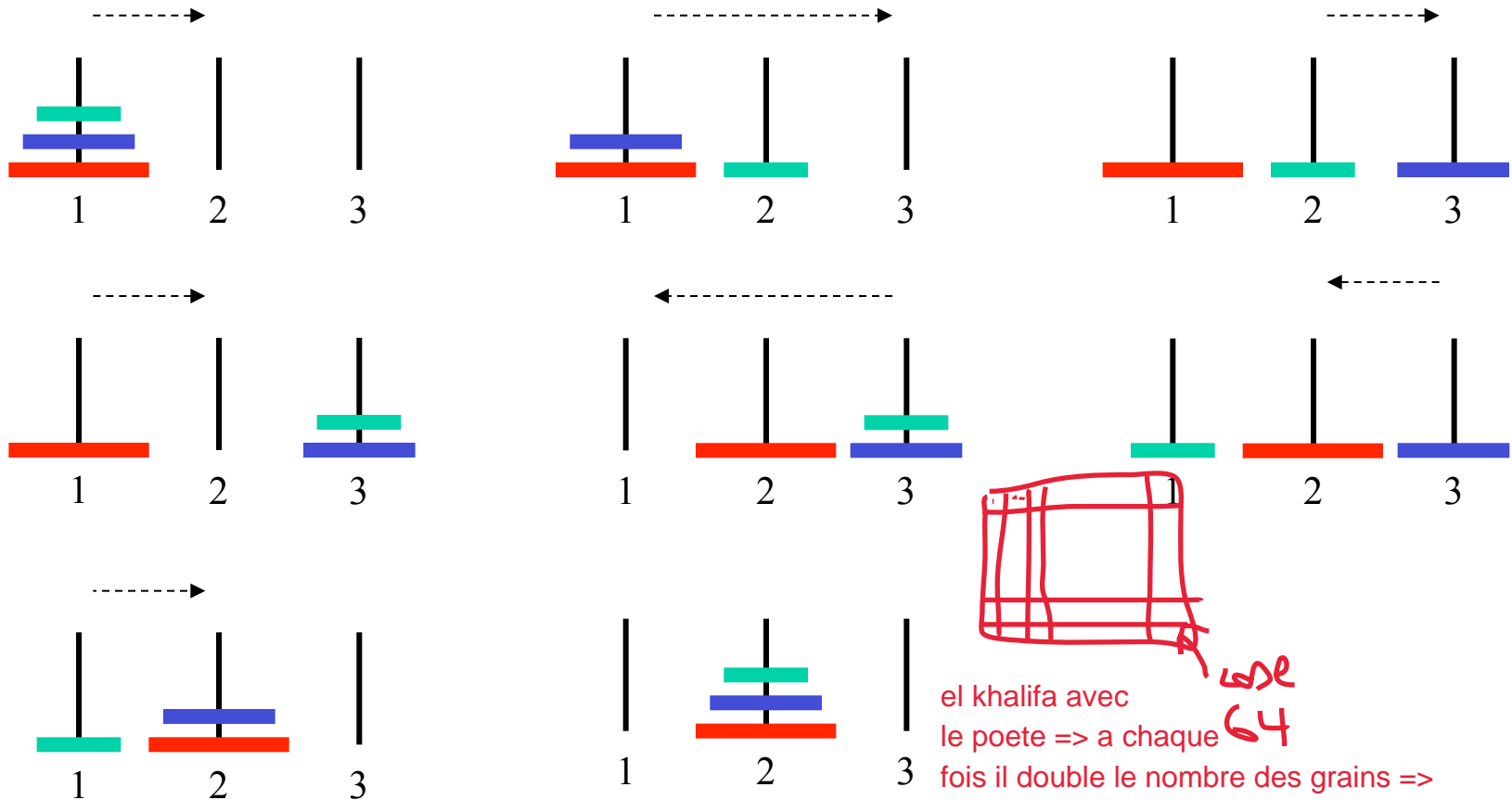
- 1 Comment allez-vous faire pour déplacer 3 disques ?
- 2 Décrire un algorithme de la solution.
- 3 Faire la trace pour  $m = 3$ .
- 4 Déterminer le nombre de déplacements en fonction de  $m$ .
- 5 Dédire l'efficacité de l'algorithme. *analyse*
- 6 Estimer le temps si  $m = 64$  et si un déplacement prend 1 seconde.
- 7 Démontrer l'optimalité de l'algorithme.

$$t = (2^{64} - 1) \times 1s$$

= 500 Miliare Année

*cad est ce que on peut faire moins de déplacement si on deplace les m-1 vers 2, on ne peut pas simplement avoir le miene disque dans ca place*

# Trace pour $m = 3$



el khalifa avec  
le poete => a chaque  
fois il double le nombre des grains =>  
 $S = 1(2^{**0}) + 2 + 4 + 8 \dots + 2^{**63}$   
 $2S = 2 + 4 + 8 + \dots + 2^{**64}$   
 $2S - S = 2^{**64} - 1 \sim 16 * 10^{**19} \Rightarrow \text{impossible}$   
[miliare et miliare de tons]

# Algorithme des tours de Hanoï

m disque

i, j sont Numero d'eguille

**procédure** Hanoï (m,i,j)

**si** m > 0 **alors**

Hanoï (m-1,i,6-i-j)

écrire (i,"-->",j)

Hanoï (m-1,6-i-j,j)

deplacer les m-1 disques  
de i a k ( il rest que le m iene )

prend moins du temps



$k = 3$  eguille

i,j,k appartient a {1,2,3}  
 $i+j+k=6 \Rightarrow k=6-i-j$

**Exercice :** Soit d(m) le nombre de déplacements

$$d(0) = 0$$

$$d(m)=0$$

$$d(m)=2d(m-1)+1$$

$$d(m) = 2d(m-1) + 1$$

Montrer que  $d(m) = 2^m - 1$

demonstration direct

par induction (recurrence) :

1-verifier pour 0,1

2- on suppose qu'il est bon pour n

3- on demontre pour n+1

$$d(m)=2d(m-1)+1$$

$$2* d(m-1)=2d(m-2)+1$$

..

$$2^{**m-1} * d(1)=2d(0)+1$$

$$d(0)=0$$

47

$$d(m)=1+2+...+2^{**m-1} = \frac{2^{**m} - 1}{2 - 1}$$

# Analyse de l'algorithme

$t(m)$  : temps d'exécution de l'algorithme sur un exemplaire de taille  $m$ .

$$t(0) = a$$

$$t(m) = 2 t(m-1) + b$$

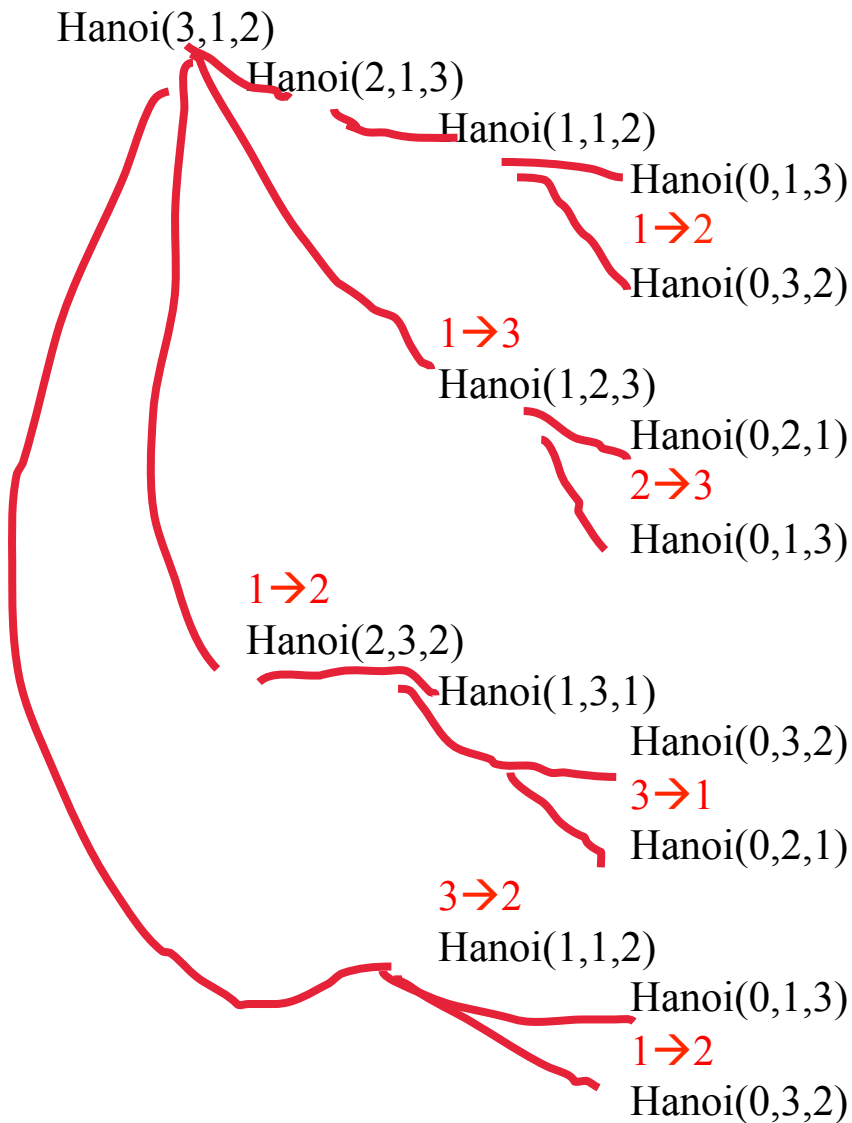
$$\begin{aligned} t(m) &= 2 t(m-1) + b = 2 [ 2 t(m-2) + b ] + b = 2^2 t(m-2) + b (2 + 1) \\ &= 2^2 [ 2 t(m-3) + b ] + b (2 + 1) = 2^3 t(m-3) + b (2^2 + 2 + 1) \\ &= \dots\dots\dots \\ &= 2^k t(m-k) + b (2^{k-1} + \dots + 2 + 1) \\ &= \dots\dots\dots \\ &= 2^m t(0) + b (2^{m-1} + \dots + 2 + 1) = a 2^m + b (2^{m-1} + \dots + 2^1 + 2^0) \\ &= a 2^m + b 2^m / (2 - 1) = a 2^m + b (2^m - 1) \end{aligned}$$

donc  $t(m) \in O(2^m)$ .

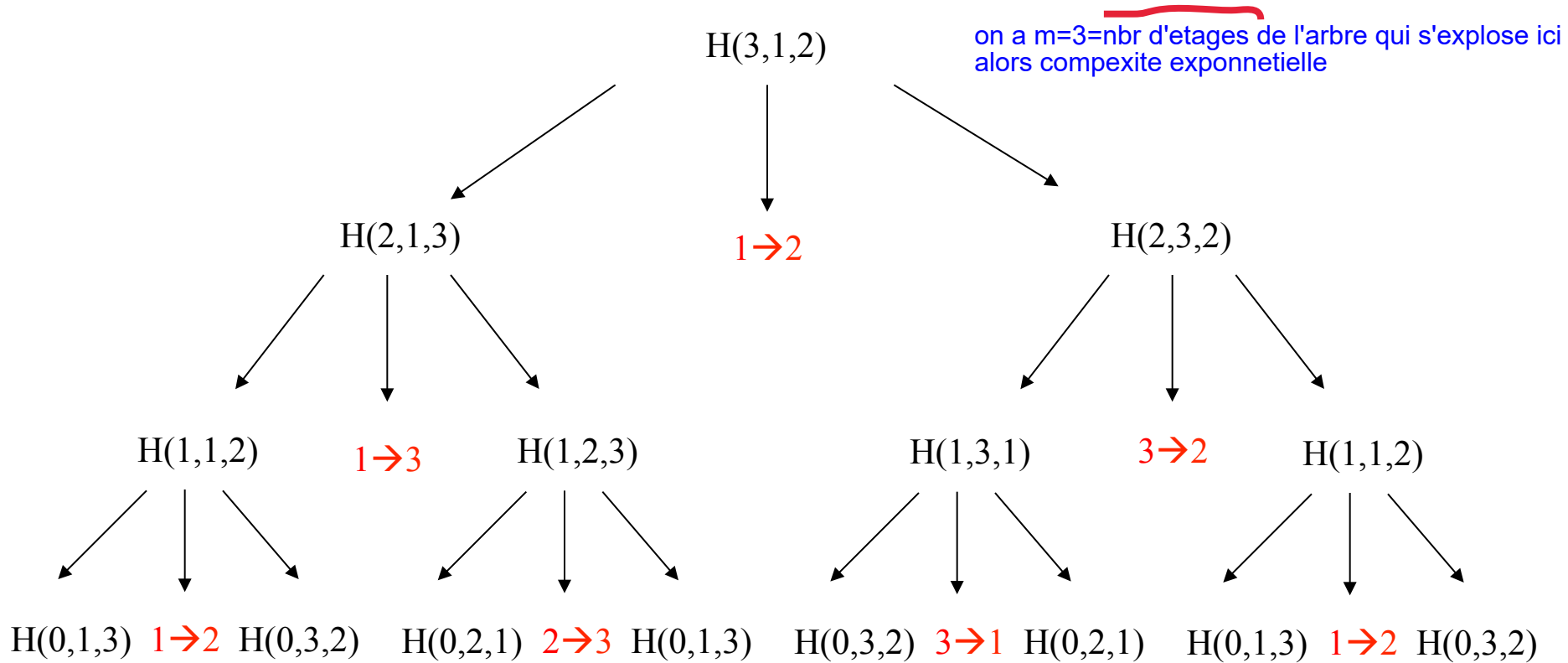
**Exercice :** Trouver une version non récursive de la procédure Hanoi.  
Trouver l'ordre de l'algorithme.



# Trace à l'aide d'appels récursif



# Trace à l'aide d'un arbre



# Chapitre 3 : Diviser-pour-régner

55

diviser-pour-régner (Divide and conquer) est une technique de conception d'algorithme composée de trois étapes :

- Décomposition de l'exemplaire en sous-exemplaires plus petits,
- Résolution des sous-exemplaires et
- Combinaison des sous-solutions.

## Plan

- 1 Fouille dichotomique
- 2 Multiplication des grands nombres
- 3 Multiplication matricielle
- 4 Exponentiation discrète

shema = regroupe plusieurs algorithmes

# Schéma des algorithmes DPR

**fonction** DPR(x)

methode classic

**si** x est suffisamment petit **alors retourner** ADHOC(x)

décomposer x en sous-exemplaires  $x_1, x_2, \dots, x_k$

**pour**  $i \leftarrow 1$  **jusqu'à** k **faire**  $y_i \leftarrow \text{DPR}(x_i)$  recursive !!

combinaison des  $y_i$  ( $1 \leq i \leq k$ ) pour obtenir une solution y

**retourner** y



# Résolution des récurrences DPR

TPE

$n=2^k$   $n=3^k$  .. pour dans les changement de variable dans la dem

**Théorème :** Soient  $a, b, c \geq 0$  et  $n = c^k$ . La solution de la récurrence de sous

$a = \text{nbr d'appel recursive}$

$b = \text{decomposition - composition des resultat}$

$c = \text{nbr exemple}$

$$T(1) = b$$

pour  $n = 1$

$$T(n) = aT(n/c) + bn$$

pour  $n > 1$

decomposition  
combinaison des resultats

est :

$$O(n)$$

si  $a < c$

$$T(n) \in O(n \log n)$$

si  $a = c$

$$O(n^{\log a})$$

si  $a > c$ , où le logarithme en base  $c$

$T(n)$  : temps d'exécution d'un algorithme DPR sur l'exemple  $n$

$t(0)=a$   
 $t(n)=t(n/2)+b$   
 $n=2^k \Rightarrow k=\log_2(n)$   
 $T[k]=t(n)$   
 $T(k)=b+T(k-1) \Rightarrow k*b + a$   
 $t(k-1)=b+t(k-2)$   
 $t(0)=a \Rightarrow$  remplacement successif  $\Rightarrow t(n)=\log_2(n)*b+a \Rightarrow$  appartient a  $O(\log(n))$

# Fouille dichotomique

dém

Problème : localisation de la valeur  $x$  dans un tableau  $T[1..n]$  trié (dictionnaire ou annuaire téléphonique)

**fonction** séquentielle ( $T[1..n], x$ )  
**pour**  $i \leftarrow 1$  jusqu'à  $n$  **faire**  
     **si**  $T[i] > x$  **alors retourner**  $i-1$   
**retourner**  $n$

$O(n)$

appartiecent a  $(n^{**3})$

**fonction** dichotomique ( $T[i..j], x$ )  
**si**  $i = j$  **alors retourner**  $i$   
 $k \leftarrow (i+j+1) \text{ div } 2$   
**si**  $x < T[k]$  **alors retourner** dichotomique ( $T[i..k-1], x$ )  
**sinon retourner** dichotomique ( $T[k..j], x$ )

$\log_2(n) \Rightarrow \log_3(n)$  .. on passe par une multiplication par une cte , et ca change pas l'ordre

$O(\log n)$

a chaque fois on devise par deux

## Exercice :

Faire la trace pour la recherche de 5 dans  $T=[1,4,5,8,10,11,13,14]$

Montrer que  $t(n) \in O(\log n)$

# Arithmétique des grands entiers

besoins : calculer les orbites .. les vehicule dans mars ..

**Utilisation :** calcul de très grande précision. En 1986,  $\Pi$  est calculé avec 30 millions de chiffres. Ce calcul a nécessité 30 heures de calcul sur un ordinateur Cray-2.

aujourd'hui , on est capable de trouver 31 milars de chiffre apres vergule dans pi

**Problème :** calcul de  $u*v$  avec  $u$  et  $v$  composés de  $n$  chiffres

chaque chiffre on le multiplie par les autre :  $1987*1917 \Rightarrow 16$  mult

**Solutions :**

Algorithme de multiplication classique des nombres  $\in O(n^2)$

4chifre\*4chifre=16 multiplication

$\rightarrow n \times n \Rightarrow n^2$  mult

**Question :**

Algorithme de multiplication des nombres DPR  $\in O(?)$

$$x=1738 \Rightarrow 17=x/100 \text{ et } 38 = \text{rest}(x=/100)$$

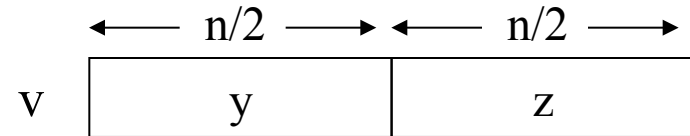
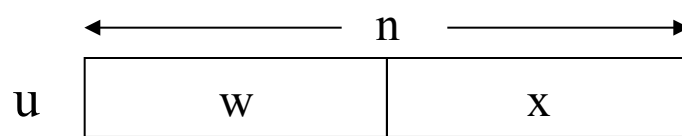
$$y=1234 \Rightarrow$$

$$12=x/100$$

$$\text{et } 34 = \text{rest}(x=/100)$$

$$\Rightarrow x*y=12*17*100 + 34*38*1$$

# Multiplication DPR des nombres (1/2)



pour simplifier

Supposons que  $n = 2^k$  (sinon considérons  $n'/n' > n$  et  $n' = 2^k$ )

$$u = 10^s w + x$$

$$v = 10^s y + z$$

$$\text{où } 0 \leq x, z < 10^s \text{ et } s = n/2$$

$$\Rightarrow u*v = 10^{2s} wy + 10^s(wz + yx) + xz$$

on passe par 4 mult au lieu de 1

gain = multiplication avec des nombres de taille tres petits



$t(1)=a$

$t(n) = 4 * t(n/2) + bn$  [bn pour les somme finaux n fois ,  $t(n/2)$  car on a devise u et v sur 2 entite , 4 appel recurs]

$n=2^k$   $n_0 < n$

$= \Rightarrow \in O(n^2)$

car  $a=4$   $c=2$  selon theoreme DPR

# Multiplication DPR des nombres (2/2)

**fonction** multA(u,v: grands-entiers) : grand-entier

$n \leftarrow \max(u,v)$

cette algorithme qui n'ameiore pas la comlxite ,  
car le classic aussi est de  $O(n^2)$  ,

si n est petit **alors**

dans la version  
amelioree

multiplier u et v par l'algorithme classique

**retourner** le résultat

mais il resuit les appel recursive ici de 4 vers 3 ,  
il reduit les appel recursive [on parle ici de l'algo  
amelioree]

sinon

$s \leftarrow n \text{ div } 2$

on partage u et v sur 2 entitee [w,x] et [y,z]

$w \leftarrow u \text{ div } 10^s, x \leftarrow u \text{ mod } 10^s$

algorithm de l'ordre  $O(n^2)$

$y \leftarrow v \text{ div } 10^s, z \leftarrow v \text{ mod } 10^s$

on a rien gagner en treme de  
complexite

**retourner**  $\text{multA}(w,y) 10^{2s} + (\text{multA}(w,z) + \text{multA}(y,x)) 10^s + \text{multA}(x,z)$

**Exercice :** Montrer que  $t(n) \in O(n^2)$

# Amélioration de la multiplication DPR (1/2)

$$\begin{cases} u = 10^s w + x \\ v = 10^s y + z \end{cases} \quad \text{où } 0 \leq x, z < 10^s \text{ et } s = n/2$$

Soient  $r$ ,  $p$  et  $q$  tels que :

$$r = (w + x)(y + z) = wy + wz + xy + xz$$

$$p = wy$$

$$q = xz$$

$$\begin{aligned} \Rightarrow u * v &= 10^{2s} wy + 10^s (wz + yx) + xz \\ &= 10^{2s} p + 10^s (r - p - q) + q \end{aligned}$$

# Amélioration de la multiplication DPR (2/2)

```

fonction multB(u,v: grands-entiers) : grand-entier
  n ← max(u,v)
  si n est petit alors
    multiplier u et v par l'algorithme classique
    retourner le résultat
  sinon
    s ← n div 2
    w ← u div 10s, x ← u mod 10s
    y ← v div 10s, z ← v mod 10s
    r ← multB(w+x, y+z)
    p ← multB(w,y)
    q ← multB(x,z)
    retourner 102sp + 10s(r-p-q) + q
  
```

3 appel recursive

$$t(1)=a$$

$$t(n)= 3 t(n/2)+bn$$

*n = taille des chiffres 1234 ⇒ n=4*

*Th*

**Exercice :** Montrer que  $t(n) \in O(n^{1,59})$

plus n est grand , plus plus d'impact

# Multiplication matricielle classique

Soient A, B deux matrices carrées d'ordre n.

$$C = AB = (c_{ij}) \ 1 \leq i, j \leq n \text{ avec } c_{ij} = \sum_{1 \leq k \leq n} a_{ik} b_{kj}$$

**fonction** mult\_classique(matrice A,B)

{n : ordre des matrices A et B}

**pour** i = 1 à n **faire**

**pour** j = 1 à n **faire**

        S ← 0

**pour** k = 1 à n **faire** { calculer un élément de C }

            S ← S + A[i,k]\*B[k,j]

        C[i,j] ← S

**retourner** C

appartient à  $(n^3)$



**Exercice :** Trouver l'ordre de l'algorithme classique.

# Multiplication matricielle DPR

on decompose des matrice d'ordre n vers des matrice d'ordre n/2

Décomposition de A et B :

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

et

$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

sous matrice d'ordre n/2

$$\text{Calcul de } C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

avec

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$

$$c_{21} = a_{21}b_{11} + a_{22}b_{21}$$

$$c_{22} = a_{21}b_{12} + a_{22}b_{22}$$

addition classique =  $K_{ij} = i + j$

t(1)=a matrice d'ordre 1[simple mult]

t(n)= 8\*t(n/2)+ bn [n decomposition (addition) temps cte]

pour trouver c => 2 multiplication sur chaque 4 sous matrice => 8 multiplication des sous matrice

Exercice : Trouver l'ordre de l'algorithme DPR

theoreme =>  $\log_8 3 \Rightarrow O(n^{**3})$  pas d'amelioration

# Algorithme de Strassen

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

et

$$C = A * B = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

4 matrices intermediaires

Calcul de  $m_1, m_2, m_3, m_4, m_5, m_6, m_7$  tels que :

$$m_1 = (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11})$$

$$m_2 = a_{11}b_{11}$$

$$m_3 = a_{12}b_{21}$$

$$m_4 = (a_{11} - a_{21})(b_{22} - b_{12})$$

$$m_5 = (a_{21} + a_{22})(b_{12} - b_{11})$$

$$m_6 = (a_{12} - a_{21} + a_{11} - a_{22})b_{22}$$

$$m_7 = a_{22}(b_{11} + b_{22} - b_{12} - b_{21})$$

et

$$c_{11} = m_1 + m_2$$

$$c_{12} = m_1 + m_2 + m_1 + m_2$$

$$c_{21} = m_1 + m_2 + m_1 + m_2$$

$$c_{22} = m_1 + m_2 + m_1 + m_2$$

$$t(1)=a$$

$$t(n)= 7*t(n/2) + bn$$

$t(n)$  appartient a  $O(n^{**\log 7})$

includ dans  $O(n^{**2,8})$

un gain tres bon pour des n plus grand !!

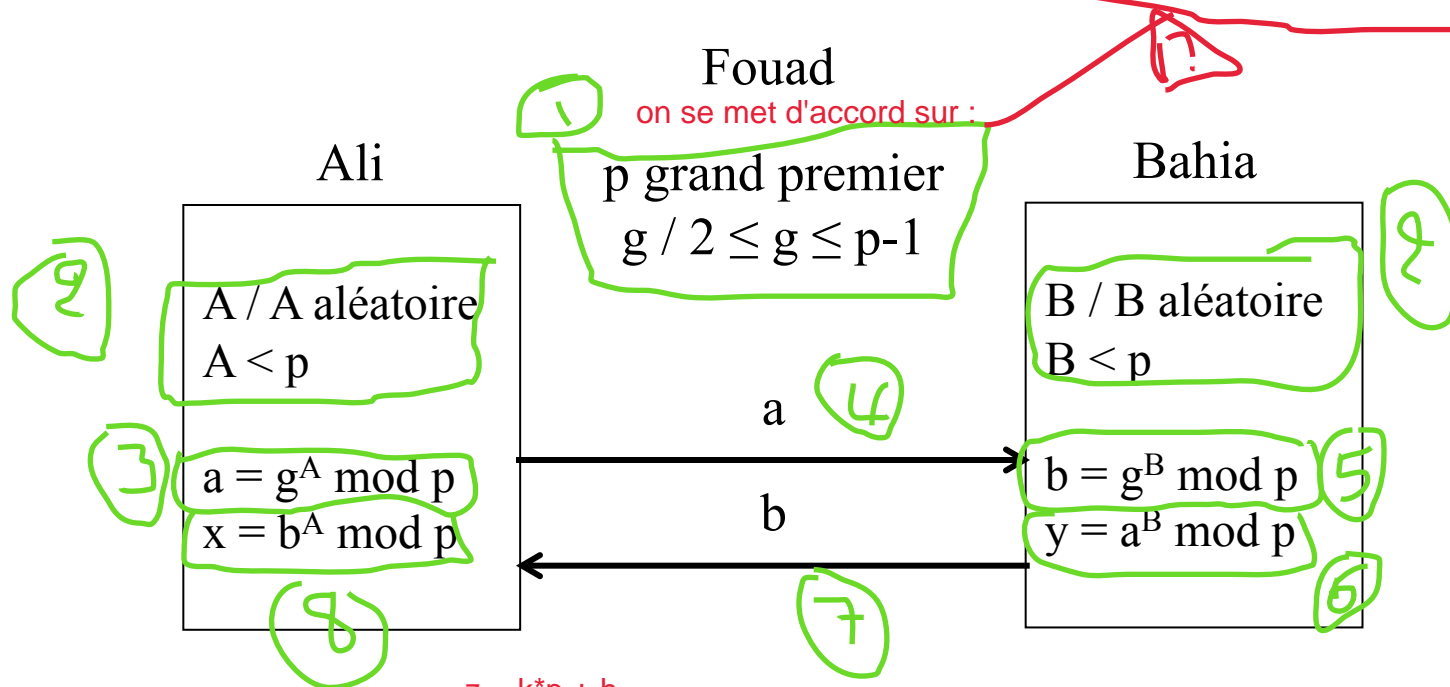
**Exercice :** Trouver l'ordre de l'algorithme de Strassen

# RSA

## Protocole de cryptage (1976)

p premier pour le modulo

p est tres grand , pour ne pas decrypter facilement



**Exercice :** Montrer  $(Z \mod p)^n \mod p = Z^n \mod p$ . Dédire que  $x = y$

**Conclusion :** Ali et Bahia partagent une information sans que Fouad puisse la détecter

# Exponentiation discrète (1/3)

$g^A$   $A$  entier

$g^A \bmod p$

fonction expod1(g,A,p)

$a \leftarrow 1$

$g^{**A}$  pour  $i \leftarrow 1$  jusqu'à  $A$  faire  $a \leftarrow a g$

retourner  $a \bmod p$

Remarque :  $xy \bmod p = ((x \bmod p) (y \bmod p)) \bmod p$  (diviser le calcul de modulo de  $g^{**A}$  (c'est pour ça qu'on le divise))

fonction expod2(g,A,p)

$a \leftarrow 1$

pour  $i \leftarrow 1$  jusqu'à  $A$  faire  $a \leftarrow a g \bmod p$

retourner  $a$

(tj n'ai a < p)

**Exercice :** Analyser et comparer le temps d'exécution de expod1 et expod2 en fonction de  $A$  et  $p$ . Pour simplifier supposer que  $g = p/2$ .



# Exponentiation discrète (2/3)

Exemple : calcul de  $x^{23}$

$x^{23} = (((...((x \ x)x)... )x) \Rightarrow 22$  multiplications

$x^{23} = (((x^2)^2x)^2x)^2x \Rightarrow 7$  multiplications

pour ne pas repeter le calcul  $x*x \dots$

$$x^{**23} = (x^{**11})^{**2} * x$$

**fonction** expodrec(x,n)

**si**  $n = 0$  **alors retourner** 1

**si**  $n$  est impair **alors**

$a \leftarrow \text{expodrec}(x, n-1)$

**retourner**  $a \ x$

**sinon**

$a \leftarrow \text{expodrec}(x, n/2)$

**retourner**  $a^2$

**fonction** expoditer(x,n)

$a \leftarrow x, b \leftarrow 1$

**tant que**  $n > 0$  **faire**

**si**  $n$  est impair **alors**

$b \leftarrow a \ b$

$a \leftarrow a^2$

$n \leftarrow n \text{ div } 2$

**retourner**  $b$

# Exponentiation discrète (3/3)

on va l'exploiter pour calculer  $F^{**}n$  pour le TP3  
voir page suivant

```
fonction expod3(g,A,p)
  si A = 0 alors retourner 1
  si A est impair alors
    a ← expod3(g,A-1,p)
    retourner a g mod p
  sinon
    a ← expod3(g,A/2,p)
    retourner a2 mod p
```

```
fonction expod4(g,A,p)
  a ← g, b ← 1 exposant
  tant que A > 0 faire
    si A est impair alors
      b ← a b mod p
    a ← a2 mod p
    A ← A div 2
  retourner b
```

# Page suivante Exercice

TP3

Soit la matrice  $F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

$F^{n+1} = \begin{pmatrix} f_n & f_{n+1} \\ f_{n-1} & f_n \end{pmatrix}$

par récurrence

$n=1$   $F = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$

a) Montrer que  $F^n = \begin{bmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{bmatrix}$

où  $f_n$  est le nième nombre de Fibonacci.

b) Dédurre un algorithme de type diviser-pour-régner pour le calcul de  $F^n$ .

c) comparer cet algorithme avec fib3 en prenant comme matrices intermédiaires :

$A = \begin{bmatrix} k & h \\ h & k+h \end{bmatrix}$  et  $B = \begin{bmatrix} i & j \\ j & i+j \end{bmatrix}$

$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

# Retour à l'algorithme fib3

on utilise dpr  
on considere des valeurs  
plus petit =>  
algorithm => matrice

$F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

**fonction** expod4(F,n)

$A \leftarrow F, B \leftarrow I$

**tant que**  $n > 0$  faire

**si**  $n$  est impair **alors**

$B \leftarrow AB$

$A \leftarrow A^2$

$n \leftarrow n \text{ div } 2$

**retourner** B

premier affectation :  
initialiser k h  
deuxieme i j

par le calcul de AB et  $A^2$   
on trouve ces expression

**fonction** expod5(n) = fib3

$k \leftarrow 0, h \leftarrow 1, i \leftarrow 1, j \leftarrow 0$

**tant que**  $n > 0$  faire

**si**  $n$  est impair **alors**

$t \leftarrow hj$

$j \leftarrow hi + kj + t$

$i \leftarrow ki + t$

$t \leftarrow h^2$

$h \leftarrow 2hk + t$

$k \leftarrow k^2 + t$

$n \leftarrow n \text{ div } 2$

**retourner** j

$O(\log(n))$

pour avoir A  
et B

# Algorithme aux voraces = ils s'attaquent à la plus grande morceau de viande

## Chapitre 4 : Algorithmes voraces

### Généralement

assez simples  $\Rightarrow$  rapides

résolution des problèmes d'optimisation

solutions approximatives (heuristique)

mais il resulte les appel recursive ici de 4 vers 3 ,  
il reduit les appel rec

### Contenu

$\log_2(n) \Rightarrow \log_3(n)$  .. on passe par une multiplication par

| Schéma général

| Remise de monnaie

✓ Plus courts chemins

Coloration d'un graphe

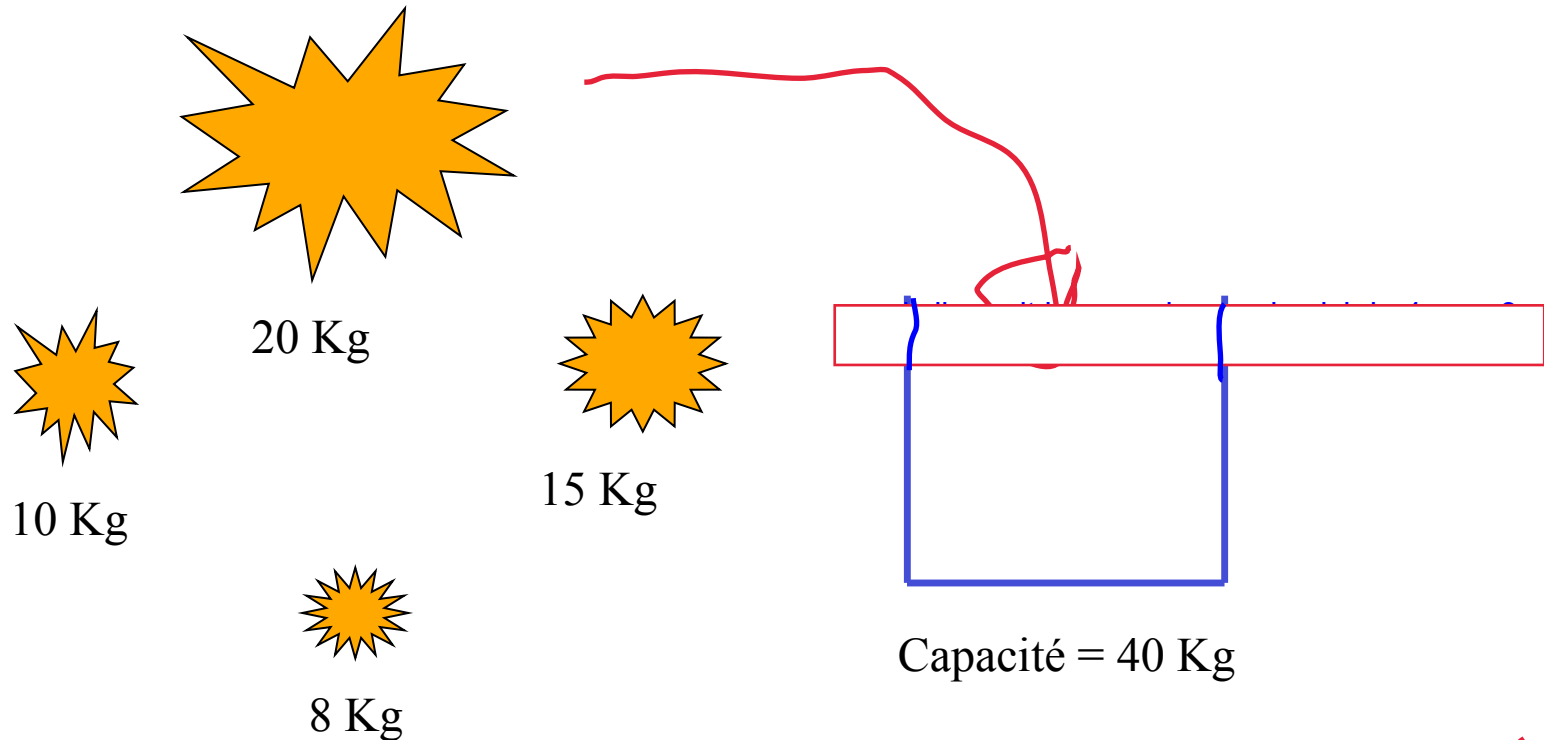
Feux de signalisation

algorithme tout les possibilite =>  $O(2^n)$  exponentiel => algorithme exostifs

algorithme vorace : n iteration sur les elts , et a chaque fois il va tester si c'est le max (2eme parcours) =>

$O(n^2)$

# Optimisation de la sélection avec contrainte



Solution vorace =  $20 + 15 = 35 < 40$

*not optimal*

20+10+8=38 mieux !

on demontre qu'il est optimale par les ensemble des possibilites

# Schéma des algorithmes voraces

**fonction** vorace ( $C$  : ensemble) : ensemble

$\{C \text{ est l'ensemble de tous les candidats}\}$

$S \leftarrow \emptyset$  {ensemble solution}

**tant que**  $\neg$  solution ( $S$ ) et  $C \neq \emptyset$  **faire**

$x \leftarrow$  l'élément de  $C$  qui optimise  $\text{select}(x)$  selon notre critere taille ...

$C \leftarrow C - \{x\}$  s'il ne dépasse pas la capacite ..

**si** réalisable ( $S \cup \{x\}$ ) **alors**  $S \leftarrow S \cup \{x\}$

**si** solution ( $S$ ) **alors retourner**  $S$

**sinon retourner** pas de solution

# Exemple 1 : Remise de monnaie

**Problème :** remettre la monnaie en donnant le moins de pièces possible.

**Solution :**

C : ensemble fini de pièces 1, 5, 10, 20, 50 et 100 DH

Solution(S) : total des pièces choisies correspond au montant à rendre

Ensemble réalisable : total des pièces n'excède pas la somme à rendre

fonction de sélection : la plus grande pièce qui reste dans C

Fonction objective : nombre de pièces utilisées dans la solution

**Exercice :** Décrire un algorithme vorace. Montrer qu'il fournit toujours une solution optimale lorsqu'elle existe



# Exemple 1 : Algorithme vorace

**fonction** Remise\_Monnaie (Montant) : tableau

$C \leftarrow \{1, 5, 10, 20, 50, 100\}$

$S \leftarrow 0$

**tant que** (Montant > 0) et ( $C \neq \emptyset$ ) **faire**

$x \leftarrow \max\{y / y \in C\}$

$C \leftarrow C - \{x\}$

$NP \leftarrow \text{Montant div } x$   $\{\text{Nombre de pièces de type } x\}$

$S[x] \leftarrow NP$

$\text{Montant} \leftarrow \text{Montant} - NP * x$

**si** (Montant = 0) **alors retourner** S

**sinon retourner** pas de solution

**Exercice :** Montrer l'Optimalité (idée ( $C_i \geq 2 C_{i-1}$ ))



## Exemple 1 : qualité de la solution

*il est heuristique*

Soit  $C' = C \cup \{12\} = \{1, 5, 10, 12, 20, 50, 100\}$

Montant = 16 =  $12 + 1 + 1 + 1 + 1$  (5 pièces)

⇒ solution non optimale

=  $10 + 5 + 1$  (3 pièces)

⇒ est la solution optimale

Soit  $C'' = C' - \{1\} = \{5, 10, 12, 20, 50, 100\}$

Montant = 15 =  $12 + ???$

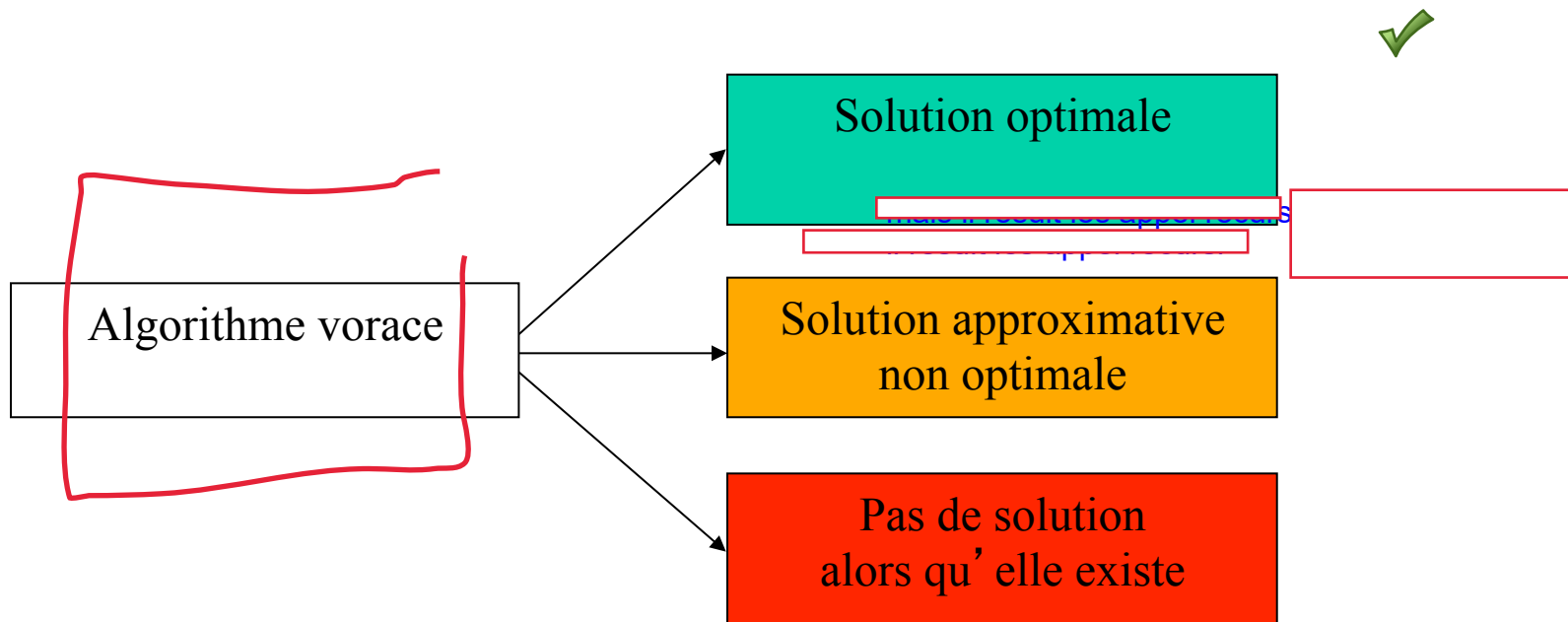
⇒ pas de solution

=  $10 + 5$

⇒ une solution existe

# Solution d'un algorithme vorace

---



## Exemple 2 : Plus courts chemins

Soit  $G = (N, A)$  un graphe orienté où  $N = \{1, 2, \dots, n\}$  ensemble des nœuds et  $A$  est l'ensemble d'arcs. A chaque arc est associé une longueur non négative. Essayons de déterminer la longueur du plus court chemin de 1 vers les autres sommets.

$L$  : matrice d'ordre  $n$  tel que  $L[i, j] \geq 0$  si l'arc  $(i, j)$  existe et  $L[i, j] = \infty$  s'il n'existe pas.

ou on enregistre les résultats

**fonction** Dijkstra( $L[1..n, 1..n]$ ) : tableau [ $2..n$ ]

$C \leftarrow \{2, 3, \dots, n\}$

**pour**  $i \leftarrow 2$  **jusqu'à**  $n$  **faire**  $D[i] \leftarrow L[1, i]$

**répéter**  $n-2$  fois

$v \leftarrow$  l'élément de  $C$  qui minimise  $D[v]$

$C \leftarrow C - \{v\}$

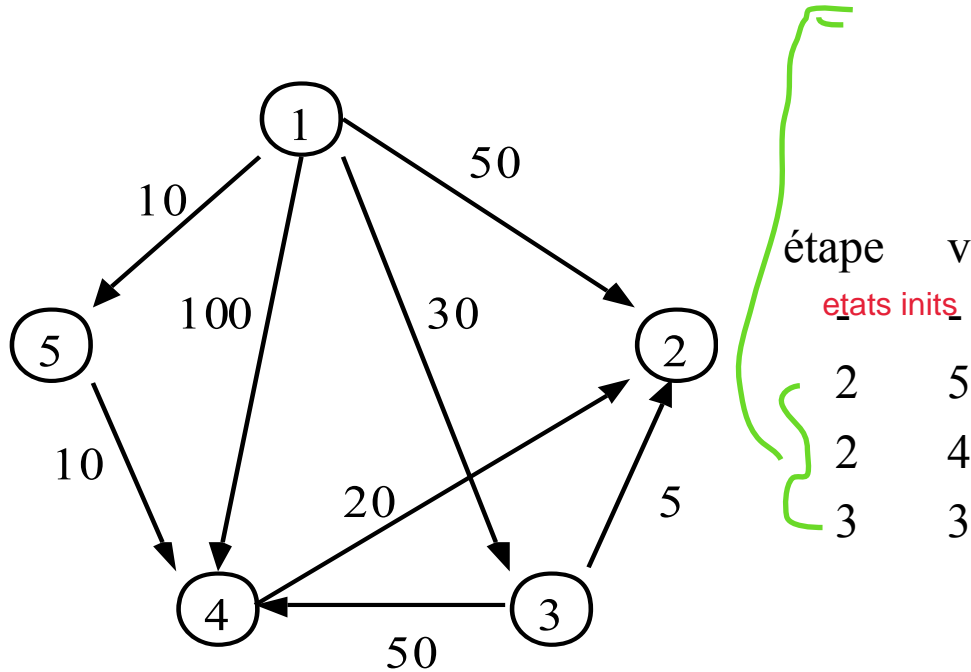
**pour** chaque élément  $w$  de  $C$  **faire**

$D[w] \leftarrow \min(D[w], D[v] + L[v, w])$

**retourner**  $D$

# Exemple 2 : Trace de l'algorithme

nbr sommet=5 et nbr iteration = 3



étape	v
états initiaux	
2	5
2	4
3	3

C	D
{2,3,4,5}	[50,30,100,10]
{2,3,4}	[50,30,20,10]
{2,3}	[40,30,20,10]
{2}	[35,30,20,10]

on a trouvé tous les courts chemins vers tous les noeuds

**Exercice :** Déterminer l'ordre de l'algorithme de Dijkstra. Modifier le afin de trouver le chemin le plus court entre tous les couples de sommets.

appliquer n fois , sur les n sommets 1 , 2 , 3 , , ,  
Dijkstra modifié appartient  $O(n^3)$

# Exemple 3 : Minimisation de l'attente

Soient un serveur,  $n$  clients et  $t_i$  : temps de service requis par le client  $i = 1, 2, \dots, n$

minimiser :  $T = \sum_{1 \leq i \leq n} (\text{temps passé dans le système par le client } i)$  *de service*

**Exemple** :  $t_1 = 4, t_2 = 7, t_3 = 3$ .



on a 6 possibilités =  $3!$  permutation possible  
 $n$  elt  $\Rightarrow n!$  permutation

ordre	T
1 2 3	$4 + (4 + 7) + (4 + 7 + 3) = 29$
1 3 2	$4 + (4 + 3) + (4 + 3 + 7) = 25$
2 1 3	$7 + (7 + 4) + (7 + 4 + 3) = 32$
2 3 1	$7 + (7 + 3) + (7 + 3 + 4) = 33$
<b>3 1 2</b>	<b><math>3 + (3 + 4) + (3 + 4 + 7) = 24</math></b>
3 2 1	$3 + (3 + 7) + (3 + 7 + 4) = 27$



$$n! \sim e^n$$

*formule*

on commence par le moins de temps

← ordre optimal

considère tt les cas possibles

**Exercice** : Montrer que l'algorithme exhaustif  $\in O(n!)$ . Décrire l'algorithme vorace et analyser son efficacité. Démontrer que sa solution est optimale.

# Exemple 3 : Algorithme vorace

**fonction** Ordonnancement (n,t) : tableau

$C \leftarrow \{1,2,3,\dots,n\}$

**pour** k=1 à n **faire**

$j \leftarrow i / t_i = \min \{t_h / h \in C\}$

$C \leftarrow C - \{j\}$

$S[k] \leftarrow j$

**retourner** S

$n^2$  I think

**Exercice :** Quel est l'ordre de l'algorithme ?

# Exemple 3 : Optimalité de l'algorithme (1/2)

Soit  $I = (i_1, i_2, \dots, i_n)$  une permutation quelconque dans  $\{1, 2, \dots, n\}$ .

Soit  $T(I)$  le temps total passé dans le système pour les clients  $i_1, i_2, \dots, i_n$ .

$$\begin{aligned} T(I) &= t_{i_1} + (t_{i_1} + t_{i_2}) + \dots + (t_{i_1} + t_{i_2} + \dots + t_{i_n}) = nt_{i_1} + (n-1)t_{i_2} + \dots + t_{i_n} \\ &= \sum_{1 \leq k \leq n} (n - k + 1)t_{i_k} \end{aligned}$$

Supposons qu'il existe dans  $I$ , deux entiers  $a$  et  $b$  tel que  $a < b$  et  $t_{i_a} > t_{i_b}$ . Invertissons l'ordre de ces deux clients et on obtient l'ordre  $I'$ .

Ordre de service	1	2	...	a	...	b	...	n
$I$	$i_1$	$i_2$	...	$i_a$	...	$i_b$	...	$i_n$
$I'$	$i_1$	$i_2$	...	$i_b$	...	$i_a$	...	$i_n$

$$T(I') = (n-a+1)t_{i_b} + (n-b+1)t_{i_a} + \sum_{1 \leq k < n \text{ et } k \neq a \text{ et } b} (n-k+1)t_{i_k}$$



## Exemple 3 : Optimalité de l'algorithme (2/2)

$$\begin{aligned} T(I) - T(I') &= (n-a+1)t_{ia} + (n-b+1)t_{ib} - (n-a+1)t_{ib} - (n-b+1)t_{ia} \\ &= (b-a)t_{ia} + (a-b)t_{ib} = (b-a)(t_{ia} - t_{ib}) \end{aligned}$$

Comme  $b - a > 0$  et  $t_{ia} - t_{ib} > 0$ , on déduit que :

$$T(I) - T(I') > 0 \text{ et par conséquent } T(I) > T(I').$$

Nous pouvons améliorer tout ordre de service où un client est servi avant un autre nécessitant moins de temps

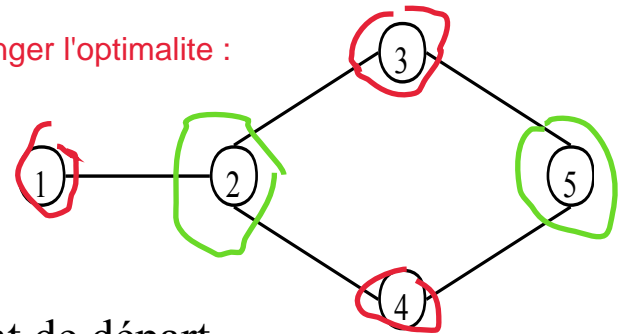
⇒ optimalité de l'algorithme

# Exemple 4 : Coloration d'un graphe

Soit  $G = (N, A)$  un graphe non orienté.

Colorer le graphe tels que deux sommets reliés doivent être de couleurs différentes.

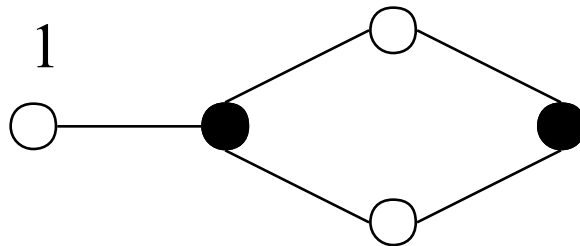
les numero peuvent changer l'optimalite :  
algo optimale ou non



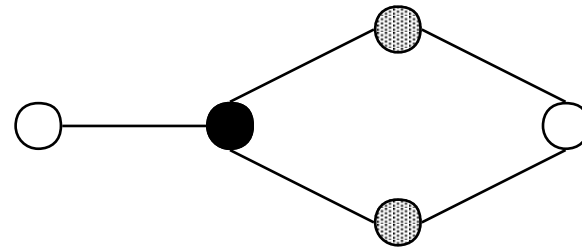
L'algorithme vorace :

- Choisir une couleur et un sommet comme point de départ
- Considérer les autres sommets et essayer de les colorer par cette couleur
- Lorsqu'on ne peut plus faire de progrès choisir une nouvelle couleur et un nouveau point de départ non coloré
- Colorer tout ce qui est possible avec cette deuxième couleur
- ainsi de suite.

# Exemple 4 : Coloration d'un graphe



2-coloration



3-coloration

## Remarques

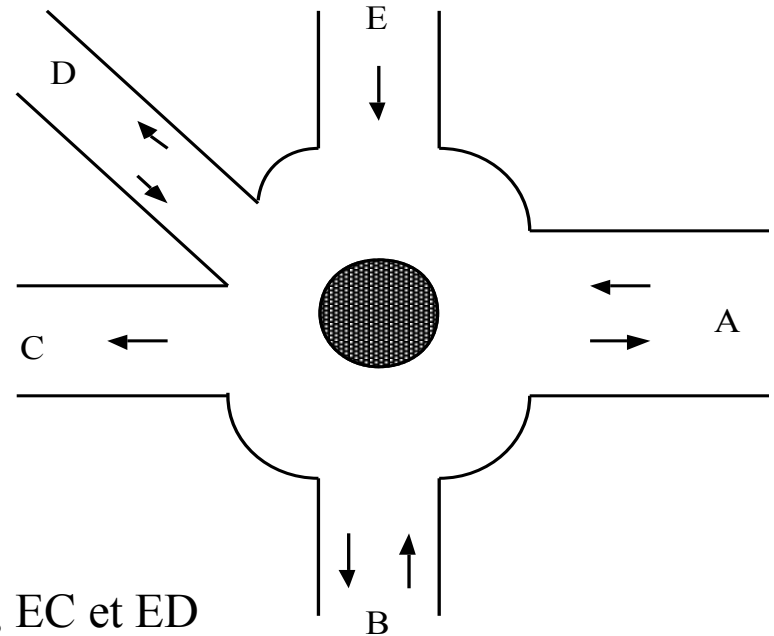
Un algorithme basé sur une heuristique vorace permet la possibilité de trouver une "bonne" solution mais pas la certitude

l'algorithme exhaustif qui produit une solution optimale est exponentiel

# Exemple 5 : Feux de signalisation

Considérons 5 artères : A, B, C, D et E.

D et E sont des artères à sens unique



## Changements de direction (13)

AB, AC, AD, BA, BC, BD, DA, DB, DC, EA, EB, EC et ED

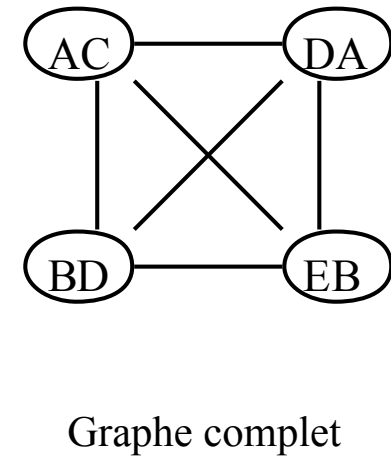
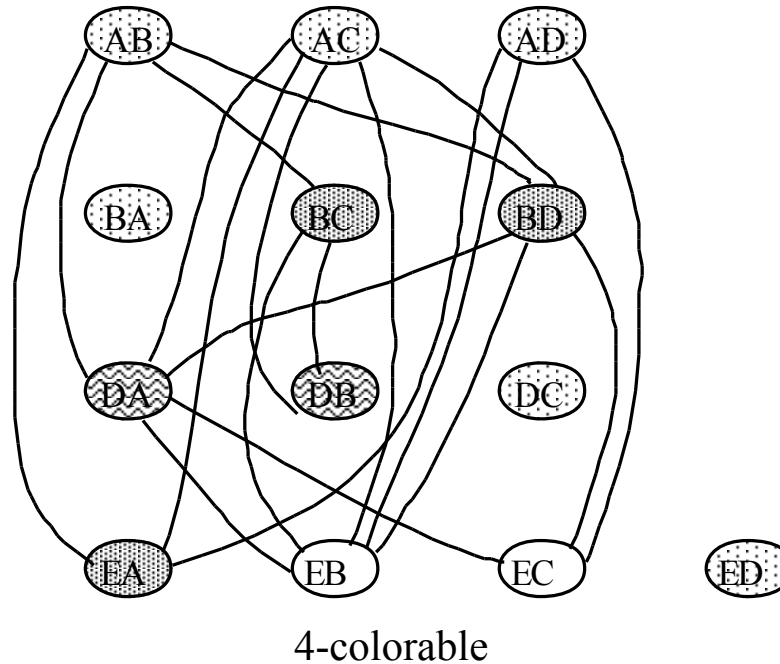
AB et EC sont possibles alors que AD et EB peuvent provoquer une collision

## Modélisation à l'aide d'un graphe

sommets correspondent aux changements de direction

arêtes joignent les couples de sommets dont les itinéraires se croisent

# Exemple 5 : Trace de l'algorithme

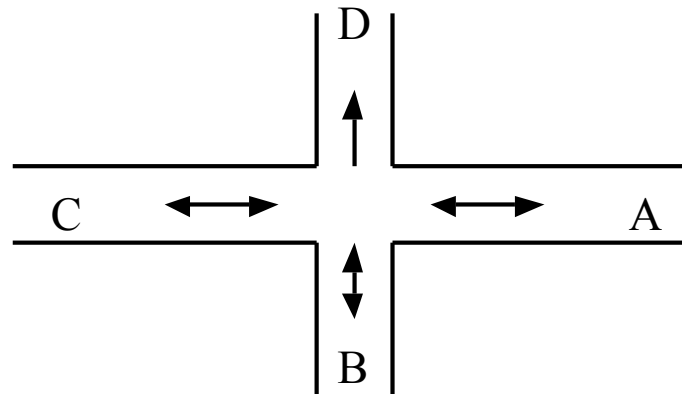


Solution optimale car le sous-graphe composé des sommets AC, DA, BD et EB est un graphe complet de 4 sommets et nécessite par conséquent quatre couleurs.

Les sommets de la même couleur correspondent aux itinéraires sans collision. Les quatre couleurs correspondent aux quatre phases nécessaires du système de signalisation

# Exercices

**Exercice 1 :** Résoudre le problème de signalisation pour le carrefour suivant :



**Exercice 2 :** Trouver un algorithme vorace pour résoudre le problème du commis voyageur en supposons que le graphe est complet.

DPR : est une methode descendante [try to concept an algo to get the solution]  
Programation Dynamique : methode ascendante [on partir de sol vers l'algorithme]

# Chapitre 5 : Programmation dynamique

diviser-pour-régner : méthode descendante

programmation dynamique : méthode ascendante + utilisation espace mémoire  
(afin d'éviter de calculer la même chose plus d'une fois)

## Contenu

Coefficient du binôme

Principe d'optimalité

Multiplication chaînée de matrices

Plus courts chemins

# Exemple 1 : Coefficient du binôme

## Coefficient binomial

$$C(n,k) = C(n-1,k-1) + C(n-1,k) \text{ si } 0 < k < n$$
$$= 1 \text{ autrement}$$

**fonction**  $C(n,k)$

**si**  $k=0$  **ou**  $k=n$  **alors retourner** 1

**sinon retourner**  $C(n-1, k-1) + C(n-1, k)$

Exemple

Def  
Direct

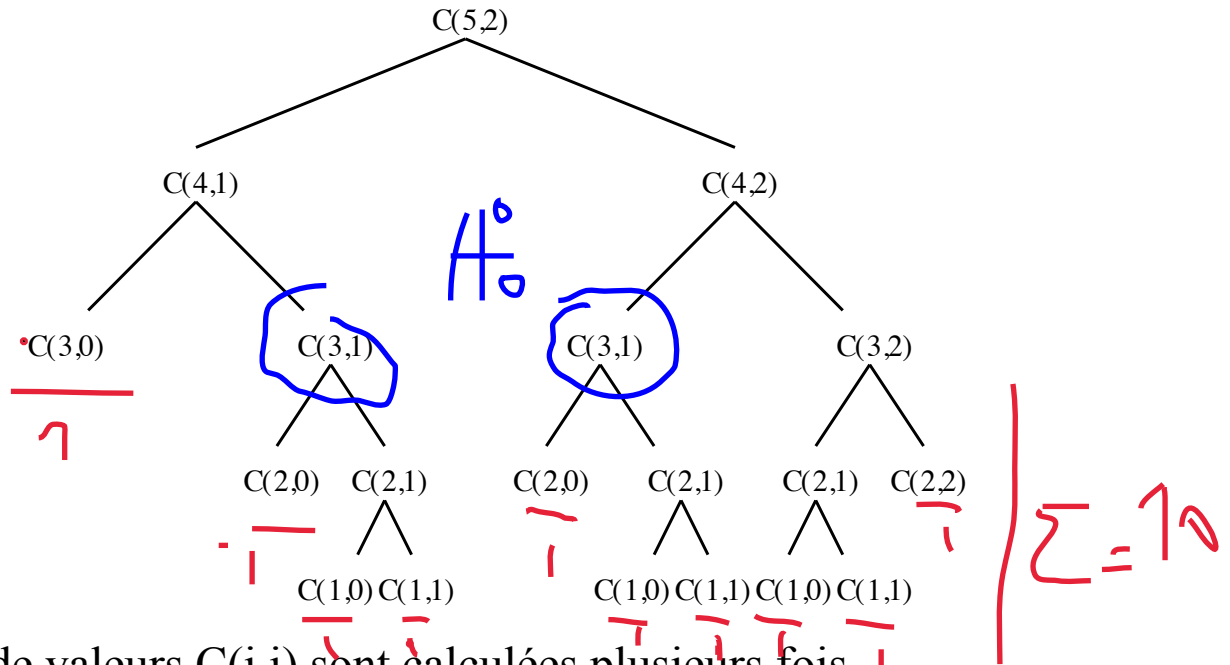
$$\frac{n!}{k!(n-k)!}$$

} complexité  
grand  
factoriel  
exp



# Exemple 1 : Trace de l'algorithme

Trace pour  $(5,2)=10$



**Remarques :** beaucoup de valeurs  $C(i,j)$  sont calculées plusieurs fois.

**Exercice :** Montrer que le nombre d'appels récurifs provoqués par  $C(n,k)$  est :

$$2 \binom{n}{k} - 2$$

# Exemple 1 : Triangle de Pascal

Procédure Triangle\_Pascal(N:entier)

Début

Var i,j: entier;

C:tableau[][] d'entier;

Pour i de 0 à N Faire

  C[i][i] <-- 1;

  C[i][0] <-- 1;

Pour i=2 à n faire

  Pour j de 1 à i-1 Faire

    C[i][j] <-- C[i-1][j]+C[i-1][j-1];

  FinPour

FinPour

Pour i de 0 à N Faire

  Pour j de 0 à i Faire

    écrire(C[i][j]);

  FinPour

Fin

	0	1	2	3	4	5	...	k-1	k
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
.									
.									
n-1	1								
n	1								

on n'a pas besoin d'aller jusqu'à la nième case, il sont que des restes [inutiles]

$C(n-1, k-1)$   $C(n-1, k)$

$\downarrow$

$C(n, k)$

appartient à  $O(n^2)$  car on a utilisé un tableau

**Exercice :** Décrire cet algorithme. Montrer qu'il demande un espace dans  $O(k)$  et un temps dans  $O(nk)$  si on compte chaque addition à coût unitaire.

**Exercice :** Parmi les algorithmes du calcul des nombres de Fibonacci, lequel est un algorithme de programme dynamique ?

- si un pb verifie le principe d'optimalite , alors on va deduire que la solution obtenue par un algorithme base sur la programmation dynamique va produire une solution optimale
- sinon : on ne peut pas savoir si la solution est optimale ou non
- cause de principe : la Prog Dynam est une methode ascendante (il resoud le probleme localement) , d'ou la necessite d'avoir l'optimum des les premier sequences de solution

# Principe d'optimalité

La programmation dynamique est souvent utilisee dans des problemes d'optimisation

La programmation dynamique est souvent utilisée pour résoudre des problèmes d'optimisation qui satisfont le principe d'optimalité suivant :

**Principe d'optimalité :** Dans une séquence optimale de décisions ou de choix, chaque sous-séquence doit être optimale.

cad pour passer d'une noeud a une autre , il faut deja etre dans l'optimume

**Exemple :** le problème du plus court chemin vérifie le principe d'optimalité

**Exercice :** Le principe d'optimalité s'applique-t-il au problème du plus long chemin simple entre deux villes ? Un chemin simple va directement de ville en ville sans passer deux fois par la même ville (sans cycle).

le plus court chemin satisfait le principe d'optimalite  
mais le plus long chemin ne le satisfait pas

# Exemple 2 : Multiplication chaînée de matrices

$$M = M_1 M_2 \dots M_n$$

*ne pas*

Comme la multiplication matricielle est associative

$$\begin{aligned} M &= (\dots((M_1 M_2) M_3) \dots) M_n \\ &= M_1 (M_2 (M_3 (\dots (M_{n-1} M_n) \dots))) \\ &= ((M_1 M_2) (M_3 M_4) \dots) \\ &= \dots \end{aligned}$$

*Handwritten notes:*

$$M_{pq} \times N_{qr} = L_{pr}$$

*obligatoire le m*

chaque elt  
demande q  
multiplication  
alors en total  
c'est p\*q\*r  
multiplication  
[ligne  
colonnes]

Le choix d'une méthode peut influencer sur le temps de calcul.

**Exercice :** Montrer que le calcul de  $AB$ , où  $A$  est d'ordre  $p \times q$  et  $B$  est d'ordre  $q \times r$ , par la méthode directe nécessite  $pqr$  multiplications de nombres scalaires.

# Exemple 2 : Application

**Exemple :** Soient quatre matrices A, B, C et D d'ordre (13x5), (5x89), (89,3) et (3, 34) respectivement.

Il y a cinq manières différentes de calculer ABCD :

5 possibilité  
d'association  
==nbr catalan

$((AB)C)D$	qui nécessite	$13 \cdot 5 \cdot 89 + 13 \cdot 89 \cdot 3 + 13 \cdot 3 \cdot 34 = 10581$	multiplications
$(AB)(CD)$	" "	54201	"
$(A(BC))D$	" "	<b>2856</b>	"
$A((BC)D)$	" "	4055	"
$A(B(CD))$	" "	26418	"

La méthode la plus efficace est 9,5 fois plus rapide que la plus lente

# Exemple 2 : Nombre de Catalan

Soit  $T(n)$  : nombre de manières différentes d'insérer des parenthèses dans  $M$

$$M = (M_1 M_2 \dots M_i) (M_{i+1} \dots M_n)$$

Nous avons  $T(i)$  manières de mettre les parenthèses dans la partie gauche de  $M_i$  et  $T(n-i)$  manières de mettre les parenthèses dans la partie droite. Par conséquent

$$T(n) = \sum_{1 \leq i \leq n-1} T(i) T(n-i)$$

$$\text{et } T(1) = 1$$

nombre d'association possibles

$T(n)$  s'appelle nombre de Catalan.

n	T(n)
1	1
2	2
3	2
4	5
5	14
10	4862

très  
croissante

**Exercice** : Montrer que  $T(n) = 1/n \binom{2n-2}{n-1}$

dans l'exemple d'avant  $n=4$   
 $T(4) = 1/4(6,3) = 5$

## Exemple 2 : Algorithme de programmation dynamique

Soit  $m_{ij}$  ( $1 \leq i \leq j \leq n$ ) la solution optimale pour la partie  $M_i \dots M_j$  du produit  $M$ .

La solution du problème est  $m_{1n}$ .

Supposons que la dimension de  $M_i$  est  $d_{i-1} \times d_i$  pour  $i = 1, 2, \dots, n$ .

Construisons la table  $m_{ij}$  diagonale par diagonale : la diagonale  $s$  contient l'élément  $m_j$  tel que  $j-i = s$ .

$$\text{Cas 1 : } s = 0 \quad \Rightarrow \quad m_{ij} = 0, \quad i = 1, 2, \dots, n$$

$$\text{Cas 2 : } s = 1 \quad \Rightarrow \quad m_{i,i+1} = d_{i-1} d_i d_{i+1}, \quad i = 1, 2, \dots, n-1$$

$$\text{Cas 3 : } 1 < s < n \quad \Rightarrow \quad m_{i,i+s} = \min_{i \leq k \leq i+s-1} (m_{ik} + m_{k+1,i+s} + d_{i-1} d_k d_{i+s})$$

Le troisième cas représente le fait que pour calculer  $(M_i M_{i+1} \dots M_{i+s})$  on essaye toutes les possibilités  $(M_i \dots M_k)(M_{k+1} \dots M_{i+s})$  pour en choisir la meilleure.

c'est une approche ascendante : on part de bas vers le haut (localement to generalement)

## Exemple 2 : Trace de l'algorithme



1er Matrice (13\*5)

2eme (5\*89) ..

Reprenons l'exemple précédent :  $d = (13, 5, 89, 3, 34)$

1er diag != 0

1er produit  $13 \times 5 \times 89$

$s = 1 : m_{12} = \underline{5785}, m_{23} = \underline{1335}$  et  $m_{34} = \underline{9078}$

2eme diag

MM

$s = 2 : m_{13} = \min(m_{11} + m_{23} + d_0 d_1 d_3, m_{12} + m_{33} + d_0 d_2 d_3) = \min(1530, 9256) = \underline{1530}$

$m_{24} = \min(m_{22} + m_{34} + d_1 d_2 d_4, m_{23} + m_{44} + d_1 d_3 d_4) = \min(24208, 1845) = \underline{1845}$

$s = 3 : m_{14} = \min(m_{11} + m_{24} + d_0 d_1 d_4, m_{12} + m_{34} + d_0 d_2 d_4, m_{13} + m_{44} + d_0 d_3 d_4)$

$= \min(\underline{4055}, 54201, 2856) = \underline{2856}$

on construit cette table diagonale par diagonale

	j=1	j=2	j=3	j=4
i=1	0	<u>5785</u>	<u>1530</u>	<b>2856</b>
i=2		0	<u>1335</u>	<u>1845</u>
i=3			0	<u>9078</u>
i=4				0

si  $i=j$  : on fait 0 produit  
matricier  $MM \Rightarrow M$   
sinon on varie  $s$

$m_1, m_2, m_3$   
 $M_1 \times M_2$



# Exercices

**Exercice 1 :** Ecrire l'algorithme qui calcule  $m_{1n}$ . Comment peut-on modifier l'algorithme si l'on veut savoir comment calculer  $M$  de façon optimale ?

**Exercice 2 :** Montrer que l'algorithme  $\in \theta(n^3)$ .

## Exemple 3 : Plus courts chemins

Soit  $G = (N, A)$  un graphe orienté où  $N = \{1, 2, \dots, n\}$ . Soit  $L$  une matrice qui donne la longueur de chaque arc. Trouvons le plus court chemin entre chaque paire de sommets.

Le principe de l'optimalité s'applique car si  $k$  est un sommet intermédiaire sur le plus court chemin entre  $i$  et  $j$  alors la portion du trajet de  $i$  à  $k$  ainsi que celle de  $k$  à  $j$  doivent aussi être optimales. A l'itération  $k$  on trouve un chemin qui ne passe que par  $\{1, 2, \dots, k\}$ .

$$D_k[i, j] = \min(D_{k-1}[i, j], D_{k-1}[i, k] + D_{k-1}[k, j])$$

**Procédure Floyd** ( $L[1 \dots n, 1 \dots n]$ ) : **tableau**  $[1 \dots n, 1 \dots n]$

$D \leftarrow L$

**pour**  $k = 1$  à  $n$  **faire**

**pour**  $i = 1$  à  $n$  **faire**

**pour**  $j = 1$  à  $n$  **faire**

$D[i, j] \leftarrow \min(D[i, j], D[i, k] + D[k, j])$

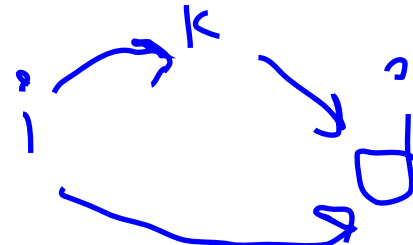
**retourner**  $D$

$D$ [matrice adjacents]

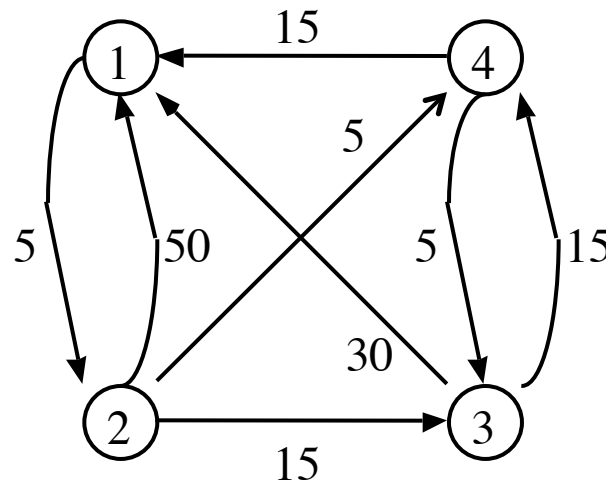
appartient à  $O(n^3)$

le plus court chemin satisfait le principe d'optimalité

**Exercice :** Montrer que  $t(n) \in O(n^3)$



# Exemple 3 : Trace de l'algorithme



TP4

$$D_0 = L = \begin{matrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{matrix} 0 & 5 & \infty & \infty \\ \infty & 0 & 15 & 5 \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \end{matrix} \end{matrix}$$

$$\text{et } D_4 = \begin{matrix} & \begin{matrix} 0 & 5 & 15 & 10 \end{matrix} \\ \begin{matrix} 0 & 5 & 15 & 10 \end{matrix} & \begin{matrix} 20 & 0 & 10 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{matrix} \end{matrix}$$

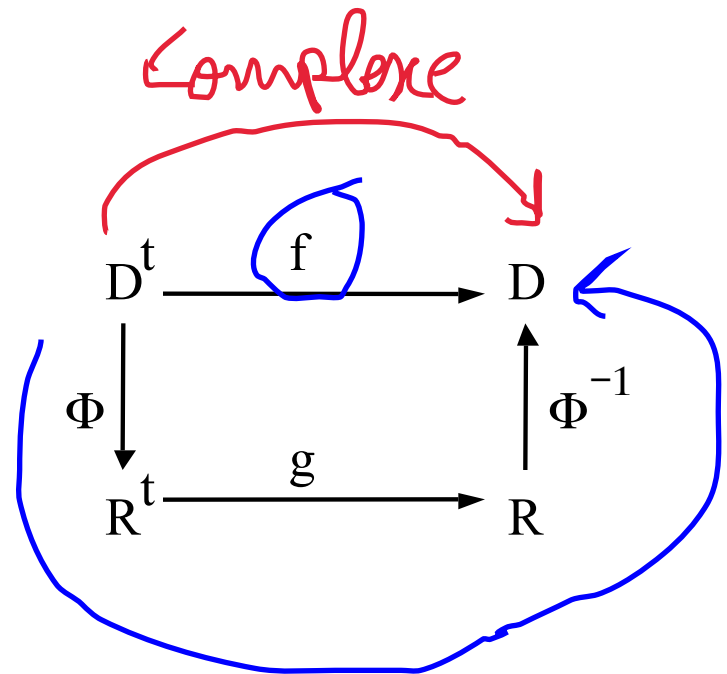
# Chapitre 6 : Transformation du domaine

Soit  $f : D^t \rightarrow D$  une fonction à calculer.

Une transformation consiste en :

- domaine  $R$
- injection  $\Phi : D \rightarrow R$
- fonction  $g : R^t \rightarrow R$

tel que :  $f = \Phi^{-1} \circ g \circ \Phi$



interet : si ce passage prend moins de ressource  
temps / memoire

# Contenu

- Exemples de transformation du domaine
- Multiplication symbolique de deux polynômes
- Transformée de Fourier Discrète
- Transformée de Fourier inverse

# Exemples de transformation du domaine

$$\underbrace{a \times b}_{\text{grand produit}} = e^{(\ln a + \ln b)}$$

- Produit de deux nombres positifs

$$D = \mathbb{R}_+^*, f(u,v) = u*v, R = \mathbb{R} \text{ et } \Phi(u) = \ln u \text{ et } g(x,y) = x+y$$

$$\begin{array}{ccc} (u,v) & \xrightarrow{f} & u*v \\ \ln \downarrow & & \uparrow e \\ (\ln u, \ln v) & \xrightarrow{g} & \ln u + \ln v = \ln u*v \end{array}$$

- Addition en représentation romaine (XVI + CIV)  $\rightarrow$  XX
- Changement de coordonnées (cartésiennes/polaires)
- Calcul dans un ordinateur (E/S en décimal et calcul en binaire)
- Calcul différentiel (transformée de Laplace).  $\text{integrale} \Rightarrow \text{transformée Laplace} \Rightarrow \text{TL inverse}$

Service cloud [cloud computing, IaaS, PaaS, SaaS, XaaS]

Conception et analyse des algorithmes, M. Eleuldj, EMI, janvier 2020

102

data => cloud +> calcul => result

# Multiplication symbolique des polynômes

Soient  $p(x) = \overbrace{a_{d-1}x^{d-1} + a_{d-2}x^{d-2} + \dots + a_0}^{n-1 \text{ mult}}$  et  $q(x) = \overbrace{b_{d-1}x^{d-1} + b_{d-2}x^{d-2} + \dots + b_0}^{n \text{ mult}}$

On veut calculer :  $r(x) = p(x) q(x)$

on va faire n fois la multiplication ]\* n  
algorithme quadratique

meme on optimise avec la methode d'horner pour  
ne pas repeter les calculs => on reste  $O(n^2)$

## algorithme classique

$\in O(d^2)$  en comptant les opérations scalaires comme élémentaires.

Horner=pour ne pas repeter les calcul (le plus optimale algorithme  
aujourd'hui)

$\in O(n \log n)$   
**Transformation du domaine**

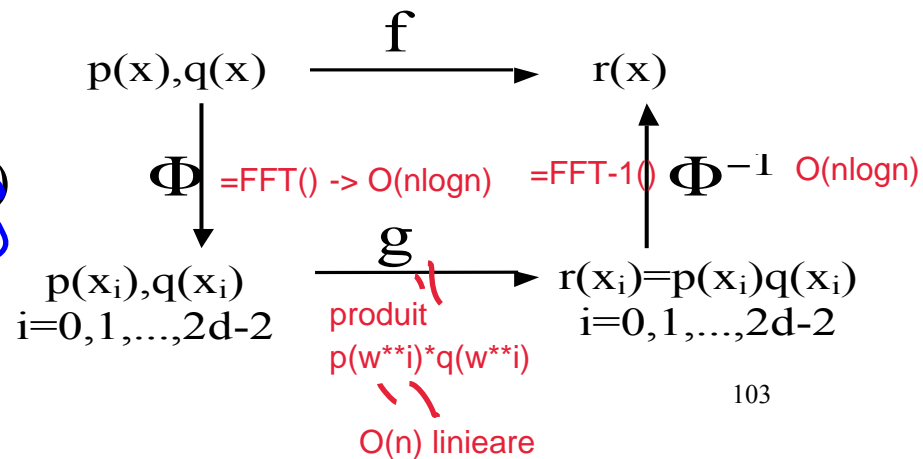
$f$  : multiplication symbolique

$\Phi$  : évaluation en  $2d - 1$  points  $\in O(d^2)$

$g$  : multiplication ponctuelle

$\Phi^{-1}$  : interpolation

Horner  $[ \dots [ a_0 ] \in O(n) \Rightarrow O(n) \times n \Rightarrow O(n^2)$



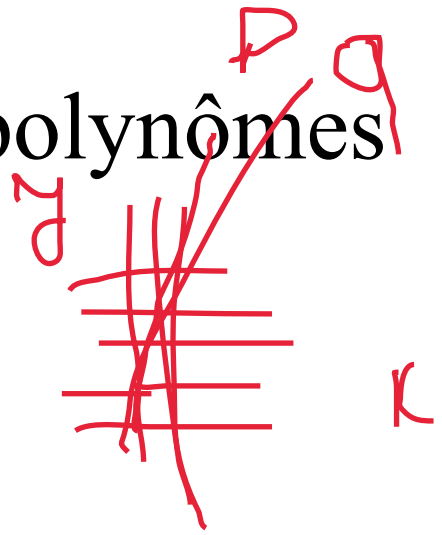
# Exemple de multiplication des polynômes

$$p(x) = 2x - 1$$

et

$$q(x) = x - 1$$

$$r(x) = p(x)q(x) = (2x - 1)(x - 1) = 2x^2 - 3x + 1$$



$p$  et  $q$  peuvent être représentées à l'aide de leur coefficients ou en 2 valeurs (0 et 1)

$$p = (-1, 1)$$

et

$$q = (-1, 0)$$

$\Rightarrow$

$$r = pq = (1, 0)$$

*évaluation*

$r(x)$  n'est pas complètement défini

$ax^2+bx+c \Rightarrow$  on a besoin de 3 pts [Degré = 2]

$r$  est de degré 2  $\Rightarrow$  il doit être défini de façon unique à l'aide de 3 valeurs

$$p = (-1, 1, 3)$$

$$\text{et } q = (-1, 0, 1)$$

$\Rightarrow$

$$r = pq = (1, 0, 3)$$

Méthode de Lagrange (ou résolution d'un système linéaire) on trouve :

$$r(x) = 2x^2 - 3x + 1$$

méthode d'interpolation :

abscisse + formule générale  $\Rightarrow$  polynôme exacte



# Transformée de Fourier Discrète

Applications (optique, acoustique, télécommunications, traitement du signal)

Soient  $n = 2^k$ ,  $k > 0$ ,

$\omega / \omega^n = 1$  (racine <sup>ième</sup> de l'unité)

## Exemples

$\omega = 4$ ,  $n = 8$  et le calcul se fait modulo 257

$\omega = (1+i)/\sqrt{2}$ ,  $n = 8$  et le calcul est en arithmétique complexe

## Définition

Soit  $a = (a_0, a_1, \dots, a_{n-1})$  vecteur qui définit  $p_a(x) = a_{n-1}x^{n-1} + \dots + a_0$

Transformée de Fourier de  $a$  relativement à  $\omega$  est :

$$F_\omega(a) = (p_a(1), p_a(\omega), \dots, p_a(\omega^{n-1}))$$

**Exercice :** Calculer  $F_\omega(a)$  et  $F_\omega(b)$  avec  $a=(1,-1,-5,3)$ ,  $b=(-2,6,-4,1)$  et  $\omega = i$ .

$$Pa(x) = 3x^3 - 5x^2 - x + 1$$

$$Pa(1) = -2 \quad Pa(i) = -4i + 6$$

il faut vérifier d'abord  
que  $w$  est une racine  
nième de l'unité  
on calcule alors  $w^{**n} = w^{**4}$   
et il doit être 1

# Algorithme de la Transformée de Fourier

Supposons  $n > 1$  et posons  $t = n/2$

Soient  $b = (a_0, a_2, \dots, a_{n-2})$  et  $c = (a_1, a_3, \dots, a_{n-1})$  tels que :

$$p_a(x) = p_b(x^2) + x p_c(x^2)$$

$= a_{n-1} * x^{n-2} + \dots + a_0 + a_{n-1} + \dots + a_1$

En particulier :

$$p_a(\omega^i) = p_b(\beta^i) + \omega^i p_c(\beta^i) \text{ où } \beta = \omega^2$$

$\beta^t = 1 \Rightarrow$  on peut de parler de  $F_\beta(b)$  et  $F_\beta(c)$

De plus

$$\beta^{t+i} = \beta^i \Rightarrow p_a(\omega^{t+i}) = p_b(b^i) + \omega^{t+i} p_c(b^i)$$

comme nous allons voir que  $\omega^t = -1$

$$\Rightarrow p_a(\omega^{t+i}) = p_b(b^i) - \omega^i p_c(b^i)$$

# Algorithme FFT

**fonction** FFT ( $a[0\dots n-1]$ ,  $\omega$ ) : **tableau**  $[0\dots n-1]$

**tableau**  $A[0\dots n-1]$  {pour recevoir le résultat}

**si**  $n=1$  **alors**  $A[0] \leftarrow a[0]$

**sinon**  $t \leftarrow n/2$

**tableaux**  $b, c, B, C[0\dots t-1]$

**pour**  $i \leftarrow 0$  **à**  $t-1$  **faire**  $b[i] \leftarrow a[2i]$ ,  $c[i] \leftarrow a[2i+1]$

$B \leftarrow \text{FFT}(b, \omega^2)$

$C \leftarrow \text{FFT}(c, \omega^2)$

$\beta \leftarrow 1$

**pour**  $i \leftarrow 0$  **à**  $t-1$  **faire**

$A[i] \leftarrow B[i] + \beta C[i]$

$A[t+i] \leftarrow B[i] - \beta C[i]$

$\beta \leftarrow \beta \omega$

**retourner**  $A$

**Exercice 1** : Montrer que  $\text{FFT} \in O(n \log n)$ .

2 appel recu  $\Rightarrow a=2$   
a chaque fois on utilise  
des sous exemplaie  
de taille  $n/2$

decomposer  
 $a \rightarrow b, c$

$t(1)=a$   
 $t(n)=2t(n/2) + bn$

2 sous exemplaie  $\Rightarrow c=2$

ADPR

$a = \tau$

# Trace de FFT

$$a = (1, -1, -5, 3), \omega = i$$

$$\text{FFT}(a, i)$$

$$t=2, b = (1, -5) \text{ et } c = (-1, 3)$$

$$B = \text{FFT}(b, 1)$$

$$t=1, b=(1) \text{ et } c=(-5)$$

$$B = \text{FFT}(b, 1) = (1)$$

$$C = \text{FFT}(c, 1) = (-5)$$

$$A = (-4, 6)$$

$$C = \text{FFT}(c, -1)$$

$$t=1, b=(-1) \text{ et } c=(3)$$

$$B = \text{FFT}(b, 1) = (-1)$$

$$C = \text{FFT}(c, 1) = (3)$$

$$A = (2, -4)$$

$$A = (-2, 6-4i, -6, 6+4i)$$

result w/

# Transformée de Fourier inverse

**Définition :** le nombre  $\omega$  est appelé une  $n$ -ième racine principale de l'unité si :

- i)  $\omega \neq 1$  (sauf si  $n=1$ )
- ii)  $\omega^n = 1$
- iii)  $\sum_{1 \leq j \leq n-1} \omega^{jp} = 0$  pour  $1 \leq p < n$

$\omega^n = 1$   
alors  $\omega$  racine unité

$\omega^n = 1$

**Remarques :**

l'inverse de  $\omega$  est  $\omega^{n-1}$

si  $p = n/2$ , la deuxième condition implique :

$$\sum_{1 \leq j \leq n-1} \omega^{jn/2} = n/2(1 + \omega^{n/2}) = 0 \Rightarrow \omega^{n/2} = -1.$$

$\omega^{-1} = \omega^{n-1}$   
 $\omega^{n/2} = 1$   
 $\omega^n = -1$   
 $\omega = \omega^{n/2}$   
 $n/2 =$  taille de b et c

**Exercice :**

- Montrer que  $\omega^{-1}$  est une  $n$ -ième racine principale de l'unité.
- Soient  $n$  et  $\omega$  des puissances positives de 2 et soit  $m = \omega^{n/2} + 1$ . Prouver que  $\omega$  est une  $n$ -ième racine principale de l'unité dans l'arithmétique modulo  $m$ . Prouver que  $n-1$  existe, en montrant que  $n^{-1} = m - (m-1)/n$ .

# Transformée de Fourier inverse

**Théorème :** Soient  $A$  la matrice  $n \times n$  /  $A = (a_{ij})$  et  $a_{ij} = \omega^{ij}$ . Comme  $\omega$  est une  $n$ -ième racine principale de l'unité et  $n^{-1}$  existe alors  $A$  est inversible, son inverse étant la matrice  $B$  /  $B = (b_{ij})$  et  $b_{ij} = \frac{1}{n} \omega^{-ij}$  et  $AB = I_n$  où  $I_n$  est la matrice identité.

$1/n$

Une autre formulation de la définition de la transformée de Fourier est :

$$F_{\omega}(a) = a A$$

**Exercice :**

Définir  $A$  et  $B$  pour  $n = 4$ .

**Corollaire :**  $0 \leq i < j < n \Rightarrow \omega^i \neq \omega^j$ .

Cette propriété est essentielle à l'interpolation du polynôme produit

# Algorithme FFT inverse

La transformée de Fourier inverse de  $a$  relativement à  $\omega$  est :

$$F^{-1}_{\omega}(a) = (n^{-1}p_a(1), n^{-1}p_a(\omega^{-1}), \dots, n^{-1}p_a(\omega^{-(n-1)}))$$

$$= n^{-1} F_{\omega^{-1}}(a) = n^{-1} F_{\omega^{n-1}}(a).$$

avec des signes negatives

**Exercice 4 :** Montrer que  $F^{-1}_{\omega}(F_{\omega}(a)) = a$  pour tout vecteur  $a$

**fonction** FFTinverse ( $a[0 \dots n-1]$ ,  $\omega$ ) : **tableau** $[0 \dots n-1]$

**tableau**  $F[0 \dots n-1]$

$F \leftarrow \text{FFT}(a, \omega^{n-1})$

**pour**  $i \leftarrow 0$  **à**  $n-1$  **faire**  $F[i] \leftarrow n^{-1}F[i]$

**retourner**  $F$

$O(n \log n)$

$O(n)$

Asymptotique

FFT inverse  $\in O(n)$

$O(n \log n)$

**Exercice :** Faire la trace pour FFTinverse( (-2, 6-4i, -6, 6+4i)

TP 6

FFT inverse (1, -5, 3)

# Application à la multiplication symbolique des polynômes

Soient  $p(x) = a^{s-1}x_{s-1} + \dots + a_0$  et  $q(x) = b_{t-1}x^{t-1} + \dots + b_0$

On veut calculer  $r(x) = p(x)q(x)$  de degré  $d = s + t$ .

Soient  $n$  la plus petite puissance de 2  $\geq d$  et  $\omega$  une  $n$ -ième racine principale de l'unité.

Soient  $a$  et  $b$  tels que  $a = (a_0, a_1, \dots, a_{s-1}, 0, \dots, 0)$  et  $b = (b_0, b_1, \dots, b_{t-1}, 0, 0, \dots, 0)$ .

Soient  $A = F_\omega(a)$  et  $B = F_\omega(b)$

$$A_i = p(\omega^i) \text{ et } B_i = q(\omega^i), i = 0, 1, \dots, n-1.$$

Soit  $C$  tel que  $C_i = A_i B_i = p(\omega^i)q(\omega^i) = r(\omega^i)$ .

$\Rightarrow C = F_\omega(c)$  correspond aux coefficients de  $r$

D'où

La multiplication symbolique des polynômes  $\in O(d \log d)$



# Trace de l'exemple

$$p(x) = 3x^3 - 5x^2 - x + 1 \text{ et } q(x) = x^3 - 4x^2 + 6x - 2$$

On choisit  $n=8$  et on sait que  $\omega=4$  est une racine principale de l'unité dans l'arithmétique modulo 257.

Soit  $a = (1, -1, -5, 3, 0, 0, 0, 0)$ ,  $b = (-2, 6, -4, 1, 0, 0, 0, 0)$ .

$$F_{\omega}(a) = (255, 109, 199, 29, 251, 247, 70, 133)$$

$$\text{et } F_{\omega}(b) = (1, 22, 82, 193, 244, 103, 179, 188).$$

Le produit point par point modulo 257 donne  $C = (255, 85, 127, 200, 78, 255, 194, 75)$

Comme  $F_{\omega}^{-1}(C) = (-2, 8, 0, -31, 37, -17, 3, 0)$ .

alors :

$$r(x) = 3x^6 - 17x^5 + 37x^4 - 31x^3 + 8x - 2$$

# Chapitre 7 : Algorithmes probabilistes

## Caractéristiques

Hasard peut prendre certaines décisions

Différentes exécutions sur le même exemplaire peuvent produire des résultats différents

Conflit avec la définition d'un algorithme

## Contenu



Ali Baba et les 40 voleurs

Calcul de  $\Pi$

Intégration numérique

Test de primalité

Algorithmes génétiques

# Exemple 1 : Ali Baba et les 40 voleurs

## Hypothèses

Valeur du trésor est  $T$

les voleurs prélèvent chaque nuit un montant  $x$

Une fois sur place Ali Baba peut reconnaître le trésor

4 jours de calcul pour déterminer le lieu exact

Indication de Jouha moyennant  $3x$

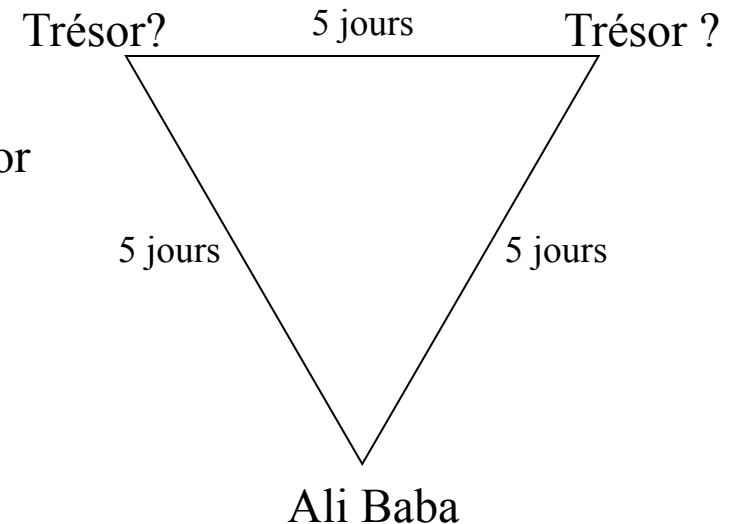
## Solutions

$$S1 = T - (4 + 5)x = T - 9x$$

$$S2 = T - (3 + 5)x = T - 8x$$

$$S3 = T - 5x$$

$$= T - 10x \quad \Rightarrow \quad \text{espérance} = (T - 5x)/2 + (T - 10x)/2 = T - 7,5x$$



## Exemple 2 : Calcul de $\Pi$

$k$  : nombre des points à l'intérieur du cercle

$n$  : nombre total de points

$S_1 = \Pi r^2$  : surface du cercle

$S_2 = (2r)^2 = 4 r^2$  : surface du carré

$S_1/S_2 = \Pi r^2/4r^2 = \Pi/4 \Rightarrow \Pi = 4 S_1/S_2 \approx 4 k/n$

**fonction**  $\Pi (n)$

$k \leftarrow 0$

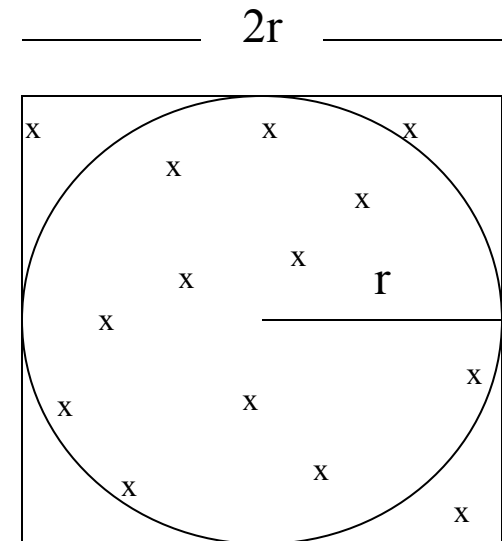
**pour**  $i = 1$  **à**  $n$  **faire**

$x \leftarrow \text{uniforme}(0,1)$

$y \leftarrow \text{uniforme}(0,1)$

**si**  $(x^2 + y^2 < 1)$  **alors**  $k \leftarrow k + 1$

**retourner**  $4k/n$



# Exemple 3 : Intégration numérique

Soit  $f : [0,1] \rightarrow [0,1]$  continue

Calculons l'intégrale  $f(x)$  entre 0 et 1

**fonction** intégration1(f,n)

$k \leftarrow 0$

**pour**  $i = 1$  à  $n$  **faire**

$x \leftarrow \text{uniforme}(0,1)$

$y \leftarrow \text{uniforme}(0,1)$

**si**  $y \leq f(x)$  **alors**  $k \leftarrow k + 1$

**retourner**  $k/n$

# Exemple 3 : Intégration numérique

```
fonction intégration2(f,n)
  somme ← 0
  pour i = 1 à n faire
    x ← uniforme(0,1)
    somme ← somme + f(x)
  retourner somme/n
```

## Remarque

Algorithme 1 et 2 sont intéressants car les algorithmes systématiques ne donnent que des valeurs approximatives

# Exemple 4 : Test de primalité

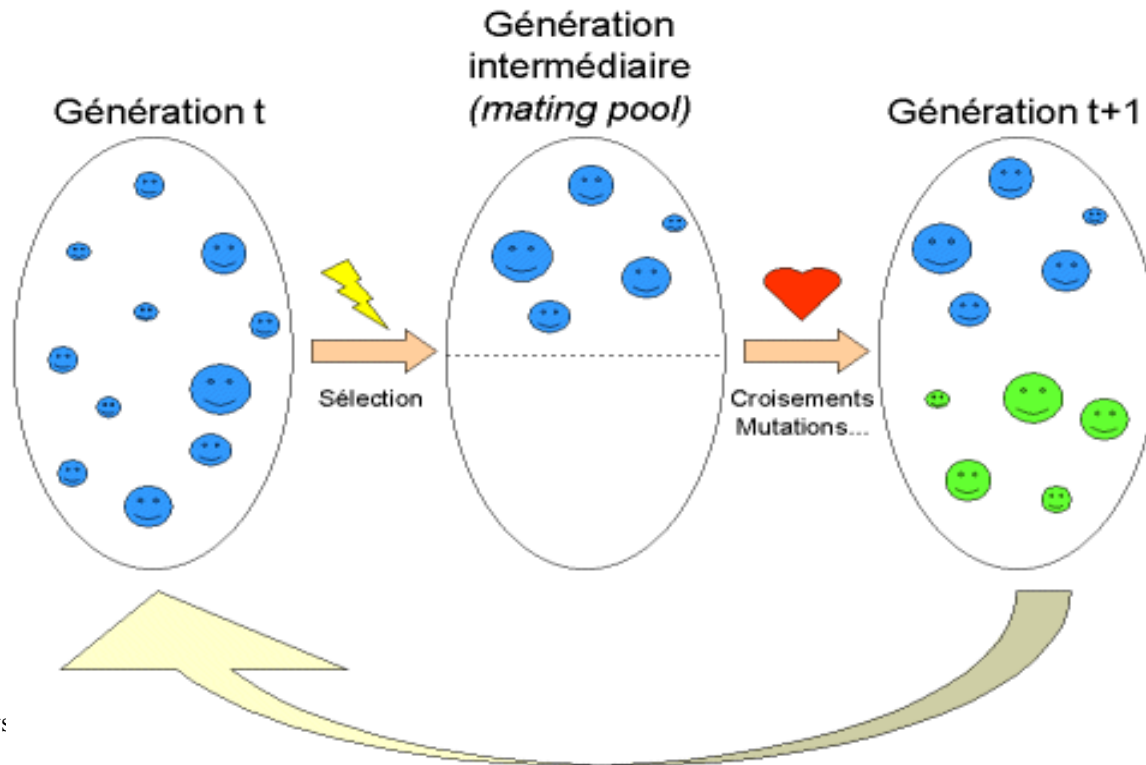
```
fonction premier(n)  
    d ← uniforme(2, √n)  
    retourner ((n mod d) != 0)
```

## Remarques

- Cas favorable :  $n = 1 \times 2 \times 3 \times \dots \times (n-1) = n!$
- Pire cas :  $n$  non premier mais est le produit de deux nombres premiers
- $n = 2623 = 43 \times 61 \Rightarrow \text{fiabilité} = 2\%$
- Plusieurs tests permettent d'augmenter la fiabilité
- Il existe d'autres moyens qui donnent des tests plus précis

# Exemple 5 : Algorithmes génétiques

- analogies avec les phénomènes biologiques (sélection naturelle de Charles Darwin)
- vocabulaire : individus (solutions potentielles), population, gènes (variables), chromosomes, parents, descendants, de reproduction, de croisement, de mutations, etc.



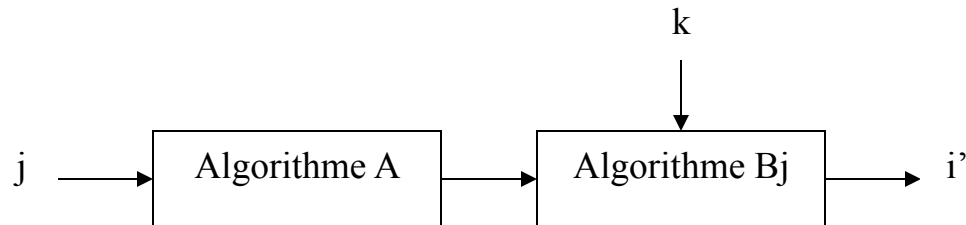


# Chapitre 8 : Pré-conditionnement

$I$  : ensemble des exemplaires

$J, K \subset I$  tel que  $i \in I, i = \langle j, k \rangle$

$i'$  est la solution de  $i$



## Exemples

Réalisation d'une application = Compilation + exécution

Recherche dans un ensemble = Construction du tas + recherche binaire en  $O(\log n)$

Evaluation répétée d'un polynôme = forme pré-conditionnée + évaluation

# Evaluation répétée d'un polynôme

J : ensemble des polynômes à une variable de degré n

K : ensemble des valeurs que la variable peut prendre

## Hypothèses

Coefficient entiers

Évaluation pour des valeurs entières

Polynôme unitaires

$$n = 2^k - 1$$

## Comptage du nombre de multiplications

$$\begin{aligned} P(x) &= x^7 - 5x^6 + 4x^5 + -13x^4 + 3x^3 - 10x^2 + 5x - 17 \\ &= (x^4 + 2)[(x^2 + 3)(x - 5) + (x + 2)] + [(x^2 - 2 - 4)x + (x + 9)] \\ &\Rightarrow 5 \text{ multiplications et } 9 \text{ additions} \end{aligned}$$

# Méthode de pré-conditionnement

Soit  $p(x)$  un polynôme unitaire de degré  $2^k - 1$

$$p'(x) = (x^{2^{k-1}} + a)q(x) + r(x)$$

avec  $a$  une constante et  $q$  et  $r$  deux polynômes unitaires de degré  $2^{k-1} - 1$

la même procédure est appliquée récursivement à  $q$  et  $r$

$p'$  appelée est la forme pré-conditionnée de  $p$

**Exercice :** trouver  $p'$  pour  $x^7 + 2x^6 - 5x^4 + 2x^3 - 6x^2 + 6x - 32$  et  $x^7$

Soit  $M(k)$  : nombre de multiplications pour évaluer  $p(x)$

$M'(k) = M(k) - k + 1$  Si on ne compte pas les multiplications pour  $x^2, x^4, \dots, x^{2^{k-1}}$

$$M'(k) = 0 \quad \text{si } k = 0$$

$$= 2 M'(k-1) + 1 \quad \text{si } k \geq 2$$

$$\Rightarrow M'(k) = 2^{k-1} - 1 \Rightarrow M(k) = 2^{k-1} + k - 2$$

D'où il suffit de faire :  $(n-3)/2 + \log(n+1)$  multiplications

# Méthodes d'évaluation d'un polynôme

Méthode	Nombre de multiplications pour p	En général
Directe	27	$n(n + 1)/2 - 1$
Dynamique	12	$2n - 2$
Horner	6	$n - 1$
Pré-conditionnement	5	$(n - 3)/2 + \log(n + 1)$